Introduction

The vulnerability in question, named CVE-2013-5838, was first discovered in 2013 by a Polish security firm Security Explorations. They released a detailed report, along with proof-of-concept exploit code, on how this bug could be exploited. This led to a series of events leading to the final fix in 2016. The first patch for this vulnerability was released by Oracle in 2013. Oracle then assured that the problem was fixed, however, that wasn’t the case. In March, 2016, Security Explorations announced, that the fix was not sufficient and the vulnerability could still be exploited. They also released a detailed report on how to bypass the broken fix, along with proof-of-concept exploit code. Final fix, which was later confirmed sufficient by Security Explorations, was finally issued in March, 2016.

In this paper I will be talking about the original vulnerability and the final fix in 2016.

What is it and how does it work?

The bug in question is a Java vulnerability that was caused by an insecure implementation of the Java Reflection API at core VM level. The problem was caused by missing type safety checks for MethodHandle calls across class loaders. What this means, is that when MethodHandle objects are invoked across two different Class Loader namespaces, no checks would be done against the type safety of their argument types[1]. Therefore it is possible to provide a spoofed definition for a given argument type, which will be treated as of a different type in target Class Loader namespace[1]. By doing this, the attacker can change privileges to gain access to client’s machine, compromising it.

Since all of this can be done remotely, without any authentication, completely compromising a system just by tricking the user into accessing a malicious website, the vulnerability was scored 9.3/10 in CVE¹ database, using CVSS².

Security Explorations reported, that the vulnerability was successfully tested in the environment of Java SE 7 Update 25 (JRE version 1.7.0_25-b16) with Internet Explorer 9 and Mozilla Firefox 20.0.1 web browsers.

The vulnerability is illustrated in Figure 1[1].

¹ Common Vulnerabilities and Exposures
² Common Vulnerability Scoring System
The final fix

After a botched patch from Oracle in 2013, the vulnerability was finally fixed with Java SE 8 update 77 and Java SE 7 update 99 in March, 2016. The exploit was blocked by implementing two methods called `checkForTypeAlias` and `isTypeVisible` in classes `java.lang.invoke.MemberName` and `sun.invoke.util.VerifyAccess`, respectively. `checkForTypeAlias` method is invoked for every successfully resolved `ObjectName` object, which then calls `isTypeVisible` method to check, if the given types are related. Type alias checking logic in `checkForTypeAlias` method relies on the fact that spoofing is impossible if a class has a name starting with “java.”. This is guaranteed by `ClassLoader.defineClass`, the API for actually loading classes, which guarantees that classes with names beginning with “java.” cannot be aliased, because class loaders cannot load them directly. [2][3]

Both of these methods can be seen in figures 2 and 3.
```java
public static boolean isTypeVisible(Class<?> type, Class<?> refc) {
    if (type == refc) {
        return true; // easy check
    }
    if (type.isPrimitive() || type == Object.class) {
        return true;
    }
    ClassLoader typeLoader = type.getClassLoader();
    ClassLoader refcLoader = refc.getClassLoader();
    if (typeLoader == refcLoader) {
        return true;
    }
    if (refcLoader == null && typeLoader != null) {
        return false;
    }
    if (typeLoader == null && type.getName().startsWith("java.")) {
        return true;
    }
    final String name = type.getName();
    Class<?> res = java.security.AccessController.doPrivileged(
        new java.security.PrivilegedAction<Class<?>>() {
            public Class<?> run() {
                try {
                    return Class.forName(name, false, refcLoader);
                } catch (ClassNotFoundException | LinkageError e) {
                    return null; // Assume the class is not found
                }
            }
        };
    return (type == res);
}
```

Figure 2 method `isTypeVisible`

```java
void checkForTypeAlias(Class<?> refc) {
    if (isInvocable()) {
        MethodType type;
        if (this.type instanceof MethodType)
            type = (MethodType) this.type;
        else
            type = this.type = getMethodType();
        if (type.erase() == type) return;
        if (VerifyAccess.isTypeVisible(type, refc)) return;
        throw new LinkageError("bad method type alias: " +type+" not visible from "+refc);
    } else {
        Class<?> type;
        if (this.type instanceof Class<?>)
            type = (Class<?>) this.type;
        else
            type = type = getClassType();
        if (VerifyAccess.isTypeVisible(type, refc)) return;
        throw new LinkageError("bad field type alias: " +type+" not visible from "+refc);
    }
}
```

Figure 3 method `checkForTypeAlias`
References and sources of information

[3] http://hg.openjdk.java.net/jdk7u/jdk7u/jdk/rev/6e34c6d3479f (Contains the broken 2013 patch code by Oracle)

http://www.securityweek.com/oracle-reissues-patch-two-year-old-java-flaw
http://www.oracle.com/technetwork/java/javase/8u77-relnotes-2944725.html
https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2016-0636
https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2013-5838