Tracing Vulnerabilities in Maven: A Study of CVE lifecycles and Dependency Networks

Corey Yang-Smith
University of Calgary
Calgary, Canada
corey.yangsmith@ucalgary.ca

Ahmad Abdellatif
University of Calgary
Calgary, Canada
ahmad.abdellatif@ucalgary.ca

***Abstract*—Software ecosystems rely on centralized package registries, such as Maven, to enable code reuse and collaboration. However, the interconnected nature of these ecosystems amplifies the risks posed by security vulnerabilities in direct and transitive dependencies. While numerous studies have examined vulnerabilities in Maven and other ecosystems, there remains a gap in understanding the behavior of vulnerabilities across parent and dependent packages, and the response times of maintainers in addressing vulnerabilities. This study analyzes the lifecycle of 3,362 CVEs in Maven to uncover patterns in vulnerability mitigation and identify factors influencing at-risk packages. We conducted a comprehensive study integrating temporal analyses of CVE lifecycles, correlation analyses of GitHub repository metrics, and assessments of library maintainers' response times to patch vulnerabilities, utilizing a package dependency graph for Maven. A key finding reveals a trend in "Publish-Before-Patch" scenarios: maintainers prioritize patching severe vulnerabilities more quickly after public disclosure, reducing response time by 48.3% from low (151 days) to critical severity (78 days). Additionally, project characteristics, such as contributor absence factor and issue activity, strongly correlate with the presence of CVEs. Leveraging tools such as the Goblin Ecosystem, OSV.dev, and OpenDigger, our findings provide insights into the practices and challenges of managing security risks in Maven.**

***Index Terms*—Software Ecosystems, Maven, Software Vulnerabilities**

I. Introduction

In the field of Software Engineering, package registries like Maven (Java), npm (JavaScript), and PyPI (Python), serve as centralized hubs for sharing code. These packages often establish intricate dependency networks, where vulnerabilities propagate through the ecosystem via library imports, affecting both direct and transitive dependencies [1], [2]. Vulnerabilities compromise projects and introduce attack vectors, risking faults such as Denial of Service (DoS) or privilege escalation [3]; however, reporting systems such as the CVE Program [4] and OSV.dev [5] provide frameworks for tracking and managing security risks, aiding developers in vulnerability management. According to the 2024 Stack Overflow Developer Survey [6], 30% of respondents reported using Java in developing software applications. As a foundational ecosystem for Java development, Maven hosts more than 47 million packages [7], underscoring the crucial need for a deeper understanding of the systemic risks introduced through vulnerabilities.

Prior research has focused on ecosystems like PyPI [8] or npm [9], [10], with some studies examining response times in Maven [11], [12]. However, there remains a gap in understanding the correlation between project characteristics and vulnerabilities in Maven, and the behavior of response times for package maintainers. By understanding the relationship between project characteristics and vulnerabilities, maintainers can prioritize secure and reliable dependencies, ultimately enhancing the overall quality and resilience of their projects.

Therefore, in this study, we aim to identify key project characteristics (e.g., number of stars, issue activity) associated with vulnerabilities, analyze their severity and impact, and examine response times to better understand systemic risks within Maven. Leveraging a dependency graph of artifacts and releases hosted on Maven, sourced from the Goblin Ecosystem [13], we address these objectives through the following research questions:

RQ1: What is the lifecycle of vulnerabilities in Maven artifacts? Our analysis reveals the majority of vulnerabilities (81.6%) are patched before public disclosure, while only a small fraction (5.6%) remain unresolved. This signals a robust effort by the community to address vulnerabilities promptly. In cases where CVE disclosure precedes patch availability (12.8%), we observe a trend of faster response times as vulnerability severity increases.

RQ2: How do project characteristics correlate with vulnerability outcomes? Higher issue activity, contributor absence factor, and participant counts (active contributors) correlate with the presence of vulnerable libraries. This suggests that larger, more active teams may be associated with elevated vulnerability risk.

RQ3: How long does it take for dependent packages to adopt a new fix? We find that dependent packages most commonly adopt fixes via Available Patch Adoption cases (62.9%), with a median of 151 days. In contrast, higher-risk Reactive Adoption (4.5%) cases have a higher median at 249 days, often due to patch unavailability. These findings underscore the need for consistent and efficient patch adoption practices among maintainers.

II. Methodology

This study examines the lifecycle and contributing factors of vulnerabilities in Maven and assesses how quickly direct dependencies adopt available fixes. In this section, we outline the data and methods used to answer our research questions.

A. Terminology

- **CVE Introduction Date**: Date the CVE was first introduced in a release.
- **CVE Publication Date**: Public disclosure date (per OSV.dev [5]).
- **CVE Patch Date**: The release date of the patched version addressing the CVE.
- **Latest Affected Version Date**: Last vulnerable release date before patching.
- **First Patch Adoption Date**: First release date adopting the patched version.

B. CVE Lifecycle Analysis

The Goblin Ecosystem [13] is an open-source platform designed for ecosystem-level dependency analysis. It combines a dependency graph metamodel (Dataset), a data miner to extract dependency-related data (Miner), and a service for on-demand metrics weaving into dependency graphs (Weaver).

The Goblin Dataset (dated 2024-08-30) is a Neo4j-based representation of the Maven ecosystem, comprised of artifacts, releases, and package interdependencies. It includes metadata such as Maven identifiers, release timestamps, versioning details, and explicitly defined dependency relationships. Using Goblin Weaver (v2.1.0) and Neo4j (Desktop Version 4.4.4), we enriched the Goblin Dataset with vulnerability information from OSV.dev [5], a platform offering standardized data aligned with open-source versioning schemes and an API to retrieve publicly available vulnerability data across software ecosystems, as utilized in prior research [14], [15]. For each CVE, we query OSV.dev to retrieve severity, vulnerable version ranges, timestamps, and alternative identifiers. We enriched the Goblin dataset with **3,362 unique CVEs** affecting 190,945 releases across 1,470 artifacts. Table I summarizes the vulnerability severity distribution. Similar to prior work [8], [12], we define the lifecycle of a CVE as having many stages, beginning on the release date of the first vulnerable version. The lifecycle is calculated as the time delta, in days, between the CVE Introduction Date and the CVE Patch Date. To classify vulnerabilities into patch categories, we compare the chronological order between two key dates: (1) the CVE Publication Date and (2) the CVE Patch Date. A vulnerability is classified as:

- **No-Patch**: No patched version exists in the dataset.
- **Publish-Before-Patch**: The patch is released after the vulnerability is published.
- **Patch-Before-Publish**: The patch is released before the vulnerability publication.

C. Package Characteristics Correlation

To analyze correlations between historical project characteristics and vulnerabilities, we identify GitHub as a source of metrics data that can help characterize packages we examine. Using Maven's API [16], we extracted pom.xml files from the Goblin Dataset (Section II-B), identifying 204 GitHub repositories via SCM link regex. Repository metrics (e.g., number of stars) were derived using OpenDigger [17], which captures GHArchive [18] historical event data. Metrics were collected up to the CVE Patch Date (or the current date for unresolved cases) to capture the state of the system during the vulnerability period.

TABLE I
DETAILED BREAKDOWN OF SEVERITY FOR ALL CVES, UNIQUE

Severity	Count
Low	105 (3.12%)
Medium	1,380 (41.05%)
High	1,279 (38.04%)
Critical	598 (17.79%)
Total	**3,362 (100.00%)**

Of 204 repositories, 29 (14.2%) were excluded for incomplete data (e.g., inaccessible repositories). We reduced OpenDigger's 27 candidate repository metrics, originally sourced from CHAOSS [19] and X-lab [20], to 17 by removing those with excessive missing data (e.g., issue response time). The final dataset comprises 175 repositories relating to 456 unique CVEs, and 17 metrics spanning popularity (e.g. number of stars, forks), contributor counts, and issue activity. A detailed list of all metrics, including those excluded, and descriptions are available in our replication package.

To evaluate associations between package characteristics and the presence of CVEs, we use the **rank-biserial correlation**—a non-parametric measure of effect size for binary-continuous relationships—paired with Mann-Whitney U tests [21] to determine statistical significance ($p < 0.05$). While these tests identify meaningful associations, the severe class imbalance between historical repositories with CVEs (456) and those without (10,675) could compromise reliability.

D. Dependency Patch Adoption

To measure the response time in dependency patch adoption by package maintainers, we identify affected releases and directly dependent packages, focusing on dependencies that rely on a vulnerable parent release. For each dependent artifact, we parse its release history and query Neo4j to extract two key dates: (1) the **latest affected date**, and (2) the **first patch adoption date**. This analysis focuses on "Publish-Before-Patch" and "Patch-Before-Publish" cases from Section II-B, but expands to include the patch adoption behavior of dependent libraries. "No-Patch" cases are excluded from the analysis. Based on the order of events in CVE publication and patching, and maintainer response, we categorize dependent patch adoption behavior into three scenarios:

- **Reactive Adoption:** CVE Publication → Patch Release → Dependent Adoption
- **Available Patch Adoption:** Patch Release → CVE Publication → Dependent Adoption
- **Proactive Adoption:** Patch Release → Dependent Adoption → CVE Publication

We calculate the response time for dependent packages as the time delta, in days, between the earliest patch date and the latest affected date. This allows us to evaluate the efficiency

of dependency patch adoption practices and identify patterns that influence the speed of mitigation in Maven.

III. Results

A. RQ1: What is the lifecycle of vulnerabilities in Maven artifacts?

The distribution of vulnerabilities across the No-Patch, Publish-Before-Patch, and Patch-Before-Patch cases reveals distinct behaviors among Maven artifacts. We find that 81.6% of CVEs in Maven are Patch-Before-Publish. Moreover, the results show that 12.8% and 5.6% of vulnerabilities are No-Patch and Publish-Before-Patch, respectively. Our results align with prior work [12], emphasizing the proactive role vendors play in managing vulnerabilities.

Table II presents a detailed summary of the average time taken to mitigate vulnerabilities. From the table, we observe a trend in "Publish-Before-Patch" cases: maintainers prioritize patching critical vulnerabilities more quickly after public disclosure (78 days), reflecting the heightened urgency associated with severe security issues compared to lower severity vulnerabilities (151 days). Conversely, for "Patch-Before-Publish" cases, moderate-severity issues are addressed the furthest in advance (1128 days), although no consistent trend emerges across severity levels. One potential reason for this finding is that moderate-severity issues may be detected and addressed during routine maintenance and refactoring [22], whereas critical-severity issues undergo the coordinated disclosure process.

B. RQ2: How do GitHub repository metrics correlate with project outcomes?

Table III presents the correlation effect size for 17 metrics sourced from OpenDigger, illustrating positive trends across all metrics ($p < 0.05$). We observe that issue activity metrics (e.g., issue comments), repository contributor count metrics (e.g., contributor absence factor) [23] and participant count strongly correlate with the presence of CVEs, suggesting that larger, more active teams may face greater vulnerability challenges. This may be in part due to communication overhead associated with team size, which introduces complexity and reduced productivity [24].

Moderate correlations were observed between vulnerability presence and characteristics such as code churn metrics (e.g., lines of code added) and popularity metrics (e.g., number of stars). This suggests that repositories with high activity and widespread usage may face challenges in vulnerability oversight. In contrast, inactive contributors showed negligible correlations, indicating limited relevance to vulnerability likelihood. However, these relationships are purely observational and do not imply causation. Unmeasured factors—such as code review practices, dependency management, or security tooling—may influence these outcomes. Our findings both align with and diverge from prior research. While some studies have reported a correlation between code churn and vulnerability presence [25], others suggest that popularity metrics do not exhibit the same relationship [26], [27].

TABLE II
Mean Time to Mitigate by Patch Type and Severity, in Days

Patch Type	Low	Moderate	High	Critical
Publish-Before-Patch	151	123	84	78
Patch-Before-Publish	−867	−1128	−897	−790

TABLE III
Rank Biserial Correlation Results

Name	Rank Biserial Correlation
Issue Comments	0.679
Issues Closed	0.660
Contributor Absence Factor	0.660
Issues New	0.657
Participants	0.651
Activity	0.645
Technical Fork	0.629
Attention	0.539
New Contributors	0.512
Change Requests	0.464
Number of Stars	0.474
Code Change Lines Sum	0.452
Code Change Lines Add	0.448
Code Change Lines Remove	0.390
Change Requests Reviews	0.336
Change Requests Accepted	0.297
Inactive Contributors	0.149

C. RQ3: How long does it take for directly dependent packages to adopt a newly available fix?

Finding the response times for direct dependencies to adopt patched versions is the first step to gaining insight about the response of the overall Maven ecosystem in addressing vulnerabilities. We observe that Available Patch Adoption cases dominate with 46,536 cases (62.9%), but despite the immediate availability of fixes, maintainers often delay implementation, resulting in a median response time of 151 days. The highest-risk group, Reactive Adoption, comprises the fewest instances with 3,313 cases (4.5%) yet exhibits a higher median time to resolution (249 days). The delay likely stems from the unavailability of patches during the critical period between vulnerability disclosure and adoption. While Proactive Adoption represents the best practice, this group is smaller with 24,144 cases (32.6%) and exhibits significant variability in adoption timing, highlighting the challenges of achieving consistent and timely vulnerability mitigation.

Figure 1 presents the distribution of response time behavior from directly dependent packages, and provides insights into the propagation of patches across Maven. Notably, the prevalence of Available Patch Adoption suggests that most vulnerabilities have remediations accessible to dependent packages, yet the slow median response time indicates systemic inefficiencies in prioritizing fixes. Reactive Adoption's extended resolution period further emphasizes the risks of relying on post-disclosure patching, particularly when fixes are delayed or unavailable during critical periods. Meanwhile, the variability observed in Proactive Adoption—where some packages adopt patches rapidly while others lag—reveals unresolved chal-

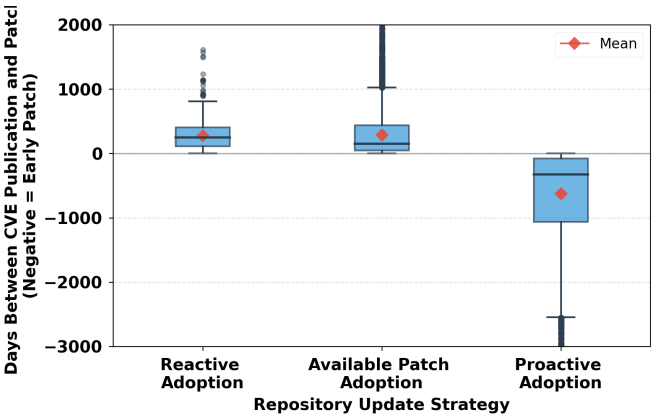

Fig. 1. Boxplot of response time per dependent behavior.

lenges in achieving consistent, timely vulnerability mitigation, even among maintainers employing forward-looking practices.

IV. Threats to Validity

Threats to construct validity pertain to the relationship between theoretical concepts and their empirical measurement. One key concern arises from incomplete CVE metadata, particularly the lack of explicit lower bounds for affected software versions [28]. Without knowing when a vulnerability was first introduced, researchers cannot reliably assess whether older package versions remain vulnerable. This limitation may misrepresent patching behaviors in legacy systems, introducing bias and compromising the accuracy of our findings. Another challenge is the absence of CVE discovery dates. Since our calculated mean time to mitigation is based solely on observed patching behavior, it may not reflect the actual duration maintainers were aware of the vulnerability. Instead, it serves as a proxy metric, potentially underestimating true vendor response time and providing a lower bound on mitigation delays.

Threats to internal validity relate to potential confounding factors that may influence our results. One such factor is project size. While our analysis finds a correlation between higher issue activity and the presence of vulnerabilities, larger projects may naturally generate more issues, independent of security concerns. This raises the possibility that issue activity is a byproduct of project scale rather than a direct predictor of vulnerabilities. Future research could mitigate this confounding effect by normalizing issue activity relative to repository age, number of contributors, or total codebase size.

Threats to external validity concern the generalizability of our findings beyond the studied ecosystems. Our analysis focuses exclusively on the Maven ecosystem and GitHub-hosted repositories. As a result, our conclusions may not fully extend to other package registries or software configuration management (SCM) platforms. We plan (and encourage others) to replicate our methodology across diverse ecosystems and non-GitHub platforms.

V. Related Work

Considering the importance of mitigating and managing vulnerabilities in software ecosystems, a number of studies have been conducted in the area of vulnerability management, with efforts spanning alerting systems, Software Composition Analysis (SCA) tools, and empirical studies on vulnerability lifecycles. Cadariu et al. [29] introduce a Vulnerability Alert Service for proactive risk identification, while Dietrich et al. [30] focus on detecting code clones, addressing gaps in SCA tools.

Prior research has examined vulnerability lifecycles in PyPI [8], [31] and npm [9], [32], [33], but fewer studies focus on Maven. For example, Heng et al. [12] manually analyze 312 CVEs in Maven, whereas our automated approach scales to 3,362 CVEs, providing broader insights. Wu et al. [11] explore transitive dependency risk, while we examine how direct dependents adopt security patches.

To mitigate dependency risks, Zhang et al. [28] propose Ranger, an automated solution for restoring compatibility in vulnerable Maven packages. Mir et al. [34] use reachability analysis to assess transitive vulnerability impacts. Our work differs by empirically measuring maintainer response times, offering new insights into how security fixes propagate in Maven.

VI. Ethical Considerations

This study raises ethical concerns about the potential misuse of the findings, particularly the correlation between larger contributor teams and increased vulnerability risks. Aligning with [35], we acknowledge that aggregated data could enable repository profiling that malicious actors might exploit. To mitigate risks, we omit repository identifiers and present results in aggregate form. By focusing on descriptive analysis rather than prescriptive measures, we limit actionable guidance that could be misapplied before vulnerabilities are resolved. Our findings aim to inform vulnerability management practices that balance ecosystem resilience with ethical responsibility.

VII. Conclusion

This study highlights key aspects of vulnerability management in the Maven ecosystem through the analysis of 3,362 CVEs and their propagation within dependency networks. While 81.6% of vulnerabilities are patched prior to public disclosure, demonstrating proactive efforts, vulnerabilities disclosed before a patch (12.8%) exhibit slow response times, particularly for lower-severity issues. Furthermore, we find that project characteristics such as contributor absence factor and issue activity correlate with vulnerability presence, suggesting that larger, more active teams face increased coordination challenges. Dependency response times vary widely, with the most common approach showing a median response time of nearly five months, while the optimal strategy is both under-utilized and inconsistent. These findings emphasize the need for consistent patch adoption practices, better collaboration in larger projects, and improved vulnerability management tools.

VIII. Data Availability

Our source code and dataset is available at: https://github.com/coreyyangsmith/msr2025

References

[1] R. Kikas, G. Gousios, M. Dumas, and D. Pfahl, "Structure and evolution of package dependency networks," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 102–112.

[2] J. Düsing and B. Hermann, "Analyzing the direct and transitive impact of vulnerabilities onto different artifact repositories," *Digital Threats: Research and Practice*, vol. 3, no. 4, pp. 1–25, 2022.

[3] M. C. Sánchez, J. M. C. de Gea, J. L. Fernández-Alemán, J. Garceran, and A. Toval, "Software vulnerabilities overview: A descriptive study," *Tsinghua Science and Technology*, vol. 25, no. 2, pp. 270–280, 2020.

[4] M. Corporation, "Common vulnerabilities and exposures (cve): Overview," 2024, accessed: 2024-11-30. [Online]. Available: https://www.cve.org/about/overview

[5] G. O. S. S. Team, "Osv: Open source vulnerabilities," 2024, accessed: 2024-11-30. [Online]. Available: https://osv.dev

[6] Stack Overflow, "Stack overflow developer survey 2024," 2024, accessed: 2024-11-30. [Online]. Available: https://survey.stackoverflow.co/2024/

[7] "Maven repository: Central repository search," 2024, accessed: 2024-12-08. [Online]. Available: https://mvnrepository.com/repos

[8] M. Alfadel, D. E. Costa, and E. Shihab, "Empirical analysis of security vulnerabilities in python packages," *Empirical Software Engineering*, vol. 28, no. 3, p. 59, 2023.

[9] M. Alfadel, D. E. Costa, E. Shihab, and B. Adams, "On the discoverability of npm vulnerabilities in node.js projects," *ACM Transactions on Software Engineering and Methodology*, vol. 32, pp. 1 – 27, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:253708713

[10] D. Setó-Rey, J. I. Santos-Martín, and C. López-Nozal, "Vulnerability of package dependency networks," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 6, pp. 3396–3408, 2023.

[11] Y. Wu, Z. Yu, M. Wen, Q. Li, D. Zou, and H. Jin, "Understanding the threats of upstream vulnerabilities to downstream projects in the maven ecosystem," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 1046–1058.

[12] Y. W. Heng, Z. Ma, H. Zhang, Z. Li *et al.*, "Discovery of timeline and crowd reaction of software vulnerability disclosures," *arXiv preprint arXiv:2411.07480*, 2024.

[13] D. Jaime, J. El Haddad, and P. Poizat, "Navigating and exploring software dependency graphs using goblin," in *Proceedings of the International Conference on Mining Software Repositories (MSR 2025)*, 2025.

[14] I. Rahman, N. Zahan, S. Magill, W. Enck, and L. Williams, "Characterizing dependency update practice of npm, pypi and cargo packages," *arXiv preprint arXiv:2403.17382*, 2024.

[15] N. Zahan, P. Kanakiya, B. Hambleton, S. Shohan, and L. Williams, "Preprint: Can the openssf scorecard be used to measure the security posture of npm and pypi?" *arXiv preprint arXiv:2208.03412*, 2022.

[16] Sonatype, Inc., "Sonatype central search rest api guide," https://central.sonatype.org/search/rest-api-guide/, n.d., accessed: 2024-11-30.

[17] S. Zhao, X. Zhang, and X. Xia, "OpenDigger," Apr. 2021. [Online]. Available: https://github.com/X-lab2017/open-digger

[18] GH Archive, "GH Archive," http://www.gharchive.org, 2024, accessed: 2024-11-30.

[19] CHAOSS Community, "CHAOSS: Community Health Analytics Open Source Software," 2025, accessed: 2025-02-05. [Online]. Available: https://chaoss.community

[20] X-Lab, "X-Lab: Unlocking Innovation," 2025, accessed: 2025-02-05. [Online]. Available: https://www.x-lab.info

[21] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, March 1947.

[22] A. Ikegami, R. G. Kula, B. Chinthanet, V. Maeprasart, A. Ouni, T. Ishio, and K. Matsumoto, "On the use of refactoring in security vulnerability fixes: An exploratory study on maven libraries," in *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 288–293. [Online]. Available: https://doi.org/10.1145/3530019.3535304

[23] CHAOSS Project, "Contributor absence factor," 2025, accessed: 2025-02-01. [Online]. Available: https://chaoss.community/kb/metric-contributor-absence-factor/

[24] D. Rodriguez, M. Sicilia, E. Barriocanal, and R. Harrison, "Empirical findings on team size and productivity in software development," *Journal of Systems and Software - JSS*, vol. 85, 03 2012.

[25] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 772–787, 2011.

[26] H. Al-Shammare, N. Al-Otaiby, M. Al-Otabi, and M. Alshayeb, "An empirical investigation of the security weaknesses in open-source projects," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 634–642. [Online]. Available: https://doi.org/10.1145/3661167.3661279

[27] M. S. Naveed, "Correlation between github stars and code vulnerabilities," *Journal of Computing & Biomedical Informatics*, vol. 4, no. 01, pp. 141–151, Dec. 2022. [Online]. Available: https://jcbi.org/index.php/Main/article/view/111

[28] L. Zhang, C. Liu, S. Chen, Z. Xu, L. Fan, L. Zhao, Y. Zhang, and Y. Liu, "Mitigating persistence of open-source vulnerabilities in maven ecosystem," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 191–203.

[29] M. Cadariu, E. Bouwers, J. Visser, and A. Van Deursen, "Tracking known security vulnerabilities in proprietary software systems," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 516–519.

[30] J. Dietrich, S. Rasheed, A. Jordan, and T. White, "On the security blind spots of software composition analysis," in *Proceedings of the 2024 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, 2023, pp. 77–87.

[31] J. Ruohonen, K. Hjerppe, and K. Rindell, "A large-scale security-oriented static analysis of python packages in pypi," in *2021 18th International Conference on Privacy, Security and Trust (PST)*, 2021, pp. 1–10.

[32] A. Zerouali, T. Mens, A. Decan, and C. De Roover, "On the impact of security vulnerabilities in the npm and rubygems dependency networks," *Empirical Softw. Engg.*, vol. 27, no. 5, Sep. 2022. [Online]. Available: https://doi.org/10.1007/s10664-022-10154-1

[33] M. Zimmermann, C.-A. Staicu, C. Tenny, and M. Pradel, "Small world with high risks: A study of security threats in the npm ecosystem," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 995–1010. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/zimmerman

[34] A. M. Mir, M. Keshani, and S. Proksch, "On the effect of transitivity and granularity on vulnerability propagation in the maven ecosystem," 2023. [Online]. Available: https://arxiv.org/abs/2301.07972

[35] N. E. Gold and J. Krinke, "Ethics in the mining of software repositories," *Empirical Software Engineering*, vol. 27, no. 1, p. 17, Nov. 2021. [Online]. Available: https://doi.org/10.1007/s10664-021-10057-7