SEC03-J. Do not load trusted classes after allowing untrusted code to load arbitrary classes

The Java classes used by a program are not necessarily loaded upon program startup. Many Java Virtual Machines (JVMs) load classes only when they need them.

If untrusted code is permitted to load classes, it may possess the ability to load a malicious class. This is a class that shares a fully-qualified name with a benign class that is required by trusted code. When the trusted code tries to load its benign class, the JVM provides it with the malicious class instead. As a result, if a program permits untrusted code to load classes, it must first *preload* any benign classes it needs. Once loaded, these benign classes cannot be replaced by untrusted code.

Noncompliant Code Example (Tomcat)

This noncompliant code example shows a vulnerability present in several versions of the Tomcat HTTP web server (fixed in version 6.0.20) that allows untrusted web applications to override the default XML parser used by the system to process `web.xml`, `context.xml` and tag library descriptor (TLD) files of other web applications deployed on the Tomcat instance. Consequently, untrusted web applications that install a parser could view and/or alter these files under certain circumstances.

The noncompliant code example shows the code associated with initialization of a new `Digester` instance in the `org.apache.catalina.startup.ContextConfig` class. "A `Digester` processes an XML input stream by matching a series of element nesting patterns to execute Rules that have been added prior to the start of parsing" [Tomcat 2009]. The code to initialize the `Digester` follows:

```
protected static Digester webDigester = null;

if (webDigester == null) {
  webDigester = createWebDigester();
}
```

The `createWebDigester()` method is responsible for creating the `Digester`. This method calls `createWebXMLDigester()`, which invokes the method `DigesterFactory.newDigester()`. This method creates the new digester instance and sets a `boolean` flag `useContextClassLoader` to `true`.

```
// This method exists in the class DigesterFactory and is called by
// ContextConfig.createWebXmlDigester().
// which is in turn called by ContextConfig.createWebDigester()
// webDigester finally contains the value of digester defined
// in this method.
public static Digester newDigester(boolean xmlValidation,
                                   boolean xmlNamespaceAware,
                                   RuleSet rule) {
  Digester digester = new Digester();
  // ...
  digester.setUseContextClassLoader(true);
  // ...
  return digester;
}
```

The `useContextClassLoader` flag is used by `Digester` to decide which `ClassLoader` to use when loading new classes. When true, it uses the `WebappClassLoader`, which is untrusted because it loads whatever classes are requested by various web applications.

```
public ClassLoader getClassLoader() {
  // ...
  if (this.useContextClassLoader) {
    // Uses the context class loader which was previously set
    // to the WebappClassLoader
    ClassLoader classLoader =
        Thread.currentThread().getContextClassLoader();
  }
  return classloader;
}
```

The `Digester.getParser()` method is subsequently called by Tomcat to process `web.xml` and other files:

```
// Digester.getParser() calls this method. It is defined in class Digester
public SAXParserFactory getFactory() {
  if (factory == null) {
    factory = SAXParserFactory.newInstance(); // Uses WebappClassLoader
    // ...
  }
  return (factory);
}
```

The underlying problem is that the `newInstance()` method is being invoked on behalf of a web application's class loader, the `WebappClassLoader`, and it loads classes before Tomcat has loaded all the classes it needs. If a web application has loaded its own Trojan `javax.xml.parsers.SAXParserFactory`, when Tomcat tries to access a `SAXParserFactory`, it accesses the Trojan `SaxParserFactory` installed by the web application rather than the standard Java `SAXParserFactory` that Tomcat depends on.

Note that the `Class.newInstance()` method requires the class to contain a no-argument constructor. If this requirement is not satisfied, a runtime exception results, which indirectly prevents a security breach.

Compliant Solution (Tomcat)

In this compliant solution, Tomcat initializes the `SAXParserFactory` when it creates the `Digester`. This guarantees that the `SAXParserFactory` is constructed using the container's class loader rather than the `WebappClassLoader`.

The `webDigester` is also declared final. This prevents any subclasses from assigning a new object reference to `webDigester`. (See rule OBJ10-J. Do not use public static nonfinal fields for more information.) It also prevents a race condition where another thread could access `webDigester` before it is fully initialized. (See rule OBJ11-J. Be wary of letting constructors throw exceptions for more information.)

```
protected static final Digester webDigester = init();

protected Digester init() {
  Digester digester = createWebDigester();
  // Does not use the context Classloader at initialization
  digester.getParser();
  return digester;
}
```

Even if the Tomcat server continues to use the `WebappClassLoader` to create the parser instance when attempting to process the `web.xml` and other files, the explicit call to `getParser()` in `init()` ensures that the default parser has been set during prior initialization and cannot be replaced. Because this is a one-time setting, future attempts to change the parser are futile.

Risk Assessment

Allowing untrusted code to load classes enables untrusted code to replace benign classes with Trojan classes.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
SEC03-J	high	probable	medium	**P12**	**L1**

Automated Detection

Tool	Version	Checker	Description
Parasoft Jtest	9.5	**SECURITY.BV.ACL**	Implemented

Related Guidelines

Secure Coding Guidelines for the Java Programming Language, Version 3.0	Guideline 6-3. Safely invoke standard APIs that bypass `SecurityManager` checks depending on the immediate caller's class loader

Android Implementation Details

On Android, the use of `DexClassLoader` or `PathClassLoader` requires caution.

Bibliography

[CVE 2011]	CVE-2009-0783
[Gong 2003]	Section 4.3.2, Class Loader Delegation Hierarchy
[JLS 2005]	§4.3.2, The Class `Object`
[Tomcat 2009]	Bug ID 29936, API Class `org.apache.tomcat.util.digester.Digester`, Security fix in v 6.0.20