INTERNATIONAL JOURNAL OF INFORMATION SECURITY SCIENCE
Ö. Kağızmandere et al., Vol.13, No.2, pp. 33-42
https://doi.org/10.55859/ijiss.1455039

Vulnerability analysis based on Software Bill of Materials (SBOM): A model proposal for automated vulnerability scanning for CI/CD pipelines

Ömercan Kağızmandere[1] , Halil Arslan[1]

[1]Computer Engineering, Sivas Cumhuriyet University, Sivas, Türkiye
Corresponding Author: harslan@cumhuriyet.edu.tr

Research Paper Received: 18.03.2024 Revised: 28.06.2024 Accepted: 28.06.2024

Abstract—The software bill of materials (SBOM) emerged in 2018 as an important component in software security and software supply chain management. SBOM is an inventory presented as a list of the components that make up software. In recent years, whether software products contain vulnerabilities is a phenomenon that should be checked regularly by the users of that product. This paper deals with the systematic identification and vulnerability analysis of software components based on the concept of software bill of materials. The fact that a software product itself does not contain vulnerabilities does not mean that the software product is secure. Even if software projects do not contain any vulnerabilities when examined alone, there may be vulnerabilities in their components. Vulnerabilities in the dependencies or components of the product may be sufficient for cyber attackers to exploit that product. Minimizing the damage caused by vulnerabilities in software components is the basis of cyber security efforts. In this study, the necessity of automatically generating software bill of materials in software development/deployment environments and performing vulnerability analysis on this bill of materials is demonstrated and a suitable model is proposed.

Keywords—Software Bill of Materials, Vulnerability analysis, Software security

1. Introduction

A software product requires many components/dependencies to perform its intended tasks. These dependencies affect the integrity of the product in terms of security as well as the software product itself. For this reason, a vulnerability in a system can be exploited and other systems can be threatened through that system. In other words, in software systems, the absence of a vulnerability in the project itself does not mean that the project is secure. If there is a vulnerability in the dependencies of the project, there is a vulnerability in the main project. Attacks can occur at any point in a normal software supply chain, and these attacks are becoming more visible, destructive and expensive in today's world. To prevent this, it has become critically important to know the dependencies of a project. A vulnerability in the dependencies can cause major damage to the main project. Providing methods to prevent this increases the importance of the research.

The International Organization for Standardization (ISO) 25010 quality model is a hierarchical structure consisting of 8 main and 31 sub-criteria (ISO/IEC 25010:2011). Within ISO 25010, security is one of the 8 main criteria. Security is a quality characteristic that indicates that a product or system protects information and data [1][2]. Addressing the security category of a software product in the context of the ISO 25010 Quality Model cannot be achieved only by examining the existing product. A vulnerability that may arise in the dependencies of the software product also puts the security parameters of the related product at risk. In this context, SBOM has been proposed to identify the dependencies of a software product, to examine the vulnerabilities contained in these dependencies, and to provide transparency and visibility to both software developers and relevant consumers regarding a software product [3]. The SBOM of a project is therefore the most up-to-date approach to see the dependencies of that project. The SBOM is a list of the components that make up the software.

This study aims to prove that projects with no apparent vulnerabilities may contain vulnerabilities due to vulnerabilities in their dependencies. It is aimed to create an end-to-end model for this purpose. It is stated that this model should be a part of the pipeline from the development phase of an application to the live deployment in the Continuous Integration/Continuous Delivery (CI/CD) phase. In future studies, the model will be integrated into CI/CD environments as a crucial step to ensure that relevant security analyses are considered in software development processes. In this study, we first dynamically generate a SBOM in the Software Package Data Exchange (SPDX) standard for tracking and documenting the dependencies used by a software product. SPDX is an industrial standard for describing software packages and their dependencies. It is designed to work with all vendors, programming languages, and frameworks [4]. Within the scope of the study, 10 different open-source projects with widespread developer and user support were analyzed in this context. Dynamically extracted SBOM were then analyzed for vulnerabilities based on product and dependency. As a result, it has been shown that SBOM data can be one of the most important parameters to be examined in terms of both ISO 25010 and cyber security, whether the projects are distributed as open source or compiled code.

According to the definitions above, the main research questions of our study are as follows:

- Research Question 1 (RQ 1): Can SBOM be automatically extracted from source code?
- Research Question 2 (RQ 2): Can vulnerabilities of the source code dependencies be found through the automatically extracted SBOM lists?
- Research Question 3 (RQ 3): Is there a correlation, a relationship between the existing product and its dependencies through vulnerabilities?

2. Literature Review

The Software Bill of Materials fulfills a critical role in ensuring software supply chain security by providing a detailed inventory of components and dependencies that are integral to software development. However, there are numerous challenges to sharing SBOM, including the potential for data falsification and hesitancy to disclose comprehensive information between software vendors. While these obstacles hinder the widespread adoption and use of SBOM, they also highlight the need for a more secure and flexible mechanism for SBOM sharing. Considering this, the details of SBOM studies are reviewed.

Axelsson et al. investigated how far the open-source software community has come in adopting SBOM and how existing SBOM have evolved, focusing on the SPDX format. For the purpose of this research, they conducted an archival study in which they searched for SBOM in open-source software projects on GitHub and analyzed their content and evolution [5].

Adewumi et al. conducted a systematic literature review to guide the formulation and development of new models by classifying existing open-source software quality assessment models according to their quality attributes, the methodologies they use, and their application domains [6].

Stoddard et al. noted in their research that while there is significant community discussion on SBOM generation, there is less discussion focused on SBOM sharing. Their work is intended to highlight currently used SBOM sharing solutions and help readers explore appropriate sharing solutions based on their needs related to the discovery, access, and transportation of SBOM [7].

Camp et al. focused on systematic vulnerability classes based on SBOM standards. As a result of their study, they concluded that SBOM is a promising solution not only to secure the supply chain but also to address the problem of identifying software components [8].

Xia et al. extended the scope of SBOM to AI systems by introducing a blockchain-enabled architecture for SBOM sharing. Thus, they coined the term Artificial Intelligence Bill of Materials (AIBOM). By demonstrating the feasibility and flexibility of the proposed SBOM sharing mechanism, they introduced a new solution for securing software supply chains [9].

Ding et al. proposed a SBOM generation method based on enterprise big data, which is used to create a complete electronic file for each product. The manual process is greatly reduced, and the accuracy of the data is improved [10].

In his study, Kemppainen stated that when testing and selecting an SBOM-tool, it is important to consider some aspects such as the software development environment, the software package managers used, and the desired SBOM format [11].

Chaora et al. underline that the widespread use of software in various sectors, including industrial, manufacturing, and municipal technologies, requires a reliable and secure software supply chain. The authors discussed various initiatives to improve risk management in software supply chains such as SBOM. The study highlights the critical role of SBOM in supply chain security and provides valuable insights for future research and development in this area [12].

The research by Harer et al. provides a new perspective on enhancing software security by introducing an automated method for vulnerability discovery using machine learning. This work contributes to the larger discussion initiated by previous research examining issues related to software supply chain attacks, risk assessment techniques, and the use of SBOM. The work of Harer et al. addresses the technical aspect of proactively finding vulnerabilities [13].

As can be seen from the reviewed studies, it can be stated that SBOM is a methodology that is a candidate to play a key role in preventing security breaches caused by software vulnerabilities. Knowing the dependencies/components of the purchased software can provide a solution to the problem of vulnerable software for buyers.

3. Software Bill of Materials (SBOM)

SBOM, serves as a comprehensive inventory detailing the components constituting a piece of software [14]. It not only identifies these software components but also provides essential information about each, along with outlining the supply chain relationships among them. The extent and nature of information within a specific SBOM can vary, influenced by factors such as the industry and the specific requirements of SBOM consumers.

Today's software packages often include numerous third-party components. Companies must actively monitor and manage each of these components to maintain security and functionality. SBOM are a new version of an old concept. Vendors have historically used bills of materials (BOM) in supply chain management to identify the many parts that make up their products. For example, the ingredient list of the food we buy at the grocery store is effectively a BOM. The application of the bill of materials idea to software is more recent. It became more prominent in May 2021, when the Biden administration issued an executive order emphasizing SBOM as a way to improve cybersecurity in the US [14]. Software vendors selling to the US federal government are required by regulation to provide SBOM for their products. To this end, organizations are required to use a software bill of materials to track these components. This machine-readable list contains the various dependencies and elements of a piece of software [7]. A software BOM lists all component parts and software dependencies involved in the development and delivery of an application. SBOM are similar to bills of materials used in supply chains and manufacturing. They provide a common framework for all vendors in the IT industry to accurately identify the key code components from which an application is built [14].

A typical SBOM contains license information, version numbers, component details, and vendors. This list allows others to understand what the software contains and act accordingly, reducing risks for both the manufacturer and the user. The main purpose of an SBOM is to uniquely and unambiguously identify components and their relationships to each other. To achieve this, a combination of fundamental information is necessary. The essential details include [15]:

- Author Name: The individual responsible for the SBOM entry (which may not always be the supplier).
- Supplier Name: The component supplier's identity in the SBOM entry, allowing for multiple names or aliases. If the author and supplier are the same, the supplier identifies the first-party authorized component. If they differ, the author provides information about a component from another supplier.
- Component Name: One or more component names, with the ability to note multiple names or aliases. Component names may incorporate supplier names and can be conveyed using a generic namespace structure.
- Version String: The version information format is flexible but must adhere to common industry standards.
- Component Hash: Utilizing a cryptographic hash as a unique identifier is the most effective way to identify a software component.
- Unique Identifier: In addition to the hash, each component must possess an identification number that uniquely distinguishes it within the SBOM.
- Relationship: This defines the connection between the component and the package. In most instances, a relationship implies that a specific component is included in a particular package.

4. Methodology

The term ”exploit” essentially means to abuse or take advantage of something. Exploits are codes and programs developed to use vulnerabilities and bugs in software or computers. These codes and programs are used to exploit the system, damaging confidentiality, integrity, and availability [16]. Exploits find the system's weakest points and infiltrate it from there. Exploits pose a great risk to the digital world. Most cyber-attacks are exploit-based [17]. In order for companies not to be affected by exploits, they should regularly scan their existing software and systems with active/passive vulnerability analysis tools and close the vulnerabilities that emerge. The vulnerability of software is a combination of the vulnerabilities of all components of that product. Therefore, an effective vulnerability analysis requires analyzing all components that make up the relevant software.

In this study, a model is proposed in which the SBOM of software is generated from the source code and vulnerability analysis of software components can be performed. The proposed model is shown in Figure 1. This model proposes adding two new steps for SBOM generation and vulnerability scanning to the pipeline from software development to deployment. The necessity and validation of the proposed model are demonstrated with concrete results. According to this model, an automated vulnerability analysis is required for code merge requests as a result of developers' pull requests to result in the application deployment step through CI/CD tools. Automatic SBOM generation is proposed for vulnerability scanning of all software components.

Jenkins is an open-source automation tool written in Java with plugins built for Continuous Integration (CI). Jenkins is used to continuously build and test software projects. It makes it easy for developers to integrate project changes and for users to get a new build. It also integrates with multiple testing and deployment technologies to ensure the continuous delivery of software. With Jenkins, organizations can accelerate their software development process through automation. Jenkins can integrate any development lifecycle process, including build, document, test, package, phase, deployment, static analysis, and much more. Plugins allow the integration of various DevOps phases [18].

Within the scope of the study, firstly, 10 popular projects (C# based) on GitHub were identified [19]. The reason we chose C# projects is that the component detection tool we used was developed by Microsoft and we predicted that it would achieve more successful results in the source code. The popularity of these projects (number of stars and forks) was taken into account while determining these projects. In addition, the existing vulnerabilities of these projects were scanned through the MITRE-CVE database on the basis of passive vulnerability analysis, and it was checked whether they contained vulnerabilities. Common Vulnerabilities and Exposures (CVE) is a publicly available vulnerability database. The purpose of developing this database is to facilitate the sharing of information about vulnerabilities. A CVE record consists of a description of the vulnerability, a vulnerability identification number, and at least one public reference [20]. CVE aims to provide common names for publicly known issues. The purpose of CVE is to facilitate data sharing between separate vulnerability capabilities (tools, repositories and services) with this 'common numbering' [21].

Then, the SBOM of the identified projects were dynamically generated. SBOM-tool was used to generate SBOM for the projects [22]. The SBOM-tool creates a JSON formatted *”manifest.spdx.json”*

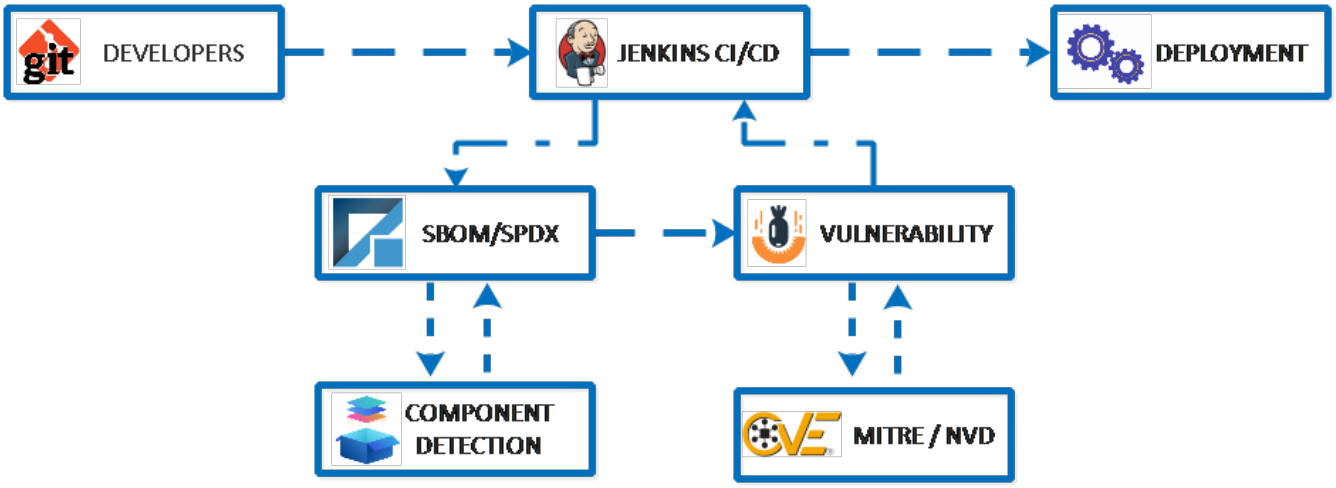

Figure 1. The proposed model overview

file of the source project. The tool used to generate SBOM in the study was developed by Microsoft. It is a highly scalable and enterprise-ready tool for generating SPDX 2.2 compliant SBOM for all types of structures. The tool uses Component Detection libraries to detect components and the ClearlyDefined API to determine license information for these components [22]. Component Detection is a package scanning tool designed for use at compile-time. It produces a graphical output of all components detected in various package ecosystems [23].

Vulnerability scans were performed on the SBOM of the projects. The open-source Bomber tool was used to determine whether the dependencies of the projects, whose SBOM were produced, contain vulnerabilities [24]. Bomber scans the "*manifest.spdx.json*" file created by the SBOM-tool and extracts vulnerabilities. As a result of these scans, the vulnerabilities within the dependencies of a project were identified and evaluated. Bomber is specifically designed to scan vulnerabilities through SBOM lists [24]. In this study, Bomber was utilized to identify potential vulnerabilities within the components comprising a software application. Bomber can process SBOM in JSON or XML-based CycloneDX, SPDX, or Syft formats and conduct vulnerability scanning.

Software can be available as open-source or compiled code. Third-party components sourced from GitHub or any other public repository are considered open-source software. In-house software developed for companies, while not publicly available, is also categorized as open-source since internal teams have access to the source code. Compiled or closed-source software, typically sourced externally, can also be internal to a company. Companies employ Software Composition Analysis (SCA) tools from vendors such as GitHub, Sonatype, Snyk, etc., to scan all types of open-source software and provide vulnerability data, sometimes including the creation of SBOM. However, passive vulnerability analysis tools cannot scan components of compiled or closed-source software. This is where SBOM provided by vendors become crucial. SBOM present the components that constitute the respective software, and tools like Bomber can ascertain whether these components include any vulnerabilities.

Table 1.
Projects selected for the experimental study

No	Project Name	Stars	Forks	Repo URI (GitHub)
1	OpenRA	13768	2638	OpenRA
2	runtime	13403	4391	dotnet
3	efcore	13060	3064	dotnet
4	abp	11916	3275	abpframework
5	aspnetboilerplate	11420	3761	aspnetboilerplate
6	Captura	9126	1754	MathewSachin
7	spectre.console	8036	421	spectre.console
8	RestSharp	9339	2335	restsharp
9	Radarr	8554	889	Radarr
10	Ocelot	7999	1613	ThreeMammals

Table 2.
Count of vulnerabilities in examined projects

No	Project Name	Vulnerabilities
1	OpenRA	1
2	runtime	2
3	efcore	2
4	abp	190
5	aspnetboilerplate	13
6	Captura	3
7	spectre.console	2
8	RestSharp	2
9	Radarr	8
10	Ocelot	10

In summary, dependencies from 10 projects listed among the top 100 most popular C# projects [19], whose source code is available on GitHub, were extracted using the SBOM-tool. Subsequently, vulnerability scanning was performed on these extracted SBOM. The vulnerabilities identified through the SBOM were evaluated by comparing them directly with vulnerabilities attributed to the relevant software products.

5. Experimental Studies

To validate the model proposed in this study, ten popular C# projects with available source codes on GitHub were first identified. The projects examined in the experimental study, along with some of their features, are listed in Table 1.

To determine the vulnerabilities of the examined projects, queries were made using the MITRE-CVE system. No vulnerability records were found on MITRE for the evaluated projects. Subsequently, SBOM of the evaluated projects were created, and their vulnerabilities were investigated through these SBOM. A sample vulnerability analysis report obtained through Bomber (pertaining to the Radarr project) is presented in Figure 2.

Vulnerabilities contained in project dependencies should be analyzed according to their severity and EPSS value, with the impact of the vulnerability considered by product users. The Exploit Prediction Scoring System (EPSS), referred to in the vulnerability analysis report presented by Bomber, is the result of a data-driven effort to predict the likelihood of a software vulnerability being exploited in a real-world environment. Its goal is to help network defenders better prioritize vulnerability remediation efforts. While other industry standards are useful for capturing the innate characteristics of a vulnerability and providing severity metrics, their ability to assess the threat is limited. EPSS fills this gap as it leverages existing threat information from the CVE and real-world exploit data. The EPSS model produces a probability score between 0 and 1 (0 and 100 percent). The higher the score, the greater the probability of a vulnerability being exploited. This value helps determine the priority of addressing the vulnerability [16]. Severity relates to the assessment of the vulnerability from a technical perspective. It focuses on the impact and possible consequences of the vulnerability. Severity statuses may include Critical, High, Moderate, and Low.

TYPE	NAME	VERSION	SEVERITY	VULNERABILITY	EPSS %
npm	traverse	7.22.11	CRITICAL	CVE-2023-45133	24%
	postcss	6.0.23	MODERATE	CVE-2021-23382,SNYK-JAVA-ORGWEBJARSNPM-1255641,SNYK-JS-POSTCSS-1255640	N/A
		6.0.23	MODERATE	CVE-2023-44270	18%
		8.4.29	MODERATE	CVE-2023-44270	18%
		8.4.23	MODERATE	CVE-2023-44270	18%
	css-tools	4.3.1	MODERATE	CVE-2023-48631	14%
	color-string	0.3.0	MODERATE	CVE-2021-29060	53%
nuget	System.Security.Cryptography.Pkcs	6.0.0	HIGH	BIT-dotnet-2023-29331,BIT-dotnet-sdk-2023-29331,CVE-2023-29331	N/A

Figure 2. Sample vulnerability analysis report from Bomber (Radarr project)

SBOM were generated for all projects selected within the scope of the experimental study, and vulnerability analysis was performed on these SBOM. The list of vulnerabilities found in the dependencies of the analyzed projects is presented in Table 2.

The findings of the experimental study demonstrate that the fact that a software product itself does not contain vulnerabilities does not guarantee it software product is secure. Vulnerabilities in the product's dependencies can still provide opportunities for attackers to exploit the software. The log4j vulnerability, which emerged in the last days of 2021, illustrates the damage that vulnerabilities in software components can cause [25].

6. Conclusion and Discussion

The SBOM of 10 open-source C# projects examined in this study were initially generated from their source codes. These SBOM were then analyzed using passive vulnerability analysis tool Bomber to scan for vulnerabilities within the software components. It was demonstrated that software packages appearing free of vulnerabilities on individual inspection may still contain vulnerabilities within their components. This finding underscores the importance of automatically generating an SBOM in software development and deployment environments and conducting vulnerability analysis on this bill of materials. Consequently, a new CI/CD model has been proposed, as depicted in Figure 1. This model advocates integrating two additional steps into the CI/CD process when a developer's pull request is made: automatic SBOM generation from source code using SBOM-tool and vulnerability scanning of this SBOM list using Bomber. The necessity of these steps was validated across the 10 sample projects. Therefore, we advocate for the integration of automatic SBOM extraction and vulnerability scanning via SBOM into CI/CD pipelines such as Jenkins, GitHub Actions, etc.

In conclusion, this study addressed three research questions. Firstly, the first and second research questions have been answered by confirming that SBOM can be automatically extracted from the source code and used to identify vulnerabilities. Secondly, regarding the correlation between product and dependency vulnerabilities, it was found that vulnerabilities in dependencies can expose otherwise seemingly secure software products to exploitation. This highlights that a software product free of vulnerabilities itself does not guarantee overall security.

In future studies, will focus on implementing the proposed model end-to-end within Jenkins integration. Additionally, with the increasing popularity of passive vulnerability analysis tools offering efficiency and continuous scanning in live environments compared to active tools, further research on SBOM generation and vulnerability scanning through passive analysis tools is crucial.

Furthermore, developing quality models based on SBOM static analysis tools presents a promising research area. These models could assess SBOM structure conformity to standards and SBOM content, enabling software vendors to monitor and enhance the quality of their software supply chains. Given the escalating complexity of modern software and the proliferation of third-party integrations, the associated security costs are rising. As software dependencies grow, so does the potential for vulnerabilities. Therefore, there is a growing need for automated structures that can detect vulnerabilities in dependencies, integrating seamlessly into CI/CD processes.

7. Threats to Validity

This section outlines the primary threats affecting the validity of our study's results. Firstly, the study is limited to 10 projects and specifically focused on C# projects. It reflects the outcomes of experiments conducted from December 2023 to January 2024. Moreover, vulnerabilities identified in these projects and their dependencies may have been resolved subsequently. Therefore, our findings should be validated in future studies encompassing different types of projects.

Acknowledgments

This study was produced as a part of the Master Thesis study of Ömercan Kağızmandere.

References

[1] E. Peters and G. K. Aggrey, "An iso 25010 based quality model for erp systems," *Adv. Sci. Technol. Eng. Syst. J*, vol. 5, no. 2, pp. 578–583, 2020.

[2] A. A. Pratama and A. B. Mutiara, "Software quality analysis for halodoc application using iso 25010: 2011," *Int. J. Adv. Comput. Sci. Appl*, vol. 12, no. 8, pp. 383–392, 2021.

[3] A. Arora and C. Garman, "Analysis of software bill of materials tools," *Cyber Security: A Peer-Reviewed Journal*, vol. 6, no. 4, pp. 334–355, 2023.

[4] S. Butler, J. Gamalielsson, B. Lundell, C. Brax, A. Mattsson, T. Gustavsson, J. Feist, B. Kvarnström, and E. Lönroth, "Considerations and challenges for the adoption of open source components in software-intensive businesses," *Journal of Systems and Software*, vol. 186, p. 111152, 2022.

[5] V. Axelsson and F. Larsson, "Understanding the software bill of material for supply-chain management in open source projects," 2023.

[6] A. Adewumi, S. Misra, and N. Omoregbe, "Evaluating open source software quality models against iso 25010," in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*. IEEE, 2015, pp. 872–877.

[7] J. T. Stoddard, M. A. Cutshaw, T. Williams, A. Friedman, and J. Murphy, "Software bill of materials (sbom) sharing lifecycle report," Idaho National Lab.(INL), Idaho Falls, ID (United States), Tech. Rep., 2023.

[8] L. J. Camp and V. Andalibi, "Sbom vulnerability assessment & corresponding requirements," *NTIA Response to Notice and Request for Comments on Software Bill of Materials Elements and Considerations*, 2021.

[9] B. Xia, D. Zhang, Y. Liu, Q. Lu, Z. Xing, and L. Zhu, "Trust in software supply chains: Blockchain-enabled sbom and the aibom future," *arXiv preprint arXiv:2307.02088*, 2023.

[10] X. Ding, F. Zhao, L. Yan, and X. Shao, "The method of building sbom based on enterprise big data," in *2019 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE)*. IEEE, 2019, pp. 1224–1228.

[11] P. Kemppainen, "Managing 3rd party software components with software bill of materials," 2023.

[12] A. Chaora, N. Ensmenger, and L. J. Camp, "Discourse, challenges, and prospects around the adoption and dissemination of software bills of materials (sboms)," in *2023 IEEE International Symposium on Technology and Society (ISTAS)*. IEEE, 2023, pp. 1–4.

[13] J. A. Harer, L. Y. Kim, R. L. Russell, O. Ozdemir, L. R. Kosta, A. Rangamani, L. H. Hamilton, G. I. Centeno, J. R. Key, P. M. Ellingwood *et al.*, "Automated software vulnerability detection with machine learning," *arXiv preprint arXiv:1803.04497*, 2018.

[14] V. V. Sehgal and P. Ambili, "A taxonomy and survey of software bill of materials (sbom) generation approaches," in *Analytics Global Conference*. Springer, 2023, pp. 40–51.

[15] É. Ó. Muirí, "Framing software component transparency: Establishing a common software bill of material (sbom)," *NTIA, Nov*, vol. 12, 2019.

[16] J. Jacobs, S. Romanosky, B. Edwards, I. Adjerid, and M. Roytman, "Exploit prediction scoring system (epss)," *Digital Threats: Research and Practice*, vol. 2, no. 3, pp. 1–17, 2021.

[17] H. Kekül, B. Ergen, and H. Arslan, "A multiclass hybrid approach to estimating software vulnerability vectors and severity score," *Journal of Information Security and Applications*, vol. 63, p. 103028, 2021.

[18] J. A. Kupsch, B. P. Miller, V. Basupalli, and J. Burger, "From continuous integration to continuous assurance," in *2017 IEEE 28th Annual Software Technology Conference (STC)*. IEEE, 2017, pp. 1–8.

[19] GitHub Ranking, "GitHub stars and forks ranking list," Accessed Nov. 20, 2023. [Online]. Available: https://github.com/EvanLi/Github-Ranking/blob/master/Top100/CSharp.md

[20] C. Hankin, P. Malacaria *et al.*, "Attack dynamics: an automatic attack graph generation framework based on system topology, capec, cwe, and cve databases," *Computers & Security*, vol. 123, p. 102938, 2022.

[21] S. Neuhaus and T. Zimmermann, "Security trend analysis with cve topic models," in *2010 IEEE 21st International Symposium on Software Reliability Engineering*. IEEE, 2010, pp. 111–120.

[22] SBOM Tool, "The SBOM tool is a highly scalable and enterprise ready tool to create SPDX 2.2 compatible SBOMs for any variety of artifacts," Accessed Nov. 1, 2023. [Online]. Available: https://github.com/microsoft/sbom-tool

[23] Component Detection, "Scans your project to determine what components you use," Accessed Nov. 1, 2023. [Online]. Available: https://github.com/microsoft/component-detection

[24] Bomber, "Scans Software Bill of Materials (SBOMs) for security vulnerabilities," Accessed Nov. 1, 2023. [Online]. Available: https://github.com/devops-kung-fu/bomber

[25] P. Ferreira, F. Caldeira, P. Martins, and M. Abbasi, "Log4j vulnerability," in *International Conference on Information Technology & Systems*. Springer, 2023, pp. 375–385.