4. Testing

David A. Wheeler <dwheeler@linuxfoundation.org> (et al)
Director of Open Source Supply Chain Security, The Linux Foundation (*willing to speak*)

> **NIST Request:** *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).

We note that this request doesn't cover just "testing" in the sense of execution testing, but the broader goal of verification (in particular, verifying that the software is adequately secure for purpose). No single type of manual analysis or automated verification tool will efficiently find all problems; in general, a collection of approaches are needed for highly secure software.

Some sources for this information include:

1. The OpenSSF's CII Best Practices badge project specifically identifies best practices for OSS, focusing on creating higher-quality secure software and including criteria to evaluate the security practices of developers and suppliers (it has over 3,800 participating projects).
2. The Secure Software Development Fundamentals set of courses available on edX to anyone at no cost, and is available in markdown format. It also includes guidelines on which tools to use, particularly in its section on verification. It particularly emphasizes that projects not doing anything should start by looking at these tools: Generic Bug-Finding Tools: Quality Tools, Compiler Warnings, and Type-Checking Tools; Security Code Scanners/Static Application Security Testing (SAST) Tools; Secret Scanning; Software Component Analysis (SCA)/Dependency Analysis; Traditional Testing (including negative tests); Fuzz Testing; and Web Application Scanners.
3. The OpenSSF Best Practices Working Group (WG) actively works to identify and promulgate best practices.
4. Reproducible builds are the primary way to give confidence that a built package matches its claimed source code. Otherwise, attacks such as those on the build system of SolarWinds' Orion will continue undetected. It is good to harden build systems against attack, but it is unwise to assume that attacks can never succeed. The documentation from reproducible-builds.org can be helpful.
5. The IDA document "State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation 2016" (aka the "Software SOAR") identifies many different types of tools, along with a set of common technical objects, and shows which types of tools tend to better meet different technical objectives. It recommends identifying a projects' technical objectives,

and then selecting tools to meet those objectives. It also provides some suggested tool types for common situations.

6. Note that when software is written in memory-safe languages (and the safeties are left on), many vulnerabilities simply cannot occur. Thus, the way that software is written impacts what tools or analysis may be appropriate.

There are also a number of projects that relate to measuring security and/or broader quality, and thus might make sense in this context:

- Community Health Analytics Open Source Software (CHAOSS) focuses on creating analytics and metrics to help define open source software community health and identify risk.
- The OpenSSF Security Metrics Project, that is in the process of development, was created to collect, aggregate, analyze, and communicate relevant security data about open source projects.
- The OpenSSF Security Reviews initiative provides a collection of security reviews of open source software.
- The OpenSSF Security Scorecards provide a set of automated pass/fail checks to provide a quick review of any open source software.

A key issue is not just what tools are used, but when and how they are used. It is widely accepted that verification tools are best placed with a continuous integration (CI) pipeline so that every proposed change will go through automated evaluation, enabling rapid feedback on proposed changes. Many discussions about DevSecOps emphasize this point. The Continuous Delivery (CD) foundation newsletter, July 2020, has articles about this.

The Defense Industrial Board (DIB) Software Acquisition and Practices (SWAP) Study discusses developing secure software. Its main report (see page 10) notes key best practices that are also very common in open source software build and test workflows. Specifically the report cites, *“Code review tools are reliable and easy to use… Unit test is ubiquitous, fully automated, and integrated into the software review process. Integration, regression, and load testing are also widely used, and these activities should be an integrated, automated part of daily workflow.”* It also notes that it is vital to add *“the integration of security at all stages of development and deployment.”*