ENV00-J. Do not sign code that performs only unprivileged operations

Java uses code signing as a requirement for granting elevated privileges to code. Many security policies permit signed code to operate with elevated privileges. For example, Java applets can escape the default sandbox restrictions when signed. Consequently, users can grant explicit permissions either to a particular codebase or to all code signed by a particular signer. This approach places control of security in the hands of the user, who can choose whether to run an application with full or restricted permissions.

Signing code, however, has its own problems. According to Bruce Schneier [Schneier 2000]:

> *First, users have no idea how to decide if a particular signer is trusted or not. Second, just because a component is signed doesn't mean that it is safe. Third, just because two components are individually signed does not mean that using them together is safe; lots of accidental harmful interactions can be exploited. Fourth, "safe" is not an all-or-nothing thing; there are degrees of safety. And fifth, the fact that the evidence of attack (the signature on the code) is stored on the computer under attack is mostly useless: The attacker could delete or modify the signature during the attack, or simply reformat the drive where the signature is stored.*

Code signing is designed to authenticate the origin of the code as well as to verify the integrity of the code. It relies on a certification authority (CA) to confirm the identity of the principal signer. Naive users should not be expected to understand how certificates and the public key infrastructure (PKI) work.

Users commonly associate digital signatures with safety of code execution, trusting the code to cause them no harm. The problem arises when a vulnerability is discovered in signed code. Because many systems are configured to permanently trust certain signing organizations, those systems fail to notify their users when downloading content signed by the trusted organization, even when that content contains vulnerabilities. An attacker can offer the users legitimately signed vulnerable content with the intention of exploiting that content.

Consider, for example, signed Java applets. When a certificate is verified, on widely used platforms, the user is presented with a security dialog in which the option "Always trust the content from the publisher" is selected by default. The dialog primarily asks whether or not the signed code should be executed. Unfortunately, if the user confirms the dialog with the check box selected, the "Always trust..." setting overrides any future warning dialogs. An attacker can take advantage of this mechanism by exploiting vulnerable code signed by the trusted organization. In this case, the code will execute with the user's implied permission and can be freely exploited.

An organization that signs its own code should not vouch for code acquired from a third party without carefully auditing the third-party code. When signing privileged code, ensure that all of the signed code is confined to a single JAR file (see ENV01-J. Place all security-sensitive code in a single JAR and sign and seal it for more information) and also that any code invoked from the privileged code is also contained in that JAR file. Nonprivileged code must be left unsigned, restricting it to the sandbox. For example, unsigned applets and Java Network Launching Protocol (JNLP) applications are granted the minimum set of privileges and are restricted to the sandbox. Finally, never sign any code that is incomprehensible or unaudited.

Exceptions

ENV00-J-EX1: An organization that has an internal PKI and uses code signing for internal development activities (such as facilitating code check-in and tracking developer activity) may sign unprivileged code. This codebase should not be carried forward to a production environment. The keys used for internal signing must be distinct from those used to sign externally available code.

ENV00-J-EX2: Oracle has deprecated the use of unsigned applets and will soon cease to support them. Applets that are signed have traditionally been run with full privileges. Since Java 1.7.0 update 21, Oracle has provided mechanisms to allow applets to be signed and yet run without full permissions. This enables applets that are today unsigned to continue to run in a security sandbox despite being signed. Signing an applet that runs with restricted privileges under versions of Java at least as recent as update 21 constitutes an exception to this rule. For more information, see Signed Java Applet Security Improvements on the CERT/CC blog.

Risk Assessment

Signing unprivileged code violates the principle of least privilege because it can circumvent security restrictions defined by the security policies of applets and JNLP applications, for example.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
ENV00-J	High	Probable	Medium	**P12**	**L1**

Automated Detection

Detecting code that should be considered privileged or sensitive requires programmer assistance. Given identified privileged code as a starting point, automated tools could compute the closure of all code that can be invoked from that point. Such a tool could plausibly determine whether a body of signed code both includes that entire closure and excludes all other code.

Related Guidelines

ISO/IEC TR 24772:2010	Adherence to least privilege [XYN]

Android Implementation Details

The Android system uses code signing as a means to identify the author of an application and establish trust relationships between applications, not as a means to grant elevated privileges to code.

Bibliography

[Dormann 2008]	
[McGraw 1999]	Appendix C, "Sign Only Privileged Code"
[Schneier 2000]