

Java 安全编程

(Bad Examples found in JDK)

Marc Schönefeld, University of
Bamberg

Illegalaccess.org



 X'con 2005

关于演讲者

◆ Marc Schönefeld, Diplom-Wirtschaftsinformatiker

◆ **For Science:** External doctoral student @ Lehrstuhl für praktische Informatik at University of Bamberg, Bavaria, Germany

◆ 课题项目:

REFACTORING OF SECURITY ANTIPATTERNS IN
DISTRIBUTED JAVA COMPONENTS

◆ **For Living:** Department for Operational Security Management at computing site for large financial group in Germany

◆ Java, J2EE, CORBA [CSMR 2002]

◆ 设计和开发

◆ 安全加固 (代码审核)



情形

- ◆ Java (我们这里讨论适用于J2SE,一些方面也适用于J2EE)
 - ◆ 被设计成一种具有与生俱来安全特性的编程语言 [Gong, Oaks]
 - ◆ JVM 级: 类型安全(Type Safety), 字节码完整性检测(Bytecode integrity checks)
 - ◆ API 级: SecurityManager, ClassLoader, CertPath, JAAS
 - ◆ 加密支持: JCA/JCE, JSSE
 - ◆ 所以, 什么会是问题呢?



一些2003/2004年的 Java 安全警告 FL2

- ◆ Java 运行时环境(Runtime Environment)可能允许非受信 Applets **提升权限** FL3
- ◆ Java Media Framework (JMF)中的一个漏洞,可能导致Java 虚拟机(JVM) **崩溃**
- ◆ ...Java Runtime Environment 远程拒绝服务攻击漏洞 (DoS) ...
尽管存在**JAVA**安全体系的先天防护,还是有许多潜在的攻击可能性

...

是什么原因导致的呢?



幻灯片 4

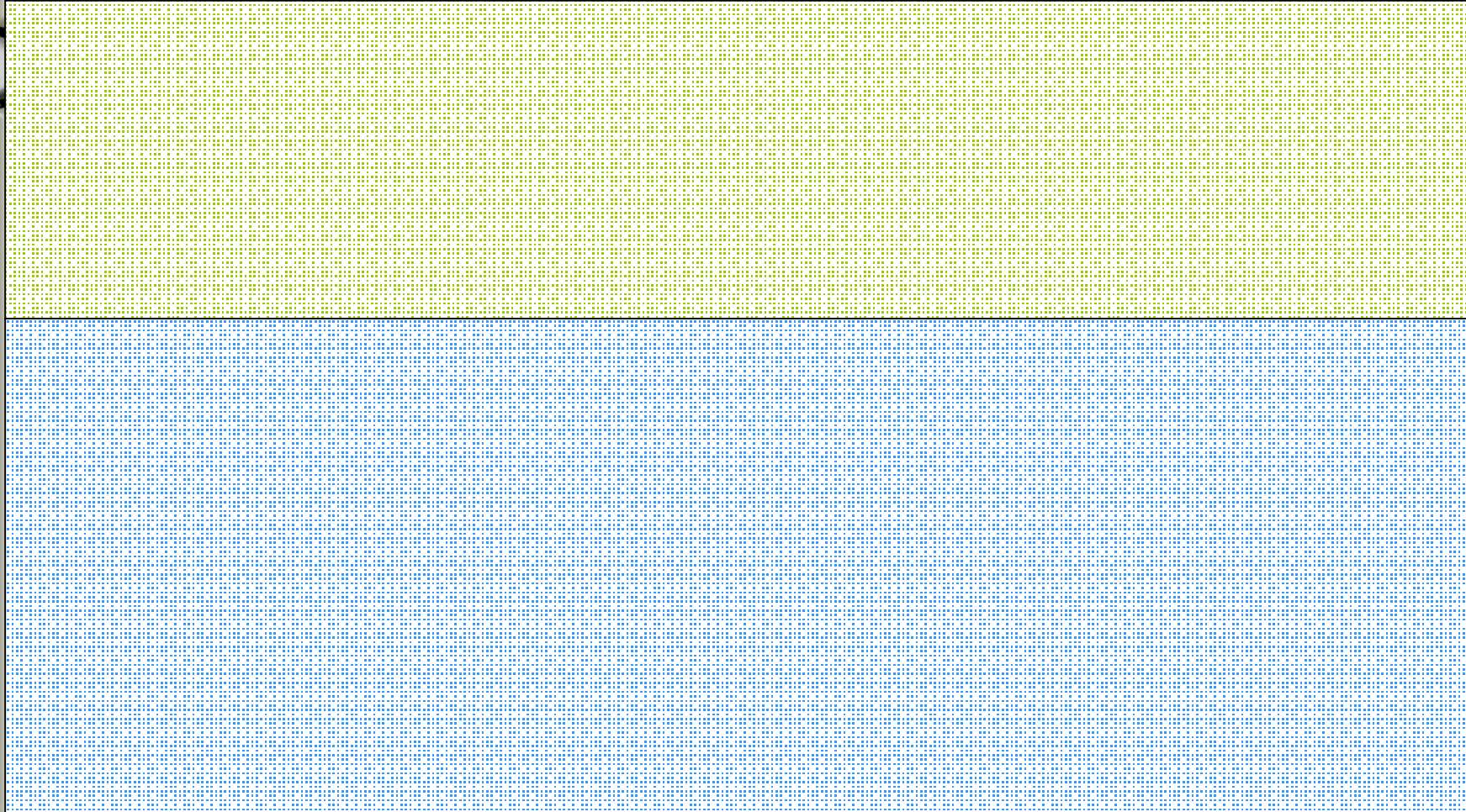
- FL2 警告
Flier Lu, 2005-8-16
- FL3 不被信任
Flier Lu, 2005-8-16

问题

- ❖ 一个平台(象Java runtime environment) 只能提供对程序员意图的支持
- ❖ 什么是程序员的意图？表现出不同的观点 ...
 - ❖ **功能** [应用程序员]
 - ❖ Java 有一套很强大的预定义的API函数 (sockets, files, ...)
 - ❖ **质量** 和 **重用** [中间件程序员]
 - ❖ Java在不同的语义级别提供通讯和编组 (marshalling) 能力(Sockets, RMI, CORBA, Raw-Serialisation, XML-Serialisation, ...)
 - ❖ **安全性** [安全架构师]
 - ❖ Java 在沙箱(sandbox)外提供隔离、加密和安全Sockets等支持
 - ❖ **恶意的意图** [敌人]
 - ❖ 通过找到 **弱点** 来破坏安全
- ❖ Java VM 和核心的lib库有 (许多?) **漏洞**



Classloaders 和保护域



为什么在JAVA代码中找安全漏洞？

❖ 基于组件的软件开发

- ❖ 第三方的中间件组件(web服务器, 图像库, PDF生成, ...) 随处可见
- ❖ 我们在受信任的地方(根类型加载器 boot classloader)重用它们
- ❖ 但是我们真的能信任它们吗?

❖ 疑问:

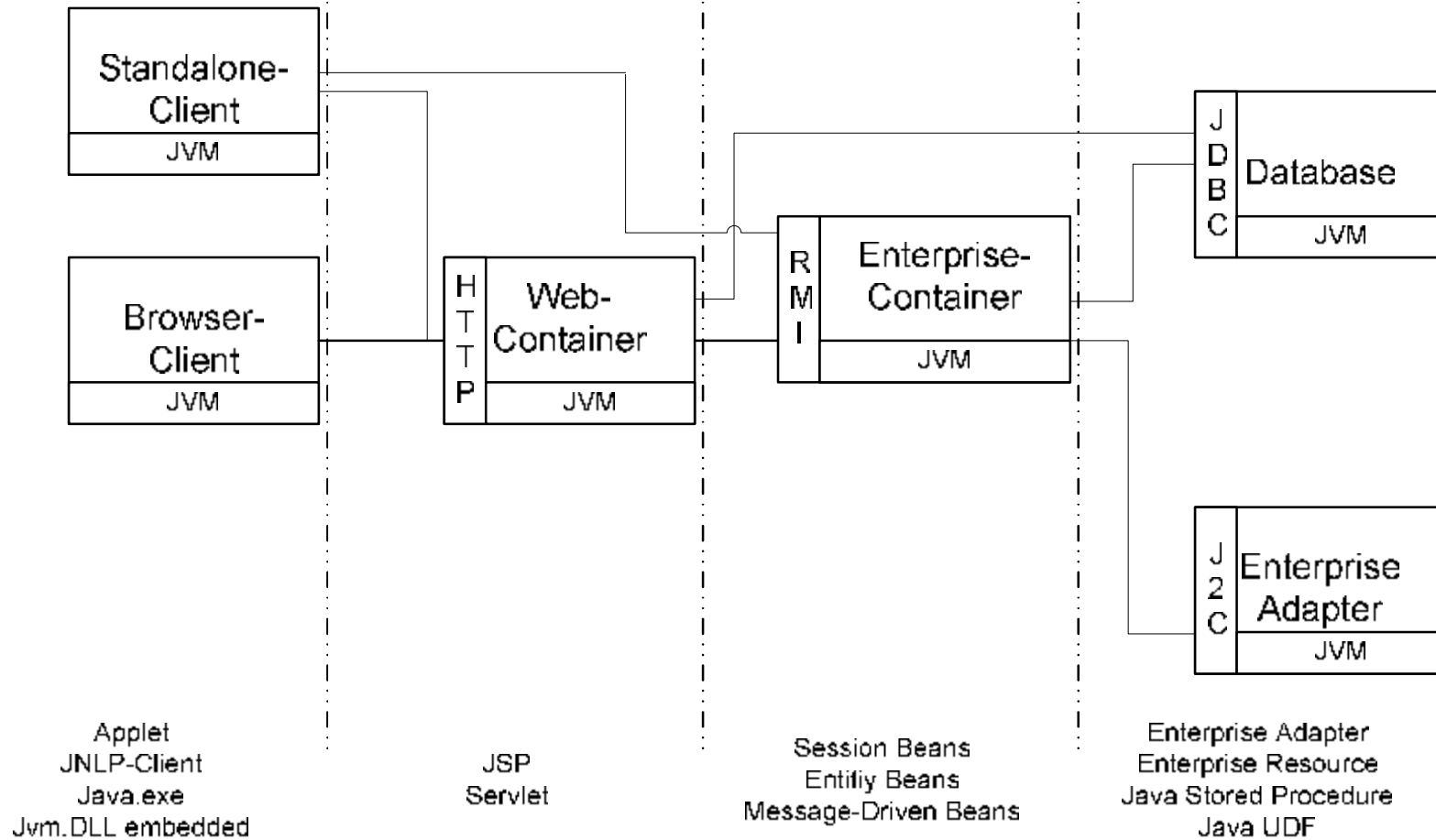
- ❖ 我的第三方图像库中是否存在可能被攻击者利用的存在漏洞的对象实现?
- ❖ JDK是否把我隔离存放的机密XML数据, 与其他存在恶意代码的 applets 加载到同一个JVM中?
- ❖ 对象序列化(serialisation)真的安全么?



J2EE 多层应用类型

X'con 2005

Client Tier Presentation Tier Enterprise Tier Backend Tier



J2EE 多层攻击类型

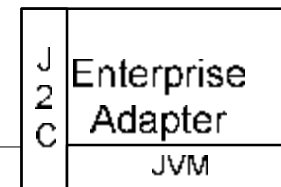
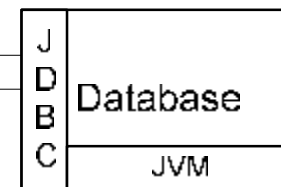
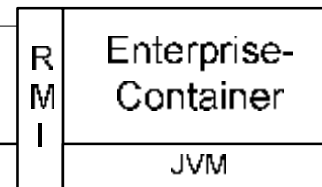
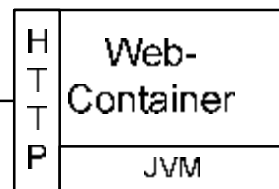
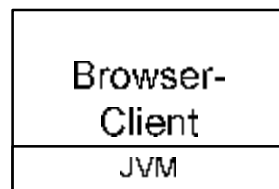
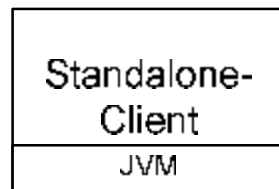
Client Tier

Presentation Tier

Enterprise Tier

Backend Tier

数据注入攻击 (SQL, 遗留格式)



Evil Twin Attack

拒绝服务攻击, 恶意序列化数据

Applet
JNLP-Client
Java.exe
Jvm.DLL embedded

JSP
Servlet

Session Beans
Entity Beans
Message-Driven Beans

Enterprise Adapter
Enterprise Resource
Java Stored Procedure
Java UDF

Java 安全模式

◆ Sun 的安全编码指导方针 (最后更新于2000年2月2日!):

1. 小心使用 **特权代码**
2. 小心处理 **静态字段**
3. 最小化作用域 (**scope**)
4. 小心选择 **公共 (public)** 方法和字段
5. 适当的 **包 (package)** 保护性
6. 尽可能使用 **不可变 (immutable)** 对象
7. 永远不要返回对包含敏感数据的内部数组的引用
8. 永远不要直接存储用户提供(**user-supplied**)的数组
9. 小心使用序列化(**Serialization**)
10. 小心使用本地方法(**native methods**)
11. 清除敏感信息

<http://java.sun.com/security/seccodeguide.html>



Java 安全反模式

- ❖ 忽略那些安全模式的代码不经意间就会造成漏洞
- ❖ 典型的 Java 安全编码反模式(Antipatterns):
 - ❖ 忽略语言特性 (例如整数溢出 (**Integer Overflow**))
 - ❖ 不注意使用序列化, 不注意使用 特权代码
 - ❖ 将字段和方法定义到不适当的可见性作用域
 - ❖ 在非终态 (non-final) 静态字段中的隐蔽通道(**Covert Channels**)
- ❖ 他们隐藏在你自己的代码和你使用的库中
- ❖ 出于学术兴趣,我们审查了 **Sun JDK 1.4.x** 的部分代码, 下面将介绍我们的发现:



如何在JAVA代码中找安全隐患？

源代码 检测工具	PMD , Checkstyle	只在源代码完全可用情况下有用 [大部分情况不是这样的]
反编译器	JAD (!), JODE	耗时的分析工作，且需要经验
字节码 审计分析器	Findbugs (基于 Apache BCEL)	字节码检测器 (访问者模式): ❖ 预定义 (软件质量) ❖ 自定义 (安全审计)
策略 评估工具	jChains (<a href="http://jchains.d
ev.java.net">http://jchains.d ev.java.net)	❖ 测试程序是否需要已指定的许 可权限，在逆向分析受保护域时 很有用。



字节码分析器

❖ 以下讨论基于JVM 字节码分析器

❖ **Findbugs** (<http://findbugs.sourceforge.net>)

❖ 针对java代码检测bug模式的静态检测器(detector)

❖ 由马里兰大学开发 (Puth 和 Hovemeyer)

❖ 开源项目

❖ 基于BCEL (Apache Bytecode Engineering Library)

❖ 通过访问者模式分析:

❖ 类的结构与继承

❖ 控制和数据流

❖ GUI/命令行界面

❖ 可扩展, 允许编写自己的检测器



Java 安全反模式

- ◆ 反模式 (错误, 缺陷) 在受信代码中 (例如 **rt.jar**) 导致漏洞出现
 - ◆ 可用性:
 - ◆ AP1: 整数溢出, 未知类型 (`java.util.zip.*`)
 - ◆ AP2: 序列化 (Serialisation) 的副作用 (`java.io.*`)
 - ◆ 完整性:
 - ◆ AP3: 特权代码副作用 (引诱打破沙箱的攻击)
 - ◆ AP4: 不恰当的作用域 (违反访问控制)
 - ◆ AP5: 非终态静态变量 (在applets之间的隐蔽通道)
 - ◆ 机密性:
 - ◆ AP6: 重用非安全组件 (`org.apache.*`, 在applets间嗅探私有的XML数据)
- ◆ 目标: 定义一个二进制审计工具集, 用来在你自己的代码和你使用的库中检测 **反模式** 以修复安全漏洞。

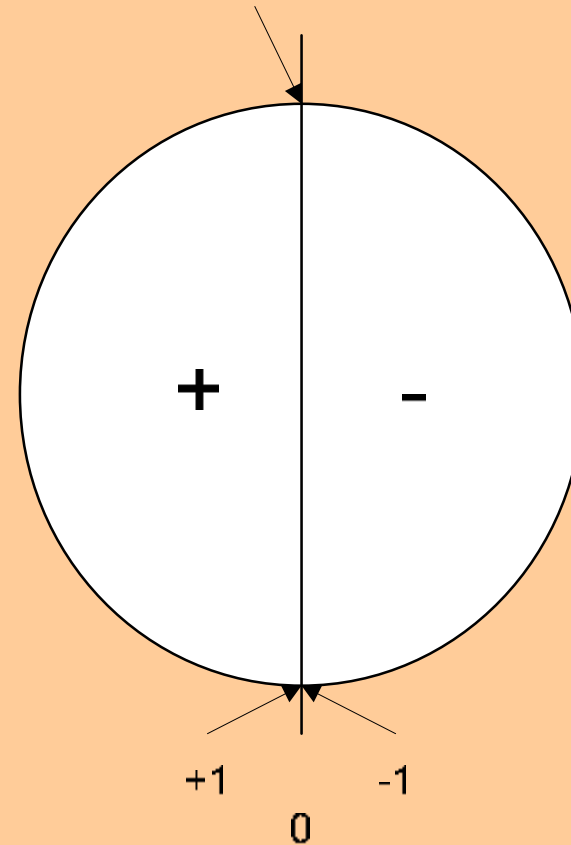


Java 反模式 1: 整数溢出

如 *blexim* (Phrack #60) 所述, 整数溢出在C/C++中是个严重的问题, 在Java中也同样如此:

- 所有Java 整数被限制在 $[-2^{31}, +2^{31}-1]$ 的范围内
- 在 Java中, 以下成立:
 $-2^{31} = 2^{31} + 1$
- 沉默溢出(Silent Overflow)是问题所在:
因为符号改变不会被报告给用户, 也没有JVM 标志被设置
- JDK **1.4.1_01** 代码是基于这样一个错误的假设: java 整数是极大的。这导致在 `java.util.zip` 包产生一系列的安全问题

`Integer.MIN_VALUE = Integer.MAX_VALUE + 1`



Java 反模式 1: 整数溢出

下列调用当参数 $x > y$ $x, y \geq 0$ 时, 会导致崩溃

```
tuple(new byte [0], x, Integer.MAX_VALUE - y)
```

è 可通过愚弄参数检查, 来使在受信JDK 函数中的数据沉默溢出 (silent overflow), 所以核心lib库和JVM都检测不到该溢出。

è 本地方法调用——**updateBytes** 访问字节数组将导致一个非法内存访问错误. 通常JVM会崩溃。

```
D:\ > java CRCCrash
An unexpected exception has been detected in native code outside the VM.
Unexpected Signal : EXCEPTION_ACCESS_VIOLATION occurred at PC=0 x6D3220A4
Function= Java_java_util_zip_ZipEntry_initFields+0x288
Library=c:\java\1.4.1\01\jre\bin\zip.dll
Current Java thread :
at java.util.zip.CRC32.updateBytes(Native Method )
at java.util.zip.CRC32.update(CRC32.java:53)
at CRCCrash.main(CRCCrash.java :3)
Dynamic libraries:
0x00400000 - 0x00406000 c:\java\1.4.1\01\jre\bin\java.exe
[... lines omitted ...]
0x76BB0000 - 0x76BBB000 C:\WINDOWS\System32\PSAPI.DLL
Local Time = Mon Mar 17 14:57:47 2003
```

Java 反模式 1: 整数溢出

CRC32 类允许计算缓冲区的校验和:

假如有一个字节缓冲区 (1,2,3,4), 要计算它的校验和, 你需要调用:

```
CRC32 c = new java.util.zip.CRC32 ();  
c.update (new byte []{1,2,3} ,0 ,3);
```

但是, 在你做以下操作的时候:

```
c.update  
3);
```

将导致 JDK 1.4.1_01 和JDK 1.3.1等版本的JVM 崩溃



Java 反模式 1: 整数溢出, 风险和范围

风险:

假如攻击者能在一个多用户共享JVM的环境中(例如 Domino 服务器或 Tomcat HTTP 服务器) 利用那些函数的话,那么就能造成拒绝服务攻击

范围:

更多受信函数被发现有漏洞:

- ◇ `java.util.zip.Adler32().update();`
- ◇ `java.util.zip.Deflater().setDictionary();`
- ◇ `java.util.zip.CRC32().update();`
- ◇ `java.util.zip.Deflater().deflate();`
- ◇ `java.util.zip.CheckedOutputStream().write();`
- ◇ `java.util.zip.CheckedInputStream().read();`
- ◇ `java.text.Bidi.<init >;`

◇ <http://developer.java.sun.com/developer/bugParade/bugs/4811913.html>

- ◇ also bugnr = {4811913, 4812181, 4812006 , 4811927 , 4811917, 4982415, 4944300, 4827312,4823885}



Java 反模式 1: 整数溢出, 重构

Before
JDK 1.4.1
01

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

After
JDK 1.4.1
02

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



Java 反模式 1: 整数溢出, 重构 (字节码)

Before (1.4.1_01)

```
12: iload_2
13: iflt 28
16: iload_3
17: iflt 28
20: iload_2
21: iload_3
22: iadd
23: aload_1
24: arraylength
25: if_icmple 36
```

整数溢出
字节码模式

After (1.4.1_02)

```
12: iload_2
13: iflt 28
16: iload_3
17: iflt 28
20: iload_2
21: aload_1
22: arraylength
23: iload_3
24: isub
25: if_icmple 36
```

重构后字节码



Java 反模式 1: 对有害的整数溢出, 如何在审核过程中发现它?

1. 通过侦测 `iadd` 指令来找潜在的问题方法
2. `iadd` 是否使用用户提供的数据 (使用的堆栈数据是否来源于 `iload`?) 来执行一个范围检查
3. 是否随后用同样的数据调用一个本地方法 (`invokevirtual`, `invokestatic`)

该过程可以使用 `Findbugs` 字节码检测器来实现



AP1: 结论和建议

❖ 不象一般的x86处理器那样(设计于1978年), JVM 没有提供一个溢出标志, 所以没有办法在运行的时候侦测这种情况。而在27年后设计的Java 1.5中 (aka 5.0 aka Tiger), JVM仍没有解决这个问题

❖ 对JDK 6.0的建议:

❖ 为了避免给程序员(没有安全意识的)增加负担, 有界基本整数类型(bounded primitive integer)会非常有帮助(象ada那样)。

subtype Month_Type is Integer range 1..12;

❖ 如果这对java编译器的处理来说太复杂的话, 至少也应提供编译警告列出潜在的溢出威胁。

(是否可能在Java 6.0中出现?)



反模式 2: 序列化的副作用

- ❖ 创建一个java对象的一般方法是使用 **new** 指令, 它将调用类的构造函数。
- ❖ 但是: Java 序列化 API (**java.io** 包的一部分) 允许跳过构造函数来创建此对象类型的新实例, 只需要简单的将数据发送到一个与socket, 文件或字节数组绑定的 **java.io.ObjectInputStream(OIS)**
- ❖ OIS 对象一般被远程通讯(如RMI)或持久化框架, 用于将预先构建的对象导入到JVM
- ❖ 当一个对象从一个OIS中被读出的时候, 大部分继承的 **readObject** 方法会被调用。



AP 2: 风险和范围

风险

- 读取序列化后的对象可能迫使JVM进入一个复杂的，容易出现漏洞代码的区域，而此过程会在 `readObject` 方法中被所调用。
- `readObject` methods 可能位于在你自己的代码，JDK 类中以及你所使用到的任何的第三方库中。
- 攻击者可能准备了手工处理过的数据包作为序列化数据。

范围

<code>java.util.regex.Pattern</code>	触发复杂的计算， „JVM may become unresponsive “ [Sun Alert 57707]
<code>java.awt.font.ICC_Profile</code>	在WIN32平台上导致JVM 崩溃
<code>java.util.HashMap</code>	触发一个未处理的 OutOfMemoryError 错误，它将终止当前监听的线程使服务失效。(作为一个错误，它将跳过 try/catch 的检查)

AP 2: 风险和范围

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

- 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
- Date Closed: 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

Match case

AP2:序列化的副作用, 重构

Before

JDK
1.4.2
05

```
private void readObject(java.io.ObjectInputStream
s)throws... {
    s.defaultReadObject();           // Initialize counts
    groupCount = 1;
    localCount = 0; // Recompile object tree
    if (pattern.length() > 0)
        compile(); // so we compile for the next 1600 years
    else
        root = new Start(lastAccept);
}
```

After

JDK
1.4.2
06

```
private void readObject(java.io.ObjectInputStream
s)throws... {
    s.defaultReadObject();           // Initialize counts
    groupCount = 1;                 // if length > 0,
    localCount = 0;                 // 使用延后(lazily)编译的模式
    compiled = false;
    if (pattern.length() == 0) {
        root = new Start(lastAccept);
        matchRoot = lastAccept;
        compiled = true;
    }
}
```



AP2:如何在审核过程中发现它?

1. 通过侦测 **readObject** 定义来找到潜在的问题类型。
2. 对于那些类来说, 看是否控制流落入到有害的代码中
 - I. 搜寻算法复杂度。(是否要用800年来编译一个正则表达式?)
 - II. 搜寻无穷循环 (bytecode backward branches)
 - III. 代码是否调用了易受攻击的native code 并且把全部或者部分的负荷传给了它?

Findbugs bytecode detector 可实现该过程。



AP2: 结论和建议

- ❖ **readObject** 方法主要被设计用来接受和检查 序列化 (**Serializable**) 的数据
- ❖ 嵌套的 **Serializable** 类型会发生嵌套的 **readObject** 调用, 所以恶意符合 (payload) 不必一定要在根对象中。
- ❖ 尝试将复杂操作从创建时推迟到第一次使用时
- ❖ 同样的规则也适用于 **readExternal** 方法, 该方法实现 **Externalizable** 接口的接收部分。



AP3: 特权代码的副作用

◆ 基本的Java 访问逻辑:

- ◆ 在如下情况，访问请求才被允许：当前执行上下文中，每个保护域都已经获得许可指定权限；换句话说，**每一个保护域中的代码和主体都获得指定权限。**
- ◆ 只有当所有的保护域 D_i 包含权限 p 的时候，权限才能被获得

$$p \in \left\{ \left[\begin{array}{c} n \\ | \\ D_i \\ | \\ i=1 \end{array} \right] \right\}$$



AP3:特权代码的副作用

- ❖ 特权代码(**doPrivileged**)可被用来跳过堆栈逻辑检查
- ❖ 在应用/用户(**user classes**)级需要的许可, 跟在执行必要的中间件/系统级(**rt.jar**)的许可是不一样的。

Graphics application	initializeDocument	Untrusted
A graphics routine	generateTmpFile	
Java.io.File	createTempFile	System
Java.io.File	checkAndCreate	
java.lang.Security.Manager	checkWrite	
java.lang.Security.Manager	checkPermission	
java.security.AccessController	checkPermission	
java.security.AccessControlContext	checkPermission	

Graphics application	initializeDocument	Untrusted
Some graphics library	generateSymbolFont	
Java.awt.Font	createFont	System
java.security.AccessController	doPrivileged	
Java.awt.Font\$1	run	
Java.io.File	createTempFile	
Java.io.File	checkAndCreate	
java.lang.Security.Manager	checkWrite	
java.lang.Security.Manager	checkPermission	
java.security.AccessController	checkPermission	
java.security.AccessControlContext	checkPermission	

AP3:特权代码的副作用: 风险和范围

风险

- ◆ 一个attacker可能滥用该条件来提升权限并且脱离受限的保护域(例如 JNLP 或 applet 沙箱)
 - ◆ 他知道JDK中的特权代码块,并且给应用程序的codesources以特权.
 - ◆ 通过一个引诱,攻击者试着诱骗控制到特权的代码块并且让该特权块使用他注射进来的payload

范围

<code>java.awt.font.ICC_Profile</code>	脱离applet 沙箱并测试客户端机器是否存在某文件.
<code>java.awt.Font (i)</code>	把临时文件(例如可执行文件)传输到客户端的机器上, 然后在合适的时候去执行 (http://www.derkeiler.com/Mailing-Lists/Full-Disclosure/2004-07/0462.html)
<code>Java.awt.Font(ii)</code>	使用内容为全零数据的大文件填充客户端机器的剩余磁盘空间
...

AP3:特权代码的副作用: 风险和范围

Full-Disclosure: [Full-Disclosure] IE sucks : sun java virtual machine insecure tmp file creation - Mozilla Firefox

http://www.derkeiler.com/Mailing-Lists/Full-Disclosure/2004-07/0462.html

Getting Started Latest Headlines

Der Keiler

ExtenXLS Java Reporting
Free Eval. Powerful, easy Java API Great Excel reports without Windows

Protect your Java Code
yGuard Ant Obfuscation Task (free) prevents decompilation of your jars

News:
Bugtraq:
MDKSA-2005-038 - Updated emacs/xemacs packages fix vulnerability

Home > Mailing-Lists > Full-Disclosure > 2004-07 News Newsgroups Service UNIX / Linux / Coding / Shop / Directory Privacy

[Full-Disclosure] IE sucks : sun java virtual machine insecure tmp file creation

From: Jelmert (jkuiperus_at_planet.nl)
Date: 07/09/04

- **Next message:** [Nick FitzGerald: "Re: \[Full-Disclosure\] No shell => secure?"](#)
- **Previous message:** [bipin gautam: "\[Full-Disclosure\] Re: Norton AntiVirus Scanner Remote DoS \[temp. FIX!\] \[Part: !!!\]"](#)
- **Next in thread:** [3APA3A: "\[Full-Disclosure\] Another IE trick \(Re: IE sucks : sun java virtual machine insecure tmp file creation\)"](#)
- **Reply:** [3APA3A: "\[Full-Disclosure\] Another IE trick \(Re: IE sucks : sun java virtual machine insecure tmp file creation\)"](#)
- **Messages sorted by:** [\[date \]](#) [\[thread \]](#) [\[subject \]](#) [\[author \]](#) [\[attachment \]](#)

To: full-disclosure@lists.netsys.com, bugtraq@securityfocus.com
Date: Fri, 09 Jul 2004 14:01:10 +0200

INTRODUCTION

Actually I wasn't really sure if I ought to post this, but after some consideration I decided that it might serve as an example of the completely messed up state we find internet explorer in today.

There's a very minor issue with the way the sun java virtual machine creates temporary files from applets. IE blows it off the chart, combining this with some unresolved issues in IE can lead to remote code execution.

Fertig

Start | 5 | 12 | 2 | 20 | t... | 3 | J... | 3 | A.. | 6 | W. | F.. | H.. | DE << 14:09

AP3: 重构

- ❖ 没有可用的修复
 - ❖ 前面描述的bugs依然存在在于JDK中，所以很不幸，没有可用的修复
 - ❖ 尽管在 Q2/2004 或者更早，那些漏洞就已报告给Sun



AP3:特权代码的副作用:如何审核?

1. 通过检查doPrivileged调用找出潜在的问题类型
2. 对于那些类,确认用户提供数据是否传递到特权代码块中?假如是的话会导致
 - I. 非受信任的代码访问到受保护资源
 - II. 把秘密数据泄露给非受信任的代码
 - III. 让非受信任代码执行不希望的修改

Findbugs 字节码检测器可以部分实现这个过程。



AP3: 结论和建议

结论

- ◆ **doPrivileged** 对保护域来说是 **强大的** 但是 **危险的** 双刃剑

建议

- ◆ 对 Sun:

- ◆ 请修复在JDK特权代码中的bugs

- ◆ 对组件用户:

- ◆ 在使用前,请检查第 三方库中的 **doPrivileged** 块,因为他们可能打破你制订的安全策略

- ◆ 对中间件开发者:

- ◆ 使你代码中的特权代码尽可能短小

- ◆ [\[http://java.sun.com/security/seccodeguide.html\]](http://java.sun.com/security/seccodeguide.html)

- ◆ 在传给特权代码前,请检查用户提供的数据有效性



AP4: 不恰当的作用域(Scope)

- ❖ 作为一条规则, 尽可能的减小方法和字段的作用域范围. 检查包私有 (*package-private*)成员是否应该是私有(*private*), 保护(*protected*)成员是否应该是包私有或私有等等. [Sun Security Code Guidelines]
- ❖ 当你设计一个受信的JDK扩展 (例如Java Media Framework (JMF)) 的时候, 该规则尤为重要。



AP4:不恰当的作用域: 风险和范围

❖ 风险

- ❖ 攻击者能利用受信的保护域([jre/lib/ext](#) 中的java扩展)的AllPermissions 来提升权限,例如 JMF
 - ❖ 安装额外受信的类到 [jre/lib/ext](#)
 - ❖ 通过本地(native)方法访问系统内存
 - ❖ 公共的 JMF 类 [com.sun.media.NBA](#) 暴露了一个指向物理内存的公共指针 [long value data]
 - ❖ 所以不受信任的applets也可以读你的系统内存



AP4:不恰当的作用域: 风险和范围

<http://classic.sunsolve.sun.com/pub-cgi/retrieve.pl?doc=fsalert%2F54760>

Support Documents

Security Information

- Latest Security Bulletin
- Security Bulletin Archive
- Security Sun Alerts
- Security T-Patches
- View T-Patches License

Jump to Applies To
Font Size [Increase] [Decrease]

document id	Synopsis	Date
54760	Java Virtual Machine (JVM) May Crash Due to Vulnerability in the Java Media Framework (JMF)	14 May 2003

Description

Sun(sm) Alert Notification

Top

1. Impact

A vulnerability in the Java(TM) Media Framework (JMF) may potentially allow an untrusted applet to exit unexpectedly ("crash") the Java Virtual Machine (JVM) or gain unauthorized privileges..

- SunSolve WorldWide
- SupportForum
- About SunSolve
- Feedback
- Site Map
- Features/etc.
- SunSolve Home

- Date Released: 14-May-2003
- Date Closed: 14-May-2003
- Date Modified:

1. Impact

A vulnerability in the Java(TM) Media Framework (JMF) may potentially allow an untrusted applet to exit

AP4:不恰当的作用域: 重构

Before (JMF 2.1.1c)		After (JMF 2.1.1e)
<pre>public class NBA { public void finalize() public Object getData() public Object clone() public void copyTo(NBA nba) public void copyTo(byte javadata[]) public long data; public int size; public Class type; }</pre>	<p>1</p> <p>2</p> <p>3</p>	<pre>public final class NBA { protected final synchronized void finalize() public synchronized Object getData() public synchronized Object clone() public synchronized void copyTo(NBA nba) public synchronized void copyTo(byte javadata[]) private long data; private int size; private Class type; }</pre>

- 1) 应该禁止对此类的继承, 以防止新方法去泄露机密数据.
- 2) public的finalize方法的作用域被降为protected, 所以没有类能覆盖它
- 3) 数据字段被移到了适当的private (class级别) 作用域



AP4:不恰当作用域的副作用: 如何审查?

1. 通过探测 **public** 类型 来找潜在的问题类型
2. 对于那些类型, 判定:
 - I. 数据字段定义为 **public**
 - II. 方法定义为 **public**
 - III. **public** 方法返回一个 `private`, `protected` 内部数据的引用

Findbugs 的预定义检测器可实现定位潜在的问题类型。



AP4: 结论和建议

结论

- ❖ 在方法和字段上不恰当的作用域可能导致跳过访问控制机制

建议 [<http://java.sun.com/security/seccodeguide.html>]

- ❖ 避免使用`public`变量,而是通过接口的访问方法(**accessor methods**)来访问内部变量. 通过这种方式,在需要的时候可以很容易增加特定的安全检查(**centralized security checks**).
- ❖ 确认任何`public`方法访问并/或修改任何敏感内部数据的时候包括安全检查.



AP5: 非终态静态变量

◆ 避免使用非终态(*non-final*)公共静态变量(*static variables*)

- ◆ 尽可能不使用非终态公共静态变量，因为没有机制来检测改变此变量的代码是否有适当的权限。
- ◆ 一般情况下，小心对待任何易变的静态状态，它能导致在两个设计上相互独立的子系统间出现不希望的交互。

[*Sun Security Code Guidelines*]

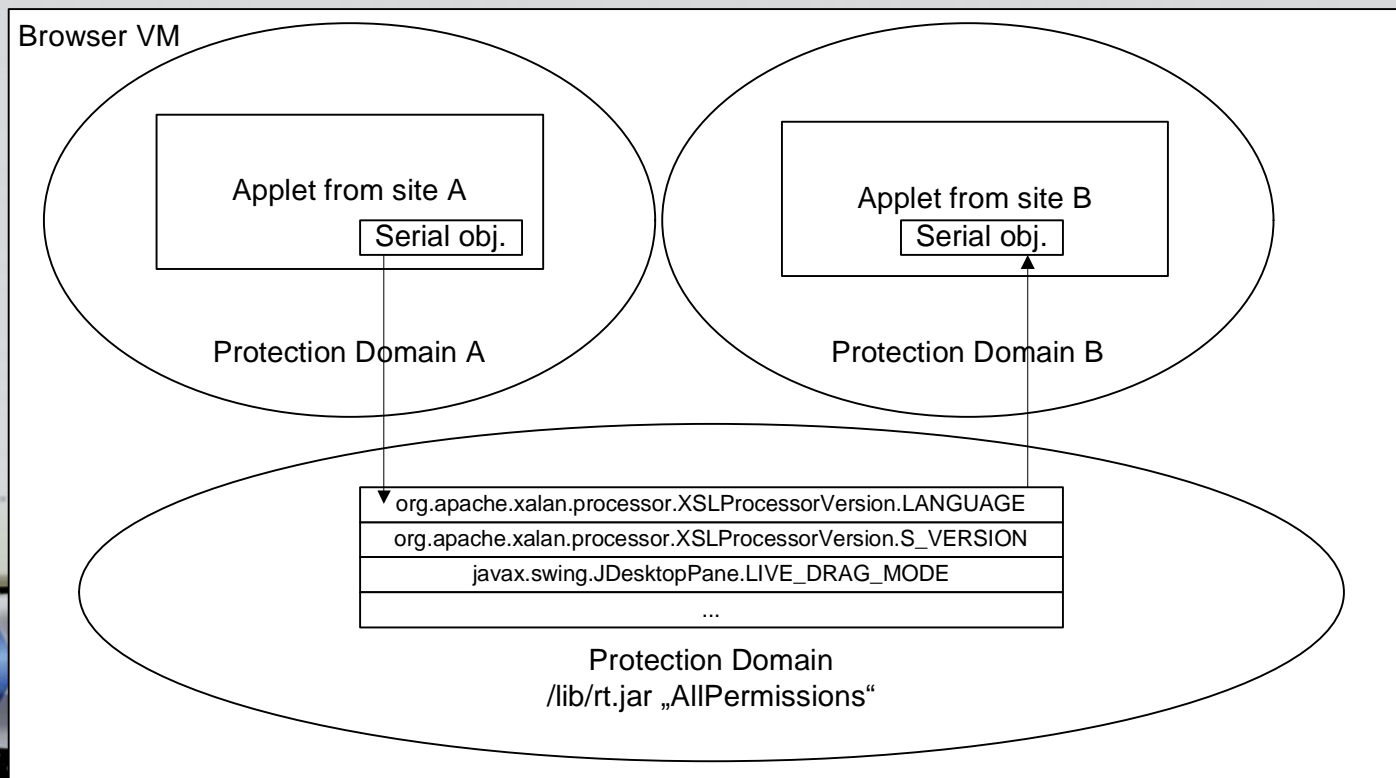
- ◆ 根据Sun Microsystems [<http://www.sun.com/software/security/glossary.html>] 术语 **covert channel** 定义如下：
 - ◆ 设计上并非用来进行数据通讯的通讯通道。它允许进程以违背安全策略意图的方式间接的传递信息。
- ◆ 我们将展示 **粗心的使用静态变量** 这样一个反模式的情景，可能导致恶意代码利用 **保护域间隐蔽通道** 进行通讯。



AP5:非终态静态变量, 风险与范围

风险

- ◆ Static Variables (被boot classloader(like the ones in `rt.jar`) 或者是扩展的classloader装载) 在一个JVM中是一个单件(singleton)
- ◆ **Non-final static String** 字段可能会把序列化好的java对象传给 没有权限访问它们的其他保护域。



AP5:非终态静态变量, 风险与范围

heise online · ct · iX · Technology Review · Telepolis · mobil · Security · ct-TV · Jobs · IT-Markt

heise online

Versteht nicht jeder. Ist auch besser so!

mit dem iX-Schnupper-Abo
3 Ausgaben zum Sonderpreis von nur 10,- €

Suche ...

7-Tage-News
News-Archiv
News mobil

news 23.10.2003 10:27

<< Vorige | Nächste >>

Hilfe

Unsignierte Java-Applets brechen aus

<http://www.heise.de/newsticker/meldung/41308>

Unsigned Java-Applets jump out of Sandbox

Leserforum
Chat-Events
English Pages

Abo & Heft
Kontakt
Mediadaten

Newsletter kostenlos
abonnieren.



sogenannte Sandbox verhindern, die den Zugriff von Applets auf Ressourcen des Systems und anderer Prozesses verbietet.

Auf Grundlage der Java-Klasse `org.apache.xalan.processor.XSLProcessorVersion` zur Verarbeitung von XML-Daten hat Schoenefeld eine Demonstration des Fehlers programmiert. Ein unsigniertes Applet liest dabei Daten aus einer Variablen eines signierten Applets einer anderen Domäne. Verändert das unsignierte Applet den Inhalt der Variablen, kann das signierte Applet sogar abstürzen. Getestet wurde das Verhalten mit [Suns JDK/SDK 1.4.2_01](#), eine Lösung für das Problem gibt es derzeit nicht.

Windows-
schmutzige
PC
21C3: Hack
Besuchern

Aktuelle

Apple klagt
Geheimnis

Organisch
Zoll für TV

AP5:非终态静态变量: 重构

Before (JDK1.42_04)

```
public class org.apache.xalan.processor.  
XSLProcessorVersion {  
    public static final java.lang.String  
    PRODUCT;  
    public static java.lang.String LANGUAGE;  
    public static int VERSION;  
    public static int RELEASE;  
    public static int MAINTENANCE;  
    public static int DEVELOPMENT;  
    public static java.lang.String S_VERSION;  
}
```

After (JDK1.42_05)

```
public class org.apache.xalan.processor.  
XSLProcessorVersion {  
    public static final java.lang.String  
    PRODUCT;  
    public static final java.lang.String  
    LANGUAGE;  
    public static final int VERSION;  
    public static final int RELEASE;  
    public static final int MAINTENANCE;  
    public static final int DEVELOPMENT;  
    public static final java.lang.String  
    S_VERSION;  
}
```

final 修饰符的变量在初始化后就不允许修改。最初，他们仅使用它来保护他们的产品名称 J



AP5:非终态静态变量: 如何审核?

1. **findbugs** 内建的检测器通过查询**public** 类型来找潜在的问题类型
2. 对于那些类，我们需要找出
 - I. 基础数据字段和String定义为**public static, non-final**
 - II. 对象类型数据字段，数组，容器定义为 **public static**
 - III. 允许访问(I + II) 中的non-public 实例的方法



AP5: 结论和建议

结论

- ◆ Non-final static 字段允许在两个保护域间建立隐蔽通道，跳过诸如applet 沙箱(sandbox)这样的约束。

建议 [<http://java.sun.com/security/seccodeguide.html>]

- ◆ 尽可能不要使用non-final public static variables，因为没有机制来检查改变该变量的代码是否有适当的权限。
- ◆ 一般的，小心的对待那些易变的static states，因为它可以在两个本来独立子系统间触发希望的交互。



反模式 6:重用非安全组件

- ❖ „分布式基于组件的应用可以由不同提供商提供的软件组件所组成。因此必须区分应用本身和软件组件提供商，他们需要有不同的保护策略“: [Hermann, Krumm]
- ❖ 第三方的组件可能是根据程序员想要的功能的；然而，JDK的限制执行模型的控制管理需要安全，而不单单是功能。
- ❖ JDK（作为一个component-structured 中间件应用）从Apache foundation那里拿来了许多XML的功能。对于嵌入到JDK中的第3方组件，是否有足够的保护来对抗弱点呢？



AP6:重用非安全组件 :风险与范围

风险

- ◆ 嵌入到JDK中的XSLT parser是直接从原先 apache **XALAN** 独立版本拿过来的, 可从 <http://xml.apache.org> 下载
- ◆ 此版本具有很强的可配置性, 特别是允许自定义函数, 在 **XSLT** (extensible stylesheet language transformations) 中被调用.
- ◆ 在受信的库中的**Non-final static** 数组可能包含对象, 允许把该对象传递到JVM的各个地方。
- ◆ 下面我们将展示 重用非安全组件 这样一个反模式, 允许恶意代码通过插入恶意回调函数获得受信代码权限



<http://classic.sunsolve.sun.com/pub-cgi/retrieve.pl?doc=fsalert%2F57613>

[\[Printer-Friendly Page\]](#)

Document Audience: PUBLIC

Document ID: 57613

Title: Document ID 57613

Synopsis: Java Runtime Environment May Allow Untrusted Applets to Escalate Privileges

Update Date: 2004-08-02

Description

Sun(sm) Alert Notification

- Sun Alert ID: 57613
- Synopsis: Java Runtime Environment May Allow Untrusted Applets to Escalate Privileges
- Category: Security
- Product: Java JRE/SDK
- BugIDs: 5020333, 4954066
- Audience: Upgrade

1. Impact The XSLT processor included with the Java Runtime Environment (JRE) may allow an untrusted applet to read data from another applet that is processed using the XSLT processor and may allow the untrusted applet to escalate privileges.

Impact The XSLT processor included with the Java Runtime Environment (JRE) may allow an untrusted applet to read data from another applet that is processed using the XSLT processor and may allow the untrusted applet to escalate privileges.

Sun acknowledges, with thanks, Marc Schoenefeld for bringing these issues to our attention.

AP6:重用非安全组件：重构

Before (JDK1.42_05)

```
public class
org.apache.xpath.compiler.FunctionTable {
    public static
    org.apache.xpath.compiler.FuncLoader[]
    m_functions;
    [...]
}
```

After (JDK1.42_06)

```
public class
org.apache.xpath.compiler.FunctionTable {
    private static
    org.apache.xpath.compiler.FuncLoader[]
    m_functions;
    [...]
}
```

该重构把组件的增强功能调整到这样的水平：在受限的执行模式中能够安全的运行。技术上来说，该重构修复了反模式 4 和反模式 5 的问题。

private 修饰阻止恶意代码对XSLT 解析器内建函数表的修改。



AP6:重用非安全组件：如何审查？

1. 第三方组件可能包括所有类型的反模式，以我们的经验至少应该检查下列的现存反模式
 1. 检查 **Integer** 溢出
 2. 检查正确的序列化, 注意其副作用
 3. 检查特权代码的使用, 特别是当使用特权或 “AllPermission” 保护域(Protection Domains).
 4. 调整不合适的作用域, 给公共可用的字段和功能加安全检查.
 5. 在 **static non-final** 字段和 **static mutable** 容器类型上关闭隐蔽通道 (也包括间接使用)。



AP6: 结论和建议

结论

- ◆ 即使你自己的代码是安全的,第 3rd 方的组件可以摧毁你的安全堡垒

建议

- ◆ 向你要重用的提供商询问,是否有用过findbugs或者类试的工具检查过他们的组件
- ◆ 在你购买之前要求一个findbugs的报告,这可能增加对组件的信任.
- ◆ 许多开源的项目已经包括了类试这样一个报告,一些闭源软件的开发应该学习下.



finally{}

Q & A

联系方式

给我发 eMail

marc@marc-schoenefeld.com

发布的检测器

下载

www.illegalaccess.org

