Security Vulnerability Notice

SE-2012-01-PUBLIC

[Security vulnerabilities in Java SE, Issue 54]

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.

VULNERABILITY DETAILS

Security Explorations discovered a security vulnerability in Java SE Platform, Standard Edition. A table below, presents its 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

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.

IMPACT

Described Issue 54 is not sufficient to implement a functional and successful attack code in the environment of Java SE 7. Security Explorations discovered another issue (number 55) affecting Oracle's Java SE 7 that allows to do this.

Issues 54 and 55, when combined together can be used to successfully achieve a complete Java security sandbox bypass in a target system. Proof of Concept code illustrating the impact of both vulnerabilities has been successfully tested in the environment of Java SE 7 Update 15 and Java SE 7 Update 17.

VENDOR'S RESPONSE

On Feb 25 2013, Security Explorations sent a vulnerability notice to Oracle containing detailed information about two discovered vulnerabilities (Issues 54 and 55). Along with that, the company was also provided with source and binary codes for a Proof of Concept codes illustrating the impact of both security issues found.

On Feb 27, 2013 Oracle provided the results of its assessment and informed that Issue 54 was not treated as a vulnerability as it demonstrated the "allowed behavior". Company's denial of the issue as a security bug was made on the following basis:

"obtaining a method handle for a protected method from a superclass is allowed behavior"

Security Explorations didn't agree with the above assessment and on the same day provided its counterarguments to Oracle. We indicated that Issue 54 abused the missing security

manager check in `resolveVirtual` method in order to gain access to method handle objects of certain security sensitive classes such as Class Loaders. In our Proof of Concept code, we were able to access Method Handle object pointing to `defineClass` method of `java.lang.ClassLoader` class.

Oracle claimed that accessing a protected member such as a Method Handle from a superclass is an allowed behavior. This is not true as demonstrated by the code below:

```
public class MyCL extends ClassLoader {
 public static void test() {
  try {
   MethodHandles.Lookup l=MethodHandles.lookup();
   System.out.println("lookup: "+l.lookupClass()+"/"+l.lookupModes());

   Class ctab[]=new Class[5];
   ctab[0]=java.lang.String.class;
   ctab[1]=(new byte[0]).getClass();
   ctab[2]=Integer.TYPE;
   ctab[3]=Integer.TYPE;
   ctab[4]=java.security.ProtectionDomain.class;

   MethodType desc=MethodType.methodType(java.lang.Class.class,ctab);

   MethodHandle
mh=l.findVirtual(java.lang.ClassLoader.class,"defineClass",desc);
   System.out.println(mh);
  } catch(Throwable t) {
   t.printStackTrace();
  }
 }
}
```

The above code does exactly the same thing as a code sequence we use in our Proof of Concept code. The only difference is in the method that gets called at the time of Method Handle resolution (here `findVirtual`, in our PoC this is `resolveVirtual`).

The above code tries to access a protected member (`defineClass` Method Handle) from the subclass of the class that declares that member. However, contrary to Oracle's claim such an access is not allowed. It is blocked by the `checkSecurityManager` method:

```
Security manager = sun.plugin2.applet.AWTAppletSecurityManager@c3cae5
lookup: class MyCL/15
java.security.AccessControlException: access denied
("java.lang.RuntimePermission" "accessDeclaredMembers")
      at java.security.AccessControlContext.checkPermission(Unknown Source)
      at java.security.AccessController.checkPermission(Unknown Source)
      at java.lang.SecurityManager.checkPermission(Unknown Source)
      at java.lang.SecurityManager.checkMemberAccess(Unknown Source)
      at java.lang.invoke.MethodHandles$Lookup.checkSecurityManager(Unknown
Source)
      at java.lang.invoke.MethodHandles$Lookup.findVirtual(Unknown Source)
      at MyCL.test(MyCL.java:39)
      at BlackBox.<init>(BlackBox.java:31)
      ...
```

The above result is consistent with Java SE documentation [1] describing Security Manager interactions conducted at the time of member lookup operations:

"If a security manager is present, member lookups are subject to additional checks."

"If the retrieved member is not public, smgr.checkMemberAccess(defc,Member.DECLARED) is called."

We also indicated to Oracle that even partially initialized Class Loader instances are not allowed in Java SE and that core Reflection API does not allow access to protected members of system classes, unless access to declared members is granted.

On 05 Mar 2013, Oracle informed us that it was continuing to evaluate Security Explorations' arguments regarding Issue 54. The company provided the following background for its analysis:

"The rules controlling runtime reflection are different from the resolution of a method handle in a class file constant pool (see [2], [3] for details). The two methods of obtaining method handles (via constant pool and reflection) have different models for when access checks are applied. For the constant pool case, the JVM applies the access control checks that are consistent for all forms of constant pool resolution. If a valid class file can contain an invokespecial (or other invoke instruction) for a method, then a method handle for that method is allowed in the constant pool. In your report #54, there is an invokespecial for:

Method java/lang/ClassLoader.defineClass:(Ljava/lang/String;[BIILjava/security/ProtectionDomain;)Ljava/lang/Class;

in MyCL.class, and thus a method handle for the same method is allowed. If this method were package private or private, the modified class would throw an IncompatibleClassChangeError at load time. While the two systems parallel one another, their behavior is different."

What's important to note is that the above background includes arguments for the "allowed behavior" again. This time this is however done in a context of JVM specification and Constant Pool resolution.

On Mar 11 2013, we asked Oracle about the final evaluation of Issue 54. In a response, the company informed us that it was still continuing to evaluate it.

As of Mar 18, 2013 we have no information that the company treats the issue as a security vulnerability.

FINAL WORDS

Security Explorations believes that 3 weeks (from Feb 25 to Mar 18) constitutes enough time for a major software vendor to be able to deliver a final confirmation or denial of a reported security issue. This especially concerns a vendor that has been a subject of a considerable criticism regarding competent and prompt handling of security vulnerabilities in its software.

Security Explorations does not agree with Oracle's arguments and reasoning provided so far with respect to Issue 54. A general rule in security is that same circumstances / constraints should lead to consistent (same, not different) security access related decisions. In case of Issue 54, resolving protected members of superclasses should be either always allowed or denied for all code paths available to untrusted code (irrespective whether a member is

resolved with the use of a public API or internally by the Java VM operating on behalf of an untrusted code).

Security Explorations is not aware of any other way to obtain a Method Handle to the protected member of `java.lang.ClassLoader` class that would not be the outcome of a security vulnerability.

Security Explorations failed to launch a successful Java security sandbox bypass scenario upon access to `defineClass` Method Handle obtained with the use of a different vulnerability (Issue 57). That contradicts the claim that Issue 54 is the "allowed behavior". It also contradicts an indirect conclusion that Issue 55 is alone sufficient to launch the attack demonstrated to the company.

Our tests indicate that Issue 55 can be combined with a Method Handle object obtained with the use of Issue 54 only.

If Oracle sticks to the "allowed behavior" scenario, in order to maintain proper consistency of security checks in Java SE, the company should relax some of security checks present in Reflection API code and apply proper changes to Java SE documentation [1] as well. The alternative is to admit to the fault regarding the evaluation of Issue 54 and begin to treat it as a security vulnerability being the result of inconsistent security design of new Reflection API (no security checks enforced by JVM specification during Method Handles resolution [2][3]).

REFERENCES

[1] Class MethodHandles.Lookup, Security manager interactions
`http://docs.oracle.com/javase/7/docs/api/java/lang/invoke/MethodHandles.Lookup.html#secmgr`
[2] The Java Virtual Machine Specification, The CONSTANT_MethodHandle_info Structure
`http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html#jvms-4.4.8`
[3] The Java Virtual Machine Specification, Method Type and Method Handle Resolution
`http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-5.html#jvms-5.4.3.5`

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.