CVE-2022-22963_SpringCloud_Function_SpEL 注入漏洞

前言

在研究分析了 CVE-2022-22980 Spring Data MongoDB SpEL 表达式注入漏洞之后，想起之前在 spring4shell 爆出之前，存在于 SpringCloud Function 中的一个 SpEL 表达式注入漏洞，编号为 CVE-2022-22963。在这里对其进行一波分析和学习。

漏洞描述

Spring Cloud Function 是基于 Spring Boot 的函数计算框架。该项目致力于促进函数为主的开发单元，它抽象出所有传输细节和基础架构，并提供一个通用的模型，用于在各种平台上部署基于函数的软件。在 Spring Cloud Function 相关版本，存在 SpEL 表达式注入。恶意攻击者无需认证可通过构造特定的 HTTP 请求头注入 SpEL 表达式，最终执行任意命令，获取服务器权限。

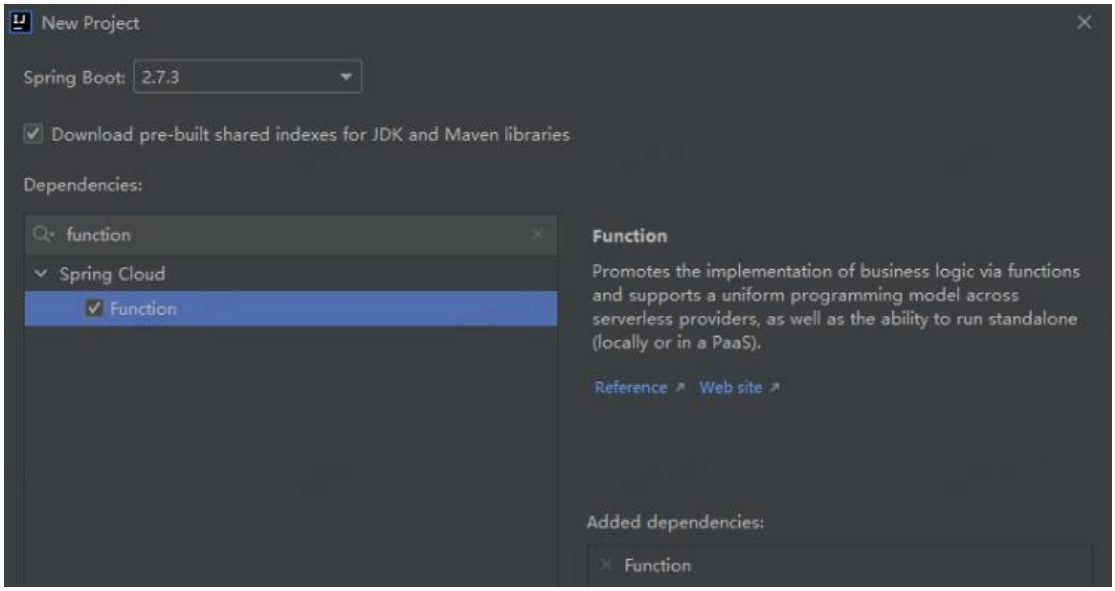
利用范围

3.0.0 <= Spring Cloud Function <= 3.2.2

漏洞分析

环境搭建

使用 idea 新建 Spring Cloud Function 项目



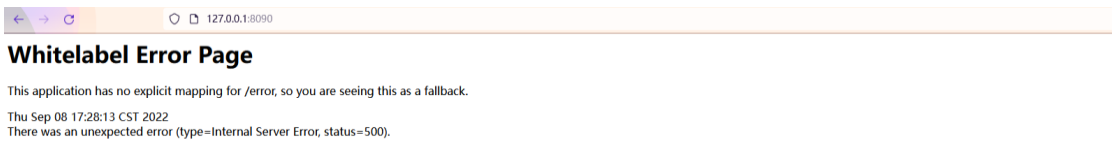
pom.xml 中引入 `spring-boot-starter-web`、`spring-cloud-function-web`

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-function-web</artifactId>
    <version>3.2.2</version>
</dependency>
```

在 application.properties 中添加 `spring.cloud.function.definition=functionRouter`

```
application.properties
spring.cloud.function.definition=functionRouter
server.port=8090
server.address=127.0.0.1
```

这里设置端口为 8090，默认为 8080



运行之后，环境搭建完成。

前置知识

SpringCloud Function 相关介绍

简单的介绍，Spring Cloud 是一系列框架的集合， 内部包含了许多框架，这些框架互相协作，共同来构建分布式系统。利用这些组件，可以非常方便的构建一个分布式系统。SpringCloudFunction 就是一个 SpringBoot 开发的 Servless 中间件（FAAS）

Spring Cloud Function功能：

- *编程风格的选择-反应式，命令式或混合式。*
- *功能组成和适应（例如，将命令性功能与反应性组合）。*
- *支持具有多个输入和输出的反应式功能，从而允许功能处理合并，联接和其他复杂的流操作。*
- *输入和输出的透明类型转换。*
- *特定于目标平台的部署打包功能（例如，Project Riff，AWS Lambda等）*
- *适配器将功能作为HTTP端点等向外界公开*
- *使用隔离的类加载器部署包含此类应用程序上下文的JAR文件，以便可以将它们打包在一起在单个JVM中。*
- *将作为Java函数体的字符串编译为字节码，然后将其转换为 `@Beans` ，可以按上述方式包装它们。*
- *适用于AWS Lambda， Microsoft Azure， Apache OpenWhisk以及其他“无服务器”服务提供商的适配器。*

这是一个完整的，可执行的，可测试的Spring Boot应用程序（实现简单的字符串操作）：

```
@SpringBootApplication
public class Application {
  public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
  }

  @Bean
  public Function<Flux<String>, Flux<String>> uppercase() {
    return flux -> flux.map(value -> value.toUpperCase());
  }
}
```

在环境搭建时，我们在 application.properties 中添加 `spring.cloud.function.definition=functionRouter`

这里的属性 `spring.cloud.function.definition` 表示声明式函数组合，简单理解就是一个默认路由。具体可参考如下说明。

Declarative Function Composition

This feature allows you to provide composition instruction in a declarative way using `|` (pipe) or `,` (comma) delimiter when providing `spring.cloud.function.definition` property.

For example

```
--spring.cloud.function.definition=uppercase|reverse
```

Here we effectively provided a definition of a single function which itself is a composition of function `uppercase` and function `reverse`. In fact that is one of the reasons why the property name is *definition* and not *name*, since the definition of a function can be a composition of several named functions. And as mentioned you can use `,` instead of pipe (such as `…definition=uppercase,reverse`).

functionRouter

Function Routing and Filtering

Since version 2.2 Spring Cloud Function provides routing feature allowing you to invoke a single function which acts as a router to an actual function you wish to invoke This feature is very useful in certain FAAS environments where maintaining configurations for several functions could be cumbersome or exposing more then one function is not possible.

The `RoutingFunction` is registered in *FunctionCatalog* under the name `functionRouter`. For simplicity and consistency you can also refer to `RoutingFunction.FUNCTION_NAME` constant.

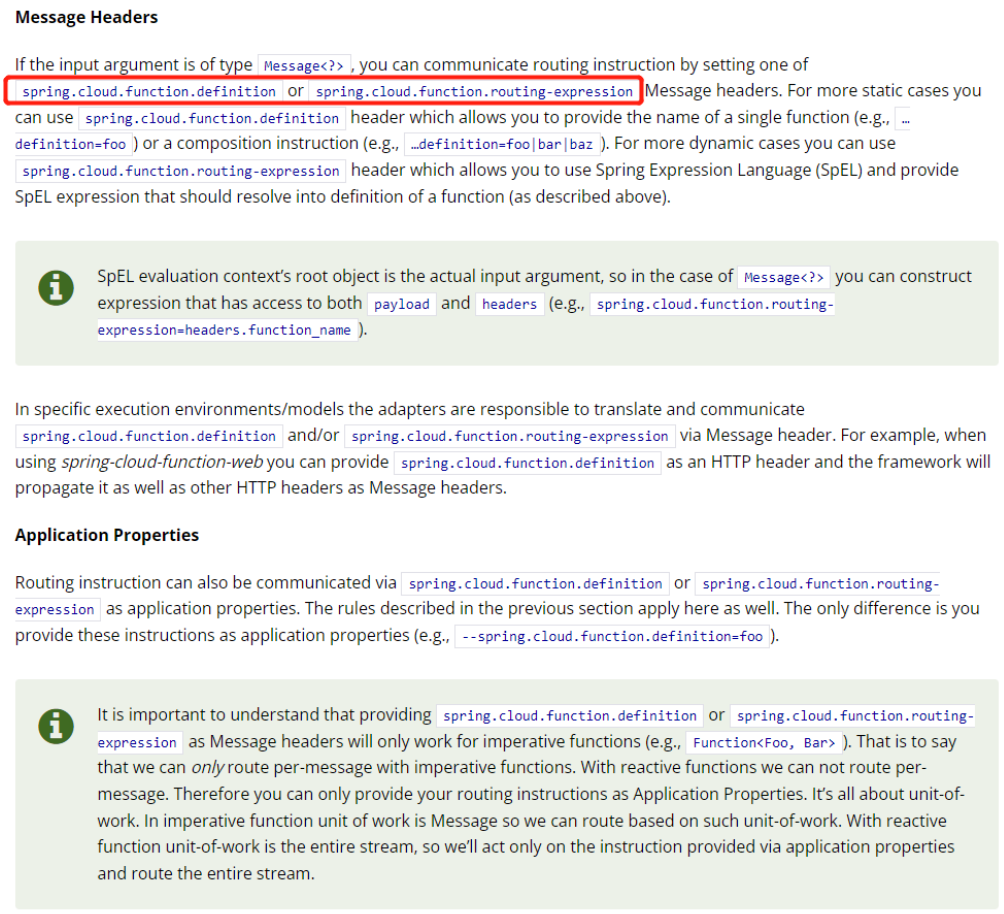
This function has the following signature:

JAVA

```
public class RoutingFunction implements Function<Object, Object> {
. . .
}
```

The routing instructions could be communicated in several ways. We support providing instructions via Message headers, System properties as well as pluggable strategy. So let's look at some of the details

我们设置 `spring.cloud.function.definition=functionRouter` 就是使默认路由绑定具体函数交由用户进行控制。



Message Headers

If the input argument is of type `Message<?>`, you can communicate routing instruction by setting one of `spring.cloud.function.definition` or `spring.cloud.function.routing-expression` Message headers. For more static cases you can use `spring.cloud.function.definition` header which allows you to provide the name of a single function (e.g., `...definition=foo`) or a composition instruction (e.g., `...definition=foo|bar|baz`). For more dynamic cases you can use `spring.cloud.function.routing-expression` header which allows you to use Spring Expression Language (SpEL) and provide SpEL expression that should resolve into definition of a function (as described above).

SpEL evaluation context's root object is the actual input argument, so in the case of `Message<?>` you can construct expression that has access to both `payload` and `headers` (e.g., `spring.cloud.function.routing-expression=headers.function_name`).

In specific execution environments/models the adapters are responsible to translate and communicate `spring.cloud.function.definition` and/or `spring.cloud.function.routing-expression` via Message header. For example, when using *spring-cloud-function-web* you can provide `spring.cloud.function.definition` as an HTTP header and the framework will propagate it as well as other HTTP headers as Message headers.

Application Properties

Routing instruction can also be communicated via `spring.cloud.function.definition` or `spring.cloud.function.routing-expression` as application properties. The rules described in the previous section apply here as well. The only difference is you provide these instructions as application properties (e.g., `--spring.cloud.function.definition=foo`).

It is important to understand that providing `spring.cloud.function.definition` or `spring.cloud.function.routing-expression` as Message headers will only work for imperative functions (e.g., `Function<Foo, Bar>`). That is to say that we can *only* route per-message with imperative functions. With reactive functions we can not route per-message. Therefore you can only provide your routing instructions as Application Properties. It's all about unit-of-work. In imperative function unit of work is Message so we can route based on such unit-of-work. With reactive function unit-of-work is the entire stream, so we'll act only on the instruction provided via application properties and route the entire stream.

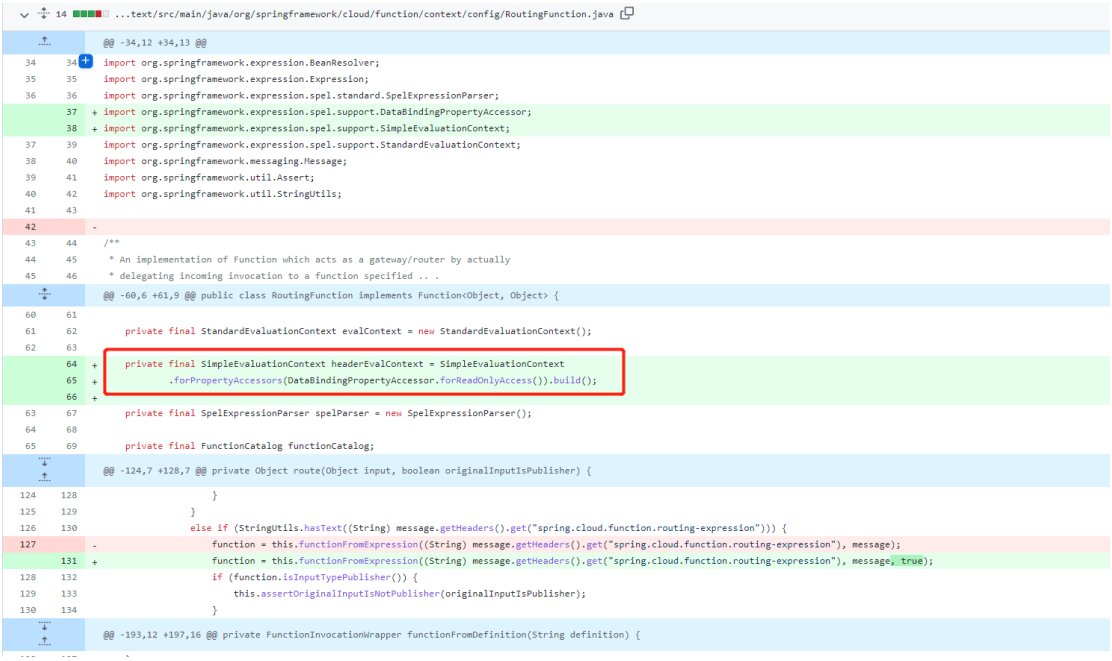
在 *spring-cloud-function-web* 中可以通过设置 Message Headers 来传达路由指令，也可以路通过 `spring.cloud.function.definition` 或 `spring.cloud.function.routing-expression` 作为应用程序属性进行通信，允许使用 Spring 表达式语言 (SpEL)

这就是产生 SpEL 注入的关键所在。

动态分析

在理解了前置知识中相关原理，其实也就能大概知晓漏洞原理。

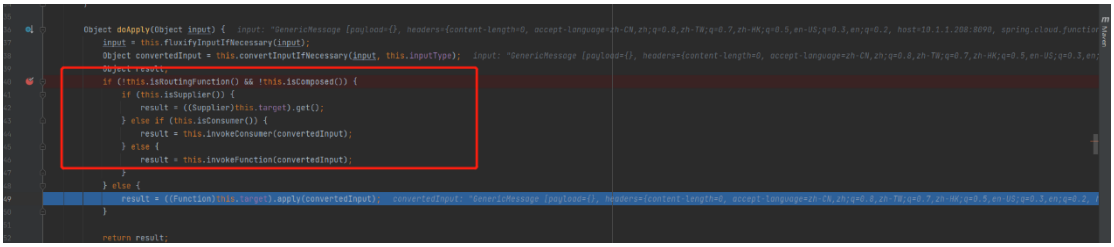
查看 DIFF 记录



```
...text/src/main/java/org/springframework/cloud/function/context/config/RoutingFunction.java

@@ -34,12 +34,13 @@
import org.springframework.expression.BeanResolver;
import org.springframework.expression.Expression;
import org.springframework.expression.spel.standard.SpelExpressionParser;
+import org.springframework.expression.spel.support.DataBindingPropertyAccessor;
+import org.springframework.expression.spel.support.SimpleEvaluationContext;
import org.springframework.expression.spel.support.StandardEvaluationContext;
import org.springframework.messaging.Message;
import org.springframework.util.Assert;
import org.springframework.util.StringUtils;

-
/**
 * An implementation of Function which acts as a gateway/router by actually
 * delegating incoming invocation to a function specified .. .
@@ -60,6 +61,9 @@ public class RoutingFunction implements Function<Object, Object> {

    private final StandardEvaluationContext evalContext = new StandardEvaluationContext();

+    private final SimpleEvaluationContext headerEvalContext = SimpleEvaluationContext
+            .forPropertyAccessors(DataBindingPropertyAccessor.forReadOnlyAccess()).build();
+
    private final SpelExpressionParser spelParser = new SpelExpressionParser();

    private final FunctionCatalog functionCatalog;
@@ -124,7 +128,7 @@ private Object route(Object input, boolean originalInputIsPublisher) {
                    }
                }
                else if (StringUtils.hasText((String) message.getHeaders().get("spring.cloud.function.routing-expression"))) {
-                    function = this.functionFromExpression((String) message.getHeaders().get("spring.cloud.function.routing-expression"), message);
+                    function = this.functionFromExpression((String) message.getHeaders().get("spring.cloud.function.routing-expression"), message, true);
                    if (function.isInputTypePublisher()) {
                        this.assertOriginalInputIsNotPublisher(originalInputIsPublisher);
                    }
@@ -193,12 +197,16 @@ private FunctionInvocationWrapper functionFromDefinition(String definition) {
```

看到从请求头中获取的 spring.cloud.function.routing-expression 之前是由 StandardEvaluationContext 解析，修复新增了 isViaHeader 变量做了一个判断，如果是从请求头中获取的 spring.cloud.function.routing-expression 值，使用 SimpleEvaluationContext 解析。

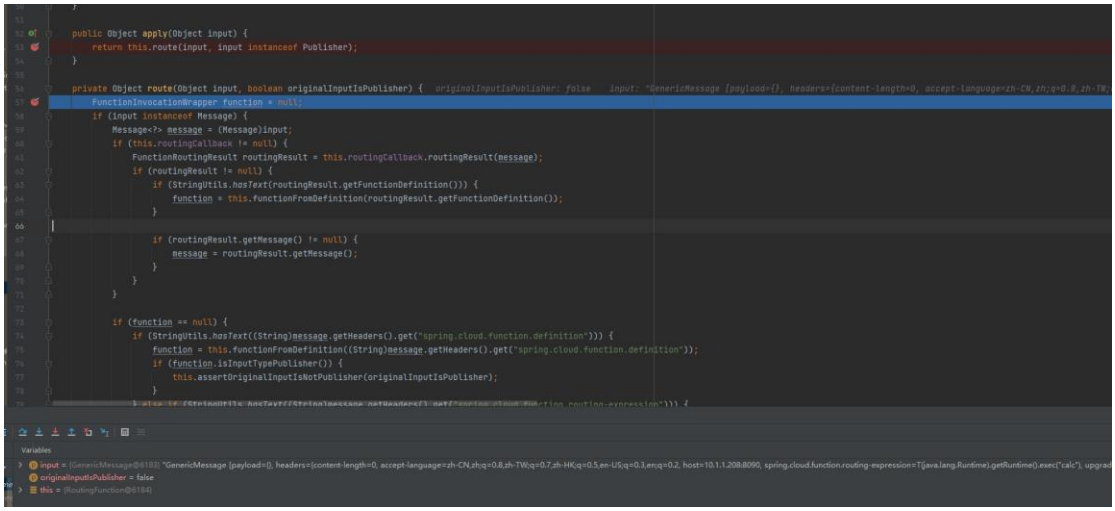
在 spring.cloud.function.context.catalog.simpleFunctionRegistry#doApply 中



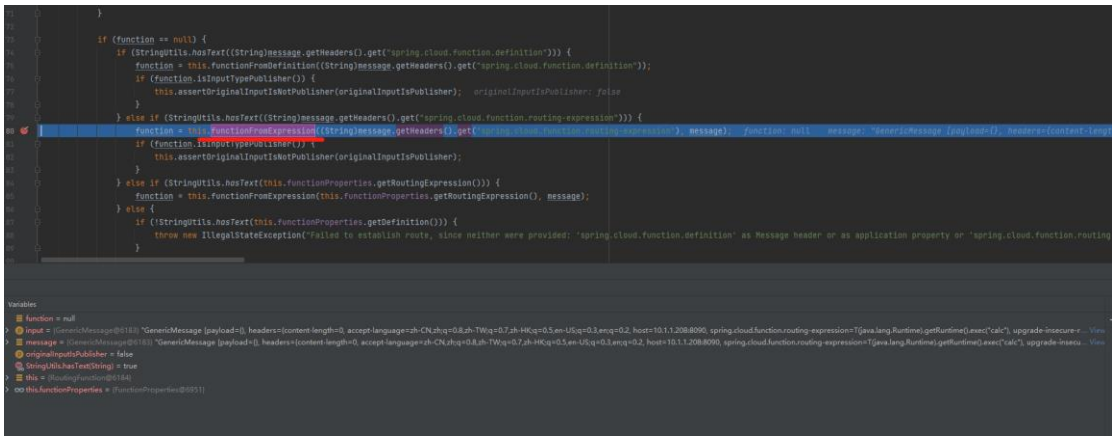
```
Object doApply(Object input) {
    input = this.fluxifyInputIfNecessary(input);
    Object convertedInput = this.convertInputIfNecessary(input, this.inputType);
    Object result;
    if (!this.isRoutingFunction() && !this.isComposed()) {
        if (this.isSupplier()) {
            result = ((Supplier)this.target).get();
        } else if (this.isConsumer()) {
            result = this.invokeConsumer(convertedInput);
        } else {
            result = this.invokeFunction(convertedInput);
        }
    } else {
        result = ((Function)this.target).apply(convertedInput);
    }

    return result;
```

在执行 function apply 方法之后，会跳转到 doApply 中，对 funtion 进行判断，判断是不是 functionRouter 方法

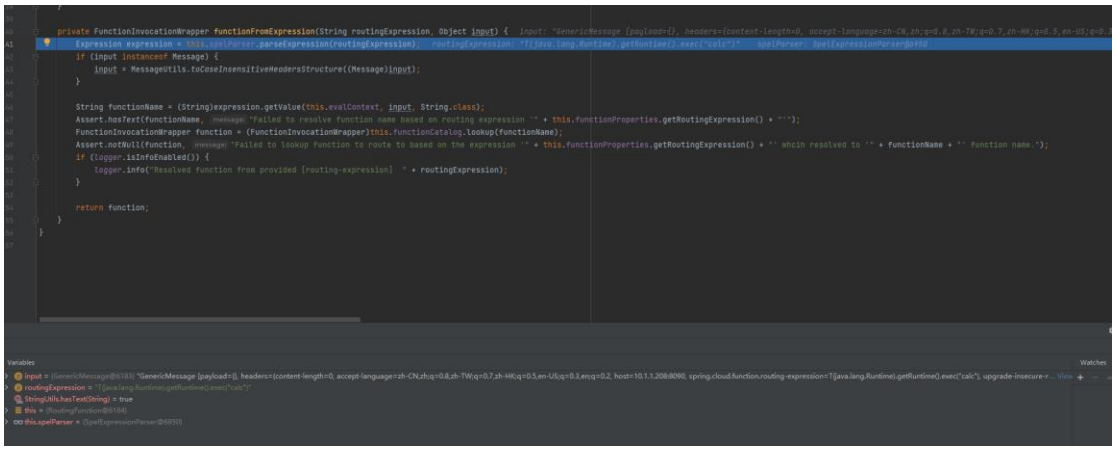
后续跟进，进入 spring.cloud.function.context.config.Routingfunction#route



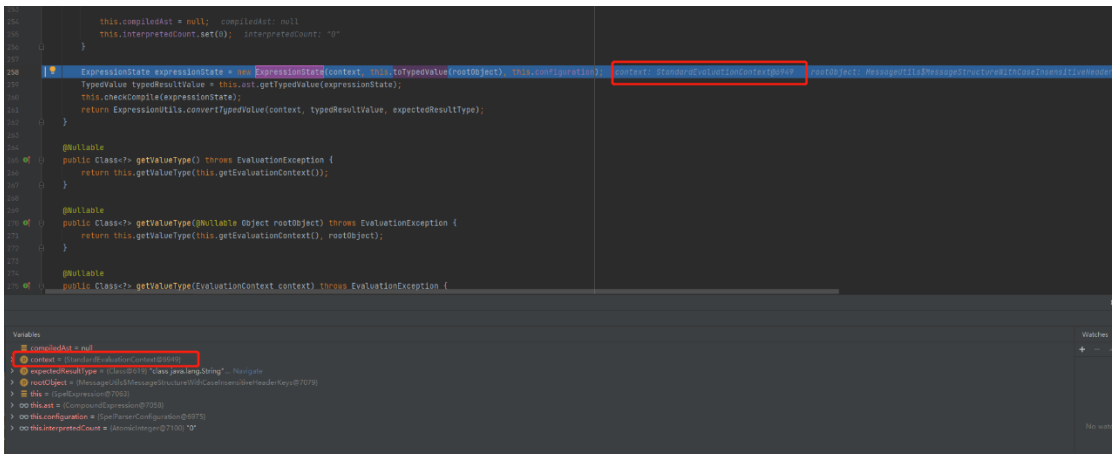
进入 else if 分支， http 头 `spring.cloud.function.routing-expression` 不为空，则传入其值到 `functionFromExpression` 方法



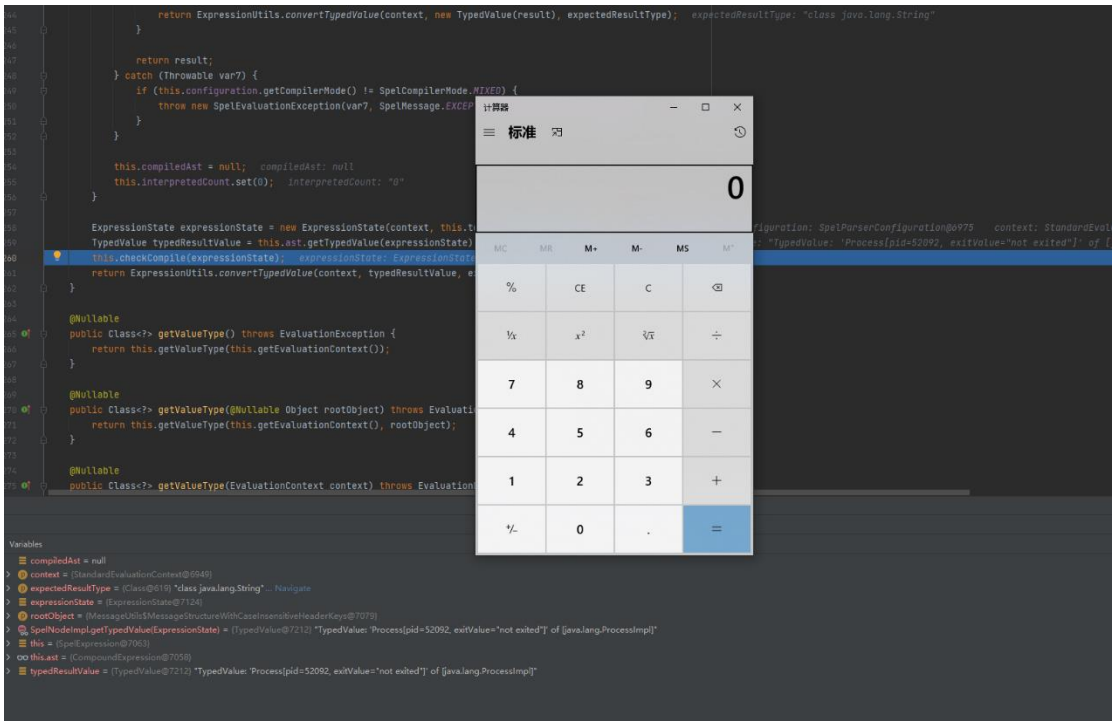
随后对传入的 header 进行解析处理



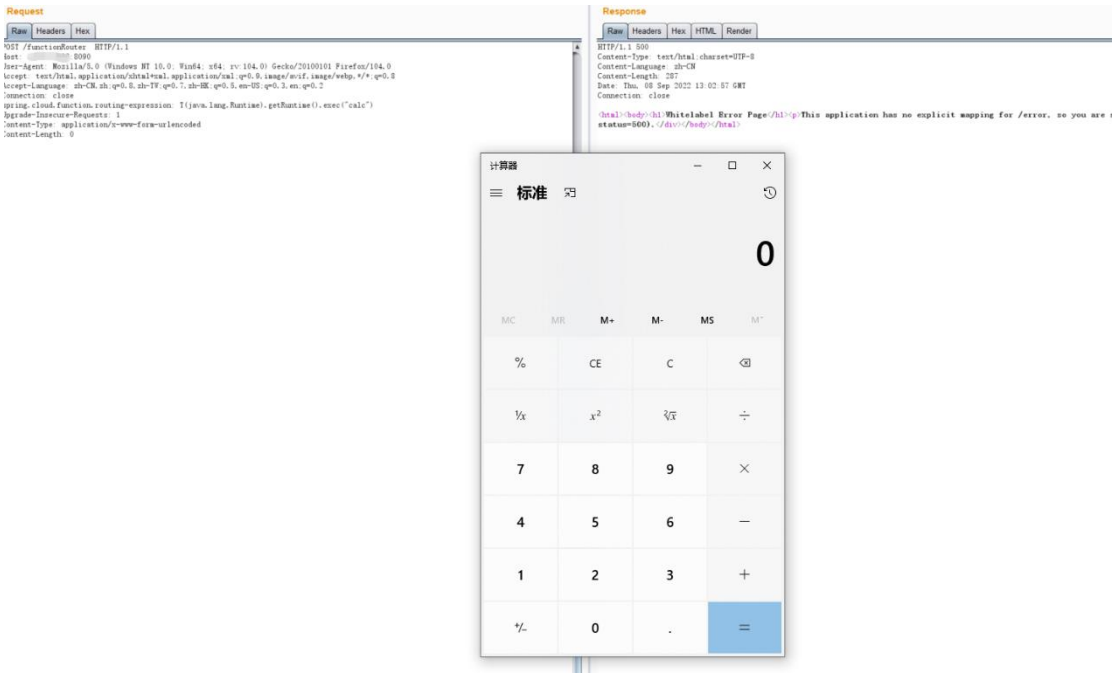
后续跟进发现对 Spel 表达式进行解析的方法就是 StandardEvaluationContext



后续跟进，在解析传入的 Spel 之后，成功触发恶意代码。



漏洞复现



修复建议

受影响的组件更新至 3.1.7、3.2.3 安全版本