

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。

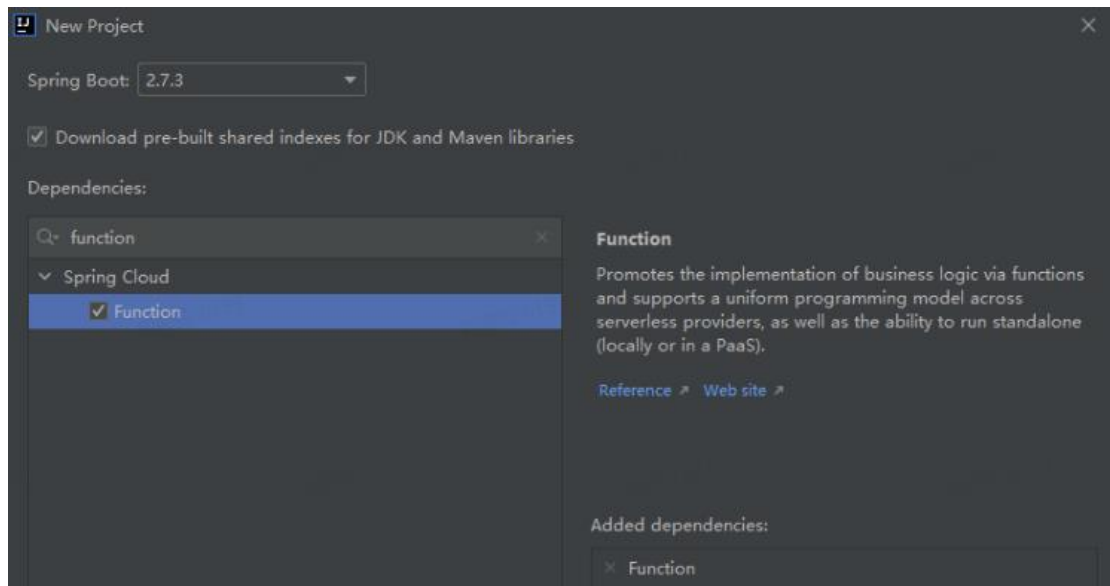
利用范围

3.0.0 <= Spring Cloud Function <= 3.2.2

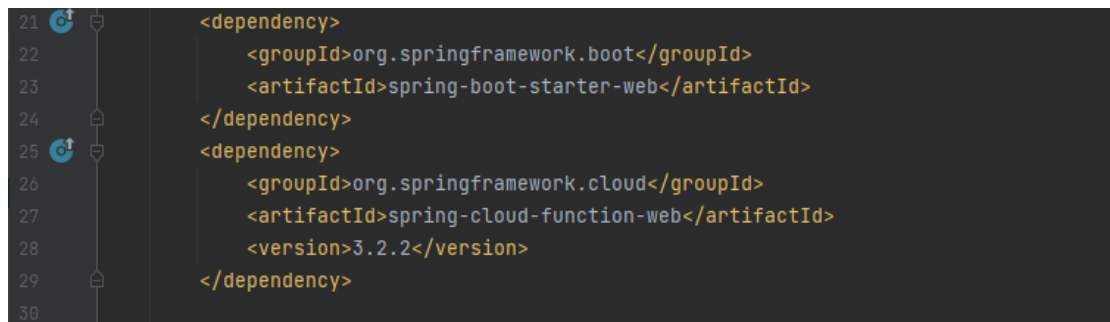
漏洞分析

环境搭建

使用 idea 新建 Spring Cloud Function 项目

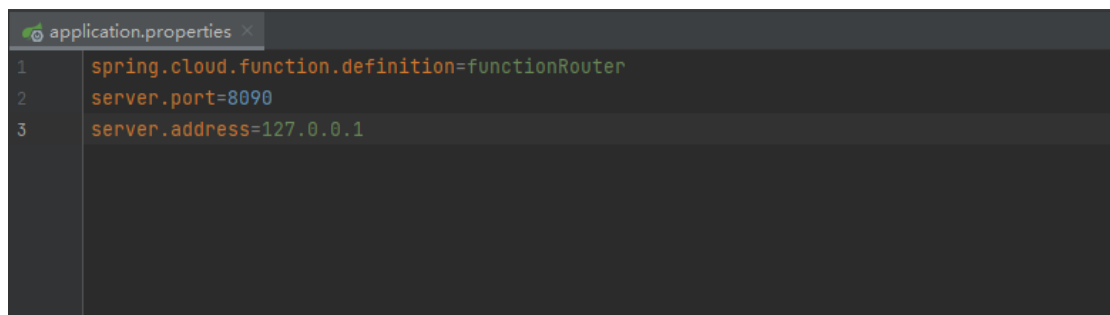


pom.xml 中引入 `spring-boot-starter-web`、`spring-cloud-function-web`



在 `application.properties` 中添加

`spring.cloud.function.definition=functionRouter`



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 than 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:

```
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

```

14 ...text/src/main/java/org/springframework/cloud/function/context/config/RouterFunction.java
34 import org.springframework.expression.BeanResolver;
35 import org.springframework.expression.Expression;
36 import org.springframework.expression.spel.standard.SpelExpressionParser;
37 + import org.springframework.expression.spel.support.DataBindingPropertyAccessor;
38 + import org.springframework.expression.spel.support.SimpleEvaluationContext;
39 import org.springframework.expression.spel.support.StandardEvaluationContext;
40 import org.springframework.messaging.Message;
41 import org.springframework.util.Assert;
42 import org.springframework.util.StringUtils;
43
44 /**
45  * An implementation of Function which acts as a gateway/router by actually
46  * delegating incoming invocation to a function specified ...
47
48 @-60,6 +61,9 @@ public class RouterFunction implements Function<Object, Object> {
49
50     private final StandardEvaluationContext evalContext = new StandardEvaluationContext();
51
52     private final SimpleEvaluationContext headerEvalContext = SimpleEvaluationContext
53     + .forPropertyAccessors(DataBindingPropertyAccessor.forReadOnlyAccess()).build();
54 +
55     private final SpelExpressionParser spelParser = new SpelExpressionParser();
56
57     private final FunctionCatalog functionCatalog;
58
59     @-124,7 +128,7 @@ private Object route(Object input, boolean originalInputIsPublisher) {
60
61     }
62
63     else if (StringUtils.hasText((String) message.getHeaders().get("spring.cloud.function.routing-expression"))) {
64
65         function = this.functionFromExpression((String) message.getHeaders().get("spring.cloud.function.routing-expression"), message);
66         function = this.functionFromExpression((String) message.getHeaders().get("spring.cloud.function.routing-expression"), message, true);
67         if (function.isInputTypePublisher()) {
68             this.assertOriginalInputIsNotPublisher(originalInputIsPublisher);
69         }
70     }
71
72     @-193,12 +197,16 @@ private FunctionInvocationWrapper functionFromDefinition(String definition) {
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

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.classifyInputIfNecessary(input);
    Object convertedInput = this.convertInputIfNecessary(input, this.inputType);
    input = "GenericMessage {payload=(), headers={content-length=0, accept-language=zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2}, host=10.1.1.200:8080, spring.cloud.function.routing-expression=...}";
    Object result;
    if (!this.isRouterFunction() && !this.isComposed()) {
        if (this.isSupplier()) {
            result = ((Supplier<T>)this.target).get();
        } else if (this.isConsumer()) {
            result = this.invokeConsumer(convertedInput);
        } else {
            result = this.invokeFunction(convertedInput);
        }
    } else {
        result = ((Function<T,T>)this.target).apply(convertedInput);
    }
    return result;
}

```

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

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

```
public Object apply(Object input) {
    return this.route(input, input instanceof Publisher);
}

private Object route(Object input, boolean originalInputIsPublisher) {
    FunctionInvocationWrapper function = null;
    if (input instanceof Message) {
        Message<?> message = (Message)input;
        if (this.routingCallback != null) {
            FunctionRoutingResult routingResult = this.routingCallback.routingResult(message);
            if (routingResult != null) {
                if (StringUtils.hasText(routingResult.getFunctionDefinition())) {
                    function = this.functionFromDefinition(routingResult.getFunctionDefinition());
                }

                if (routingResult.getMessage() != null) {
                    message = routingResult.getMessage();
                }
            }

            if (function == null) {
                if (StringUtils.hasText((String)message.getHeaders().get("spring.cloud.function.definition"))) {
                    function = this.functionFromDefinition((String)message.getHeaders().get("spring.cloud.function.definition"));
                    if (function.isInputTypePublisher()) {
                        this.assertOriginalInputIsNotPublisher(originalInputIsPublisher);
                    }
                }
            }
        }
    }

    if (function == null) {
        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);
            if (function.isInputTypePublisher()) {
                this.assertOriginalInputIsNotPublisher(originalInputIsPublisher);
            }
        }
    }
}
```

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

```
if (function == null) {
    if (StringUtils.hasText((String)message.getHeaders().get("spring.cloud.function.definition"))) {
        function = this.functionFromDefinition((String)message.getHeaders().get("spring.cloud.function.definition"));
        if (function.isInputTypePublisher()) {
            this.assertOriginalInputIsNotPublisher(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);
        if (function.isInputTypePublisher()) {
            this.assertOriginalInputIsNotPublisher(originalInputIsPublisher);
        }
    } else if (StringUtils.hasText(this.functionProperties.getRoutingExpression())) {
        function = this.functionFromExpression(this.functionProperties.getRoutingExpression(), message);
    } else {
        if (!StringUtils.hasText(this.functionProperties.getDefinition())) {
            throw new IllegalStateException("Failed to establish route, since neither were provided: 'spring.cloud.function.definition' as Message header or as application property or 'spring.cloud.function.routing-expression' as Message header or as application property");
        }
    }
}
```

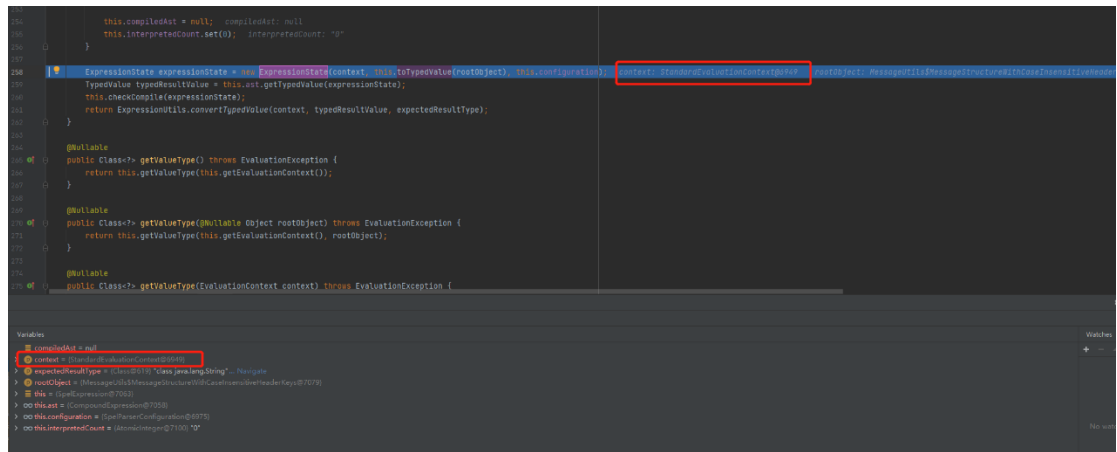
随后对传入的 header 进行解析处理

```
private FunctionInvocationWrapper functionFromExpression(String routingExpression, Object input) {
    Expression expression = this.functionParser.parseExpression(routingExpression);
    if (input instanceof Message) {
        input = MessageUtils.toCaseInsensitiveHeadersStructure((Message)input);
    }

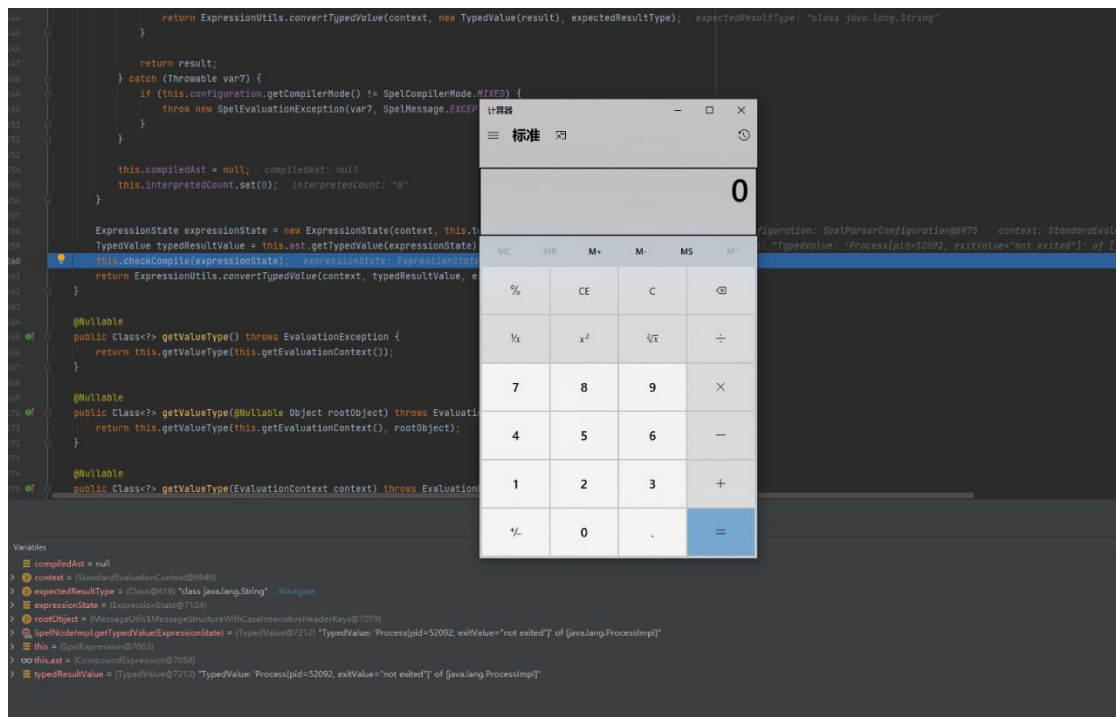
    String functionName = (String)expression.getValue(this.evalContext, input, String.class);
    Assert.hasText(functionName, "Message failed to resolve function name based on routing expression '" + this.functionProperties.getRoutingExpression() + "'");
    FunctionInvocationWrapper function = (FunctionInvocationWrapper)this.functionDefinitionLookup(functionName);
    Assert.notNull(function, "Message failed to lookup function to route to based on the expression '" + this.functionProperties.getRoutingExpression() + "' which resolved to '" + functionName + "' function name.");
    if (logger.isDebugEnabled()) {
        logger.info("Resolved function from provided [routing-expression] '" + routingExpression);
    }

    return function;
}
```

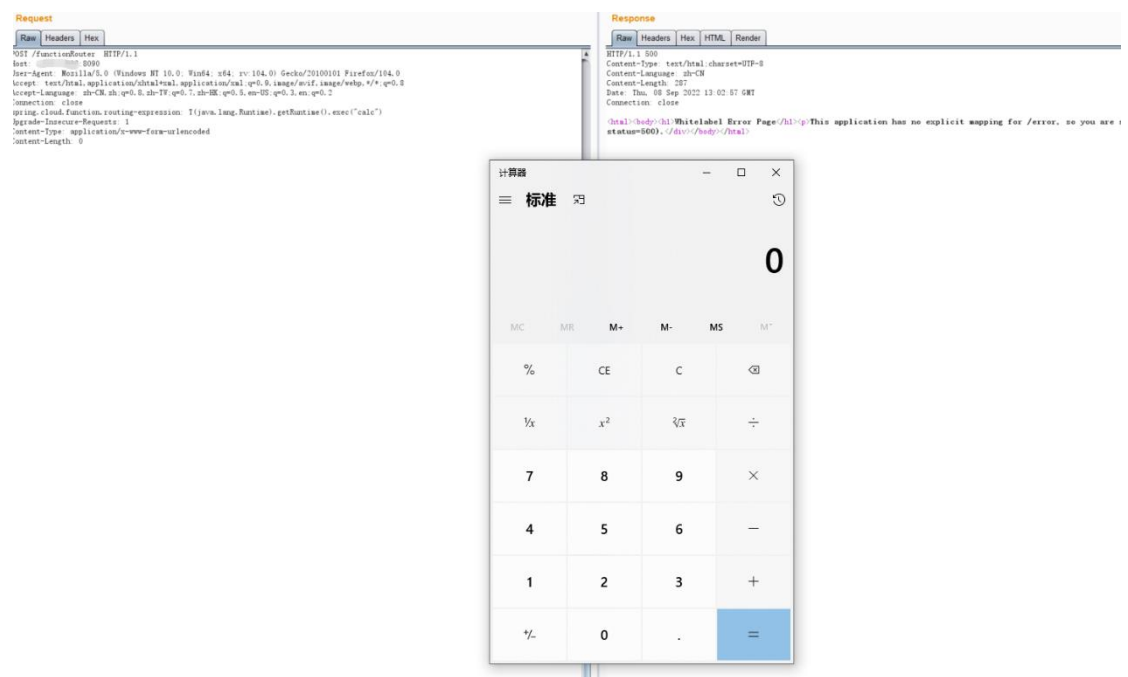
后续跟进发现对 Spel 表达式进行解析的方法就是 `StandardEvaluationContext`



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