CVE-2022-22980_Spring_Data_MongoDB_SpEL 表达式注入漏洞

漏洞描述

6 月 20 号，VMware 发布安全公告，修复了 spring Data MongoDB 组件中的一个 SpEL 表达式注入漏洞，该漏洞的 CVSSv3 评分为 8.2，漏洞编号：CVE-2022-22980，漏洞威胁等级：高危。

Spring Data MongoDB 应用程序在对包含查询参数占位符的 SpEL 表达式使用 @Query 或@Aggregation 注解的查询方法进行值绑定时，若输入未被过滤，则易遭受 SpEL 注入攻击。该漏洞允许未经身份验证的攻击者构造恶意数据执行远程代码，最终获取服务器权限。

相关介绍

Spring Data for MongoDB 是 Spring Data 项目的一部分，该项目旨在为新的数据存储提供熟悉和一致的基于 Spring 的编程模型，同时保留存储的特定特征和功能。Spring 表达式语言(简称 SpEL)：是一个支持运行时查询和操作对象图的强大的表达式语言，也是一种简洁的装配 Bean 的方式，它通过运行期执行的表达式将值装配到 Bean 的属性或构造器参数中。

通过 SpEL 可以实现：通过 bean 的 id 对 bean 进行引用；调用方式以及引用对象中的属性；计算表达式的值；正则表达式的匹配。

利用范围

Spring Data MongoDB == 3.4.0

3.3.0 <= Spring Data MongoDB <= 3.3.4

更早或不再受支持的 Spring Data MongoDB 版本也受到此漏洞影响

漏洞分析

环境搭建

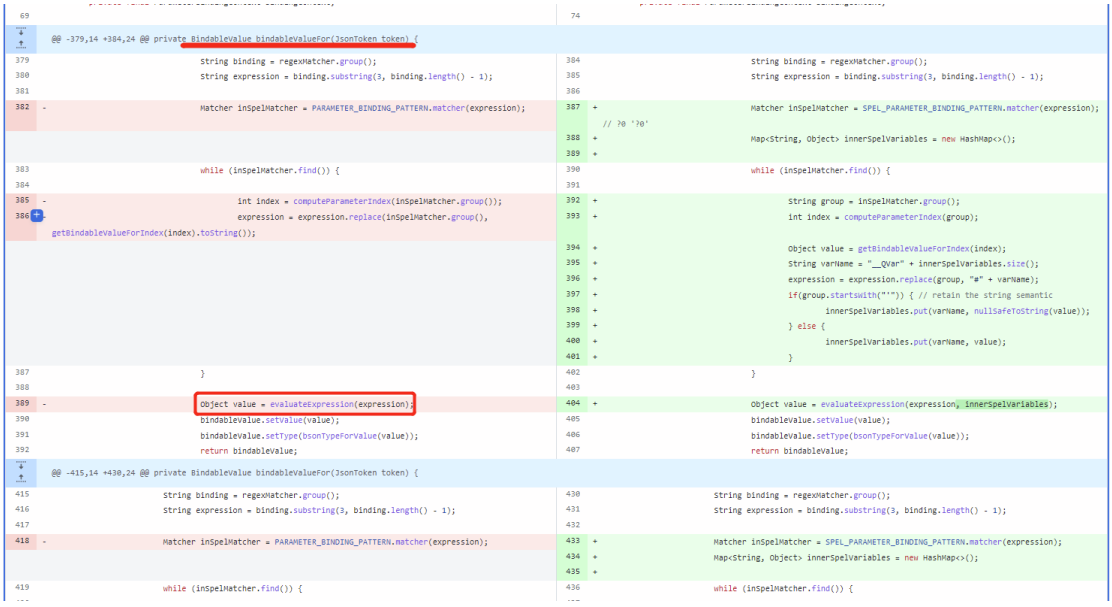
此次采用 threedr3am 师傅的漏洞 demo 进行复现分析。

动态调式

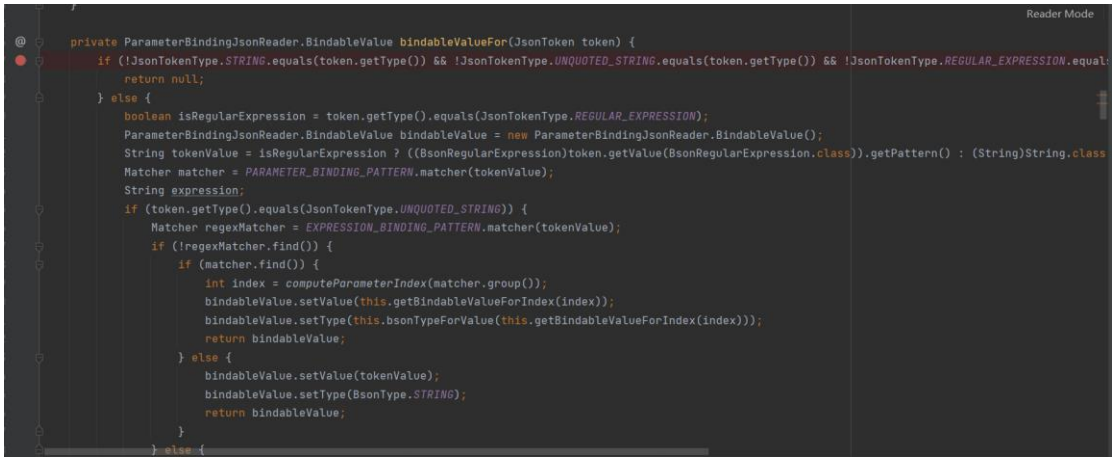
在调试之前查看一下 `demo` 中的 `DemoController`，其构造的请求路径为/demo，请求参数为 keyword。

```java
package com.threedr3am.bug.spring.data.mongodb.controller;

import ...

@RestController
public class DemoController {

  private final DemoRepository demoRepository;

  public DemoController(DemoRepository demoRepository) { this.demoRepository = demoRepository; }

  @RequestMapping(value = "/demo")
  public List<?> demo(@RequestParam(name = "keyword") String keyword) {
    return demoRepository.findAllByIdLike(keyword);
  }
}
```

根据 diff 记录发现，此次漏洞修复的主要位置在 `ParameterBindingJsonReader` 类的 `bindableValueFor` 函数

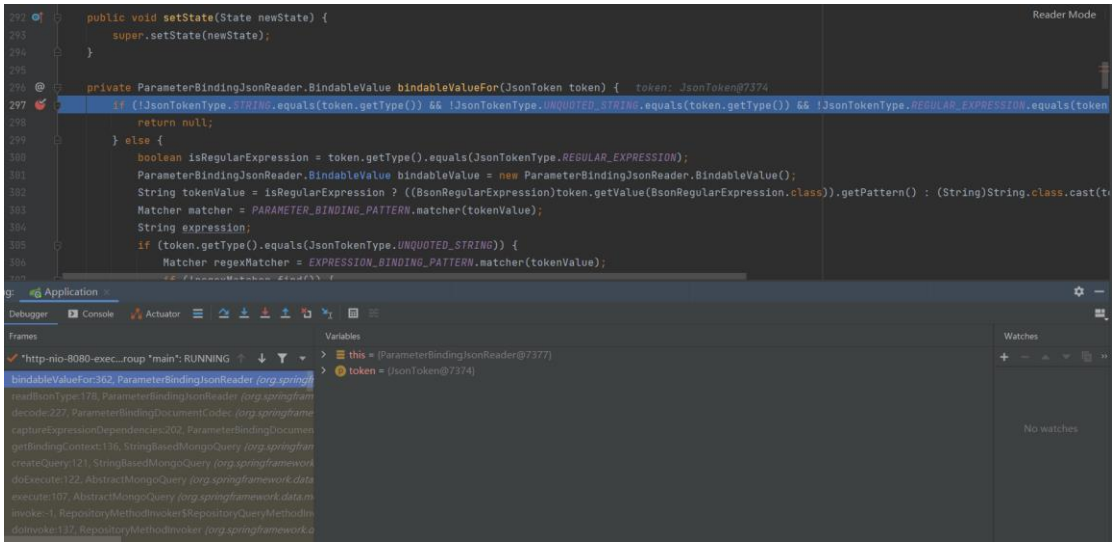


话不多说，先在 `org.springframework.data.mongodb.util.json.ParameterBindingJsonReader#bindableValueFor` 函数处打下断点



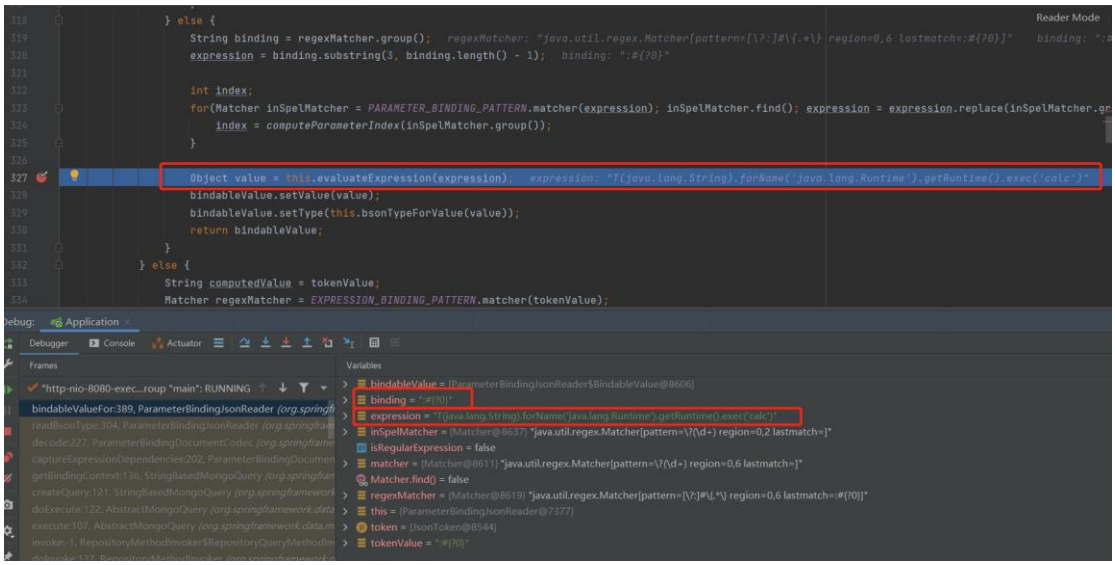
```
Reader Mode
private ParameterBindingJsonReader.BindableValue bindableValueFor(JsonToken token) {
    if (!JsonTokenType.STRING.equals(token.getType()) && !JsonTokenType.UNQUOTED_STRING.equals(token.getType()) && !JsonTokenType.REGULAR_EXPRESSION.equal
        return null;
    } else {
        boolean isRegularExpression = token.getType().equals(JsonTokenType.REGULAR_EXPRESSION);
        ParameterBindingJsonReader.BindableValue bindableValue = new ParameterBindingJsonReader.BindableValue();
        String tokenValue = isRegularExpression ? ((BsonRegularExpression)token.getValue(BsonRegularExpression.class)).getPattern() : (String)String.class
        Matcher matcher = PARAMETER_BINDING_PATTERN.matcher(tokenValue);
        String expression;
        if (token.getType().equals(JsonTokenType.UNQUOTED_STRING)) {
            Matcher regexMatcher = EXPRESSION_BINDING_PATTERN.matcher(tokenValue);
            if (!regexMatcher.find()) {
                if (matcher.find()) {
                    int index = computeParameterIndex(matcher.group());
                    bindableValue.setValue(this.getBindableValueForIndex(index));
                    bindableValue.setType(this.bsonTypeForValue(this.getBindableValueForIndex(index)));
                    return bindableValue;
                } else {
                    bindableValue.setValue(tokenValue);
                    bindableValue.setType(BsonType.STRING);
                    return bindableValue;
                }
            } else {
```

将环境运行起来后开启 debug 模式。使用 burp 抓包并传入 payload 后，立即触发断点。



```
Reader Mode
public void setState(State newState) {
    super.setState(newState);
}

private ParameterBindingJsonReader.BindableValue bindableValueFor(JsonToken token) {  token: JsonToken@7374
    if (!JsonTokenType.STRING.equals(token.getType()) && !JsonTokenType.UNQUOTED_STRING.equals(token.getType()) && !JsonTokenType.REGULAR_EXPRESSION.equals(token
        return null;
    } else {
        boolean isRegularExpression = token.getType().equals(JsonTokenType.REGULAR_EXPRESSION);
        ParameterBindingJsonReader.BindableValue bindableValue = new ParameterBindingJsonReader.BindableValue();
        String tokenValue = isRegularExpression ? ((BsonRegularExpression)token.getValue(BsonRegularExpression.class)).getPattern() : (String)String.class.cast(t
        Matcher matcher = PARAMETER_BINDING_PATTERN.matcher(tokenValue);
        String expression;
        if (token.getType().equals(JsonTokenType.UNQUOTED_STRING)) {
            Matcher regexMatcher = EXPRESSION_BINDING_PATTERN.matcher(tokenValue);
Application
Debugger  Console  Actuator
Frames
"http-nio-8080-exec...roup "main": RUNNING
bindableValueFor:362, ParameterBindingJsonReader (org.springf
readBsonType:178, ParameterBindingJsonReader (org.springfram
decode:227, ParameterBindingDocumentCodec (org.springframe
captureExpressionDependencies:202, ParameterBindingDocumen
getBindingContext:136, StringBasedMongoQuery (org.springfran
createQuery:121, StringBasedMongoQuery (org.springframewor
doExecute:122, AbstractMongoQuery (org.springframework.data
execute:107, AbstractMongoQuery (org.springframework.data.m
invoke:-1, RepositoryMethodInvoker$RepositoryQueryMethodIn
doInvoke:137, RepositoryMethodInvoker (org.springframework.c
Variables
this = {ParameterBindingJsonReader@7377}
token = {JsonToken@7374}
Watches
No watches
```

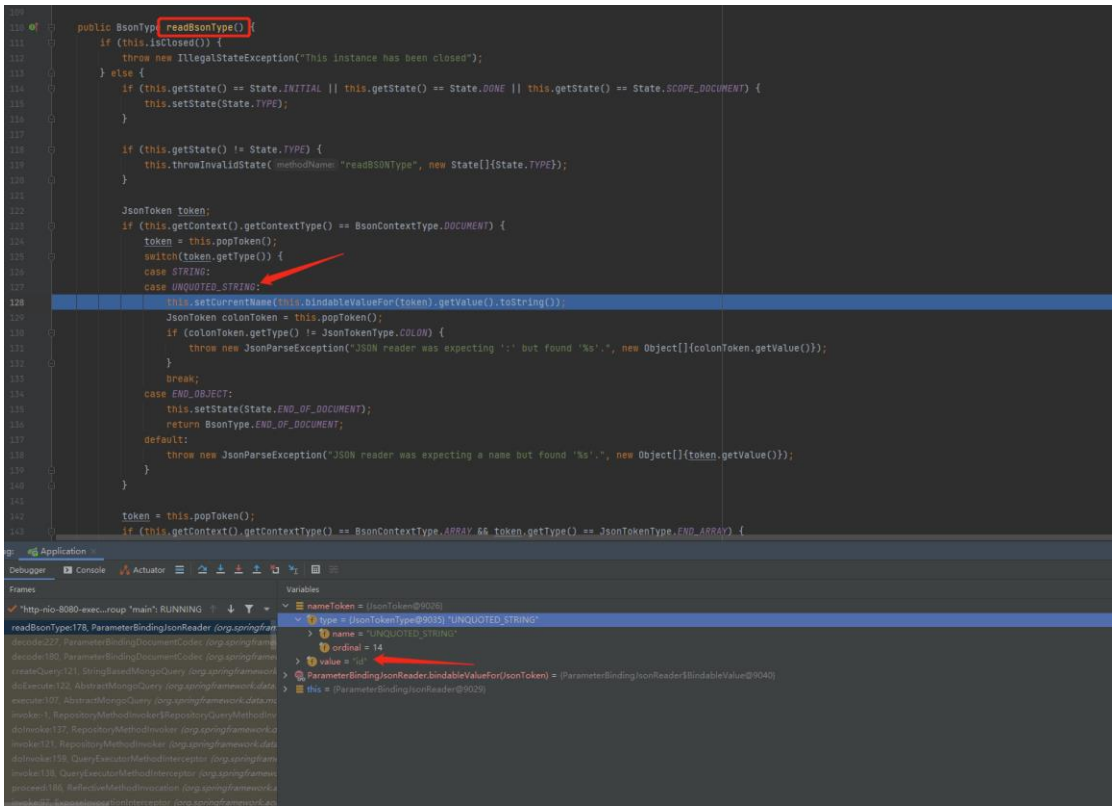
持续跟进，当第一次到达漏洞触发点时，发现并未成功触发 `payload`



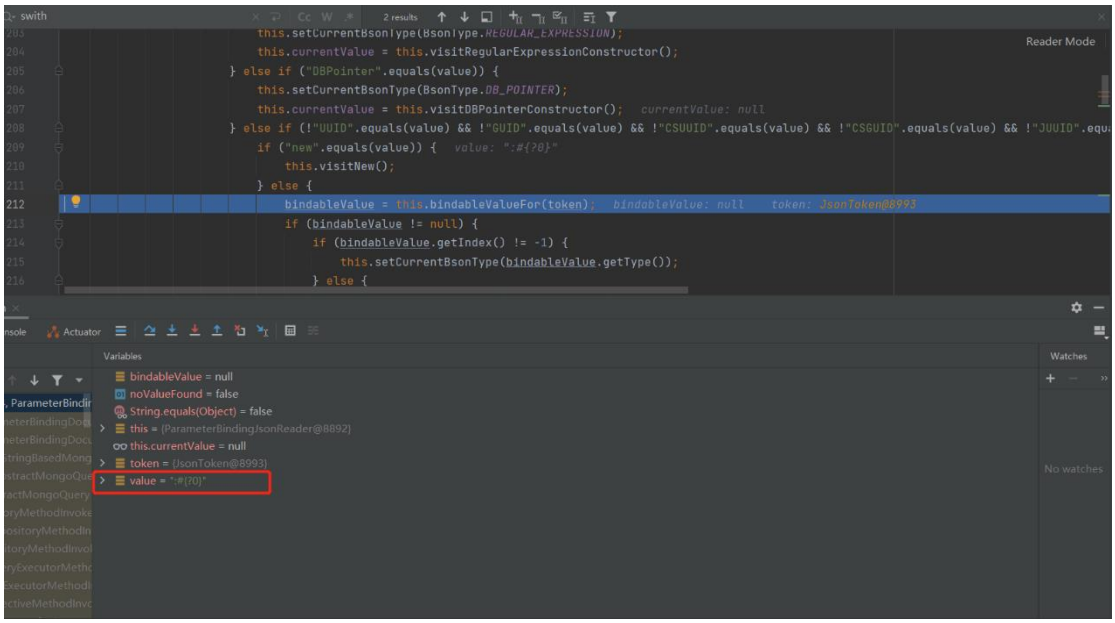
```
Reader Mode
        } else {
            String binding = regexMatcher.group();  regexMatcher: "java.util.regex.Matcher[pattern=\?:]#\{.*\} region=0,6 lastmatch=:#{?0}]"  binding: ":
            expression = binding.substring(3, binding.length() - 1);  binding: ":#{?0}"

            int index;
            for(Matcher inSpelMatcher = PARAMETER_BINDING_PATTERN.matcher(expression); inSpelMatcher.find(); expression = expression.replace(inSpelMatcher.gr
                index = computeParameterIndex(inSpelMatcher.group());
            }

            Object value = this.evaluateExpression(expression);  expression: "T(java.lang.String).forName('java.lang.Runtime').getRuntime().exec('calc')"
            bindableValue.setValue(value);
            bindableValue.setType(this.bsonTypeForValue(value));
            return bindableValue;
        }
    } else {
        String computedValue = tokenValue;
        Matcher regexMatcher = EXPRESSION_BINDING_PATTERN.matcher(tokenValue);
Debug:  Application
Debugger  Console  Actuator
Frames
"http-nio-8080-exec...roup "main": RUNNING
bindableValueFor:389, ParameterBindingJsonReader (org.springf
readBsonType:304, ParameterBindingJsonReader (org.springfram
decode:227, ParameterBindingDocumentCodec (org.springframe
captureExpressionDependencies:202, ParameterBindingDocumen
getBindingContext:136, StringBasedMongoQuery (org.springfran
createQuery:121, StringBasedMongoQuery (org.springframewor
doExecute:122, AbstractMongoQuery (org.springframework.data
execute:107, AbstractMongoQuery (org.springframework.data.m
invoke:-1, RepositoryMethodInvoker$RepositoryQueryMethodIn
Variables
bindableValue = {ParameterBindingJsonReader$BindableValue@8606}
binding = ":#{?0}"
expression = "T(java.lang.String).forName('java.lang.Runtime').getRuntime().exec('calc')"
inSpelMatcher = {Matcher@8637} "java.util.regex.Matcher[pattern=\?(\d+) region=0,2 lastmatch=]"
isRegularExpression = false
matcher = {Matcher@8611} "java.util.regex.Matcher[pattern=\?(\d+) region=0,6 lastmatch=]"
Matcher.find() = false
regexMatcher = {Matcher@8619} "java.util.regex.Matcher[pattern=[\?:]#\{.*\} region=0,6 lastmatch=:#{?0}]"
this = {ParameterBindingJsonReader@7377}
token = {JsonToken@8544}
tokenValue = ":#{?0}"
```

继续跟进，发现在

`org.springframework.data.mongodb.util.json.ParameterBindingJsonReader#readBsonType` 函数中判断 token 的 Type 属性后，进入到 `UNQUOTED_STRING`，在这里进行 `setCurrentName` 操作，`value` 为 `id`



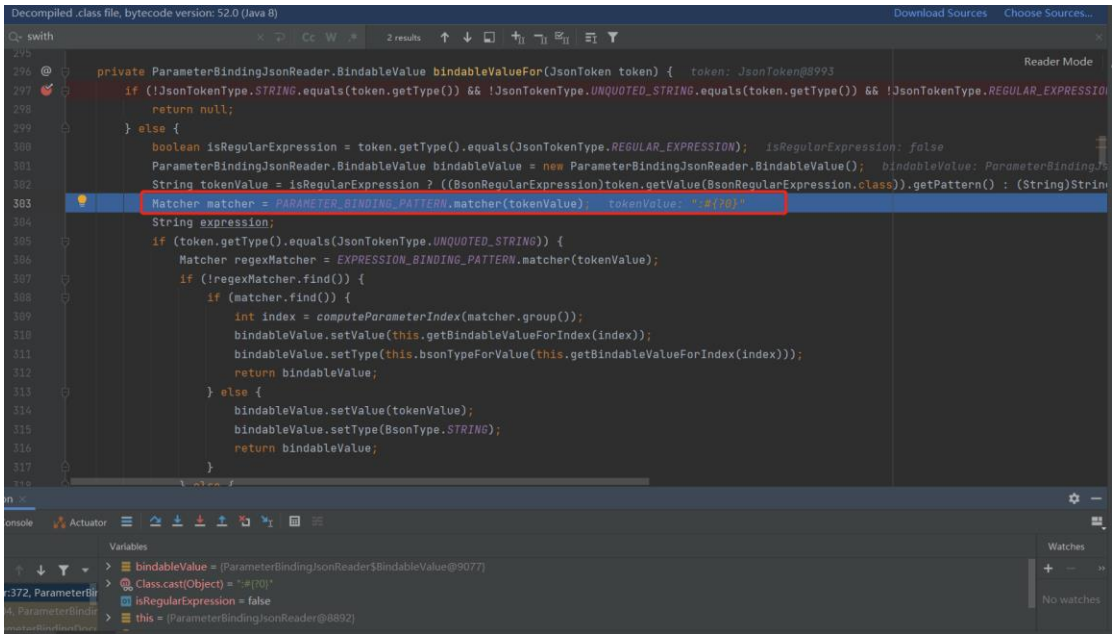
随后回到 `bindableValueFor` 函数，后续经过对 `value` 的处理，`value` 由 `id` 变为了`:#{?0}`



在 `value` 为`:#{?0}`后，会再次进入

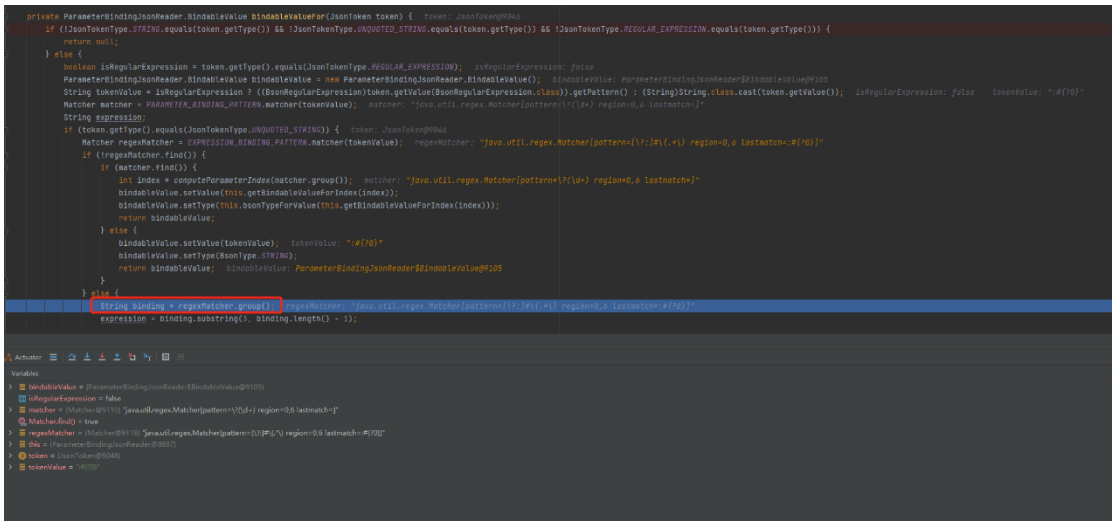
`org.springframework.data.mongodb.util.json.ParameterBindingJsonReader#bindableValueFor` 函数

在 `bindableValueFor` 函数中首先对 `tokenValue` 进行了赋值，随后对 `tokenValue` 进行 `PARAMETER_BINDING_PATTERN` 和 `EXPRESSION_BINDING_PATTERN` 规则匹配

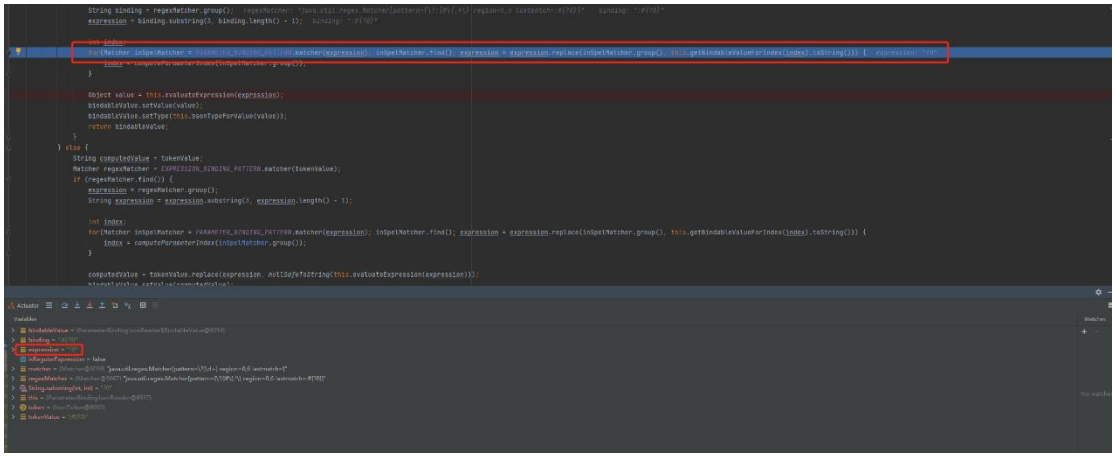


`EXPRESSION_BINDING_PATTERN` 只能匹配 `?#{}` 或者：`#{}`形式的字符串

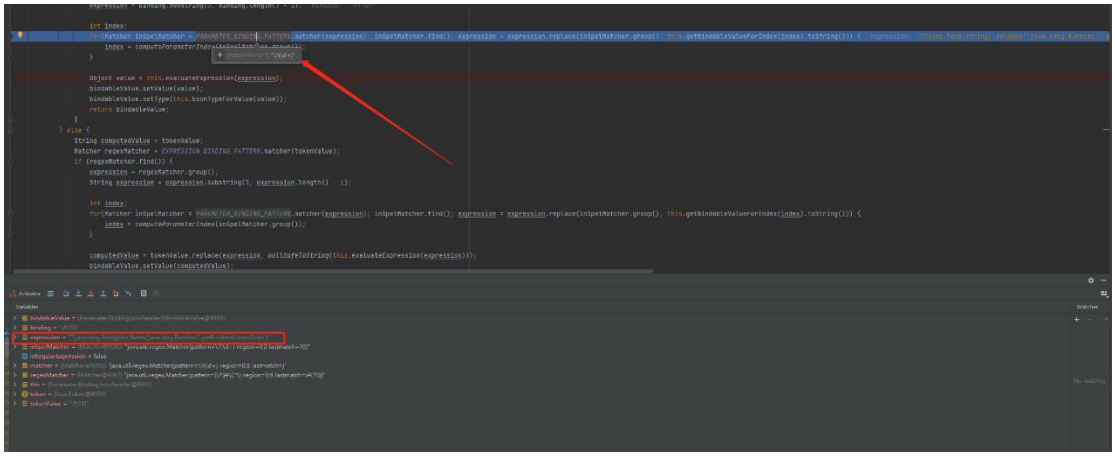
随后，将赋值交给 `binding`，在通过 `substring` 取出占位符`?0`



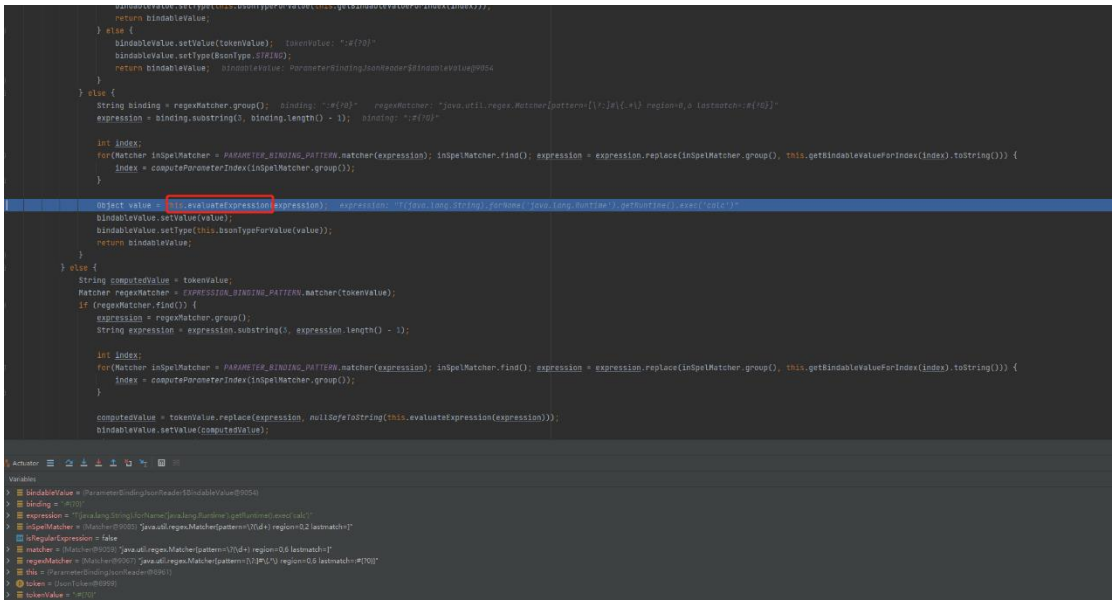
接下来通过 for 循环将占位符和传入的 `payload` 进行替换。



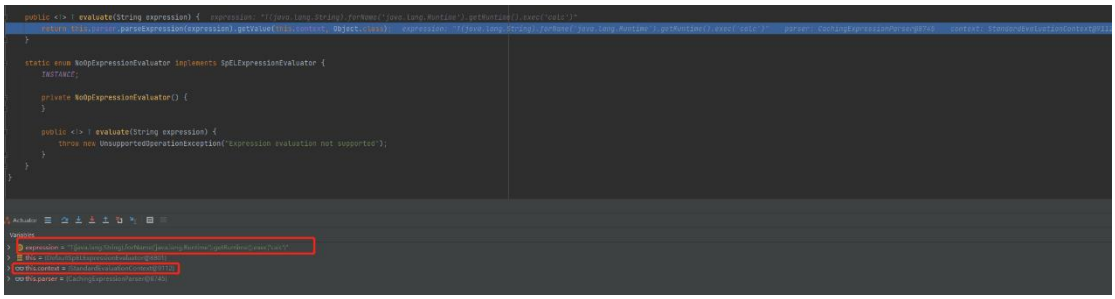
同时通过 `PARAMETER_BINDING_PATTERN` 规则匹配成功后即认为是 `spel` 表达式格式，此时 `expression` 为传入 `payload`



执行 `this.evaluateExpression`

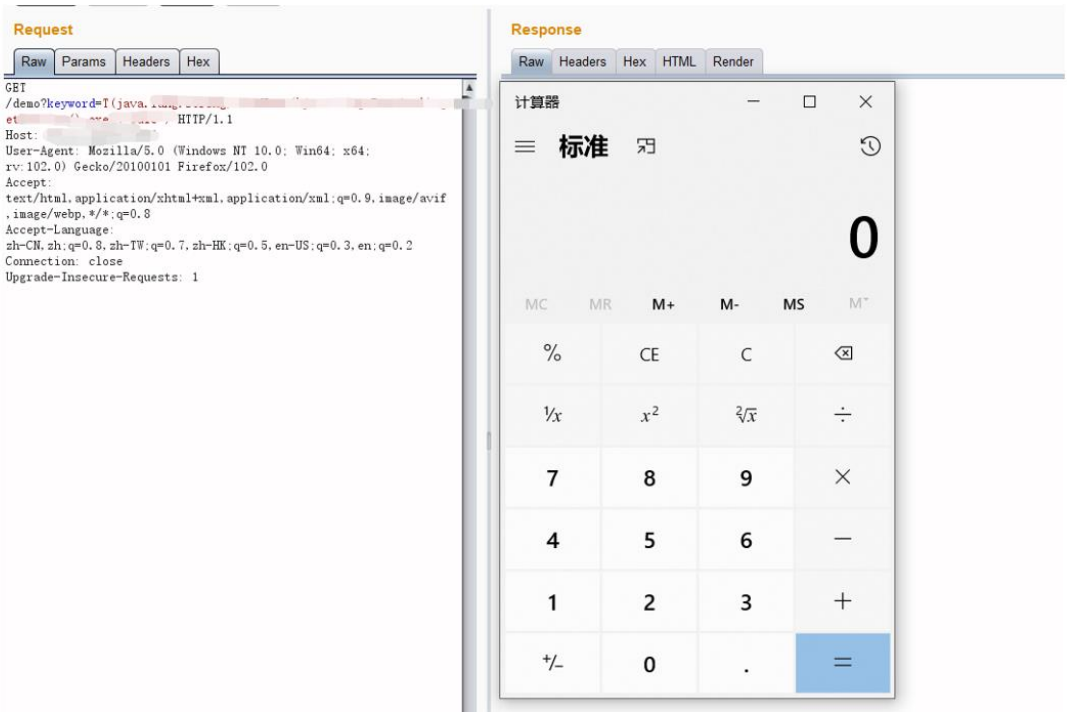


最终进入 `org.springframework.data.mongodb.repository.query.DefaultSpELExpressionEvaluator#evaluate` 函数，此时使用的是 `StandardEvaluationContext` 类型，包含了 `SpEL` 所有的功能



此时的 `SpEL` 表达式为之前构造的恶意攻击载荷，可成功命令执行。

漏洞复现



修复建议

目前此漏洞已经修复，受影响的用户建议尽快升级至官方修护版本:

Spring Data MongoDB 3.4.1 或更高版本;

Spring Data MongoDB 3.3.5 或更高版本。

下载链接:

https://github.com/spring-projects/spring-data-mongodb/tags