SPC-0 · Mobile Threat Catalogue

Archived: 2026-04-05 17:03:05 UTC

Mobile Threat Catalogue

Malicious Code in Open Source Software

Contribute

Threat Category: Supply Chain

ID: SPC-0

Threat Description: An adversary with access to open source code and knowledge of its particular use for the system being acquired can insert malicious code into open source software used for libraries.[1]

Threat Origin

Supply Chain Attack Framework and Attack Patterns [1]

Exploit Examples

CVE Examples

Possible Countermeasures

Mobile App Developer

To increase the complexity of this attack, prefer open source software libraries for which integrity-checking mechanisms are provided (e.g., strong cryptographic hashes of source files, digital signatures) so the authenticity of the open source library can be verified.

To increase the complexity of this attack, when possible, obtain multiple instances of the same library as hosted by various sources (e.g., FTP mirrors) from which it should be available. Then evaluate all obtained versions for consistency (e.g., compare strong hashes). If any discrepancies are detected, contact the open source software developer.

To detect compromise of the integrity checking mechanisms of a given source of open source libraries, particularly for security sensitive library functions, such as math or cryptographic libraries, contact the developer to verify the library is authentic.

To reduce the probability this variety of attack goes undetected at runtime, implement defensive programming. Any call to untrusted code that can impact critical functionality of the system should include checks on the output for conditions that should always be true given an assumption the library behaves as expected.

To protect open source library used by a product from modification, then if possible, package a verified authentic instance of the open source library and apply cryptographic protections (e.g., strong hashing, digital signatures) to the product to allow customers to verify the authenticity and integrity of all packaged components.

To prevent distributing a software package that contains maliciously modified open source libraries, perform sufficient functional testing of the complete system to verify that it exhibits correct and consistent behavior.

Application Developer

To increase the complexity of this attack, prefer open source software libraries for which integrity-checking mechanisms are provided (e.g., strong cryptographic hashes of source files, digital signatures) so the authenticity of the open source library can be verified.

To increase the complexity of this attack, when possible, obtain multiple instances of the same library as hosted by various sources (e.g., FTP mirrors) from which it should be available. Then evaluate all obtained versions for consistency (e.g., compare strong hashes). If any discrepancies are detected, contact the open source software developer.

To detect compromise of the integrity checking mechanisms of a given source of open source libraries, particularly for security sensitive library functions, such as math or cryptographic libraries, contact the developer to verify the library is authentic.

To reduce the probability this variety of attack goes undetected at runtime, implement defensive programming. Any call to untrusted code that can impact critical functionality of the system should include checks on the output for conditions that should always be true given an assumption the library behaves as expected.

To reduce the probability this variety of attack goes undetected at runtime, implement defensive programming. Any call to untrusted code that can impact critical functionality of the system should include checks on the output for conditions that should always be true given an assumption the library behaves as expected.

To protect open source library used by a product from modification, then if possible, package a verified authentic instance of the open source library and apply cryptographic protections (e.g., strong hashing, digital signatures) to the product to allow customers to verify the authenticity and integrity of all packaged components.

To prevent distributing a software package that contains maliciously modified open source libraries, perform sufficient functional testing of the complete system to verify that it exhibits correct and consistent behavior.

Enterprise

To increase the complexity of this attack, prefer open source software libraries for which integrity-checking mechanisms are provided (e.g., strong cryptographic hashes of source files, digital signatures) so the authenticity of the open source library can be verified.

To increase the complexity of this attack, when possible, obtain multiple instances of the same library as hosted by various sources (e.g., FTP mirrors) from which it should be available. Then evaluate all obtained versions for consistency (e.g., compare strong hashes). If any discrepancies are detected, contact the open source software developer.

To detect compromise of the integrity checking mechanisms of a given source of open source libraries, particularly for security sensitive library functions, such as math or cryptographic libraries, contact the developer to verify the library is authentic.

To protect open source library used by a product from modification, then if possible, package a verified authentic instance of the open source library and apply cryptographic protections (e.g., strong hashing, digital signatures) to the product to allow customers to verify the authenticity and integrity of all packaged components.

To prevent distributing a software package that contains maliciously modified open source libraries, perform sufficient functional testing of the complete system to verify that it exhibits correct and consistent behavior.

To prevent executing an application that relies upon a maliciously modified version of an open source library that is loaded dynamically at runtime (e.g., Dynamic Linked Library), perform verification of the library file prior to execution. This may involve validating hashes, verifying digital signatures, or other integrity protection or detection mechanisms on the host system.

References
