Security Vulnerability Report

SE-2012-01-PUBLIC

[Security vulnerabilities in Java SE, Issue 22]

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 the latest version of Apple QuickTime software and its Java extensions in particular. It is similar to the two vulnerabilities reported to Apple in year 2007. At that time 16 security issues in Java extensions of Apple QuickTime software were reported to the company. The following three issues were among them (quotes taken from original reports):

1] disabling security checks relying on hasSecurityRestrictions() call

This new vulnerability allows to completely disable security checks implemented by the Apple QuickTime Java extension classes. It has its origin in quicktime.QTSession class. By issuing the following code sequence:

QTSession.initialize(0,Class.forName(name_of_some_signed_class));

all security checks relying on a static call to hasSecurityRestrictions() of QTSession class will be simply turned off.

In a result of successful exploitation, the vulnerability could be used to completely bypass applet sandbox restrictions

2] quicktime.util.QTByteObject initialization security checks bypass

It is possible to create partially initialized (though fully functional) instances of QTByteObject class by hijacking its finalize() method. In a result arbitrary read/write access to process memory can be gained.

3] quicktime.util.QTByteObject initialization security checks bypass (2)

It is possible to create partially initialized (though fully functional) instances of QTByteObject class through serialization. In a result arbitrary read/write access to process memory can be gained."

Apple addressed the abovementioned issues in the following way:

- security check verifying for all permissions (fully trusted code) was implemented in a static class initializer method of `quicktime.QTSession` class,
- `reallyLeakObject` method was introduced to the code. Its goal was to prevent leaking of a reference to the partially initialized object of `quicktime.util.QTByteObject` subclass via `finalize` method call,
- security check was added to the `readObject` method call verifying that a serialized class was coming from the system classloader namespace.

Security Explorations discovered that all of the abovementioned checks could be successfully bypassed. The checks for the two `QTByteObject` issues could be bypassed by simply combining them together. They are at the origin of a security vulnerability (Issue 22), which is a subject of this report.

In order to understand the nature of a discovered vulnerability, one needs to be familiar with certain implementation details of Apple's `QTByteObject` class such as those denoted below:

- `QTByteObject` class can be potentially subclassed by an untrusted (unsigned) code. This is due to the use of a `protected` keyword for one its initializer methods:

```
protected QTByteObject(int i) {
    _doSC();
    if(i > 0)
        bytes = new byte[i];
}
```

- instantiation of `QTByteObject` class is guarded by a security check implemented in a `_doSC` method. This method makes sure that any `QTByteObject` instance is created by a trusted (signed) Java code only. If not, a reference to current, partially initialized `QTByteObject` is put into a static list of objects (`reallyLeakObject` call):

```
private final void _doSC() {
    if (QTSession.hasSecurityRestrictions())
        if (getClass().getClassLoader() != null) {
            if (!getClass().getName().startsWith("quicktime.")) {
                reallyLeakObject(this);
                throw new SecurityException(getClass().getName() + " cannot
be constructed with current security settings");
            }
```

  The goal of the check is to keep a reference to any partially initialized `QTByteObject` "live". This will prevent JVM's Garbage Collector from collecting the object and from calling attacker's provided `finalize` method in particular (the source of Issue 2 from 2007):
- `readObject` method contains a security check that verifies whether a call to it is done in a context of a trusted (signed) Java class from a system class loader namespace:

```
private final void readObject(ObjectInputStream objectinputstream)
    throws IOException, ClassNotFoundException {
    if (QTSession.hasSecurityRestrictions() &&
        getClass().getClassLoader()!=
        Class.forName("quicktime.util.QTByteObject").getClassLoader()) {
            throw new SecurityException(getClass().getName() +
                " cannot be deserialised with current security settings");
    } else {
        objectinputstream.defaultReadObject();
```

```
                return;
            }
        }
```

The goal of the check is to prohibit the stealing of a partially initialized `QTByteObject` reference from within the overloaded, attacker's provided implementation of `readObject` method (the source of Issue 3 from 2007),

- `QTByteObject` class implements certain unsafe native methods such as `getIntAt` or `setIntAt`. These methods are unsafe as they allow for arbitrary read / write memory access past Java object bounds.

Although Apple's implementation of security checks for 2007 `QTByteObject` issues addressed each of them, it did not take into account the situation where both issues could be combined together to achieve same security bypass condition.

In our Proof of Concept code we instantiate a specially crafted subclass of a `QTByteObject` class with the use of serialization:

```
public static void get_mqtbo() {
 try {
  try {
   ByteArrayInputStream bais=new ByteArrayInputStream(MyQTByteObject_body);
   ObjectInputStream ois=new ObjectInputStream(bais);
   ois.readObject();
  } catch(Throwable e) {}
  while(BlackBox.mqtbo==null) {
   Runtime.getRuntime().runFinalization();
   System.gc();
  }
  System.out.println("Created: "+BlackBox.mqtbo);
 } catch (Throwable e) {
  e.printStackTrace();
 }
}
```

The object that gets read from arbitrary input stream (`MyQTByteObject_body` table) is an instance of the following class:

```
public class MyQTByteObject extends QTByteObject {
 public void finalize() {
  BlackBox.mqtbo=this;
```

```
 }
 public MyQTByteObject() {
  super(0);
 }
 public void call_setIntAt(int i, int j) {
  super.setIntAt(i,j);
 }
 public int call_getIntAt(int i) {
  return super.getIntAt(i);
 }
 public void setBytes(byte tab[]) {
  bytes=tab;
 }
}
```

During deserialization of `MyQTByteObject` class, a call to its initializer method is never made. This is due to the way serialization works in Java. For objects implementing `java.io.Serializable` interface, this is a constructor method of a first non-serializable class that is called at the time of instantiating a deserialized object. In this particular case, it's a constructor method of `java.lang.Object` class.

As a result of the above, `_doSC` method never gets called as it is invoked from a constructor of `QTByteObject` class. Security checks implemented by `_doSC` method are thus not in effect and a reference to the partially initialized, but fully functional instance of `MyQTByteObject` class can be successfully obtained by an attacker from an overloaded `finalize` method call of `MyQTByteObject` class. Access to `QTByteObject` class can be again exploited to gain arbitrary read/write access to the target process memory by the means of unsafe `getIntAt` or `setIntAt` native methods.

IMPACT

Described Issue 22 is not sufficient to implement a functional and successful attack code in the environment of Java and QuickTime software. Security Explorations discovered another issue in Java SE coming from Oracle that allows to do this. More specifically, Issue 15 [1] reported to Oracle on Apr 2, 2012 allows to beat additional security check implemented in Apple code, so that functionality of QuickTime for Java can be called from unsigned Applets.

Issues 15 and 22 when combined together allow for a complete compromise of JVM security sandbox. The exploitation scenario makes use of the possibility to read / write process memory. Full sandbox bypass is achieved via a properly setup type confusion condition. Attached to this report, there is a Proof of Concept code that illustrates this. It has been

successfully tested in a Windows OS environment and with the latest versions of Java SE 6 and QuickTime software installed:

- JRE/JDK 6u31 (version 1.6.0_31-b05)
- QuickTime 7.7.1 (version 7.71.80.42)

In order to test the attached Proof of Concept code, additional setup of the QuickTime environment needs to be done. This is required for the purpose of mimicking Issue 15 (undisclosed and not yet patched vulnerability in non-Apple software). Please, see the README file for instructions.

The code targets 32-bit Java Plugin only (the default for 32-bit web browsers) and Apple QuickTime 7.7.1. It has been successfully tested across the following Java SE, OS and web browsers combinations (all with latest patches applied):

- Windows XP SP3, Windows 7 HP 64-bit, Windows 7 Pro 32-bit,
- Mozilla Firefox 11.0, Internet Explorer 9.0, Opera 11.62,
- JRE/JDK 6 Update 31.

We were unable to exploit the issue in a 64-bit JRE environment. This was due to the fact that 32-bit web browsers require 32-bit JRE and do not start 64-bit Java at all. For 64-bit Internet Explorer 9.0 and JRE Plugin, no QuickTime Java extensions were visible to the 64-bit Java VM.

VENDOR'S RESPONSE

On Apr 12 2012, Security Explorations sent a vulnerability notice to Apple containing detailed information about a discovered vulnerability. Along with that, the company was also provided with source and binary codes for a Proof of Concept codes illustrating the impact of a security issue found.

On Jun 12, 2012, Apple released Java security updates for MacOS [2] which among other things incorporated additional security fix for the QuickTime issue. Regardless of that, Apple's publication neither contained any information about the addressing of a QuickTime issue, nor the company notified us about the upcoming release of a Java security update for MacOS.

When inquired by Security Explorations with respect to the "silent fix / no credit" approach taken, on Jun 18, 2012, Apple responded that a reported issue was considered to be a "hardening issue", rather than a security bug. Apple's reasoning behind such an evaluation was based on the fact that "the issue depended on another issue already fixed and credited by Oracle".

Contrary to Apple's belief, the depending Oracle's Issue 15 has not been fixed yet. Oracle's Java CPU released on Jun 12, 2012 [3] fixed only 4 out of almost 30 security issues that were reported to the company in Apr 2012. And Issue 15 was not among them. We have made that information available in our FAQ on Jun, 13 2012 [4].

The tests conducted on Jun 19, 2012 clearly show that users of a fully patched Windows 7 OS that have latest Java (1.6.0_33-b03) and Apple QuickTime (7.72.80.56) software installed are still at risk of being successfully exploited. Our proof of concept code for Oracle's issues 15 and Apple's issue 22 demonstrated a complete Java sandbox bypass on Firefox 13.0.1, Opera 12 and IE 9.

In response to Apple's evaluation of Issue 22, Security Explorations provided the company with its arguments. We emphasized that Apple QuickTime issue allowed for a bypass of a security check in Apple's code. As a result, unsafe memory access condition could be created. This alone is a serious violation of Java VM security sandbox and is usually sufficient to classify a given issue as a security bug, rather than a "security hardening enhancement". We also indicated that Issue 22 was a combination of two old issues. In the past, such issues were classified by Apple as security bugs, not "hardening issues". Apple did communicate to the public corresponding fixes for these issues and also gave proper credits to the reporting researcher [5][6][7].

On Jun 22, 2012 Apple delivered a response to our message and did confirm its stance. The company informed that it considered "issues that were not sufficient by themselves to lead to a security compromise to be security hardening enhancements".

REFERENCES

[1] SE-2012-01 Project, Security Vulnerabilities in Java SE, http://www.security-explorations.com/en/SE-2012-01-press.html

[2] About the security content of Java for OS X 2012-004 and Java for Mac OS X 10.6 Update 9, http://support.apple.com/kb/HT5319

[3] Oracle Java SE Critical Patch Update Advisory - June 2012, http://www.oracle.com/technetwork/topics/security/javacpujun2012-1515912.html

[4] SE-2012-01 Frequently Asked Questions, http://www.security-explorations.com/en/SE-2012-01-faq.html

[5] Archived - About the security content of QuickTime 7.2, http://support.apple.com/kb/TA24829?viewlocale=en_US

[6] Archived - About the security content of QuickTime 7.3, http://support.apple.com/kb/HT2047?viewlocale=en_US

[7] Archived - About the security content of QuickTime 7.4.5, http://support.apple.com/kb/HT1241

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.