Security Vulnerability Notice

SE-2012-01-IBM-2

[Security vulnerabilities in Java SE, Issues 62-68]

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 7 additional security issues in the latest version of IBM SDK, Java Technology Edition software [1]. Most of them are related to unsafe use or implementation of Java Reflection API. A table below, presents their technical summary:

ISSUE #	TECHNICAL DETAILS	
62	origin	*Class File* parsing (IBM J9 Java VM)
	cause	no receiver binding for protected members of arbitrary classes
	impact	invocation of protected methods of arbitrary objects
	type	complete security bypass vulnerability
63	origin	*Class File* parsing (IBM J9 Java VM)
	cause	interpretation of `EnclosingMethod` attribute
	impact	access to declared Method objects of arbitrary classes
	type	partial security bypass vulnerability
64	origin	`java.lang.MethodHandles.Lookup`
	cause	no receiver binding for protected members of arbitrary classes
	impact	invocation of protected methods of arbitrary objects
	type	partial security bypass vulnerability
65	origin	`com.ibm.rmi.io.ValueHandlerImpl`
	cause	unsafe implementation of deserialization functionality
	impact	access to arbitrary fields of `Serializable` classes
	type	partial security bypass vulnerability
66	origin	`java.lang.invoke.MethodType`
	cause	unsafe deserialization of `MethodType` objects
	impact	mutable `MethodType` objects
	type	partial security bypass vulnerability
67	origin	`com.ibm.CORBA.iiop.ClientDelegate`
	cause	insecure use of `invoke` method of `java.lang.reflect.Method` class
	impact	arbitrary method invocation inside `AccessController`'s `doPrivileged` block
	type	complete security bypass vulnerability
68	origin	`com.ibm.rmi.io.ObjectStreamClass`
	cause	insecure implementation of reflective Field access
	impact	privileged access to arbitrary fields of `Serializable` classes
	type	complete security bypass vulnerability

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

- Issues 62 and Issue 63 are similar. They both allow to obtain access to protected members of system classes such as Class Loaders. Issue 62 allows to obtain `MethodHandle` objects which are not bound to the `MethodHandles.Lookup` class instance that produced them. Issue 63 provides access to declared `Method` objects of system classes, which can be further turned into unbound instances of `MethodHandle` class with the use of `unreflect` call of the `MethodHandles.Lookup` class (Issue 64).
- Issues 65 and 66, when combined together can be used to break immutability of a `MethodType` class. Issue 65 allows to obtain access to arbitrary fields of serializable classes. Issue 66 exploits the fact that a serialization process of `MethodType` class operates on real instance field values, rather than on their copies. In our Proof of Concept code, access to `arguments` array of a given `MethodType` instance is

abused to create a specially crafted type confusion condition with the use of static getter `MethodHandle` objects.

- Issues 67 is yet another instance of insecure use of `invoke` method of `java.lang.reflect.Method` class. It is exploited to successfully call `setSecurityManager` method of `java.lang.System` class.
- Issues 68 allows to obtain access to private `Field` objects of `Serializable` classes. In our Proof of Concept code, this condition is abused to set value of a `protectionDomain` field of `java.lang.Class` objects corresponding to user loaded classes. This is sufficient to mark them as fully privileged and to successfully invoke security sensitive methods inside `AccessController`'s `doPrivileged` block.

Additionally to the above, we would like to inform you that several issues reported to IBM in Sep 2012 had not been fixed correctly. This in particular includes Issues 35, 36, 37 and 49 as illustrated by a sample fix for Issue 37:

```
SecurityManager securitymanager = System.getSecurityManager();

If (securitymanager != null && this_obj != null) {
      Class class2 =  (this_obj instanceof Class) ?
                      (Class)this_obj : this_obj.getClass();
      String package = JavaUtil.getPackageName(class2.getName());
      securitymanager.checkPackageAccess(package);
}

Object res = method.invoke(this_obj, args);

If (securitymanager != null && res != null) {
      Class res_class = (res instanceof Class) ?
                        (Class)res : res.getClass();
      String package = JavaUtil.getPackageName(res_class.getName());
      securitymanager.checkPackageAccess(package);
}
```

The above fix only tries to detect the use of a restricted Class object as either an argument or a result of the `invoke` call. This fix doesn't take into account the possibility to load Class object with the use of a class array signature. It doesn't guard against the invocation of other security sensitive methods either. This in particular includes new Reflection API calls that rely on a caller class for security purposes.

Fix for issue 49 does not sufficiently protect against access to privileged `ByteCodeArraysClassLoader` class as subclasses of this class are still allowed (protected static access). Additionally, `defineClass` method does not use `ProtectionDomain` of `ByteCodeArraysClassLoader` subclass, but a privileged domain of a system class. That's due to the fact that this is `ByteCodeArraysClassLoader` class, not its user provided subclass that gets instantiated in `newByteCodeArraysClassLoader` method.

Attached to this report, there are 9 Proof of Concept codes that illustrate all of the reported issues (4 broken fixes and 5 new ones). Each of them demonstrates a complete compromise

of a Java security sandbox. They have been successfully tested in a 32-bit Linux OS environment and with the following version of IBM SDK:

- IBM SDK, Java Technology Edition, Version 7.0 SR4 FP1 for Linux (32-bit x86), build pxi3270sr4fp1-20130325_01(SR4 FP1)

REFERENCES

[1] IBM developer kits , http://www.ibm.com/developerworks/java/jdk/

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.