Software Supply Chain Security: A Multi-faceted Approach to Mitigation

1 Abstract

This paper proposes a comprehensive approach to software supply chain security, drawing upon insights from recent research and highlighting the critical need for proactive, data-driven strategies. We address the multifaceted challenges in securing the software supply chain, the limitations of current approaches, and how recent research contributes to the solution space.

2 Problem Statement

The modern software landscape relies heavily on reusable components, creating a complex **software supply chain**. This interconnectedness introduces significant security risks, as vulnerabilities in upstream components can propagate to numerous downstream projects. The increase in software supply chain attacks is alarming, with malicious actors injecting vulnerabilities into open-source software, compromising build and deployment pipelines. The following factors further complicate this problem:

- **Widespread use of open-source software**: A vast majority of codebases contain open-source software. The sheer volume and complexity of these dependencies make it challenging to identify and remediate vulnerabilities.
- **Lack of transparency**: Organizations often lack complete visibility into the components and dependencies within their software, hindering their ability to assess and manage risk.
- **Difficulty in choosing secure dependencies:** Developers lack reliable metrics and processes to select secure dependencies and often struggle with the overwhelming number of vulnerabilities identified by software composition analysis tools (SCA).
- **Challenges in updating dependencies:** Concerns about breaking changes and the time involved often lead to delayed updates, leaving systems vulnerable. Automated patching without human intervention is not always accepted.
- **Malicious commits and packages**: Threat actors intentionally inject malicious code through commits and by creating malicious packages. Detecting these malicious activities is difficult due to obfuscation, stolen credentials, and impersonation.
- **Abandoned and orphaned dependencies:** Open-source projects are often maintained by volunteers, and the abandonment of projects can lead to vulnerabilities that are not addressed, leaving projects relying on them exposed. Vulnerability fixes may not be propagated to previous versions, resulting in residual vulnerabilities and orphaned vulnerabilities that happen through copying dependencies.
- **Evolving attack vectors**: Attackers are constantly finding new ways to exploit vulnerabilities in the software supply chain, including build infrastructure and human factors.
- **Inconsistent vulnerability data**: Public vulnerability databases often lack complete information, including patch links and CVE identifiers.
- **Lack of standardization**: A lack of a standardized format for data interchange across multiple vulnerability and ecosystem API databases complicates the tracking of dependencies and hinders collaboration between different vulnerability databases.

3 How Previous Works Fit into the Picture

Research has explored many facets of this complex problem, providing insights into effective mitigation strategies. The sources in this proposition highlight the following key areas:

- **Dependency Management**:
  - **Metrics for dependency selection**: The OpenSSF Scorecard assesses open-source project security , and research is being conducted on developing metrics, like dependency popularity, contributor reputation, and maintenance [2]. However, the effectiveness of these metrics needs to be improved.
  - **Dependency update practices**: Studies characterize dependency update practices in different ecosystems , proposing metrics like Mean Time To Update (MTTU) and Mean Time To Remediate (MTTR), and explore the impact of dependency specifying strategies (pinned vs. floating versions) on these metrics [2].
  - **Detection of malicious packages**: Researchers have developed methods for detecting malicious packages using rule-based, heuristic, differential analysis, machine learning, and Large Language Model (LLM) approaches. Datasets of malicious packages have been created for research purposes [5].
  - **Vulnerability remediation**: Tools such as Dependabot and Renovatebot help automate dependency updates.
  - **Vulnerability databases**: Public vulnerability databases such as the National Vulnerability Database (NVD) and Open Source Vulnerability (OSV) [2] aggregate vulnerability information. Research also focuses on tools to find missing patch information [5].
- **Software Bill of Materials (SBOM)**:
  - **SBOM Importance**: The generation of an SBOM is essential for tracking third-party components to enable transparency and visibility [5].
  - **SBOM Generation Challenges**: There are issues with SBOM generation, including deficiencies, a lack of generalizability, and incompatibility of standards. Research focuses on improving the quality of SBOMs and tools for generating them [5].
- **CI/CD pipeline security**: Research focuses on integrating security tools into CI/CD pipelines, addressing vulnerabilities introduced through the automation of building, testing, and deployment [5].
- **The role of LLMs**: While LLMs can be used to automate tasks like vulnerability detection [13-19] and code repair, there are risks associated with their use in software development. LLMs can be used as an attack vector and can introduce new vulnerabilities into the software supply chain [5].
- **Security Controls:**
  - **Taxonomy of Challenges**: A taxonomy has been developed for the challenges of implementing security controls [4], highlighting issues such as the subjectivity of human intervention, false positives, and a lack of context.
  - **Control Effectiveness**: Research focuses on the effectiveness of security controls in preventing attacks, and suggests that regular patching and attack surface management are the most effective interventions. A security control metric suite has been developed to measure the degree of mitigation by security controls against attack techniques [3].
- **Sensitive API Usage**:
  - **Security-sensitive API lists**: The creation of lists of security-sensitive APIs is an approach for assessing package security risk, with researchers compiling a list of 219 Java security-sensitive APIs.

  - **Heatmap visualization**: Heatmap visualizations can be used to convey the use of security-sensitive APIs in different packages [1].

- **Human Factors**: Research has identified the need to support developers, recognizing that many companies have basic policies for including external code, but developers need more resources to audit and secure components [1]. Ethical considerations in security research, including privacy and consent, are also explored [5].

4 Discussion and Recommendations

Based on these findings, the following recommendations are proposed:

- **Prioritize a multi-faceted approach**: Addressing software supply chain security requires a multi-layered approach, combining technical solutions with policy and process improvements.
- **Improve Dependency Management**:
  - **Develop and adopt robust metrics**: Focus on developing and using more comprehensive and accurate metrics for evaluating the security of dependencies beyond just the OpenSSF Scorecard.
  - **Promote proactive dependency updates**: Encourage developers to adopt strategies for timely dependency updates, providing better tooling and support, and emphasizing the need to update to remediate vulnerabilities.
  - **Enhance malicious package detection**: Improve existing tools for detecting malicious packages with more advanced machine learning techniques and cross-language analysis.
- **Increase transparency through SBOMs**:
  - **Standardize SBOM generation**: Work towards standardizing the SBOM generation process and improving the quality and reliability of SBOMs.
  - **Automate SBOM generation:** Develop more automated ways to generate SBOMs for programs that utilize multiple languages.
- **Secure CI/CD pipelines**: Integrate security tools into every stage of CI/CD pipelines, specifically addressing the vulnerabilities introduced through automation.
- **Harness LLMs responsibly**: While LLMs offer benefits in software development, be aware of potential risks. Implement thorough reviews and mitigate risks like prompt injection.
- **Improve vulnerability data**: Address the gaps in vulnerability databases by developing tools to backfill missing patch information and encourage greater data standardization.
- **Enhance control implementation**: Treat control implementation as a continuous process, not just a checklist. Prioritize tackling individual challenges, recognizing that addressing one issue might worsen another.
  - **Use a risk-based approach**: Prioritize controls based on their potential to mitigate attacks effectively, considering factors such as technique and tactic coverage, redundancy, and impact reduction.
  - **Adapt security measures**: Emphasize proactive and preventive controls in addition to reactive and detection-based controls.
  - **Recognize the limits of automated systems**: Acknowledge the need for human intervention, while also seeking ways to mitigate risks introduced by human factors.
- **Promote collaboration and knowledge sharing**: Encourage greater collaboration between researchers, practitioners, and open-source maintainers. Sharing knowledge about security incidents including ineffective control implementations can help the community learn from past mistakes.
- **Embrace ethical practices**: Adhere to ethical guidelines, particularly when conducting research that involves human participants or sensitive data.

5 Conclusion

Software supply chain security requires continuous vigilance and a proactive approach. By combining the insights from the research discussed, and following the recommendations made here, we can move towards a more secure and resilient software ecosystem. The key is a data-driven approach to decision-making that incorporates insights on the threat landscape and the effectiveness of defensive measures.

References

[1] Imranur Rahman, Ranidya Paramitha, Henrik Plate, Dominik Wermke, and Laurie Williams. Less is more: A mixed-methods study on security-sensitive api calls in java for better dependency selection, 2024.

[2] Imranur Rahman, Nusrat Zahan, Stephen Magill, William Enck, and Laurie Williams. Characterizing dependency update practice of npm, pypi and cargo packages, 2024.

[3] Md Rayhanur Rahman, Imranur Rahman, and Laurie Williams. If you cannot measure it, you cannot secure it. a case study on metrics for informed choice of security controls, 2024.

[4] Md Rayhanur Rahman, Brandon Wroblewski, Mahzabin Tamanna, Imranur Rahman, Andrew Anufryienak, and Laurie Williams. Towards a taxonomy of challenges in security control implementation. In *Proceedings of the 40th Annual Computer Security Applications Conference*, 2024.

[5] Laurie Williams, Giacomo Benedetti, Sivana Hamer, Ranindya Paramitha, Imranur Rahman, Mahzabin Tamanna, Greg Tystahl, Nusrat Zahan, Patrick Morrison, Yasemin Acar, Michel Cukier, Christian Kästner, Alexandros Kapravelos, Dominik Wermke, and William Enck. Research directions in software supply chain security. *ACM Trans. Softw. Eng. Methodol.*, December 2024. Just Accepted.