Assessing Security Risks of Software Supply Chains Using Software Bill of Materials

Eric O'Donoghue
Gianforte School of Computing
Montana State University
Bozeman, USA
ericodonoghue@montana.edu

Ann Marie Reinhold
Gianforte School of Computing
Montana State University
Bozeman, USA
annmarie.reinhold@montana.edu

Clemente Izurieta
Gianforte School of Computing
Montana State University
Pacific Northwest National Laboratory
Idaho National Laboratory
Bozeman, USA
clemente.izurieta@montana.edu

***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.**

***Index Terms*—Software Supply Chain Security, Software Bill of Materials, Mining Software Repositories, Third Party Code**

I. 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 [1]. 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 [15]. 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 [2].

As SBOM technology evolves, numerous complementary static analysis tools (open-source and proprietary) have been developed [7] [8] [9] [10] 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 [3] 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 [4] [5] [6].

In recent years several studies have analyzed software supply chain security risks. These studies focus on open-source development environments and package managers [11] [12]. 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.

II. 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 [13] [14].

In a study performed by Yan et al. [15], 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) [15].

Additional efforts have explored how vulnerability mapping techniques applied to public data help assess security risks in the supply chain. Benthall [16] 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. [11] 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. [12], 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.

Finally, Haque [17], 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.

III. 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) [7] and Trivy (version 0.44.1) [8]. 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 [18]. In this study, we leverage the SBOMs mined by Interlynk. This rich dataset 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.

A. 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.

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

[2]https://cve.mitre.org

[3]https://github.com/advisories

[4]https://www.first.org/cvss

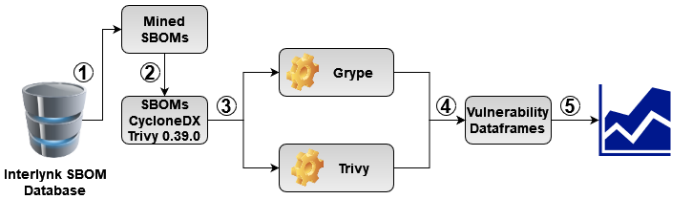


Fig. 1: The study design consists of 5 steps. Data extraction occurs in steps 1 and 2, data processing is performed in steps 3 and 4, and the analysis is carried out in step 5.

TABLE I: Trivy & Grype Findings Total Counts

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

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 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.

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

C. Data Analysis

To address **RQ1** we built scatter plots using outputs from Trivy and Grype, shown in Figure 2a and Figure 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.

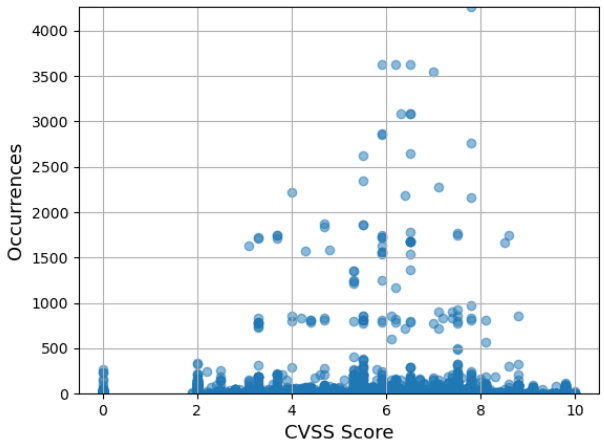
To measure the number of occurrences of each vulnerability, we analyzed the static analysis tool results and built a comprehensive list of each unique vulnerability as well as a count of the number of times that a unique vulnerability occurred. The scatter plots provide insights regarding the distribution of vulnerabilities across common third-party code. Additionally, using these plots, we can highlight the presence of high-critical vulnerabilities. Table I, 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 3 and Figure 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.

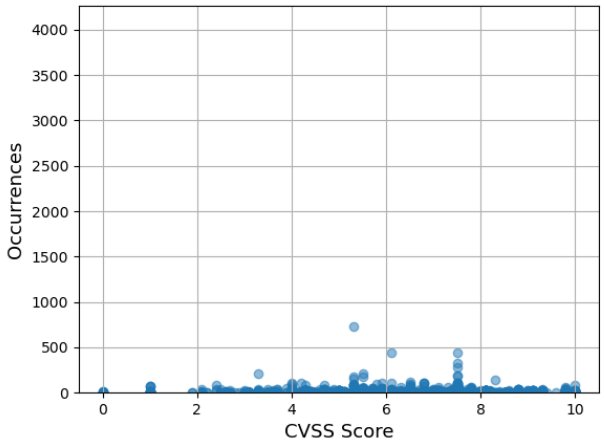
IV. Results

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 I. The dataset consists of 1,151 SBOMs. Trivy reported 309,022 vulnerabilities and Grype reported 43,553 vulnerabilities. Further analysis from Figure 2a and Figure 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.

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



(a) Trivy Findings



(b) Grype Findings

Fig. 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.

The data shown in Table I, and figures 2a and 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 3 and 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 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.

V. 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 2a and 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 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 [19]. Another vulnerable package, Tensorflow, has a 37% market share of the data science and machine learning category [20]. 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

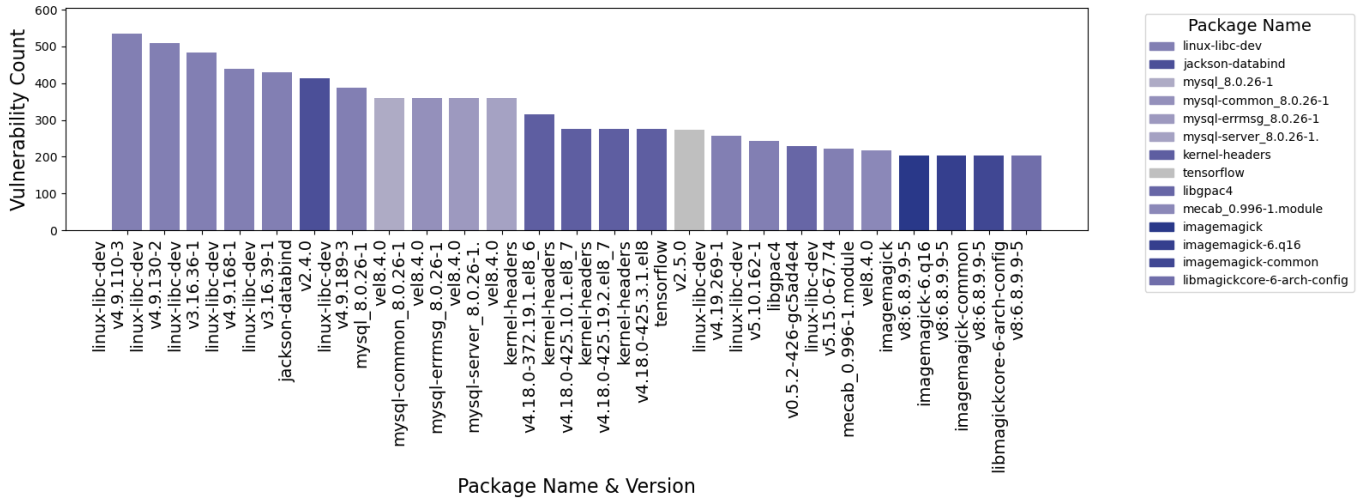


Fig. 3: Top 25 most vulnerable packages found by Trivy across the mined SBOMs.

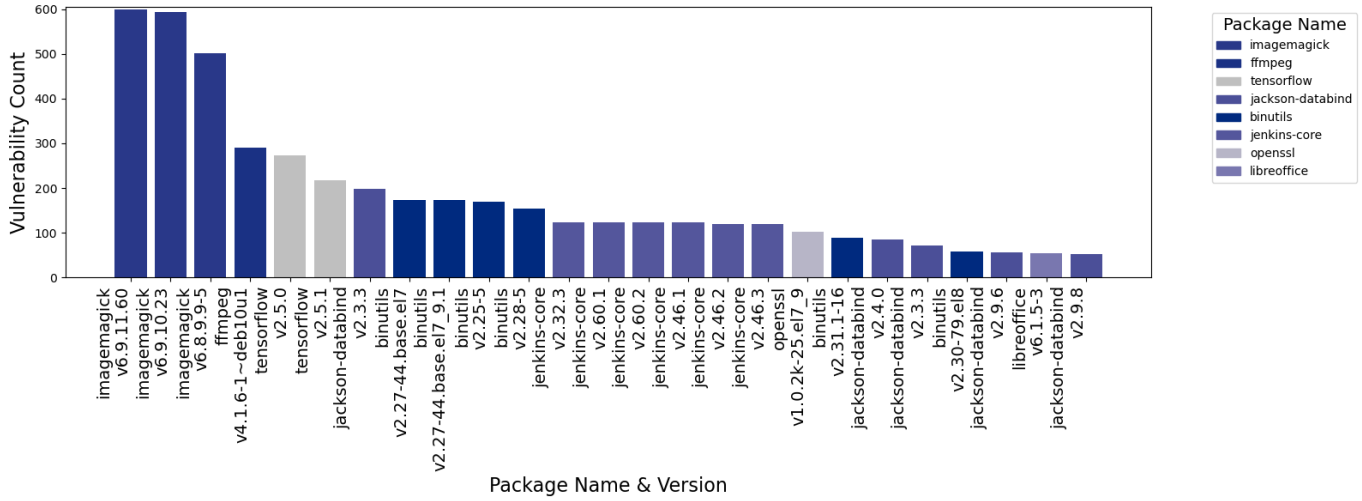


Fig. 4: Top 25 most vulnerable packages found by Grype across the mined SBOMs.

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 [21].

VI. 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 [22] and of Campbell et al. [23].

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 [25]. 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 [24], 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 [21] 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 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.

VII. 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 [21]. 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.

VIII. Conclusion

This study provided an initial examination of the security risks within the software supply chain, by focusing on 1,151

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

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.

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

[2] 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/

[3] 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

[4] 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.

[5] 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.

[6] 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.

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

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

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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

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

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

[20] 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).

[21] 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.

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

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

[24] 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. 10.1109/ICISA.2013.6579461

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