Security Vulnerability Notice

SE-2014-02-GOOGLE-2

[Google App Engine Java security sandbox bypasses, Issue 2 #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 2 hasn't been properly addressed by Google. As a result, arbitrary instances of `java.net.URLClassLoader` class can be still created in GAE. This can be further exploited to gain a complete GAE Java security sandbox escape. Below, more details are provided with respect to the improperly implemented patch.

Issue 2 exploits the implementation of a whitelisted `java.security.Provider.Service` class for a system Class Loader instantiation [1]. As part of the fix for the problem, Google modified the implementation of a `Service.Provider` class and added an invocation of a security check method to its `getImplClass` method. This is illustrated below:

```
private Class getImplClass() throws NoSuchAlgorithmException {
    try {
        Reference reference = classRef;
        Class class1 = reference != null ? (Class)reference.get() : null;
        if (class1 == null) {
            ClassLoader classloader = provider.getClass().getClassLoader();
            if (classloader == null)
                class1 = Class.forName(className);
            else
                class1 = classloader.loadClass(className);
            if (!Modifier.isPublic(class1.getModifiers())) {
             ...
            }
            classRef = new WeakReference(class1);
        }
        checkImplClass(class1);          <---- CALL TO SECURITY CHECK
        return class1;
    } catch(ClassNotFoundException classnotfoundexception) {
  ...
  }
}
```

The `checkImplClass` method verifies the caller of a `newInstance` method of the `Security.Provider` class. This is conducted by the means of the inspection of Java stack frames pushed onto the call stack. The name of the class and a method of a stack frame corresponding to the caller of the `newInstance` method is checked against the set of allowed class / method name pairs (`ALLOWED_NEW_INSTANCE_CALLERS`). If a caller is not included in this set, a `SecurityException` is thrown:

```
private static void checkImplClass(Class class1) {
    ...
    StackTraceElement ste;
    JavaLangAccess javalangaccess = SharedSecrets.getJavaLangAccess();
    int i = 1;
    do {
        StackTraceElement tmp =
                      javalangaccess.getStackTraceElement(throwable, i);
        if (!tmp.getClassName().equals("java.security.Provider$Service")) {
            ste = tmp;
            break;
        }
        i++;
    } while(true);
    String s = ste.getClassName()+"."+ste.getMethodName();
    if (!ALLOWED_NEW_INSTANCE_CALLERS.contains(s)) {   <---- SECURITY CHECK
     throw new SecurityException("User code cannot construct "+class1);
    }
    ...
 }
```

The described implementation makes it easy to bypass the security check of the checkImplClass method. All that's needed to accomplish that is to call the newInstance method of a Provider.Service class from within the allowed class and method. In our Proof of Concept code, the com.sun.net.ssl.SSLSecurity class and its getImpl1 method are used for that purpose:

```
package com.sun.net.ssl;

import java.lang.*;
import java.net.*;
import java.security.*;

public class SSLSecurity {
 public static ClassLoader getImpl1(Provider.Service ps,URL utab[]) {
  ClassLoader cl=null;

  try {
   cl=(ClassLoader)ps.newInstance(utab);
  } catch(Throwable t) {}

  return cl;
 }
}
```

In Java, different implementations of a class with the same name can coexist in different Class Loader namespaces[1]. This is also the reason, why the above implementation of com.sun.net.ssl.SSLSecurity class can be successfully defined in GAE although its original definition exists in a bootstrap Class Loader namespace.

The ability to create arbitrary instances of URLClassLoader class can be easily exploited to gain a complete GAE Java security sandbox escape. Issue 2 can be combined with Issues 19 and 22 for that purpose. It's worth to note that Issue 19 was evaluated by Google as *working as intended* (WAI) issue. Issue 22 should have been fixed[2], but this hasn't been done. It is likely due to Google's vulnerability evaluation methodology focused on a root cause tracking. We have warned Google[3] that by focusing on the so called root cause, it could easily miss an innocent vulnerability that may turn out to be helpful in a future attack. We didn't need to wait long for that to happen.

Attached to this report, there is a Proof of Concept code that illustrates the impact of the reported issue. It has been successfully tested in a production GAE environment patched against security vulnerabilities we reported to Google in Dec 2014 / Jan 2015.

REFERENCES

[1] "Google App Engine Java security sandbox bypasses", technical report
http://www.security-explorations.com/materials/se-2014-02-report.pdf
[2] The Java Virtual Machine Specification, Java SE 7 Edition
http://docs.oracle.com/javase/specs/jvms/se7/html/

[1] Class Loader constraints are enforced when method / field references are resolved, not when classes are defined in a JVM [2].
[2] a status report from Google received on 04-Mar-2015 stated that all issues, except Issue 21, are fixed and shouldn't work anymore [3].
[3] we did it on 27-Dec-2014 at the time of providing Google with arguments regarding WAI Issues 17-20. These arguments have been also presented in our technical report (2.4.1.2 Closing thoughts).

[3] SE-2014-02 Vendors status
http://www.security-explorations.com/en/SE-2014-02-status.html

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.