2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion) | 979-8-3503-2263-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE-COMPANION58688.2023.00068

Software Supply Chain Risk Assessment Framework

Nusrat Zahan
North Carolina State University, Raleigh, USA
Email: nzahan@ncsu.edu

***Abstract*—Sonatype has recorded an average 700% jump in software supply chain attacks [1], measured by the number of newly-published malicious packages in open-source repositories. The 2022 Synopsys report [2] assessed the reliance of the software industry on open-source software (OSS), and estimated that 97% of applications use OSS and 78% of the code comes from OSS. Practitioners did not anticipate how the software supply chain would become a deliberate attack vector and how the risk of the software supply chain would keep growing. Practitioners are more aware of the supply chain risks and want to know how to detect the implementation of package security practices and the security risk so they can make informed decisions to select dependencies for their projects. *The goal of this research is to aid practitioners in producing more secure software products that are resistant to supply chain attacks through the identification and evaluation of actionable security metrics to detect risky components in the dependency graph.* To achieve this goal, the thesis presents research on software security metrics evaluation in different ecosystems by leveraging software security frameworks, malicious attack vectors, and the OpenSSF Scorecard project to detect the implementation of secure practices and their significance to security outcomes.**

Keywords: Software supply chain security, security metrics, weak link signal, risk assessment framework

I. Introduction

Recent high-profile supply chain attacks, such as Solarwinds [3] and Codecov [4], made headlines and directed attention towards the importance of software supply chain security. Sonatype has recorded an average 700% jump in supply chain attacks in 2022 [1]. Software developers largely did not anticipate how the software supply chain would become a deliberate attack vector. Many stakeholders have started to recognize this urgent concern; most prominently, Executive Order 14028 [5], issued May 12, 2021, calls for enhancing software supply chain security with a focus on identifying practices that enhance the security of the software supply chain. The most recently released memo from the White House Office of Management and Budget (OMB) [6] requires government agencies to provide self-attestation of security practices. Organizations that supply critical software to the US government must comply with secure software supply chain practices.

Our increasing reliance on OSS creates a large attack surface. The software industry has moved from passive adversaries finding and exploiting vulnerabilities contributed by well-intentioned developers to a new generation of software supply chain attacks, where attackers inject vulnerabilities directly into dependencies to find their way into builds and deployments. Practitioners are more aware of supply chain risks and are motivated to understand how to improve package security through the adoption of security practices to resist supply chain attacks. Therefore, practitioners will be benefited from knowing techniques to assess the risk associated with a package and its components.

To reduce supply chain attack vectors, we need research to develop actionable security metrics to detect risky components that contain signals indicating: ii) that the development team did not use software security practices, ii) weak links in the dependency graph that exposes a package to a higher risk of a supply chain attack, and an attacker can exploit the signal to execute a supply chain attack. Empirical data demonstrating the value of security metrics are required to substantiate the security practice adoption. To that end, *the goal of this research is to aid practitioners in producing more secure software products that are resistant to supply chain attacks through the development and evaluation of security metrics to detect risky components in the dependency graph.*

Research Questions: To achieve our research goal, this thesis addresses multiple research questions:

- **RQ1**: Which security practices should be adopted to improve the package's security?
  - RQ1.1: Which security practices can be automatically-detectable in source code repositories and adopted by practitioners?
  - RQ1.2: Which security practices assist in mitigating supply chain attack vectors?
- **RQ2**: How can security metrics be served to detect weak links in software ecosystems?
  - RQ2.1: What type of weak link signals exists in the software supply chain?
  - RQ2.2: What weak link signals are observed in the maintainers' activities?
- **RQ3**: How can automatically-detectable metrics be used as evidence of security practice use?
  - RQ3.1: Which security practices are most important to understand the relationship between security practices and vulnerability counts in regression models?
  - RQ3.2: Do packages with higher security scores have fewer vulnerabilities?
  - RQ3.3: How do relationships between mean time to remediate (MTTR) and security practices differ for malicious and vulnerable packages?

Contribution: This thesis will develop a Software Supply Chain Risk Assessment Framework (SSCRAF) to aid in evaluating the implementation of software security practices and

979-8-3503-2263-7/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE-Companion58688.2023.00068

detection of weak link signals to resist supply chain attacks. To that, we will conduct empirical research to identify a list of scientifically-evaluated security metrics that drive the adoption of effective security practices. We aid practitioners in determining if a candidate dependency is risky by identifying signals indicating security risks and implementing security practices in the package repository. We also contribute through the automation of the isolation of risky dependencies. This research proposal aims to achieve the following contributions:

- Empirical evaluation of open-source software to identify actionable security practice metrics.
- Empirical evaluation of malicious packages to identify weak link signals
- A Software Supply Chain Risk Assessment Framework (SSCRAF) to assess and evaluate the risk of software products.

II. Research Problem and Related Work

This section discussed prior related studies proposed to improve package security and highlighted the research problem we are addressing in this thesis.

A. Security Practice Guidelines:

While organizations seek to address escalating security risks and comply with regulations, a myriad of activities are available to improve software security. Different organizations issued guidance [7]–[13] on software development practices that enhance the security of the software supply chain. The Open Source Security Foundation (OpenSSF) [14], a cross-industry organization hosted at the Linux Foundation, provides the vehicle for collaboration on tools, services, training, infrastructure, and resources for securing open-source projects.

Problem: While different frameworks and OpenSSF projects provide guidelines and comprehensive lists of security practices, challenges arise in validating whether these practices improve security. Organizations may not have the resources to adopt a full suite of these practices. They would like to understand the key drivers of success, especially which of many possible software security practices to undertake first. Practitioners will comply with these guidelines if data-driven research studies on security practice measurement and feedback loops validate the improvement in security outcomes.

B. Security Risk Detection:

Many prior works have leveraged past supply chain records to show how catastrophic a supply chain attack can be. Zimmermann et al. [15] provided evidence that popular packages and highly active developers in npm may suffer from single points of failure due to large direct and transitive dependencies. Ohm et al.'s study [16] investigated different supply chain attacks from 174 malicious packages and contributed an enriched dataset for future research on malicious package detection. Duan et al. [17] identified 339 malicious packages from their proposed unsupervised learning framework. Gonzalez et al. [18] used repository and commit metadata to detect malicious packages automatically. Vu et al. [19] and Scalco et al. [20] analyze the discrepancy between source code and the deployed package in PyPI and JavaScript packages as a way to detect malicious injections. Sejfia et al. [21] propose a machine learning-based approach for the automated detection of malicious npm packages trained on a labeled dataset.

Problem: These studies are effective in preventing malicious code distribution. Recent attacks show evidence that out-of-the-box exploit strategies will appear again and again [3], [4]. Any ad-hoc solution is not enough to prevent an attack that we have not yet witnessed. A better approach is needed to embrace a proactive strategy that predicts package susceptibility as a potential threat and helps package managers and maintainers detect those security risks before the malicious code is distributed in the supply chain.

III. Prior Research

In this section, we discuss our research findings as of 2022.

A. Security Practice Metrics:

The OpenSSF Scorecard project [22] auto-generates a "security score" for open source projects to aid in risk and trust decisions on the security posture of open source projects. We used the tool to understand the security practices and gaps in npm and PyPI ecosystems and to evaluate the applicability of the Scorecard tool for different ecosystems [23]. We also study whether the tool's guidance is consistent with developer actions. An overview of our findings is shown in Table I. We found that 13 Scorecard security metrics were compatible with the SSDF framework [7]. Next, we identified 9 Scorecard security metrics that can be used to measure npm and PyPI package security and are consistent with developer actions. Then, "Dangerous Workflow" can aid in preventing malicious attacks. Five metrics, however, necessitate industry agreement. Both ecosystems showed gaps in implementing Token-Permission, License, Code-Review, Maintained, Branch Protection, and Security Policy practices.

Additionally, we also look into the relationship between "Security Practice data" and "Security Outcomes as measured by the vulnerability count in each package" to determine how security practices should be prioritized, particularly about which security practices have the biggest impact on security outcomes [24]. Our models found that maintained Code Review, Branch Protection, and Security Policy are the most important practices that practitioners can adopt to improve package security outcomes by minimizing vulnerabilities. However, both ecosystems had gaps in implementing these practices. We also looked into how multiple security practices and their aggregate score affect the security outcome of software. We found that packages with increased good security practices also had increased reported vulnerability counts, demonstrating the need for confounding variable control and more vulnerability data to ascertain whether the relationship is explained by other factors.

B. Security Risk Metrics

We performed a data-driven study to detect weak link signals in npm ecosystem and identified the following six signals

TABLE I
SUMMARY OF SCORECARD SECURITY METRICS EVALUATION

Scorecard Metrics	Evaluation of this study	Scorecard Metrics	Evaluation of this study
Dangerous-Workflow	Effective Metric, Need-for-Improvement	Branch-Protection	Effective Metric, Lack of adoption
Pinned-Dependencies	Need-for-Improvement	Dependency-Update-Tool	Require Industry consensus
Vulnerabilities	Effective Metric	Security-Policy	Effective Metric, Lack of adoption
Binary-Artifacts	Effective Metric, Need-for-Improvement	Packaging	Require Industry consensus
Token-Permissions	Effective Metric, Need-for-Improvement, Lack of adoption	CII-Best-Practices	Require Industry consensus
License	Effective Metric, Need-for-Improvement, Lack of adoption	Signed-Releases	Require Industry consensus
Code-Review	Effective Metric, Lack of adoption	Fuzzing	Require Industry consensus
Maintained	Effective Metric, Lack of adoption		

that are indicative of security weakness in npm software supply chain: 1) expired email domain; 2) install scripts; 3) unmaintained packages; 4) too many maintainers; 5) too many contributors, and 6) overloaded maintainers. Additionally, we obtained feedback on our weak link signals through a survey responded to by 470 npm package developers [25]. We found 2,818 maintainer email addresses associated with expired domains, allowing an attacker to hijack 8,494 packages by taking over the npm accounts, and 94% of malicious packages contain install scripts, indicating the risk of installing scripts in npm. Then, 58% of npm packages are unmaintained. Among all the maintainers of those packages, 44% do not own any active packages in the entire npm platform. We also observed that maintainers are threatened to become overwhelmed since 4,743 maintainers own 52% of the 1.6M packages in npm packages.

This work has been recognized and adopted in the industry, and security specialists are now promoting these signals as part of proactive software supply chain security [26]. Motivated by our study, **npm, GitHub and PyPI** are now enforcing **two-factor authentication** on developer accounts [26]. Industry tools are integrating six signals as automated supply chain security checks.

IV. PROPOSED APPROACH

In this thesis, we will develop a **Software Supply Chain Risk Assessment Framework (SSCRAF)** that can be used by software practitioners to make informed decisions and produce secure software by assessing software security risks. The framework will aid in evaluating the implementation of software security practices and the detection of weak link signals to resist supply chain attacks. To that, we will conduct empirical research to identify a list of scientifically-evaluated security metrics that drive the adoption of effective security practices. The framework will be developed through the following triangulation of qualitative and quantitative research to identify scientifically evaluated security metrics.

A. Security Practice Metrics Identification

We found from our study III-A that not all Scorecard proposed security practices were adopted by npm and PyPI ecosystems. Scorecards need to include ecosystem-specific security practices for better adoption. To enhance the Scorecard security practice list, we will study and explore different security frameworks [7]–[13], [27] and supply chain attack vectors [28] to identify recommended metrics that help mitigating supply chain attack vectors. We will contribute to Scorecard to include new security practices or improve the existing security practices.

B. Security Risk Metrics Identification

The npm repository removes malicious packages from the registry and replaces the original metadata with dummy metadata. Thus researchers cannot extract meaningful information once a package is removed from the repositories. Since only a few malicious packages are available to study in deprecated mirrors, internet archives, and public research repositories, the effectiveness of prior studies [16], [17] are limited in detecting weak links in the dependency graph. We will conduct a longitudinal study to understand how malicious packages are included in a dependency graph and what weak links were exploited by attackers to find their way into builds and deployments. To that, we will identify patterns in malicious packages and develop metrics that signal the weak links in npm software supply chain.

We have collected npm package metadata from the package registry since January 2022. A longitudinal study on npm registry will help us collect data on malicious packages before npm removes such packages from the registry. We will study npm malicious package data, including the identification of maintainers, contributors with patterns of malicious actions, script and attack types, binary files, and attack timelines to identify actionable metrics. For instance, we will analyze whether maintainer metrics in the dependency graph, such as unstable ownership, unmaintained packages, identity, contribution, etc., relating to the attacker compromising the supply chain. We will also examine the attack strategy and target dependents. For instance, the % of new malicious packages without any dependents versus the % of benign packages that evolve into malicious ones over time. Based on these findings, we will develop techniques to automatically identify

and annotate suspicious packages that increase the attack surface risk of the downstream project.

C. Metrics Evaluation

Once we have a list of security metrics and security risks from the above two steps, we will assess whether security metrics improved security package outcomes. With this technique, we hypothesize that-

- **H1**: Packages with higher security scores promote better security practices and outcomes than those with lower security scores. Here, better security outcomes indicate fewer vulnerabilities and the duration of patching is faster.
- **H2**: As package grows in size (code, developer count, dependency, dependents), they adopt safer security practices, but at the same time they grow more complex. Better, safer security practices might be corresponding to higher complexity, affecting their security outcomes metrics negatively.
- **H3**: Weak link signals exploited by attackers can be witnessed in benign packages.

In this study, we will study 4 security outcomes metrics taken from the literature-

- **Vulnerability Count**- the number of vulnerabilities detected and reported in security advisory databases.
- **Mean time to remediate (MTTR)**- is the average time required for a component to adopt security-relevant updates to dependencies.
- **Median time to update (MTTU)**- is the median time required for the component to update to the newer dependency.
- **Cumulative days open normalized (CDO/N)**- is the number of days open for any finding/incidents. Normalized by the number of findings added in the last year.

Using APIs tailored to the ecosystem registry, we will calculate the time-to-remediate (TTR), and time-to-update (TTU) for each package. We will collect vulnerability count data from different security advisory data. To test our hypothesis, we will evaluate security outcomes metrics, and security metrics and collect popularity metrics to understand how package complexity affects security outcomes. We will build machine learning models to understand whether the implementation of security practices improved security outcomes. Through this process, we will gain a more holistic understanding to assess the degree to which automatically-detected metrics can be considered to be verified and help developers to remediate risk.

V. Evaluation of SSCRAF

This thesis is divided into two steps research plan. First, we will identify the automatically-detectable security metrics that help practitioners to assess the security risk in the supply chain. To the extent practicable, we will validate each metric's significance on security outcomes by using vulnerability count, malicious package patterns, MTTU, MTTR, and CDO/N metrics. We will also conduct an ecosystems analysis to evaluate whether the framework's guidance is consistent with developer actions. As a second phase of this research, we will assess the validity of the proposed model and framework. We will interview and survey to understand developers' perspectives on our proposed framework. Developers' feedback will help us to refine our proposed framework.

VI. Timeline

The author is a 4th year PhD student at North Carolina State University (NCSU). Two of our initial research [24], [25] have been accepted in ICSE SEIP, and another initial study is under revision [23]. We intend to complete the scientific evaluation of security metrics by 2023, and we will evaluate practitioners' perspectives on our proposed framework by 2024 Summer. We will complete this dissertation by the fall of 2024.

VII. Acknowledgements

This thesis is advised by Dr. Laurie Williams, Distinguished University Professor at NCSU. Different part of this work was supported and funded by Microsoft Research internship, Cisco, and NCSU Secure Computing Institute. Since 2023 the work has been supported by National Science Foundation Grant No. 2207008. Any opinions expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We thank the npm, GitHub, OpenSSF Scorecard Team, and Google GOSST for encouraging us to conduct this research. We also thank the NCSU Realsearch group for valuable feedback.

References

[1] Sonatype, "700% average increase in open source supply chain attacks," *https://www.sonatype.com/press-releases/sonatype-finds-700-average-increase-in-open-source-supply-chain-attacks*, 2022.

[2] Synopsys, "2022 open source security and risk analysis," 2022. [Online]. Available: https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html

[3] J. M. Germain, "Lessons learned from the solarwinds supply chain hack," 2021. [Online]. Available: https://www.technewsworld.com/story/lessons-learned-from-the-solarwinds-supply-chain-hack-87029.html

[4] M. JACKSON, "Codecov supply chain breach - explained step by step," *https://blog.gitguardian.com/codecov-supply-chain-breach/*, 2021.

[5] D. The White House, Washington, "Executive order on improving the nation's cybersecurity," *https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity*, 2021.

[6] S. D. Young, "M-22-18 memorandum for the heads of executive departments and agencies," *https://www.whitehouse.gov/wp-content/uploads/2022/09/M-22-18.pdf*, 2022.

[7] M. Souppaya, K. Scarfone, and D. Dodson, "Secure software development framework (ssdf) version 1.1," *NIST Special Publication*, vol. 800, p. 218, 2022.

[8] NIST, "Software supply chain security guidance under executive order (eo) 14028 section 4e," *NIST Special Publication*, 2022.

[9] C. N. C. Foundation, "Software supply chain best practices," *https://github.com/cncf/tag-security/tree/main/supply-chain-security/supply-chain-security-paper*, 2021.

[10] BSIMM, "Bsimm12 2021 foundations report," *https://www.bsimm.com/content/dam/bsimm/reports/bsimm12-foundations.pdf*, 2021.

[11] OWASP, "Software component verification standard (scvs)," *https://owasp-scvs.gitbook.io/scvs/*, 2020.

[12] Microsoft, "Oss ssc framework," *https://github.com/microsoft/oss-ssc-framework*, 2022.

[13] SLSA, "Supply chain levels for software artifacts," *https://slsa.dev/*, 2022.
[14] L. Foundation), "Open source security foundation," 2021. [Online]. Available: https://openssf.org/
[15] M. Zimmermann *et al.*, "Small world with high risks: A study of security threats in the npm ecosystem," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 995–1010.
[16] Ohm et al., "Backstabber's knife collection: A review of open source software supply chain attacks," *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2020.
[17] Duan et al., "Towards measuring supply chain attacks on package managers for interpreted languages," *arXiv preprint arXiv:2002.01139*, 2020.
[18] D. Gonzalez *et al.*, "Anomalicious: Automated detection of anomalous and potentially malicious commits on github," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2021, pp. 258–267.
[19] D.-L. Vu, F. Massacci, I. Pashchenko, H. Plate, and A. Sabetta, "Lastpymile: identifying the discrepancy between sources and packages," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 780–792.
[20] S. Scalco, R. Paramitha, D.-L. Vu, and F. Massacci, "On the feasibility of detecting injections in malicious npm packages," in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, 2022, pp. 1–8.
[21] Sejfia et al., "Practical automated detection of malicious npm packages," *arXiv preprint arXiv:2202.13953*, 2022.
[22] OpenSSF, "Security scorecards for open source projects," 2021. [Online]. Available: https://github.com/ossf/scorecard
[23] N. Zahan *et al.*, "Openssf scorecard: On the path toward ecosystem-wide automated security metrics," *arXiv preprint arXiv:2208.03412*, 2023.
[24] N. Zahan, S. Shohan, D. Harris, and L. Williams, "Preprint: Do openssf scorecard practices contribute to fewer vulnerabilities?" *arXiv preprint arXiv:2210.14884*, 2022.
[25] N. Zahan *et al.*, "What are weak links in the npm supply chain?" in *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, 2022, pp. 331–340.
[26] N. Zahan, "Impact of the research study- what are weak links in the npm supply chain?" *https://www.nzahan.net/research-project*, 2022.
[27] E. S. Framework, "Securing the software supply chain: Recommended practices guide for developers," *https://www.cisa.gov/uscert/sites/default/files/publications/*, 2022.
[28] P. Ladisa *et al.*, "Taxonomy of attacks on open-source software supply chains," *arXiv preprint arXiv:2204.04008*, 2022.