Vulnerability Fix Experiment

SE-2012-01-ORACLE-7

[Security vulnerabilities in Java SE, Issue 50]

DISLAIMER

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.

As a result of Oracle response received recently regarding the lengthy and complex CPU preparation process, we have decided to conduct an experiment in order to verify how hard it is to fix the vulnerability reported to the company on Sep 25, 2012. Below, we are providing you with the results of our analysis.

Issue 50 is caused by insecure implementation of a serialization functionality implemented by `com.sun.corba.se.impl.io.ObjectStreamClass` class. The problem with `com.sun.corba.se.impl.io.ObjectStreamClass` class stems from the fact that it can reuse information about class fields layout declared in an incompatible class and cached by the `translateFields` method in a `translatedFields` hashtable.

A quick look at the `translateFields` method reveals the following implementation:

```
    private static Object[] translateFields(Object objs[])

        throws NoSuchFieldException {

        try{

            java.io.ObjectStreamField fields[] = (java.io.ObjectStreamField[])objs;

            Object translation[] = null;


            if (translatedFields == null)

                translatedFields = new Hashtable();


            translation = (Object[])translatedFields.get(fields);


            if (translation != null)

                return translation;

            else {

                Class osfClass = Class.forName("com.sun.corba.se.impl.io.ObjectStreamField");

                translation     =     (Object[])java.lang.reflect.Array.newInstance(osfClass,
objs.length);

                Object arg[] = new Object[2];

                Class types[] = {String.class, Class.class};

                Constructor constructor = osfClass.getDeclaredConstructor(types);

                for (int i = fields.length -1; i >= 0; i--){

                    arg[0] = fields[i].getName();

                    arg[1] = fields[i].getType();


                    translation[i] = constructor.newInstance(arg);

                }
                translatedFields.put(fields, translation);

            }

            return (Object[])translation;
        }
        catch(Throwable t){
            NoSuchFieldException nsfe = new NoSuchFieldException();
            nsfe.initCause( t ) ;
            throw nsfe ;
        }
    }
```

The above implementation is buggy because exactly same `fields` values will result in same `translation` values regardless of the classes serial persistent fields are declared at.

In order to block our attack, we decided to change the `translateFields` method in the following form. Instead of using the `fields` value for the hashtable lookup, we decided to use the actual class value from which the persistent fields are to be translated from. For that purpose, we need an additional argument to the `translateFields` method and a minor modification to its implementation as presented below. All new changes / additions to the code are marked with a green color.

```
    private static Object[] translateFields(Class c,Object objs[])
        throws NoSuchFieldException {
        try{
            java.io.ObjectStreamField fields[] = (java.io.ObjectStreamField[])objs;
            Object translation[] = null;

            if (translatedFields == null)
                translatedFields = new Hashtable();

            translation = (Object[])translatedFields.get(c);

            if (translation != null)
                return translation;
            else {
                Class osfClass = Class.forName("com.sun.corba.se.impl.io.ObjectStreamField");

                translation     =     (Object[])java.lang.reflect.Array.newInstance(osfClass,
objs.length);

                Object arg[] = new Object[2];

                Class types[] = {String.class, Class.class};

                Constructor constructor = osfClass.getDeclaredConstructor(types);

                for (int i = fields.length -1; i >= 0; i--){

                    arg[0] = fields[i].getName();

                    arg[1] = fields[i].getType();


                    translation[i] = constructor.newInstance(arg);

                }

                translatedFields.put(c, translation);


            }


            return (Object[])translation;

        }

        catch(Throwable t){

            NoSuchFieldException nsfe = new NoSuchFieldException();

            nsfe.initCause( t ) ;

            throw nsfe ;

        }

    }
```

Similarly, due to the fact that an extra argument was introduced to the `translateFields` method, we need to change the way in which this method is invoked from the code implementing `com.sun.corba.se.impl.io.ObjectStreamClass`. This is illustrated below:

```
            AccessController.doPrivileged(new PrivilegedAction() {

                public Object run() {

                /* Fill in the list of persistent fields.

                 * If it is declared, use the declared serialPersistentFields.

                 * Otherwise, extract the fields from the class itself.

                 */

                try {
                    Field pf = cl.getDeclaredField("serialPersistentFields");
                    // serial bug 7; the serialPersistentFields were not
                    // being read and stored as Accessible bit was not set
                    pf.setAccessible(true);
                    // serial bug 7; need to find if the field is of type
                    // java.io.ObjectStreamField
                    java.io.ObjectStreamField[] f =
                           (java.io.ObjectStreamField[])pf.get(cl);
                    int mods = pf.getModifiers();
                    if ((Modifier.isPrivate(mods)) &&
                        (Modifier.isStatic(mods)) &&
                        (Modifier.isFinal(mods)))
                    {
                        fields                                                =
(ObjectStreamField[])translateFields(cl,(Object[])pf.get(cl));
                    }
                } catch (NoSuchFieldException e) {
                    fields = null;
                } catch (IllegalAccessException e) {
                    fields = null;
                } catch (IllegalArgumentException e) {
                    fields = null;
                } catch (ClassCastException e) {
                    /* Thrown if a field serialPersistentField exists
                     * but it is not of type ObjectStreamField.
                     */
                    fields = null;
                }
```

The described fix has the following advantages:

- it is relatively quick as it requires 25 bytes of modifications to be applied to the source code of `com.sun.corba.se.impl.io.ObjectStreamClass`,
    - 12 characters need to be removed from the code,
    - 13 characters need to be added to the code.
- the fix introduces an instance of a system, difficult to hijack `java.lang.Class` class as the source for the hashtable lookup operation,

- serial persistent fields translations are now cached with respect to the classes where they are actually declared,
- the fix does not seem to require any integration tests with other Oracle application software:
    - code logic is not changed at all, translation method operates in the same way, it just uses a different object for the hashtable lookup operation,
    - minor changes are applied only to two private methods,
    - none of the changes influences any additional variables beyond those modified by the original code,
    - none of the changes go beyond the private scope of the modified class.

Attached to this report, there is a modified source code of `com.sun.corba.se.impl.io.ObjectStreamClass` originating from OpenJDK 7 that implements the described fix. Along with that, we also provide its compiled version in a form of the output class files and the diff for the source code of a target class.

We verified that by replacing original class files from the `rt.jar` binary of JRE 7 Update 9 with those implementing the fix, our Proof of Concept code for Issue 50 did not work anymore.

At the end, we would like to indicate that it took us less than half an hour to implement and verify the presented fix (start time 22:37 is the time of the OpenJDK7 source code archive download, end time 23:03 is the time of the original `rt.jar` file replacement).

We hope that the results of this quick experiment sufficiently challenges Oracle and that this analysis leads to the verification of the company's statements, especially the one claiming the need for four additional months to implement and release a security update for a critical security vulnerability in Java (Issue 50), which we believe (and are hopefully correct with respect to the analysis conducted) can be fixed within less than half an hour time.

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.