Security Considerations: Software Supply Chain

Alexander M. Hoole
Fortify Software Security Research
Micro Focus International PLC
Santa Clara, USA
alexander.hoole@microfocus.com

Abstract— **Enhancing security of the software supply chain is already a work in progress, however, recent events accentuating attacking the software supply chain have shown there is still a long way to go. This position paper presents items we believe are relevant to defining Standards and Guidelines to Enhance Software Supply Chain Security related to Executive Order (EO) 140228 sections: EO 4(g), EO 4(e)(i, ii, ix, and x), EO 4(e)(iv and v) and 4(r).**

0. Introduction

Critical software systems and the supply chains that deliver them should be designed, and evaluated, with security considerations relevant to confidentiality, integrity, and availability (CIA). If a critical system has not been initially designed with security at the fore, then security assessment and mitigation activities should begin as early as possible to minimize risk over time. Numerous articles, standards and best practices have already entered the literature including those by the National Institute of Standards and Technology (NIST) [1][2][3][4], the Common Criteria ISO/IEC 15408, the collection of guides from Defense Information Systems Agency (DISA) Security Technical Implementation Guide (STIG), OASIS SARIF [5], PCI SSF, and OWASP ASVS.

In accordance with Section 4 of the Executive Order on Improving the Cybersecurity of the Federal Government (14028), this position paper focusses on identified standards, tools, best practices, and other guidelines relevant to helping improve software supply chain security. Specifically, Section 1 relates to EO 4(g), Section 2 focusses on EO 4(e)(i, ii, ix, and x), and Section 3 describes topics under EO 4(e)(iv and v) and r(4).

1. Criteria for designating "Critical Software"

Ultimately, critical software should be clearly defined and specifically scoped. The evaluation of critical software, however, must factor in the observation that critical software is part of critical systems. Criteria for evaluating the security of critical software must be considered iteratively. First, direct evaluation of the product/component under test without a contextual environment. Second, the evaluation of the product/component combined with its known dependencies without a contextual environment. Finally, evaluation of the product/component, along with its known dependencies, within its expected environment(s). The concept is to uncover risks using different approaches that are suited for different phases of product maturity during development and maintenance. Continuous evaluation is needed to account for evolving knowledge, technology, and threats. We propose the above observations should be considered when specifying criteria for designating "Critical Software".

2. Initial list of secure software development lifecycle standards, best practices, and other guidelines acceptable for the development of software for purchase by the federal government

Secure software development requires many different facets, each which may have different levels of maturity within a particular organization. These facets have been documented and tracked by maturity models such as OWASP Software Assurance Maturity Model (SAMM) and Building Security In Maturity Model (BSIMM). Security and privacy controls for mitigating security risks are described in NIST SP 800-53 [3] and the Common Criteria (ISO/IEC 15408) provides tools for specifying, evaluating, and verifying assurances. However, there are topics on the horizon which deserve open debate.

A. Software Bill of Materials Visibility

Software Bill of Materials (SBOMs) promises to ease knowledge sharing between parties interested in knowing and understanding the inherent risks associated with a given product by enumerating the software components that the product is dependent upon. Transparency and principle of least privilege can be viewed as opposing views, or positions, and the benefits of having SBOMs available publicly vs. available on a need-to-know basis for critical software should be debated. Consider the audience who need to view SBOMs related to open-source projects/components that are widely consumed by other projects. For such open-source projects, one could argue that the SBOM should be public knowledge (by the very definition of open source). In contrast, consider a critical proprietary component/product that is backing critical infrastructure. In the case of the proprietary software, only those who license the technology would have access to the software artifacts and would have a need-to-know for the SBOM. Proposed formats for SBOM information include the following: Software Package Data Exchange (SPDX), CycloneDX, and Software Identification (SWID) (ISO/IEC 19770-2:2015). While having SBOMs should be viewed as critical for software supply chain security, who has access to specific SBOMs should be critically considered.

B. Automation via integration into DevSecOps pipelines

While need-to-know is relevant to SBOMs, the level of obtained assurances from deployed mechanisms during secure software development is relevant to automation integrations. Specifically, scalability versus coverage should be a consideration in DevSecOps pipelines. Guidelines and best practices must ensure that the security attestations, specifying which risks have been evaluated, for a delivered artifact, meaningfully reduce relevant risks related to the software under test. Selected technologies and processes must support the attestations when recommending automation approaches.

C. Risk consideration related to adopting control(s)

Standards, best practices, and other guidelines often recommend specific technologies to accomplish desired goals.

In the case of software supply chain security, however, the industry has observed attackers leveraging vulnerabilities of existing systems within a targeted environment as the launching point for exploits. Thus, due diligence is required to ensure that technologies deployed to mitigate security risks, and the environments they are conducted within, do not introduce additional risk(s) which could be more severe than what they are preventing.

3. INITIAL MINIMUM REQUIREMENTS FOR TESTING SOFTWARE SOURCE CODE

Within the context of standards, best practices, and other guidelines for secure software development lifecycles lies the domain of testing software source code. In this section we briefly introduce a series of positions to be considered when specifying minimum requirements.

A. ***Coverage***: In terms of reducing the risk of a breach, minimizing false negatives related to "critical" unknown weaknesses and vulnerabilities is more important than checking a compliance box that "some testing" has been completed. The capabilities of the selected technologies must be aligned to the security requirements of the critical software under test regarding identification and mitigation of relevant risks. Vectors include the following: depth of analysis over context of source code, programming language coverage, API coverage, weakness/vulnerability coverage, and type of analysis. Each of these vectors have an impact on the ability to detect a given flaw.

B. ***Metrics and Filters:*** Metrics are continually evolving to esnure that software development can be aware of the most impactful critical vulnerabilities to be mitigated (e.g. CWE Top 25, OWASP Top 10, etc.). Presently, however, the majority of thes metrics are industry and technology agnostic. Moving forward, technology/industry domain-specific and role-specific reporting of issues that are relevant for target audiences can help reduce waisted resources (e.g. suppress reported issues not relevant to a target audience).

C. ***Software Component Analysis***: Improving patch management practices and component tracking to reduce risk related to publicly known vulnerabilities is critical to risk reduction. In parallel to reducing risk in the software we write is the need to identify and protect against the risks introduced by the code we consume. We must continue to expand capabilities in software component/composition analysis domain to provide more precise risk assessment related to the susceptibility of risks impacting a specific product under test. Consideration must also be given to risks that bring into context risks which may be viewed as out of context. Specifically, vulnerabilities related to deserialization and reflection have the ability to instantiate code that naively could be considered out of scope, For example, a gadget in the Java classpath could bring a CVE into scope that otherwise is not visibly referenced in the source code of the application under test.

D. ***Insider Threat***: Supply chain threats must be considered from both an internal and external risk perspective. Starting from the broader scope of insider threat, such as covered in the Insider Threat Mitigation Guide by the Cybersecurity and Infrastructure Security Agency, we must eventually arive at the specifics for identifying insider threats technically and indications to whether or not they are intentional or accidental. Technologies which support the identification of "injected flaws" and how they came to be could reduce future risk.

E. ***Auditing and triaging***: Far too many individuals expect results from application security products to be actionable intelligence and skip the auditing step to verify a given finding. When discussing process, it would be prudent to emphasize that while security tools should be integrated into the DevSecOps pipeline and run automatically, providing mechanisms and guidance for developers/security personnel to validate, verify, and triage results in systems which provide efficient and time-saving interfaces and controls is also needed.

   Leveraging AI/ML systems to be a force multiplier when dealing with testing results, to capture the knowledge of their most proficient appsec professionals, and adapt to the specific threat postures of an organization can also be considered a best practice.

F. ***Education:*** Continuous targeted training and education of development community towards the goal of reducing the number of future weaknesses/vulnerabilities. Since the risk related to attrition and turnover is very real in the software development and application security domain, education is required to ensure minimal levels of competencies.

ACKNOWLEDGMENT

We appreciate the opportunity to provide comments to the NIST on the Standards and Guidelines to Enhance Software Supply Chain Security.

REFERENCES

[1] Boyens, J. , Paulsen, C. , Bartol, N. , Winkler, K. and Gimbi, J. (2021), Key Practices in Cyber Supply Chain Risk Management: Observations from Industry, NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, Gaithersburg, MD, [online], https://doi.org/10.6028/NIST.IR.8276 (Accessed May 26, 2021).

[2] Boyens, J. , Paulsen, C. , Moorthy, R. and Bartol, N. (2015), Supply Chain Risk Management Practices for Federal Information Systems and Organizations, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, [online], https://doi.org/10.6028/NIST.SP.800-161 (Accessed May 26, 2021).

[3] Ross, R. and Pillitteri, V. (2020), Security and Privacy Controls for Information Systems and Organizations, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, [online], https://doi.org/10.6028/NIST.SP.800-53r5 (Accessed May 26, 2021).

[4] Ross, R. (2018), Risk Management Framework for Information Systems and Organizations: A System Life Cycle Approach for Security and Privacy, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, [online], https://doi.org/10.6028/NIST.SP.800-37r2 (Accessed May 26, 2021).

[5] Static Analysis Results Interchange Format (SARIF) Version 2.0. Edited by Michael Fanning and Laurence J. Golding. 27 May 2019. OASIS Committee Specification Draft 02 / Public Review Draft 02. https://docs.oasis-open.org/sarif/sarif/v2.0/csprd02/sarif-v2.0-csprd02.html. Latest version: https://docs.oasis-open.org/sarif/sarif/v2.0/sarif-v2.0.html.