Security Vulnerability Notice

SE-2012-01-ORACLE-10

[Security vulnerabilities in Java SE, Issues 54 and 55]

DISCLAIMER

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 two security vulnerabilities in Java SE Platform, Standard Edition. A table below, presents their technical summary:

ISSUE #	TECHNICAL DETAILS	
54	origin	`java.lang.invoke.MethodHandles`
	cause	The lack of security checks in a family of `MethodHandle` resolving methods
	impact	Access to protected members of arbitrary classes
	type	partial security bypass vulnerability
55	origin	`java.lang.invoke.MethodHandles`
	cause	Insufficient type checks
	impact	The possibility to change the receiver object of arbitrary `MethodHandle` object to the one of incompatible type
	type	partial security bypass vulnerability

Issue 54 stems from the fact that certain `MethodHandle` lookup methods (`resolveVirtual`, `resolveStatic`, etc.) of `java.lang.invoke.MethodHandles` class do not invoke the `checkSecurityManager` method during target class member resolution process. This is clearly visible when arbitrary *find* and *resolve* methods corresponding to a given `MethodHandle` lookup operation are compared as in the case of `findVirtual` and `resolveVirtual` methods denoted below:

```
        public MethodHandle findVirtual(Class class1, String s, MethodType
methodtype) throws NoSuchMethodException, IllegalAccessException {
            MemberName membername = resolveOrFail(class1, s, methodtype, false);
            checkSecurityManager(class1, membername); ← this call is missing below
            Class class2 = findBoundCallerClass(membername);
            return accessVirtual(class1, membername, class2);
        }

        private MethodHandle resolveVirtual(Class class1, String s, MethodType
methodtype) throws NoSuchMethodException, IllegalAccessException {
            MemberName membername = resolveOrFail(class1, s, methodtype, false);
            return accessVirtual(class1, membername, lookupClass);
        }
```

The above indicates the lack of a security check in `resolveVirtual` method. Although, this method is private and is not invoked by any publicly available API method, it may be still called by the Java VM during Class file parsing. This is in particular done whenever `MethodHandle` entries are encountered in a target Class file's *ConstantPool*.

For the purpose of our Proof of Concept code we generate a specially crafted `MyCL` class file containing a `MethodHandle` reference to `defineClass` method of `java.lang.ClassLoader` class in its *ConstantPool*. A dump of the resulting file is provided below:

```
public class MyCL extends java.lang.ClassLoader
  SourceFile: "MyCL.java"
  minor version: 0
  major version: 51
  flags: ACC_PUBLIC, ACC_SUPER
Constant pool:
```

```
   #1 = Methodref          #5.#16         //  java/lang/ClassLoader."<init>":()V
   #2 = Methodref          #5.#17         //
java/lang/ClassLoader.defineClass:(Ljava/lang/String;[BIILjava/security/ProtectionD
omain;)Ljava/lang/Class;
   #3 = String             #10            //  dummy
   #4 = Class              #18            //  MyCL
   #5 = Class              #19            //  java/lang/ClassLoader
   #6 = Utf8               <init>
   #7 = Utf8               ()V
   #8 = Utf8               Code
   #9 = Utf8               LineNumberTable
  #10 = Utf8               dummy
  #11 = Utf8
(Ljava/lang/String;[BIILjava/security/ProtectionDomain;)V
  #12 = Utf8               get_defineClass_mh
  #13 = Utf8               ()Ljava/lang/Object;
  #14 = Utf8               SourceFile
  #15 = Utf8               MyCL.java
  #16 = NameAndType        #6:#7          //  "<init>":()V
  #17 = NameAndType        #20:#21        //
defineClass:(Ljava/lang/String;[BIILjava/security/ProtectionDomain;)Ljava/lang/Clas
s;
  #18 = Utf8               MyCL
  #19 = Utf8               java/lang/ClassLoader
  #20 = Utf8               defineClass
  #21 = Utf8
(Ljava/lang/String;[BIILjava/security/ProtectionDomain;)Ljava/lang/Class;
  #22 = MethodHandle       #5:#2          //  invokevirtual
java/lang/ClassLoader.defineClass:(Ljava/lang/String;[BIILjava/security/ProtectionD
omain;)Ljava/lang/Class;
```

ConstantPool at index 22 contains the `MethodHandle` entry which will be successfully resolved with the use of the `resolveVirtual` method during Class file parsing. This can be accomplished due to the missing security checks in the abovementioned method.

Issue 55 relies on the possibility to bind the receiver of a target `MethodHandle` object to the object instance of incompatible type. In case of the `defineClass MethodHandle` we retrieve in our code, one can bind its receiver object to the instance of `java.lang.ClassLoader` class, regardless of the fact that the receiver object type is originally restricted to `MyCL` class:

```
MethodHandle(MyCL,String,byte[],int,int,ProtectionDomain)Class
```

Issues 54 and 55, when combined together can be used to successfully achieve a complete JVM sandbox bypass in a target system. The ability to define custom classes with arbitrary user provided Protection Domain is sufficient to achieve that. It is very probable that Issue 55 could be used alone to achieve a complete sandbox bypass via a type confusion attack. That however requires more thorough investigation.

Attached to this report, there is a Proof of Concept code that illustrates the impact of both vulnerabilities. It has been successfully tested in the environment of Java SE 7 Update 15 (JRE version 1.7.0_15-b03).

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.