SEC55-J. Ensure that security-sensitive methods are called with validated arguments

Application code that calls security-sensitive methods must validate the arguments being passed to the methods. In particular, `null` values may be interpreted as benign by certain security-sensitive methods but may override default settings. Although security-sensitive methods should be coded defensively, the client code must validate arguments that the method might otherwise accept as valid. Failure to do so can result in privilege escalation and execution of arbitrary code.

Noncompliant Code Example

This noncompliant code example shows the two-argument `doPrivileged()` method that takes an access control context as the second argument. This code restores privileges from a previously saved context.

```
AccessController.doPrivileged(
  new PrivilegedAction<Void>() {
    public Void run() {
      // ...
    }
  }, accessControlContext);
```

When passed a null access control context, the two-argument `doPrivileged()` method fails to reduce the current privileges to those of the previously saved context. Consequently, this code can grant excess privileges when the `accessControlContext` argument is null. Programmer who intend to call `AccessController.doPrivileged()` with a null access control context should explicitly pass the `null` constant or use the one-argument version of `AccessController.doPrivileged()`.

Compliant Solution

This compliant solution prevents granting of excess privileges by ensuring that `accessControlContext` is non-null:

```
if (accessControlContext == null) {
  throw new SecurityException("Missing AccessControlContext");
}
AccessController.doPrivileged(
  new PrivilegedAction<Void>() {
    public Void run() {
      // ...
    }
  }, accessControlContext);
```

Applicability

Security-sensitive methods must be thoroughly understood and their parameters validated to prevent corner cases with unexpected argument values (such as null arguments). If unexpected argument values are passed to security-sensitive methods, arbitrary code execution becomes possible, and privilege escalation becomes likely.

Bibliography

[API 2013]	`AccessController.doPrivileged()`, `System.setSecurityManager()`