Analýza Java 0-day (CVE-2012-4681)

WWMAD?

Nič ..?

Ľubomír Nagajda
Ústav Informatiky, UPJŠ v Ke

Java?

Java Platform, Enterprise Edition alebo v skratke Java EE je platforma pre programovanie aplikácií na strane servera v jazyku Java. Okrem štandardných vlastností a knižníc podporuje prácu s technológiami JavaBean, Java Messaging Service (JMS), Persistence, atď.

Servlety?

Servlet je trieda v Java EE, definovaná podľa Java Servlet API - protokolu, ktorý dokáže spracovať a odpovedať na HTTP požiadavky.

Takto môže tvorca webovej aplikácie využiť programovací jazyk Java na tvorbu dynamických webových aplikácií.

Generovaný obsah je zvyčajne štandardná HTML stránka, no sú podporované aj ostatné druhy obsahu, definované podľa HTTP (content-type).

O chod a životný cyklus servletu sa stará tzv. Web Container.

Web Container je komponent serveru, ktorý interaguje so serverom a zabezpečuje zavedenie a zrušenie servletu, bezpečnosť, súbežnosť a ostatné podobné služby(Tomcat).

Java Applet?

Applet doručený používateľovi ako Java bytecode. Možno spustiť vo webovom prehliadači pomocou JVM.

Využíva aj 3D akceleráciu a je porovnateľne rýchli s inými program. Jazykmi ako C++, no niekoľkonásobne rýchejší ako JavaScript.

Applety sú cross-platform, preto môžu byť vykonané prehliadačom rôznej platformi.

Takisto možnosť púštať v offline režime.

Gondvv

Nádherný exploit navrhnutý 28.augusta 2012.

Krása tohto exploitu je práve v tom, že jeho spoľahlivosť je 100percentná a je samozrejme multiplatformový.

Jeho budúce využitie bude pravdepodobne rozsiahle a nesmie chýbať v bežnej výbave pentestera.

V podstate exploit je vytvorenie inštancie *java.security.AccessControlContext* a *java.security.ProtectionDomain*, ktorý má úplné privilégiá a potom nahradiť AccessControlContext ištanciou java.beans.Statement, aby sme boli schopní spustiť kód s plnými oprávneniami.

Java.beans.Statement

Volanie metódy getContext nastaví AccessControlContext na kontext apletu , ktorý má obmedzenia s limitovanou ProtectionDomain, ktora samozrejme nie je privilegovaná vôbec.

```
public class Statement {

private static Object[] emptyArray = new Object[]{};

static ExceptionListener defaultExceptionListener = new ExceptionListener() {
 public void exceptionThrown(Exception e) {
  System.err.println(e);
  System.err.println("Continuing ...");
 }
};

private final AccessControlContext acc = AccessController.getContext();
private final Object target;
private final String methodName;
private final Object[] arguments;
ClassLoader loader;

[...]

}
```

Stroj času a.k.a Jaro Slavik

Vrátime sa o 2 roky, kedy Sami Koivu zverejnil analyzu Java problému ktorý sa týkal hlavne Java Runtime.

Využime príkoria tejto doby a ozrejmime si, kde nastal problém a ako sa vyriešil *AccessControlContext*.

Dedenie dôveryhodnosti

-Java kód pochadzajúci z hlavných tried – dôveryhodný

-Java kód z Java appletov – nedôveryhodný ..rly?

-Keď Java rieši práva pre danú operáciu, predpokladá oprávnenia pre celý zásobnik – call stack

-privilégia métód su privilégiami tried, kde su metódy definované.

Predstavme si, že trieda A definuje metódu call() a trieda B je subclass A, pričom B je na stacku s metódou call(), tak privlégia su dedné od A, nie od B.

Takto dokážeme vytvárať reťazce metód, kde na konci bude privelégovana trieda, od ktorej sa bude dediť.

ZDI-10-056/CVE-2010-0840 1.

-vytvorme vuln.Link (subclass z java.beans.Expression) použime invoke() s parametramy class: java.lang.System, method: setSecurityManager, args: null

- Vytvorme nový java.util.HashSet a vložme vuln.Link do Set-u

- Vytovorme anonymnu podtredu HashMap-y kde entrySet vráti inštanciu definovanú vyššie

- Vytvorme javax.swing.JList objekt, obsahujúci inštanciu HashMap definovanú vyššie

- Pridajme Jlist do componentu (ako napr applet)

ZDI-10-056/CVE-2010-0840 2.

- Vlákno GUI chce použiť JList a volá JList.paint

- JList pri vykreslovaní, zavolá toString na zistenie obsahu, vrátane spomínanej anonymnej subclassy HashMap-y

- HashMap zdedí z AbstractMap toString metódu, ktorá volá entrySet().Iterator() a používa GetValue na každom i-vstupe.

- Jeden z vstupných objektov je subclass zo Statement, ktorá má implementovanú metódu Statement.getValue ()

- Expression.getValue () volá Statement.invoke ()

- Statement.invoke () bol nastavený tak. Aby volal System.setSecurityManager (null)

==Java Security je vypnutá

= Fix? = AccessControlContext namiesto Statement.invoke()

AccessControlContext Fix

Ak ak pridanie AccessControlContext field do Statement triedy rieši predošlú zraniteľnosť, ako chceme teraz zmeniť private field?

sun.awt.SunToolkit

Trik je tu použiť sun.awt.SunToolkit triedu, ktorá obsahuje veľmi zaujímavú verejnú statickú metódu:

```
 public static Field getField(final Class klass, final String fieldName) {
return AccessController.doPrivileged(new PrivilegedAction<Field>() {
 public Field run() {
  try {
   Field field = klass.getDeclaredField(fieldName);
   assert (field != null);
   field.setAccessible(true);
   return field;
 }
  catch (SecurityException e) {
   assert false;
 }
  catch (NoSuchFieldException e) {
   assert false;
 }
 return null;
 }
});
}
```

AccessControlException

Je potrebné si uvedomiť, že triedy niektorých balíčkov sú obmedzené na appety a nemožno k nim pristupovať, alebo ich použiť.

Sú to balíčky

- com.sun.deploy.*
- com.sun.imageio.*
- com.sun.javaws.*
- com.sun.jnlp.*
- com.sun.xml.internal.bind.*
- com.sun.xml.internal.ws.*
- Sun.*

Snaha vytvoriť inštanciu alebo používať triedy v týchto balíčkoch bude mať za následok AccessControlException.

AccessControlException

To znamená, že nie sme schopní dostať odkaz na sun.awt.SunToolkit triedu z našeho applet-u.

Ale základná vec, ktorú potrebujeme vedieť, je, že volanie niektorých metód môže potenciálne obísť SecurityManager kontrolujúci závislosti na práve zavolanej triede

- http://www.oracle.com/technetwork/java/seccodeguide-139067.html#ref-1 Guideline 9-8

Chyby vhodné pre Gondvv

Áno, presne tak.. chyby využité v exploite ktorý som spomenl na žaciatku su rovno 2.

Prvá sa zneužíva na získanie referencie na triedu sun.awt.SunToolkit.

Druhá sa zneužíva na zavolanie verejnej metódy GetField na tejto triede.

Exploit využíva java.beans.Expression čo je subclass java.beans.Statement.

Statement.invokeInternal

Keď je volaná metóda Expression.execute, skončí sa volanie Statement.invokeInternal metódy.
Preto sa poďme pozrieť na Ḱód...

<<Statement.invokeInternal >>

A

```
static Method getMethod(Class<?> type, String name, Class<?>... args) {
        try {
                return MethodFinder.findMethod(type, name, args);
        }
        catch (NoSuchMethodException exception) {
                return null;
        }
}
```

Všimnite si metódy com.sun.beans.finder.ClassFinder.resolveClass a com.sun.beans.finder.MethodFinder.findMethod

1, com.sun.beans.finder. ClassFinder.findClass

Statement.invokeInternal metóda volá com.sun.beans.finder.ClassFinder.resolveClass a ak sa pozrieme na jeho realizáciu budeme vidieť, že to skončí volaním metódy com.sun.beans.finder.ClassFinder.findClass

```
public static Class<?> resolveClass(String name, ClassLoader loader) throws ClassNotFoundException {
        Class<?> type = PrimitiveTypeMap.getType(name);
        return (type == null) ? findClass(name, loader): type;
}
```

```
public static Class<?> findClass(String name) throws ClassNotFoundException {
        try {
                ClassLoader loader = Thread.currentThread().getContextClassLoader();
                if (loader == null) {
                        loader = ClassLoader.getSystemClassLoader();
                }
                if (loader != null) {
                        return Class.forName(name, false, loader);
                }
        } catch (ClassNotFoundException exception) {
        // use current class loader instead
        } catch (SecurityException exception) {
        // use current class loader instead
        }
return Class.forName(name);
}
```

Guideline 9-9:=>ClassLoader

Debugovanie appletu

ClassLoader – súčasť JDK = dôveryhodné!!!

Debug - java.lang.Class - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Debug

Gondvv [Remote Java Application]
Java HotSpot(TM) 64-Bit Server VM[localhost:6789]
Thread [thread applet-Gondvv.class] (Suspended)
Class<T>.forName(String) line: 186
ClassFinder.findClass(String) line: 72
ClassFinder.findClass(String, ClassLoader) line: 106
ClassFinder.resolveClass(String, ClassLoader) line: 166
Expression(Statement).invokeInternal() line: 213 [local variables unavailable]
Statement.access$000(Statement) line: 58
Statement$2.run() line: 185 [local variables unavailable]
AccessController.doPrivileged(PrivilegedExceptionAction<T>, AccessControlContext) line: not available [native method]
Expression(Statement).invoke() line: 182 [local variables unavailable]
Expression.execute() line: 121 [local variables unavailable]
Gondvv.GetClass(String) line: 38
Gondvv.SetField(Class, String, Object, Object) line: 46

Variables | Breakpoints

Name	Value
arg0	"sun.awt.SunToolkit" (id=39)

Gondvv.java | Class.class | Statement.class | Expression.class | MethodFinder.class | AbstractFinder.class | ClassFinder.class

```
178     *             specified name.
179     * @exception LinkageError if the linkage fails
180     * @exception ExceptionInInitializerError if the initialization provoked
181     *            by this method fails
182     * @exception ClassNotFoundException if the class cannot be located
183     */
184    public static Class<?> forName(String className)
185                throws ClassNotFoundException {
186        return forName0(className, true, ClassLoader.getCallerClassLoader());
187    }
188
189
190    /**
191     * Returns the {@code Class} object associated with the class or
192     * interface with the given string name, using the given class loader.
193     * Given the fully qualified name for a class or interface (in the same
194     * format returned by {@code getName}) this method attempts to
195     * locate, load, and link the class or interface.  The specified class
196     * loader is used to load the class or interface.  If the parameter
197     * {@code loader} is null, the class is loaded through the bootstrap
198     * class loader.  The class is initialized only if the
199     * {@code initialize} parameter is {@code true} and if it has
200     * not been initialized earlier.
201     *
202     * <p> If {@code name} denotes a primitive type or void, an attempt
203     * will be made to locate a user-defined class in the unnamed package whose
```

Outline

useCaches : boolean
addAll(Collection<Field>, Field[]) : void
argumentTypesToString(Class<?>[]) : String
arrayContentsEq(Object[], Object[]) : boolean
checkInitted() : void
copyConstructors(Constructor<U>[]) <U> : Constructor
copyFields(Field[]) : Field[]
copyMethods(Method[]) : Method[]
desiredAssertionStatus0(Class<?>) : boolean
forName(String) : Class<?>
forName(String, boolean, ClassLoader) : Class<?>
forName0(String, boolean, ClassLoader) : Class<?>
getPrimitiveClass(String) : Class
getReflectionFactory() : ReflectionFactory
isAsciiDigit(char) : boolean
registerNatives() : void
searchMethods(Method[], String, Class<?>[]) : Method
toClass(Type) : Class<?>
annotations : Map<Class<? extends Annotation>, Anno

Read-Only | Smart Insert | 186 : 1

2, MethodFinder.findMethod

Výzva k findAccessibleMethod skončí volaním java.lang.Class.getMethods a volajúci v zásobníku je com.sun.beans.finder.MethodFinder, ktorý je dôveryhodný, pretože je súčasťou JDK tak obchádzame bezpečnostné kontroly.

```
public static Method findMethod(Class<?> type, String name, Class<?>...args) throws NoSuchMethodException {
        if (name == null) {
                throw new IllegalArgumentException("Method name is not set
        }
        PrimitiveWrapperMap.replacePrimitivesWithWrappers(args);
        Signature signature = new Signature(type, name, args);

        Method method = CACHE.get(signature);
        if (method != null) {
                return method;
        }//
        method = findAccessibleMethod(new MethodFinder(name, args).find(type.getMethods()));
        CACHE.put(signature, method);
return method;
}
```

Debugovanie

Debug - java.lang.Class - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Debug

Thread [thread applet-Gondvv.class] (Suspended)
- Class<T>.getMethods() line: 1426 [local variables unavailable]
- MethodFinder.findMethod(Class<?>, String, Class<?>...) line: 70
- Statement.getMethod(Class<?>, String, Class<?>...) line: 357
- Expression(Statement).invokeInternal() line: 261
- Statement.access$000(Statement) line: 58
- Statement$2.run() line: 185
- AccessController.doPrivileged(PrivilegedExceptionAction<T>, AccessControlContext) line: not available [native method]
- Expression(Statement).invoke() line: 182
- Expression.execute() line: 121
- Gondvv.SetField(Class, String, Object, Object) line: 47
- Gondvv.disableSecurity() line: 30
- Gondvv.init() line: 53
- AppletViewerPanel(AppletPanel).run() line: 434
- Thread.run() line: 722

Variables | Breakpoints

Name	Value
this	Class<T> (sun.awt.SunToolkit) (id=45)

Gondvv.java | Class.class | Statement.class | Expression.class | MethodFinder.class | AbstractFinder.class

```
1418     *             </ul>
1419     *
1420     * @since JDK1.1
1421     */
1422    public Method[] getMethods() throws SecurityException {
1423        // be very careful not to change the stack depth of this
1424        // checkMemberAccess call for security reasons
1425        // see java.lang.SecurityManager.checkMemberAccess
1426        checkMemberAccess(Member.PUBLIC, ClassLoader.getCallerClassLoader());
1427        return copyMethods(privateGetPublicMethods());
1428    }
1429
1430
1431    /**
1432     * Returns an array containing {@code Constructor} objects reflecting
1433     * all the public constructors of the class represented by this
1434     * {@code Class} object.  An array of length 0 is returned if the
1435     * class has no public constructors, or if the class is an array class, or
1436     * if the class reflects a primitive type or void.
1437     *
1438     * Note that while this method returns an array of {@code
1439     * Constructor<T>} objects (that is an array of constructors from
1440     * this class), the return type of this method is {@code
1441     * Constructor<?>[]} and <em>not</em> {@code Constructor<T>[]} as
1442     * might be expected.  This less informative return type is
1443     * necessary since after being returned from this method, the
```

Outline

- getField0(String) : Field
- getFields() : Field[]
- getGenericInfo() : ClassRepository
- getGenericInterfaces() : Type[]
- getGenericSignature() : String
- getGenericSuperclass() : Type
- getInterfaces() : Class<?>[]
- getMethod(String, Class<?>...) : Method
- getMethod0(String, Class<?>[]) : Method
- getMethods() : Method[]
- getModifiers() : int
- getName() : String
- getName0() : String
- getPackage() : Package
- getProtectionDomain() : ProtectionDomain
- getProtectionDomain0() : ProtectionDomain
- getRawAnnotations() : byte[]
- getResource(String) : URL
- getResourceAsStream(String) : InputStream

Read-Only | Smart Insert | 1426 : 1

SUM

WWMED:

-Vytvorí inštanciu Statement, ktorá bude volať System.setSecurityManager(null)

-Vytvorí vlastný AccessControlContext s úplnými oprávneniami.

-S jednou chybou, dostane odkaz na triedu sun.awt.SunToolkit, ktorá je obmedzená na applety.

-S druhou chybu, zavolá GetField, verejnú statickú metódu na sun.awt.SunToolkit s využitím dôveryhodného priameho volajúceho, ktorá obchádza bezpečnostné kontroly.

-S metódou GetField dostane odkaz na private field Statement.acc a nastaví jeho hodnotu na vlastnú, skôr vytvorenú inštanciu AccessControlContext.

-Nakoniec sa vykoná Statement, ktorý zakáže Security Manager-a, pretože má plné oprávnenie uvedené v jeho AccessControlContext.

THX4ALL

Otázky, či hinty smerujte na l.nagajda@gmail.com

Ďakujem za pozornosť.

1337!