Code Signing Best Practices

Code Signing Best Practices

The biggest issue with code signing is the protection of the private signing key associated with the code signing certificate. If a key is compromised, the certificate loses trust and value, jeopardizing the software that you have already signed.

Consider the following code signing best practices:

1. **Minimize access to private keys**
   - Allow minimal connections to computers with keys
   - Minimize the number of users who have key access
   - Use physical security controls to reduce access to keys

2. **Protect private keys with cryptographic hardware products**
   - Cryptographic hardware does not allow export of the private key to software where it could be attacked
   - Use a FIPS 140 Level 2-certified product (or better)
   - If private keys will be transported, ensure the cryptographic hardware is protected with a randomly generated password of at least 16 characters which contains uppercase letters, lowercase letters, numbers and special characters

3. **Time-stamp code**
   - Time-stamping allows code to be verified after the certificate has expired or been revoked
   - Time-stamp certificates can be issued for a maximum of 135 months which can support the signed software to be validated for up to 11 years

4. **Understand the difference between test-signing and release-signing**
   - Test-signing private keys and certificates requires less security access controls than production code signing private keys and certificates
   - Test-signing certificates can be self-signed or come from an internal test CA
   - Test certificates must chain to a completely different root certificate than the root certificate that is used to sign publicly released products; this precaution helps ensure that test certificates are trusted only within the intended test environment
   - Establish a separate test code signing infrastructure to test-sign pre-release builds of software

5. **Authenticate code to be signed**
    - Any code that is submitted for signing should be strongly authenticated before it is signed and released
    - Implement a code signing submission and approval process to prevent the signing of unapproved or malicious code
    - Log all code signing activities for auditing and/or incident-response purposes

6. **Virus scan code before signing**
    - Code signing does not confirm the safety or quality of the code; it confirms the publisher and whether or not the code has been changed
    - Take care when incorporating code from other sources
    - Implement virus-scanning to help improve the quality of the released code

7. **Do not over-use any one key (distribute risk with multiple certificates)**
    - If code is found with a security flaw, then publishers may want to prompt a User Account Control dialogue box to appear when the code is installed in the future; this can be done by revoking the code signing certificate so a revoked prompt will occur
    - If the code with the security flaw was issued before more good code was issued, then revoking the certificate will impact the good code as well
    - Changing keys and certificates often will help to avoid this conflict

8. **Revoking compromised certificates**
    - Report key compromise or signed malware to your certification authority
    - Compromised keys or signed malware of suspect code will require the code signing certificate to be revoked
    - Assuming that all signed code has been time-stamped, then the revocation date can be selected before the time of compromise. This will mean that code signed before the revocation date may not be impacted.

References

The following references will support Windows and Java code signing.

- Guide to Code Signing for Authenticode
- Guide to Code Signing for Java
- Guide to Code Signing for Windows Macros & Visual Basic
- Guide to Code Signing and EV Code Signing Certificate for Download and Installation