USING SOFTWARE BILL OF MATERIALS FOR SOFTWARE SUPPLY CHAIN SECURITY AND ITS GENERATION IMPACT ON VULNERABILITY DETECTION

by

Eric Jeffery O'Donoghue

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 2024

ACKNOWLEDGEMENTS

I would like acknowledge my advisor, Dr. Clemente Izurieta, for the mentorship and support throughout the entirety of this thesis. I would also like to acknowledge my committee members, Dr. Ann Marie Reinhold and Dr. Matthew Revelle, for their support and guidance throughout this project. Montana State University Software Engineering and Cybersecurity Lab (MSU SECL) members provided many valuable insights and discussions that helped advance this work.

This research was conducted with the U.S. Department of Homeland Security (DHS) Science and Technology Directorate (S&T) under contract 70RSAT22CB0000005. Any opinions contained herein are those of the author and do not necessarily reflect those of DHS S&T.

TABLE OF CONTENTS

1. INTRODUCTION 1

2. RELATED WORK 4

Informal Literature Review of Assessing Software Supply Chain Security Using Software Bill of Materials 4
Search Strategy 4
Results 5
Software Supply Chain Security 5

3. RESEARCH GOALS 8

Goal Question Metric 8

4. ASSESSING SECURITY RISKS OF SOFTWARE SUPPLY CHAINS USING SOFTWARE BILL OF MATERIALS 10

Contribution of Authors and Co-Authors 10
Manuscript Information 11
Abstract 12
Introduction 13
Related Work 14
Methodology 16
Research Questions 17
Data Extraction and Processing 18
Data Analysis 19
Results 20
Discussion 23
Threats to Validity 24
Future Work 26
Conclusion 28
Acknowledgements 28
References 29

5. IMPACTS OF SOFTWARE BILL OF MATERIALS GENERATION ON VULNERABILITY DETECTION 32

Contribution of Authors and Co-Authors 32
Manuscript Information 33
Abstract 34
Introduction 35

TABLE OF CONTENTS – CONTINUED

Methodology 38
Results 40
RQ1: What impact do SBOM generation tools have on vulnerability detection in microservices? 43
RQ2: What impact do SBOM formats have on vulnerability detection in microservices? 46
Discussion 47
Implications for Practitioners and End Users 48
Threats to Validity 49
Future Work 51
Conclusion 51
Acknowledgements 52
References 53

6. CONCLUSION 56

REFERENCES CITED 58

LIST OF TABLES

Table Page

4.1 Trivy & Grype Findings Total Counts. © 2024 IEEE. 19

5.1 Confidence intervals from bootstrap analysis comparing vulnerability detection by SBOM analysis tools when varying SBOM generation tool and format. We calculated vlaues as follows: Rows 1 and 2, subtracted mean findings for Syft generated SBOMs from mean findings for Trivy$_G$ generated SBOMs and divided by their pooled standard deviation. Rows 3 and 4, subtracted mean findings for Syft generated SBOMs from mean findings for Trivy$_G$ generated SBOMs and divided by their pooled standard deviation.............................. 43

5.2 Cohen's D values measuring the effect size of variation in vulnerability detection by SBOM analysis tools when varying SBOM generation tool and format. We calucated values as follows: Rows 1 and 2, subtracted mean findings for Syft generated SBOMs from mean findings for Trivy$_G$ generated SBOMs and divided by their pooled standard deviation. Rows 3 and 4, subtracted mean findings for SPDX 2.3 SBOMs from mean findings for CycloneDX 1.5 SBOMs and divided by their pooled standard deviation SBOMs. 44

LIST OF FIGURES

Figure Page

3.1 Hierarchical structure of Basili's Goal-Question-Metric methodology for this study [49]. The manuscript information that addresses each research question is placed next to the corresponding research questions. The outline of the research questions aligns with which manuscript addresses them. 8

4.1 The study design consists of 5 steps. Data extraction occurs in steps 1 and 2, data processing is performed in steps 3 and 4, and the analysis is carried out in step 5. © 2024 IEEE. 18

4.2 Distribution of CVSS vulnerability scores from static analysis tools. Trivy and Grype, labeled respectively. Each blue dot represents a single vulnerability. The y-axis shows the total number of occurrences found for a given vulnerability. © 2024 IEEE. 21

4.3 Top 25 most vulnerable packages found by Trivy across the mined SBOMs.. 22

4.4 Top 25 most vulnerable packages found by Grype across the mined SBOMs. © 2024 IEEE.. 22

5.1 Overview of the methodology to study SBOM generation impacts on vulnerability reporting for microservices. 38

LIST OF FIGURES – CONTINUED

Figure Page

5.2 Findings reported by Trivy_A, Grype, and CVE-bin-tool for the collection of SBOMs. SBOMs for the same software artifact were run through the selected analysis tools. The findings for SBOMs generated using Syft CycloneDX 1.5 and Trivy_G CycloneDX 1.5 were compared and findings for SBOMs generated using Syft SPDX 2.3 and Trivy_G SPDX 2.3 were compared; thus, variation in vulnerability findings is attributable to impacts of SBOM generation tools Trivy_G and Syft. Subplots A and C hold format constant in CycloneDX 1.5, subplots B and D hold format constant in SPDX 2.3. A-B: Symmetrical density plots depicting the distribution of vulnerabilities across the SBOM collection when using SBOM generation tools Trivy_G and Syft and holding format constant. C-D: Alluvial plots portraying changes in the sum of reported findings by Trivy_A, Grype, and CVE-bin-tool. The stacked bars colors denotes the sum of findings reported for an SBOM built with the generation tools, Trivy and Syft, indicated on the x-axis. Across stacked bars, each SBOM is represented by a thin line. Lines that connect bars of different colors signify SBOMs which had different numbers of findings when using Syft versus Trivy. The ranges in findings associated with the color ramp were determined using Jenks natural breaks optimization (see Methods)........ 41

LIST OF FIGURES – CONTINUED

Figure Page

5.3 Findings reported by Trivy$_A$, Grype, and CVE-bin-tool for the collection of SBOMs. SBOMs for the same software artifact were run through the selected analysis tools. The findings for SBOMs generated using Syft CycloneDX 1.5 and Syft SPDX 2.3 were compared and findings for SBOMs generated using Trivy$_G$ CycloneDX 1.5 and Trivy$_G$ SPDX 2.3 were compared; thus, variation in vulnerability findings is attributable to impacts of SBOM formats CycloneDX 1.5 and SPDX 2.3. Subplots A and C hold generation tool constant with Syft, subplots B and D hold generation tool constant in Trivy$_G$. A-B: Symmetrical density plots depicting the distribution of vulnerabilities across the SBOM collection when using SBOM formats CycloneDX 1.5 and SPDX 2. and holding generation tool constant. C-D: Alluvial plots portraying changes in the sum of reported findings by Trivy$_A$, Grype, and CVE-bin-tool. The stacked bars colors denotes the sum of findings reported for an SBOM built with formats, CycloneDX 1.5 and SPDX 2.3, indicated on the x-axis. Across stacked bars, each SBOM is represented by a thin line. Lines that connect bars of different colors signify SBOMs which had different numbers of findings when using CycloneDX 1.5 versus SPDX 2.3. The ranges in findings associated with the color ramp were determined using Jenks natural breaks optimization (see Methods)........ 42

NOMENCLATURE

SSC	Software Supply Chain
SBOM	Software Bill of Materials
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
GHSA	Github Security Advisory
MSUSEL	Montana State University Software Engineering Lab
DHS	U.S. Department of Homeland Security
NVD	National Vulnerability Database

ABSTRACT

Cybersecurity attacks threaten the lives and safety of individuals around the world. Improving defense mechanisms across all vulnerable surfaces is essential. Among surfaces, the software supply chain (SSC) stands out as particularly vulnerable to cyber threats. This thesis investigates how Software Bill of Materials (SBOM) can be utilized to assess and improve the security of software supply chains. An informal literature review reveals the paucity of studies utilizing SBOM to assess SSC security, which further motivates this research.

Our research adopts the Goal/Question/Metric paradigm with two goals: firstly, to utilize SBOM technology to assess SSC security; secondly, to examine the impact of SBOM generation on vulnerability detection. The study unfolds in two phases. Initially, we introduce a novel approach to assess SSC security risks using SBOM technology. Utilizing analysis tools Trivy and Grype, we identify vulnerabilities across a corpus of 1,151 SBOMs. The second phase investigates how SBOM generation affects vulnerability detection. We analyzed four SBOM corpora derived from 2,313 Docker images by varying the SBOM generation tools (Syft and Trivy) and formats (CycloneDX 1.5 and SPDX 2.3). Using SBOM analysis tools (Trivy, Grype, CVE-bin-tool), we investigated how the vulnerability findings for the same software artifact changed according to the SBOM generation tool and format.

The first phase demonstrates SBOMs use in identifying SSC vulnerabilities, showcasing their utility in enhancing security postures. The subsequent analysis reveals significant discrepancies in vulnerability detection outcomes, influenced by SBOM generation tools and formats. These variations underscore the necessity for rigorous validation and enhancement of SBOM technologies to secure SSCs effectively.

This thesis demonstrates the use of SBOMs in assessing the security of SSCs. We underscore the need for stringent standards and rigorous validation mechanisms to ensure the accuracy and reliability of SBOM data. We reveal how SBOM generation affects vulnerability detection, offering insights that enhanced SBOM methodologies can help improve security. While SBOM is promising for enhancing SSC security, it is clear the SBOM space is immature. Extensive development, validation, and verification of analysis tools, generation tools, and formats are required to improve the usefulness of SBOMs for SSC security.

INTRODUCTION

Security attacks continue to increase in frequency, sophistication, and severity. With the rise of cybersecurity threats, improving defense mechanisms across all vulnerable surfaces is essential. Among surfaces, the software supply chain (SSC) stands out as particularly vulnerable to cyber threats. There was a 650% year-over-year increase in SSC attacks in 2021, up from 430% in 2020 [42]. Software supply chains are composed of a growing number of components including binaries, libraries, tools, and microservices necessary to meet the requirements of modern software. A large part of SSCs are made up of third-party dependencies, incorporated to address software needs [38]. With more software providers integrating third-party code into their SSCs a large cybersecurity attack surface has emerged, revealing a significant security concern [17]. Moreover, the issue is compounded by the potentially many third-party projects that rely on transitive dependencies further expanding the attack surface [14]. Vulnerabilities accumulated along the SSC from upstream dependencies cascade downwards, rendering the core software susceptible to exploitation. Additionally, these exploits can be extremely damaging as seen with the Apache Log4j [4] and Solar Winds [35] attacks. As a result, different technologies have emerged to assist secure SSCs. This thesis focuses on one emerging technology, the Software Bill Of Materials (SBOM).

Software Bill of Materials is a rapidly advancing technology that is becoming a pivotal element in ensuring the security of SSCs. In 2021 the U.S. government issued Executive Order (EO) 14028: Improving the Nation's Cybersecurity[1], which explicitly acknowledges

[1]`https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/`

the SBOM as a crucial component. Software Bills of Materials serve as comprehensive inventories of all software dependencies employed in a specific application or system. This inventory enables security practitioners to detect and mitigate security vulnerabilities in SSCs primarily through the use of SBOM analysis tools.

As SBOM technology evolves, numerous complementary analysis tools (open-source and proprietary) have been developed [3] [5] [30] [16]. These tools ingest SBOMs and detect security weaknesses and vulnerabilities in SSCs through scanning dependencies then performing vulnerability matching against vulnerability databases and utilizing Common Platform Enumeration (CPE)[2]. The use of SBOMs to detect and communicate vulnerabilities present in SSCs plays a vital role in securing SSCs of all software types. Due to the young age of SBOM technology, studies assessing SSC security utilizing SBOMs are lacking. Additionally, studies assessing SBOM tooling directly are few and far between.

In this thesis, I leverage SBOM technology to assess security risks present in SSCs as well as identifies and investigate SBOM aspects that impact vulnerability reporting. My first study, *Assessing Security Risks of Software Supply Chains Using Software Bill of Materials*, presents a novel approach to assess the current state of security risks in SSCs utilizing SBOM technology. This approach is grounded in foundational quality assurance techniques that aggregates outputs from SBOM technology to inform practitioners regarding potential cybersecurity risks. I analyzed SBOMs mined from common open-source repositories and Docker images that cover a wide range of ecosystems and domains, therefore enabling the analysis of third-party SSC security risks in a broader context.

Initial explorations into utilizing SBOM technology to assess SSC security risk unveiled multiple limitations within the space. A key issue is the tooling surrounding SBOM analysis. I uncovered large disparities in vulnerability findings from popular SBOM analysis tools Trivy

[2]`https://nvd.nist.gov/products/cpe`

and Grype [5] [3] (Chapter 4). Additionally, an empirical assessment of the current state of SBOM by Xia et al. [6] revealed, "There is a lack of maturity in SBOM tooling. More reliable, user-friendly, standard-conformable, and interoperable enterprise-level SBOM tools, especially SBOM consumption tools, are needed". These findings served as the impetus for further investigations into the causes of variability in vulnerability reporting from SBOM analysis tools.

Building upon these findings, subsequent work delved deeper into the complexities of SBOM generation and its potential impact on vulnerability detection by SBOM analysis tools. My second study, *Impacts of Software Bill of Materials Generation on Vulnerability Detection*, focuses on how decisions made at the SBOM generation phase impact vulnerability detection by SBOM analysis tools. I assess the impacts of popular SBOM generation tools Syft built by Anchore[3] and Trivy built by Aqua Security[4] as well as the impacts of SBOM specifications Software Package Data Exchange (SPDX)[5] and CycloneDX[6]. Due to the rise in both the use and importance of microservices [43], focus is placed on the microservices domain, specifically Docker[7] images. This work examines the inherent risks associated with blindly trusting the outputs of SBOM generation tools and specifications merely based on their popularity and industry standing and investigates a potentially dangerous gap between perceived reliability and actual performance.

[3]`https://anchore.com/`
[4]`https://www.aquasec.com/`
[5]`https://spdx.github.io/spdx-spec/v2.2/`
[6]`https://cyclonedx.org/docs/`
[7]`https://www.docker.com/`

RELATED WORK

With the growing adoption of third-party code, there has been heightened interest in examining security risks associated with SSCs. Using SBOMs to explore this attack surface is a new approach that motivates our work.

Informal Literature Review of Assessing Software Supply Chain Security Using Software Bill of Materials

To review the current state of utilizing SBOMs to assess SSC security, we conducted an informal literature review. With this review we look to answer the following questions: How are SBOMs being used to assess SSC security? And what are the limitations of leveraging SBOMs to assess SSC security? These answers will give us an understanding of the gaps present in the SBOM for SSC security space. We start with defining our search strategy and then move onto the results of this review.

Search Strategy

Our search strategy was to use IEEE Xplore[1] and ACM Digital Library[2] to search for papers from journals and conferences from 2010-2023. We start our search at 2010 as SBOMs were first introduced in around that time. We are seeking papers that contain ”software bill of materials” and ”software supply chain” and ”security” in the abstract. To accomplish this we used IEEE Xplore’s command search with the command: *”Abstract”:”software bill of materials” AND ”Abstract”:”software supply chain” AND ”Abstract”:”security”*. Additionally we used ACM Digital Library’s advanced search with the command: *Abstract:(”software bill of materials”) AND Abstract:(”software supply chain”)*

[1]`https://ieeexplore.ieee.org/Xplore/home.jsp`
[2]`https://dl.acm.org/`

AND Abstract:("security"). The objective of this search strategy is to keep the search broad in order to identify any and all papers relevant to the using SBOM for SSC security.

To accept a paper, the paper must be focused on a method for using SBOMs to assess SSC security or discuss the limitations of using SBOMs to enhance SSC security. Initially, we read papers' abstracts to eliminate papers that are clearly not focused on the use or limitations of SBOM to assess SSC security. After the initial pass of reading abstracts to reduce the number of papers, we read through the remaining papers in their entirety to determine if the paper fits the criteria and should be accepted. We also inspect the references of papers we gather to identify any additional papers that should be included.

Results

After performing the searches above, four papers were found that match the search terms on IEEE Xplore, and two papers were found that match the search terms on ACM Digital Library. After removing duplicates and investigating references, we were left with five papers [6] [44] [10] [41] [32] to investigate. Assessing the five papers we identify one that is relevant, an empirical study of the SBOM space by Xia et al. [6]. Although this work does not cover how SBOMs are being used to assess SSC security, it uncovers limitations experienced by SBOM users. A key limitation identified is the lack of maturity in the SBOM tooling space, especially in SBOM consumption tools [6]. SBOM tooling lacks reliability, interoperability, and standard-conformation [6]. These findings as well as the scarcity of papers that fit the search criteria indicates a gap in the space and motivates our research on applying SBOMs to enhance SSC security.

Software Supply Chain Security

The paucity of work found in the utilization of SBOMs for SSC security, lead us to expand our investigation to the more broad SSC security space. We found several studies that analyze SSC security risks [15]. These studies tend to focus on open-source development environments and packages managers.

In a study conducted by Yan et al. [14], the authors delved into the attack surface of SSCs. Analyzing 50 open-source Java projects, they aimed to understand three aspects of SSC attacks: the impact of vulnerabilities on dependent components, the vulnerability exposure of supplier software, and the specific vulnerability locations within SSCs. Their findings highlighted the pervasive security risks posed by residual vulnerabilities in open-source SSCs, emphasizing the need for community attention [14]. Additional work has been performed on assessing security risks inherent in common open-source SSCs. Zahan et al. [29] scrutinized 1.63 million JavaScript npm[3] packages, uncovering numerous entry points vulnerable to exploitation by malicious actors. Similarly, Wang et al. [51] examined the security risks associated with Java open-source libraries, revealing significant risks intertwined with library usage.

Other investigations have explored different techniques for identifying SSC security risks. Benthall [39] examined, leveraging vulnerability mapping techniques using public data to evaluate security risks within the supply chain. Benthall utilized data from the NVD to analyze the security posture of OpenSSL[4], a widely adopted open-source project. By connecting NVD data with OpenSSL, Benthall revealed vulnerabilities as well as future risks of vulnerabilities being introduced. Here, the utility of public vulnerability data in assessing security risks is illustrated. Pfretzschner and Othmane take a different angle for to identify SSC security risks. The authors investigate npm packages and propose a tool that can detect four types of SSC attacks through static analysis [8]. This work aims to catch vulnerabilities present in dependencies before they make it into a SSC and highlights the importance of assessing dependency security.

Although these papers and others reveal a better understanding of SSC attacks

[3]`https://www.npmjs.com/`
[4]`https://www.openssl.org/`

through third-party packages and libraries, they often only address one type of development ecosystem, such as Java. This makes it difficult to generalize findings for SSC attacks and the state of software supply chain security. These studies collectively underscore the latent threats emanating from third-party providers within SSCs, furthering our motivation to leverage SBOMs for enhanced SSC security.

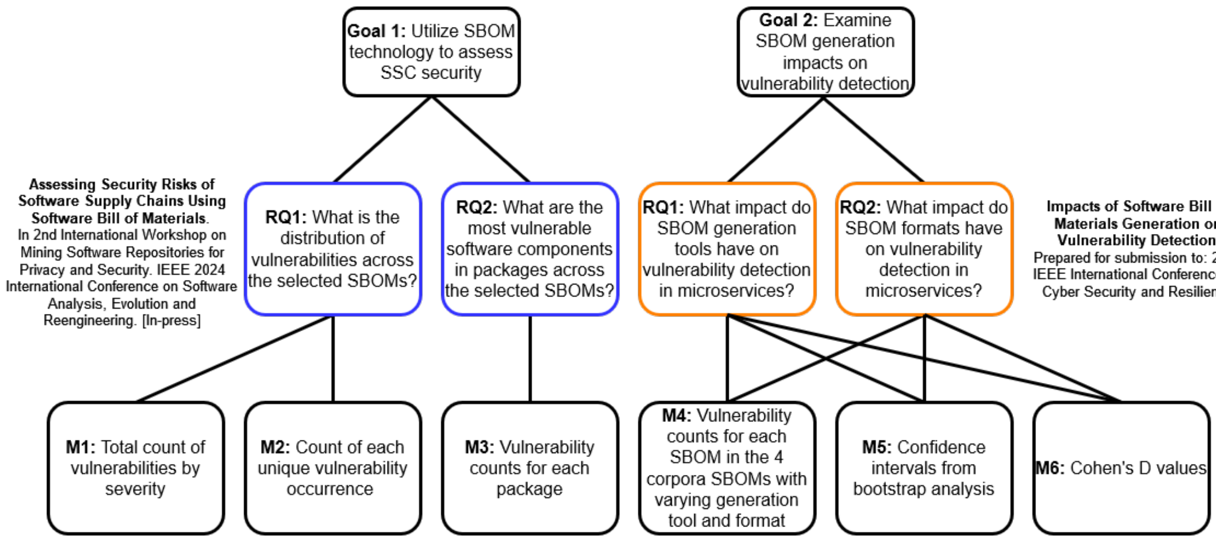

Goal 1: Utilize SBOM technology to assess SSC security
Goal 2: Examine SBOM generation impacts on vulnerability detection
Assessing Security Risks of Software Supply Chains Using Software Bill of Materials. In 2nd International Workshop on Mining Software Repositories for Privacy and Security. IEEE 2024 International Conference on Software Analysis, Evolution and Reengineering. [In-press]
RQ1: What is the distribution of vulnerabilities across the selected SBOMs?
RQ2: What are the most vulnerable software components in packages across the selected SBOMs?
RQ1: What impact do SBOM generation tools have on vulnerability detection in microservices?
RQ2: What impact do SBOM formats have on vulnerability detection in microservices?
Impacts of Software Bill of Materials Generation on Vulnerability Detection. Prepared for submission to: 2024 IEEE International Conference on Cyber Security and Resilience
M1: Total count of vulnerabilities by severity
M2: Count of each unique vulnerability occurrence
M3: Vulnerability counts for each package
M4: Vulnerability counts for each SBOM in the 4 corpora SBOMs with varying generation tool and format
M5: Confidence intervals from bootstrap analysis
M6: Cohen's D values

Figure 3.1: Hierarchical structure of Basili's Goal-Question-Metric methodology for this study [49]. The manuscript information that addresses each research question is placed next to the corresponding research questions. The outline of the research questions aligns with which manuscript addresses them.

Goal Question Metric

In order to utilize SBOM technology to assess security risks present in SSCs as well as identify and investigate SBOM aspects that impact vulnerability reporting, we adopt Basili's Goal-Question-Metric methodology for our research [49]. This method delineates our research into two goals, which we aim to achieve through answering specific questions. These questions are answered through a series of metrics. The specific goals, questions, and metrics are presented below and shown in Figure 3.1.

Goal 1: Utilize SBOM technology to assess SSC security.

- **RQ1** What is the distribution of vulnerabilities across the selected SBOMs?
  - **M1** Total count of vulnerabilities by severity

  - **M2** Count of each unique vulnerability occurrence

- **RQ2** What are the most vulnerable software components in packages across the selected SBOMs?
  - **M3** Vulnerability counts for each package

Goal 2: Examine SBOM generation impacts on vulnerability detection.

- **RQ1** What impact do SBOM generation tools have on vulnerability detection in microservices?
  - **M4** Vulnerability counts for each SBOM in the 4 corpora SBOMs with varying generation tool and format
  - **M5** Confidence intervals from bootstrap analysis
  - **M6** Cohen's D values
- **RQ2** What impact do SBOM formats have on vulnerability detection in microservices?
  - **M4** Vulnerability counts for each SBOM in the 4 corpora SBOMs with varying generation tool and format
  - **M5** Confidence intervals from bootstrap analysis
  - **M6** Cohen's D values

ASSESSING SECURITY RISKS OF SOFTWARE SUPPLY CHAINS USING SOFTWARE BILL OF MATERIALS

Contribution of Authors and Co-Authors

Manuscript in following chapter

Author: Eric O'Donoghue

Contributions: Developed study concept and design, data collection, analysis and interpretation of results, and wrote the manuscript

Co-Author: Ann Marie Reinhold

Contributions: Obtained funding, provided feedback on the analyses, and edited the manuscript.

Co-Author: Clemente Izurieta

Contributions: Obtained funding, provided feedback on the analyses, and edited the manuscript.

Manuscript Information

Eric O'Donoghue, Ann Marie Reinhold, Clemente Izurieta

2nd International Workshop on Mining Software Repositories for Privacy and Security

Status of Manuscript:
____ Prepared for submission to a peer-reviewed journal
____ Officially submitted to a peer-reviewed journal
____ Accepted by a peer-reviewed journal
_X_ Published in a peer-reviewed journal

Publisher: IEEE
In-press

Abstract

The software supply chain is composed of a growing number of components including binaries, libraries, tools, and microservices necessary to meet the requirements of modern software. Products assembled by software vendors are usually comprised of open-source and commercial components. Software supply chain attacks are one of the largest growing categories of cybersecurity threats and the large number of dependencies of a vendor's product makes it possible for a single vulnerability to propagate to many vendor products. Additionally, the software supply chain offers a large attack surface that allows vulnerabilities in upstream transitive dependencies to affect the core software. Software Bill Of Materials (SBOM) is an emerging technology that can be used in tandem with analysis tools to detect and mitigate security vulnerabilities in software supply chains. In this research, we use open-source tools Trivy and Grype to assess the security of 1,151 SBOMs mined from third-party software repositories of various domains and sizes. We explore the distribution of software vulnerabilities across SBOMs and look for the most vulnerable software components. We conclude that this research demonstrates the threat of security via software supply chain vulnerabilities as well as the viability of using SBOMs to help assess security in the software supply chain.

Introduction

Security attacks continue to increase in frequency, sophistication, and severity of breaches. With the rise of cybersecurity threats in recent years, improving defense mechanisms across all vulnerable surfaces is essential. A particularly vulnerable and important software surface facing a large increase in cybersecurity attacks is the software supply chain. According to Sonatype's 2021 report, there was a 650% year-over-year increase in software supply chain attacks, up from 430% in 2020 [22]. With more software providers integrating third-party code into their software supply chains a large cybersecurity attack surface has emerged. This is in large part due to the dependencies that make up the final software product. The final software package can then contain bugs and vulnerabilities collected along the supply chain, leaving the core software vulnerable. Further, this problem is compounded by the potentially many third-party projects that rely on transitive dependencies from additional projects. These upstream dependencies propagate down the supply chain further increasing the attack surface [10]. Herein, we present a novel approach to assess the current state of security risks inherent in project supply chains. We develop an approach grounded in foundational quality assurance techniques that aggregate outputs from Software Bill Of Materials (SBOM) technology to inform practitioners regarding potential cybersecurity risks.

SBOMs provide an inventory of all software components used in a particular application or system and are an emerging technology that is quickly evolving into a fundamental cornerstone of software supply chain security. The United States government issued Executive Order (EO) 14028: Improving the Nation's Cybersecurity, where the SBOM is specifically recognized as a critical component. The consequences of the EO require strict adherence by software providers who wish to engage in business with the US federal government, to deliver SBOMs compliant with standards developed by NTIA [24].

As SBOM technology evolves, numerous complementary static analysis tools (open-source and proprietary) have been developed [3] [4] [17] [11] to help assess their quality. Static analysis tools are external resources that help asses a target product by parsing its source code, byte code, or compiled source code. Results of static analysis tools are typically reported using various metrics, counts or by finding relevant data [8] that could impact the target. Using dedicated static analysis tools on SBOMs to assess the security risks of software supply chains has emerged as a viable approach for detecting potential security vulnerabilities [18] [15] [25].

In recent years several studies have analyzed software supply chain security risks. These studies focus on open-source development environments and package managers [16] [26]. Although these papers and others reveal a better understanding of software supply chain attacks through third-party packages and libraries, they often only address one type of development ecosystem, such as Java. This makes it difficult to generalize findings for software supply chain attacks and the state of software supply chain security. Thus, there exists a need to assess the state of software supply chain security with a broader view. Additionally, due to the young age of SBOM technology, studies assessing software supply chain security utilizing SBOMs are lacking.

In this study we analyze SBOMs mined from common open-source repositories and Docker images that cover a wide range of ecosystems and domains, therefore enabling the analysis of third-party software supply chain security risks in a broader context.

Related Work

With the growing adoption of third-party code, there has been heightened interest in examining security risks associated with supply chains. Using SBOMs to explore this attack surface is a new approach that motivates our work. A search on IEEE Xplore with the string ”Abstract:software bill of materials AND Abstract:software supply chain AND

Abstract:security", returns only two results –neither of which address assessing software supply chain security risks utilizing SBOMs [6] [19].

In a study performed by Yan et al. [14], the authors investigated the attack surface of software supply chains, where they analyzed 50 open-source Java projects to gain an understanding of three key aspects. Specifically, the extent to which dependent components are impacted by vulnerabilities in supply chains, the extent to which the supplier software is affected by vulnerabilities and specific locations of vulnerabilities in software supply chains. A key finding is that "the security problem from the residual vulnerabilities in the open-source software supply chain can have a broad impact on their dependents, which should be captured attention by the community" (sic)[14].

Additional efforts have explored how vulnerability mapping techniques applied to public data help assess security risks in the supply chain. Benthall [21] links data from the National Vulnerability Database (NVD)[1] to the widely used open-source project, OpenSSL. Benthall used NVD data to investigate the security of OpenSSL as well as assess the future risk of vulnerabilities being introduced. The study demonstrates that by linking publicly available vulnerability data to a project that is pervasive across the supply chain, we are able to assess present and potential future security risks.

Other work has been performed that investigates the security risks of common open-source software supply chains. A study by Zahan et al. [16] that analyzed 1.63 million JavaScript npm packages, found multiple entry points containing additional packages that a malicious actor could exploit. In another study performed by Wang et al.[26], the authors examined the risks associated with Java open-source libraries and found many library-associated risks. These studies help exemplify the latent dangers associated with third-party providers of software supply chain packages.

[1] `https://nvd.nist.gov/`

Finally, Haque [5], provides a comprehensive overview of how SBOMs can be used to help enhance software supply chain security. This study assesses the effectiveness of actualizing SBOM security use cases.

Methodology

To assess software supply chain security risks, we define risk as the number of times a vulnerability occurs across the mined software repositories. Associated with each vulnerability is the corresponding severity of the vulnerability. We also define how we find the occurrence of vulnerabilities in the mined SBOMs and how we measure the severity of these vulnerabilities. To obtain a count of vulnerability occurrences we selected the static analysis tools Grype (version 0.53.1) [3] and Trivy (version 0.44.1) [4]. Both of these tools analyze each component present in a given SBOM and use mapping techniques to detect potential vulnerabilities present within the components. Each tool generates a comprehensive report containing all vulnerability occurrences in the SBOM. Both tools report vulnerabilities found from the Common Vulnerabilities and Exposures (CVE)[2] and the GitHub Security Advisories (GHSA)[3]. We selected these tools through feedback from industry partners and subject matter experts. Severity is calculated using the Common Vulnerability Scoring System (CVSS)[4]. We make two key assumptions that have threats to validity implications. We assume that the selected static analysis tools report accurate findings and that CVSS scores are an objective way to measure severity.

Our investigation into mining software repositories for obtaining SBOMs led us to a collection of SBOMs mined from common open-source repositories and Docker images by Interlynk [13]. In this study, we leverage the SBOMs mined by Interlynk. This rich dataset

[2]`https://cve.mitre.org`
[3]`https://github.com/advisories`
[4]`https://www.first.org/cvss`

spans many different domains of software including, for example, utility, machine learning, front-end development, and databases. As the SBOMs are derived from common open-source repositories and Docker images, the dependencies within the SBOMs are expected to be prevalent in software development. Thus, we can gain an understanding of the security of software dependencies commonly used in today's software development landscape. The dataset is stored in a publicly available database that houses SBOM metadata as well as a url to an S3 bucket where the SBOM is stored. The count of third-party dependencies present in the selected SBOMs ranges from 1 to 4,135 with an average dependency count of approximately 350.

Research Questions

We use SBOM technology to investigate the security risks associated with software supply chains. We explore two research questions:

RQ1: What is the distribution of vulnerabilities across the selected SBOMs?

The answer to this question allows practitioners to understand the distribution of vulnerabilities associated with components of software that aggregate third-party software. Special focus is given to high-critical vulnerabilities. We hypothesize that as the occurrence of a vulnerability increases, the severity of the vulnerability will decrease. This is because software developers tend to be more concerned with high-severity vulnerabilities and will issue fixes faster than for low-severity vulnerabilities.

RQ2: What are the most vulnerable software components in packages across the selected SBOMs?

The answer to this question provides practitioners with insights regarding security risks found in common third-party source code used by software developers. By leveraging a repository of SBOMs mined from a wide variety of software, we can analyze a large number of SBOMs that span multiple domains. The implications for practitioners are important in

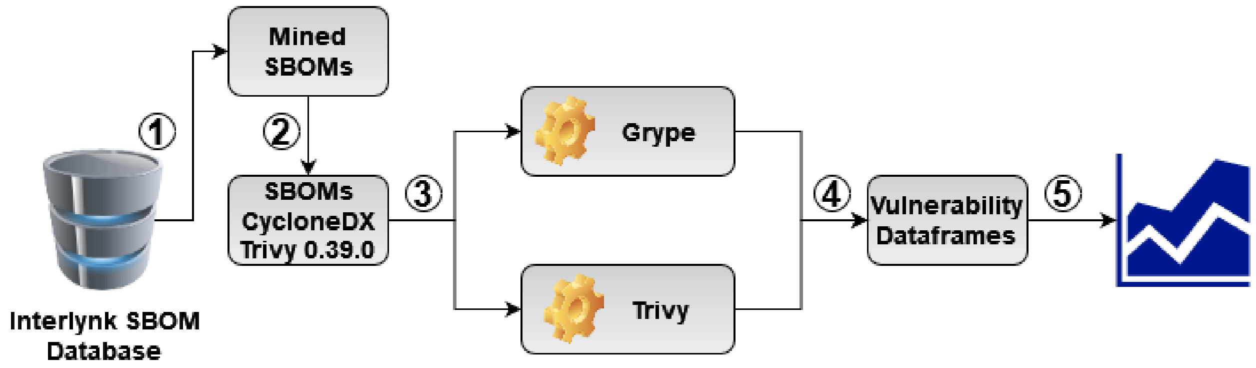


Figure 4.1: The study design consists of 5 steps. Data extraction occurs in steps 1 and 2, data processing is performed in steps 3 and 4, and the analysis is carried out in step 5. © 2024 IEEE.

that it allows them to understand the sources of vulnerable third-party code and whether the risks of including a packet are worth the risks.

Data Extraction and Processing

We analyzed the security of 1,151 SBOMs from software repositories of varying domains and sizes. The design of this study including data extraction, processing, and analysis consists of 5 steps and is depicted in Figure 4.1 .

In *step 1* we extracted and downloaded all SBOMs from Interlynk's database through a combination of SQL queries and HTTP requests. In *step 2* we narrow the selection of SBOMs. We focused on SBOMs created using the generation tool Trivy (version 0.39.0). Next, in *step 3*, we ran the selected static analysis tools, Trivy and Grype, on each SBOM obtained from the previous step and outputted their results in SARIF format. In *step 4*, we extracted the reported vulnerabilities and built two R dataframes. The first dataframe contains each vulnerability found across the mined SBOMs, a count of how many times that vulnerability occurred across the mined SBOMs, and the CVSS score for each vulnerability. The second dataframe contains each dependency found across the mined repositories and the count of unique vulnerabilities found in the dependency. Finally in *step 5* we performed

Table 4.1: Trivy & Grype Findings Total Counts. © 2024 IEEE.

Severity Level	**Trivy**	**Grype**
None	2,920	45
Low	45,380	1,594
Medium	170,422	20,619
High	83,123	17,329
Critical	7,177	3,966
Total	309,022	43,553
Unique Number of Vulnerabilities	7,244	4,061

data analysis focusing on the generation of graphs [5].

Data Analysis

To address **RQ1** we built scatter plots using outputs from Trivy and Grype, shown in Figure 4.2a and Figure 4.2b respectively.

Each scatter plot helps us understand the distribution of vulnerability findings across SBOMs. Every blue dot corresponds to a single vulnerability. The x-axis is the CVSS score for each vulnerability and the y-axis represents the total number of occurrences for a given vulnerability.

To measure the number of occurrences of each vulnerability, we analyzed the static analysis tool results and built a comprehensive list of each unique vulnerability as well as a count of the number of times that a unique vulnerability occurred. The scatter plots provide insights regarding the distribution of vulnerabilities across common third-party code.

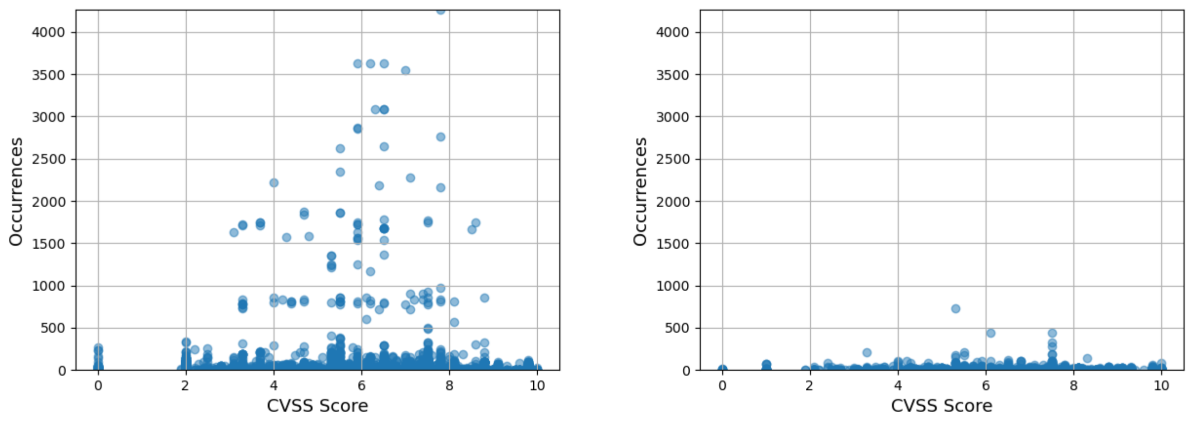
[5]`https://github.com/MSUSEL/msusecl-sbom-security-tool-pipeline`

Additionally, using these plots, we can highlight the presence of high-critical vulnerabilities. Table 4.1, provides details regarding specific counts of low to critical vulnerabilities found by each tool across the selected SBOMs.

To address **RQ2** we built bar graphs with outputs from Trivy and Grype. The plots, shown in Figure 4.3 and Figure 4.4 display the top 25 most vulnerable packages found across the mined SBOMs. We determined the most vulnerable packages through a count of vulnerabilities. Counts for each package were obtained by looking at every vulnerability found by the static analysis tools. We then extracted the metadata associated with each vulnerability, which includes the package where the vulnerability originates from, and used this data to compile a comprehensive list of the total vulnerabilities present in each package across the mined SBOMs.

Results

RQ1: What is the distribution of vulnerabilities across the selected SBOMs? Raw numbers associated with every severity level from all mined SBOMs are shown in Table 4.1. The dataset consists of 1,151 SBOMs. Trivy reported 309,022 vulnerabilities and Grype reported 43,553 vulnerabilities. Further analysis from Figure 4.2a and Figure 4.2b, reveals that our findings differ significantly from our hypothesis that as the occurrence of a vulnerability increases or decreases, the severity will correspondingly decrease or increase. Both tools report a small number of vulnerabilities with low CVSS severity scores. The largest grouping of vulnerabilities occurs in the medium to high range with 82% of Trivy's findings and 87% of Grype's findings assessed as medium or high. Additionally, Trivy reported vulnerabilities that occurred far more often which consisted of mostly medium and high vulnerabilities. Trivy reported a vulnerability as having 4,259 instances across the selected SBOMs. This is a large number of instances for a single vulnerability. These findings demonstrate a need for additional assessments of software supply chain security.



(a) Trivy Findings

(b) Grype Findings

Figure 4.2: Distribution of CVSS vulnerability scores from static analysis tools. Trivy and Grype, labeled respectively. Each blue dot represents a single vulnerability. The y-axis shows the total number of occurrences found for a given vulnerability. © 2024 IEEE.

The data shown in Table 4.1, and figures 4.2a and 4.2b show a significant difference in the results of Trivy and Grype. Trivy found 7,177 unique vulnerabilities and 309,022 total vulnerabilities whereas Grype only found 4,061 unique vulnerabilities and 43,553 total vulnerabilities. Additionally, Trivy reported a large group of vulnerabilities that have a high occurrence across the mined repositories not present in Grype's findings. These findings reveal high variability that makes us question tool accuracy and validity. Specifically, Trivy reported over seven times as many vulnerabilities as Grype.

RQ2: What are the most vulnerable software components in packages across the selected SBOMs? An examination of results shown in figures 4.3 and 4.4 reveals that Trivy found 14 unique packages with vulnerability counts ranging from 204 to 534 and Grype found 8 unique packages with vulnerability counts ranging from 53 to 600. This illustrates that using such packages expands the potential attack surface in a software product, which falls beyond the direct control of the software developer. Additionally, these results show that it is not uncommon for packages to remain vulnerable across tool versions. This is

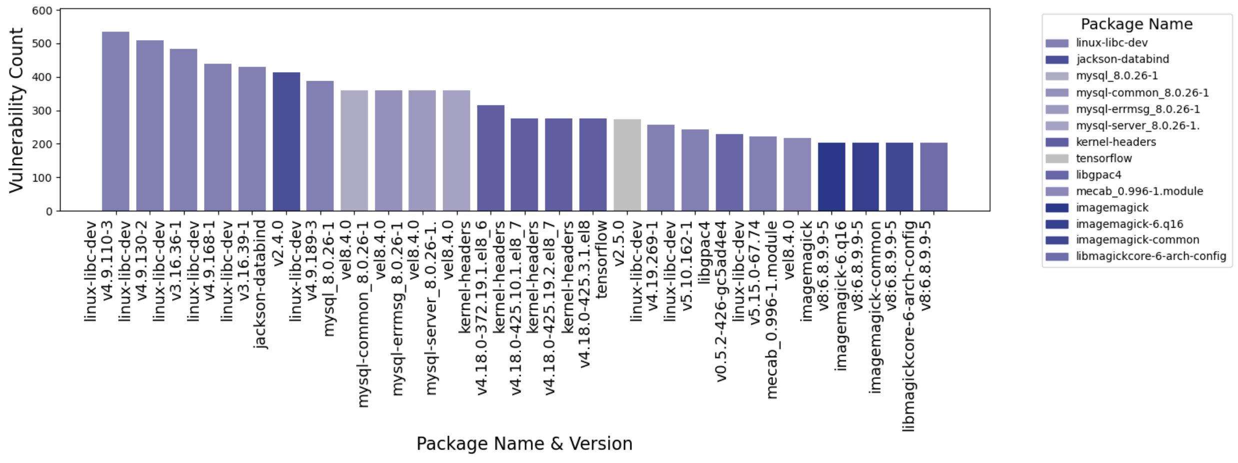


Figure 4.3: Top 25 most vulnerable packages found by Trivy across the mined SBOMs.

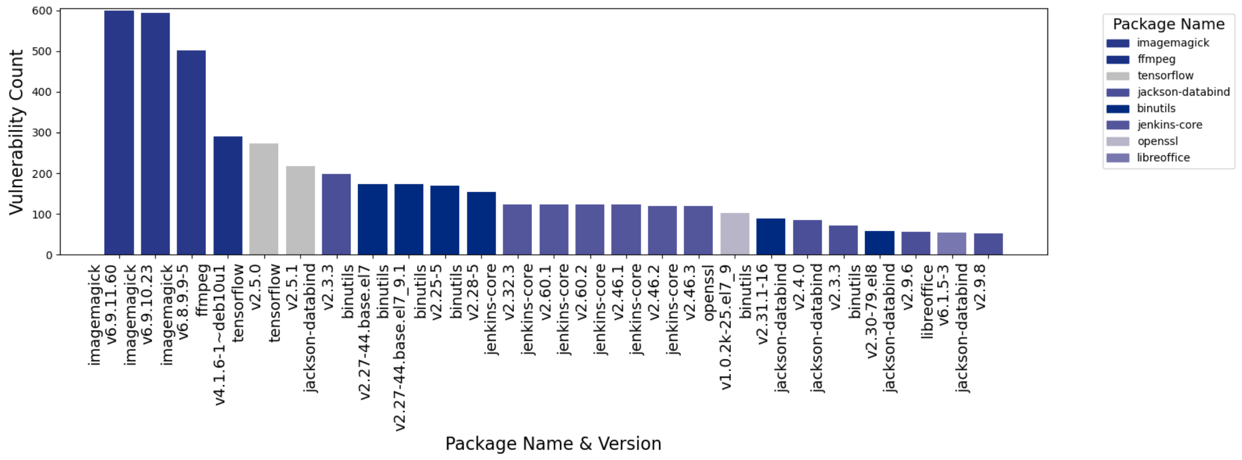


Figure 4.4: Top 25 most vulnerable packages found by Grype across the mined SBOMs. © 2024 IEEE.

made evident by observing the repetition of the same package across different versions in the top 25 most vulnerable packages reported by both tools. For example, Grype reported six versions of Jenkins Core with vulnerability counts ranging from 119-124. Finally, these findings also reveal the disagreements between Trivy and Grype. Only two packages across the top 25 reported by each tool were common and the vulnerability counts vary greatly between the two plots.

Discussion

Our results indicate a significant amount of vulnerabilities in common third-party code. In this study, Trivy found 309,022 vulnerabilities and Grype found 43,553 across 1,151 SBOMs. Additionally, both tools reported a large amount of critical vulnerabilities with Trivy finding 7,244 and Grype finding 3,966. We borrow from the practitioner community to label vulnerabilities with a high or critical CVSS score and a high occurrence rate as *showstoppers*. Results from Trivy show a significant number of showstopper vulnerabilities in the selected SBOMs. A review of Figures 4.2a and 4.2b, shows that Grype's reported showstopper vulnerabilities are hard to identify when comparing the two plots. This is due to Trivy's large amount of findings, causing it to appear that Grype reported almost none. However, a close examination of Figure 4.2b reveals that there are many vulnerabilities reported by Grype with a high CVSS rating that occur hundreds of times over the selected SBOMs. This displays that while Grype found significantly less vulnerabilities than Trivy, it still reported many showstopper vulnerabilities. These findings suggest the potential latent risks of common third-party code.

Although showstopper issues in traditional quality assurance (QA) environments force teams to halt deployment until fixes or patches are issued, it is less clear how the handling of such showstoppers is handled when hidden in packages. The packages identified in our findings are widely used in software development with many familiar names such as OpenSSL, BinUtils, and FFMPEG. The most vulnerable package found by Grype, ImageMagick, has over 10,000 stars on GitHub [12]. Another vulnerable package, Tensorflow, has a 37% market share of the data science and machine learning category [2]. Our results allow software vendors an opportunity to assess the security of packages that they may need to include in their projects. This allows software developers to mitigate potential security risks that can impact the software supply chain.

Finally, a key finding from these results is the disparity in vulnerabilities reported by the selected static analysis tools, Trivy and Grype. Trivy reported over four times as many vulnerabilities as Grype, 3,211 more critical severity vulnerabilities, and over four and half times the number of high severity vulnerabilities (i.e. showstoppers). This has implications in the SBOM tooling space and brings up a lot of questions regarding the validity and accuracy of these tools. Through discussions with our industry partners and subject matter experts, we found that Trivy and Grype are two of the most widely used SBOM static analysis tools in the software supply chain security assessment, thus if Grype is significantly under-reporting vulnerabilities in software supply chains, software providers utilizing Grype in their supply chain security assessment could be uninformed regarding large amounts of vulnerabilities. On the other hand, Trivy could be over-reporting the number of vulnerabilities in software supply chains. This could lead software providers using Trivy to waste effort tracking vulnerabilities that do not exist in their software supply chain. Additionally, the lack of agreement between tools is symptomatic of consistency problems in static analysis tools when reporting results. These problems occur across vendors and even across different versions of the same tool [20].

Threats to Validity

In order to determine security risks, we selected two popular open source static analysis tools, Trivy and Grype, that ingest SBOMs and report vulnerabilities in a software's supply chain. The results of this study are reliant on the ability of Trivy and Grype to report accurate vulnerability findings, therefore the selected tools introduce multiple threats to validity. We examine four different types of threats to validity: construct validity, content validity, internal validity, and external validity; which are based on the classification scheme of Cook, Campbell and Day [23] and of Campbell et al. [9].

In this paper, construct and content validity refer to the meaningfulness of the measurements produced by Grype and Trivy to accurately represent the security risk posed by SBOMs that are used in supply chains. Both tools produce vulnerability counts that are present in SBOMs, however some vulnerabilities reported by the tools may not be exploitable. This can be due to the vulnerability residing in a function that is not called, or the vulnerability being unreachable. Therefore the security risk indicated by these tools could include false positives. To mitigate construct and content validity, we have also made use of the CVSS score metric associated with vulnerabilities.

Internal validity refers to cause and effect relationships between independent and dependent variables [7]. The independent variables in this study include the number of vulnerabilities reported by each Grype and Trivy, and the CVSS score. Our dependent variable is the security risk. There are a number of problems that threaten internal validity. Trivy and Grype each use their own internal vulnerability database to perform vulnerability mapping while scanning SBOMs. Additionally, Trivy and Grype use different data sources. For example Trivy stores Photon Security Advisory[6] information in its vulnerability database while Grype does not. These differences can cause the tools to miss the same vulnerabilities due to differences in their respective internal databases. We also make the assumption that the CVSS score is an accurate measure of a vulnerability's severity, however the subjectivity of this metric allows for human error and uncertainty [14], which can cause vulnerabilities to be assigned inaccurate ratings. Further, not all vulnerabilities are assigned a CVSS score, and assigning a new score increases uncertainty between cause and effect of security risks. To mitigate threats to internal validity we can force both tools to use the same external sources (i.e., databases). Further, recent studies [20] find that different versions of the same static analysis tool often produce different results when analyzing the same inputs–calling the accuracy and trustworthiness of static analysis tools into question. Due to the disparity in findings from Trivy and Grype, it is possible that we selected versions of Grype and

[6]`https://packages.vmware.com/photon/photon_cve_metadata/`

Trivy that contained bugs thus giving inaccurate results. In order to mitigate this threat, a systematic analysis of all versions of each static analysis tool over a large corpus of SBOMs is needed. We hope to perform such analysis in the future as discussed in the future work section.

External Validity refers to the ability to generalize results. Whilst this study attempts to be broad and cover a wide range of software, we only analyzed 1,151 SBOMs. To mitigate this threat, a larger dataset would allow us to randomly sample SBOMs from multiple domains and statistically generalize conclusions to larger populations.

Future Work

Potential areas for future work include the validation of findings found by static analysis tools Trivy and Grype. We would also like to expand the corpus of available tools to further explore SBOM static analysis. This is important, especially to quality assurance engineers and developers, because we want to ensure accurate scoring when assessing software supply chain security risks. Part of this work includes building SBOMs injected with a predetermined and known set of vulnerabilities and then comparing the outputs of the static analysis tools to expected outputs. Additionally, investigating the accuracy of the selected static analysis tools across versions is needed due to the possibility of bugs introducing false positives [20]. Lastly, comparing the findings of SBOM static analysis tools across a large corpus of SBOMs will help increase the significance and power of the results.

Further statistical analysis is planned for the dataset. We will use descriptive statistics to better summarize the dataset. Measures of central tendency, dispersion, and frequency can reveal interesting characteristics about the distributions of vulnerabilities across the selected SBOMs and across results from both tools. To gain deeper insights into the outcomes generated by Grype and Trivy tools, we will first employ unconstrained ordination to better understand overall patterns in the data and follow up with hierarchical cluster analysis.

Cluster analysis groups SBOMs according to the commonalities of vulnerabilities found in them. By clustering on SBOM vulnerabilities, we can potentially identify the third-party code responsible for these similarities. For instance, we can determine the extent to which commonalities in imported libraries are driving the groupings in the cluster analysis. This approach provides valuable additional insights into the security risks associated with software supply chains, and will directly assist software practitioners in identifying packages that introduce vulnerabilities. Finally, we plan on performing controlled experimentation with both tools. The goal is to understand if variations in tool versions and variations in security risk scores across different domains are statistically significant. Depending on the characteristics of the distributions, both parametric and non-parametric techniques are anticipated.

Another interesting area of exploration is whether elements of SBOMs, such as generation tools and specifications influence the ability of SBOM static analysis tools to provide accurate findings. If such an influence exists, the researchers should identify the specific aspects that significantly impact the accuracy of the results.

Finally, investigations that explore software quality modeling techniques and how they can be used to characterize software supply chain security risks are of utmost interest. The development of quality models that utilize findings from SBOM static analysis tools is a potential area of research that would allow integration into CI/CD environments. Software quality modeling approaches would allow software providers to track and monitor the quality of their software supply chain using the evaluation of SBOM structure (i.e., compliance to standards) as well as the content of the SBOM. QA practitioners and users can benefit by setting and maintaining supply chain security quality goals.

Conclusion

This study provided an initial examination of the security risks within the software supply chain, by focusing on 1,151 SBOMs extracted from third-party repositories. These repositories encompass a diverse range of software, spanning various domains and project sizes.

We found large amounts of vulnerabilities present within the software supply chains of the analyzed projects. Additionally, security risks can vary drastically depending on the static analysis tool used. The calibration and empirical validation of current and new SBOM analysis tools continue to require attention. We can also assert that dependencies between packages can further contribute to latent security risks that affect software supply chains. Some packages are much more vulnerable than others given the number of vulnerability counts found by the tools. The most vulnerable packages were found to be very common in software development and these findings will aid software providers when assessing third-party software.

This research demonstrates the potential of SBOMs in evaluating software supply chain security. However, substantial work is still required in the SBOM domain before software providers can completely rely on the results derived from leveraging SBOMs.

Acknowledgements

This research was conducted with the U.S. Department of Homeland Security (DHS) Science and Technology Directorate (S&T) under contract 70RSAT22CB0000005. Any opinions contained herein are those of the authors and do not necessarily reflect those of DHS S&T.

References

[1] © 2024 IEEE. Reprinted, with permission, from E. O'Donoghue, A. Reinhold, C. Izurieta., "Assessing Security Risks of Software Supply Chains Using Software Bill of Materials," in 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering 2nd International Workshop on Mining Software Repositories for Privacy and Security, Rovaniemi, Finland: IEEE, March 2024. [In-press].

[2] 6sense. "TensorFlow - Market Share, Competitor Insights in Data Science And Machine Learning." [Online]. Available: `https://www.6sense.com/tech/data-science-machine-learning/tensorflow-market-share` (Accessed: Nov. 25, 2023).

[3] Anchore Inc. "anchore/grype." GitHub.com. [Online]. Available: `https://github.com/anchore/grype` (Accessed: Sep. 14 2023).

[4] Aqua Security Software Ltd. "aquasecurity/trivy." GitHub.com [Online]. Available: `https://github.com/aquasecurity/trivy` (Accessed: Sep. 14, 2023).

[5] B. M. R. Haque, "An Analysis of SBOM in the Context of Software Supply-chain Risk Management," M.S. thesis, 2023. Accessed: Nov. 23, 2023. [Online]. Available: `https://www.duo.uio.no/handle/10852/103847`

[6] 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 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia: IEEE, May 2023, pp. 2630–2642. doi: 10.1109/ICSE48619.2023.00219.

[7] Cahit KAYA., "Internal validity: A must in research designs.", Educational Research and Reviews 10, no. 2 (2015): 111-118.

[8] D. M. Rice, "An Extensible, Hierarchical Architecture for Analysis of Software Quality Assurance," M.S. thesis, Gianforte School of Computing, Montana State University, Bozeman, 2020

[9] D. T. Campbell, J. C. Stanley, and N. L. Gage. 1963. Experimental and Quasi-experimental Designs for Research. Houghton Mifflin, Boston, MA.

[10] D. Yan, Y. Niu, K. Liu, Z. Liu, Z. Liu and T. F. Bissyandé, "Estimating the Attack Surface from Residual Vulnerabilities in Open Source Software Supply Chain," 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS), Hainan, China, 2021, pp. 493-502, doi: 10.1109/QRS54544.2021.00060.

[11] FOSSA. "Audit-Grade Open Source Dependency Protection." fossa.com [Online]. Available: `https://fossa.com/` (Accessed: Sep. 14, 2023).

[12] ImageMagick Studio LLC. “ImageMagick.” GitHub.com. [Online] Available: https://github.com/ImageMagick/ImageMagick (Accessed: Nov. 25, 2023).

[13] Interlynk. “SBOM Benchmark — Build Better SBOM.” [Online]. Available: https://sbombenchmark.dev (Accessed: Nov. 23, 2023).

[14] Izurieta C., Griffith I., Reimanis D., Luhr R., ”On the Uncertainty of Technical Debt Measurements,” IEEE ICISA 2013, International Conference on Information Science and Applications, Pattaya, Thailand, June 2014. DOI: 10.1109/ICISA.2013.6579461

[15] N. Jovanovic, C. Kruegel, and E. Kirda, “Pixy: a static analysis tool for detecting Web application vulnerabilities,” in 2006 IEEE Symposium on Security and Privacy (S&P’06), Berkeley/Oakland, CA: IEEE, 2006, p. 6 pp. – 263. doi: 10.1109/SP.2006.29.

[16] 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 the 44th International Conference on Software Engineering: Software Engineering in Practice, May 2022, pp. 331–340. doi: 10.1145/3510457.3513044.

[17] NewYork-Presbyterian. “nyph-infosec/daggerboard.” GitHub.com [Online]. Available: https://github.com/nyph-infosec/daggerboard (Accessed: Sep. 14, 2023).

[18] P. Emanuelsson and U. Nilsson, “A Comparative Study of Industrial Static Analysis Tools,” Electronic Notes in Theoretical Computer Science, vol. 217, pp. 5–21, Jul. 2008, doi: 10.1016/j.entcs.2008.06.039.

[19] P. J. Caven, S. R. Gopavaram, and L. J. Camp. “Integrating Human Intelligence to Bypass Information Asymmetry in Procurement Decision-Making,” in MILCOM 2022 - 2022 IEEE Military Communications Conference (MILCOM), Rockville, MD, USA: IEEE, Nov. 2022, pp. 687–692. doi: 10.1109/MILCOM55135.2022.10017736.s

[20] Reinhold A.M., Weber T., Lemak, C, Reimanis D., Izurieta C., ”New Version, New Answer: Investigating Cybersecurity Static-Analysis Tool Findings,” IEEE International Conference on Cybersecurity and Resilience, CSR 2023, Venice Italy, July 2023.

[21] S. Benthall, ”Assessing software supply chain risk using public data,” 2017 IEEE 28th Annual Software Technology Conference (STC), Gaithersburg, MD, USA, 2017, pp. 1-5, doi: 10.1109/STC.2017.8234461.

[22] Sonatype, “The 2021 State of the Software Supply Chain Report.” [Online]. Available: https://www.sonatype.com/resources/state-of-the-software-supply-chain-2021

[23] T. D. Cook, D. T. Campbell, and A. Day. 1979. Quasiexperimentation: Design & Analysis Issues for Field Settings. Houghton Mifflin, Boston, MA

[24] The White House, Executive Order 14028. (2021, May 12). “Executive Order on Improving the Nation’s Cybersecurity.” [Online]. Available: https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

[25] V. B. Livshits and M. S. Lam, ”Finding security vulnerabilities in java applications with static analysis,” in Proceedings of the 14th conference on USENIX Security Symposium - Volume 14 (SSYM’05). USENIX Association, USA, 18.

[26] Y. Wang et al., “An Empirical Study of Usages, Updates and Risks of Third-Party Libraries in Java Projects,” in 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), Adelaide, Australia: IEEE, Sep. 2020, pp. 35–45. doi: 10.1109/ICSME46990.2020.00014.

IMPACTS OF SOFTWARE BILL OF MATERIALS GENERATION ON VULNERABILITY DETECTION

Contribution of Authors and Co-Authors

Manuscript in following chapter

Author: Eric O'Donoghue

Contributions: Developed study concept and design, data collection, analysis and interpretation of results, and wrote the manuscript

Co-Author: Ann Marie Reinhold

Contributions: Obtained funding, provided feedback on the analyses, and edited the manuscript.

Co-Author: Clemente Izurieta

Contributions: Obtained funding, provided feedback on the analyses, and edited the manuscript.

Manuscript Information

Eric O'Donoghue, Ann Marie Reinhold, Clemente Izurieta

2024 IEEE International Conference on Cyber Security and Resilience

Status of Manuscript:
_X_ Prepared for submission to a peer-reviewed journal
____ Officially submitted to a peer-reviewed journal
____ Accepted by a peer-reviewed journal
____ Published in a peer-reviewed journal

Publisher: IEEE
Submission Date: May 2nd 2024

Abstract

The software supply chain (SSC) continues to face cybersecurity threats. To assist in securing SSCs, Software Bill of Materials (SBOM) has emerged as a pivotal technology. Despite the increasing use of SBOM, the influence of SBOM generation on vulnerability detection was unaddressed. We created four corpora of SBOMs from 2,313 Docker images by varying SBOM generation tool (Syft, Trivy) and SBOM format (CycloneDX 1.5, SPDX 2.3). Using three common SBOM analysis tools (Trivy, Grype, CVE-bin-tool), we investigated how the vulnerability findings for the same software artifact changed according to the SBOM generation tool and format. With the complex nature of SBOM generation and consumption, we expected some variation in findings. However, we reveal high variability in vulnerability reporting attributed to SBOM generation. The variation in the quantity of vulnerabilities discovered in the same targets highlights the need for rigorous validation and enhancement of SBOM technologies to best secure SSCs.

Introduction

The software supply chain (SSC) is a vital part of modern software applications and is facing a large increase in cybersecurity attacks [26]. Software supply chains are composed of a growing number of components including binaries, libraries, tools, and microservices necessary to meet the requirements of modern software. A large part of SSCs is made up of third-party dependencies, incorporated to address software needs [24]. The substantial and expanding number of dependencies present in SSCs reveals a significant security concern [10]. A 650% year-over-year increase in software supply chain attacks was found in 2021, up from 430% in 2020 [26]. Additionally, SSC attacks are often extremely damaging as seen with the Apache Log4j [3] and Solar Winds [22] attacks. As a result, different technologies have emerged to assist secure SSCs. Our research focuses on one emerging technology, the Software Bill Of Materials (SBOM).

SBOM is a rapidly advancing technology that is becoming a pivotal element in ensuring the security of SSCs. In 2021 the U.S. government issued Executive Order 14028: Improving the Nation's Cybersecurity[1], which explicitly acknowledges the SBOM as a crucial component. SBOMs serve as comprehensive inventories of all software dependencies employed in a specific application or system. This inventory enables security practitioners to detect and mitigate security vulnerabilities in SSCs. Vulnerability detection is primarily accomplished through the use of SBOM analysis tools. Open-source and proprietary SBOM analysis tools have been developed [2, 4, 9, 18] that detect vulnerabilities in SSCs through vulnerability matching against vulnerability databases as well as Common Platform Enumeration (CPE)[2]. Utilizing SBOMs to detect and communicate vulnerabilities present

[1]https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

[2]https://nvd.nist.gov/products/cpe

in SSCs plays a vital role in securing SSCs of all software types. An important use case of SBOMs is within the microservice domain.

The use and importance of microservices continues to rise[27]. The dynamic and decentralized nature of microservices poses significant challenges for security and vulnerability management [13]. Leveraging SBOMs for microservices enables visibility into their software supply chain. This visibility leads to effective identification and mitigation of security vulnerabilities. In the context of microservice architecture, Docker[3] images have become a popular choice for packaging and deploying individual microservices. They are a popular choice due to their lightweight nature and scalability benefits [28]. Therefore in this study, we focus on SBOM generation for Docker images.

To leverage SBOMs for enhanced SSC security, first an SBOM is built. SBOM generation in the domain of microservices, specifically Docker images, involves understanding and documenting the dependencies present in each image. This process requires tracing software components and libraries bundled within images as well as identifying their relationships. This process is inherently complex and prone to error [17].

Challenges associated with SBOM generation have given rise to numerous SBOM generation tools and formats [17]. We focus on two popular open-source SBOM generation tools that build SBOMs for Docker images, Syft v0.102.0 built by Anchore[4] and Trivy v0.49.0 built by Aqua Security[5]. For each tool, we chose the latest release. SBOM formats are a set of standards that describe how to store and save SBOM information. These formats aid in the automation of accurate vulnerability detection. Formats recognized by Executive Order 14028 include Software Package Data Exchange (SPDX)[6], CycloneDX[7], and Software

[3]`https://www.docker.com/`
[4]`https://anchore.com/`
[5]`https://www.aquasec.com/`
[6]`https://spdx.github.io/spdx-spec/v2.3/`
[7]`https://cyclonedx.org/docs/`

Identification (SWID) tags[8]. Formats SPDX 2.3 and CycloneDX 1.5 are popular and widely used within the SBOM community, thus we focus on them for SBOM generation.

Within SBOM generation, Syft and Trivy operate at the frontier of SBOM and SSC security research and offer a promising solution to the challenging problem of building SBOMs. These tools claim “reliable” and “exceptional” vulnerability scanning for SBOMs generated by them. But it has been found that when using Trivy generated SBOMs, vulnerability findings can vary drastically [19]. Additionally, SBOM formats CycloneDX and SPDX are widely adopted industry standards [5]. As users of SBOM technology who make decisions based on their outputs, we sought to understand the impact of these generation tools and formats on the vulnerability findings of popular SBOM analysis tools. Thus, our research addresses the following questions:

RQ1: What impact do SBOM generation tools have on vulnerability detection in microservices?

RQ2: What impact do SBOM formats have on vulnerability detection in microservices?

We selected the following SBOM analysis tool due to their popularity as well as input from industry partner subject matter experts: Trivy v0.49.0 built by Aqua Security, Grype v0.74.3 built by Anchore, and CVE-bin-tool v3.2.1 built by Intel[9]. Once again for each tool, we chose the latest release. It is important to note that our study does not involve direct comparisons between these tools. Instead, we focus on a ”within-tool” examination, specifically investigating the consistency of findings within each tool across varying SBOM generation methods. Trivy both generates and analyzes SBOMs, therefore we refer to Trivy as Trivy$_G$ when used for generation and as Trivy$_A$ when used for analysis.

[8]`http://dx.doi.org/10.6028/NIST.IR.8060`

[9]`https://www.intel.com/content/www/us/en/developer/topic-technology/open/overview.html`

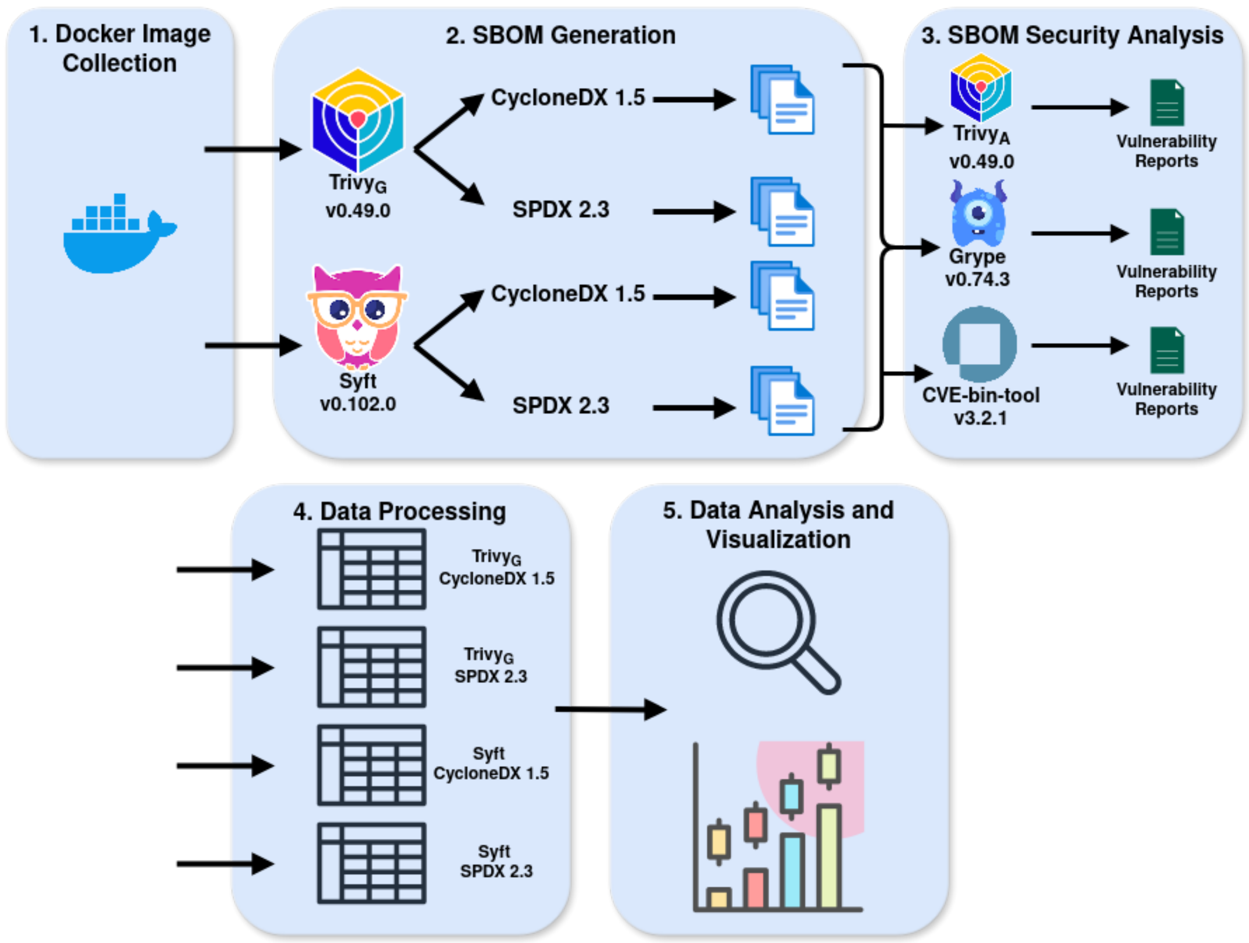


Figure 5.1: Overview of the methodology to study SBOM generation impacts on vulnerability reporting for microservices.

Methodology

To address our research questions, we collected a large corpus of Docker images, from which we created four corpora of SBOMs. We generate the SBOMs using the selected generation tools (Syft and Trivy$_G$) in the selected formats (CycloneDX 1.5 and SPDX 2.3). Next, each corpora of SBOMs is ran through the selected analysis tools. To build our Docker image corpus, we first obtained the tag history for the 100 most pulled images available on docker hub[10]. We omitted two of these images, Tomcat and Drupal, due to not being able to collect their entire tag history. We then selected 25 tags for each docker image spaced evenly throughout the tag history. If an image had less than 25 tags we selected all available tags. Additionally, we omitted any images in which either of the selected SBOM generation tools failed. The design of this study including data extraction, processing, and analysis consists of five steps and is depicted in Fig. 5.1.

Importantly, to attribute the variation in tool output to either generation tool or format we held as many factors as we could constant. One, we assessed the impacts of SBOM generation on the same version of each SBOM analysis tool; thus, differences in vulnerability reporting will not stem from differences in vulnerability reporting across versions [23]. Two, when investigating the impact of the generation tool on vulnerability scanning, we held the format constant. That is, we compare the analysis tool findings for SBOMs generated with Syft CycloneDX 1.5 with findings for SBOMs generated with Trivy_G CycloneDX 1.5 and we compared findings for SBOMs generated with Syft SPDX 2.3 with findings for SBOMs generated with Trivy_G SPDX 2.3. Finally, when investigating the impact of format on vulnerability reporting, we held the generation tool constant. That is, we compare the analysis tool findings for SBOMs generated with Syft CycloneDX 1.5 with the results for SBOMs generated with Syft SPDX 2.3 and we compared the findings for SBOMs generated with Trivy_G CycloneDX 1.5 with the findings for SBOMs generated with Trivy_G SPDX 2.3.

In comparing findings reported by the same analysis tools while varying SBOM generation we remove any entries in which the analysis tool failed to run. This means the results collected are dependent on what data is being compared. The corpus size for each generation and analysis tool configuration are presented in Figs. 5.2 and 5.3.

We followed the analytical methodology of Reinhold et al. [23] to guide data analysis.

[10]`https://hub.docker.com/`

We employed Python 3.10.12 with the Numpy [12], Matplotlib [14], and Seaborn [30] packages, along with the R statistical computing environment [21]. For plotting, we utilized "ggplot2" [11] with colors selected from the "viridis" package [25]. To capture variability in findings, we took a two-pronged approach. First, the overall findings for each SBOM analysis tool in each SBOM dataset were depicted using violin plots (Fig. 5.2A-B & Fig. 5.3A-B). Second, Jenks natural breaks optimization[11] was used to categorize the total findings, which were then plotted for each tool and corpus in Sankey plots (Fig. 5.2C-D & Fig. 5.3C-D). Results did not follow a normal distribution and the differences were not symmetrical thus parametric testing and non-parametric testing would not accurately test the data. Therefore, we employed bootstrapping to assess the impact of SBOM generation on findings. We used 20,000 bootstrap samples and a sample size of $N - 1$, where N represents the total dataset size. Each sample underwent iterative resampling with replacement. This method allowed us to calculate mean point estimates and confidence intervals. Through this approach, the variability in vulnerability detection attributed to differences in SBOM generation was thoroughly explored. Additionally, we investigated the practical significance (effect size) of the results using Cohen's D values [16].

This investigation can be replicated using the data science pipeline located on our GitHub page at **https://github.com/MSUSEL/msusecl-sbom-generation-and-analysis-pipeline**.

Results

The influence of SBOM generation on vulnerability detection is evident. SBOM generation tools demonstrate significant impacts on vulnerability detection, while SBOM formats have a less clear impact on vulnerability detection.

[11]"BAMMtools" package's "getJenksBreaks" function [7]

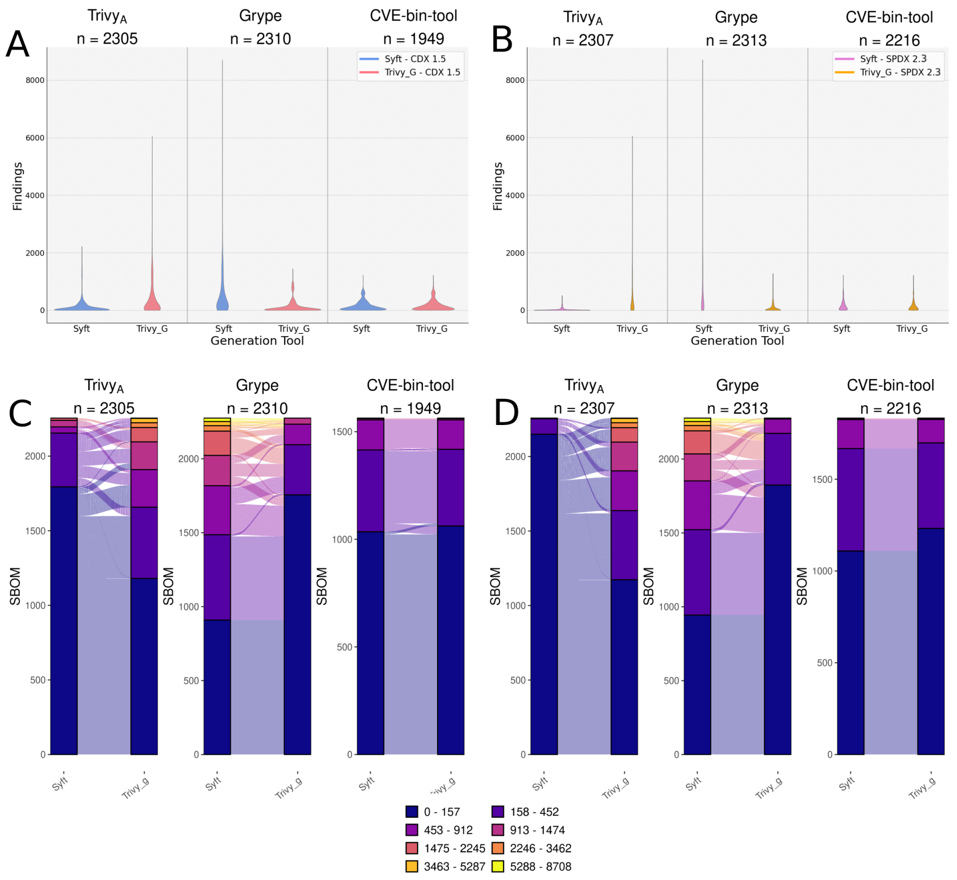


Figure 5.2: Findings reported by Trivy$_A$, Grype, and CVE-bin-tool for the collection of SBOMs. SBOMs for the same software artifact were run through the selected analysis tools. The findings for SBOMs generated using Syft CycloneDX 1.5 and Trivy$_G$ CycloneDX 1.5 were compared and findings for SBOMs generated using Syft SPDX 2.3 and Trivy$_G$ SPDX 2.3 were compared; thus, variation in vulnerability findings is attributable to impacts of SBOM generation tools Trivy$_G$ and Syft. Subplots A and C hold format constant in CycloneDX 1.5, subplots B and D hold format constant in SPDX 2.3. A-B: Symmetrical density plots depicting the distribution of vulnerabilities across the SBOM collection when using SBOM generation tools Trivy$_G$ and Syft and holding format constant. C-D: Alluvial plots portraying changes in the sum of reported findings by Trivy$_A$, Grype, and CVE-bin-tool. The stacked bars colors denotes the sum of findings reported for an SBOM built with the generation tools, Trivy and Syft, indicated on the x-axis. Across stacked bars, each SBOM is represented by a thin line. Lines that connect bars of different colors signify SBOMs which had different numbers of findings when using Syft versus Trivy. The ranges in findings associated with the color ramp were determined using Jenks natural breaks optimization (see Methods).

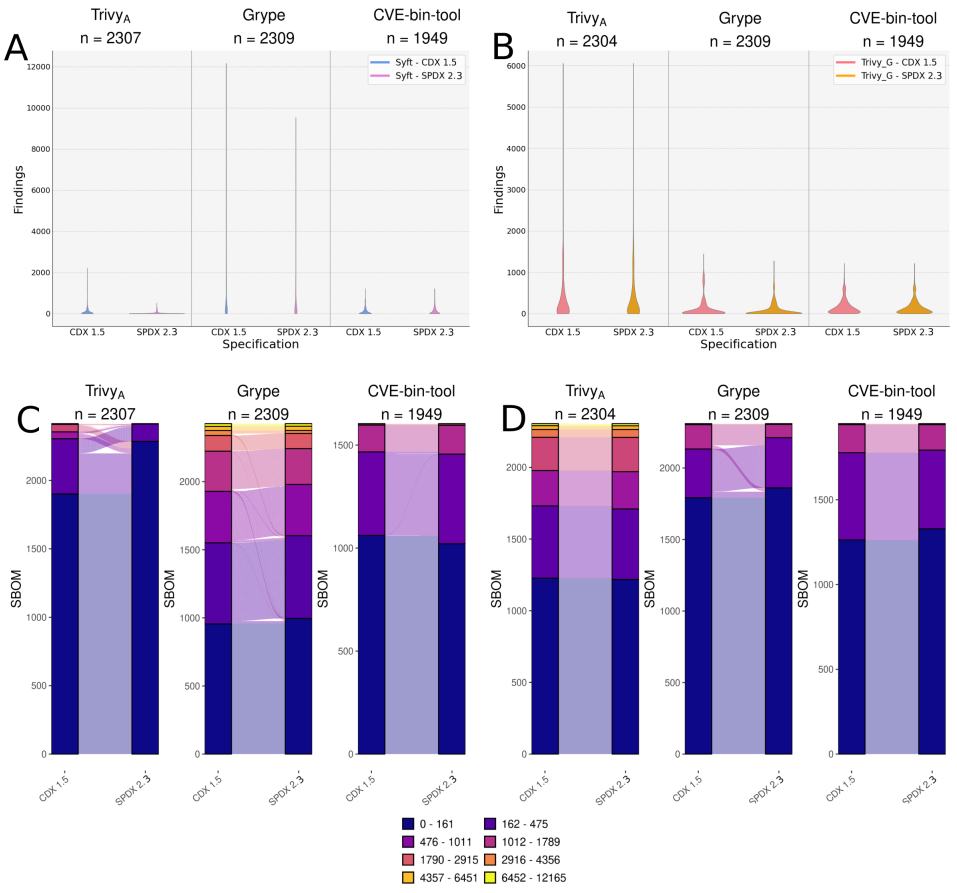


Figure 5.3: Findings reported by Trivy$_A$, Grype, and CVE-bin-tool for the collection of SBOMs. SBOMs for the same software artifact were run through the selected analysis tools. The findings for SBOMs generated using Syft CycloneDX 1.5 and Syft SPDX 2.3 were compared and findings for SBOMs generated using Trivy$_G$ CycloneDX 1.5 and Trivy$_G$ SPDX 2.3 were compared; thus, variation in vulnerability findings is attributable to impacts of SBOM formats CycloneDX 1.5 and SPDX 2.3. Subplots A and C hold generation tool constant with Syft, subplots B and D hold generation tool constant in Trivy$_G$. A-B: Symmetrical density plots depicting the distribution of vulnerabilities across the SBOM collection when using SBOM formats CycloneDX 1.5 and SPDX 2. and holding generation tool constant. C-D: Alluvial plots portraying changes in the sum of reported findings by Trivy$_A$, Grype, and CVE-bin-tool. The stacked bars colors denotes the sum of findings reported for an SBOM built with formats, CycloneDX 1.5 and SPDX 2.3, indicated on the x-axis. Across stacked bars, each SBOM is represented by a thin line. Lines that connect bars of different colors signify SBOMs which had different numbers of findings when using CycloneDX 1.5 versus SPDX 2.3. The ranges in findings associated with the color ramp were determined using Jenks natural breaks optimization (see Methods).

Table 5.1: Confidence intervals from bootstrap analysis comparing vulnerability detection by SBOM analysis tools when varying SBOM generation tool and format. We calculated vlaues as follows: Rows 1 and 2, subtracted mean findings for Syft generated SBOMs from mean findings for Trivy$_G$ generated SBOMs and divided by their pooled standard deviation. Rows 3 and 4, subtracted mean findings for Syft generated SBOMs from mean findings for Trivy$_G$ generated SBOMs and divided by their pooled standard deviation.

	Trivy$_\mathbf{A}$	**Grype**	**CVE-bin-tool**
Trivy$_\mathbf{G}$ vs Syft (CDX)	290.81, 342.62	-488.14, -423.66	-7.24, -4.09
Trivy$_\mathbf{G}$ vs Syft (SPDX)	384.17, 443.38	-502.92, -435.10	-25.95, -21.62
CDX vs SPDX (Trivy$_\mathbf{G}$)	4.81, 7.71	-38.24, -29.27	-10.23, -7.82
CDX vs SPDX (Syft)	-103.32, -85.24	-59.57, -30.76	7.54 10.52

In this section, we explore the effects of SBOM generation tools Syft and Trivy$_G$, on vulnerability detection using SBOM analysis tool Trivy$_A$, Grype, and CVE-bin-tool. Then, we investigate the effects of SBOM formats CycloneDX 1.5 and SPDX 2.3, on vulnerability detection using SBOM analysis tools, Trivy$_A$, Grype, and CVE-bin-tool.

RQ1: What impact do SBOM generation tools have on vulnerability detection in microservices?

Findings from SBOM analysis tools are highly dependent on the vendor that produces the generation tool and analysis tool. More specifically, when the same vendor is used for the generation and analysis of an SBOM, the number of findings over the corpus of SBOMs is higher.

Employing the vendor, Anchore, to generate (Syft) and analyze (Grype) the corpus of SBOMs resulted in the highest variation in findings. The high variation indicates substantial discrepancies in vulnerability detection outcomes. Utilizing the vendor, Aqua Security, to generate (Trivy$_G$) and analyze (Trivy$_A$) the corpus of SBOMs, we encountered the second highest variation, again showing notable differences in vulnerability assessment results when

Table 5.2: Cohen's D values measuring the effect size of variation in vulnerability detection by SBOM analysis tools when varying SBOM generation tool and format. We calucated values as follows: Rows 1 and 2, subtracted mean findings for Syft generated SBOMs from mean findings for Trivy$_G$ generated SBOMs and divided by their pooled standard deviation. Rows 3 and 4, subtracted mean findings for SPDX 2.3 SBOMs from mean findings for CycloneDX 1.5 SBOMs and divided by their pooled standard deviation SBOMs.

	Trivy$_\mathbf{A}$	**Grype**	**CVE-bin-tool**
Trivy$_\mathbf{G}$ vs Syft (CDX)	0.58	0.65	0.09
Trivy$_\mathbf{G}$ vs Syft (SPDX)	0.80	0.69	0.03
CDX vs SPDX (Trivy$_\mathbf{G}$)	0.01	0.17	0.04
CDX vs SPDX (Syft)	0.51	0.04	0.12

the same vendor is used to generate and analyze SBOMs. The disparities in findings from the same analysis tool when using different generation tools, as illustrated in Fig. 5.2, are not only statistically significant (Table 5.1) but also practically significant (Table 5.2). These statistical metrics illustrate the extent of the observed differences and highlight the influence of the vendor responsible for developing the generation and analysis tool.

In assessing the SBOM corpus with an analysis tool (CVE-bin-tool) that did not have affiliation with the generation tools vendor, findings remained consistent. Despite the statistical significance of these findings, as evidenced by the confidence intervals not crossing 0 (Table 5.1), the consistency of the findings is underscored by a small effect size (Table 5.2). This suggests that while the observed differences may be statistically significant, their practical relevance is relatively minor. This emphasizes the lack of variability in vulnerability reporting when using generation and analysis from different vendors.

When varying SBOM generation tool the reported number of findings for identical software artifacts rarely remained consistent. With respect to Trivy$_A$, only 14% of SBOMs generated with different tools in CycloneDX 1.5 yield the same number of findings. The median number of findings for SBOMs generated with Trivy$_G$ was 102 higher than in SBOMs generated with Syft (Fig. 5.2A). The extent of the variability in the findings is further shown (Fig. 5.2A and Fig. 5.2C) with differences ranging from 94 fewer to 5456 greater findings (standard deviation [SD] = 627). Similar trends were observed when analyzing SBOMs generated with different tools in the SPDX 2.3 format, with 11% of SBOMs found to have the same number of findings. The median number of findings for SBOMs generated with Trivy$_G$ exceeded those generated with Syft by 138 (Fig. 5.2B). Similarly, the high variability in the findings is further demonstrated (Fig. 5.2B and Fig. 5.2D) with differences ranging from 65 fewer to 6058 greater findings (standard deviation [SD] = 717).

With Grype, only 4% of SBOMs exhibited an identical number of findings when varying the generation tool in the CycloneDX 1.5 format. The median number of findings for SBOMs generated with Syft exceeded those generated with Trivy$_G$ by 198 (Fig. 5.2A). Again, the extent of variability is shown (Fig. 5.2A and Fig. 5.2C) with differences ranging from 164 fewer to 8171 greater findings (SD = 782). As with Trivy$_A$, similar trends were observed here. In SPDX 2.3, just 5% of SBOMs yield the same number of Grype findings. The median number of findings reported for SBOMs generated with Syft was 200 higher those generated with Trivy$_G$. The variability in the findings is further displayed (Fig. 5.2B and Fig. 5.2D) with differences ranging from 2 fewer to 8202 greater findings (standard deviation [SD] = 627).

Finally, with respect to CVE-bin-tool findings, 51% of CycloneDX 1.5 SBOMs yield identical findings (28% for SPDX 2.3). The median number of findings reported by CVE-bin-tool for SBOMs generated with Syft was just 7 higher than the findings for SBOMs generated with Trivy$_G$ in CycloneDX 1.5 (Fig. 5.2A) and only 19 higher for SBOMs generated in SPDX 2.3 (Fig. 5.2B). A closer look at Figs. 5.2C-D revealed that a significant majority of SBOMs (CycloneDX 1.5: 96%, SPDX 2.3: 92%) reported findings within the same range. Additionally, a pragmatic assessment of the findings' practical significance (Table 5.2), suggests negligible differences. These findings underscore the lack of variability in vulnerability reporting when using generation and analysis from different vendors.

RQ2: What impact do SBOM formats have on vulnerability detection in microservices?

SBOM format influences the SBOM analysis tools outcomes. Although format effects may not be as pronounced as that of the generation tool used, it remains a crucial factor. Inconsistencies or ambiguities in these formats can lead to these discrepancies in vulnerability reporting.

Analyzing the SBOM corpus with Trivy$_A$, 21% of Syft-generated SBOMs yield consistent findings, while 79% of Trivy$_G$ SBOMs exhibit consistent findings. The median number of findings reported by Trivy$_A$ for SBOMs generated with Trivy$_G$ in CycloneDX 1.5 falls short of those in SPDX 2.3 by 4 (Fig. 5.3A), with differences ranging from 567 fewer to 2 greater findings (SD = 35). For SBOMs generated with Syft in CycloneDX 1.5, the median was 34 higher than in SPDX 2.3 (Fig. 5.3B), with differences ranging from 2104 fewer to 32 greater findings (SD = 226).

Analyzing the SBOM corpus with Grype, 83% of Syft-generated SBOMs yield consistent findings, while 32% of Trivy$_G$ SBOMs exhibit consistent findings. The median number of findings reported by Grype for SBOMs generated with Trivy$_G$ in CycloneDX 1.5 exceeded those in SPDX 2.3 by 14 (Fig. 5.3A), with differences ranging from 3 fewer to 654 greater findings (SD = 110). Similarly, for SBOMs generated with Syft in CycloneDX 1.5, the median was 13 higher than in SPDX 2.3 (Fig. 5.3B), with differences ranging from 15 fewer to 12,112 greater findings (SD = 110).

Finally, when analyzing the SBOM corpus with CVE-bin-tool, 56% of Syft generated SBOMs show identical findings, while 78% of Trivy$_G$ SBOMs yield consistent findings. The median number of findings reported by CVE-bin-tool for SBOMs generated with Trivy$_G$ in

CycloneDX 1.5 exceeded those in SPDX 2.3 by 8 (Fig. 5.3A), with differences ranging from 22 fewer to 263 greater findings (SD = 26). Similarly, for SBOMs generated with Syft in CycloneDX 1.5, the median was 21 lower than in SPDX 2.3 (Fig. 5.3B), with differences ranging from 382 fewer to 192 greater findings (SD = 30).

These descriptive statistics appear to suggest that SBOM formats have an impact on vulnerability detection. However, investigating the practical significance of these results reveals a different story. Only when comparing Trivy$_A$ results for SBOMs generated with Syft in CycloneDX 1.5 versus SPDX 2.3 we found practical significance (Table 5.2). This is an instance where the vendor generating and analyzing the SBOM corpus is different. As our results from **RQ1** show this could be the cause of this variability and not due to format. Although using different SBOM formats does cause variability in vulnerability reporting, the effect size of the variability is small, especially in comparison to the generation tool.

Discussion

Security practitioners need confidence in the SBOMs they are building to assess SSC security. Understanding the impact that SBOM generation tools and SBOM formats have on vulnerability reporting, is crucial to effectively secure SSCs.

Building SBOMs is a particularly challenging objective due to the complexity of SSCs and SBOM generation. More specifically, SBOM generation tools have the challenging objective of capturing dependencies within SSCs for a given software artifact. The observed high variability in findings when using different SBOM generation tools (Fig. 5.2) illustrates this challenge. SBOM formats are complex because they need to provide support for storing SBOMs that enable accurate vulnerability detection. As demonstrated in our study, variations in SBOM formats (Fig. 5.3) can introduce variability in vulnerability detection outcomes, further complicating risk assessment efforts for security practitioners.

With respect to SBOM generation tools, we posit that the observed variability could be attributed to the vendor used for the generation of the SBOM in combination with the vendor used for the analysis of the SBOM. When generation and analysis tools from the same vendor are used, significantly more vulnerabilities are found. On the other hand when a different vendor is used for generation and then analysis, far fewer vulnerabilities are found with more consistent results. As SBOMs are difficult to create, developers who possess insights into the SBOM build process can more effectively construct analysis tools that get the maximum information out of the SBOM.

Concerning the SBOM format, we posit that differences in vulnerability reporting when varying SBOM format came from inconsistencies or ambiguities in the formats. For example, there are disagreements between CycloneDX 1.5 and SPDX 2.3 on which field supplier information should be placed [1]. This can lead to analysis tools not being able to find supplier names. As the supplier name is used for CPE matching, this reduces the analysis tool's ability to report vulnerabilities. These inconsistencies and others reduce the ability of SBOM analysis tools to report consistent vulnerability findings.

The observed variability from varying SBOM generation methods decreases practitioners' confidence in the SBOMs being built with popular tools and formats. In turn, this hampers their confidence in leveraging SBOM technology for enhanced SSC security.

Implications for Practitioners and End Users

SBOM is a promising SSC security solution that enables practitioners and end-users to quickly mitigate software supply chain security risks [20] [15]. The variability in tool output demonstrates that neither of the popular SBOM generation tools (Syft and Trivy$_G$) or SBOM formats (CycloneDX 1.5 and SPDX 2.3) have solved the complex challenge of building SBOMs that enable consistent vulnerability reporting. Variability from SBOM generation methods presents a multifaceted challenge for practitioners and end-users alike, casting a shadow of uncertainty over the reliability and trustworthiness of SBOMs. While these tools and formats are promising, it is clear the SBOM space is immature. Extensive development, validation, and verification of both SBOM generation tools and formats are required to improve the usefulness of SBOMs for SSC security.

Moreover, our research highlights the inherent risks associated with blindly trusting the outputs of SBOM generation tools and formats merely based on their popularity and industry standing. We reveal a dangerous gap between perceived reliability and actual performance, underscoring the need for cautious skepticism and rigorous evaluation in adopting these technologies. Software developers and security practitioners must recognize that popularity and industry recognition alone do not guarantee the accuracy and consistency of vulnerability reports derived from SBOMs. While SBOM provides a promising solution to the difficult challenge of securing SSCs, it is clear that a meticulous approach involving thorough scrutiny and validation of SBOMs and SBOM technology is essential to ensure the integrity and effectiveness leveraging SBOM for SSC security measures.

Threats to Validity

Threats to the validity of this study rest in the selected SBOM generation and analysis tools. We examine four different types of threats to validity: construct validity, content validity, internal validity, and external validity; which are based on the classification scheme of Cook, Campbell and Day [28] and of Campbell et al. [8].

In this study construct and content validity refer to the meaningfulness of the measurements produced by the SBOM analysis tools Trivy$_A$, Grype, and CVE-bin-tool. These tools produce vulnerability counts that are present in SBOMs, we use these counts to assess variability introduced by SBOM generation. Recent studies [23] find that different versions of the same static analysis tool often produce different results across the same input. It is possible that we selected versions of the analysis tool that contained bugs thus giving inaccurate results. In order to mitigate this threat, a systematic analysis of all versions of each SBOM analysis tool over a large corpus of SBOMs is needed. We hope to perform such analysis in the future as discussed in the future work section.

Internal validity refers to cause and effect relationships between independent and dependent variables [6]. The independent variables in this study include both the SBOM generation tool as well as the SBOM format. Our dependent variable is the findings reported by selected SBOM analysis tools. As with analysis tools there is the possibility of the selected generation tool versions to contain bugs. This could result in incomplete or incorrect SBOMs, leading to discrepancies in vulnerability reporting. Additionally, the SBOM analysis tools each use their own internal database which pulls from multiple data sources i.e. Redhat[12], Gitlab Advisory Database[13]. It is possible that depending on when internal databases are updated it could lead to different findings that are not attributed to SBOM generation tool or SBOM format. To mitigate this threat each SBOM was run concurrently through the analysis tools.

External Validity refers to the ability to generalize results. Whilst this study attempts to be broad and cover a wide range of software, we only analyzed SBOMs generated from 100 unique docker images and ∼25 versions of those images. To mitigate this threat, a larger dataset would allow us to randomly sample SBOMs from multiple domains and statistically generalize conclusions to larger populations. Additionally we only assessed the impacts of two SBOM generation tools on three SBOM analysis tools. Expanding the corpus of both SBOM generation tools as well as SBOM analysis tools would enable us to further generalize the impacts of SBOM generation tool on SBOM analysis tool's ability to report vulnerabilities.

[12]`https://access.redhat.com/documentation/en-us/red_hat_security_data_api`
[13]`https://advisories.gitlab.com/about/index.html`

Future Work

Potential areas for future work include the validation of the SBOMs being generated by Syft and Trivy_G. This will include the collection of software artifacts and obtaining a ground truth for each artifact of what the SBOM should contain, then comparing this ground truth against the SBOMs generated by Syft and Trivy_G. Additionally, expanding the corpus of SBOM generation tools would allow us to further generalize their impacts and develop a deeper understanding of how vendor of the generation tool and analysis tool plays a role in vulnerability detection.

Another interesting area of exploration is investigating what dependencies are commonly missed by SBOM generation tools as well as a broader assessment of the most used dependencies across an SBOM corpus. This is important as it would enable security practitioners to better understand potential attack targets present in their systems.

Finally, investigations that explore how software quality modeling techniques could be used to help address variability when using different SBOM generation tool, analysis tool combinations are of the utmost interest. Integrating multiple SBOM analysis tools into a single model would assist in having the broadest coverage of vulnerabilities. Additionally utilizing multiple analysis tools could possibly lessen the impacts from the SBOM generation tool vendor.

Conclusion

Our study delves into Software Bill Of Materials (SBOM), a critical technology in SSC security. Through the analysis of a large and diverse corpus of Docker images, we've uncovered how SBOM generation tools and formats impact vulnerability reporting within the microservice domain. We emphasize the substantial role of SBOM generation tools in vulnerability detection, showcasing significant variability in outcomes. Additionally, SBOM formats exert influence. Their impact is less significant but still affects consistent vulnerability reporting. These insights hold significance for practitioners and end-users, illuminating the challenges in constructing dependable SBOMs for SSC security and emphasizing the necessity for rigorous validation and enhancement within the SBOM space.

Acknowledgements

This research was conducted with the U.S. Department of Homeland Security (DHS) Science and Technology Directorate (S&T) under contract 70RSAT22CB0000005. Any opinions contained herein are those of the authors and do not necessarily reflect those of DHS S&T.

References

[1] Alrich, Tom. ”Introduction to SBOM and VEX.” 2024.

[2] Anchore Inc. “anchore/grype.” GitHub.com. [Online]. Available: https://github.com/anchore/grype(Accessed:Sep.142023).

[3] Apache Log4j Security Vulnerabilities. (2021). Log4J. [Online]. Available: https://logging.apache.org/log4j/2.x/security.html

[4] Aqua Security Software Ltd. “aquasecurity/trivy.” github.com [Online]. Available: https://github.com/aquasecurity/trivy (Accessed: Sep. 14, 2023).

[5] 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 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia: IEEE, May 2023, pp. 2630–2642. doi: 10.1109/ICSE48619.2023.00219.

[6] Cahit KAYA., ”Internal validity: A must in research designs.”, Educational Research and Reviews 10, no. 2 (2015): 111-118.

[7] D. Rabosky, M. Grundler, C. Anderson, P. Title, J. Shi, J. Brown, H. Huang, and J. Larson, “BAMMtools: an r package for the analysis of evolutionary dynamics on phylogenetic trees,” Methods in Ecology and Evolution, vol. 5, pp. 701–707, 2014.

[8] D. T. Campbell, J. C. Stanley, and N. L. Gage. 1963. Experimental and Quasi-experimental Designs for Research. Houghton Mifflin, Boston, MA.

[9] FOSSA. “Audit-Grade Open Source Dependency Protection.” fossa.com [Online]. Available: https://fossa.com/ (Accessed: Sep. 14, 2023).

[10] Gkortzis, A., Feitosa, D., Spinellis, D., 2021. ”Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities.” J. Syst. Softw. 172, 110653.

[11] H. Wickham, ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016. [Online]. Available: https://ggplot2.tidyverse.org.

[12] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.

[13] Hemanth Gopal, Guanqun Song, Ting Zhu, ”Security, Privacy and Challenges in Microservices Architecture and Cloud Computing- Survey,” 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2212.14422

[14] J. D. Hunter, ”Matplotlib: A 2D Graphics Environment”, Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

[15] L. Vaas. *One Year After Log4Shell, Firms Still Struggle to Hunt Down Log4j.* (2022). Constast Security. [Online]. Available: https://www.constastsecurity.com/security-influencers/one-year-after-log4shell-firms-still-struggle-to-hunt-down-log4j

[16] Lee, D., "Alternatives to P value: confidence interval and effect size." Korean journal of anesthesiology vol. 69,6 (2016): 555-562. doi:10.4097/kjae.2016.69.6.555.

[17] M. Balliu et al., "Challenges of Producing Software Bill of Materials for Java," in IEEE Security & Privacy, vol. 21, no. 6, pp. 12-23, Nov.-Dec. 2023, doi: 10.1109/MSEC.2023.3302956.

[18] NewYork-Presbyterian. "nyph-infosec/daggerboard." github.com [Online]. Available: `https://github.com/nyph-infosec/daggerboard(Accessed:Sep.14,2023)`.

[19] O'Donoghue, E., Reinhold, A. M., and Izurieta, C. (2024). Assessing Security Risks of Software Supply Chains Using Software Bill of Materials. In 2nd International Workshop on Mining Software Repositories for Privacy and Security. IEEE International Conference on Software Analysis, Evolution and Reengineering. [In-press]

[20] P. Roberts. "Log4j is why you need a software bill of materials (SBOM)." Reversing Labs. Accessed: April 28th, 2024. [Online]. Available: `https://www.reversinglabs.com/blog/log4j-is-why-you-need-an-sbom`

[21] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2022. [Online]. Available: `https://www.R-project.org`.

[22] R. Alkhadra, J. Abuzaid, M. AlShammari, and N. Mohammad, "Solar winds hack: In-depth analysis and countermeasures," in Proc. 12th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT), Jul. 2021, pp. 1–7

[23] Reinhold A.M., Weber T., Lemak, C, Reimanis D., Izurieta C., "New Version, New Answer: Investigating Cybersecurity Static-Analysis Tool Findings," IEEE International Conference on Cybersecurity and Resilience, CSR 2023, Venice Italy, July 2023.

[24] Russ Cox. 2019. "Surviving Software Dependencies." Commun. ACM 62, 9 (September 2019), 36–43. `https://doi.org/10.1145/3347446`

[25] S. Garnier, N. Ross, R. Rudis, A. P. Camargo, M. Sciaini, and C. Scherer, viridis - Colorblind-Friendly Color Maps for R, 2021, r package version 0.6.2. [Online]. Available: `https://sjmgarnier.github.io/viridis/`.

[26] Sonatype, "The 2021 State of the Software Supply Chain Report." [Online]. Available: `https://www.sonatype.com/resources/state-of-the-software-supply-chain-2021`

[27] Söylemez M, Tekinerdogan B, Kolukısa Tarhan A. "Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review." Applied Sciences. 2022; 12(11):5507. `https://doi.org/10.3390/app12115507`

[28] T. D. Cook, D. T. Campbell, and A. Day. 1979. Quasiexperimentation: Design & Analysis Issues for Field Settings. Houghton Mifflin, Boston, MA

[29] Vase, Tuomas. "Advantages of Docker." B.S. thesis, 2015. Accessed: Mar. 28, 2024. [Online]. Available: `https://jyx.jyu.fi/handle/123456789/48029#`

[30] Waskom, M. L., (2021). seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021, https://doi.org/10.21105/joss.03021.

CONCLUSION

Software supply chains are a vital part of all modern software. The intricate web of dependencies within SSCs has introduced a large attack surface that is susceptible to extremely damaging attacks. In response to this challenge, Software Bill of Materials technology has emerged as a promising solution for enhancing SSC security. I investigate SBOM technology including how it can be leveraged to assess SSC security and its limitations.

I provide an initial examination of utilizing SBOM to assess the security risks within SSCs. I assessed the SSC chain security of 1,151 popular open source repositories and Docker images using SBOM technology. Leveraging SBOM technology, I found large amounts of vulnerabilities within SSCs as well as high security risks associated with popular projects (Chapter 4). Additionally, these findings assert that commonly used dependencies contain numerous vulnerabilities (Chapter 4). I show that utilizing SBOM enables security practitioners to assess the security risk present within SSCs. However, my investigations also revealed clear challenges and limitations associated with using SBOM for SSC security. Notably, I found significant variability in security risks depending on the SBOM analysis tool used (Chapter 4).

I investigated the cause of variability in SBOM analysis tool vulnerability reporting. SBOM generation is a difficult process [27], thus I focused on if decisions made at the SBOM generation stage impacted vulnerability reporting. Specifically I focused on the impacts of SBOM generation tool and SBOM format. Through the analysis of a large and diverse corpus of Docker images, I uncover how SBOM generation tools and formats impact SBOM vulnerability reporting within the microservice domain. I emphasize the substantial role of SBOM generation tools in vulnerability detection, showcasing significant variability in outcomes (Chapter 5). Additionally, I found that SBOM formats exert influence (Chapter

5). Their impact is less significant but still affects consistent vulnerability reporting. I highlight the inherent risks associated with blindly trusting the outputs of SBOM generation tools and specifications merely based on their popularity and industry standing. I reveal a dangerous gap between perceived reliability and actual performance, underscoring the need for rigorous evaluation in adopting SBOM technology for SSC security risk assessment.

This thesis contributes to the ongoing efforts to enhance SSC security and demonstrates the potential of SBOMs in evaluating software supply chain security. However, it is clear that substantial work is still required in the SBOM domain before software providers can completely rely on the results derived from leveraging SBOMs. By addressing these challenges, we can better strengthen our defenses against cybersecurity threats and safeguard critical software systems and infrastructure.

REFERENCES CITED

[1] © 2024 IEEE. Reprinted, with permission, from E. O'Donoghue, A. Reinhold, C. Izurieta., "Assessing Security Risks of Software Supply Chains Using Software Bill of Materials," in 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering 2nd International Workshop on Mining Software Repositories for Privacy and Security, Rovaniemi, Finland: IEEE, March 2024. [In-press].

[2] 6sense. "TensorFlow - Market Share, Competitor Insights in Data Science And Machine Learning." [Online]. Available: https://www.6sense.com/tech/data-science-machine-learning/tensorflow-market-share (Accessed: Nov. 25, 2023).

[3] Anchore Inc. "anchore/grype." GitHub.com. [Online]. Available: https://github.com/anchore/grype (Accessed: Sep. 14 2023).

[4] Apache Log4j Security Vulnerabilities. (2021). Log4J. [Online]. Available: https://logging.apache.org/log4j/2.x/security.html

[5] Aqua Security Software Ltd. "aquasecurity/trivy." GitHub.com [Online]. Available: https://github.com/aquasecurity/trivy (Accessed: Sep. 14, 2023).

[6] 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 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia: IEEE, May 2023, pp. 2630–2642. doi: 10.1109/ICSE48619.2023.00219.

[7] B. M. R. Haque, "An Analysis of SBOM in the Context of Software Supply-chain Risk Management," M.S. thesis, 2023. Accessed: Nov. 23, 2023. [Online]. Available: https://www.duo.uio.no/handle/10852/103847

[8] Brian Pfretzschner, Lotfi ben Othmane. "Identification of Dependency-based Attacks on Node.js," in Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES '17). Association for Computing Machinery, New York, NY, USA, Article 68, 1–6. https://doi.org/10.1145/3098954.3120928

[9] Cahit KAYA., "Internal validity: A must in research designs.", Educational Research and Reviews 10, no. 2 (2015): 111-118.

[10] D. E. Hyeon, J. H. Park and H. Y. Youm, "A secure firmware and software update model based on blockchains for Internet of Things devices using SBOM," 2023 18th Asia Joint Conference on Information Security (AsiaJCIS), Koganei, Japan, 2023, pp. 53-58, doi: 10.1109/AsiaJCIS60284.2023.00019.

[11] D. M. Rice, “An Extensible, Hierarchical Architecture for Analysis of Software Quality Assurance,” M.S. thesis, Gianforte School of Computing, Montana State University, Bozeman, 2020

[12] D. Rabosky, M. Grundler, C. Anderson, P. Title, J. Shi, J. Brown, H. Huang, and J. Larson, “BAMMtools: an r package for the analysis of evolutionary dynamics on phylogenetic trees,” Methods in Ecology and Evolution, vol. 5, pp. 701–707, 2014.

[13] D. T. Campbell, J. C. Stanley, and N. L. Gage. 1963. Experimental and Quasi-experimental Designs for Research. Houghton Mifflin, Boston, MA.

[14] D. Yan, Y. Niu, K. Liu, Z. Liu, Z. Liu and T. F. Bissyandé, ”Estimating the Attack Surface from Residual Vulnerabilities in Open Source Software Supply Chain,” 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS), Hainan, China, 2021, pp. 493-502, doi: 10.1109/QRS54544.2021.00060.

[15] E. O’Donoghue, A. Reinhold, C. Izurieta., ”Assessing Security Risks of Software Supply Chains Using Software Bill of Materials,” in 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering 2nd International Workshop on Mining Software Repositories for Privacy and Security, Rovaniemi, Finland: IEEE, March 2024. [In-press].

[16] FOSSA. “Audit-Grade Open Source Dependency Protection.” fossa.com [Online]. Available: `https://fossa.com/` (Accessed: Sep. 14, 2023).

[17] Gkortzis, A., Feitosa, D., Spinellis, D., 2021. ”Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities.” J. Syst. Softw. 172, 110653.

[18] H. Wickham, ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016. [Online]. Available: `https://ggplot2.tidyverse.org`.

[19] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.

[20] Hemanth Gopal, Guanqun Song, Ting Zhu, ”Security, Privacy and Challenges in Microservices Architecture and Cloud Computing- Survey,” 2022. [Online]. Available: `https://doi.org/10.48550/arXiv.2212.14422`

[21] ImageMagick Studio LLC. “ImageMagick.” GitHub.com. [Online] Available: `https://github.com/ImageMagick/ImageMagick` (Accessed: Nov. 25, 2023).

[22] Interlynk. “SBOM Benchmark — Build Better SBOM.” [Online]. Available: `https://sbombenchmark.dev` (Accessed: Nov. 23, 2023).

[23] Izurieta C., Griffith I., Reimanis D., Luhr R., ”On the Uncertainty of Technical Debt Measurements,” IEEE ICISA 2013, International Conference on Information Science and Applications, Pattaya, Thailand, June 2014. DOI: 10.1109/ICISA.2013.6579461

[24] J. D. Hunter, ”Matplotlib: A 2D Graphics Environment”, Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

[25] Lee, D., “Alternatives to P value: confidence interval and effect size.” Korean journal of anesthesiology vol. 69,6 (2016): 555-562. doi:10.4097/kjae.2016.69.6.555.

[26] L. Vaas. *One Year After Log4Shell, Firms Still Struggle to Hunt Down Log4j.* (2022). Constast Security. [Online]. Avaliable: https://www.constastsecurity.com/security-influencers/one-year-after-log4shell-firms-still-struggle-to-hunt-down-log4j

[27] M. Balliu et al., ”Challenges of Producing Software Bill of Materials for Java,” in IEEE Security & Privacy, vol. 21, no. 6, pp. 12-23, Nov.-Dec. 2023, doi: 10.1109/M-SEC.2023.3302956.

[28] N. Jovanovic, C. Kruegel, and E. Kirda, “Pixy: a static analysis tool for detecting Web application vulnerabilities,” in 2006 IEEE Symposium on Security and Privacy (S&P’06), Berkeley/Oakland, CA: IEEE, 2006, p. 6 pp. – 263. doi: 10.1109/SP.2006.29.

[29] 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 the 44th International Conference on Software Engineering: Software Engineering in Practice, May 2022, pp. 331–340. doi: 10.1145/3510457.3513044.

[30] NewYork-Presbyterian. “nyph-infosec/daggerboard.” GitHub.com [Online]. Available: `https://github.com/nyph-infosec/daggerboard` (Accessed: Sep. 14, 2023).

[31] P. Emanuelsson and U. Nilsson, “A Comparative Study of Industrial Static Analysis Tools,” Electronic Notes in Theoretical Computer Science, vol. 217, pp. 5–21, Jul. 2008, doi: 10.1016/j.entcs.2008.06.039.

[32] P. J. Caven, S. R. Gopavaram, and L. J. Camp. “Integrating Human Intelligence to Bypass Information Asymmetry in Procurement Decision-Making,” in MILCOM 2022 - 2022 IEEE Military Communications Conference (MILCOM), Rockville, MD, USA: IEEE, Nov. 2022, pp. 687–692. doi: 10.1109/MILCOM55135.2022.10017736.s

[33] P. Roberts. ”Log4j is why you need a software bill of materials (SBOM).” Reversing Labs. Accessed: April 28th, 2024. [Online]. Available: `https://www.reversinglabs.com/blog/log4j-is-why-you-need-an-sbom`

[34] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2022. [Online]. Available: `https://www.R-project.org`.

[35] R. Alkhadra, J. Abuzaid, M. AlShammari, and N. Mohammad, ”Solar winds hack: In-depth analysis and countermeasures,” in Proc. 12th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT), Jul. 2021, pp. 1–7

[36] Reinhold A.M., Weber T., Lemak, C, Reimanis D., Izurieta C., ”New Version, New Answer: Investigating Cybersecurity Static-Analysis Tool Findings,” IEEE International Conference on Cybersecurity and Resilience, CSR 2023, Venice Italy, July 2023.

[37] NTIA, “The Minimum Elements For a Software Bill of Materials (SBOM).” [Online]. Available: https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf

[38] Russ Cox. 2019. ”Surviving Software Dependencies.” Commun. ACM 62, 9 (September 2019), 36–43. https://doi.org/10.1145/3347446

[39] S. Benthall, ”Assessing software supply chain risk using public data,” 2017 IEEE 28th Annual Software Technology Conference (STC), Gaithersburg, MD, USA, 2017, pp. 1-5, doi: 10.1109/STC.2017.8234461.

[40] S. Garnier, N. Ross, R. Rudis, A. P. Camargo, M. Sciaini, and C. Scherer, viridis - Colorblind-Friendly Color Maps for R, 2021, r package version 0.6.2. [Online]. Available: https://sjmgarnier.github.io/viridis/.

[41] S. Nocera, S. Romano, M. D. Penta, R. Francese and G. Scanniello, ”Software Bill of Materials Adoption: A Mining Study from GitHub,” 2023 IEEE International Conference on Software Maintenance and Evolution (ICSME), Bogotá, Colombia, 2023, pp. 39-49, doi: 10.1109/ICSME58846.2023.00016.

[42] Sonatype, “The 2021 State of the Software Supply Chain Report.” [Online]. Available: https://www.sonatype.com/resources/state-of-the-software-supply-chain-2021

[43] Söylemez M, Tekinerdogan B, Kolukısa Tarhan A. ”Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review.” Applied Sciences. 2022; 12(11):5507. https://doi.org/10.3390/app12115507

[44] T. Bi, B. Xia Z. Xing, Q. Lu, and L. Zhu. ”On the Way to SBOMs: Investigating Design Issues and Solutions in Practice,” in 2024 ACM Trans. Softw. Eng. Methodol. Just Accepted (March 2024). https://doi.org/10.1145/3654442

[45] T. D. Cook, D. T. Campbell, and A. Day. 1979. Quasiexperimentation: Design & Analysis Issues for Field Settings. Houghton Mifflin, Boston, MA

[46] The White House, Executive Order 14028. (2021, May 12). “Executive Order on Improving the Nation’s Cybersecurity.” [Online]. Available: https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

[47] V. B. Livshits and M. S. Lam, ”Finding security vulnerabilities in java applications with static analysis,” in Proceedings of the 14th conference on USENIX Security Symposium - Volume 14 (SSYM’05). USENIX Association, USA, 18.

[48] Vase, Tuomas. ”Advantages of Docker.” B.S. thesis, 2015. Accessed: Mar. 28, 2024. [Online]. Available: `https://jyx.jyu.fi/handle/123456789/48029`

[49] Victor R Basili, Gianluigi Caldiera, and H dieter Rombach. The goal question metric approach. *Encyclopedia of Software Engineering*, 2:528-532, 1994.

[50] Waskom, M. L., (2021). seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021, `https://doi.org/10.21105/joss.03021`.

[51] Y. Wang et al., “An Empirical Study of Usages, Updates and Risks of Third-Party Libraries in Java Projects,” in 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), Adelaide, Australia: IEEE, Sep. 2020, pp. 35–45. doi: 10.1109/ICSME46990.2020.00014.