SYNOPSYS®

Guidelines for software integrity chains and provenance

Software is no longer exclusively created within the proverbial four walls of a vendor. Instead, globally distributed teams of developers create software components and solutions following open source development patterns. Managing open source usage is fundamentally different than managing commercial or contracted software, but even commercial/proprietary or contracted software will incorporate open source components.

Unless response to Executive Order 14028 addresses the complexity of modern distributed software development and provides transparency into both the provenance of code and the associated testing performed at each lifecycle stage, Executive Order 14028 might fail to meet its longer-term objectives. Meeting that desired outcome requires greater transparency into the software development processes employed by all stakeholders, including:

- An understanding of where all code comprising an application originated. For commercial/proprietary and contracted applications, this origin point may be a software vendor, but with the prevalence of open source software and the reality that significant portions of commercial/proprietary and contracted software are of open source origin, how open source software is developed and released must be factored in. Key to this understanding is a recognition that there is no unique repository for most open source components, nor is there a single download location for any given open source component. As an example, the source code for OpenSSL is currently available from almost 7000 forks on GitHub and from numerous vendors each with varying implementations;
- An understanding of how update and patch information is communicated to users of software applications. Where commercial and contracted software suppliers know their customers and can proactively push update information to those customers, creators of open source solutions often have no knowledge of who their users are, or how those users are using the open source solution. The responsibility for obtaining updates or patches for open source components then falls on the user who is expected to actively subscribe to update information and perhaps actively tell all their customers to apply the patch;
- A recognition that procurement of open source components may be as simple as downloading the component and using it – without the benefit of any security reviews, audit trails, or other source code management controls;
- A recognition that open source software is accessible in both a pre-release, or source form, and in a released form, such as with a purchased maintenance agreement, where pre-released software may have limited security testing if any;
- A recognition that malicious actors do attempt to contribute code changes to legitimate open source projects and that such malicious code might remain available for download in a project fork or branch even after the malicious code is removed from the primary repository;
- A recognition that the current CVE scheme isn't designed to communicate the presence of malicious code within a codebase that is available in pre-release form;
- A recognition that the ability to remediate defects, weaknesses, and vulnerabilities in code is a function of how well the source code is understood by the current development team. In effect, as development teams evolve, there will be legacy code within any application and if an exploitable weakness is present in such code, remediation will take longer and potentially introduce additional weaknesses due to a lack of familiarity with such legacy code;

- A recognition that most open source projects lack comprehensive threat monitoring, robust security disclosure and triage processes. While commercial organizations offering open source solutions may employ dedicated security teams, such practices are far from common within the general open source world;
- A recognition that many organizations specifically and purposely ignore open source software during static analysis testing to shorten testing cycles by focusing on the code they created and thus intend to fix. In effect, those organizations view the identification of defects or weaknesses in open source code as implicitly, and potentially subconsciously, the responsibility of the open source community;
- A recognition that while code signing principles as a measure of code provenance solves several problems with commercial/proprietary or contracted software and with compiled binaries; code signing where the source code is freely available for download from a public repository presents a different challenge. A significant majority of open source repositories are based on “git”. git assigns a unique SHA1 identifier for each commit made, and that SHA1 is computed based on both committer and author information meaning that a git identifier could be a proxy for code signing provenance, but only if the source is obtained via a git client and not as a simple download from a repository. Additionally, while the git community is moving towards SHA256 based commit identifiers, that work is ongoing with no indication from public git repositories of their roadmap for implementation of SHA256.

Each of the bullets above highlights how open source software differs from that of commercial/proprietary or contracted software. With commercial/proprietary and contracted software, the onus for responsible development and management practices can be placed on the supplier, but that paradigm is reversed when open source software is used. Proper usage of open source software obligates the consumer of that software to either implement robust controls to validate its suitability to a specific purpose, or to contract with an entity that will proxy that responsibility and become an official suppler for specific open source components. Even when such a proxy supplier is used, there needs to be a process to ensure that individual components subject to that proxy relationship are not replaced with versions accessible from a public open source repository.

Verifiable composition and process integrity through standardized machine-readable SBoM-based supply chain metadata, including open source software risk scoring, is required.

- Software Bills of Material (SBoMs) must provide enough details to convey provenance, pedigree, and linkage to describe how software is connected together, along with attestations about steps in the production chain, including security testing stages.
- Open source software risk scoring should be required, and as an exemplar, it could be adapted from OpenSSF's CII Best Practices badge project:
    - Community Health Analytics Open Source Software (CHAOSS) that focuses on creating analytics and metrics to help define community health and identify risk
    - The OpenSSF Security Metrics Project, to collect, aggregate, analyze, and communicate relevant security data about open source projects.
    - The OpenSSF Security Reviews to provide a collection of security reviews of open-source software.
    - The OpenSSF Security Scorecards to provide a set of automated pass/fail checks to provide a quick review of arbitrary OSS.