Summary Review: Software Composition Analysis and Supply Chain Security in Apache Projects: an Empirical Study

Xiangrui Ke
x3ke@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

Review : Software Composition Analysis and Supply Chain Security in Apache Projects: an Empirical Study

1 Summary

This paper investigates the causal relationship between the adoption of a Software Composition Analysis (SCA) tool—OWASP Dependency-Check (OWASP DC), and the reduction of vulnerabilities in open-source software supply chains. The authors conduct a retrospective cohort study-which is popular in epidemiology study, using 298 Java Maven projects maintained by the Apache Software Foundation (ASF) to compare the evolution of security vulnerabilities in projects that adopted OWASP DC versus those that did not. Employing rigorous statistical techniques, including confounder control and matched sampling to eliminate potential bias, the study finds that OWASP DC is associated with a small but statistically significant reduction in both the number and severity of vulnerabilities, particularly those of high severity. Interestingly, the use of OWASP DC also correlates with a slight increase in low-severity vulnerabilities, possibly reflecting developers' prioritization strategies. This work contributes novel causal evidence to a domain that has previously relied heavily on descriptive or correlational analyses, and it further validates the applicability of cohort design methodologies within software engineering research.

2 Strength

- The paper demonstrates strong methodological rigor. The paper employs a well-designed cohort study framework with careful modified restrictions to confounder control, matching techniques, and appropriate statistical modeling. This enhances the internal validity of the findings and demonstrates methodological sophistication rarely seen in empirical software engineering studies.
- The paper addresses an important gap in the literature: previous work has evaluated how well SCA tools detect vulnerabilities, but no study has examined whether using these tools actually improves software supply chain security over time. By comparing projects that adopt OWASP DC with those that do not, the authors provide the first causal, longitudinal evidence that SCA adoption can reduce vulnerabilities in real-world open-source projects.
- This work holds practical relevance and replicability, since the use of real-world OSS projects from a prominent ecosystem and the open availability of data and scripts make the study highly relevant to practitioners and reproducible for other researchers aiming to replicate or extend the work in different ecosystems. The obtained results have several practical implications and complement the current literature.

3 Weakness

Although the authors justify focusing on Java Maven projects due to their popularity in the JVM ecosystem, the study remains confined to a single language and package manager. This raises concerns about the generalizability of the findings, as SCA effectiveness may vary significantly in ecosystems like npm for JavaScript, PyPI for Python, or Cargo for Rust, where dependency practices and vulnerability profiles differ. A broader analysis would provide a more holistic understanding of SCA tool impact across the software landscape.

4 Future Work

- Future research could explore why low-severity vulnerabilities tend to increase after adopting OWASP DC. This may reflect developer prioritization, but the exact reasons remain unclear.
- To investigate how developers respond to OWASP DC alerts in real-world scenarios. A follow-up study using surveys or structured questionnaires could assess whether vulnerabilities detected by OWASP DC actually lead to remediation actions and how developers prioritize fixes based on severity, perceived risk, or other factors.

5 Discussion on Class

- **Discussion Point 1:** Cross-language Replication
  Feasibility of cross-language studies: The class discussed the potential for replicating this study across different programming languages. The challenge lies in finding equivalent SCA (Software Composition Analysis) tools and dependency management systems across languages that would allow for meaningful comparisons.
  Language-specific security patterns: Different programming languages have different vulnerability patterns and supply chain ecosystems. For example, npm (JavaScript), PyPI (Python), and Maven (Java) each have unique characteristics that might affect the generalizability of findings.
  Tool availability and maturity: The availability and maturity of OWASP Dependency-Check varies across language ecosystems. Some languages may have more comprehensive or different types of SCA tools, which could affect the comparability of results.
- **Discussion Point 2:**

Understanding developer behavior: There was interest in conducting developer surveys or interviews to understand whether OWASP DC alerts actually trigger remediation actions, or if they are often ignored due to alert fatigue or other factors.
Real-world adoption challenges: The class noted that the paper measures correlation between tool adoption and vulnerability reduction, but doesn't deeply explore the human factors like do developers actually act on the alerts, and what barriers prevent remediation?
Behavioral insights needed: Understanding the gap between tool deployment and actual security improvements requires qualitative research into developer workflows, decision-making processes, and organizational constraints.

- **Discussion Point 3:**
  Automated vulnerability assessment: The class explored how AI tools could be integrated into SCA processes to help prioritize vulnerabilities, suggest fixes, or automate remediation for certain types of dependency issues.
  Enhanced developer experience:AI could potentially reduce alert fatigue by providing better context, filtering false positives, or explaining vulnerabilities in more accessible language, making it easier for developers to take action.
  Workflow integration opportunities:
  Discussion touched on how AI assistants could be embedded in the development workflow - from IDE integration to automated pull request generation for dependency updates, making security more seamless.
- **Discussion Point 4:**
  Confounding factors: The class identified that the paper's methodology may not adequately address confounding variables. Projects that adopt OWASP DC might already have better security practices or more security-conscious maintainers, which could explain the reduced vulnerabilities rather than the tool itself.
  Selection bias concerns: There's potential selection bias in which projects choose to adopt SCA tools. More mature, well-resourced, or security-focused projects may be more likely to adopt these tools, making it difficult to establish causation.
  Alternative methodologies suggested:
  The class suggested that stronger causal inference methods (such as difference-in-differences, propensity score matching, or instrumental variables) might be needed to more convincingly demonstrate that the tool adoption itself causes the reduction in vulnerabilities.
  Temporal aspects: The timing of tool adoption relative to vulnerability discovery and fixing needs more careful analysis. Did vulnerabilities decrease because of the tool, or were projects already on an improvement trajectory?

6 Rating

4/5

This paper offers a methodologically rigorous and timely empirical investigation into the real-world impact of software composition analysis tools. While the observed effects are modest, the study makes a meaningful contribution to both academic understanding and industry practice regarding software supply chain security. It demonstrates that OWASP DC can positively impact project security, especially for high-severity vulnerabilities, albeit with small practical effect sizes. The work exemplifies strong empirical research and offers a foundation for further exploration in this increasingly critical domain.