Minimum Secure Software Development Testing Requirements at Scale and Pace

A Position Paper submitted by
Dr. Malek Ben Salem, Technology Research Director, Accenture Security &
Chase Sylvester, Security Consulting Manager, Accenture Security
{malek.ben.salem,chase.sylvester}@accenture.com

NIST Workshop on Standards and Guidelines to Enhance Software Supply Chain Security, June 2nd-3rd, 2021

***Abstract* – Software attacks are growing in number due to the complexity of applications, the increased usage of unpatched and third-party software. To address these threats and increase security in the software supply chain, Accenture summarizes the minimal requirements for testing software source code in this paper and identifies best practices and innovations that organizations should adopt to minimize their risk by scaling security.**

INTRODUCTION

An organization's software (SW) supply chain is anything that goes into or affects the organization's code, including open-source code, cryptographic libraries, and Integrated Development Environment tools. In this position paper, we focus on the minimum requirements for testing source code and list best practices that organizations need to adopt to improve the results of their SW testing processes and reduce their susceptibility to SW supply chain attacks.

Accenture believes that every organization should follow a robust Secure Software Development Lifecycle (SSDLC) model that includes security in every phase of the development process. The process starts with threat modeling and security stories to identify the business/mission use case risks and ends with continuous monitoring, threat intelligence assessment, and auto-remediation.

In the following section, we highlight the requirements and steps most relevant to testing SW source code in various phases of the SSDLC process based on Accenture's experience and innovations. We conclude the paper with recommendations and best practices that organizations can adopt to minimize their SW supply chain risk.

MINIMAL TESTING REQUIREMENTS

I. Pre-Commit Stage

In the pre-commit stage organizations should perform the following assessments before each code commit:

- **An architecture/design security assessment** where a threat model is performed to identify risks in the architecture and design of the application. This security review proactively identifies security issues in the application's architecture and prevents critical security design flaws from entering the application's software supply chain. Tools such as IRIUSRisk [1] can help organize and streamline application threat models.
- **A user story assessment** finds relevant security risks and assigns a security acceptance criteria for each user story through creation of a **"Security Story"**. Accenture works with clients to implement **automated security story tagging** of user stories as part of the development lifecycle to **reduce development time**, provide **secure code frameworks** for developers to use during code development and bring awareness to developers regarding **key risks affecting the release**. SAFECode provides resources for setting security acceptance criteria for user stories [2].
- **A manual security code review** with the developer's peers to identify potential security risks before code commit: Resources such as OWASP's security testing guides and CERT's Secure Coding standards can be of use in this step [3]-[4].

II. Commit and Acceptance Stage

During the commit and acceptance stage, the source code is integrated and deployed to a pre-production environment. Several types of tests should be performed in this step:

- **Static Application Security Testing (SAST):** SAST tools scan the source code and locate security vulnerabilities within it during the code build stage. The exact line number when the security issue occurs is identified and shared with the developers for remediation. SAST tools vary in their ability to identify all vulnerabilities. Leading organizations optimize time and resources in this process, through **triaging** of the discovered vulnerabilities to identify false positives. Accenture's AI/ML can automate the triaging process and separate the exploitable vulnerabilities from false positives. **About 64% of security analysts' time can be saved** by automating and scaling this step based on field tests. **Exploitability rankings** can be added to prioritize the vulnerabilities that need to be worked on first. It is also possible to automatically remediate these issues saving development time even further. In our work with a global company, we proved that AI models can automatically remediate **60% of Java vulnerabilities, saving 5 hours of developer/tester** time per vulnerability on average.
- **Static Composition Analysis (SCA):** The SCA process identifies vulnerabilities in open-source repositories used by the application. This is a critical but often neglected step as **98% of the codebases include at least one open-source component** according to a 2021 report

by Synopsys [5]. It is also critical to discover any transitive dependencies in the codebase and environment and evaluate the risks of these dependencies, including vulnerabilities and licensing restrictions. Organizations can subscribe to Github's Dependabot alerts to learn about repositories impacted by a newly discovered vulnerability based on the dependency graph and GitHub Advisory Database [6]. They can also use Dependabot security updates to patch the vulnerabilities [7].

- **Dynamic Application Security Testing (DAST):** This is a blackbox test which identifies vulnerabilities in an application's production-like environment. It is effective in finding externally visible issues in Web applications. However, it does not scan all the source code as it cannot cover more complex flows requiring dynamic data. Accenture is developing **automated mitigation** capabilities based on the findings of DAST scans, where the configurations of WAF and RASP tools are automatically updated to mitigate vulnerabilities in a timely manner. A critical key to the success of this strategy is having a pre-production environment that is configured exactly like the production environment.
- **Interactive Application Security Testing (IAST):** Like DAST, IAST occurs while the application is running in the staging environment. IAST can identify the line of code causing security issues just like SAST, but unlike SAST, it runs during the post-build stage. Because it combines DAST and SAST techniques, IAST can eliminate many false positives, while pinpointing to the exact location of the vulnerability in the source code.
- **Infrastructure As Code, Container and Cloud Testing**: Container images and infrastructure are relatively new, but critical, components of the SW supply chain. Accenture creates and reuses automation playbooks across clients to actively identify critical security configuration issues and risks that scale to thousands of systems. When combined with auto-remediation, **years of manual assessment time** and **hours of development time are saved**, and security risks take **seconds to fix**. Useful tools include Chef, Puppet, Ansible, SaltStack, Terraform, Nessus, NMAP, and CIS Benchmark Tests.
- **API/Web Application Security Fuzzing**: Fuzzing is a blackbox testing technique that identifies security risks focused on the OWASP Top 10 most common security vulnerabilities. Popular automation tools include POSTMAN Collection Runner [8], SOAPUI [9] OWASP ZAP, BurpSuite Pro and SQLMAP.
- **Penetration Testing:** Manual Penetration testing should be performed by an ethical hacker in pre-production and production environments to identify the most active and urgent security issues. Kali Linux has a library of industry standard tools for this testing [10]. Intelligence-driven penetration testing can improve the quality and consistency of the testing done.

RECOMMENDED BEST PRACTICES

Besides the minimal requirements listed in the previous sections, Accenture recommends that organizations adopt the following best practices to minimize SW supply chain risk:

- **Continuous scanning:** Organizations should embrace a continuous scanning approach in lieu of snapshot testing. They can leverage an **automated scanning orchestration platform** to execute the automated tests as part of the software delivery pipeline. They need to compare the results of these scans and measure progress in business-relevant terms/KPIs. An orchestration platform provides additional savings in time and testing costs for the organization.
- **Maintaining a SW Bill of Materials (SBOM):** A SBOM is an inventory of all the components included in the application. This ensures the integrity of the SW supply chain by tracking the "provenance" of each component and maintaining signed builds. It is also important to keep copies of validated good SW components and require that your SW vendors provide a detailed and regularly updated SW BOM.
- **Testing 3rd AI tools:** Testing 3rd components in a codebase should include AI/ML-powered components. The SBOM for such components should include additional metadata, such as the training and testing datasets for the ML model, the accuracy of the model's predictions and the environments they were tested in, etc.

CONCLUSION

The recent increase in SW supply chain attacks calls for more stringent testing requirements of SW source code. In this paper Accenture highlighted the minimal requirements to secure SW source code and listed some best practices that organizations should embrace. By being proactive at addressing the SW supply chain threat, organizations can reduce their risk and save development time while also accelerating business objectives.

REFERENCES

[1] IRIUSRisk: https://www.iriusrisk.com/
[2] SAFECode: https://safecode.org/category/resource-secure-development-practices/
[3] OWASP Top Ten: https://owasp.org/www-project-top-ten/
[4] SEI CERT: https://www.sei.cmu.edu/
[5] Synopsys 2021 Open Source Security and Risk Analysis Report: https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html?cmp=pr-sig
[6] GitHub Dependabot alerts: https://docs.github.com/en/code-security/supply-chain-security/managing-vulnerabilities-in-your-projects-dependencies/about-alerts-for-vulnerable-dependencies
[7] Dependabot Security Updates: https://docs.github.com/en/code-security/supply-chain-security/managing-vulnerabilities-in-your-projects-dependencies/configuring-dependabot-security-updates
[8] POSTMAN Collection Runner: https://learning.postman.com/docs/running-collections/intro-to-collection-runs/
[9] SoapUI – An API testing tool: https://www.soapui.org/
[10] KALI Linux: https://www.kali.org/