by 空虚浪子心 http://www.inbreak.net 微博：http://t.qq.com/javasecurity

**摘要：**

2012年，我在[《攻击JAVA WEB》](http://www.inbreak.net/?p=477" \t "_blank)，文中提多关于“classLoader导致特定环境下的DOS漏洞”，当时并没有更加深入的说明，这几天struts官方修补了这个漏洞，本文是对这个漏洞的深入研究。

**正文：**

这一切，得从我们控制了classLoader说起，曾经写过一篇文章，提到了一个小小的技术细节，非常不起眼的一个鸡肋。  
引用[《Spring framework（cve-2010-1622）漏洞利用指南》](http://www.inbreak.net/?p=377)：  
—  
Struts2其实也本该是个导致远程代码执行漏洞才对，只是因为它的字段映射问题，只映射基础类型，默认不负责映射其他类型，所以当攻击者直接提交URLs[0]=xxx时，直接爆字段类型转换错误，结果才侥幸逃过一劫罢了。  
—

     tomcat8.0出来后，这个问题爆发了，这是一个鸡肋漏洞的逆袭。

      在struts2任何一个action运行之前，一旦接受到用户提交参数xx=zzzzz时，就由Ognl负责调用对应的当前action的setXxx方法，至于set方法到底是干什么的，其实不重要，里面的逻辑也不重要，我们只关注这个方法调用了，参数传递了。这种对属性的改变，有时候是可以很大程度的影响后续复杂逻辑。

**普及一点基础**

      Object是java的基础类，所有的class生成的对象，都会继承Object的所有属性和方法，因此当前action无论是什么代码，必须有Object自带的getClass方法，这个方法会返回一个Class对象，Class对象又一定会有getClassLoader方法，最终在每个action都可以

|  |
| --- |
| getClass().getClassLoader() |

      拿到当前的ClassLoader。

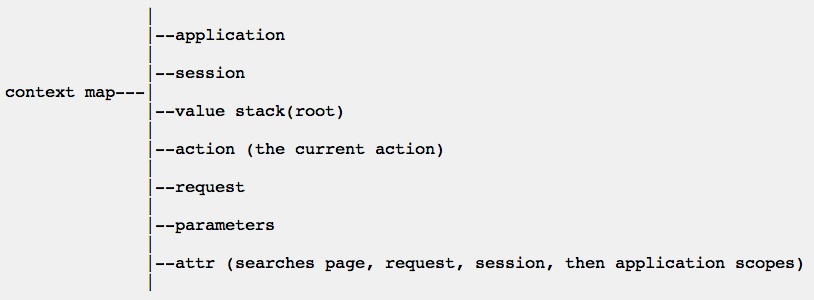
      我研究这个问题，在几年前了，这个东西理解起来不容易，尤其是各个web容器不一致，刚巧当时有个阿里巴巴内部《tomcat等容器的classLoader加载原理》培训，收获匪浅。本文篇幅有限，简单的讲一下。

      在JRE启动中，每个Class都会有自己的ClassLoader。web容器，为了方便的管理启动过程，通常都有实现自定义的ClassLoader。《Spring framework》的漏洞的利用场景真的非常幸运，利用了web容器的特性getURLs方法，所有容器的servlet的ClassLoader都会通过继承父类UrlClassLoader得到getURLs这个方法，所以这个漏洞可以不受容器影响。事实上，每个容器的ClassLoader都是自己实现的，环境必然会有所不同，那次struts2侥幸逃过一劫，所以我的一个关注点，一都放在几大web容器的ClassLoader代码变化上，哪天看到tomcat8居然把resources放进ClassLoader上，而ServletContext刚巧挂在resources上，顿时知道肉戏来了。

**上传webshell的可能性研究**

      多次的远程代码执行漏洞洗礼，我一直在脑海里模拟“ServletContext被控制了，这次能干什么”，究竟有哪些路线，可以通往代码执行的领域。  
比如：  
Struts2会去servletContext里取到一个值，然后把它作为Ognl执行掉。这个太简单了，我自己都不信。

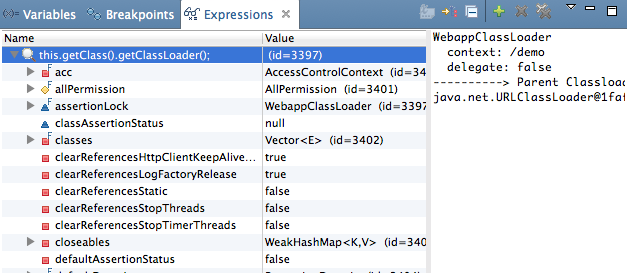
Ognl的Context树形结构：



      servletContext被转换成Map，变成了图中的application子项，这个位址很尴尬，如果是上一层Node，从上到下找到value Stack，确实有实现这个思路的可能，但现在看来，这条路断了，它不支持找到父节点。经过多次找寻后，确认Ognl出局，只能从web容器本身入手。

      运行在Tomcat8下的struts，在随便哪个action代码中，插入这段，下断点，

|  |
| --- |
| **this**.getClass().getClassLoader(); |



      任何一个Action的classLoader都是org.apache.catalina.loader.WebappClassLoader，这是漏洞的源头。

我的思路，是给context赋予初始化参数readOnly=false。因为在tomcat上，所有的请求，都会默认由defaultServlet上处理，或者由jspServet处理。只要在context初始化时，readOnly=false，接下来defaultServlet就会真的处理PUT请求DELETE请求等等大威力请求，于是我们可以直接PUT文件上来。

      这个思路的前提，是defaultservlet再被初始化一次。现实很残酷，这个思路最终没有得到执行，原因是servlet只会初始化一次，而readOnly属性只有在servlet初始化时，才会真的给servet的readOnly属性赋值。这次突破，宣告失败。

**几个这个漏洞的调试小技巧：**

      1、仅仅从debug上查看ognl的赋值情况，是不准确的，debug只能看到这个类定义好的变量。

如果有一个代码是这样的：

|  |
| --- |
| **public** **void** setName(String name){…} |

            但是并没有定义过这个属性，这时debug无法看到这个东西，但是其实ognl可以直接通过name=zzzzz调用，并且能把zzzz传递过去。

      2、或者只有一个私有属性，但是没有set方法，其实也是ognl不能赋值的。

            这个debug，观察这个漏洞时，仅仅是个参考，要真正深入进去看代码才能和ognl的视线保持一致。

      3、一个final的对象，或者只是get方法返回一个对象，看起来像是只读的，其实对象的属性还是可以改的，这个只是对象的引用。

            你可以理解为指针指向的地址不能变，但是指向的那个对象的属性，是可以修改的。

举例：

|  |
| --- |
| **public** User getUser()  {  **return** **this**.user;  }  **public** **final** User user; |

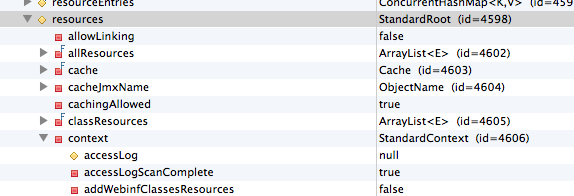
      这两处代码，其实真正返回给OGNL的都是user对象，对象的属性只要还有set方法，也都是可以被修改的。依然可以通过

|  |
| --- |
| url?user.name=xxx |

      对user的name赋值。

**struts2运行在tomcat8.0.1rc5(2013,11月前)的任意文件读取漏洞**

      在tomcat的环境下，classLoader会挂载一个resources，类名叫做“StandardRoot”，这个恐怖的东西，和tomcat的资源文件管理有关，debug看到的第一个属性就是非常危险的属性“allowLinking”。



      这个事情，要从很久很久以前，struts修补的一个任意文件读取漏洞说起。

            http://struts.apache.org/release/2.3.x/docs/s2-004.html

      这是一个目录列表+文件读取漏洞，修补方案非常阴险，没有采用正规的手段，在框架层解决漏洞，而是利用了web容器的一个公约特性，jsp的web容器都遵守一个规则。

当一个路径叫做“/var/www/tomcat/webapps/demo/struts/../”时，调用

|  |
| --- |
| getClassLoader().getResource(Path) |

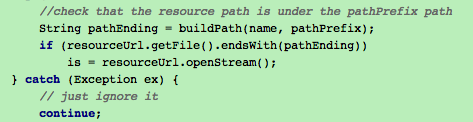
返回路径为：

|  |
| --- |
| “/var/www/tomcat/webapps/demo/” |

会把/../去掉，并且直接到达目的目录。

这个叫做web容器特性，由web容器说了算，哪天web容器生气了，想变了，struts没有话语权。事实上，我一直喜欢讲框架安全，其中一条准则，就是“框架不要依靠web容器的特性做防御”，当然，今天不讨论这个话题，只是稍微做个铺垫。

当时修补代码为：



      用户提交“struts/../”时，pathEnding=“struts/../”。

但是

|  |
| --- |
| resourceUrl=“/var/www/tomcat/webapps/demo/” |

      所以并不以pathEnding结尾。这种猥琐的做法，当时确实有效。

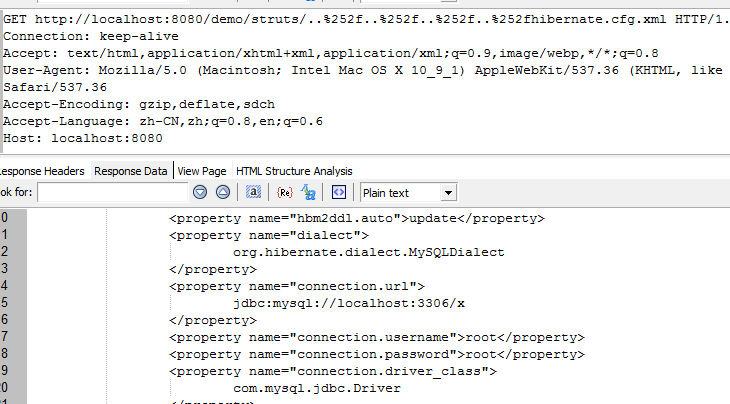
tomcat8这个版本突然抽风了，重写了这个方法，还真的返回了

|  |
| --- |
| “/var/www/tomcat/webapps/demo/struts/../” |

宣告沦陷。但是代码实际运行中，有个要求，就是“StandardRoot.allowLinking”必须是true（默认肯定是false）。  
机会来了。  
首先提交：

|  |
| --- |
| http:*//localhost:8080/demo/t1.action?class.classLoader.resources.allowLinking=true* |

debug可以看到已经是true。  
然后按照以前的攻击方法：



      就可以轻易读取任意文件，拿到数据库密码等等。

这是两个漏洞结合的成果，非常遗憾的是，在RC5这个版本之后，有人给tomcat提交了一个需求，大概在2013年11月左右，tomcat的一个不重要的需求中（刚好这个需求涉及到资源文件相关代码），tomcat维护人员也许并没有意识到了这里存在读取资源文件的威胁，仅仅是把读取资源功能重新抽象规划了一次，结果顺带修补了这个漏洞，这问题产生的理由非常冤屈，修补的理由非常冤屈，最郁闷的是我，活生生的，0day没了。原本沾沾自喜以为可以大杀四方，结果大神打了个喷嚏。

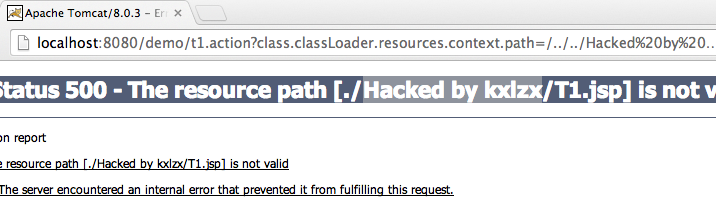
最后顺带说一句，这个漏洞只在windows下正常，linux下未知原因抽风。

**tomcat8下黑掉struts2应用的首页**

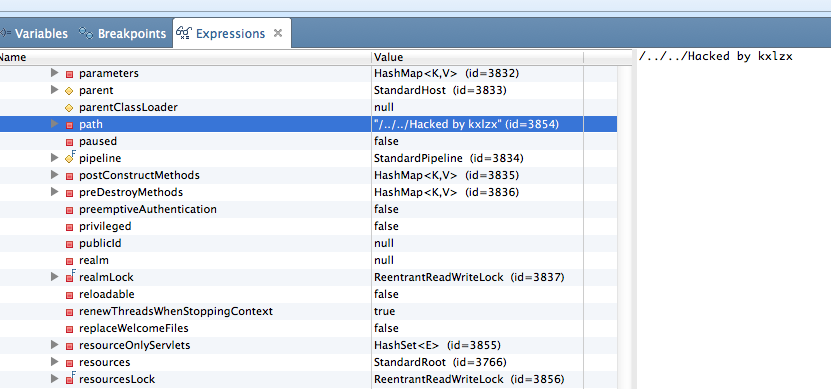
但是不要紧，tomcat是不可能给struts解决根本问题的，standardroot暴露出来，可以顺带影响很多东西。这个算DDOS么？其实我可以把“Hacked by kxlzx”写到你的应用首页去。

成因非常的“正常”，因为这个context属性代表了tomcat的/conf/context.xml文件的配置，我现在把path给你改了，那么struts去处理result时，会用路径拼接读取文件，现在目录名被改掉了，自然就读不到文件了，可惜这里不能00截断，否则又是一个任意文件读取漏洞。

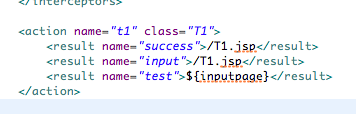
下面是被干掉的网站，访问任何一个action，都会有如下效果：



看看debug的情况：



      当前action叫做T1，会找到T1.jsp，但是现在目录名已经被修改了，所以报错。



      这个问题可以影响tomcat8所有版本下运行的struts2，对了，你们得自己设计EXP哈，不要乱入。

**jetty7关机指令**

      既然提到了web容器，只有研究tomcat，肯定不能覆盖大家关心的地方，于是我选择了另一个开源免费并且使用量大的轻量级web容器：jetty。

现在先看看jetty是否有突破的口子。这次讲解路线反过来，先找个影响“不大”，各位“不是”很关心的漏洞。

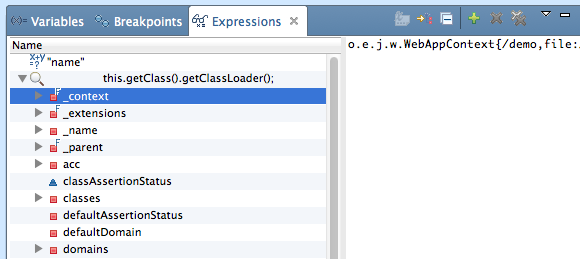
还是先看看web结构,使用老办法断点：

|  |
| --- |
| **this**.getClass().getClassLoader(); |

看到一个class：

|  |
| --- |
| org.eclipse.jetty.webapp.WebAppClassLoader |

jetty的漏洞，没有tomcat那么含蓄，非常直接的，context就挂载在classLoader上。



      jetty在运行过程中，会去实时查看ContextHandler中的shutdown属性（webappcontext通过几层继承关系，继承了这个类，其实亲戚关系有八丈远），一旦发现true，直接就不工作了，这是典型的，使用一个状态判断业务是否同行。基于这个原理，只要如下访问，以后这个应用，就只剩下404了。



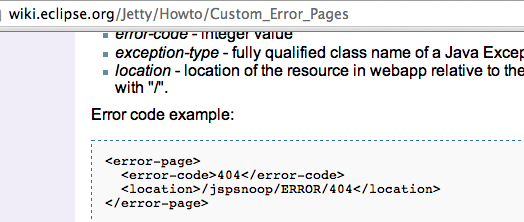
      无论是什么action，都只会返回这个错误，后续的执行，jetty都以为真的shutdown了。并且这个过程没有任何补救措施，只能管理员手工重启了，各位SA亲们，你们准备好了么？

**jetty任意文件下载**

我们让404为这个漏洞服务到底。

事实上，下面说的这个问题发生在jetty上，tomcat真的是巧合的逃过一劫。

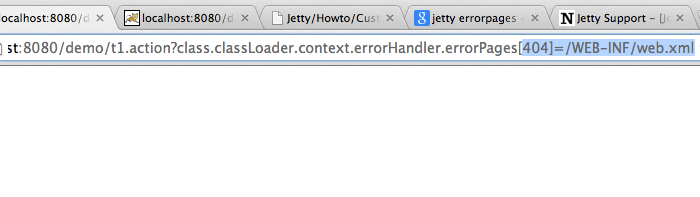
我们看看jetty对于自定义错误文件的配置过程：



      这段配置文件，可以自定义404错误文件，这里可以指定/WEB-INF/目录下的文件，一旦配置之后，由ErrorPageErrorHandler负责给errorPages（这也是个map）添加一个对应关系。这个类最终会被挂在到context中，那么依照这个漏洞的原理，我们可以层层调用，最终制定一种错误，比如404错误。

Jetty把errorHandler挂载到context上，errorHander有个errorPages属性，这其实是个map，代表错误页面，key是一个返回码数字，value就是错误后显示的文件。

所以打开：



      访问图片中这条URL后，效果如下，任何一个不存在的页面

都会显示web.xml的内容：



      有了这个问题，就可以读取数据库文件，查看数据库密码，读取代码文件，查找隐藏的业务逻辑漏洞。注意，是任何人遇到404都可以看到这个页面，最好等夜深人静的时候再使用，用完了还得恢复原样。

**漏洞修补：**

这漏洞已经被官方修补了，2012年发出来的老问题，只是没有单独提交官方而已，居然也能拖到现在，建议各位下定决心换个框架。  
by 空虚浪子心 http://www.inbreak.net 微博：http://t.qq.com/javasecurity