NISTIR 8397
Guidelines on Minimum Standards for Developer Verification of Software

Executive Order on Improving the Nation's Cybersecurity

Issued May 12, 2021 by President Biden

One Key Areas Covered by this Executive Order: Enhancing Software Supply Chain Security

- NIST, in consultation with other named federal agencies, is directed to solicit "input from the Federal Government, private sector, academia, and other appropriate actors to identify existing or develop new standards, tools, and best practices for complying with the standards, procedures, or criteria. The guidelines shall include criteria that can be used to evaluate software security, include criteria to evaluate the security practices of the developers and suppliers themselves, and identify innovative tools or methods to demonstrate conformance with secure practices"
- To respond to this requirement NIST created NISTIR 8397 Guidelines on Minimum Standards for Developer Verification of Software

NISTIR 8397
Guidelines on Minimum Standards for Developer Verification of Software

- Recommends **minimum standards** (not "best practices") of software verification by software producers
- Based on assumption no single software security verification standard can encompass all types of software and be both specific and prescriptive while supporting efficient and effective verification
- Recommends guidelines for software producers to use in creating their own processes
- Provides for the process to be very specific and tailored to the software products, technology (e.g., language and platform), toolchain, and development lifecycle model

NISTIR 8397
Guidelines on Minimum Standards for Developer Verification of Software

Key Terms

- Software: Executable computer programs
- Testing: Any technique or procedure performed on the software itself to gain assurance that the software will perform as desired, has the necessary properties, and has no important vulnerabilities
- Verification: Includes methods such as static analysis and code review, in addition to dynamic analysis or running programs
  - Verification assumes standard language semantics, correct and robust compilation or interpretation engines, and a reliable and accurate execution environment, such as containers, virtual machines, operating systems, and hardware. Verification may or may not be performed in the intended operational environment.
  - Includes vendor and developer testing

NISTIR 8397
Guidelines on Minimum Standards for Developer Verification of Software

SCOPE

- Includes “software source code” and software in general including binaries, bytecode, and executables, such as libraries and packages
- Does not include specialized testing regimes such as real-time software, firmware (microcode), embedded/cyberphysical software, machine learning (ML) or neural net code
- Excludes ancillary yet vital material such as configuration files, file or execution permissions, operational procedures, and hardware

Minimum Standards for Developer Testing

- #1. Do Threat Modeling
  - Use threat modeling early in order to identify design-level security issues and to focus verification
  - Software needs should drive the threat modeling method(s) used.
  - Should be done *multiple* times during development, especially when developing new capabilities, to capture new threats and improve modeling
  - Test cases should be more comprehensive in areas of greatest consequences, as indicated by the threat assessment or threat scenarios.
  - Threat modeling can also indicate which input vectors are of most concern. Testing variations of these particular inputs should be higher priority.
  - Threat modeling may reveal that certain small pieces of code, typically less than 100 lines, pose significant risk and require additional code review

Minimum Standards for Developer Testing

- #2. Do Automated Testing
  - Can be as simple as a script that reruns static analysis, then runs the program on a set of inputs, captures the outputs, and compares the outputs to expected results
  - Can be as sophisticated as a tool that sets up the environment, runs the test, then checks for success.
  - Recommend automated verification to:
    - ensure that static analysis does not report new weaknesses,
    - run tests consistently,
    - check results accurately, and
    - minimize the need for human effort and expertise.
  - Automated verification can be integrated into the existing workflow or issue tracking system
  - Because verification is automated, it can be repeated often, for instance, upon every commit or before an issue is retired

Minimum Standards for Developer Testing

- #3. Code-Based, or Static, Analysis
  - Divided into two approaches: 1) code-based or static analysis (e.g., Static Application Security Testing—SAST) and 2) execution-based or dynamic analysis (e.g., Dynamic Application Security Testing—DAST).
  - Pure code-based analysis is independent of program execution. Questions that a scanner may address include:
    - Does this software *always* satisfy the required security policy?
    - Does it satisfy important properties?
    - Would any input cause it to fail?
  - Recommend using a static analysis tool to check code for many kinds of vulnerabilities and for compliance with the organization's coding standards.

Minimum Standards for Developer Testing

- #4. Review for Hardcoded Secrets
  - Recommend using heuristic tools to examine the code for hardcoded passwords and private encryption keys. Heuristic tools may assist by identifying small sections of code that are suspicious, possibly triggering manual review
- #5. Run with Language-Provided Checks and Protection
  - Use Programming languages, both compiled and interpreted, provided many built-in checks and protections both during development and in the software shipped
  - Enable hardware and operating system security and vulnerability mitigation mechanisms, too
  - For software written in languages that are not memory-safe, consider using techniques that enforce memory safety
  - Interpreted languages typically have significant security enforcement built-in, although additional measures can be enabled. In addition, you may use a static analyzer, sometimes called a “linter”, which checks for dangerous functions, problematic parameters, and other possible vulnerabilities

Minimum Standards for Developer Testing

- #6. Black Box Test Cases
  - "Black box" tests are not based on the implementation or the particular code. Instead, they are based on functional specifications or requirements, negative tests (invalid inputs and testing what the software should *not* do), denial of service and overload, described in Sec. 3.8, input boundary analysis, and input combinations
  - Tests cases should be more comprehensive in areas indicated as security sensitive or critical by general security principles
  - If you can formally prove that classes of errors cannot occur, some of the testing described above may not be needed
  - Additionally, rigorous process metrics may show that the benefit of some testing is small compared to the cost

Minimum Standards for Developer Testing

- #7. Code-Based Test Cases
  - Code-based, or structural, test cases are based on the implementation, that is, the specifics of the code
  - Code-based test cases may also come from coverage metrics. E.g., the software may record which branches, blocks, function calls, etc., in the code are exercised or "covered". Tools then analyze this information to compute metrics. Additional test cases can be added to increase coverage
  - Most code should be executed during unit testing
  - Recommend that executing the test suite achieves a minimum of 80% statement coverage

Minimum Standards for Developer Testing

- #8. Historical Test Cases
  - Some test cases are created specifically to show the presence (and later, the absence) of a bug. These are sometimes called "regression tests"
  - An even better option is adoption of an assurance approach, such as choice of language, that precludes the bug entirely
  - Inputs recorded from production operations may also be good sources of test cases
- #9. Fuzzing
  - Recommend using a fuzzer, which performs automatic active testing; fuzzers create huge numbers of inputs during testing. Typically, only a tiny fraction of the inputs trigger code problems. In addition, these tools only perform a general check to determine that the software handled the test correctly. Typically, only broad output characteristics and gross behavior, such as application crashes, are monitored
  - The advantage of generality is that such tools can try an immense number of inputs with minimal human supervision. The tools can be programmed with inputs that often reveal bugs, such as very long or empty inputs and special characters

Minimum Standards for Developer Testing

- #10. Web Application Scanning
  - If the software provides a web service, use a dynamic application security testing (DAST) tool, e.g., web application scanner (IAST) tool to detect vulnerabilities. Web app scanners create inputs as they run. A web app scanner monitors for general unusual behavior. A hybrid or IAST tool may also monitor program execution for internal faults. When an input causes some detectable anomaly, the tool can use variations of the input to probe for failures
- #11. Check Included Software Components
  - The components of the software must be continually monitored against databases of known vulnerabilities; a new vulnerability in existing code may be reported at any time
  - A Software Composition Analysis (SCA) or Origin Analyzer (OA) tool can help you identify what open-source libraries, suites, packages, bundles, kits, etc., the software uses and noting software that is out of date or has known vulnerabilities

Beyond Software Verification

- Good Software Development Practices
  - Follow the recommendations in NIST Special Publication 800-218 Secure Software Development Framework (SSDF) Version 1.1**:** *Recommendations for Mitigating the Risk of Software Vulnerabilities*
  - Enterprises with secure development should include the following characteristics:
    - Create a culture where security is everyone's responsibility. This includes integrating a security specialist into the development team, training all developers to know how to design and implement secure software, and using automated tools that allow both developers and security staff to track vulnerabilities
    - Uses tools to automate security checking, often referred to as Security as Code
    - Tracks threats and vulnerabilities, in addition to typical system metrics
    - Shares software development task information, security threat, and vulnerability knowledge between the security team, developers, and operations personnel

Beyond Software Verification

- Good Software Installation and Operation Practices
  - Configuration files:
    - Software releases should include secure default settings and caveats regarding deviations from those settings.
    - Security verification should include all valid settings and (possibly) assurance that invalid settings will be caught by run-time checks.
    - The acquirer should be warned or notified that settings other than those explicitly permitted will invalidate developer's security assertions
  - File Permissions:
    - File ownership and permissions to read, write, execute, and delete files need to be established using the principle of least privilege.
    - The ability to change file permissions needs to be restricted to explicitly authorized subjects that are authenticated in a manner that is commensurate with the impact of a compromise of the software
    - The role of file permissions in maintaining security assertions needs to be explicit

Beyond Software Verification

- Good Software Installation and Operation Practices (cont'd)
  - Network configuration:
    - Verification needs to cover all valid network configuration settings and (possibly) provide assurance that invalid settings will be caught by run-time checks
    - The role of network configuration in scoping the applicability of security assertions needs to be explicit
  - Operational configuration:
    - Verification needs to be conducted in an environment that is consistent with the anticipated operational configurations
    - Any dependence of the security assertions on implementing software or other aspects of operational configuration needs to be made explicit by the developer
    - Supply chain integrity must be maintained

Beyond Software Verification

- Additional Software Assurance Technology
  - Nearer-term advances that may add to security assurance based on verification include:
    - Applying machine learning to reduce false positives from automated security scanning tools and to increase the vulnerabilities that these tools can detect
    - Adapting tools designed for automated web interface tests, e.g., Selenium, to produce security tests for applications
    - Improving scalability of model-based security testing for complex systems
    - Improving automated web-application security assessment tools with respect to:
      - Session state management
      - Script parsing
      - Logical flow
      - Custom uniform resource locators (URLs)
      - Privilege escalation
    - Applying observability tools to provide security assurance in cloud environment
    - Adapting current security testing to achieve cloud service security assurance