Executive Order – NIST workshop position paper #4

(Testing software source code)

Microsoft Corporation

Focus Area

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).

Microsoft Response

Existing detailed requirements and guidance for secure software testing as outlined in the NIST Secure Software Development Framework and Microsoft's published guidance on the Security Development Lifecycle should be referenced and leveraged for initial minimum requirements. The below recommendations provide further context for both initial minimum requirements and ongoing advancements that could be considered as requirements evolve. An iterative approach to the development and implementation of requirements creates opportunities to reflect dynamic threat models and incorporate practice enhancements, and a phased roll out promotes clarity and consistency for supplier planning and engineering and minimizes disruptions to agencies.

Recommendations

1 – Automated and Manual Testing

Microsoft advocates for utilizing multiple security testing methodologies and technologies in conjunction, as each approach has limitations individually, and testing should include source code, compiled binaries, and the operational environment when that is under the supplier's control (i.e., software is hosted rather than distributed to customers). These can be put into two categories that inform the approach to their use: solutions that are scalable enough to be applied broadly to all applicable source code, and solutions that must be targeted in their usage due to their high manual effort or per source code base cost.

Manual code reviews, built in compiler security verifiers, a variety of Static Analysis Security Testing (SAST) tools, and Dynamic Analysis Security Testing (DAST) tools that are designed to automatically test or scan should be applied to all applicable code (e.g., Web Application vulnerability scanners, configuration, and network/host vulnerability scanners, etc.). Code Reviews performed by an appropriate number of people should be conducted on any commits prior to their merge into code branches. Microsoft recommends SAST automation for Software Composition Analysis, Binary verification (for optional platform mitigations, digital signatures, etc.), scanning for any hardcoded passwords/tokens/cryptographic keys, and general vulnerability analysis. The capability of the SAST for general vulnerability analysis should be proportional to the complexity of the source code analyzed (e.g., individual scripts only require local scope analysis, full web applications justify interprocedural analysis). Given that the more sophisticated SAST and DAST tools can take significant time to execute, particularly against larger code bases, Microsoft recommends that narrower scope automation be run on each commit, and more sophisticated SAST and DAST be run on a reoccurring cadence and at software development milestones.

For testing methodologies and technologies that are not feasible to apply broadly to all source code, such as manual security testing by experts or forms of DAST that require significant per component investment to implement or execute (e.g., Fuzzing, runtime analysis that requires custom code to execute application functions, etc.), suppliers should have documented rationale for identifying and prioritizing to what source code those activities are applied. Additionally, all source code implementing security features or security mitigations should have purpose built positive and negative unit tests to verify that they function as intended and fail securely when presented with unexpected inputs.

Suppliers should have an inventory of the detections or approaches (when predefined) they run against each of the code bases and document the rationale for their selection. The ruleset should focus on vulnerability patterns whose context to identify the issue is available to scrutinize for that form of automation (e.g., static analysis running on source should not try to detect issues that require awareness of the configuration of the end device environment it may execute in) and be relevant to the type of component being tested. Not all programming languages will have the same maturity of automation; newer languages are informed by the security lessons of previous languages. Rather than preclude the use of languages, other aspects of assessment, including manual assessment, should be used to compensate and achieve assurance.

2 – Range of Requirements

Microsoft believes that source code should undergo security testing, using an appropriate combination of people, process, and technology. This testing should be automated to the extent feasible and, where possible, built into the systems and workflows that software developers already use. All identified security defects should be triaged and fixed within an agreed-upon SLA (based on severity). Security testing is often context-specific, depending on things like the programming languages used, the type of environment the software is deployed to, the system’s threat model, and the existence of complementary security controls.

High-impact software or software components should undergo additional scrutiny, including the following:

- The system’s threat model should be periodically validated to ensure it reflects the current system design and expected threats, and that all mitigations are sufficient and working properly.
- The system should undergo periodic penetration testing by qualified personnel.
- All third-party and open-source software components should also be analyzed using automated SAST tools, triage, and actioned appropriately.
- Use fuzzing to identify security vulnerabilities in software written in non-memory-safe languages.
- All released software artifacts should be tested to ensure they are deterministic.

3 – Verification of Conformity

Once the testing requirements have been published and suppliers begin applying them to their processes, there will need to be a way to verify that the requirements have been followed. This can be done both prior to implementing a supplier’s solution and/or periodically throughout its usage lifecycle. The solution for measuring and ensuring conformity can be a range of options from self-attestation, audit, to 3rd party accreditation. Each approach varies in terms of effort, resources, level of assurance, and impact to providers. In a self-attestation model, the supplier asserts their compliance to source code testing requirements. Microsoft recommends a scalable self-attestation model, backed by a Software Bill of Materials (SBOM) that includes a list of verification/testing performed. A supplier provided SBOM with a standard schema and minimal elements that are specific to testing performed and validation results should be required for high-impact software.

4 – Frequency of validation of source code

Microsoft recommends performing validation:

a. Before making the software generally available; and
b. As early and often as reasonable, given the context, tooling limitations, etc.

If validation is performed before making software generally available, then it may be necessary to delay the release of software if anything is found. Ideally some portion of SAST should be run before completing acceptance, blocking issues from appearing on the source code repository. Dependencies on third party code bases require a full software release validation for each dependency update.