Submission to NIST CFP for "Initial minimum requirements for testing software source code"
by Chris Wysopal, CTO & co-founder, Veracode, cwysopal@veracode.com

Veracode is the largest independent vendor of Application Security Testing services, enabling enterprises and software vendors to create secure software, by helping to both find and fix vulnerabilities in software code. Our recommendations are backed by 15 years of insights scanning nearly 30 trillion lines of code and helping customers fix more than 60 million security flaws. In addition to automated testing platform (SAST, DAST, IAST, SCA), we have a team of researchers who are steeped in the process of identifying flaws and remediation best practices, as well as a team of manual pen testers.

There are strengths and weaknesses for all software security tools and manual processes. This has led to most organizations using multiple techniques to uncover potential vulnerabilities. Yet there are diminishing returns in findings and increased time and cost associated with using multiple tools especially when their capabilities overlap.

Automation has the benefit of being fast, repeatable, and less expensive than manual processes. This makes automated tools a good foundation onto which manual process are added to fill in analysis gaps present from the automated tools. In general, automated tools are good for detecting coding implementation errors that lead to vulnerabilities and manual processes are good for detecting design and business logic errors that lead to vulnerabilities.

We recommend a minimum requirement of automated tools and manual processes.

Static analysis tools (SAST) are fast and excel at finding coding implementation errors with excellent code coverage. However, they do not take the execution environment into consideration because they cannot see interactions with other systems, and they do not understand the software's design. As a result, some static analysis findings may be mitigated by the software's architecture or execution environment. The SAST tool must support the languages and frameworks used in the source code.

We recommend using static analysis tools with a mitigation workflow that provides an auditable workflow for documenting findings that are mitigated by other factors and therefore do not need to be corrected in the code.

We recommend security critical areas of code, such as authentication, authorization, data validation, cryptography, etc. receive manual code reviews for correctness.

Dynamic analysis tools (DAST) do take the execution environment into consideration at the expense of needing to interact with the software in real-time which can lead to long testing times for complex software. Another weakness of dynamic analysis tools is code coverage. It is difficult for the tools to discover and attack all the code paths in a reasonable amount of time. Even with these weaknesses, simulating the way an attacker interacts with a running web application or API can find vulnerabilities that have complexities which make it difficult for static analysis tools to find.

We recommend augmenting static analysis tools with dynamic analysis tools.

Third party code components included as code dependencies present a different software assurance problem. That is the issue of a known vulnerability inherited from a publicly available component. No matter how that known vulnerability was discovered, if it creates a vulnerability in the software it is included in, it must be

remediated. This is typically done by updating to a newer version of the component where that vulnerability has been remediated. If that is not possible, the first-party code may need to be changed to avoid the vulnerable code path, or the component itself could be edited manually until a safe version is released. A class of tool named software composition analysis (SCA) can determine what third party components and which versions of those components are being used in software. The tool can cross-reference against public and private vulnerability databases to determine if the 3rd party components have known vulnerabilities. SCA tools must support the language of the source code and components.

We recommend using SCA tools to identify and remediate vulnerabilities inherited from third-party components and can cross reference against NVD.

SCA tools can also create an SBOM which can be monitored to alert downstream consumers of the software when new public vulnerabilities in the 3rd party components are discovered. However, not all vulnerabilities in components create vulnerabilities in the software that depends on them. SBOMs should include information as to whether a component with a vulnerability makes the software that uses it vulnerable.

We recommend the use of SCA tools that can create an SBOM and has a way to communicate to downstream consumers if vulnerabilities in components impact the software.

We recommend the combination of SAST, DAST, and SCA so that testing covers top vulnerability categories including the OWASP Top 10 and CWE Top 25.

Automated design tools and automated design analysis are unfortunately limited and not much in use today. This means a manual process must be undertaken to identify vulnerabilities in the software design. The best process to perform is threat modeling. As with all manual processes the people conducting the process must be trained to perform it. A standardized artifact should be created from the process which documents the risks uncovered and how they were mitigated. Since threat modeling is a manual process, it is impractical to perform comprehensively for every new feature or design change. In those cases, that part of the software should be threat modeled and the overall threat model updated accordingly.

We recommend a full threat model with continuous incremental threat modeling performed as needed.

Unfortunately, business logic issues and many complex design issues will not be detected with threat modeling or automated testing. Manual penetration testing (MPT) addresses these gaps simulating a skilled adversary to identify complex vulnerabilities that a human attacker would exploit. MPT requires testers properly trained. Since it is a manual process the entire software functionality cannot be tested each time the code changes. The full software should be tested at a point in time, augmented by incremental testing when a certain threshold of code change has occurred.

We recommend a full MPT periodically with continuous incremental MPT performed as needed.

To document the process and results for consumers/operators of the software a summary report should contain: description of the code assessed by source and/or binary filenames and hashes, tools used by type, name and version, manual processes performed with name of organization(s), summary of residual risk detected but not remediated, mitigations accepted, and an SBOM.

Appendix

About the author, Chris Wysopal

Chris Wysopal is Co-Founder and Chief Technology Officer at Veracode, which pioneered the concept of using automated static binary analysis to discover vulnerabilities in software. He is a member of the executive team. Prior to Veracode, Chris was vice president of research and development at security consultancy @stake, which was acquired by Symantec.

In the 1990's, Chris was one of the original vulnerability researchers at The L0pht, a hacker think tank, where he was one of the first to publicize the risks of insecure software. He has testified to the US Congress about government security and how vulnerabilities are discovered in software.

Chris started his career as software engineer that first built commercial software and then migrated to the specialty of testing software for vulnerabilities. He researched software security for the first vulnerability research think tank, L0pht Heavy Industries, from 1994-1999. He is the author of *The Art of Software Security Testing*, published by Addison-Wesley.

Examples of supply chain artifacts and programs:

1. Description of **Veracode Verified** 3-tier application security testing levels. https://www.veracode.com/sites/default/files/pdf/resources/infosheets/veracode-verified-transformation-veracode-infosheet.pdf
2. Directory of software products that have attained a **Veracode Verified** status: https://www.veracode.com/verified/directory
3. Veracode **Vendor Application Security Testing (VAST)** program for managing software supply chain risk. https://www.veracode.com/sites/default/files/Resources/Datasheets/vast-program-for-enterprise-datasheet.pdf
4. Examples of a vendor sharing application **security testing summary report**: https://clients.collegesource.com/home/download/attachments/143360393/SummaryReport_Transferology_9_Oct_2020.pdf?version=1&modificationDate=1602257222570&api=v2