Security Vulnerability Notice

SE-2012-01-ORACLE-14

[Security vulnerabilities in Java SE, Issue 69#2]

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 that Issue 69 [1] reported to Oracle in Jul 2013 was improperly fixed. According to the company, the vulnerability was addressed by a backported (from JDK 8) implementation of the affected component (method handles API) in JDK 7 Update 40 from Sep 2013. Below, technical details of the flawed fix implementation are provided.

Issue 69 had its origin in insecure implementation of new Reflection API. When Method Handle objects were invoked across two different Class Loader namespaces, no checks were done against the type safety of their argument types. As a result, it was possible to provide a spoofed definition for a given argument type, which could be treated as of a completely different type in a target Class Loader namespace.

Oracle patch for Issue 69 incorporated a check for type aliasing (spoofing). It has a form of the `checkForTypeAlias` method, which is invoked for each successfully resolved `MemberName` object . This is illustrated on Fig. 1.

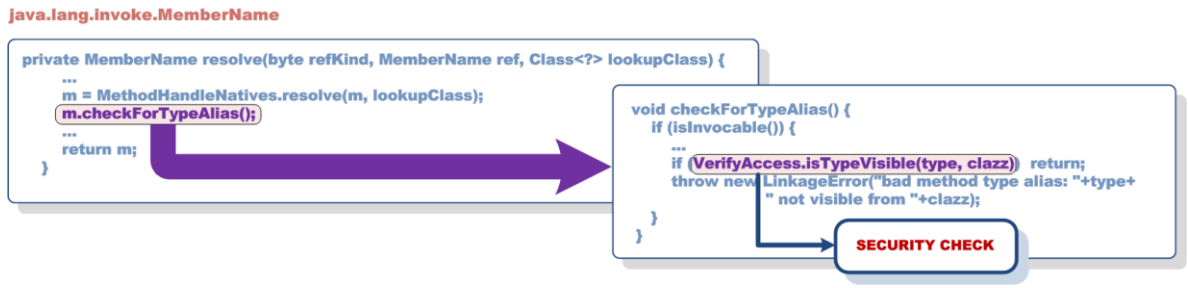

Fig. 1 The implementation of a `checkForTypeAlias` method.

The `checkForTypeAlias` method further calls the `isTypeVisible` method of `sun.invoke.util.VerifyAccess` class (Fig. 2). It takes two arguments, which correspond to the type (class) of a member to check against spoofing and a lookup Class used for its resolving.

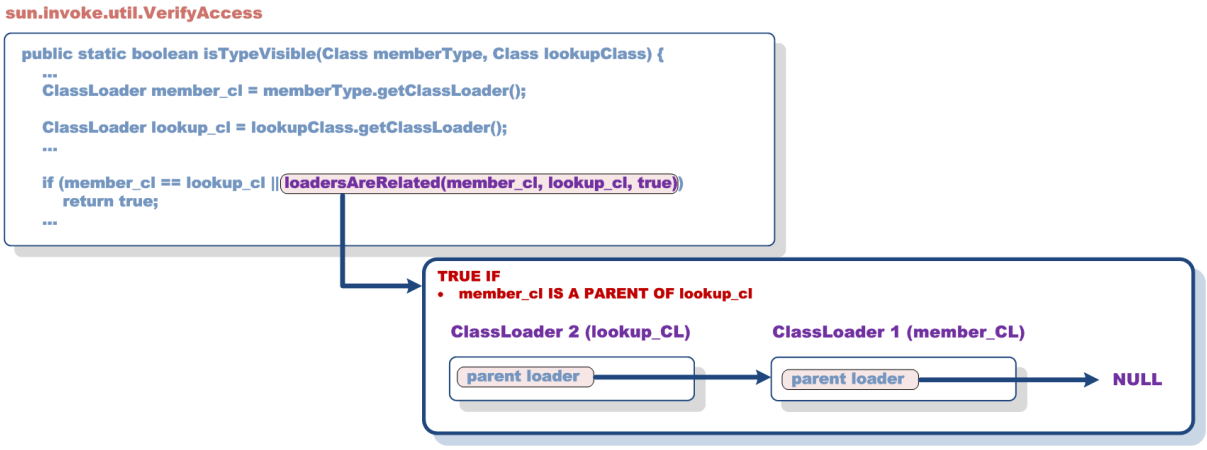

Fig. 2 The implementation of an `isTypeVisible` method.

As part of a type visibility check, a call to `loadersAreRelated` method is made, which verifies whether Class Loaders of the member type and a lookup class are related. The loaders are related when one of them is a parent of the other one. In this particular case, a Class Loader of a member type (`member_CL`) needs to be a parent of the lookup loader (`lookup_CL`).

Fulfilling the `loadersArRelated` condition is thus fairly simple. It requires that the following change is applied to our original Proof of Concept code from 2013 (`Vuln69.java` file):

Original code sequence:

```
URLClassLoader cl2=URLClassLoader.newInstance(utab,null);
```

New code sequence enforcing the `loadersArRelated` condition:

```
URLClassLoader cl2=URLClassLoader.newInstance(utab,cl1);
```

There is however one more obstacle that needs to be overcome in order to achieve the type spoofing condition from our original POC code.

When a request to load Class A is initiated by the `lookup_CL` (from Class Loader 2 namespace), its loading is delegated to the parent loader (`member_CL`). As a result, requested class definition is provided from Class Loader 1 namespace. However, successful type spoofing requires that this definition comes from a `lookup_CL` (Class Loader 2 namespace). This implicates the use of a custom HTTP server that enforces the *404 Not Found error*, when an attempt to load Class A from Class Loader 1 namespace occurs for the first time. This is illustrated on Fig. 3. As a result, `ClassNotFoundException` is thrown by `loadClass` method of the parent loader and Class loading proceeds with the use of a `lookup_CL` (Class Loader 2 namespace).

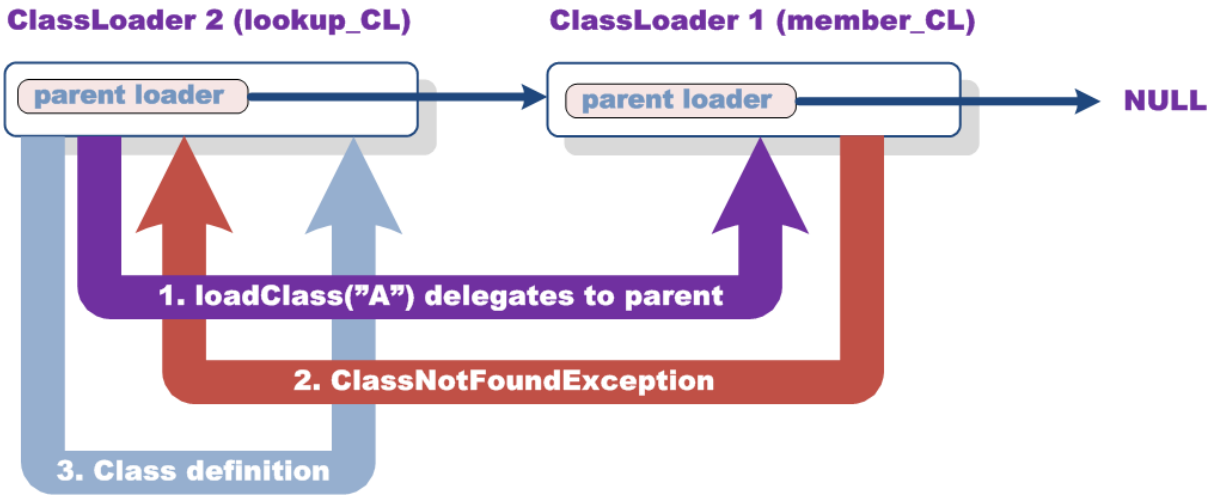

Fig. 3 The enforcement of a class spoofing condition (Class A loading from Class Loader 2 namespace).

This tricked Class loading process is illustrated by the following custom HTTP server output:

```
requesting file: /Issue69/index.html
requesting file: /Issue69/BlackBox.class
requesting file: /Issue69/Vuln69.class
requesting file: /Issue69/A.class
- enforcing 404 (Not Found)
requesting file: /Issue69//data/A.class
requesting file: /Issue69/Helper.class
requesting file: /Issue69//data/Helper.class
```

```
requesting file: /Issue69/Exploit.class
requesting file: /Issue69//data/Exploit.class
requesting file: /Issue69/A.class
requesting file: /Issue69/MyPermissions.class
requesting file: /Issue69//data/MyPermissions.class
requesting file: /Issue69/MyAccessControlContext.class
requesting file: /Issue69//data/MyAccessControlContext.class
requesting file: /Issue69/MyProtectionDomain.class
requesting file: /Issue69//data/MyProtectionDomain.class
requesting file: /Issue69/BlackBox.class
```

We implemented a Proof of Concept code that illustrates the impact of the broken fix described above. It has been successfully tested in the environment of Java SE 7 Update 97, Java SE 8 Update 74 and Java SE 9 Early Access Build 108. In all cases, a complete Java security sandbox escape could be achieved.

At the end, it's worth to note that Issue 69 (CVE-2013-5838) was also improperly evaluated by Oracle in terms of a vulnerability impact. Oracle Critical Patch Update from Oct 2013 indicated that Issue 69 could "be exploited only through sandboxed Java Web Start applications and sandboxed Java applets" (Fig. 4).

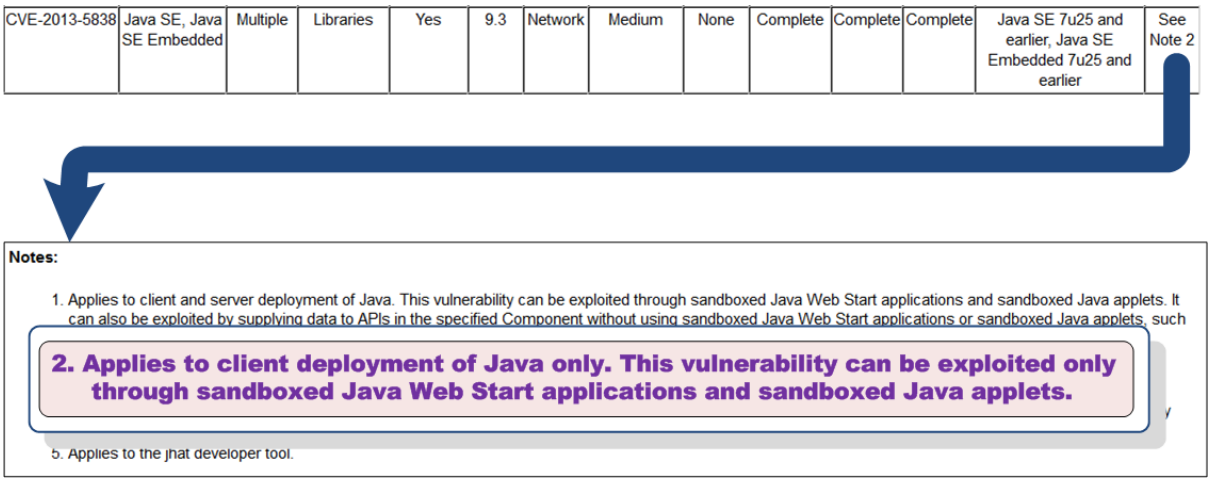

Fig. 4 False statement regarding Issue 69 impact.

This is not true. We proved that Issue 69 could be successfully exploited in a server environment as well such as Google App Engine for Java [2].

REFERENCES

[1] SE-2012-01-ORACLE-13, Issue 69
http://www.security-explorations.com/materials/SE-2012-01-ORACLE-13.pdf
[2] SE-2014-02, Issue21 (POC23)
http://www.security-explorations.com/materials/se-2014-02-32-34.zip

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.