# shiro认证绕过漏洞分析(CVE-2020-13933)

## 0x00 什么是shiro

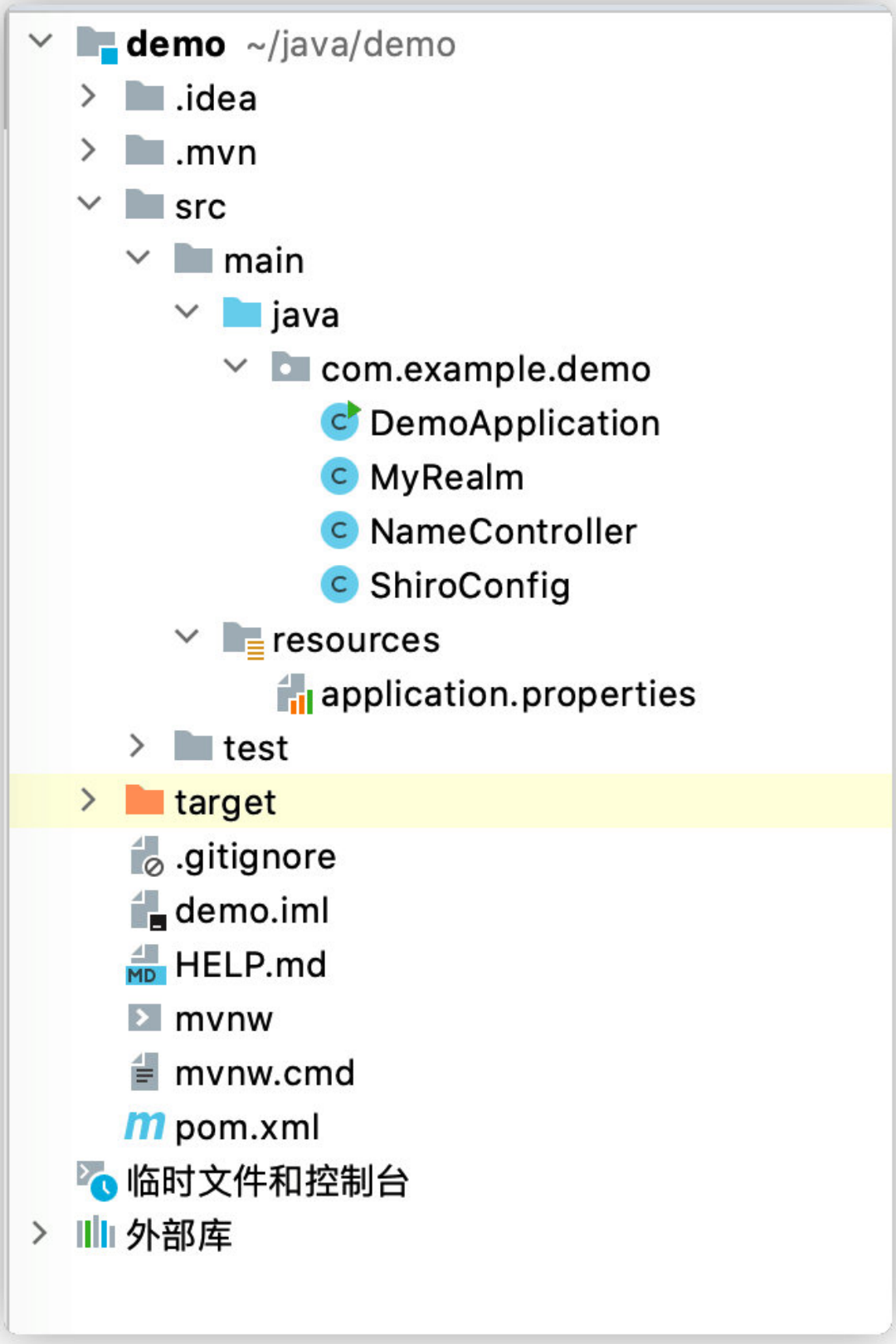
Apache Shiro是一个强大且易用的Java安全框架,执行身份验证、授权、密码和会话管理。使用Shiro的易于理解的API,您可以快速、轻松地获得任何应用程序,从最小的移动应用程序到最大的网络和企业应用程序。

## 0x01 环境搭建

- springboot
- shiro1.5.3

1、去`https://start.spring.io/`使用springboot脚手架搭建一个java8的springboot项目

2、用IDEA打开该项目，目录结构与文件如下：




3、创建`MyRealm`、`NameController`、`ShiroConfig`三个class文件。

4、pom.xml增加对应的`dependency`

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.3.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>demo</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <optional>true</optional>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-web</artifactId>
            <version>1.5.3</version>
        </dependency>
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-spring</artifactId>
            <version>1.5.3</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
```

```
            <version>5.2.5.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <fork>true</fork>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

ShiroConfig.java代码如下：

```
package com.example.demo;

import org.apache.shiro.mgt.SecurityManager;
import org.apache.shiro.spring.web.ShiroFilterFactoryBean;
import org.apache.shiro.web.mgt.DefaultWebSecurityManager;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.LinkedHashMap;
import java.util.Map;

@Configuration
public class ShiroConfig {
    @Bean
    MyRealm myRealm(){
        return new MyRealm();
    }
    @Bean
    SecurityManager securityManager(){
        DefaultWebSecurityManager manager = new DefaultWebSecurityManager();
        manager.setRealm(myRealm());
        return manager;
    }
    @Bean
    ShiroFilterFactoryBean shiroFilterFactoryBean(){
        ShiroFilterFactoryBean bean = new ShiroFilterFactoryBean();
        bean.setSecurityManager(securityManager());
        bean.setLoginUrl("/login");
        bean.setSuccessUrl("/index");
        bean.setUnauthorizedUrl("/unauthorizedUrl");
        Map<String,String> map= new LinkedHashMap<>();
```

```
        map.put("/doLogin/","anon");
        map.put("/test/*","authc");
        bean.setFilterChainDefinitionMap(map);
        return bean;

    }
}
```

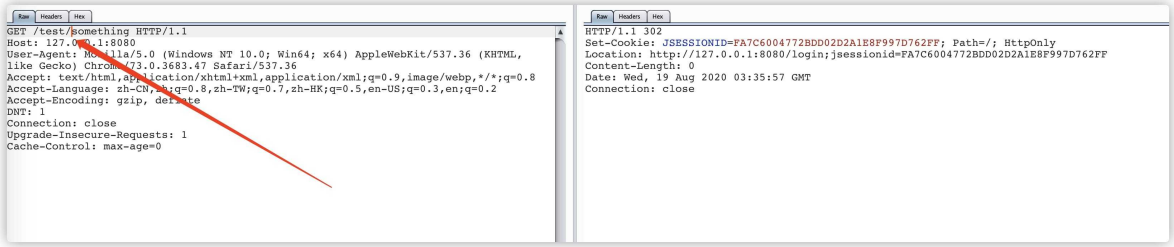
NameController代码如下:

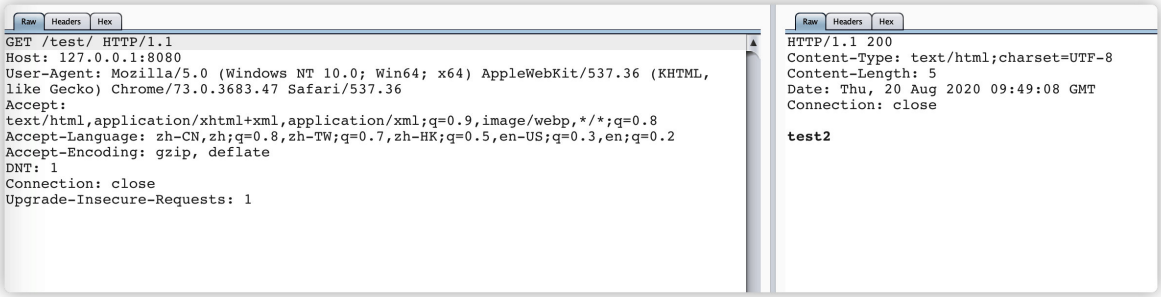
```
package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class NameController {

    @GetMapping("/test/{name}")
    public String test(@PathVariable String name){
        return "s1111";
    }

    @GetMapping("/test/")
    public String test2(){
        return "test2";
    }
}
```

MyRealm代码如下:

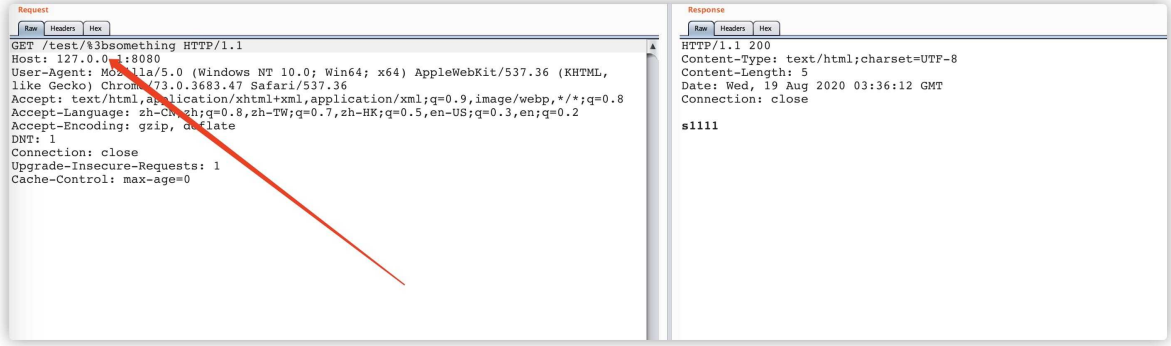
```
package com.example.demo;

import org.apache.shiro.authc.*;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;

public class MyRealm extends AuthorizingRealm {
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principalCollection) {
        return null;
    }

    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken
authenticationToken) throws AuthenticationException {
        return null;
    }
}
```

# 0x02 poc测试




原本需要认证的test/{name}可以访问到了




证明漏洞利用成功。

# 0x03 漏洞原理分析

根据shiro历史上的认证绕过漏洞，本质问题就是springboot对url的处理和shiro的处理不一致导致的认证绕过。
其中shiro的url处理的问题都出在`org/apache/shiro/web/util/WebUtils.java`类下面，在return

```
    public static String getPathWithinApplication(HttpServletRequest request) {
        return normalize(removeSemicolon(getServletPath(request) +
getPathInfo(request)));
    }
```

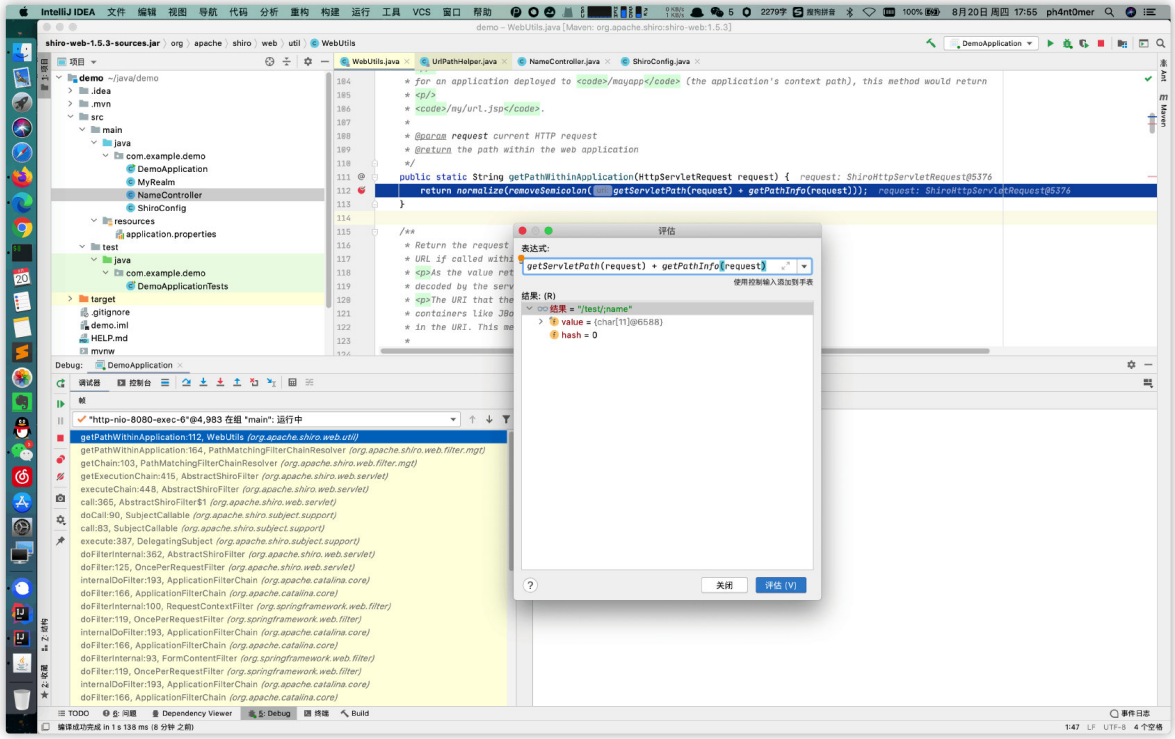
断点，
然后我们去springboot的处理url的地方进行断点，
`org/springframework/web/util/UrlPathHelper.java`。

在return uri；进行

```
    private String decodeAndCleanUriString(HttpServletRequest request, String
uri) {
        uri = removeSemicolonContent(uri);
        uri = decodeRequestString(request, uri);
        uri = getSanitizedPath(uri);
        return uri;
    }
```

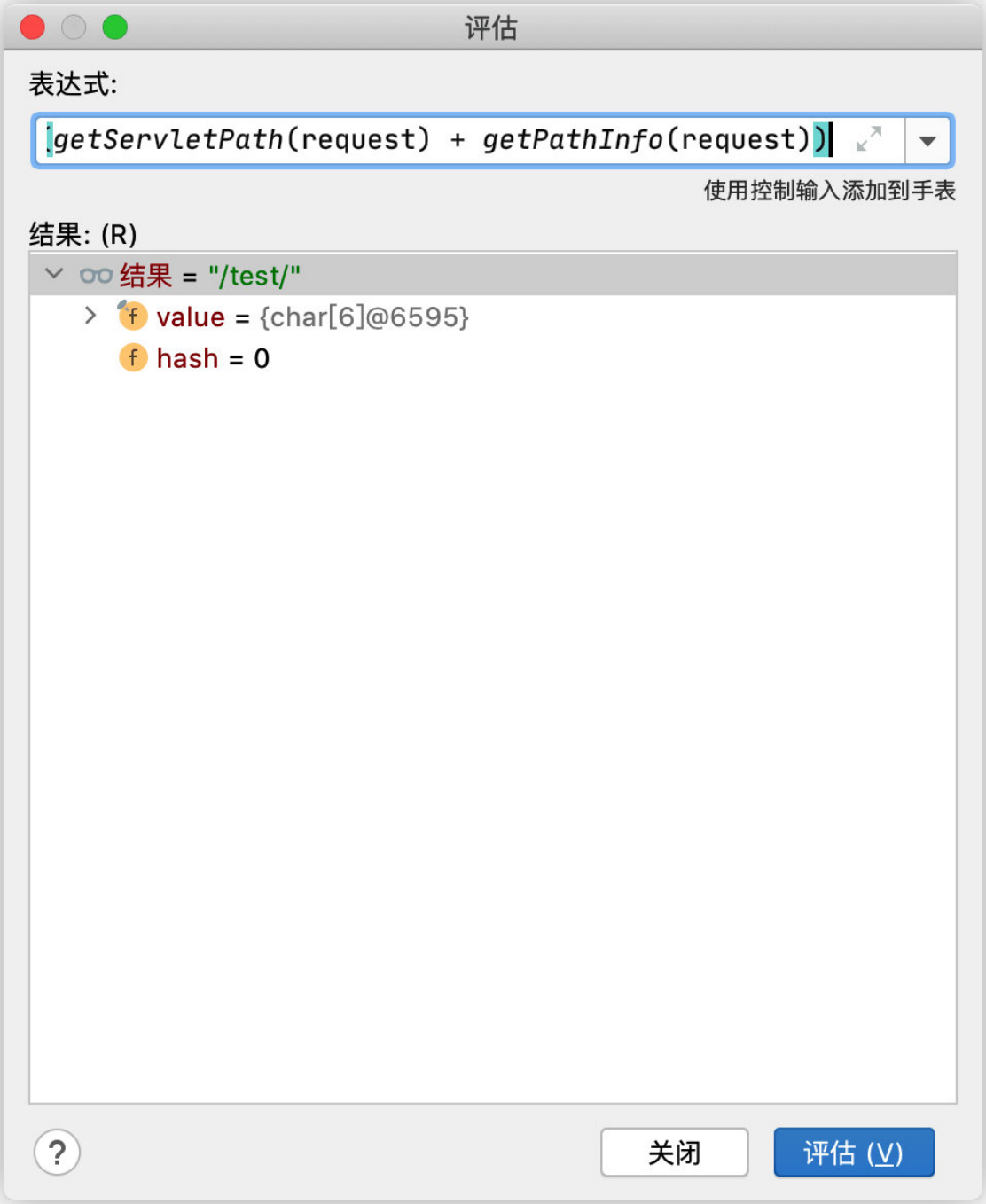
断点。

访问/test/%3bname，先来到shiro的url处理块，




根据图纸可知，获取到的url在处理前已经进行了一次urldecode。然后在进入`removeSemicolon`操作，最后结果集在交给`normalize`操作。

根据计算器可知removeSemicolon会把url里;后面的内容给删除(包括;)



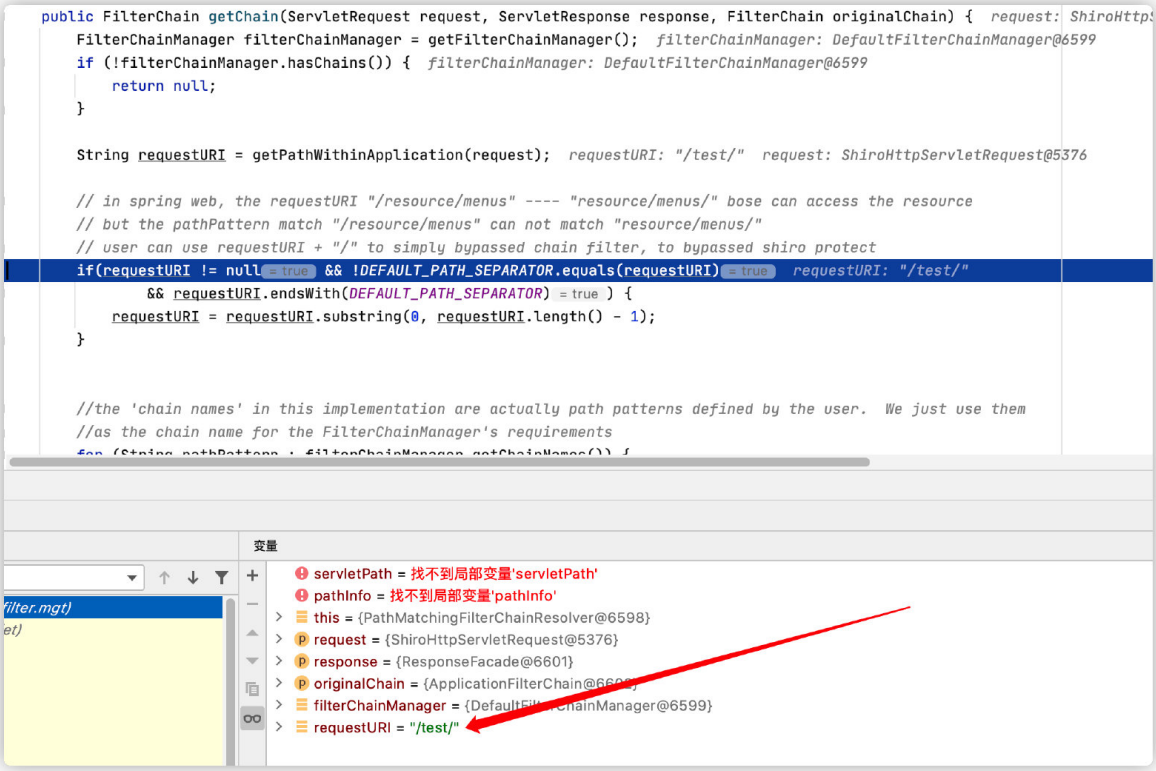

跟进removeSemicolon看看。

实现代码如下:

```
    private static String removeSemicolon(String uri) {
        int semicolonIndex = uri.indexOf(';');
        return (semicolonIndex != -1 ? uri.substring(0, semicolonIndex) : uri);
    }
```

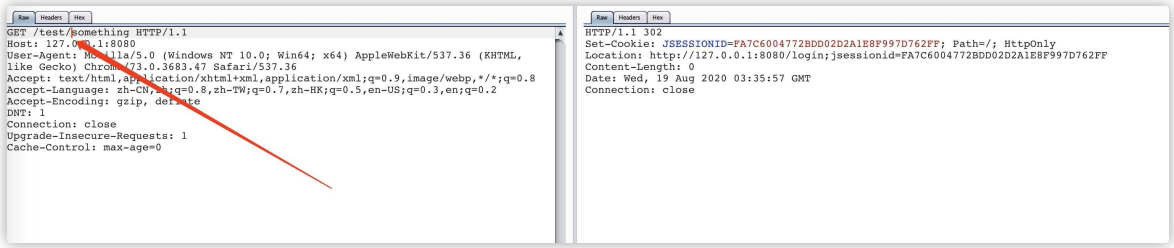
确实是把;后面的内容给删除(包括;)了。

一路F8，最后果然把/test/赋值给requestURI变量了。

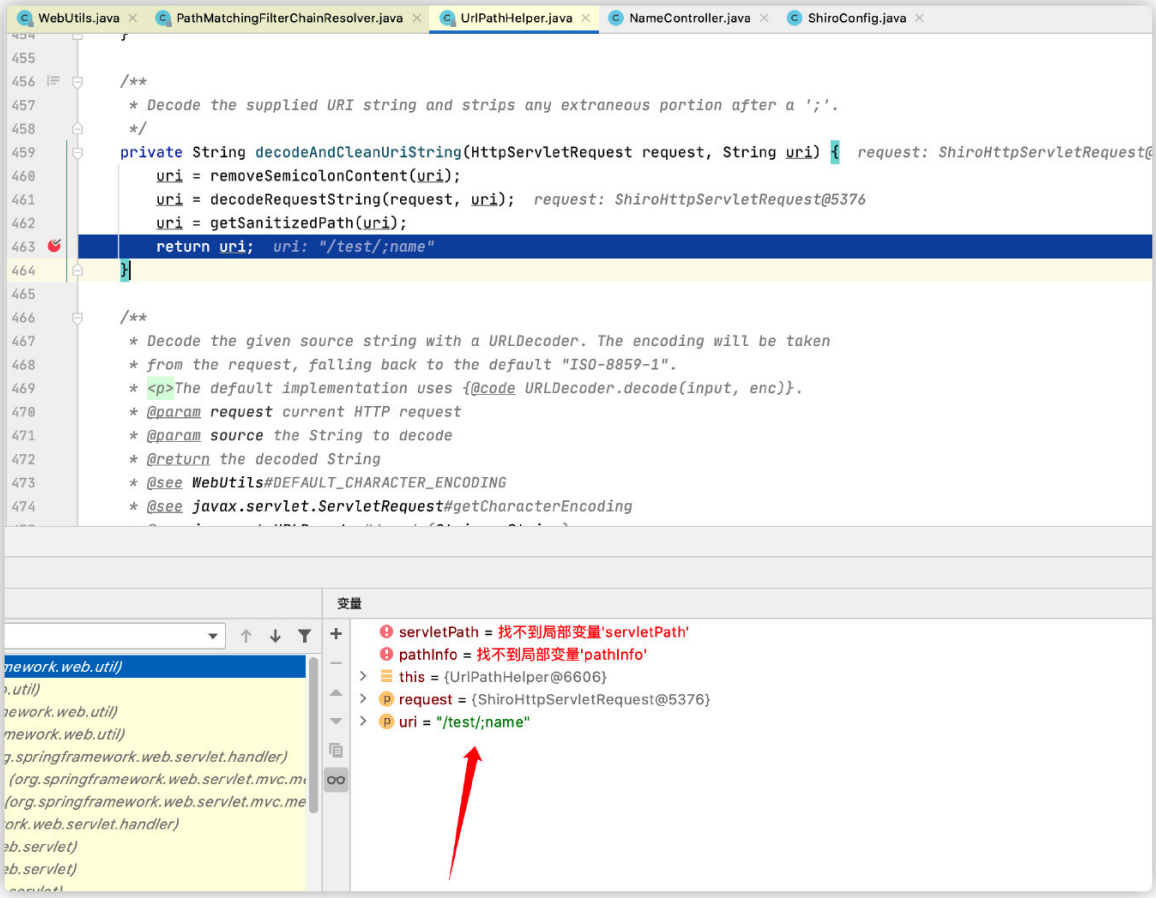



根据测试当初访问/test/


可以看到，如果test目录的话，是默认有权限访问的，但是/test/后面的路由是需要验证的。




接着我们来到springboot，看看springboot是怎么处理URL的问题。




uri取到的是`/test/;name`，可以看到springboot对url做了三个操作后才返回的，`removeSemicolonContent`,`decodeRequestString`,`getSanitizedPath`。

- removeSemicolonContent 是把（url未解码前的uri里的;后面的内容给删除）
- decodeRequestString把uri进行urldecode编码
- getSanitizedPath 是把"//" 替换成 "/"

这是简单对比下shiro的对url的操作顺序:

- uri 进行urldecode
- uri 删除;后面的内容，包括;

因为shiro的处理和springboot的处理顺序不同，导致我们构造的poc在shiro侧理解的是访问的/test/，/test/我们本身就没有限制权限，放过了这个原本需要认证权限的请求，而springboot侧则是访问的是/test/;name，然后springboot把;name当做一个字符串去寻找对应的路由，返回了对应的字符串。

# 0x04 总结

此漏洞主要是两个组件之间对某个关键信息处理的逻辑未进行统一约定，可以理解成前后端对同一个信息处理的不同步导致的安全漏洞。

同样的认证绕过在shiro历史上出现过好几次，具体可以去参考链接学习。

# 0x05 参考

- https://www.freebuf.com/vuls/231909.html
- https://xlab.tencent.com/cn/2020/06/30/xlab-20-002/