CERT/CC Comments on Standards and Guidelines to Enhance Software Supply Chain Security (Questions 2-5)[1]

Corresponding author: Art Manion <amanion@cert.org>

For questions 2, 4, and 5, how will standards and guidelines apply to Free/Libre and Open Source Software (FLOSS)?

2. Secure software development lifecycle practices

These four supplier[2] practices reduce the risks associated with software vulnerabilities and other security issues. These practices can be observed externally, thus supporting "attestation of conformity."

2.1 Coordinated vulnerability disclosure

Suppliers should practice coordinated vulnerability disclosure (CVD). Despite the application of security practices throughout the software development lifecycle (SDL), essentially all software is delivered with as-yet-unknown security vulnerabilities. Therefore, suppliers must be able to respond effectively when vulnerabilities are discovered. One form of CVD is a vulnerability disclosure program (VDP) as set out in §4.e.viii of the E.O.[3] and required by CISA Binding Operational Directive 20-01.[4] A functional CVD or VDP capability requires more than "...reporting and disclosure process[es]," it also includes triage, investigation, analysis, reproduction, fix development, and communication management.[5]

2.2 Secure updates

In many cases, the solution to a vulnerability in fielded software is an update (patch, hotfix, upgrade). Suppliers must be able to securely deliver updates to users. Assuring the authenticity and integrity of updates is critical. As demonstrated by incidents like SolarWinds and NotPetya, adversaries target software delivery and update mechanisms. Centralized update mechanisms also centralize risk. Suppliers and developers should proactively mitigate the risk associated with a single, centralized secure update mechanism. Suppliers should also enable users to control update deployment.

2.3 Supply chain transparency

Nearly all modern software systems depend on other software. Vulnerabilities in upstream dependencies are nearly impossible to identify and track without knowledge of the supply chain. Software composition analysis is a functional "after-the-fact" dependency detection method. A more efficient option is for suppliers to provide software bills of materials (SBOM). An SBOM is a list of software components and their dependencies.[6] Government acquirers and users (including other suppliers) should require SBOMs from their suppliers.

2.4 End of security support

Old software typically accumulates known vulnerabilities that may be fixed in newer releases.[7] Security support may end without users realizing it. Suppliers should provide information about when software components and systems will no longer receive security updates. Either the end-of-support date or the amount of notice the supplier must give before support ends should be provided at the time of acquisition.

[1] https://www.nist.gov/itl/executive-order-improving-nations-cybersecurity/workshop-and-call-position-papers
[2] We use "supplier" throughout as the entity providing a software component, system, or service to the government. A supplier may also be a developer, vendor, maintainer, manufacturer, integrator, or service provider.
[3] https://www.federalregister.gov/d/2021-10460/p-72
[4] https://cyber.dhs.gov/bod/20-01/
[5] https://vuls.cert.org/confluence/display/CVD/4.+Phases+of+CVD
[6] https://www.ntia.gov/sbom
[7] See https://libyear.com/ and https://ericbouwers.github.io/papers/icse15.pdf

3. Federal government use of critical software

There is perhaps too much existing guidance for government administration and operation of software systems.[8] It is unclear if compliance is beneficial even if it is strictly followed.[9] Review and consolidate existing guidance. Consider material from NIST SPs 800-53 and 800-171 (possibly CMMC level 4) as minimum requirements for the use of critical software.[10]

Incidents will occur, so agencies should have incident response capability,[11] either in-house or on contract. Some of this capability involves preparation and basic system administration practices that facilitate incident response. Suppliers sometimes lock users out of basic administrative access to products. License and contract terms may enforce lock-out. Lock-out may inhibit incident response by creating dependency on the supplier to create file system images, decrypt diagnostic reports, access the product remotely, or to perform other incident response tasks. Contracts that allow lock-out should require rapid incident response support, even when many users are responding to incidents at the same time.

Agencies should perform vulnerability management. Asset management is a prerequisite for vulnerability management (see also 2.3). Agencies should be able to notice new vulnerabilities, prioritize responses,[12] and quickly[13] apply updates or other mitigations.

Agencies should carefully consider the benefits and risks of acquiring new software and enabling features.[14] Operating less software and enabling fewer features reduces attack surface.

4. Testing software security

Grey-box fuzzing, which uses program instrumentation to compute code coverage and detect when new areas of the software are exercised, has been shown to be effective at both discovering security vulnerabilities and generating test corpora that exercise a broad range of software functionality.[15] A variety of implementations have shown grey-box fuzzing can be implemented at both the binary and source-code level with relatively low performance overhead.[16]

Secure coding standards for a variety of programming languages exist, as do tools that analyze source code for defects that may cause vulnerabilities.[17] Because different tools and techniques are better at identifying different types of defects, a consolidator tool should be used to merge results, collapse duplicates, and reduce reevaluation of previously reported findings.[18]

5. Software integrity chains and provenance

Comprehensive knowledge of the composition of software systems (baseline SBOM as noted in 2.3) is necessary but insufficient to provide high assurance of software supply chains.[19] Integrity and authentication (part of provenance) almost certainly require digital signatures and their accompanying infrastructure, including strong identification of suppliers and other parties involved in the supply chains. See *Deliver Uncompromised: Securing Critical Software Supply Chains*.[20]

[8] NIST SPs 800-53, 800-61, 800-160, 800-161, 800-171; RMF; CMMC; CSF; FedRAMP; and FIPS to name a few
[9] https://www.ndss-symposium.org/wp-content/uploads/2020/02/24003.pdf
[10] https://www.acq.osd.mil/cmmc/draft.html
[11] We suggest NIST SP 800-61 and the FIRST CSIRT Services Framework: https://www.first.org/standards/frameworks/csirts/csirt_services_framework_v2.1
[12] https://resources.sei.cmu.edu/asset_files/WhitePaper/2021_019_001_653461.pdf
[13] For the 4% of vulnerabilities analyzed that had a public exploit, the median time to public exploit availability was 2 days: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=644720
[14] https://www.consumerreports.org/car-maintenance/does-your-car-need-undercoating/
[15] https://www.fuzzingbook.org/
[16] https://arxiv.org/abs/1812.00140
[17] https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682
[18] Examples include Code DX, ThreadFix, and SCALe.
[19] https://www.ntia.gov/files/ntia/publications/sbom_options_and_decision_points_20210427-1.pdf
[20] https://www.mitre.org/sites/default/files/publications/pr-21-0278-deliver-uncompromised-securing-critical-software-supply-chains.pdf