Security Vulnerability Notice

SE-2012-01-ORACLE-11

[Security vulnerabilities in Java SE, Issues 56-60]

DISCLAIMER

Security Explorations discovered additional 5 security vulnerabilities in Java SE Platform, Standard Edition. A table below, presents their technical summary:

ISSUE #	TECHNICAL DETAILS	
56	origin	`BytecodeVerifier`
	cause	Bytecode verifier implementation (no real code flow tracking)
	impact	the possibility to create a valid class that does not call an inaccessible constructor of its superclass
	type	partial security bypass vulnerability
57	origin	`java.lang.invoke.MethodHandles.Lookup`
	cause	`MemberName` instance returned by `resolveOrFail` method points to a reference class, instead of a declaring class
	impact	direct access to security sensitive `MethodHandle` objects
	type	partial security bypass vulnerability
58	origin	`java.lang.invoke.MethodHandleProxies`
	cause	insufficient checks for `MethodHandle` object implementing given proxy class functionality
	impact	the use of arbitrary (not type compatible) MethodHandle instance as a target interface implementation
	type	partial security bypass vulnerability
59	origin	`sun.plugin.javascript.JSClassLoader` `sun.plugin.javascript.JSInvoke`
	cause	the possibility to issue limited Reflection API calls in a trampoline class loader namespace
	impact	`sun.*` package access bypass (access to a restricted class and its methods)
	type	partial security bypass vulnerability
60	origin	`sun.plugin.javascript.JSInvoke`
	cause	Arbitrary `invoke` method call done from a privileged class
	impact	the possibility to call `doPrivileged` method from a trusted caller frame
	type	partial security bypass vulnerability

Issue 56 allows to create a valid subclass of a given target class regardless of the fact that a target class has inaccessible constructor (such as with a private access). Bytecode Verifier used in Java SE 7 does not track code flows, but rather relies on type checks conducted at specific code paths merging locations (targets of jumps, exception handlers, etc.). This creates a possibility for abuse. Instead of triggering the exception resulting from an illegal access to a given superclass constructor, one can create a specially crafted instance initialization method that will successfully pass Bytecode Verfier checks regardless of the fact that a superclass constructor is never called. We abuse the above by implementing a specially crafted subclass of `MethodHandleProxies` class:

```
.class public MHP
.super java/lang/invoke/MethodHandleProxies


.method public <init>()V
.limit stack 2
.limit locals 2
l1:
    goto l1
    return
.end method
```

Regardless of the fact that MethodHandleProxies class has a private constructor and that it is never called by its subclass, a valid MHP class can be created in a target Java VM.

Issue 57 allows to obtain direct access to certain security sensitive methods such as asInterfaceInstance method of MethodHandleProxies class. In normal circumstances, returned MethodHandle object for the abovementioned method should be bound to the caller's class. However, due to the fact that MemberName instance returned by resolveOrFail method of MethodHandles.Lookup class points to a reference class, instead of a declaring class, one can successfully bypass a check conducted by isCallerSensitive method of MethodHandleNatives class. This can be accomplished by issuing a method lookup operation (findStatic , etc.) on a subclass of a given target class (MHP in our case), not a security sensitive class.

Issue 58 stems from the fact that it is possible to call an arbitrary, user provided MethodHandle object as if it was a method handle of a different, fixed type. This can be accomplished with the use of a specially crafted method handle instance which inserts additional arguments, before calling the original method handle object. The type of the new method handle drops the types for the inserted (bound) parameters from the original target type, since the new method handle will no longer require those arguments to be supplied by its callers. In our case, we convert a MethodHandle object of (SecurityManager)void type to the ()void type by creating a new MethodHandle object that binds the SecurityManager argument to the NULL value. This is accomplished by the means of insertArguments method of java.lang.reflect.invoke.MethodHandles class. The idea is to dispatch a call to setSecurityManager method of java.lang.System class with the use of a MethodHandle of which type corresponds to run() method of java.security.PrivilegedAction interface.

Issue 59 allows to get access to restricted sun.plugin.javascript.JSInvoke class and its methods. This is caused by the fact that one can successfully issue Reflection API calls on objects that belong to same class loader namespace (JSClassLoader in our case) as the caller of Reflection API calls.

Issue 60 relies on the possibility to call doPrivileged method of java.security.AccessController class with a privileged class set as a caller. In some of our Proof of Concept codes reported to Oracle in 2012, we relied on a possibility to invoke this method through the wrapper doPrivilegedWithCombiner call. At that time, we treated this issue more as a feature than a security bug. However, due to the fact that Oracle has addressed the abovementioned behavior and made it impossible to call a custom (including those defined in a fully privileged Class Loader namespace) PrivilegedAction objects via the wrapper doPrivilegedWithCombiner method call, we now treat it as a bug. A successful call to the doPrivileged method can be now accomplished with the use of the invoke method of sun.plugin.javascript.JSInvoke class. This method is declared in a non-null, but fully privileged Class Loader namespace. This is sufficient for the target call to succeed when invoked through the abovementioned `invoke` method.

Issues 56-60, when combined together can be used to successfully achieve a complete JVM sandbox bypass in a target system. We abuse Issues 56 and 57 to get access to direct `MethodHandle` object pointing to `asInterfaceInstance` method of `MethodHandleProxies` class. We further abuse Issue 58 to create a specially crafted `PrivilegedAction` object instance. This is a `MethodHandleProxy` implementing `java.security.PrivilegedAction` interface. As an argument to the created proxy, we provide a specially crafted instance corresponding to `setSecurityManager` method of `java.lang.System` class. The idea is to have this method called with a prepended NULL argument, in place of the expected `MethodHandle` object pointing the `run()` method of the `PrivilegedAction` interface. Finally we abuse Issues 59 and 60 to get access to the `invoke` method of `sun.plugin.javascript.JSInvoke` class through which a call to the `doPrivileged` method is made with a specially crafted `PrivilegedAction` object provided as an argument. As a result, a successful call to `setSecurityManager` method is issued with a NULL argument, which switches off all Java VM security restrictions.

Attached to this report, there is a Proof of Concept code that illustrates the impact of all the vulnerabilities described above. It has been successfully tested in the environment of Java SE 7 Update 15 (JRE version 1.7.0_15-b03) and both Firefox and Google Chrome web browsers. Please, note that due to the interaction occurring between JavaScript and Java, the HTML file used for Applet launch may need to be modified to achieve same results in the environment of other web browsers such as Internet Explorer, Safari, etc.

About Security Explorations

Security Explorations (`http://www.security-explorations.com`) is a security start-up company from Poland, providing various services in the area of security and vulnerability research. The company came to life in a result of a true passion of its founder for breaking security of things and analyzing software for security defects. Adam Gowdiak is the company's founder and its CEO. Adam is an experienced Java Virtual Machine hacker, with over 50 security issues uncovered in the Java technology over the recent years. He is also the hacking contest co-winner and the man who has put Microsoft Windows to its knees (vide MS03-026). He was also the first one to present successful and widespread attack against mobile Java platform in 2004.