Exploiting Memory Corruption Vulnerabilities in the Java Runtime

Joshua J. Drake
Inaugural DerbyCon
October 2\textsuperscript{nd} 2011
About the Presenter

• Joshua J. Drake, aka jduck
  – Employed with Accuvant LABS
    • Research
      – Vulnerabilities & Exploitation
    • Consulting
      – Binary/Source Audit, Reverse Engineering
  – metasploit® Contributor
    • Formerly Lead Exploit Developer
Overview

- Background
- Hurdles
- Exploiting
- Demos
- Conclusion
Motivation

- ...share information and techniques to make Java Runtime Environment (JRE) exploitation easier.
  
  - JRE architecture information
  - Various hurdles encountered during dev
    - i.e. CVE-2009-3867, CVE-2009-3869
  - Provide tools for future work
Background

- Why Java?
- Popular?
- Maybe a ‘lil.

More claims here:
http://www.java.com/en/about/

That's a lot of great targets...
pic.twitter.com/SFbyyyko
• Java is cross-platform!

Java SE Runtime Environment 6 Update 27

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

- Accept License Agreement
- Decline License Agreement

<table>
<thead>
<tr>
<th>Product / File Description</th>
<th>File Size</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux x86 - RPM Installer</td>
<td>20.10 MB</td>
<td>jre-6u27-linux-i586-rpm.bin</td>
</tr>
<tr>
<td>Linux x86 - Self Extracting Installer</td>
<td>20.63 MB</td>
<td>jre-6u27-linux-i586.bin</td>
</tr>
<tr>
<td>Linux x64 - RPM Installer</td>
<td>19.67 MB</td>
<td>jre-6u27-linux-x64-rpm.bin</td>
</tr>
<tr>
<td>Linux x64 - Self Extracting Installer</td>
<td>20.25 MB</td>
<td>jre-6u27-linux-x64.bin</td>
</tr>
<tr>
<td>Solaris x86 - Self Extracting Binary</td>
<td>20.27 MB</td>
<td>jre-6u27-solaris-i586.sh</td>
</tr>
<tr>
<td>Solaris SPARC - Self Extracting Binary</td>
<td>25.10 MB</td>
<td>jre-6u27-solaris-sparc.sh</td>
</tr>
<tr>
<td>Solaris SPARC 64-bit - Self Extracting Binary</td>
<td>11.19 MB</td>
<td>jre-6u27-solaris-sparcv9.sh</td>
</tr>
<tr>
<td>Solaris x64 - Self Extracting Binary</td>
<td>7.48 MB</td>
<td>jre-6u27-solaris-x64.sh</td>
</tr>
<tr>
<td>Windows x86 Kernel</td>
<td>0.87 MB</td>
<td>jre-6u27-windows-i586-iftw-k.exe</td>
</tr>
<tr>
<td>Windows x86 Online</td>
<td>0.87 MB</td>
<td>jre-6u27-windows-i586-iftw.exe</td>
</tr>
<tr>
<td>Windows x86 Offline</td>
<td>16.12 MB</td>
<td>jre-6u27-windows-i586.exe</td>
</tr>
<tr>
<td>Windows x64</td>
<td>16.39 MB</td>
<td>jre-6u27-windows-x64.exe</td>
</tr>
</tbody>
</table>
• Java SE 6 focus
  – Tested latest (6u27)
  – JRE 7 GA is released!
    • Buggy!
  – Slow adoption...
27 updates over about 5 years
Well over 100 CVEs
Targeted in 73% of exploit kits

10 exploits in metasploit®
- 4 Windows specific
- 1 meatware attack (java_signed_applet)
- 3 involve memory corruption
What does the “JRE” include?

http://java.sun.com/products/hotspot/whitepaper.html - Recommended Reading
• Java has a plentiful attack surface!
  – Browser Plug-in
    • Automatically installed
    • Applets
      – 70% of Metasploit Java exploits use Applets
    • “LiveConnect” Java/Browser interface
  – Java Web Start & JNLP
  – More
• Attackers use applets because...
  – Applet Java code and JAR contents are 100% attacker controlled
  – Tons of native library code is reachable
    • Images, Sounds, Compressors and more
    • Includes embedded copies of open source (zlib, etc)

<table>
<thead>
<tr>
<th>Trusted</th>
<th>Untrusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signed</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Runs with full user privileges</td>
<td>Subject to Java “sandbox”</td>
</tr>
<tr>
<td>User is Prompted</td>
<td>No prompting</td>
</tr>
</tbody>
</table>
Java Virtual Machine (JVM)
- Named “HotSpot”
- Written in native code
- Processes Java Bytecode
- Might just-in-time compile
- Executes or Interprets resulting code
• Process Architecture
  – Plug-in loads in Browser address space
    • Includes several libraries
  – Since Update 10
    • Java.exe runs as an external process
    • Can Pass options to Java.exe via HTML

  – Still no DEP
  – Still no ASLR
• All JRE 6 releases ship same msvcr71.dll
  – v7.10.3052.4
    • md5 86f1895ae8c5e8b17d99ece768a70732

• Loads in all components!
  – Browser itself
  – Java.exe for applets

• Public ROP chains target this DLL
Two major kinds of heaps

- Java Object heap (more in a sec)
- Native heap (from msvcr71.dll)
  - msvcr7.dll implements malloc too, nothing imports it
  - Just a wrapper around HeapAlloc
    - OS-specific allocator security properties apply
      » ASLR
      » Safe-unlinking
      » Meta-data validation
      » etc

Someone had fun!
Background - Technical

• Java Object heap
  – Garbage Collected
  – Allocated via VirtualAlloc
  – Was Read/Write/Execute until update 18 !!
  – Predictable address
    • Between 0x22000000 and 0x26000000
    • Due to “Class Data Sharing” ??
Hurdles

Joshua J. Drake
Inaugural DerbyCon
October 2\textsuperscript{nd} 2011
• Debugging JVM started from browser
• Process terminates out from under you! – Surprise!
• Why does this happen?

```
ntdll!DbgBreakPoint: 7709000c cc int 3 0:043> g
(c28.8b4): Single step exception - code 80000004 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00010220 ebx=77094180 ecx=000002f0 edx=77080000 esi=770901f0 edi=770921e0
eip=770b016e esp=07b7fa90 ebp=07b7fb0c iopl=0 nv up ei ng nz na pe cy
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000287
ntdll!LdrpSnapThunk+0x1c1:770b016e 03c2 add eax,edx 0:046> g
eax=00000000 ebx=00000000 ecx=00000000 edx=00000000 esi=77182100 edi=771820e0
eip=7709fcb2 esp=0354fde8 ebp=0354fde8 iopl=0 nv up ei pl zr na pe nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000246
ntdll!NtTerminateProcess+0x12:7709fcb2 83c404 add esp,4
```

Continue after a while
Single step exception?!
Oh no! Process DIED!
• Java Plugin Watchdog
  – Watches over external jp2launcher.exe process

Java_java_lang_ProcessImpl_destroy
(inside java.dll)

TerminateProcess
Hurdles - Watchdog

• Prevent the watchdog from interfering!

1. Patch up the “java.dll” binary
   – NOP out the TerminateProcess call
   – Or just change JNZ -> JMP

2. Use breakpoints, runtime patching, etc
   – Must be done each execution 😞
Hurdles - Watchdog

```
; Exported entry 149. __Java_java_lang_ProcessImpl_destroy@8
; Attributes: bp-based frame
; stdcall Java_java_lang_ProcessImpl_destroy(x, x)
public __Java_java_lang_ProcessImpl_destroy
__Java_java_lang_ProcessImpl_destroy@8 proc near
var 4 = dword ptr -4
arg_0 = dword ptr 8
arg_4 = dword ptr 0Ch
push ebp
mov ebp, esp
push ecx
push edx
push ecx
push [ebp+arg_4]
push [ebp+arg_0]
call JNU_GetFieldByNum@20 ; JNU_GetFieldByNum(x,x,x,x)
cmp byte ptr [ebp+arg_4+3], 0
mov [ebp+var_4], edx
ret

Breakpoint settings
```

- Location: 0x6D3280DS
- Condition: !str(!"y") && SetRegValue(0x6D3280ed0, \"eip\") && 0
- Settings: Enabled
- Actions: Break
- Hardware: Off
- Module relative: Off
- Symbolic: Off
- Low level condition: Off

Hardware breakpoint mode:
- Read/Write
- Size: 0x1
• Spurious access violations while debugging
• Not sure why... Let’s speculate.
  – Expected AV in JIT’d code?
  – Crap code wrapped in catch-all handler?
  – If you know or have another idea, speak up!

• Just pass and pretend its not happening ;-P
• Java uses UTF-8 for all strings
  – Invalid sequences replaced with ‘?’

• Check this out: (from @mihi42)

```java
public class OMGWTF {
    public static void main(String[] args) throws Exception {
        /*
        \u006a\u0075\u006e\u006b\u0079\u002a\u002f
        \u0053\u0079\u0073\u0074\u0065\u006d\u002e
        \u006f\u0075\u0073\u0074\u002e\u0074\u0063\u0068\u0069\u0066\u0065
        \u0073\u0072\u0069\u0062\u006c\u006f\u0072\u0069\u006e\u0067
        \u0074\u002e\u0074\u0063\u0068\u0069\u006e\u0073
        \u002e\u0063\u006f\u006e\u0066\u0069\u006e\u0067
        \u002e\u006f\u006e\u0065\u0063\u0072\u0065\u006e\u0069
        \u006e\u0067
        */
    }
}
```
Hurdles - Encoding

- Compile and run it...

```
fear:0:~$ javac OMGWTF.java; java OMGWTF
How?
fear:0:~$  
```

- But it was all comments?!

```
fear:0:~$ native2ascii -reverse OMGWTF.java
public class OMGWTF {
    public static void main(String[] args) throws Exception {
        /*
         * junky*/
        System.out.println("How?");
    } /* by
    @mihi42
    */
}
```

- Java pre-processes those UTF escapes!
• Don’t use strings! Use arrays
  – Their values are represented in memory contiguously

• Better, but there’s still an issue...
Hurdles – Integers

- In Java, all integers are signed!

- Use next larger type
  - For 0xff byte, use short integer
  - For 0xffff short, use long integer
  - etc
Hurdles - Reachability

• Code that seems unreachable at first
  – Was the case in CVE-2009-3869

• You can reach more by using Java tricks
  – Sub-classing
  – Reflection
  – Abusing complex interfaces
    • i.e. A class that takes a instance as a parameter
Exploiting
(yay)
• Used a custom JNI (vuln_jni.dll) for testing
  – Covers several common exploit primitives

```c
JNIEXPORT void JNICALL Java_Vuln_ArbCall
  (JNIEnv * env, jobject obj, jint addr) {
}

JNIEXPORT void JNICALL Java_Vuln_Write4
  (JNIEnv * env, jobject obj, jint what, jint where) {
}

JNIEXPORT jstring JNICALL Java_Vuln_sprintf
  (JNIEnv * env, jobject obj, jstring string) {
}

JNIEXPORT void JNICALL Java_Vuln_hsprintf
  (JNIEnv * env, jobject obj, jstring string) {
}
Exploiting: Arbitrary Call

- Fun and simple..
  - Just need somewhere to jump!

```java
JNIEXPORT void JNICALL Java_Vuln_ArbCall
  (JNIEnv * env, jobject obj, int addr) {
    void (*f)(void) = (void (*)(void))addr;
    f();
  }
```

- Good thing JRE 6 doesn’t support ASLR!
  - Public ROPs work great
- Nor does it support DEP!
  - Let’s jump into a DLL .data section!
Here’s the code:

```c
JNIEXPORT jstring JNICALL Java_Vuln_sprintf
  (JNIEnv * env, jobject obj, jstring string) {
    jboolean copied;
    const char *str = (*env)->GetStringUTFChars(env, string, &copied);
    char buf[1024];
    sprintf(buf, str);
    return (*env)->NewStringUTF(env, buf);
}
```

Two issues in this function
– CWE-121: Stack Buffer Overflow
– CWE-134: Uncontrolled Format String
One of my personal favorites

Java’s C runtime has “%n” disabled
  – (Un)fortunately?

May still be useful
  – Leak memory contents
  – Cause buffer overflows (%1024xAAAAABBBBB)
• Pet peeve: NOT A STACK OVERFLOW

• Traditional methods can be tricky due to UTF8 issues
  – Just pad with stuff and control EIP
  – Some characters still aren’t usable

• CVE-2009-3867 / CVE-2009-3869
Exploiting: Write4

• Surgical!
  – Need to target something used for control flow
  • Must know it’s address (within margin of error)

```java
JNIEXPORT void JNICALL Java_Vuln_Write4
  (JNIEnv * env, jobject obj, int what, int where) {
  int *p = (int *)where;
  *p = what;
}
```

• A plethora of stuff to surgically overwrite
  – Again, lack of ASLR / DEP FTW
Exploiting: Heap BOF

• Heap Buffer Overflow
  – Depends on what you corrupt!

• Unlikely to overflow Java Object Heap data
  – An interesting area to research =)

• Native heap protections make for pain and suffering.
Exploiting: CVE-2009-3867

- **getSoundbank file:// URI Stack BOF**
  - Affects JRE $\leq 6u16, 5u21, 1.4.2\_24, 1.3.1\_26$
- **KF’s PoC showed cross-platform PC control**
- **metasploit** version
  - Passes “np” & “sc” applet PARAMs
    - Nops and Shellcode – allows cross-platform targeting
  - Sprays the Java Object Heap
  - Overwrites saved PC (no SEH)
  - Jumps to Java Object Heap (was still RWX)
Exploiting: CVE-2009-3869

- **setDiffICM Stack BOF**
  - Similar to previous (exec’s Java Object Heap)
- **Native Method:** `Java_sun_awt_image_ImageRepresentation_setDiffICM`
  - Called from `ImageRepresentation.setPixels`
- **sun.awt.* can’t be used in an Applet!**
  - `java.security.AccessControlException: access denied` (java.lang.RuntimePermission accessClassInPackage.sun.awt.image)
- **Using a custom ImageFilter we can!**
Demos!

Joshua J. Drake
Inaugural DerbyCon
October 2\textsuperscript{nd} 2011
Conclusions

• Exploiting JRE 6 can be painful, but...
• It’s easier than it should be.
  – Well behind the mitigation curve
    • No ASLR or DEP
    • Predictable memory layout
  – Vast attack surface
  – Buggy

• Check out the examples!
Recommendations

• **Good:**
  – Use EMET to force ASLR and DEP
  – Prepare for migration to JRE 7
  – Use 64-bit browser / plug-in

• **Better:**
  – Disable browser plug-ins and JNLP/Web Start
    • Chrome neuters Java by default

• **BEST: UNINSTALL JRE !!**
Future Directions

• Mapping Java code constructs to Native-land
  – How does scope translate?

• Investigate JIT Spraying
  – Code region is RWX!

• More work with JRE 7
  – Does the new ASLR/DEP opt-in really help?
ANY QUESTIONS?

Feel free to contact me...

- @jduck1337
- IRC: jduck
- Email: jdrake [circled-a] Accuvant.com
- Email: jduck [circled-a] metasploit.com
References

Slide 3
http://kelseywinterkorn.com/

Slide 7
http://weblogs.java.net/blog/chet/archive/2007/05/consumer_jre_le.html
http://adtmag.com/articles/2011/08/01/java-7-crashing.aspx

Slide 8

Slide 9

Slide 10
https://twitter.com/#!/ifindkarma/status/115962954301714432

Slide 12

Slide 13
http://www.oracle.com/technetwork/java/javase/system-configurations-135212.html

Slide 16
http://download.oracle.com/javase/6/docs/technotes/guides/vm/class-data-sharing.html

Slide ?
http://www.oracle.com/technetwork/java/javase/index-135519.html
http://www.oracle.com/technetwork/java/javase/jre-install-137694.html
http://www.oracle.com/technetwork/java/javase/releasenotes-136954.html
– Update 10
  • New browser plug-in
    – Always installed (no custom install options)
    – Runs an external java.exe process
    – Allows controlling heap size
    – Allows selecting JRE version
  • Patch-in-place or Static

– Update 18
  • Java Heap no longer RWX!
  • Auto-updater a separate package (can remove)

– Prompt changes?