JNI01-J. Safely invoke standard APIs that perform tasks using the immediate caller's class loader instance (loadLibrary)

(THIS CODING RULE OR GUIDELINE IS UNDER CONSTRUCTION)

Many static methods in standard Java APIs vary their behavior according to the immediate caller's class. Such methods are considered to be caller-sensitive. For example, the `java.lang.System.loadLibrary(library)` method uses the immediate caller's class loader to find and dynamically load the specified library containing native method definitions. Because native code bypasses all of the security checks enforced by the Java Runtime Environment and other built-in protections provided by the Java virtual machine, only trusted code should be allowed to load native libraries. None of the loadLibrary methods in the standard APIs should be invoked on behalf of untrusted code since untrusted code may not have the necessary permissions to load the same libraries using its own class loader instance [Oracle 2014].

Noncompliant Code Example

In this noncompliant example, the Trusted class has permission to load libraries while the Untrusted class does not. However, the Trusted class provides a library loading service through a public method thus allowing the Untrusted class to load any libraries it desires.

```
// Trusted.java

import java.security.*;

public class Trusted {

   public static void loadLibrary(final String library){
      AccessController.doPrivileged(new PrivilegedAction<Void>() {
         public Void run() {
             System.loadLibrary(library);
             return null;
         }
      });
   }
}

---------------------------------------------------------------------------------

// Untrusted.java

public class Untrusted {

   private native void nativeOperation();

   public static void main(String[] args) {
      String library = new String("NativeMethodLib");
      Trusted.loadLibrary(library);
      new Untrusted.nativeOperation();  // invoke the native method
   }
}
```

Compliant Solution

In this compliant example, the Trusted class loads any necessary native libraries during initialization and then provides access through public native method wrappers. These wrappers perform the necessary security checks and data validation to ensure that untrusted code cannot exploit the native methods (see JNI00-J. Define wrappers around native methods) .

```
 // Trusted.java

import java.security.*;

public class Trusted {

   // load native libraries
   static{
      System.loadLibrary("NativeMethodLib1");
          System.loadLibrary("NativeMethodLib2");
          ...
   }

   // private native methods
   private native void nativeOperation1(byte[] data, int offset, int len);
   private native void nativeOperation2(...)
   ...

   // wrapper methods perform SecurityManager and input validation checks
   public void doOperation1(byte[] data, int offset, int len) {
      // permission needed to invoke native method
      securityManagerCheck();

      if (data == null) {
         throw new NullPointerException();
      }

      // copy mutable input
      data = data.clone();

      // validate input
      if ((offset < 0) || (len < 0) || (offset > (data.length - len))) {
         throw new IllegalArgumentException();
      }

      nativeOperation1(data, offset, len);
   }

   public void doOperation2(...){
      ...
   }
}
```

Exceptions

Risk Assessment

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
JNI01-J	high	likely	low	P27	L1

Automated Detection

Detecting calls, such as `java.lang.System.loadLibrary()`, that perform tasks using the immediate caller's class loader can be detected automatically. Determining whether the use of these calls is safe cannot be done automatically.

Tool	Version	Checker	Description
Parasoft Jtest	9.5	**BD.SECURITY.TDLIB**	Implemented

Related Guidelines

MITRE CWE	CWE-111. Direct use of unsafe JNI
Secure Coding Guidelines for Java SE, Version 5.0	Guideline 9-9. Safely invoke standard APIs that perform tasks using the immediate caller's class loader instance

Bibliography

[Oracle 2014]	