digicert

Sample Enterprise Code Signing Policy

Updated August 16, 2024

1. Overview

Multiple groups within modern organizations publish software for internal and external use. To help protect against software tampering and confirm the authenticity of software releases, best practices state that software binaries should signed with cryptographic keys. However, trust in these signatures requires appropriate controls and monitoring of code signing processes.

2. Purpose

This policy outlines the requirements and procedures for controlling software code signing processes to ensure that only authorized and verified code is signed and distributed, enhancing the security and integrity of the software.

This policy will ensure the organization[1] knows what it is publishing, and that the software published does not introduce unacceptable security risks. It will also allow users to verify the integrity of the files to demonstrate that they have not been tampered with. Conversely, these procedures will also allow users to demonstrate that software falsely purported to be created by the organization was not.

Rigorous code signing with good record keeping can also facilitate the work of auditors, both internal and external.

3. Scope

This policy applies to all personnel, processes, and systems involved in the code signing process within the organization, including internal and external developers, signers, reviewers, and auditors. It does not apply to third-party software[2] the organization uses (e.g., productivity tools like Microsoft Office or development tools like JFrog).

4. Policy

4.1 Authentication

Users accessing the software development environment must use approved authentication services or methods. They must not use shared accounts, insecurely stored credentials, local accounts, or other means of authentication that violate the enterprise authentication policy.

4.2 Define Roles[3]

Under best practices, the organization should define separate roles[4] for code signing activities. These roles should minimally include Submitter, Signer, and Security Officer and expand to include Reviewer, Auditor, and System Administrator.

---

1. As explained in Section 3 on scope, where appropriate, these policies are designed with large and distributed groups in mind. Most are still practical and remain best practices for smaller teams and individual developers.
2. Any respectable third-party software will be digitally signed by its own creator. Other policies should ensure that only software signed by the proper creator is used in the organization.
3. Sections 4.2 (Roles) and 4.3 (Segregation of Duties) describe what is generically known as Role-Based Access Control (RBAC), a best practice in IT security. These policies apply RBAC specifically in the code signing context. As the definition of RBAC by NIST (the National Institute of Standards and Technology) states, many regulatory compliance standards (such as NIST SP 800-95 and NIST SP 800-53) rely on the proper implementation of RBAC.
4. These suggested roles are designed for a large, potentially distributed team. In the real world, many teams are small, and even a single developer can still follow best practices to some degree. Adapt the role definitions and assignments to your own circumstances. It may make sense to define many roles, even if they all have only one or two users, as it eases the process of adding others, such as a consultant.

4.3 Segregation of Duties

Users may have multiple roles[5] based on commercially reasonable practices. However, no single user[6] may have control over a process that introduces a single point of failure for security.

4.4 Criticality of Software[7]

The criticality of each software product or project must be assessed based on the potential damage if the software contained malicious code or exploitable vulnerabilities that would result in unauthorized access to data, loss of system functionality, or other failure.

Sufficiently critical software may justify a requirement of more than one authorization for signing.[8]

4.4.1 Limit Access to Critical Resources

All access, but especially access to privileged resources, like build servers and administration of HSMs, should be assigned based on the defined roles and not specific users (or unrelated groups like Administrator or C-Suite).

4.5 Security Reviews

Software source code must pass a security review according to the secure Software Development Life Cycle (SDLC) policy.[9]

Each release and associated component must be scanned for malicious indicators and known vulnerabilities.[10] Do not sign software if it contains confirmed malware or an exploitable vulnerability.

4.6 Key Management

Unique code signing keys and certificates must be used for non-production and production environments.[11] Test environments that are remotely accessible and process sensitive data must be treated as production environments.

Production code signing keys must be generated and stored securely[12] using a hardware security module compliant with FIPS 140-2 Level 2 or Common Criteria EAL 4+ and used according to enterprise Cryptographic Management and Certificate policies.

Multiple users must not share production code signing keys unless activity logging is identifiable to a unique user

---

5. See note 4 on definitions of roles.
6. Pursuant to the Segregation of Duties, any particular role should, if possible, be assigned to more than one user.
7. NIST provides many definitions and supplementary materials related to the criticality of software. While you must examine your own applications one by one to determine their criticality, the term has a legal meaning since President Biden issued Executive Order 14028 in May 2021, directing measures to ensure "the security and integrity of 'critical software'." The directive applies to government agencies and private actors that contract with them.
8. Many certificate management products provide a mechanism to require approval from two or more users to authorize certain critical operations, such as keypair export, keypair deletion, and certificate revocation.
9. See OWASP for assistance in developing a secure SDLC and other useful security policies.
10. Malware and vulnerable software are as unacceptable in testing and other non-production environments as in production. For this reason, the development cycle, or CI/CD, should scan for these problems. Not all vulnerabilities are worth aborting a build or even a code shipment. Many have low severity, and others are difficult or impossible to exploit. You need to decide a threshold for "showstopper" vulnerabilities. If malware or a showstopper is found in a scan, the CI/CD should exit before executing the test code and report the problems.
11. The required number of keys and certificates follows a one-to-one ratio. At a minimum, you need one for production and one for non-production. Theoretically, you could generate a new certificate for every build, and some do this. This procedure has the advantage of limiting the exposure caused by a compromised key.
12. Where should you generate, store, and use keys for non-production code? Ideally, you would also want to use an HSM compliant with FIPS 140-2 Level 2 or Common Criteria EAL 4+. Still, less expensive measures, such as a software-based secrets management system, are preferred. Hazardous practices like storing keys on insecure file systems or in source code are never acceptable.

4.6.1 Maintain an inventory of ALL code signing certificates. Monitor expiration dates and generate new certificates proactively.[13]

4.6.2 Keys must be regularly rotated, and certificates must be revoked and reissued as needed and based on the frequency and/or expected lifespan of software releases.[14]

4.7 Incident Response[15]

Define and, to the extent possible, automate procedures for documenting, revoking, and reissuing compromised keys.

4.8 Code Signing Process

Best practice is to automate the code signing process as part of the CI/CD pipeline.[16] Use an approved Certificate Authority (CA) for code signing certificates. Ideally, developers should use a unique signing key to sign any code they check-in.[17] The individual signatures must be verified before signing a release.

Releases that will be valid after the code signing certificate has expired must include a time stamp from an approved timestamping service.

4.9 Software Bill of Materials (SBOM)

Release artifacts, including the software code branch, SBOM, and Vulnerability Exchange Documents, must be signed to help detect tampering with assertions and archives.

4.10 Auditing and Logging

Maintain a comprehensive audit trail, including proof of code reviews, security scans, and code signing key action. Activities must be associated with unique users and service accounts.

5 Review and Updates

This policy will be reviewed annually and updated as necessary to ensure its effectiveness and alignment with industry standards and organizational changes.

For an editable version of this document, please click here.

13. In practice, in all but very small organizations, inventory and certificate management can only be done effectively using a Certificate Lifecycle Management (CLM) system that automates the process. If you are using a minimal number of certificates and keys, such as one for testing and one for production, ensure they are backed up in some secure location.
14. See note 13 on automation through a CLM system
15. The incident would be the compromise or potential compromise of a signing key. After following your procedure for documenting, revoking, and reissuing the compromised keys, you will need to resign and redeploy the code.
16. Organizations that automate the process can exploit the security advantages of frequently rotating keys and certificates. Without automation, a "lazy" key rotation policy may be the most practical approach. For production releases, consider the product's lifespan when planning the signing certificate's lifespan.
17. In many cases, it is best practice to sign every artifact with different keys. For example, if a key is compromised, it is best when you need to invalidate only one file rather than all the files in the release.