Raytheon Technologies Position Paper (item 4) on Standards and Guidelines to Enhance Software Supply Chain Security

May 26, 2021

Authors

Primary: Randall Brooks (Principal Engineering Fellow, Raytheon Intelligence and Space)

Secondary: Kelly FitzGerald (Associate Director and Architect, CODE Center Product Security Architecture and Risk Evaluation, Raytheon Intelligence & Space)

Contributors: Edward Bonver (Director, Product Security Architecture and Risk Evaluation for CODE Center, Raytheon Intelligence & Space); Fred Jones (Munitions Cyber Research & Development Principal Investigator, Raytheon Missiles & Defense); Jay Lala (Senior Principal Engineering Fellow at Raytheon Technologies, Product Cybersecurity Officer (PCO), Raytheon Missiles and Defense); William Lamberti (Director, Software & Product Intelligence, Pratt & Whitney); Bret Lynch (Manager, Software & Product Intelligence, Pratt & Whitney); Anne Murray (Director of Cyber & Secure Systems, Raytheon Intelligence & Space); Linda Peyton (Senior Director of Engineering Infrastructure & Integrity, Collins Aerospace); Gregory Ritter (Engineering Fellow; Raytheon Intelligence & Space); Stuart Schwartz (Principal Security Engineer, CODE Center Product Security Architecture and Risk Evaluation, Raytheon Intelligence & Space); Harold Toomey (Associate Director, CODE Center Product Security Architecture and Risk Evaluation, Raytheon Intelligence & Space)

Overview

This position paper is to address NIST's request for:

> *4. Initial minimum requirements for testing software source code including defining types of manual or automated testing (such as code review tools, static and dynamic analysis, software composition tools, and penetration testing), their recommended uses, best practices, and setting realistic expectations for security benefits. See EO Sections 4(e)(iv and v) and 4(r).*

This paper focuses on covering the initial minimum requirements for testing software with consideration for the process of developing secure software (referred to as Software Assurance or Application Security). This testing includes the source code, as specified, and includes the compiled binary and its execution.

When looking at the minimum requirements for testing, one must first look at current practices that are being utilized in the industry to find vulnerabilities. These practices are Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), Interactive Application Security Testing (IAST), Cognative Assistance and Visualization, and Automation.

SAST

The minimum requirements should include static code analysis (SCA) checking for the Common Weakness Enumerations (CWEs) via automated tooling and manual processes. The minimum should include checking for the most common CWEs with the understanding that no single tool will find all CWEs (e.g. CWE Top 25). This is because each tool has its own bias, approach and techniques, which means the strongest approach is a garden of tools, not a monoculture of a single tool. It should also find and deny the use of any deprecated functions to protect the software from potential harm.

Static analysis tools must have CWE checkers on by default as a simple out of the box scan will not yield any Cybersecurity relevant findings. A stretch goal should consider the use of multiple static analysis tools, such as what is done in the formally DHS-funded Software Assurance Marketplace (SWAMP). Additionally, we would like to see a sense of determining one's own "Top 25" as Web software is different than application or embedded software. Static analysis should include documented peer reviews as no single tool can determine the intention of the developer.

At a minimum deja vmpedigree and provenance of the application scanning should be performed to find any Free and open-source software (FOSS) components the application is using that have vulnerabilities that are publicly disclosed in the Common Vulnerabilities and Exposures (CVE) database. When a new vulnerability is publicly disclosed then the project owner can be notified so the vulnerable FOSS component can be updated. This will help prevent situations such as Equifax where their system was compromised due to a vulnerable FOSS component.

Beyond SAST of source code, the minimum requirements should include an analysis of the compiled binary through reverse engineering. Tools like IDA-pro, Ghidra, Binary Ninja, etc. can expand code coverage and include aspects of Cyber Supply Chain Risk Management (C-SCRM). Code integrated through standard development process includes code that becomes a part of one's product through static and dynamic linked libraries. These tools should be required to look for "back doors" or built-in credentials in these libraries.

DAST

As SAST analysis will help by finding CWEs, to determine if the CWEs are potentially exploitable, testing running code should be a minimal requirement. DAST should include nominal and off nominal testing. The code should be stressed by DAST for items likely memory leaks and denial of service. DAST testing should test for how memory is utilized and whether it is properly handled. Fuzz testing should be applied to all interfaces as determined by an attack surface analysis of a Threat Model. Fuzz testing can be done on defined protocols or the more common methodology of custom protocols for the product being developed. An important metric to determine is the duration of time DAST must be conducted (to include IAST). The longer the testing is done; the more code will be exercised. DAST can have the sense of diminishing rate of returns when the same style of approach of testing is repeatedly done (i.e. fuzz testing exercised with the same data set). For those projects utilizing agile methods, an additional sprint should be allocated to DAST testing.

Penetration testing is another form of dynamically running the code. The focus of penetration testing should be 3rd party testing, which is covered later.

IAST

IAST extends the minimum requirements of DAST by adding instrumentation. This instrumentation should be able to tell if an input can have direct control on execution. Thus all processor registers should be able to be read to determine if the DAST testing yields a read or write to the heap or stack or even a registry, data store / file system, etc. This can be done in a debug state or instrumented virtual systems. Testing tools that can run DAST testing such as an instrumented virtual machine provide the benefit of recording a history of execution to understand why a crash occurred. This type of testing should be considered as a stretch goal with general debugging capabilities as a required minimum.

Cognitive Assistance and Visualization

As a best practice, human-based analysis of software systems will continue to be an important part of software risk mitigation going forward. This human-in-the-loop analysis requires technical support to be effective and efficient. Software visualization and cognitive assistance technologies enable quicker, deeper understanding of software design intent and the motivations behind implementation decisions. Software visualization should support understanding of the existing source code as well as its evolution though analysis of configuration management data. The ideal software visualization tool will provide context to software changes and support quantitative software risk analysis.

Automation

It should be a goal in every Cybersecurity-focused project to get the “Security” out of the way. For terms like DevSecOps, the focus should be Dev and Ops with Cybersecurity built into the process. This can be done through automation. SAST, DAST and IAST should be automated as much as possible to remove human error. However, automation is not the only answer. Bug bars should be set, and code can be tested through its lifecycle pushing towards operation of said code. If the code fails the bug bars, feedback must be given to the developers. This will likely yield additional manual tests.

Conclusion

Additionally, this position paper must state that tools such as Nessus, Metasploit, etc. are targeted for known vulnerabilities. The testing of software should be focused on finding the unknown vulnerabilities. These tools should be considered as optional. They cover penetration testing or vulnerability assessment of a system but will not provide any benefit over the testing outlined above. 3rd Party testing will provide a greater benefit. Developers can sometimes not see the issues they are faced with. If they are the ones testing the software, they will likely be too closely coupled to the functionality of the code and not have a “Think Evil” mentality on their own system. At a minimum either SAST, DAST or IAST should be done by a 3rd party who focuses on vulnerability research. We would recommend this for SAST as that tends to happen earlier in the lifecycle. A stretch goal would include DAST or IAST when the software system is complete.

A simple statement with regards to testing is that the last time one has full control of their software is the last day they test and ship. An attacker is not limited by a deadline. In the end no amount of testing will find all bugs and even architectural flaws pose other risk. Setting a minimum set will greatly improve what is done today.