Concise Guide for Developing More Secure Software

by the Open Source Security Foundation (OpenSSF) Best Practices Working Group, 2023-06-14

Here is a concise guide for all software developers for secure software development, building, and distribution. All tools or services listed are merely examples.

1. **Ensure all privileged developers use multi-factor authentication (MFA) tokens**. This includes those with commit or accept privileges. MFA hinders attackers from "taking over" these accounts.
2. **Learn about secure software development.** Take, e.g., the free OpenSSF course or the hands-on Security Knowledge Framework course. SAFECode's Fundamental Practices for Secure Software Development provides a helpful summary.
3. **Use a combination of tools in your CI pipeline to detect vulnerabilities**. See the OpenSSF guide to security tools. Tools shouldn't be the *only* mechanism, but they scale.
4. **Evaluate software before selecting it as a direct dependency**. Only add it if needed, evaluate it (see Concise Guide for Evaluating Open Source Software, double-check its name (to counter typosquatting), and ensure it's retrieved from the correct repository.
5. **Use package managers**. Use package managers (system, language-level, and/or container-level) to automatically manage dependencies and enable rapid updates.
6. **Implement automated tests**. Include negative tests (tests that what shouldn't happen doesn't happen) and ensure the test suite is thorough enough to "ship if it passes the tests".
7. **Monitor known vulnerabilities in your software's direct & indirect dependencies**. E.g., enable basic scanning via GitHub's dependabot or GitLab dependency scanning. Many other third party Software Composition Analysis (SCA) tools are also available. Quickly update vulnerable dependencies.
8. **Keep dependencies reasonably up-to-date**. Otherwise, it's hard to update for vulnerabilities.
9. **Do not push secrets to a repository**. Use tools to detect pushing secrets to a repository.
10. **Review before accepting changes**. Enforce it, e.g., GitHub or GitLab protected branches.
11. **Prominently document how to report vulnerabilities & prepare for them.**
    - Use resources like the Guide to coordinated vulnerability disclosure.
    - Explicitly disclose security issues affecting vendored dependencies.
    - Create a security policy. Provide contacts.
12. **Make it easy for your users to update**. Implement stable APIs, e.g., support old names when new ones are added. Use semantic versioning. Have a deprecation process.
13. **Sign your project's important releases**. Use standard tools and signing formats for your distribution. See the cosign tool from the sigstore project to sign containers and other artifacts.
14. Earn an OpenSSF Best Practices badge for your open source project. At least earn "passing". Plan and roadmap to eventually earn silver & gold.

15. **Improve your OpenSSF Scorecards score (if OSS and on GitHub)**. You can read the Scorecards checks. Use the Allstar monitor.
16. **Notify the community of vulnerabilities in your project.** Publish security advisories with accurate & precise information, e.g., what usage & versions are vulnerable, mitigations, and fixed version(s). Get a CVE ID. On GitHub, create your security advisory & request a CVE.
17. **Improve your Supply chain Levels for Software Artifacts (SLSA) level**. This hardens the integrity of your build and distribution process against attacks.
18. **Publish and consume a software bill of materials (SBOM)**. This lets users verify inventory, id known vulnerabilities, & id potential legal issues. Consider SPDX or CycloneDX.
19. **Onboard your project into LFX Security if you manage a Linux Foundation project**.
20. **Apply the CNCF Security TAG Software Supply Chain Best Practices guide**.
21. **Implement ASVS and follow relevant cheatsheets**.
22. **Apply SAFECode's Fundamental Practices for Secure Software Development**.
23. **Complete a third-party security code review/audit**. Expect this to be USD$50K or more.
24. **Continuously improve**. Improve scores, look for tips, & apply as appropriate.
25. **Manage succession**. Have clear governance & work to add active, trustworthy maintainer(s).
26. **Prefer memory-safe languages**. Many vulnerabilities involve memory safety. Where practical, use memory-safe programming languages (most are) and keep memory safety enabled. Otherwise, use mechanisms like extra tools and peer review to reduce risk.

Welcome suggestions and updates! Please open an issue or post a pull request.
