SEC00-J. Do not allow privileged blocks to leak sensitive information across a trust boundary

The `java.security.AccessController` class is part of Java's security mechanism; it is responsible for enforcing the applicable security policy. This class's static `doPrivileged()` method executes a code block with a relaxed security policy. The `doPrivileged()` method stops permissions from being checked further down the call chain.

Consequently, any method that invokes `doPrivileged()` must assume responsibility for enforcing its own security on the code block supplied to `doPrivileged()`. Likewise, code in the `doPrivileged()` method must not leak sensitive information or capabilities.

For example, suppose that a web application must maintain a sensitive password file for a web service and also must run untrusted code. The application could then enforce a security policy preventing the majority of its own code—as well as all untrusted code—from accessing the sensitive file. Because it must also provide mechanisms for adding and changing passwords, it can call the `doPrivileged()` method to temporarily allow untrusted code to access the sensitive file. In this case, any privileged block must prevent all information about passwords from being accessible to untrusted code.

Noncompliant Code Example

In this noncompliant code example, the `doPrivileged()` method is called from the `openPasswordFile()` method. The `openPasswordFile()` method is privileged and returns a `FileInputStream` for the sensitive password file. Because the method is public, it could be invoked by an untrusted caller.

```
public class PasswordManager {

  public static void changePassword() throws FileNotFoundException {
    FileInputStream fin = openPasswordFile();

    // Test old password with password in file contents; change password,
    // then close the password file
  }

  public static FileInputStream openPasswordFile()
      throws FileNotFoundException {
    final String password_file = "password";
    FileInputStream fin = null;
    try {
      fin = AccessController.doPrivileged(
        new PrivilegedExceptionAction<FileInputStream>() {
          public FileInputStream run() throws FileNotFoundException {
            // Sensitive action; can't be done outside privileged block
            FileInputStream in = new FileInputStream(password_file);
            return in;
          }
      });
    } catch (PrivilegedActionException x) {
      Exception cause = x.getException();
      if (cause instanceof FileNotFoundException) {
        throw (FileNotFoundException) cause;
      } else {
        throw new Error("Unexpected exception type", cause);
      }
    }
    return fin;
  }
}
```

Compliant Solution

In general, when any method containing a privileged block exposes a field (such as an object reference) beyond its own boundary, it becomes trivial for untrusted callers to exploit the program. This compliant solution mitigates the vulnerability by declaring `openPasswordFile()` to be private. Consequently, an untrusted caller can call `changePassword()` but cannot directly invoke the `openPasswordFile()` method.

```
public class PasswordManager {
  public static void changePassword() throws FileNotFoundException {
    // ...
  }

  private static FileInputStream openPasswordFile()
     throws FileNotFoundException {
    // ...
  }
}
```

Compliant Solution (Hiding Exceptions)

The previous noncompliant code example and the previous compliant solution throw a `FileNotFoundException` when the password file is missing. If the existence of the password file is itself considered sensitive information, this exception also must be prevented from leaking outside the trusted code.

This compliant solution suppresses the exception, leaving the array to contain a single null value to indicate that the file does not exist. It uses the simpler `PrivilegedAction` class rather than `PrivilegedExceptionAction` to prevent exceptions from propagating out of the `doPrivileged()` block. The `Void` return type is recommended for privileged actions that do not return any value.

```
class PasswordManager {

  public static void changePassword() {
    FileInputStream fin = openPasswordFile();
    if (fin == null) {
      // No password file; handle error
    }

    // Test old password with password in file contents; change password
  }

  private static FileInputStream openPasswordFile() {
    final String password_file = "password";
    final FileInputStream fin[] = { null };
    AccessController.doPrivileged(new PrivilegedAction<Void>() {
        public Void run() {
          try {
            // Sensitive action; can't be done outside
            // doPrivileged() block
            fin[0] = new FileInputStream(password_file);
          } catch (FileNotFoundException x) {
            // Report to handler
          }
          return null;
        }
    });
    return fin[0];
  }
}
```

Risk Assessment

Returning references to sensitive resources from within a `doPrivileged()` block can break encapsulation and confinement and can leak capabilities. Any caller who can invoke the privileged code directly and obtain a reference to a sensitive resource or field can maliciously modify its elements.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
SEC00-J	Medium	Likely	High	P6	L2

Automated Detection

Identifying sensitive information requires assistance from the programmer; fully automated identification of sensitive information is beyond the current state of the art.

Assuming user-provided tagging of sensitive information, escape analysis could be performed on the `doPrivileged()` blocks to prove that nothing sensitive leaks out from them. Methods similar to those used in thread-role analysis could be used to identify the methods that must, or must not, be called from `doPrivileged()` blocks.

Related Guidelines

MITRE CWE	CWE-266, Incorrect Privilege Assignment CWE-272, Least Privilege Violation
Secure Coding Guidelines for Java SE, Version 5.0	Guideline 9-3 / ACCESS-3: Safely invoke `java.security.AccessController.doPrivileged`

Android Implementation Details

The `java.security` package exists on Android for compatibility purposes only, and it should not be used.

Bibliography

[API 2014]	Method `doPrivileged()`
[Gong 2003]	Section 6.4, "`AccessController`" Section 9.5, "Privileged Code"