Security Vulnerability Notice

SE-2012-01-ORACLE

[Security vulnerabilities in Java SE, Issues 1-19]

DISLAIMER

INFORMATION PROVIDED IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW NEITHER SECURITY EXPLORATIONS, ITS LICENSORS OR AFFILIATES, NOR THE COPYRIGHT HOLDERS MAKE ANY REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR THAT THE INFORMATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS, OR OTHER RIGHTS. THERE IS NO WARRANTY BY SECURITY EXPLORATIONS OR BY ANY OTHER PARTY THAT THE INFORMATION CONTAINED IN THE THIS DOCUMENT WILL MEET YOUR REQUIREMENTS OR THAT IT WILL BE ERROR-FREE.

YOU ASSUME ALL RESPONSIBILITY AND RISK FOR THE SELECTION AND USE OF THE INFORMATION TO ACHIEVE YOUR INTENDED RESULTS AND FOR THE INSTALLATION, USE, AND RESULTS OBTAINED FROM IT.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL SECURITY EXPLORATIONS, ITS EMPLOYEES OR LICENSORS OR AFFILIATES BE LIABLE FOR ANY LOST PROFITS, REVENUE, SALES, DATA, OR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, PROPERTY DAMAGE, PERSONAL INJURY, INTERRUPTION OF BUSINESS, LOSS OF BUSINESS INFORMATION, OR FOR ANY SPECIAL, DIRECT, INDIRECT, INCIDENTAL, ECONOMIC, COVER, PUNITIVE, SPECIAL, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND WHETHER ARISING UNDER CONTRACT, TORT, NEGLIGENCE, OR OTHER THEORY OF LIABILITY ARISING OUT OF THE USE OF OR INABILITY TO USE THE INFORMATION CONTAINED IN THIS DOCUMENT, EVEN IF SECURITY EXPLORATIONS OR ITS LICENSORS OR AFFILIATES ARE ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS.

Security Explorations discovered 19 security issues in the latest version of Java Platform, Standard Edition. Most of them are caused by the unsafe use of Reflection API. Since, security checks in use by the aforementioned API rely on a caller's class, proper delegation of the calls from untrusted code may lead to the successful bypass of these checks. This may further lead to the creation of arbitrary class instances from restricted packages as well as to the invocation of arbitrary methods on such objects. As a result, complete Java security sandbox compromise can be usually obtained.

A table below, presents a technical summary of all of the issues found:

ISSUE #	TECHNICAL DETAILS	
1	origin	com.sun.org.glassfish.external.statistics.impl.AverageRangeStatisticImpl class
	cause	insecure use of invoke method of java.lang.reflect.Method class
	impact	arbitrary invocation of static methods with user provided arguments
	type	complete security bypass vulnerability
2	origin	com.sun.org.glassfish.external.statistics.impl.BoundaryStatisticImpl class
	cause	insecure use of invoke method of java.lang.reflect.Method class
	impact	arbitrary invocation of static methods with user provided arguments
	type	complete security bypass vulnerability
3	origin	com.sun.org.glassfish.external.statistics.impl.BoundedRangeStatisticImpl class
	cause	insecure use of invoke method of java.lang.reflect.Method class
	impact	arbitrary invocation of static methods with user provided arguments
	type	complete security bypass vulnerability
4	origin	com.sun.org.glassfish.external.statistics.impl.CountStatisticImpl class
	cause	insecure use of invoke method of java.lang.reflect.Method class
	impact	arbitrary invocation of static methods with user provided arguments
	type	complete security bypass vulnerability
5	origin	com.sun.org.glassfish.external.statistics.impl.RangeStatisticImpl class
	cause	insecure use of invoke method of java.lang.reflect.Method class
	impact	arbitrary invocation of static methods with user provided arguments
	type	complete security bypass vulnerability
6	origin	com.sun.org.glassfish.external.statistics.impl.StringStatisticImpl class
	cause	insecure use of invoke method of java.lang.reflect.Method class
	impact	arbitrary invocation of static methods with user provided arguments
	type	complete security bypass vulnerability
7	origin	com.sun.org.glassfish.external.statistics.impl.TimeStatisticImpl class
	cause	insecure use of invoke method of java.lang.reflect.Method class
	impact	arbitrary invocation of static methods with user provided arguments
	type	complete security bypass vulnerability
8	origin	javax.management.remote.rmi.RMIConnectionImpl class
	cause	the use of OrderClassLoaders as Thread's contextClassLoader
	impact	arbitrary access to restricted classes
	type	partial security bypass vulnerability
9	origin	javax.management.remote.rmi.RMIConnectionImpl class
	cause	the use of null class loader as Thread's contextClassLoader

	impact	arbitrary access to restricted classes
	type	partial security bypass vulnerability
10	origin	bytecode verifier for Java SE 7
	cause	wrong check for a target of `invokespecial` bytecode (it is not limited to `this` and `super` classes in case of an `<init>` method)
	impact	ability to create object instances without the need to call superclass' initializer, arbitrary access to restricted classes via custom class loader objects, further impact not yet evaluated
	type	partial security bypass vulnerability
11	origin	JVM implementation of finalizers
	cause	the use of `null` class loader as Thread's `contextClassLoader`
	impact	arbitrary access to restricted classes
	type	partial security bypass vulnerability
12	origin	difficult to classify
	cause	unrestricted `getClass` method call
	impact	arbitrary access to restricted classes
	type	partial security bypass vulnerability
13	origin	`java.lang.invoke.MethodTypes` class
	cause	no security check in the `in` method
	impact	the ability to create `java.lang.invoke.MethodTypes.Lookup` object with a system `lookupClass`, this allows to obtain method handles from restricted classes and to issue calls on them
	type	partial security bypass vulnerability
14	origin	`com.sun.jmx.mbeanserver.GetPropertyAction` class
	cause	public class
	impact	arbitrary access to Java system properties
	type	partial security bypass vulnerability
15	origin	`java.util.logging.LogManager` class
	cause	lack of a type check of a logger handler prior to creating its instance
	impact	the ability to bypass security checks implemented in static class initializers of a 3rd party software
	type	partial security bypass vulnerability
16	origin	`com.sun.beans.finder.MethodFinder` class
	cause	insecure use of `getMethod` method of `java.lang.Class` class
	impact	access to method objects of restricted classes
	type	exploitation vector (requires a security bypass precondition)
17	origin	`com.sun.beans.finder.ConstructorFinder` class
	cause	insecure use of `getConstructors` method of `java.lang.Class` class
	impact	arbitrary access to constructors of restricted classes, creation of restricted public classes
	type	exploitation vector (requires a security bypass precondition)
18	origin	`com.sun.org.glassfish.gmbal.util.GenericConstructor` class
	cause	insecure use of `getDeclaredConstructors` and `newInstance` methods of `java.lang.Class` class
	impact	creation of restricted public classes
	type	exploitation vector (requires a security bypass precondition)
19	origin	`com.sun.org.glassfish.gmbal.ManagedObjectManagerFactory` class
	cause	insecure use of `getDeclaredMethod` method of `java.lang.Class` class
	impact	access to method objects of restricted classes
	type	exploitation vector (requires a security bypass precondition)

Below, we provide additional comments with respect to the issues presented in the table above:

- complete security bypass (CSB) issues (1-7)
  These issues allow to bypass security checks relying on the class loader of a caller class. In our Proof of Concept codes, we exploit them for the purpose of:
  - issuing a call to `forName` method of `java.lang.Class` class in order to obtain a reference to the restricted class (from `sun` package),
  - creating an instance of `java.lang.invoke.MethodHandles.Lookup` object with a system class object in the `lookupClass` field. Such a `Lookup` object allows to obtain and call arbitrary methods of restricted classes.

  The above is sufficient to obtain a complete compromise of JVM security sandbox. A common exploitation scenario makes use of the created object instance of `java.lang.invoke.MethodHandles.Lookup` class:

  - a `MethodHandle` object to the `getField` method of `sun.awt.SunToolkit` class is obtained and called in order to obtain a privileged instance of `unsafe` field object of `java.util.concurrent.atomic.AtomicBoolean` class,
  - the actual value held by a static `unsafe` field object is obtained (instance of `sun.misc.Unsafe` class),
  - a `MethodHandle` object to `defineClass` method of `sun.misc.Unsafe` class is obtained and called in order to define a custom `Helper` class in a system (`null`) class loader's namespace and in a system (`null`) protection domain. As a result, `Helper` class is fully privileged and can for example make a successful call to `setSecurityManager` method of `java.lang.System` class and can switch off the security manager completely (all in a proper `doPrivileged` block).
- partial security bypass (PSB) issues (8-11)
  These issues allow to bypass security manager's check verifying access to restricted packages such as `sun`, `com.sun.imageio`, `com.sun.xml.internal.bind` and `com.sun.xml.internal.ws`. The bypass is always done with respect to some class loader object (implicitly created or set as a `contextClassLoader` of the current thread). In our Proof of Concept codes, we exploit them for the purpose of issuing a call to `forName` method of `java.lang.Class` class. As a result, we are able to obtain a reference to the class instance of a restricted class object (usually `sun.swing.SwingLazyValue` or `sun.awt.SunToolkit`).
- partial security bypass (PSB) issues (12-13)
  These issues when combined together allow for a complete compromise of JVM security sandbox. The exploitation scenario is similar to those presented for issues 1-7.
- partial security bypass (PSB) issue 15
  We verified that this issue can help bypass security of some 3rd party software (security check in a static class initializer). In the sample Proof of Concept code, an instance of `java.lang.SecurityManager` class is successfully created (console log shows an

exception after `newInstance` invocation and as a result of an illegal type cast operation).

- exploitation vectors (EV) issues (16-19)
  These issues allow to achieve a full JVM sandbox compromise upon the condition set up by one of the partial security bypass issues. Each exploitation vector relies on a carefully crafted sequence of Reflection API calls implemented by one publicly available class (denoting the primary vector issue) and at least one class from a restricted `sun` package (usually `sun.swing.SwingLazyValue` or `sun.awt.SunToolkit`). The goal of a publicly available class is to either obtain a constructor or a method object of the restricted class, so that its instance could be created or a method called. Exploitation scenario is usually the same with respect to the Reflection API sequence making use of restricted classes:
  - a call to `getField` method of `sun.awt.SunToolkit` class is made in order to obtain a privileged instance of `unsafe` field object of `java.util.concurrent.atomic.AtomicBoolean` class,
  - a call to `getMethod` method of `sun.awt.SunToolkit` class is made in order to obtain a privileged instance of `defineClass` method object of `sun.misc.Unsafe` class,
  - the actual value held by a static `unsafe` field object is obtained (instance of `sun.misc.Unsafe` class),
  - static `defineClass` method is invoked on the obtained instance of `sun.misc.Unsafe` class. As a result, custom `Helper` class is defined in a system (`null`) class loader's namespace and in a system (`null`) protection domain. As a result, `Helper` class is fully privileged and can for example make a successful call to `setSecurityManager` method of `java.lang.System` class and can switch off the security manager completely (all in a proper `doPrivileged` block).

Neither PSB issues 8-11, nor EV issues 16-19 could be used alone to achieve full JVM compromise. However, when combined together (1 PSB issue + 1 EV issue), complete JVM security sandbox escape could be achieved (malicious Java code could run unrestricted in the context of JVM process). As a result, it is possible to form 12 independent complete security bypass exploits (Issues 1-7, Issue 8 + 16, Issue 9 + 17, Issue 10 + 18, Issue 11 + 19, Issue 12 + 13).

Presented security issues violate many Secure Coding Guidelines for the Java Programming Language [1]. This includes, but is not limited to:

- Guideline 4-4: Limit exposure of ClassLoader instances
- Guideline 4-5: Limit the extensibility of classes and methods
- Guideline 5-1: Validate inputs
- Guideline 7-3: Defend against partially initialized instances of non-final classes
- Guideline 9-1: Understand how permissions are checked
- Guideline 9-8: Safely invoke standard APIs that bypass SecurityManager checks depending on the immediate caller's class loader

- Guideline 9-9: Safely invoke standard APIs that perform tasks using the immediate caller's class loader instance
- Guideline 9-10: Be aware of standard APIs that perform Java language access checks against the immediate caller
- Guideline 9-11: Be aware java.lang.reflect.Method.invoke is ignored for checking the immediate caller

Attached to this report, there are 15 Proof of Concept codes that illustrate each of the reported issues. The codes use the following convention when it comes to the class names:

- VulnX
  Code implementing security bypass issue number X.
- VectorX
  Code implementing exploitation vector number X.

All Proof of Concept codes have been successfully tested in a Windows OS environment and with the following versions of Java SE:

- JRE/JDK 7 (version 1.7.0-b147)
- JRE/JDK 7u1 (version 1.7.0_01-b08)
- JRE/JDK 7u2 (version 1.7.0_02-b13)
- JRE/JDK 7u3 (version 1.7.0_03-b05)
- JRE/JDK 7u4 (version 1.7.0_04-ea-b18, early access release from 29 Mar 2012)

REFERENCES

[1] Secure Coding Guidelines for the Java Programming Language, Version 4.0, http://www.oracle.com/technetwork/java/seccodeguide-139067.html

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.