SSRF漏洞

1. SSRF漏洞

1.1. 原理

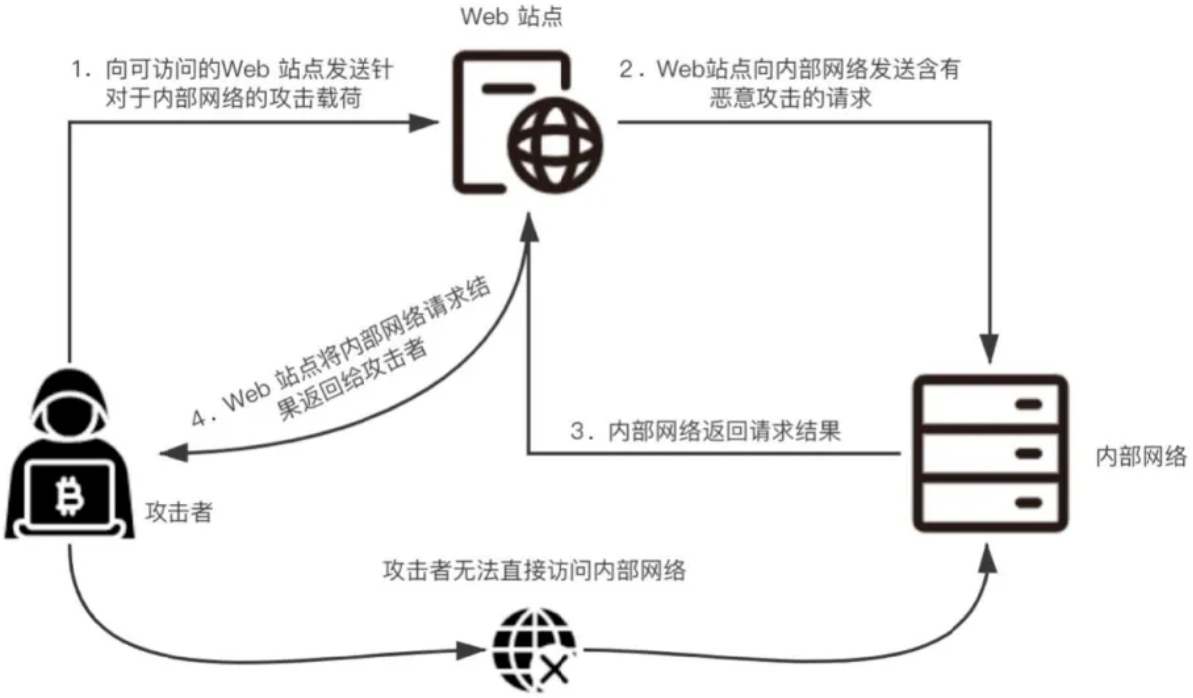
CSRF（Cross-site request forgery）：跨站请求伪造，客户端请求伪造。

SSRF（Server-Side Request Forge）：服务端请求伪造，是一种由攻击者构造形成由服务器发起请求的一个安全漏洞。是由攻击者构造的 payload 传给服务端，服务端对传回的 payload 未作处理直接执行

后造成的漏洞，一般用于在内网探测或攻击内网服务。利用服务器发起请求，攻击内网或敏感资源。

SSRF 形成的原因大都是由于服务端提供了从其他服务器应用获取数据的功能，且没有对目标地址做过滤和限制。

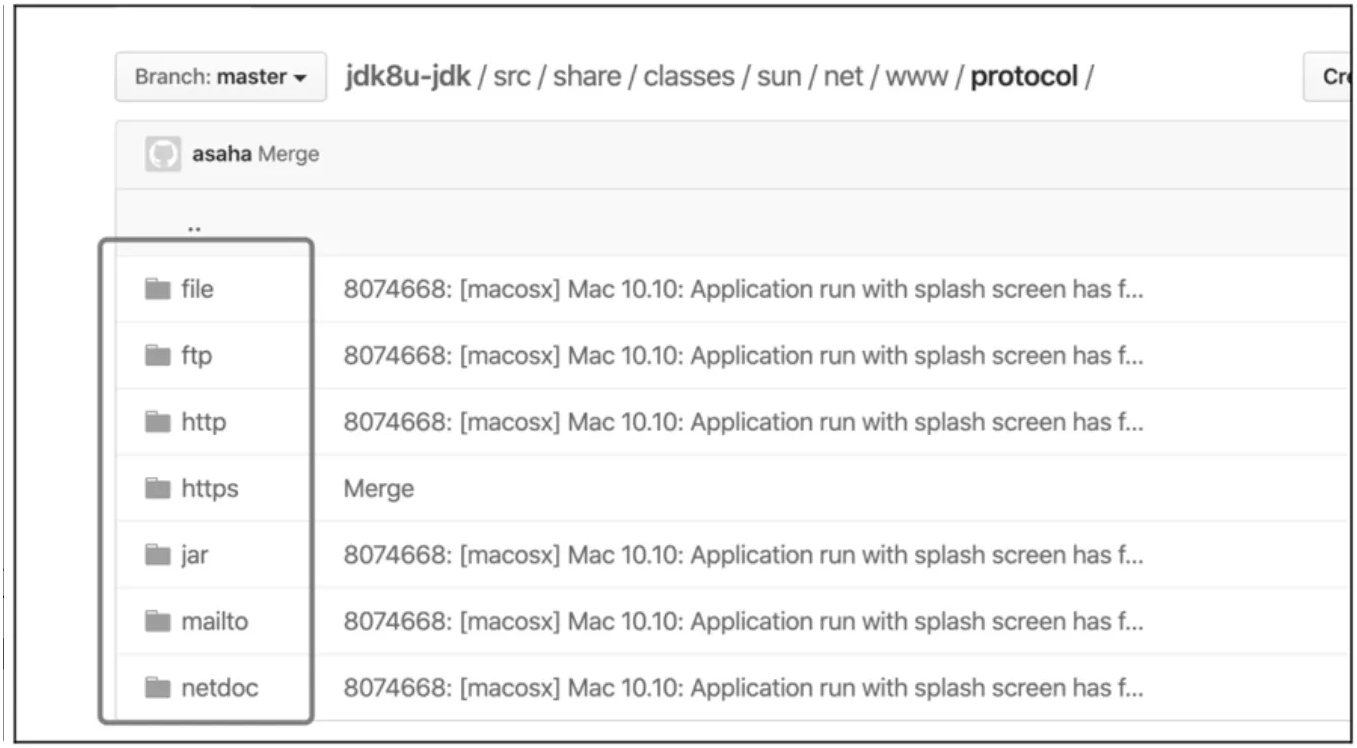
一般情况下，SSRF的攻击目标是从外网无法访问的内网系统。



SSRF 攻击流程。攻击者首先向可直接访问的 Web 站点发送攻击载荷，该攻击载荷的攻击对象为内部网络。然后， Web 站点作为“中间人”，将包含有恶意攻击请求的请求传递给内部网络，内部网络接受请求并处理后，将结果返回给 Web 站点。最后， Web 站点将内部网络返回的结果传递给攻击者，以此达到攻击内部网络的目的。

在 Java 网络请求支持的协议很多，包括：http、https、file、ftp、mailto、jar 及 netdoc协议，如下图。

Java 支持的协议可以在 sun.net.www.protocol 包下看到



Java SSRF支持sun.net.www.protocol下所有的协议

1.2. 漏洞危害

主要是用来进行内网探测，也可以说是内网信息收集，探测存活主机和开放端口。

1. 扫描内网、向内部任意主机的任意端口发送精心构造的攻击载荷请求；获取内网主机、端口和banner信息。
2. 对内网web应用进行指纹识别。
3. 攻击内网的web应用，主要是使用get参数就可以实现的攻击（比如struts2,sq注入等）。
4. 利用 file 协议读取文件。
5. 拒绝服务攻击。
6. 各种伪协议进行探测：http、file、dict、ftp、gopher等。

   http:192.168.64.14/phpmyadmin/

   file:///D:/www.txt

   dict://192.168.64.144:3306/info

   ftp://192.168.64.144:21

1.3. 容易出现漏洞的地方

基本都是发起 URL 请求的地方：

1. 分享：通过 URL 地址分享网页内容
2. 通过网址在线翻译，比如nginx.org
3. **图片加载与下载**：通过 URL 地址加载或下载图片
4. 图片、文章收藏功能
5. 从 URL 关键字中寻找：share、wap、url、link、src、source、target、u、display、sourceURL、imageURL、domain
6. 云服务器商，从远程服务器请求资源（各种网站数据库操作）

代码审计时，需要关注发起 HTTP 请求的类及函数，部分如下：

HttpURLConnection.getInputStream	HttpClients.execute
URLConnection.getInputStream	HttpClient.execute
Request.Get.execute	BasicHttpEntityEnclosingRequest()
Request.Post.execute	DefaultBHttpClientConnection()
URL.openStream	BasicHttpRequest()
ImageIO.read	OkHttpClient.newCall.execute

2. 漏洞审计点

下面是几种常见的可以发起网络请求，并且会导致SSRF漏洞的错误写法。

2.1. URLConnection

urlConnection是一个抽象类，表示指向URL指定资源的活动链接，其本身依赖于 Socket 类实现网络连接。它有两个直接子类，分别是HttpURLConnection和JarURLConnection。支持的协议有：file ftp mailto http https jar netdoc gopher。在默认情况下，urlConnection 的参数没有有效控制时会引起SSRF漏洞。

```java
String url = "file:///Users/queen/Desktop/1.txt";
//构造一个 URL 对象
URL u = new URL(url);
//调用 URL.openConnection() 方法来获取一个 URLConnection 实例
URLConnection urlConnection = u.openConnection();
//调用 getInputStream() 拿到请求的响应流，此时已经建立连接。
BufferedReader in = new BufferedReader(new InputStreamReader(urlConnection
.getInputStream())); //发起请求
String inputLine;
StringBuffer html = new StringBuffer();
while ((inputLine = in.readLine()) != null) {
    html.append(inputLine);
}
System.out.println("html:" + html.toString());
in.close();
```

URLStreamHandler 是一个抽象类，每个协议都有继承它的子类 —— Handler。Handler定义了该如何去打开一个连接，即 openConnection()。 如果直接传入一个URL 字符串，会在构造对象时，根据 protocol 自动创建对应的 Handler 对象。

在调用 URL.openConnection() 获取 URLConnection 实例的时候，真实的网络连接实际上并没有建立，只有在调用 URLConnection.connect() 方法后才会建立连接。

```java
public URLConnection openConnection() throws java.io.IOException {
    return handler.openConnection( u: this);
}
```

openConnection() 方法

2.2. HttpURLConnection

HttpURLConnection是Java的标准类，它继承自URLConnection，可用于向指定网站发送GET请求与POST请求。同样的，在没有过滤的默认情况下其会产生SSRF漏洞。

```java
String url = "https://www.baidu.com";
URL u = new URL(url);
URLConnection urlConnection = u.openConnection();
HttpURLConnection httpUrl = (HttpURLConnection) urlConnection;
BufferedReader base = new BufferedReader(new InputStreamReader(httpUrl.getInputStream(), "UTF-8"));
StringBuffer html = new StringBuffer();
while ((htmlContent = base.readLine()) != null) {
    html.append(htmlContent);
}
base.close();
System.out.println("探测：  "+url);
System.out.println("----------Response-----------");
System.out.println(html);
```

2.3. Request

Request与Python中的request对象类似，其主要用来发送HTTP请求。在没有过滤的默认情况下会产生SSRF漏洞。

```java
String html = Request.Get("https://www.baidu.com/").execute().returnContent().toString();
```

2.4. openStream

通过URL对象的openStream()方法，能够得到指定资源的输入流。这时如果URL对象可控，则会产生SSRF漏洞。

```java
String url = "https://www.baidu.com";
URL u = new URL(url);
System.out.println("探测：" +url);
System.out.println("----------Response----------");
BufferedReader base = new BufferedReader(new InputStreamReader(u.openStream(), "UTF-8")); //获取url中的资源
StringBuffer html = new StringBuffer();
while ((htmlContent = base.readLine()) != null) {
    html.append(htmlContent); //htmlContent添加到html里面
}
System.out.println(html);
```

2.5. HttpClient

HttpClient是Apache Jakarta Common下的一个子项目，用来提供高效的、最新的、功能丰富的支持HTTP协议的客户端编程工具包，并且它支持HTTP协议最新的版本和建议。但是在默认情况下，其也会产生SSRF漏洞。

```java
String htmlContent;
String url = "https://www.baidu.com";
CloseableHttpClient client = HttpClients.createDefault();
HttpGet httpGet = new HttpGet(url);
System.out.println("探测：" +url);
System.out.println("----------Response----------");
HttpResponse httpResponse = client.execute(httpGet); //发起请求
BufferedReader base = new BufferedReader(new InputStreamReader(
        httpResponse.getEntity().getContent()));
StringBuffer html = new StringBuffer();
while ((htmlContent = base.readLine()) != null) {
    html.append(htmlContent); //htmlContent添加到html里面
}
System.out.println(html);
```

3. 漏洞审计案例

3.1. SecExample

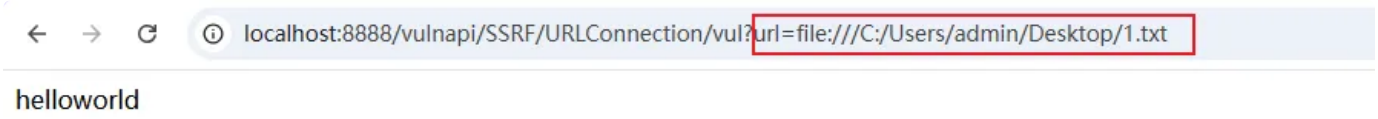
您请求的结果为:

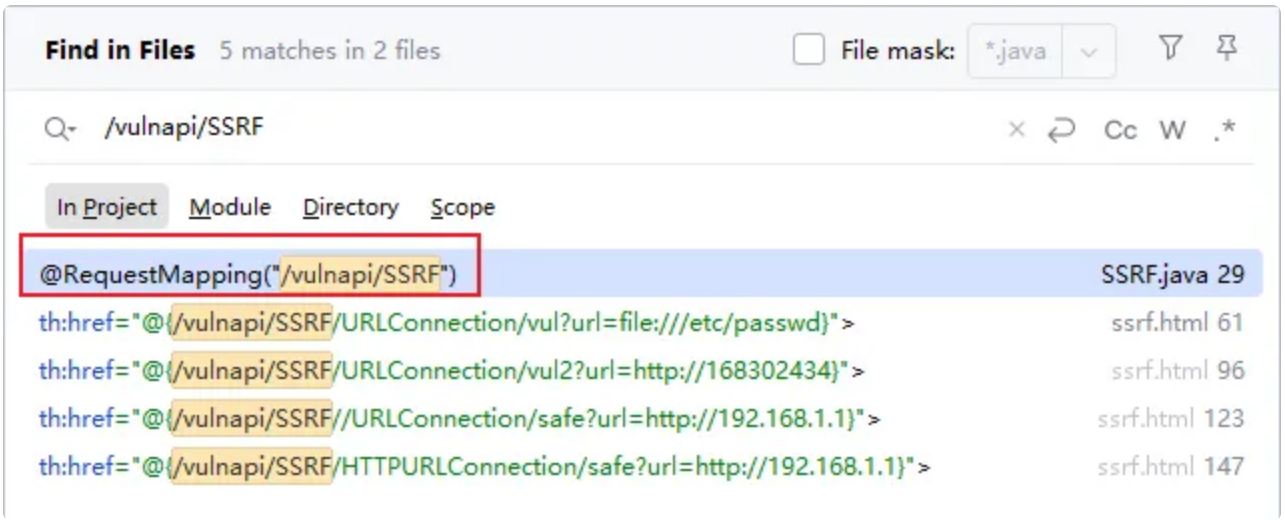
<!DOCTYPE html><!--STATUS OK--><html> <head><meta http-equiv=content-type content=text/html;charset=utf-8><meta http-equiv=X-UA-Compatible content=IE=Edge><meta content=always name=referrer><link rel=stylesheet type=text/css href=http://s1.bdstatic.com/r/www/cache/bdorz/baidu.min.css><title>百度一下，你就知道</title></head> <body link=#0000cc>   <img hidefocus=true src=//www.baidu.com/img/bd_logo1.png width=270 height=129> <form id=form name=f action=//www.baidu.com/s class=fm> <input type=hidden name=bdorz_come value=1> <input type=hidden name=ie value=utf-8> <input type=hidden name=f value=8> <input type=hidden name=rsv_bp value=1> <input type=hidden name=rsv_idx value=1> <input type=hidden name=tn value=baidu><input id=kw name=wd class=s_ipt value maxlength=255 autocomplete=off autofocus><input type=submit id=su value=百度一下 class="bg s_btn"> </form>   新闻 hao123 地图 视频 贴吧 <noscript> 登录 </noscript> <script>document.write('登录');</script> 更多产品   关于百度 About Baidu ©2017 Baidu 使用百度前必读 意见反馈 京ICP证030173号 <img src=//www.baidu.com/img/gs.gif>   </body> </html>

您请求的结果为:

<!doctype html><html lang="en"><head> <meta charset="UTF-8"> <title>Java漏洞靶场</title> <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/semantic-ui@2.4.2/dist/semantic.min.css"></head><body> <h1>Java漏洞演示平台</h1> <button class="ui inverted secondary basic button">靶机源码</button>   注入漏洞-SQL注入 SQL注入通过把SQL命令插入到Web表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行指定的SQL语句 测试漏洞   注入漏洞-命令注入 RCE (remote code execution):指用户通过浏览器提交执行命令，由于服务器端没有针对执行函数做过滤，导致在没有指定绝对路径的情况下就执行命令，可能会允许攻击者通过改变 $PATH 或程序执行环境的其他方面来执行一个恶意构造的代码。 测试漏洞   注入漏洞-spel表达式注入 spel表达式注入 (Spring Expression Language):是一种功能强大的表达式语言，用于在运行时查询和操作对象图；语法上称为Unified EL，但提供了更多的特性，特别是方法调用和基本字符SpEL的生成是为了给Spring社区提供一种能够与Spring生态系统所有产品无缝对接，能提供一站式支持的表达式语言。 测试漏洞   XSS漏洞 XSS(Cross Site Scripting):跨站脚本攻击是指恶意攻击者往Web页面里插入恶意Script代码,当用户浏览该页之时,嵌入其中Web里面的Script代码会被执行,从而达到恶意攻击用户的目的 测试漏洞   CSRF漏洞 CSRF(Cross-site request forgery):CSRF，跨站请求伪造，在受害者通过浏览器登录某个恶意URL的时候，通过伪造请求达到跨站请求伪造（常见于商城类网站或者自己开发的会员系统） 测试漏洞   SSRF漏洞 SSRF(Server-Side Request Forgery):服务器端请求伪造是一种由攻击者构造形成由服务端发起请求的一个安全漏洞。一般情况下，SSRF攻击的目标是从外网无法访问的内部系统。 测试漏洞   CORS漏洞 CORS(Cross-origin resource sharing)，因为出于安全的考虑,浏览器不允许Ajax调用当前源之外的资源，即浏览器的同源策略，但一个请求url的协议、域名、端口三者之间任意一个与当前页面不同即为跨域，它允许浏览器向跨源服务器发送XMLHttpRequest请求，从而克服AJAX只能同源使用的限制 测试漏洞   反序列化漏洞-Fastjson反序列化 序列化和反序列化本身并不存在问题。但当输入的反序列化的数据可被用户控制，那么攻击者即可通过构造恶意输入，让反序列化产生非预期的对象，在此过程中执行构造的任意代码。 测试漏洞   验证码相关漏洞 短信回显
短信轰炸
前端绕过验证
验证码爆破 测试漏洞 </body></html>

3.2. Hello






```java
@Api("SSRF")  1 usage
@RestController
@RequestMapping("/vulnapi/SSRF")
public class SSRF {
    Logger log = LoggerFactory.getLogger(SSRF.class);  5 usages

    /**
     * @poc http://127.0.0.1:8888/SSRF/URLConnection/vul?url=http://www.baidu.com
     * @poc http://127.0.0.1:8888/SSRF/URLConnection/vul?url=file:///etc/passwd
     */
    @ApiOperation(value = "vul: HTTPURLConnection类")  no usages
    @GetMapping("/URLConnection/vul")
    public String URLConnection(String url) {
        log.info("[vul] SSRF: " + url);
        return HttpClientUtils.URLConnection(url);
    }
```

4. 漏洞绕过

1. 采用短网址绕过，也叫做url短链接。比如百度短地址https://dwz.cn/
2. 采用进制转换。127.0.0.1　八进制：0177.0.0.1　　十六进制：0x7f.0.0.1　　十进制：2130706433

5. 漏洞修复

对于SSRF漏洞的修复比较简单，总结下来主要包括以下几点：

（1）正确处理302跳转（在业务角度看，不能直接禁止302，而是对跳转的地址重新进行检查）。

（2）限制协议只能为 HTTP/HTTPS，防止跨协议。

（3）设置内网IP黑名单（正确判定内网IP、正确获取host）。

（4）在内网防火墙上设置常见的Web端口白名单（防止端口扫描，则可能业务受限比较大）。

```java
private static int connectTime = 5 * 1000;
public static boolean checkSsrf(String url) {
HttpURLConnection httpURLConnection;
String finalUrl = url;
try {
    do {
        if(!Pattern.matches("^https?://.*/.*$", finalUrl)) { //只允许 http/https 协议
            return false;
        }
        if(isInnerIp(url)) { //判断是否为内网 ip
            return false;
        }
        httpURLConnection = (HttpURLConnection) new URL(finalUrl).openConnection();
        httpURLConnection.setInstanceFollowRedirects(false); //不跟随跳转
        httpURLConnection.setUseCaches(false); //不使用缓存
        httpURLConnection.setConnectTimeout(connectTime); //设置超时时间
        httpURLConnection.connect(); //send dns request
        int statusCode = httpURLConnection.getResponseCode();
        if (statusCode >= 300 && statusCode <=307 && statusCode != 304 && statusCode != 306) {
            String redirectedUrl = httpURLConnection.getHeaderField("Location");
            if (null == redirectedUrl)
                break;
            finalUrl = redirectedUrl; //获取到跳转之后的 url，再次进行判断
        } else {
            break;
        }
    } while (httpURLConnection.getResponseCode() != HttpURLConnection.HTTP_OK);//如果没有返回 200，则继续对跳转后的链接进行检查
    httpURLConnection.disconnect();
} catch (Exception e) {
    return true;
}
return true;
}
private static boolean isInnerIp(String url) throws URISyntaxException, UnknownHostException {
    URI uri = new URI(url);
    String host = uri.getHost(); //url 转 host
    //这一步会发送 dns 请求， host 转 ip，各种进制也会转化为常见的 x.x.x.x 的格式
    InetAddress inetAddress = InetAddress.getByName(host);
    String ip = inetAddress.getHostAddress();
```

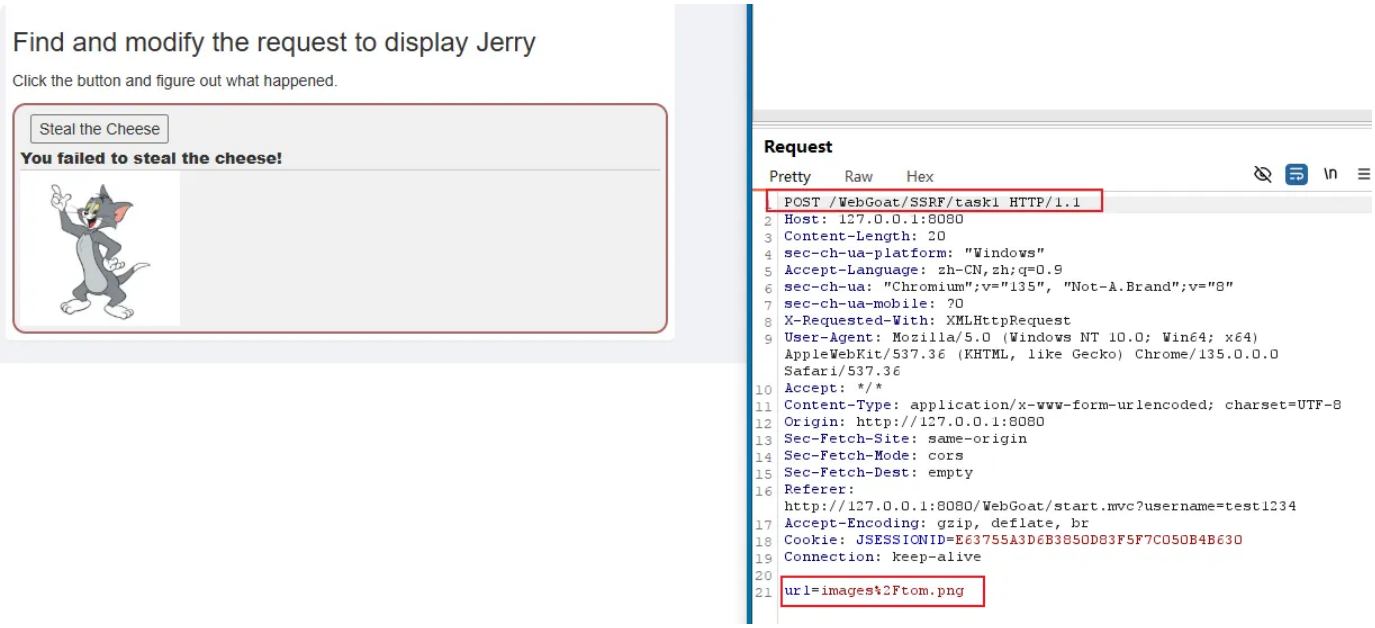
```
    String blackSubnetlist[] = {"10.0.0.0/8", "172.16.0.0/12", "192.168.0.0/16", "127.0.0.0/8"}; //内网 ip 段
    for(String subnet : blackSubnetlist) {
        SubnetUtils subnetUtils = new SubnetUtils(subnet); //commons-net 3.6
        if(subnetUtils.getInfo().isInRange(ip)) {
            return true; //如果 ip 在内网段中，则直接返回
        }
    }
    return false;
}
private static boolean isInnerIp(String url) throws URISyntaxException, UnknownHostException {
    URI uri = new URI(url);
    String host = uri.getHost(); //url 转 host
    //这一步会发送 dns 请求， host 转 ip，各种进制也会转化为常见的 x.x.x.x 的格式
    InetAddress inetAddress = InetAddress.getByName(host);
    String ip = inetAddress.getHostAddress();
    String blackSubnetlist[] = {"10.0.0.0/8", "172.16.0.0/12", "192.168.0.0/16", "127.0.0.0/8"}; //内网 ip 段
    for(String subnet : blackSubnetlist) {
        SubnetUtils subnetUtils = new SubnetUtils(subnet); //commons-net 3.6
        if(subnetUtils.getInfo().isInRange(ip)) {
            return true; //如果 ip 在内网段中，则直接返回
        }
    }
    return false;
}
```

6. Webgoat SSRF

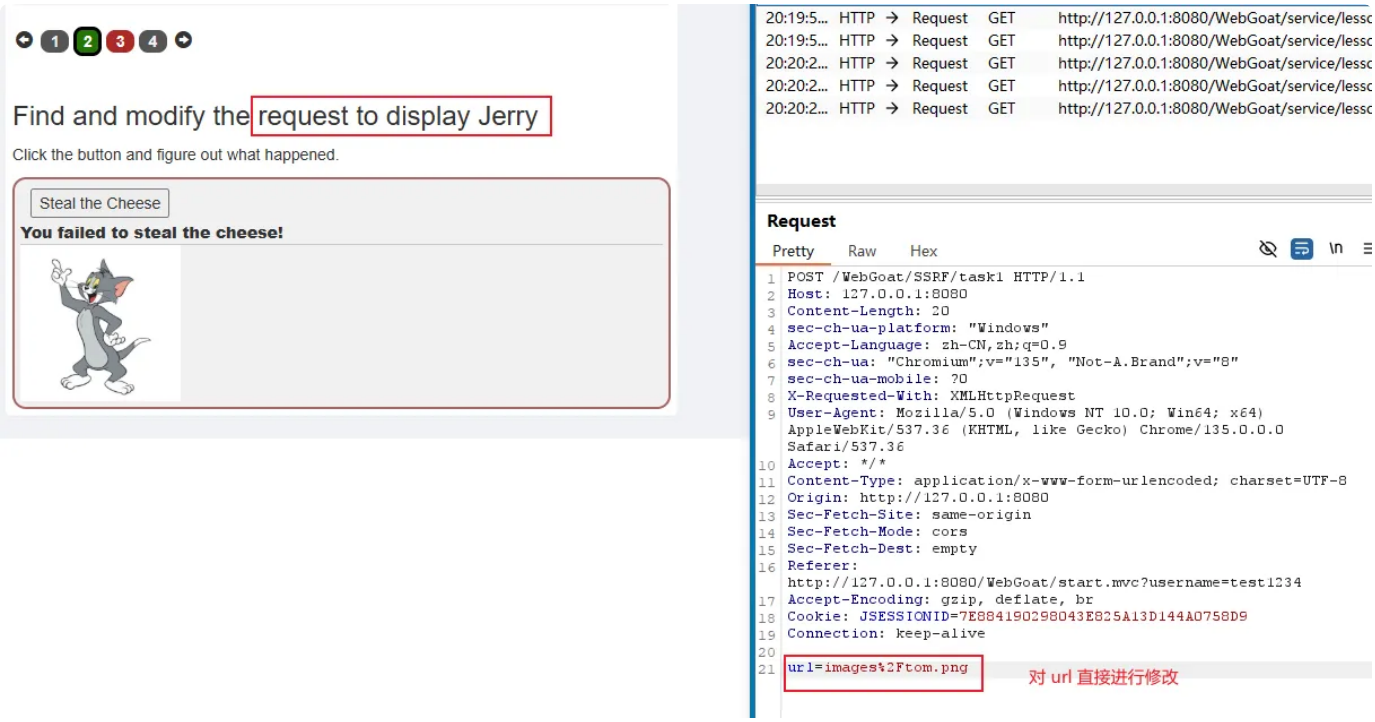
6.1. 2关 find and modify the request to display jerry

根据题意，显示jerry的图片

(1)点击按钮查看效果，返回一张tom的图片



(2)拦截发送的请求，修改为jerry



(3)发送请求，查看页面变化

Request

Pretty Raw Hex

```
1  POST /WebGoat/SSRF/task1 HTTP/1.1
2  Host: 127.0.0.1:8080
3  Content-Length: 20
4  sec-ch-ua-platform: "Windows"
5  Accept-Language: zh-CN,zh;q=0.9
6  sec-ch-ua: "Chromium";v="135", "Not-A.Brand";v="8"
7  sec-ch-ua-mobile: ?0
8  X-Requested-With: XMLHttpRequest
9  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0
   Safari/537.36
10 Accept: */*
11 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
12 Origin: http://127.0.0.1:8080
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer:
   http://127.0.0.1:8080/WebGoat/start.mvc?username=test1234
17 Accept-Encoding: gzip, deflate, br
18 Cookie: JSESSIONID=7E884190298043E825A13D144A0758D9
19 Connection: keep-alive
20
21 url=images%2Fjerry.png
```

1 2 3 4

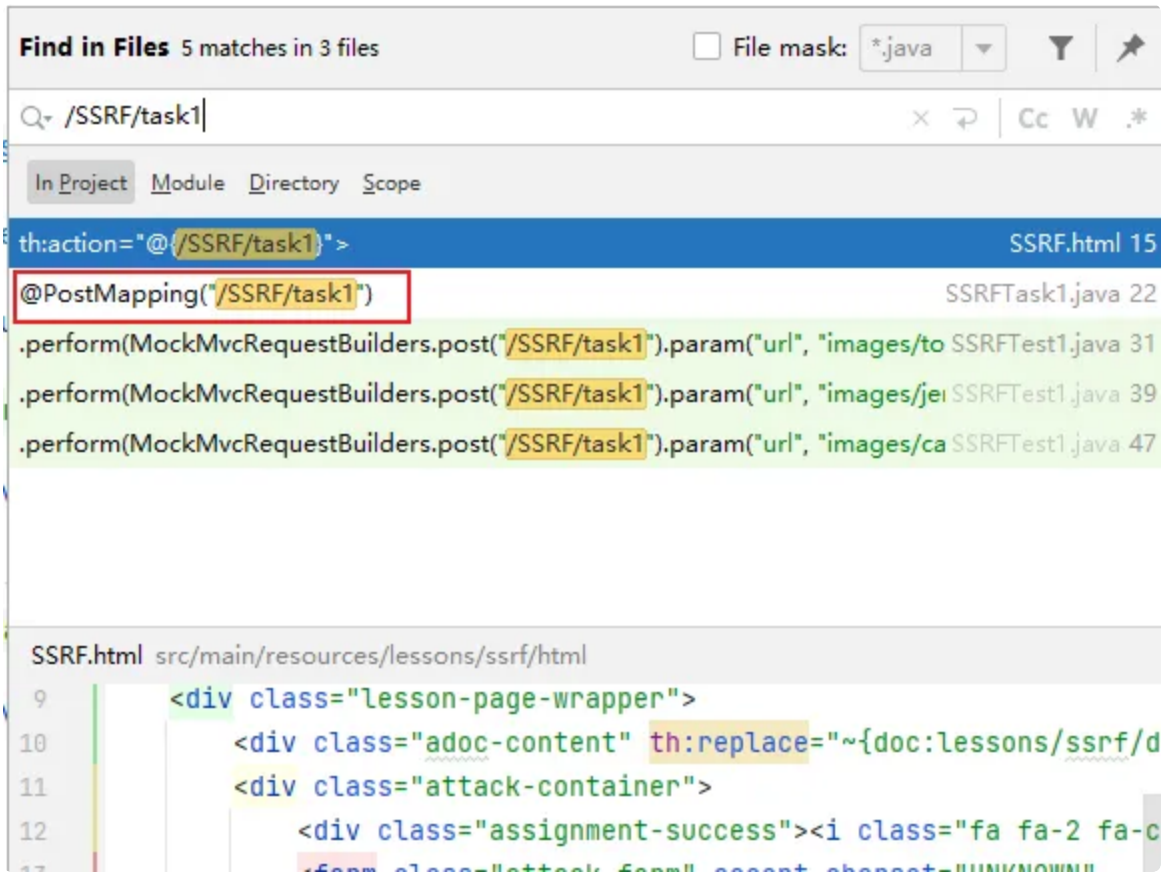
Find and modify the request to display Jerry

Click the button and figure out what happened.

Steal the Cheese

You rocked the SSRF!

（4）查看后端源码



（5）发现进入了completed 函数，通过@RequestParam 注解的方式获取的url。然后把 未经限制条件的 url 传递给 stealTheCheese 方法。在 stealTheCheese 方法中，直接对 url 进行图片匹配，也没有添加任何限制，所以存在 ssrf 漏洞。

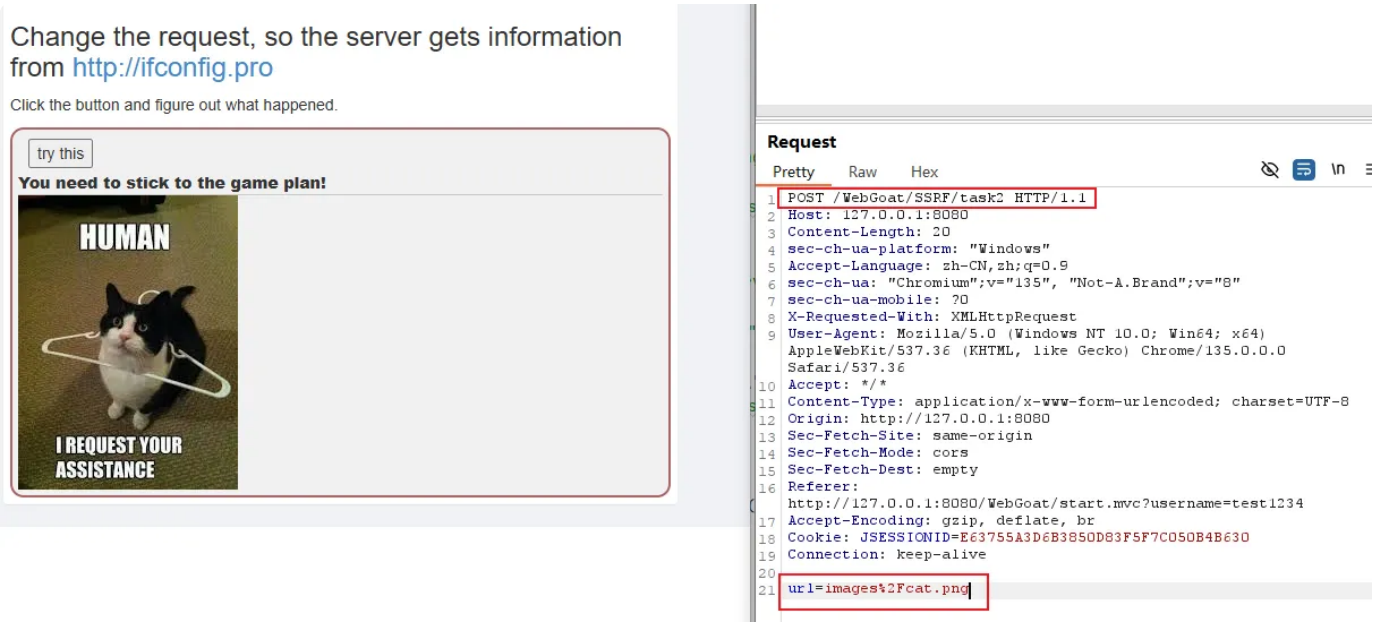
```java
@RestController
@AssignmentHints({"ssrf.hint1", "ssrf.hint2"})
public class SSRFTask1 implements AssignmentEndpoint {

  @PostMapping("/SSRF/task1")
  @ResponseBody
  public AttackResult completed(@RequestParam String url) { return stealTheCheese(url); }
  1 usage
  protected AttackResult stealTheCheese(String url) {
    try {
      StringBuilder html = new StringBuilder();

      if (url.matches( regex: "images/tom\\.png")) {
        html.append(
            "<img class=\"image\" alt=\"Tom\" src=\"images/tom.png\" width=\"25%\""
                + " height=\"25%\">");
        return failed( assignment: this).feedback( resourceBundleKey: "ssrf.tom").output(html.toString()).build();
      } else if (url.matches( regex: "images/jerry\\.png")) {
        html.append(
            "<img class=\"image\" alt=\"Jerry\" src=\"images/jerry.png\" width=\"25%\""
                + " height=\"25%\">");
        return success( assignment: this).feedback( resourceBundleKey: "ssrf.success").output(html.toString()).build();
      } else {
        html.append("<img class=\"image\" alt=\"Silly Cat\" src=\"images/cat.jpg\">");
        return failed( assignment: this).feedback( resourceBundleKey: "ssrf.failure").output(html.toString()).build();
      }
    } catch (Exception e) {
      e.printStackTrace();
      return failed( assignment: this).output(e.getMessage()).build();
    }
  }
}
```

6.2. 3关 change the request so the server gets information from http://ifconfig.pro

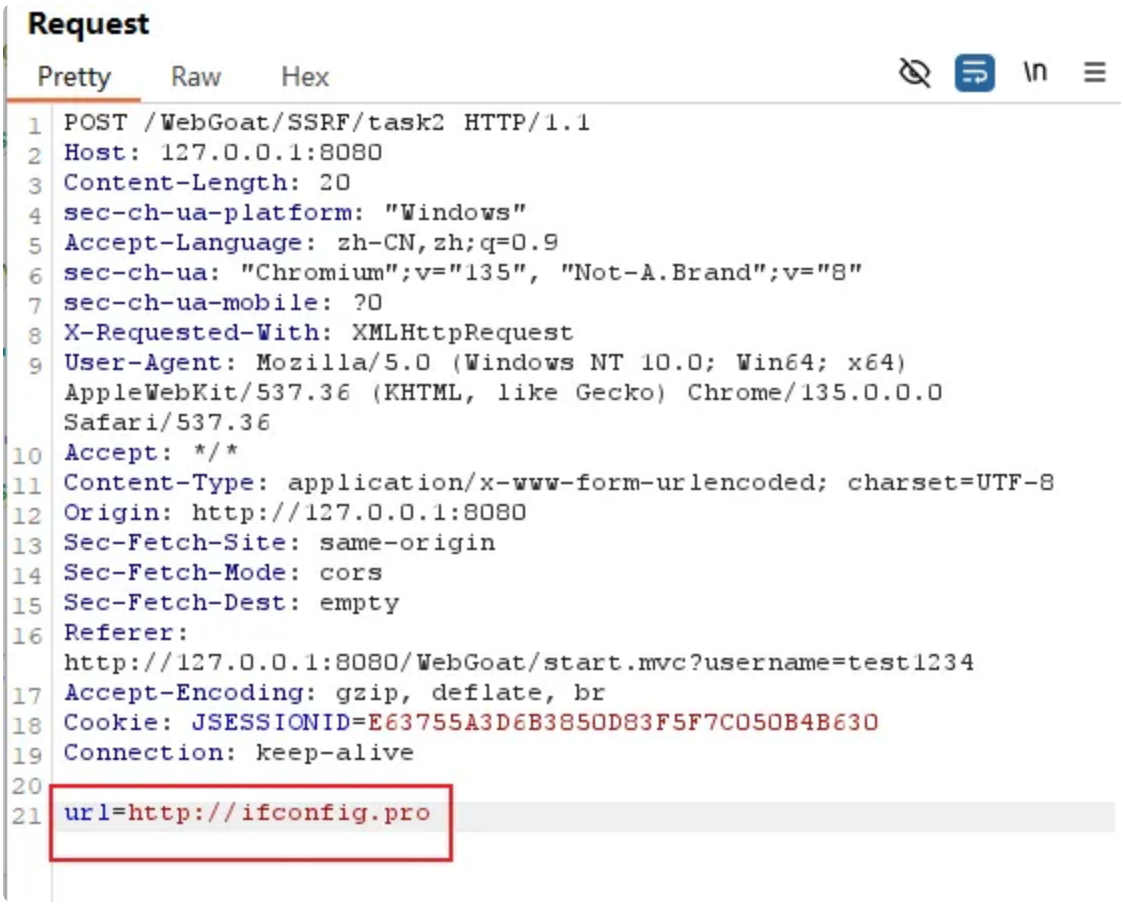
根据题意，改变请求让服务器去获取http://ifconfig.pro的信息（该网站似乎不可用）

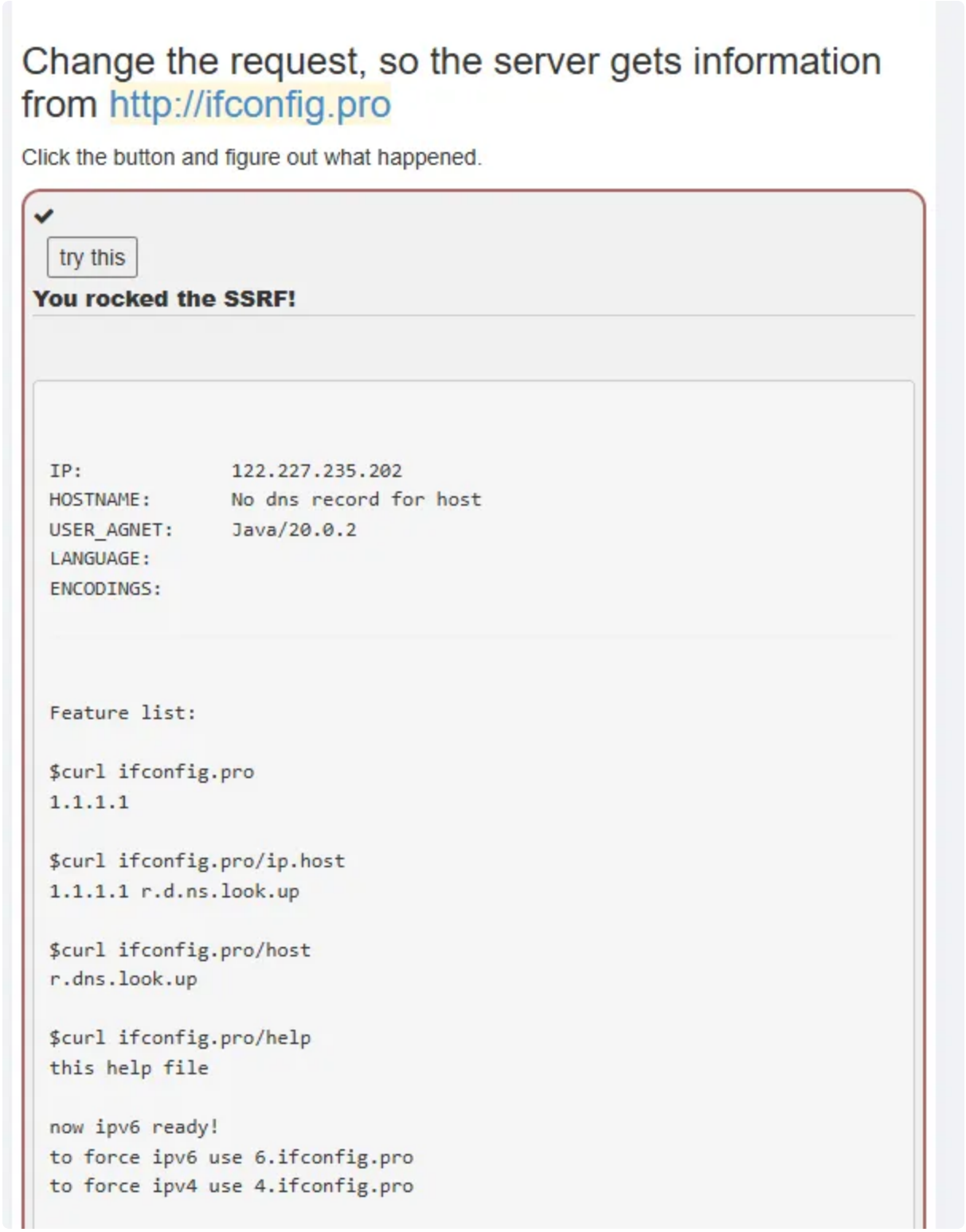
(1)点击按钮，查看效果，出来一张cat图片

(2)拦截请求，修改url

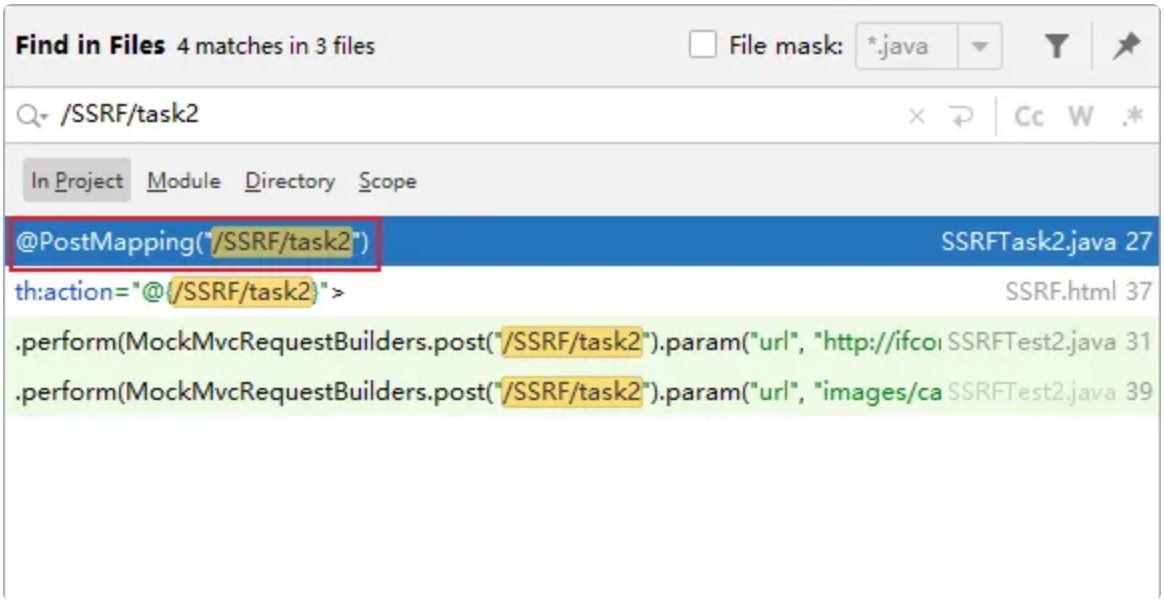


(3)修改发送的url，并检查完成效果





（4）查看后端源码，与task1同理。




```java
@RestController
@AssignmentHints({"ssrf.hint3"})
public class SSRFTask2 implements AssignmentEndpoint {

  @PostMapping("/SSRF/task2")
  @ResponseBody
  public AttackResult completed(@RequestParam String url) { return furBall(url); }

  1 usage
  protected AttackResult furBall(String url) {
    if (url.matches( regex: "http://ifconfig\\.pro")) {
      String html;
      try (InputStream in = new URL(url).openStream()) {
        html =
            new String(in.readAllBytes(), StandardCharsets.UTF_8)
                .replaceAll( regex: "\n",  replacement: "<br>"); // Otherwise the \n gets escaped in the response
      } catch (MalformedURLException e) {
        return getFailedResult(e.getMessage());
      } catch (IOException e) {
        // in case the external site is down, the test and lesson should still be ok
        html =
            "<html><body>Although the http://ifconfig.pro site is down, you still managed to solve"
                + " this exercise the right way!</body></html>";
      }
      return success( assignment: this).feedback( resourceBundleKey: "ssrf.success").output(html).build();
    }
    var html = "<img class=\"image\" alt=\"image post\" src=\"images/cat.jpg\">";
    return getFailedResult(html);
  }
```

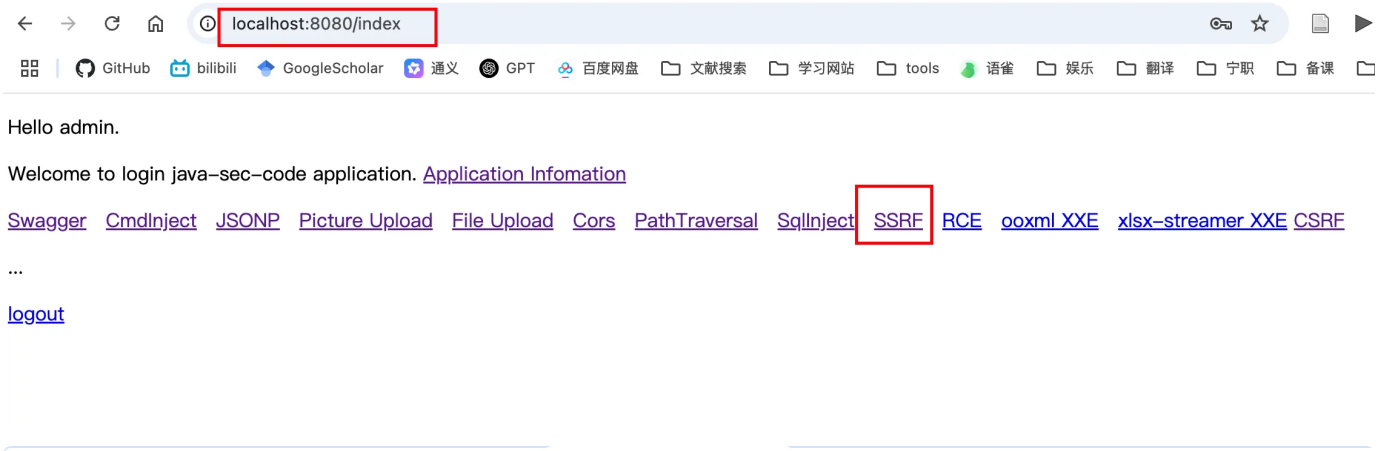
6.3. 修复

1. 配置允许web服务器从哪些域、哪些资源、哪些协议中获取资源的白名单。

2. 任何接受的用户输入都应进行验证，如果不符合预期的积极规范，则应予以拒绝。
3. 如果可能的话，不要在控制web服务器可以从哪里获取资源的函数中接受用户输入。

7. java-sec-code-master

1. 测试 ssrf 漏洞



2. 复制此 url 到项目中全局搜索，发现此 url 在前端被写死

想要修改访问的 url，需要在此页面进行修改

3. 同时找到后端对应源码。HttpServletResponse response 把处理后的结果直接返回给浏览器。**接收一个远程URL地址，下载远程文件作为图片读取，强制转成PNG格式后，再返回给客户端。**浏览器访问 /ImageIO/vul?url=http://xxx，就能直接下载或查看处理后的 PNG 图片。

   可以利用这个接口：

   - 访问内网系统，比如 http://127.0.0.1:8080/admin
   - 读取本地文件，比如 file:///etc/passwd（虽然 ImageIO.read 不一定能解析普通文本，但攻击者可以组合攻击）

- 打穿防火墙，进行横向渗透

```
28    @RestController
29    @RequestMapping("/ssrf")
30    public class SSRF {
31
```

```
@GetMapping("/ImageIO/vul")
public void ImageIO(@RequestParam String url, HttpServletResponse response) {
    try {
        ServletOutputStream outputStream = response.getOutputStream();//从 response 里拿到输出流，准备把处理后的内容写回给客户端浏览器
        ByteArrayOutputStream os = new ByteArrayOutputStream();//创建 ByteArrayOutputStream，临时存储图片处理后的字节数据
        URL u = new URL(url);
        InputStream istream = u.openStream();//创建一个 URL 对象并打开连接，拿到对应的输入流，这个流里是远程文件或者图片的原始数据
        ImageInputStream stream = ImageIO.createImageInputStream(istream); //获取文件流，专门用来读取图片格式数据的流
        BufferedImage bi = ImageIO.read(stream); //BufferedImage作为供给的，把图片流读取成一个 BufferedImage 对象
        ImageIO.write(bi, formatName: "png", os); //写成 PNG 格式，输出到 ByteArrayOutputStream 里
        InputStream input = new ByteArrayInputStream(os.toByteArray());//ByteArrayOutputStream 生成一个新的 InputStream，方便一段段读取输出
        int len;
        byte[] bytes = new byte[1024];
        while ((len = input.read(bytes)) > 0) {//写到 response 的输出流，返回给浏览器
            outputStream.write(bytes, off: 0, len);
        }
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```