此文档为《Java代码审计零基础入门到项目实战》配套教材，由【闪石星曜CyberSecurity】出品。

请勿对外泄露，一经发现严格处理！

课程学习中有任何疑问，可添加好友 Power_7089 寻求帮助，为你答疑解惑。

一、SSRF（服务端请求伪造）漏洞介绍

1、什么是SSRF漏洞

SSRF漏洞，全称`Server Side Request Forgery`（服务端请求伪造）。是一种 Web 安全漏洞，允许攻击者诱导服务器端应用程序向非预期位置发出请求。SSRF漏洞攻击的目标是从外网无法访问的内网系统。

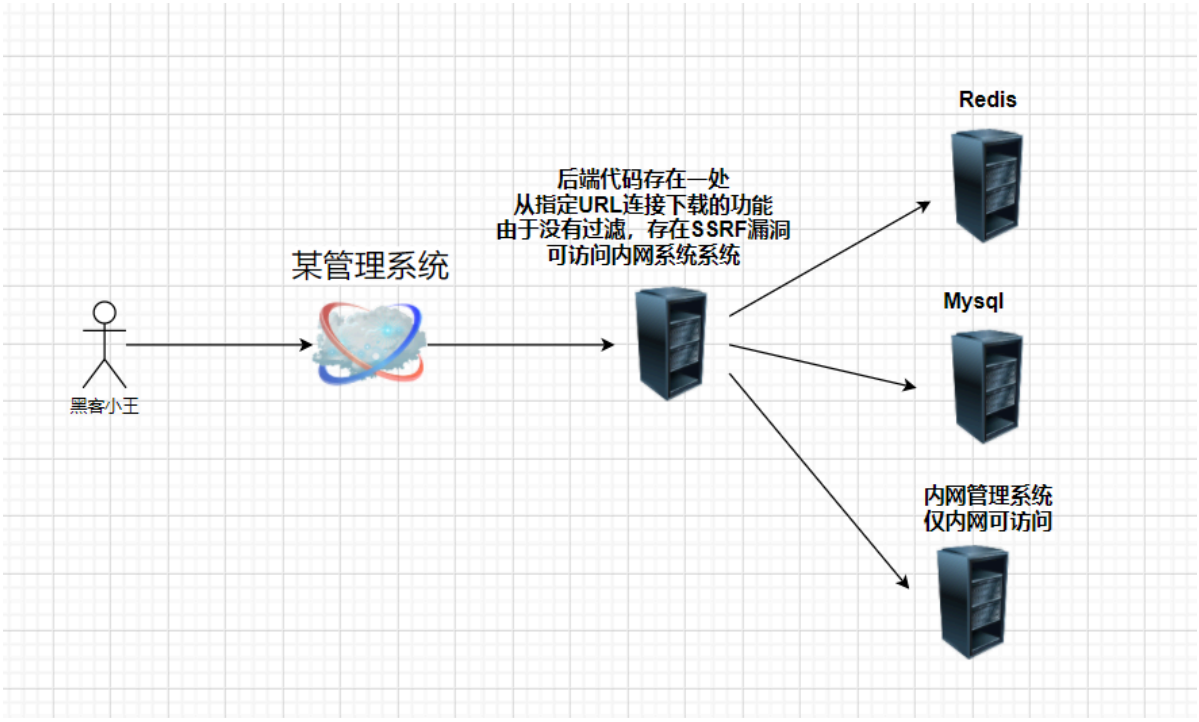
在典型的SSRF漏洞攻击中，攻击者可通过该漏洞攻击目标服务器的内网系统，比如：探测内网服务（通过响应不同判断），扫描开放端口（通过响应不同判断），使用File协议读取本地文件等操作。

在Java代码审计中SSRF漏洞成因大多是因为后端代码中存在向其他服务器请求的功能，但没有做好过滤。

黑客小王发现某管理系统存在一个可从外部服务器下载资源的功能，经代码审计发现后端使用了HTTP请求的相关代码，并且代码中没有对目标地址进行过滤，导致存在SSRF漏洞。黑客小王将请求参数`url`改为了内网敏感系统，最终发现可以访问，黑客小王按捺不住躁动的小手，进行了一番攻击操作，最终喜提一副银手镯。

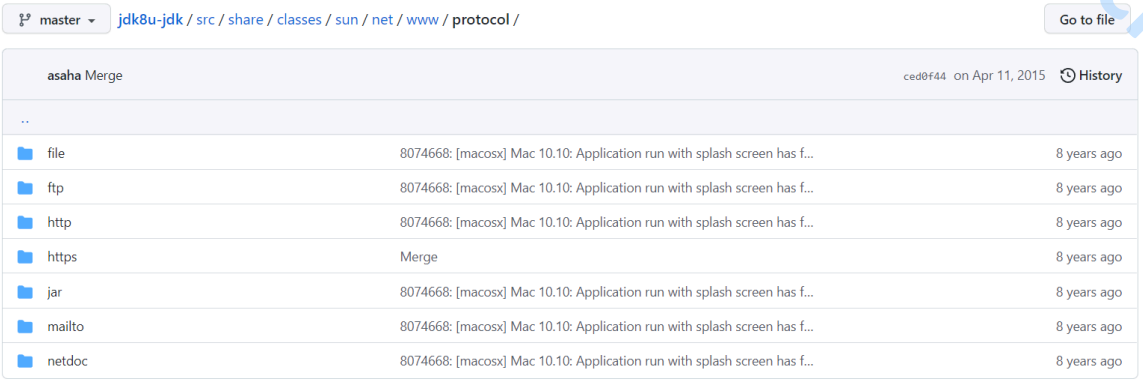
这个案例非常简化，为了初步理解SSRF漏洞。

也告诉我们发现SSRF漏洞，可配合使用DNSLog探测，非授权测试不要做过激行为。



2、Java 中支持的协议

Java网络请求支持的协议包括：http，https，file，ftp，mailto，jar，netdoc。如下图所示：



地址：`https://github.com/frohoff/jdk8u-jdk/tree/master/src/share/classes/sun/net/www/protocol`

> 1. **HTTP（Hypertext Transfer Protocol）**：HTTP协议是用于在客户端和服务器之间传输超文本的协议。SSRF漏洞可通过向内部系统发起HTTP请求，从而利用HTTP协议。攻击者可以通过指定特定的URL，向内部网络或本地主机发送HTTP请求，获取敏感信息或利用其他漏洞进行攻击。
> 2. **HTTPS（HTTP Secure）**：HTTPS是通过TLS/SSL加密协议对HTTP进行加密的版本，用于安全地传输数据。对于SSRF漏洞，HTTPS和HTTP的利用方式基本相同，但发送的请求会经过加密传输，提高了数据的保密性。
> 3. **File**: File协议用于从文件系统中获取文件。在SSRF攻击中，攻击者可以使用file协议来读取本地文件系统中的敏感文件，如/etc/passwd等，然后将文件内容发送到指定的外部服务器。
> 4. **FTP（File Transfer Protocol）**：FTP协议用于在网络上传输文件。通过SSRF漏洞，攻击者可以向内部网络的FTP服务器发起FTP请求，并执行文件传输操作，例如上传或下载文件。这可能导致泄露敏感文件或在内部网络中执行恶意文件。
> 5. **Mailto**: Mailto协议用于发送电子邮件。攻击者可以通过SSRF漏洞利用mailto协议，向内部网络中的电子邮件服务器发送电子邮件，可能用于发送恶意软件、钓鱼攻击等。
> 6. **Jar**: Jar协议用于从Java归档文件（JAR文件）中获取资源。通过SSRF漏洞，攻击者可以使用jar协议来读取JAR文件中的类文件，并执行其中的代码。这可能导致服务器端的远程代码执行漏洞。
> 7. **Netdoc**: Netdoc协议用于访问Javadoc文档。攻击者可以利用SSRF漏洞，通过netdoc协议获取内部网络中的Javadoc文档，并可能从中获取敏感信息或构造更多的攻击。

二、SSRF（服务端请求伪造）代码案例

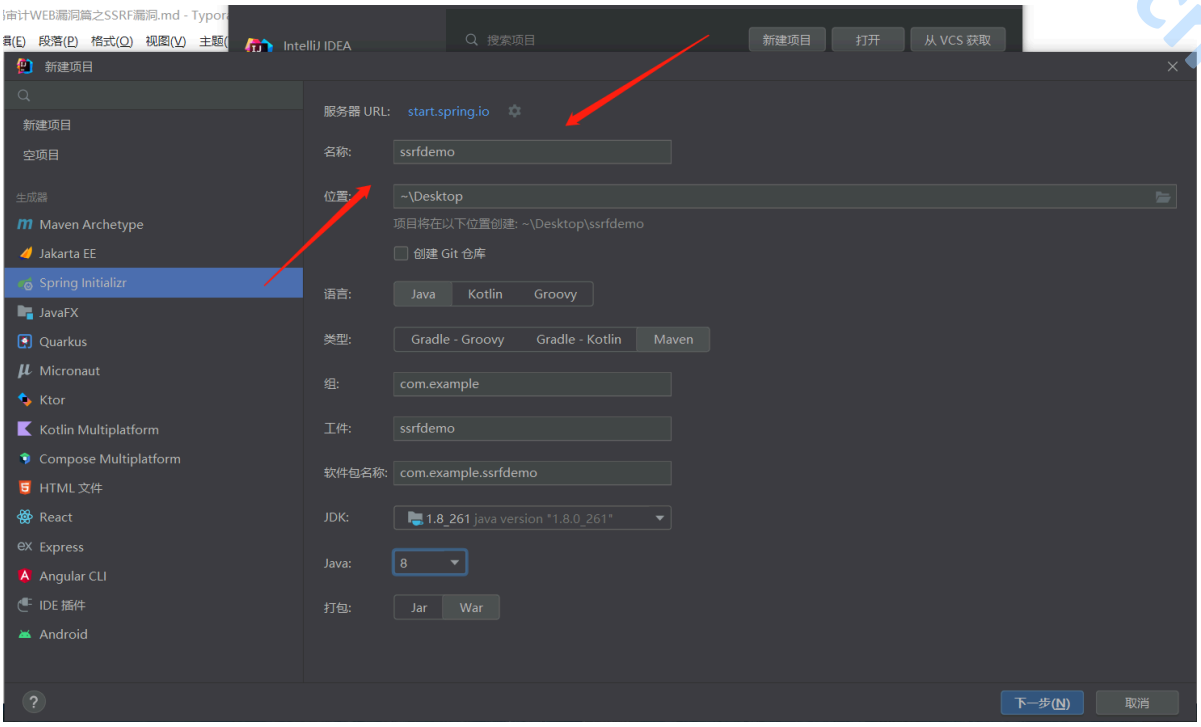
老规矩，我们新建一个名为`ssrfdemo`的项目工程，用于调试下面的代码。

由于我的IDEA版本换成了最新的`2022.3`了，因此创建项目与之前有点不同。最新破解教程可以访问这个帖子：`https://t.zsxq.com/089yMeGC6`。

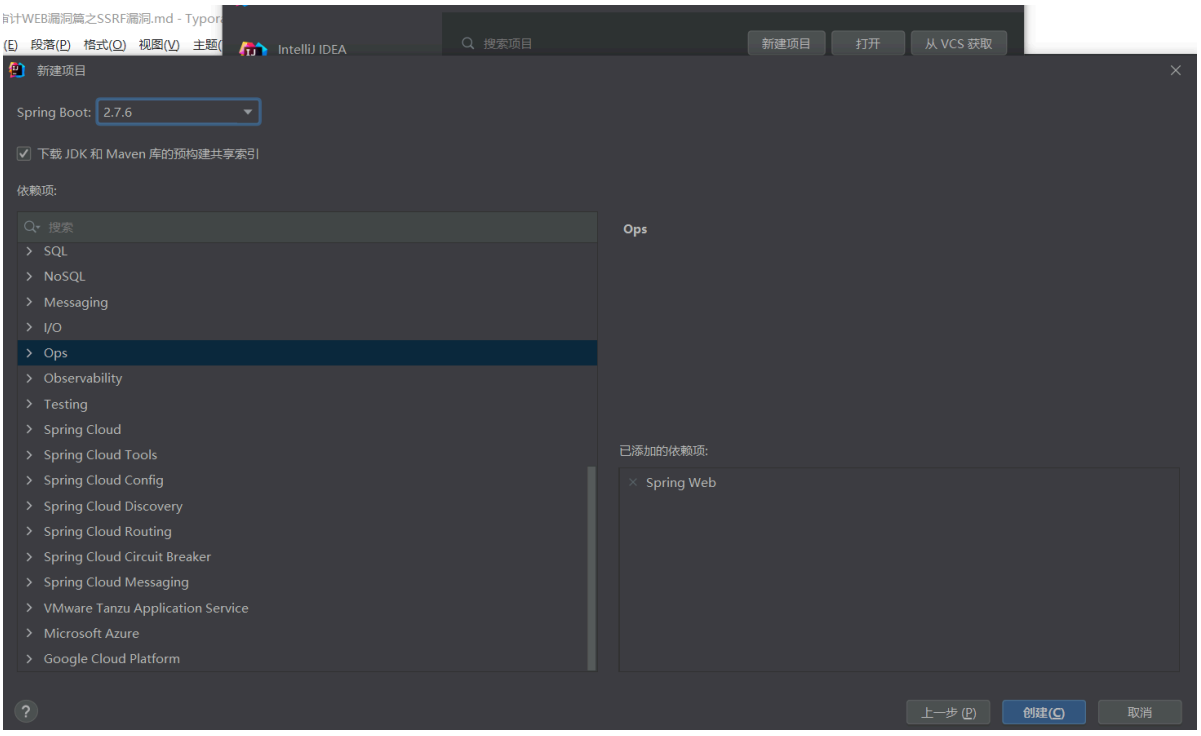
IDEA老版本的朋友可以参考往期创建项目工程的方法。往期我的IDEA版本是`IDEA2020.X`。

下面开始新建工程。

①、打开IDEA，选择新建项目。选择`Spring Initializr`，配置选项如下图所示：



②、点击下一步，依赖选择添加Web -> Spring Web，版本选择如下图所示：



最后点击创建即可。

1、HttpClient

1.1、介绍

HttpClient 是 Apache Jakarta Common 下的子项目，可以用来提供高效的、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。

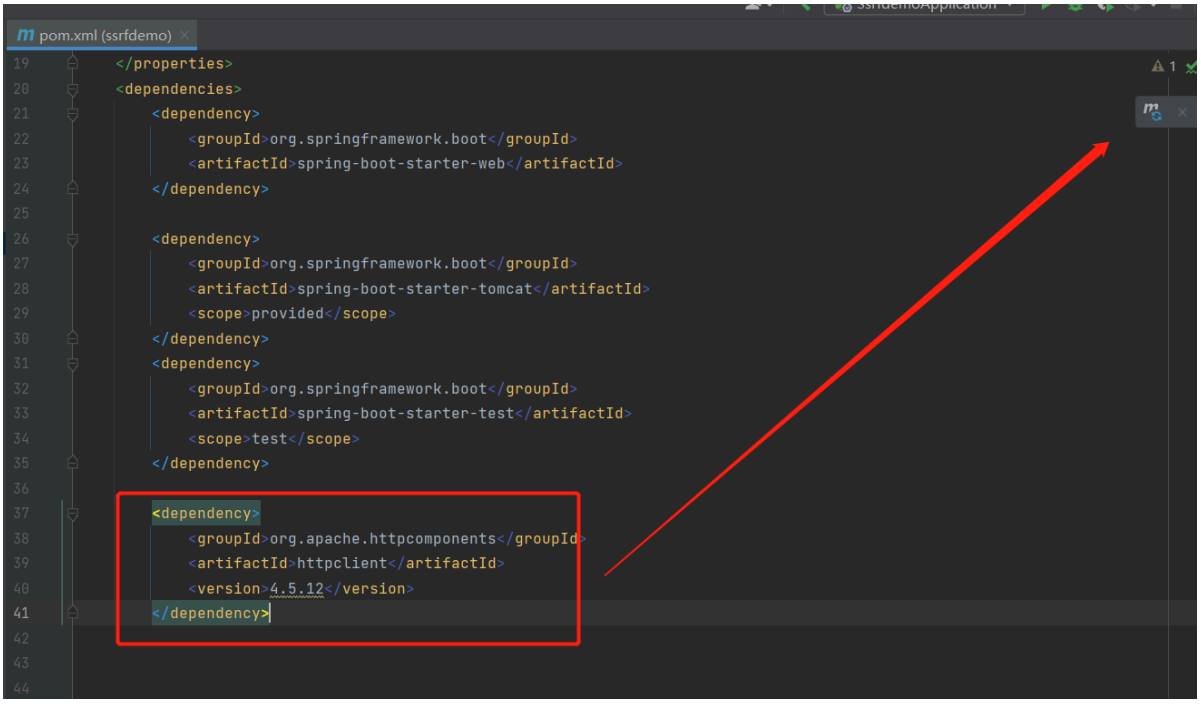
HttpClient 实现了 HTTP1.0 和 HTTP1.1。也实现了 HTTP 全部的方法，如：`GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE`。

官方介绍：`https://hc.apache.org/httpcomponents-client-5.2.x/index.html`

1.2、代码示例

首先，在`pom.xml`中引入 HttpClient 依赖，记得点击左上角重新加载Mavan变更，最终如下图所示：

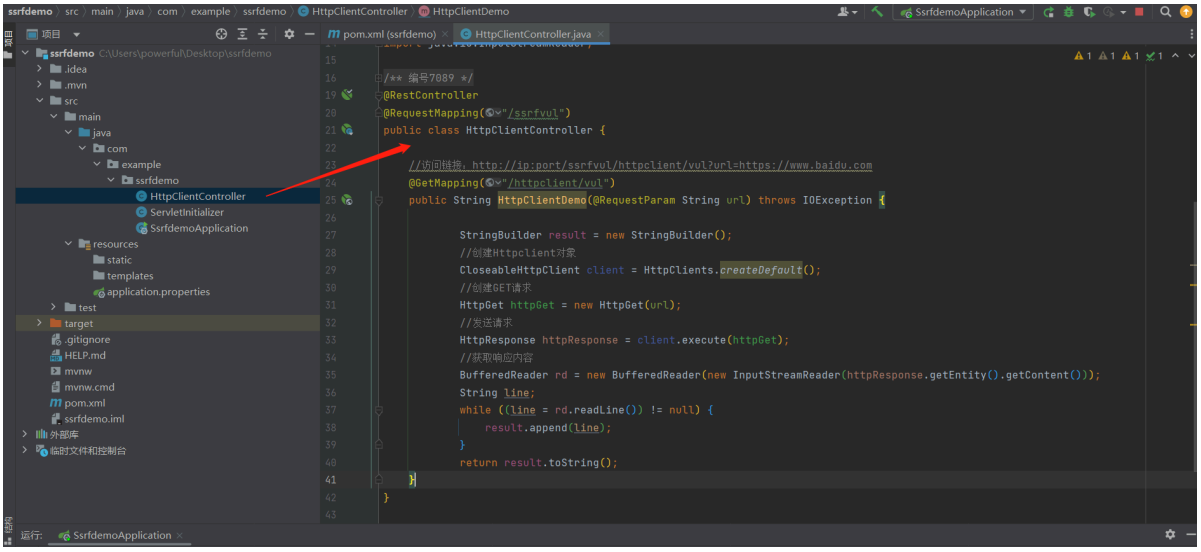
```
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.12</version>
</dependency>
```



然后在`src\main\java\com\example\ssrfdemo`下新建一个名为`HttpClientController`的 Java Class，并键入以下代码，最终如下图所示：

```
@RestController
@RequestMapping("/ssrfvul")
public class HttpClientController {

    //访问链接：http://ip:port/ssrfvul/httpclient/vul?url=https://www.baidu.com
    @GetMapping("/httpclient/vul")
    public String HttpClientDemo(@RequestParam String url) throws IOException {

            StringBuilder result = new StringBuilder();
            //创建 Httpclient 对象
            CloseableHttpClient client = HttpClients.createDefault();
            //创建 GET 请求
            HttpGet httpGet = new HttpGet(url);
            //发送请求
            HttpResponse httpResponse = client.execute(httpGet);
            //获取响应内容
            BufferedReader rd = new BufferedReader(new
InputStreamReader(httpResponse.getEntity().getContent()));
            String line;
            while ((line = rd.readLine()) != null) {
                result.append(line);
            }
```

```
            return result.toString();
    }

}
```



在该示例中，使用了`execute()`方法执行了 HTTP 请求。

启动项目，浏览器访问地址：`http://ip:port/ssrfvul/httpclient/vul?url=https://www.baidu.com`。

2、HttpAsyncClient

2.1、介绍

HttpAsyncClient 是一个异步的 HTTP 客户端开发包，基于 HttpCore NIO 和 HttpClient 组件。

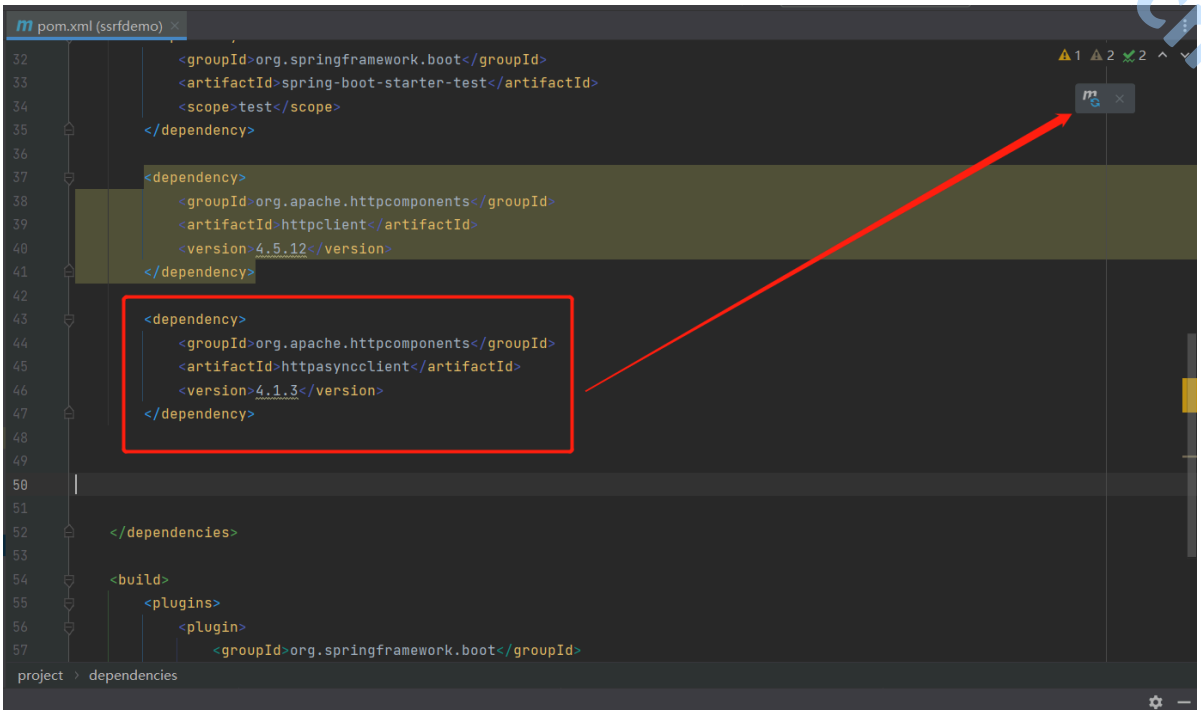
HttpAsyncClient 的出现并不是为了替换 HttpClient，而是作为一个补充用于需要大量并发连接，对性能要求非常高的基于 HTTP 的原生数据通信，而且提供了事件驱动的 API。

官方介绍：`https://hc.apache.org/httpcomponents-asyncclient-4.1.x/index.html`

2.2、代码示例

首先，在`pom.xml`中引入 HttpAsyncClient 依赖，最终如下图所示：

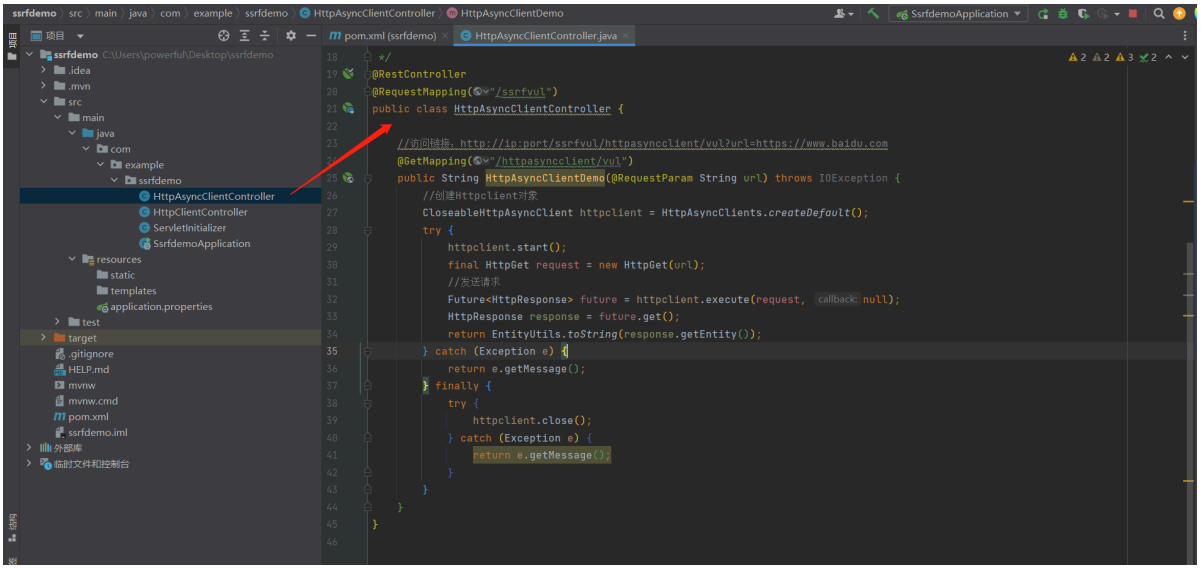
```
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpasyncclient</artifactId>
    <version>4.1.3</version>
</dependency>
```



然后在`src\main\java\com\example\ssrfdemo`下新建一个名为`HttpAsyncClientController`的Java Class，并键入以下代码，最终如下图所示：

```
package com.example.ssrfdemo;

import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.nio.client.CloseableHttpAsyncClient;
import org.apache.http.impl.nio.client.HttpAsyncClients;
import org.apache.http.util.EntityUtils;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import java.io.IOException;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;

/**
 * 编号7089
 */
@RestController
@RequestMapping("/ssrfvul")
public class HttpAsyncClientController {

    //访问链接：http://ip:port/ssrfvul/httpasyncclient/vul?
url=https://www.baidu.com
    @GetMapping("/httpasyncclient/vul")
    public String HttpAsyncClientDemo(@RequestParam String url) throws
IOException {
        //创建Httpclient对象
        CloseableHttpAsyncClient httpclient = HttpAsyncClients.createDefault();
        try {
            httpclient.start();
            final HttpGet request = new HttpGet(url);
            //发送请求
```

```
            Future<HttpResponse> future = httpclient.execute(request, null);
            HttpResponse response = future.get();
            return EntityUtils.toString(response.getEntity());
        } catch (Exception e) {
            return e.getMessage();
        } finally {
            try {
                httpclient.close();
            } catch (Exception e) {
                return e.getMessage();
            }
        }
    }
}
```



在该示例中，使用了`execute()`方法执行了HTTP请求。

启动项目，浏览器访问地址：`http://ip:port/ssrfvul/httpasyncclient/vul?url=https://www.baidu.com`。

3、java.net.URLConnection

3.1、介绍

java.net.URLConnection，是 Java 原生的 HTTP 请求方法。URLConnection 类包含了许多方法可以让你的 URL 在网络上通信。此类的实例既可用于读取URL所引用的资源，也可用于写入URL所引用资源。

官方介绍：`https://docs.oracle.com/javase/8/docs/api/java/net/URLConnection.html`

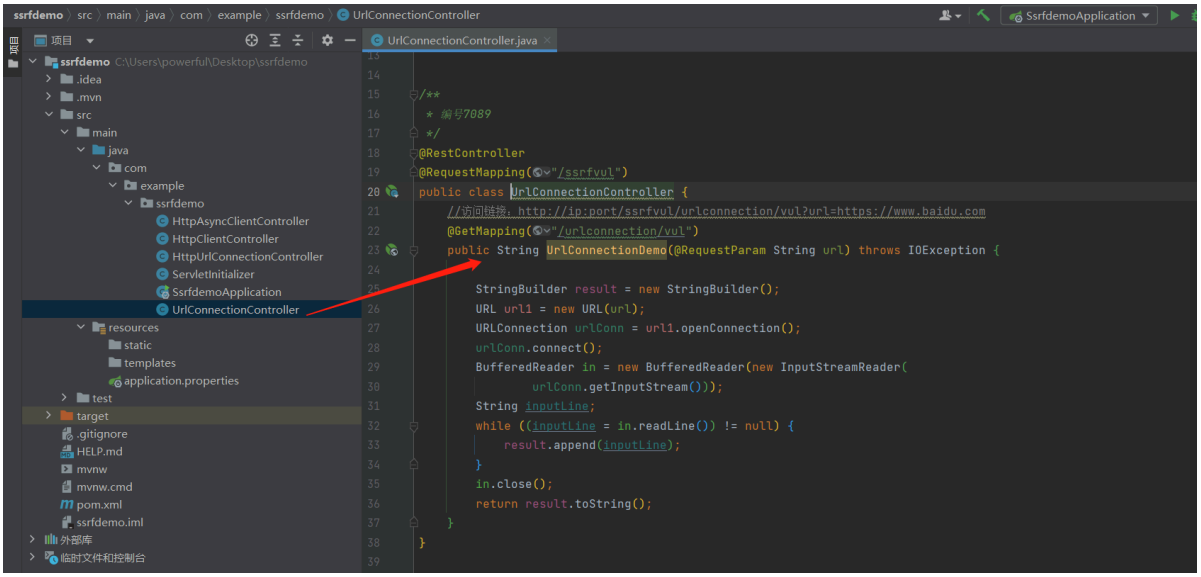
3.2、代码示例

java.net.URLConnection不需要额外引入依赖，已封装在JDK中。

然后在`src\main\java\com\example\ssrfdemo`下新建一个名为`UrlConnectionController`的 Java Class，并键入以下代码，最终如下图所示：

```
package com.example.ssrfdemo;
```

```java
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;


@RestController
@RequestMapping("/ssrfvul")
public class UrlConnectionController {
    //访问链接：http://ip:port/ssrfvul/urlconnection/vul?url=https://www.baidu.com
    @GetMapping("/urlconnection/vul")
    public String UrlConnectionDemo(@RequestParam String url) throws IOException
{

        StringBuilder result = new StringBuilder();
        URL url1 = new URL(url);
        URLConnection urlConn = url1.openConnection();
        urlConn.connect();
        BufferedReader in = new BufferedReader(new InputStreamReader(
                urlConn.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            result.append(inputLine);
        }
        in.close();
        return result.toString();
    }
}
```



在该示例中，使用了`openConnection()`方法执行了HTTP请求。

启动项目，浏览器访问地址：`http://ip:port/ssrfvul/urlconnection/vul?url=https://www.baidu.com`。

4、java.net.HttpURLConnection

4.1、介绍

HttpURLConnection 继承自 URLConnection。可以向指定网站发起GET或POST请求。

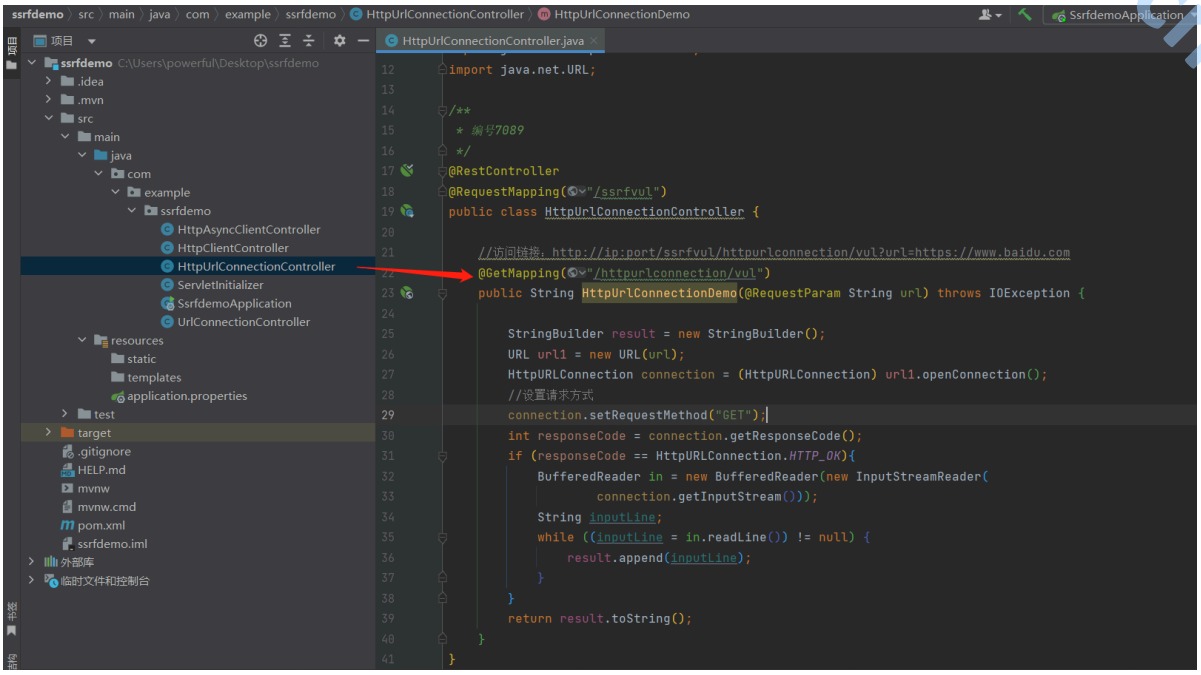
官方介绍：`https://docs.oracle.com/javase/8/docs/api/java/net/HttpURLConnection.html`

4.2、代码示例

java.net.HttpURLConnection不需要额外引入依赖，已内嵌在JDK中。

然后在`src\main\java\com\example\ssrfdemo`下新建一个名为`HttpUrlConnectionController`的Java Class，并键入以下代码，最终如下图所示：

```
package com.example.ssrfdemo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

@RestController
@RequestMapping("/ssrfvul")
public class HttpUrlConnectionController {

    //访问链接：http://ip:port/ssrfvul/httpurlconnection/vul?
url=https://www.baidu.com
    @GetMapping("/httpurlconnection/vul")
    public String HttpUrlConnectionDemo(@RequestParam String url) throws
IOException {

        StringBuilder result = new StringBuilder();
        URL url1 = new URL(url);
        HttpURLConnection connection = (HttpURLConnection) url1.openConnection();
        //设置请求方式
        connection.setRequestMethod("GET");
        int responseCode = connection.getResponseCode();
        if (responseCode == HttpURLConnection.HTTP_OK){
            BufferedReader in = new BufferedReader(new InputStreamReader(
                    connection.getInputStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                result.append(inputLine);
            }
        }
        return result.toString();
    }
}
```



在该示例中，使用了`openConnection()`方法执行了HTTP请求。

启动项目，浏览器访问地址：`http://ip:port/ssrfvul/httpurlconnection/vul?url=https://www.baidu.com`。

5、java.net.URL

5.1、介绍

在 java.net 包中定义了 URL 类，该类用来处理有关 URL 的内容。通过使用 URL 对象的`openStream()`方法创建打开指定 URL 链接，以获取输入流资源内容。

官方介绍：`https://docs.oracle.com/javase/8/docs/api/java/net/URL.html`

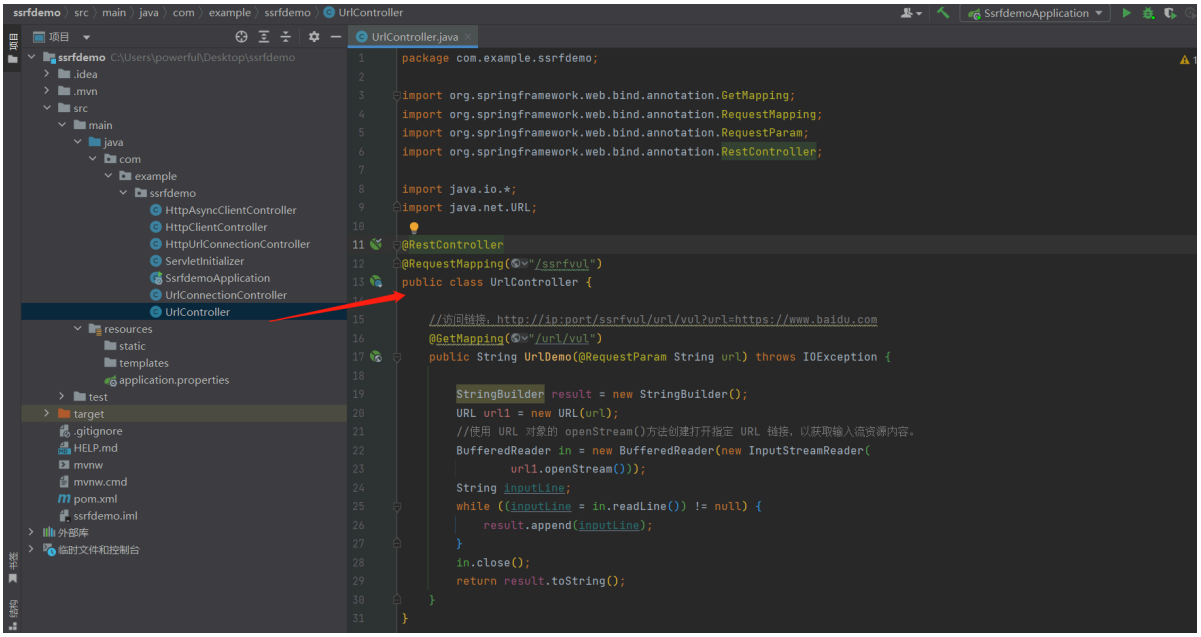
5.2、代码示例

java.net.URL 不需要额外引入依赖，已内嵌在JDK中。

然后在`src\main\java\com\example\ssrfdemo`下新建一个名为`UrlController`的 Java Class，并键入以下代码，最终如下图所示：

```
package com.example.ssrfdemo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.io.*;
import java.net.URL;


@RestController
@RequestMapping("/ssrfvul")
public class UrlController {
```

```
    //访问链接：http://ip:port/ssrfvul/url/vul?url=https://www.baidu.com
    @GetMapping("/url/vul")
    public String UrlDemo(@RequestParam String url) throws IOException {

        StringBuilder result = new StringBuilder();
        URL url1 = new URL(url);
        //使用 URL 对象的 openStream()方法创建打开指定 URL 链接，以获取输入流资源内容。
        BufferedReader in = new BufferedReader(new InputStreamReader(
                url1.openStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            result.append(inputLine);
        }
        in.close();
        return result.toString();
    }
}
```



在该示例中，使用了`openStream()`方法执行了HTTP请求。

启动项目，浏览器访问地址：`http://ip:port/ssrfvul/url/vul?url=https://www.baidu.com`。

6、java.net.Socket

6.1、介绍

java.net.Socket 是 Java 套接字编程使用的类。提供了两台计算机之间的通信机制。在Java代码审计中，我们可能会遇见使用Socket判断IP与端口连通性的代码。如果IP和端口接受外部输入，那么极有可能存在SSRF漏洞。

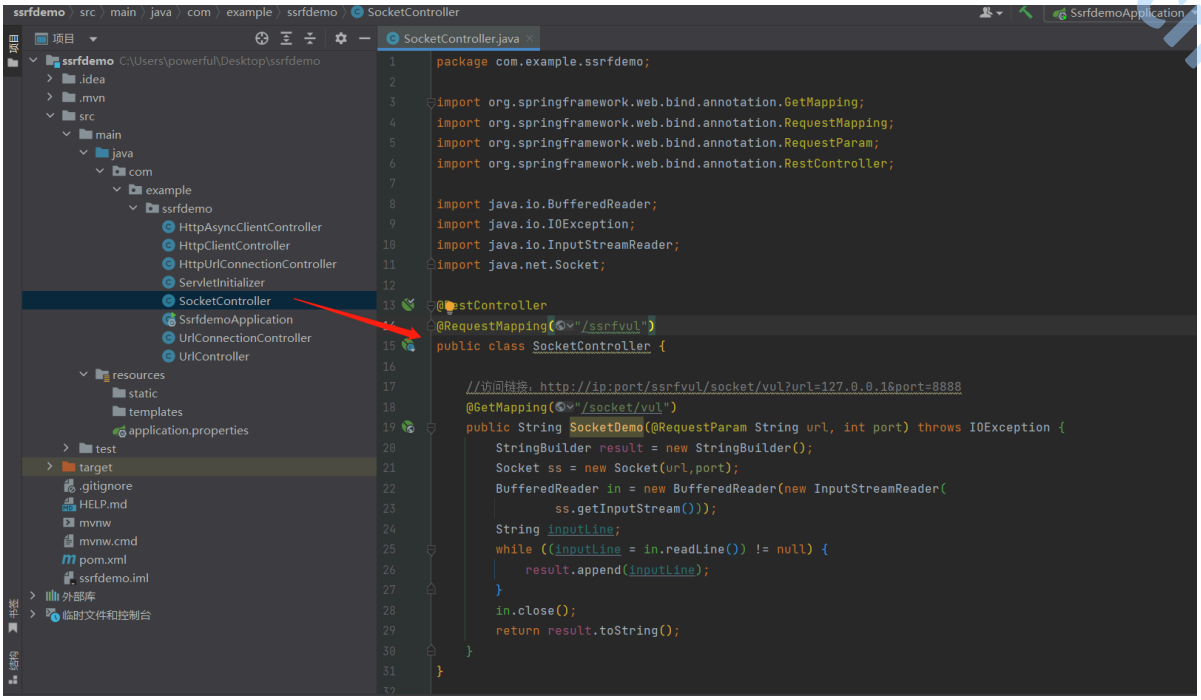
官方介绍：`https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html`

6.2、代码示例

java.net.Socket 不需要额外引入依赖，已内嵌在JDK中。

然后在`src\main\java\com\example\ssrfdemo`下新建一个名为`SocketController`的 Java Class，并键入以下代码，最终如下图所示：

```
package com.example.ssrfdemo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;

@RestController
@RequestMapping("/ssrfvul")
public class SocketController {

    //访问链接：http://ip:port/ssrfvul/socket/vul?url=127.0.0.1&port=8888
    @GetMapping("/socket/vul")
    public String SocketDemo(@RequestParam String url, int port) throws
IOException {

        StringBuilder result = new StringBuilder();
        Socket ss = new Socket(url,port);
        BufferedReader in = new BufferedReader(new InputStreamReader(
                ss.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            result.append(inputLine);
        }
        in.close();
        return result.toString();
    }
}
```



在该示例中，在实例化Sokcet时执行了网络请求。

启动项目，浏览器分别访问地址：`http://ip:port/ssrfvul/url/vul?url=127.0.0.1&port=8088`和`http://ip:port/ssrfvul/url/vul?url=127.0.0.1&port=9999`。其中8088端口是本项目启动的端口。

7、OkHttp

7.1、介绍

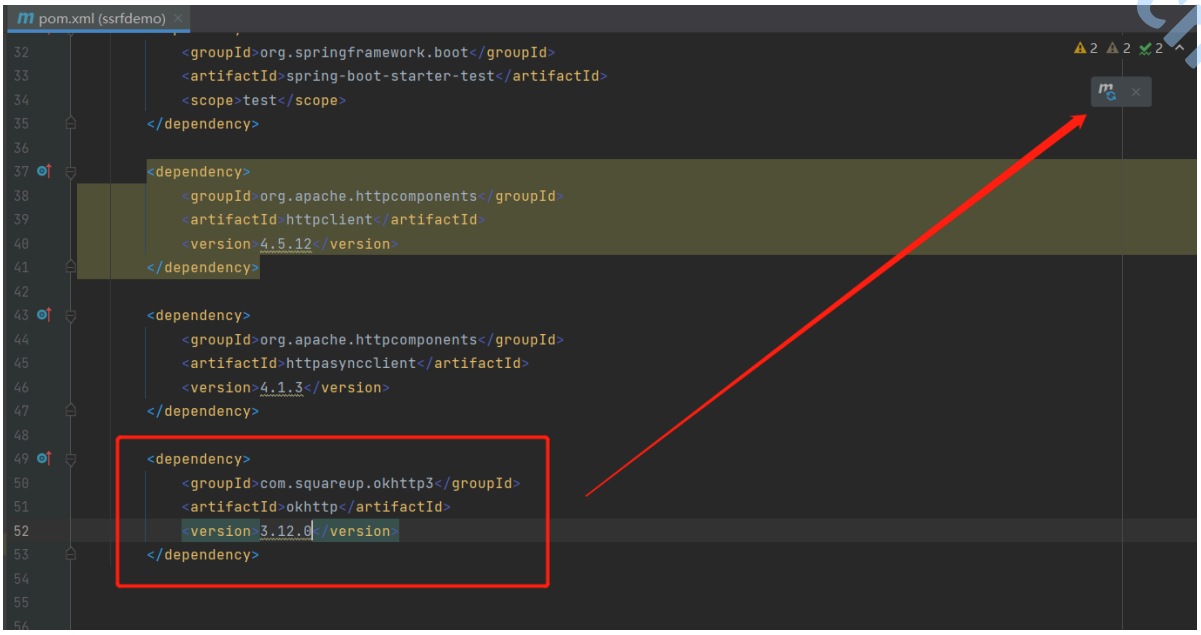
OKHttp 是一个网络请求框架，OKHttp会为每个客户端创建自己的连接池和线程池。重用连接和线程可以减少延迟并节省内存。OkHttp中请求方式分为同步请求（`client.newCall(request).execute()`）和异步请求（`client.newCall(request).enqueue()`）两种。

官方介绍：`https://square.github.io/okhttp/4.x/okhttp/okhttp3/-ok-http-client/`

7.2、代码示例

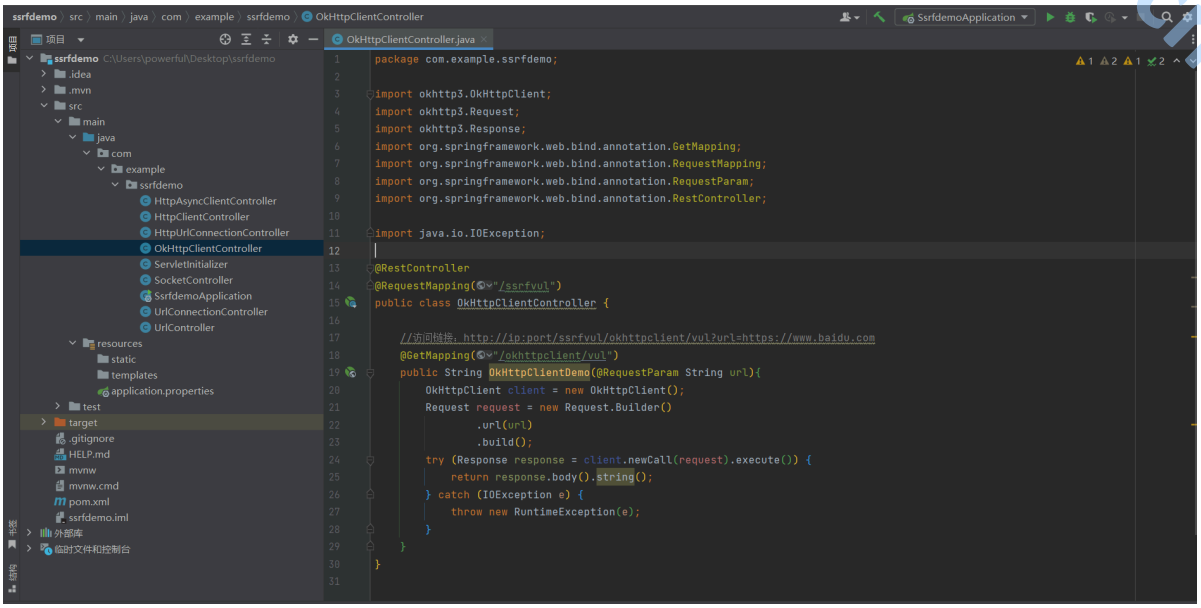
首先，在`pom.xml`中引入 OkHttpClient 依赖，最终如下图所示：

```
<dependency>
    <groupId>com.squareup.okhttp3</groupId>
    <artifactId>okhttp</artifactId>
    <version>3.12.0</version>
</dependency>
```



然后在`src\main\java\com\example\ssrfdemo`下新建一个名为`OkHttpClientController`的 Java Class，并键入以下代码，最终如下图所示：

```
package com.example.ssrfdemo;

import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.io.IOException;

@RestController
@RequestMapping("/ssrfvul")
public class OkHttpClientController {

    //访问链接：http://ip:port/ssrfvul/okhttpclient/vul?url=https://www.baidu.com
    @GetMapping("/okhttpclient/vul")
    public String OkHttpClientDemo(@RequestParam String url){
        OkHttpClient client = new OkHttpClient();
        Request request = new Request.Builder()
                .url(url)
                .build();
        try (Response response = client.newCall(request).execute()) {
            return response.body().string();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }


    }
}
```



如果运行报错：`java: 程序包okhttp3不存在`，需要访问`Settings-->Build-->Build Tools-->Maven-->Runner-->勾选上Delegagte IDE build/run actions to Maven`。

在该示例中，使用了`.newCall(request).execute()`方法执行了HTTP请求。

启动项目，访问`http://127.0.0.1:8088/ssrfvul/okhttpclient/vul?url=https://www.baidu.com`。

8、ImageIO

8.1、介绍

ImageIO 是Java读写图片操作的一个类。在代码审计中，如果目标使用了`ImageIO.read`读取图片，且读取的图片地址可控的话，可能会存在SSRF漏洞。

官方介绍：`https://docs.oracle.com/javase/8/docs/api/javax/imageio/ImageIO.html`

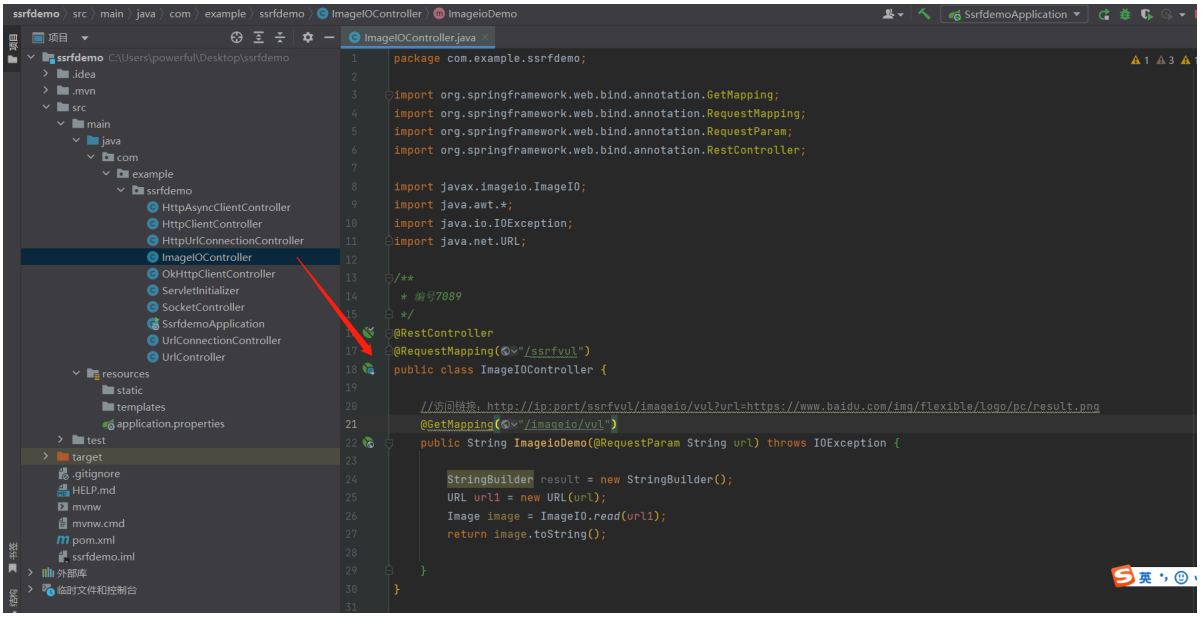
8.2、代码示例

javax.imageio.ImageIO 不需要额外引入依赖，已封装在JDK中。

然后在`src\main\java\com\example\ssrfdemo`下新建一个名为`ImageIOController`的 Java Class，并键入以下代码，最终如下图所示：

```
package com.example.ssrfdemo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import javax.imageio.ImageIO;
import java.awt.*;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;

@RestController
```

```java
@RequestMapping("/ssrfvul")
public class ImageIOController {

    //访问链接：http://ip:port/ssrfvul/imageio/vul?
url=https://www.baidu.com/img/flexible/logo/pc/result.png
    @GetMapping("/imageio/vul")
    public String ImageioDemo(@RequestParam String url) throws IOException {

        StringBuilder result = new StringBuilder();
        URL url1 = new URL(url);
        Image image = ImageIO.read(url1);
        return image.toString();

    }
}
```



在该示例中，使用了`ImageIO.read()`方法执行了HTTP请求。

启动项目，访问`http://ip:port/ssrfvul/imageio/vul?url=https://www.baidu.com/img/flexible/logo/pc/result.png`。

9、Hutool

9.1、介绍

Hutool是一个小而全的Java工具类库，通过静态方法封装，降低相关API的学习成本，提高工作效率，使Java拥有函数式语言般的优雅。

在Hutool中，也实现了HTTP客户端，Hutool-http针对JDK的HttpUrlConnection做一层封装，简化了HTTPS请求、文件上传、Cookie记忆等操作，使Http请求变得无比简单。

Hutool-http的核心集中在两个类：

- HttpRequest
- HttpResponse

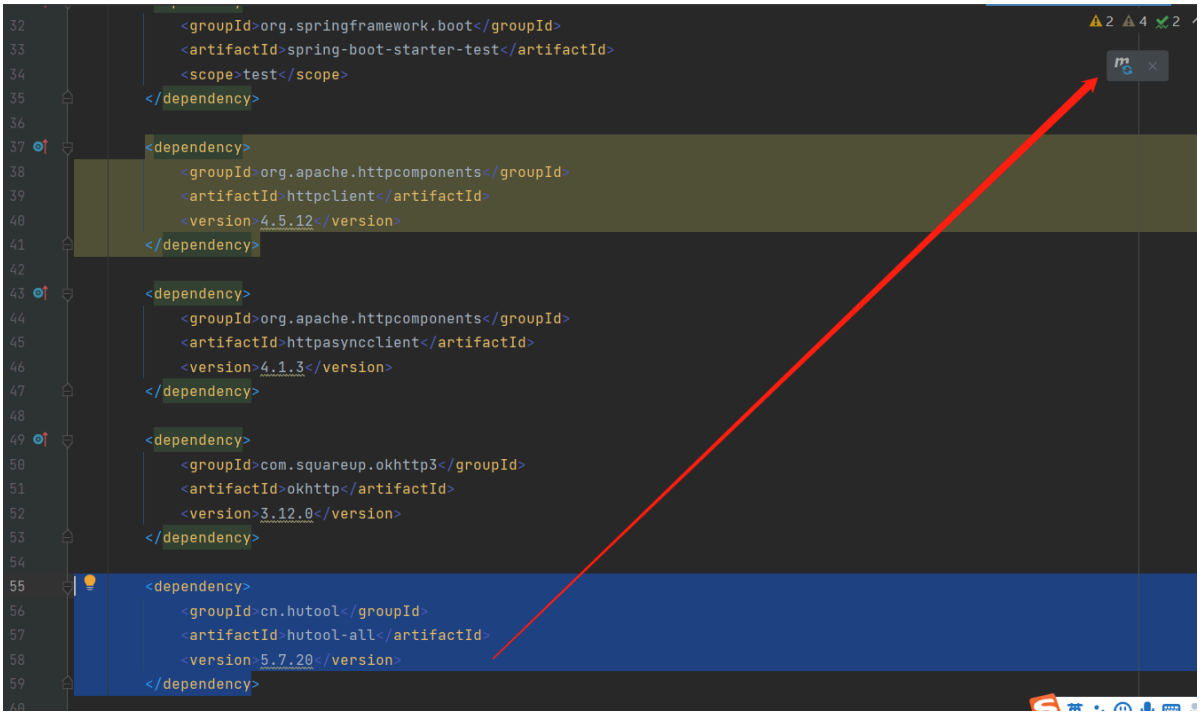
官方网站：`https://www.hutool.cn/`

Http客户端官方介绍：`https://hutool.cn/docs/#/http/%E6%A6%82%E8%BF%B0`

9.2、代码示例

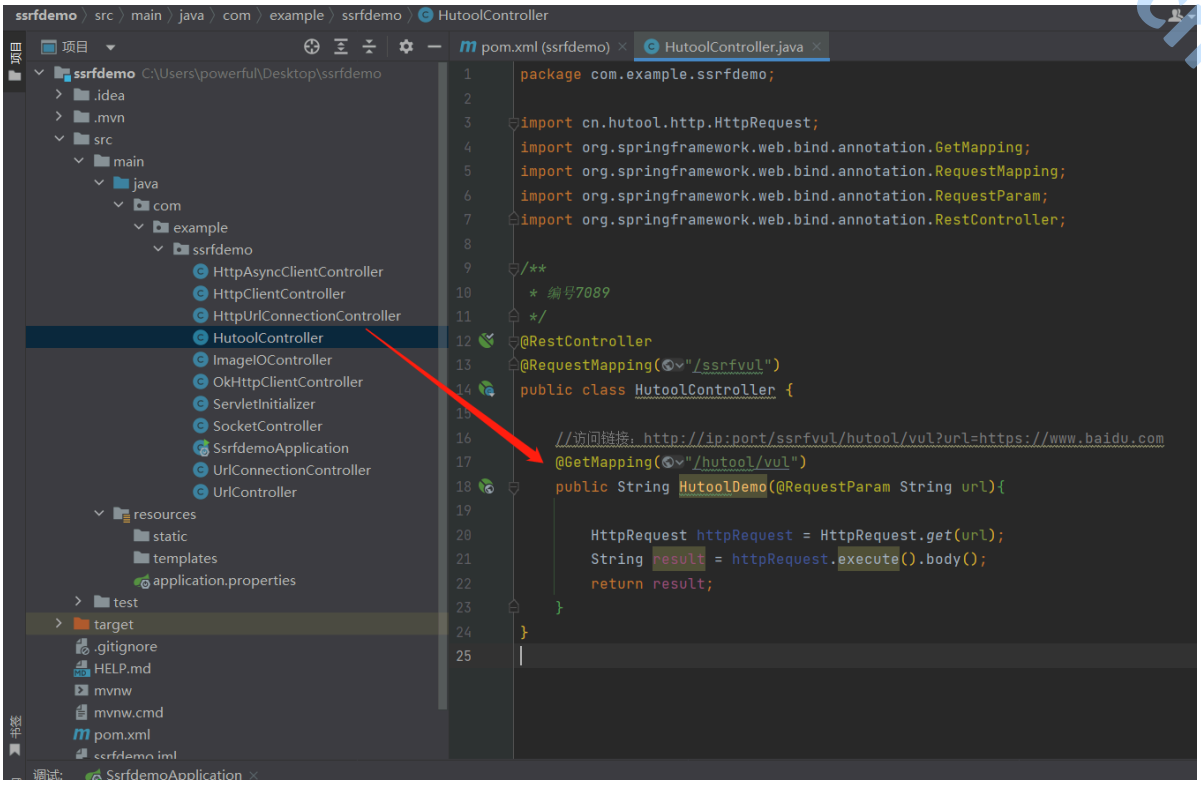
首先，在`pom.xml`中引入 Hutool依赖，最终如下图所示：

```
        <dependency>
            <groupId>cn.hutool</groupId>
            <artifactId>hutool-all</artifactId>
            <version>5.7.20</version>
        </dependency
```



然后在`src\main\java\com\example\ssrfdemo`下新建一个名为`HutoolController`的 Java Class，并键入以下代码，最终如下图所示：

```
package com.example.ssrfdemo;

import cn.hutool.http.HttpRequest;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/ssrfvul")
public class HutoolController {

    //访问链接：http://ip:port/ssrfvul/hutool/vul?url=https://www.baidu.com
    @GetMapping("/hutool/vul")
    public String HutoolDemo(@RequestParam String url){

        HttpRequest httpRequest = HttpRequest.get(url);
        String result = httpRequest.execute().body();
        return result;
    }
}
```



在该示例中，使用了`execute()`方法执行了HTTP请求。

启动项目，访问：`http://ip:port/ssrfvul/hutool/vul?url=https://www.baidu.com`。

10、Jsoup

10.1、介绍

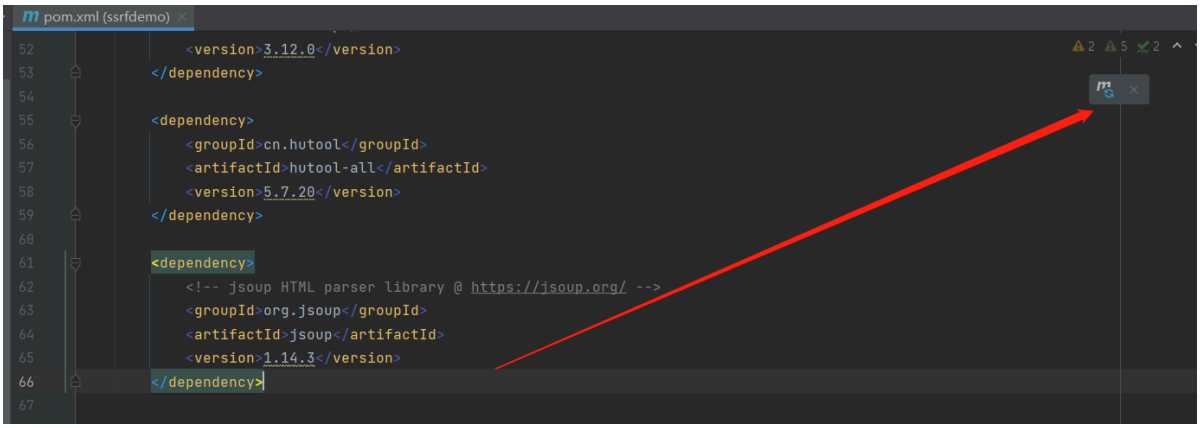
Jsoup 是基于 Java 的 HTML 解析器，可以从指定的 URL 中解析 HTML 内容。

官方介绍：`https://jsoup.org/`

10.2、代码示例

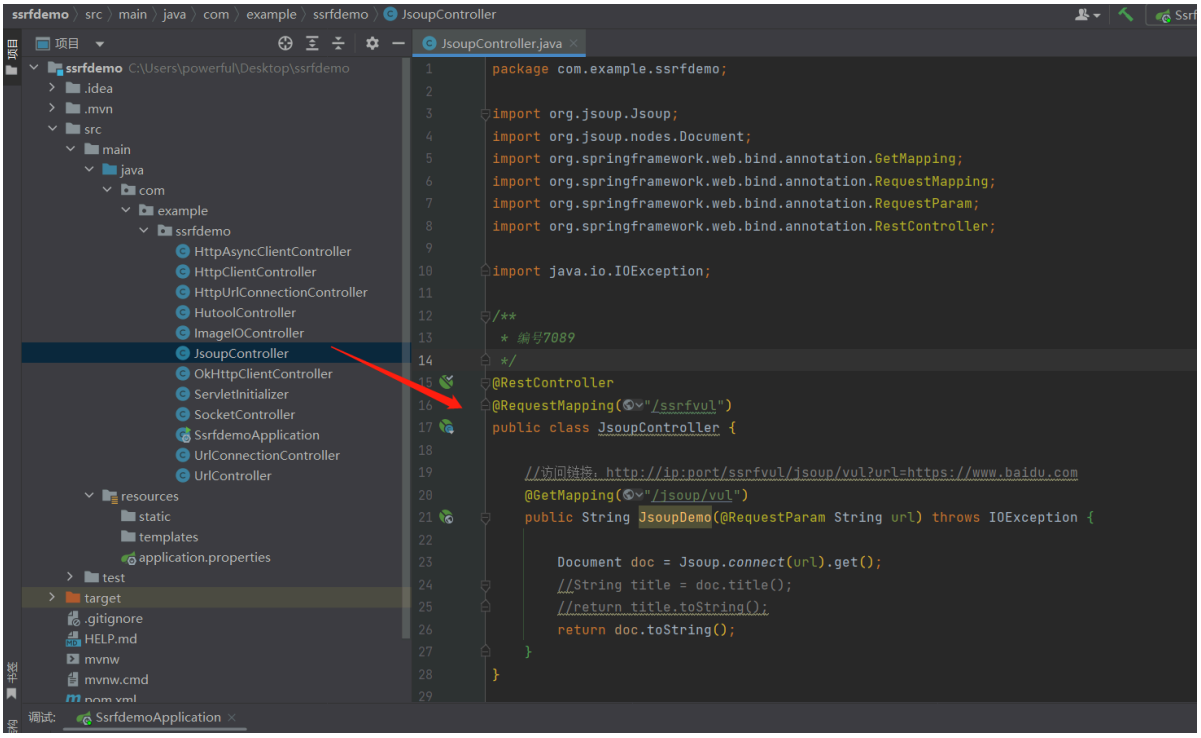
首先，在`pom.xml`中引入 jsoup 依赖，最终如下图所示：

```
<dependency>
  <!-- jsoup HTML parser library @ https://jsoup.org/ -->
  <groupId>org.jsoup</groupId>
  <artifactId>jsoup</artifactId>
  <version>1.15.3</version>
</dependency>
```



然后在`src\main\java\com\example\ssrfdemo`下新建一个名为`JsoupController`的Java Class，并键入以下代码，最终如下图所示：

```
package com.example.ssrfdemo;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.io.IOException;

@RestController
@RequestMapping("/ssrfvul")
public class JsoupController {

    //访问链接：http://ip:port/ssrfvul/jsoup/vul?url=https://www.baidu.com
    @GetMapping("/jsoup/vul")
    public String JsoupDemo(@RequestParam String url) throws IOException {

        Document doc = Jsoup.connect(url).get();
        //String title = doc.title();
        //return title.toString();
        return doc.toString();
    }
}
```



在该示例中，使用了`Jsoup.connect()`方法执行了HTTP请求。

启动项目，访问：`http://ip:port/ssrfvul/jsoup/vul?url=https://www.baidu.com`。

11、RestTemplate

11.1、介绍

RestTemplate 是从Spring3.0 开始支持的一个HTTP 请求工具，它提供了常见的REST请求方案的模版，例如GET 请求、POST 请求、PUT 请求等等。从名称上来看，是更针对RESTFUL风格API设计的。但通过他调用普通的HTTP接口也是可以的。

官方介绍：`https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html`

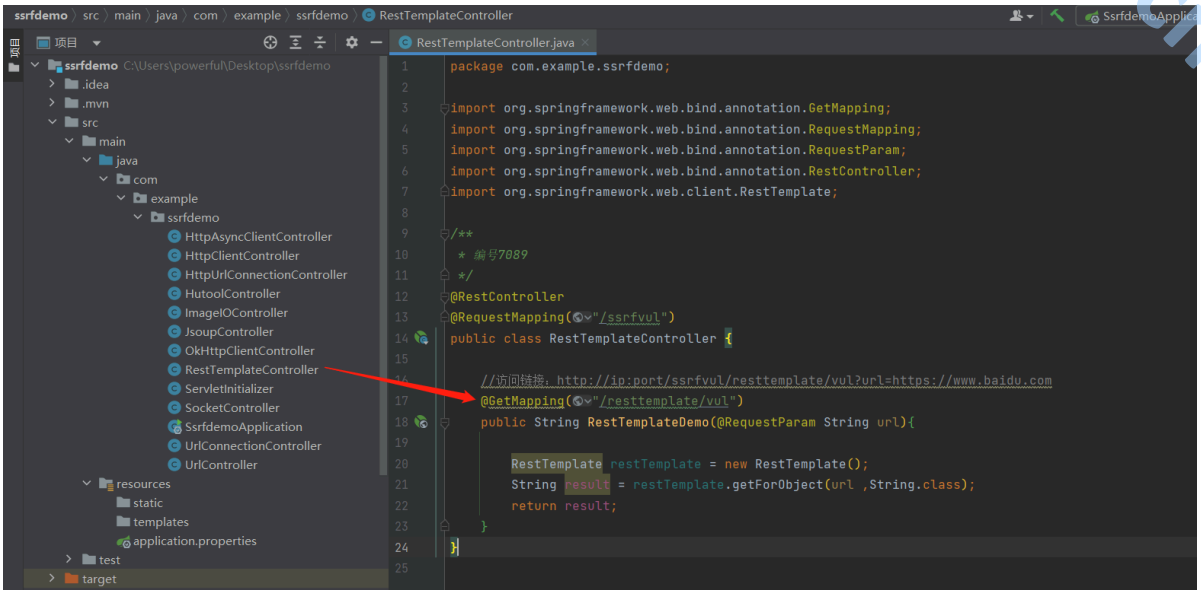
11.2、代码示例

首先，在`pom.xml`中引入 RestTemplate 依赖，RestTemplate 依赖其实就在spring-web这个包下面，引入该依赖即可，如下：

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
```

然后在`src\main\java\com\example\ssrfdemo`下新建一个名为`xxxController`的 Java Class，并键入以下代码，最终如下图所示：

```
package com.example.ssrfdemo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
@RequestMapping("/ssrfvul")
public class RestTemplateController {

    //访问链接：http://ip:port/ssrfvul/resttemplate/vul?url=https://www.baidu.com
    @GetMapping("/resttemplate/vul")
    public String RestTemplateDemo(@RequestParam String url){

        RestTemplate restTemplate = new RestTemplate();
        String result = restTemplate.getForObject(url ,String.class);
        return result;
    }
}
```



在该示例中，使用了`getForObject()`方法执行了HTTP请求。

启动项目，访问：`http://ip:port/ssrfvul/resttemplate/vul?url=https://www.baidu.com`。

以上代码仅是为了演示一些网络请求所使用的的工具，然大家先了解这些依赖和基础使用方法。

代码都写在了Controller层，没有复杂的逻辑。在实际项目中，大多都写在工具类中，方便调用。

另外上面给出的有些方法支持多个协议，比如说`urlConnection`，也支持`file`协议，那么我们是不是可以使用该漏洞配合`file`协议进行任意文件读取攻击呢?

总之在了解了这些网络请求工具后，可以更快更准确定位系统中是否存在HTTP请求，进而判断是否存在SSRF漏洞。

三、SSRF（服务端请求伪造）代码审计

1、SSRF代码审计关键字

下面关键字是基于上述常用依赖整理出来的，通过关键字可快速定位是否使用了该依赖以及相关HTTP请求方法，具体逻辑还需根据实际代码分析。

```
HttpRequest.get
HttpRequest.post
Jsoup.connect
getForObject
RestTemplate
postForObject
httpclient
execute
HttpClients.createDefault
httpasyncclient
HttpAsyncClients.createDefault
java.net.URLConnection
openConnection
```

```
java.net.HttpURLConnection
openStream
Socket
java.net.Socket
okhttp
OkHttpClient
newCall
ImageIO.read
javax.imageio.ImageIO
HttpRequest.get
jsoup
Jsoup.connect
RestTemplate
org.springframework.web.client.RestTemplate
```

2、实战系统分析案例一：

简介：系统 Jspxcms 版本 V9.0.0 存在未授权 SSRF 漏洞

全局搜索关键字，发现有几处使用了 openConnection，经过追踪分析发现 UploadControllerAbstract#ueditorCatchImage() 方法下存在漏洞点，当控制参数 source[]，即可实现资源请求操作，进而造成 SSRF 漏洞。

3、实战系统分析案例二：

简介：系统 Jspxcms 版本 V9.0.0 存在 SSRF 漏洞

全局搜索关键字，发现有几处使用了 HttpClient.execute，经过追踪分析发现 Collect#fetchHtml() 放下存在漏洞。追踪发现最终输入点位于 CollectController#fetchUrl 方法，接口为：/cmscp/ext/collect/fetch_url.do，当控制参数 url，即可实现资源请求操作，进而造成 SSRF 漏洞。

4、实战系统分析案例三：

简介：Rebuild 管理系统 3.5.5 版本以下存在 SSRF 漏洞。CVE 编号为 CVE-2024-1021。

http://changge.online/blog-50.html

通过上述文章，给出的 payload 我们可以定位到漏洞点，并进行分析，如下：

```
GET /filex/read-raw?url=http://onptdn.dnslog.cn&cut=1 HTTP/1.1
Host:
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/115.0.0.0 Safari/537.36
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Connection: close
```

全局搜索 `filex/read-raw`。

代码位于：com/rebuild/web/commons/FileDownloader.java#readRawText

漏洞点位于第 183 行，首先在第 182 行进行了一个判断，该判断跟踪进去发现为判断为是否是 http 或 https 开头，如果是 filepath 是以这两个开头，则进入 183 行，调用 OkHttpUtils.get 方法。

跟进该方法，发现为作者写的网络请求工具类，本质是使用的 OkHttp 类。

可以和前面讲的 OkHttp 示例代码进行对比分析。

其中 cut 参数不能为 null，因此要设置一个整型值。

5、实战系统分析案例四：

简介：Rebuild 管理系统某版本以下存在 SSRF 漏洞，具体可参考：https://github.com/getrebuild/rebuild/issues/460。

通过分析该漏洞点发现 3.5.4 这个版本已经修复。对请求的 url 值进行了判断，判断开头是否以 http:// 或这 https:// 开头。

如果是的话则报错。

目前没发现绕过的方式。

四、SSRF（服务端请求伪造）漏洞修复

1、安全代码参考来源：`https://github.com/j3ers3/Hello-Java-Sec`

省事不打开版。

```
// 判断是否是http类型
public static boolean isHttp(String url) {
    return url.startsWith("http://") || url.startsWith("https://");
}

// 判断是否为内网
public static boolean isIntranet(String url) {
    Pattern reg = Pattern.compile("^(127\\.0\\.0\\.1)|(localhost)|
(10\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3})|(172\\.((1[6-9])|(2\\d)|
(3[01]))\\.\\d{1,3}\\.\\d{1,3})|(192\\.168\\.\\d{1,3}\\.\\d{1,3})$");
    Matcher match = reg.matcher(url);
    Boolean a = match.find();
    return a;
}

// 不允许跳转或判断跳转
HttpURLConnection conn = (HttpURLConnection) u.openConnection();
conn.setInstanceFollowRedirects(false); // 不允许重定向或者对重定向后的地址做二次判断
conn.connect();
```

2、安全代码参考来源：`https://github.com/JoyChou93/java-sec-code/blob/master/src/main/java/org/joychou/security/SecurityUtil.java`

省事不打开版。

```
package org.joychou.security;
```

```
import org.joychou.config.WebConfig;
import org.joychou.security.ssrf.SSRFChecker;
import org.joychou.security.ssrf.SocketHook;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.URLDecoder;
import java.util.ArrayList;
import java.util.regex.Pattern;


public class SecurityUtil {

    private static final Pattern FILTER_PATTERN = Pattern.compile("^[a-zA-Z0-
9_/\\.-]+$");
    private static Logger logger = LoggerFactory.getLogger(SecurityUtil.class);


    /**
     * Determine if the URL starts with HTTP.
     *
     * @param url url
     * @return true or false
     */
    public static boolean isHttp(String url) {
        return url.startsWith("http://") || url.startsWith("https://");
    }


    /**
     * Get http url host.
     *
     * @param url url
     * @return host
     */
    public static String gethost(String url) {
        try {
            URI uri = new URI(url);
            return uri.getHost().toLowerCase();
        } catch (URISyntaxException e) {
            return "";
        }
    }


    /**
     * 同时支持一级域名和多级域名，相关配置在resources目录下url/url_safe_domain.xml文件。
     * 优先判断黑名单，如果满足黑名单return null。
     *
     * @param url the url need to check
     * @return Safe url returns original url; Illegal url returns null;
```

```
     */
    public static String checkURL(String url) {

        if (null == url){
            return null;
        }

        ArrayList<String> safeDomains = WebConfig.getSafeDomains();
        ArrayList<String> blockDomains = WebConfig.getBlockDomains();

        try {
            String host = gethost(url);

            // 必须http/https
            if (!isHttp(url)) {
                return null;
            }

            // 如果满足黑名单返回null
            if (blockDomains.contains(host)){
                return null;
            }
            for(String blockDomain: blockDomains) {
                if(host.endsWith("." + blockDomain)) {
                    return null;
                }
            }

            // 支持多级域名
            if (safeDomains.contains(host)){
                return url;
            }

            // 支持一级域名
            for(String safedomain: safeDomains) {
                if(host.endsWith("." + safedomain)) {
                    return url;
                }
            }
            return null;
        } catch (NullPointerException e) {
            logger.error(e.toString());
            return null;
        }
    }


    /**
     * 通过自定义白名单域名处理SSRF漏洞。如果URL范围收敛，强烈建议使用该方案。
     * 这是最简单也最有效的修复方式。因为SSRF都是发起URL请求时造成，大多数场景是图片场景，一般图片的域名都是CDN或者OSS等，所以限定域名白名单即可完成SSRF漏洞修复。
     *
     * @author JoyChou @ 2020-03-30
     * @param url 需要校验的url
     * @return Safe url returns true. Dangerous url returns false.
     */
```

```
    public static boolean checkSSRFByWhitehosts(String url) {
        return SSRFChecker.checkURLFckSSRF(url);
    }


    /**
     * 解析URL的IP，判断IP是否是内网IP。如果有重定向跳转，循环解析重定向跳转的IP。不建议使用该方案。
     *
     * 存在的问题：
     *   1、会主动发起请求，可能会有性能问题
     *   2、设置重定向跳转为第一次302不跳转，第二次302跳转到内网IP 即可绕过该防御方案
     *   3、TTL设置为0会被绕过
     *
     * @param url check的url
     * @return 安全返回true，危险返回false
     */
    @Deprecated
    public static boolean checkSSRF(String url) {
        int checkTimes = 10;
        return SSRFChecker.checkSSRF(url, checkTimes);
    }


    /**
     * 不能使用白名单的情况下建议使用该方案。前提是禁用重定向并且TTL默认不为0。
     *
     * 存在问题：
     *  1、TTL为0会被绕过
     *  2、使用重定向可绕过
     *
     * @param url The url that needs to check.
     * @return Safe url returns true. Dangerous url returns false.
     */
    public static boolean checkSSRFWithoutRedirect(String url) {
        if(url == null) {
            return false;
        }
        return !SSRFChecker.isInternalIpByUrl(url);
    }

    /**
     * Check ssrf by hook socket. Start socket hook.
     *
     * @author liergou @ 2020-04-04 02:15
     */
    public static void startSSRFHook() throws IOException {
        SocketHook.startHook();
    }

    /**
     * Close socket hook.
     *
     * @author liergou @ 2020-04-04 02:15
     **/
    public static void stopSSRFHook(){
```

```
        SocketHook.stopHook();
    }



    /**
     * Filter file path to prevent path traversal vulns.
     *
     * @param filepath file path
     * @return illegal file path return null
     */
    public static String pathFilter(String filepath) {
        String temp = filepath;

        // use while to sovle multi urlencode
        while (temp.indexOf('%') != -1) {
            try {
                temp = URLDecoder.decode(temp, "utf-8");
            } catch (UnsupportedEncodingException e) {
                logger.info("Unsupported encoding exception: " + filepath);
                return null;
            } catch (Exception e) {
                logger.info(e.toString());
                return null;
            }
        }

        if (temp.contains("..") || temp.charAt(0) == '/') {
            return null;
        }

        return filepath;
    }


    public static String cmdFilter(String input) {
        if (!FILTER_PATTERN.matcher(input).matches()) {
            return null;
        }

        return input;
    }


    /**
     * 过滤mybatis中order by不能用#的情况。
     * 严格限制用户输入只能包含<code>a-zA-Z0-9_-.</code>字符。
     *
     * @param sql sql
     * @return 安全sql，否则返回null
     */
    public static String sqlFilter(String sql) {
        if (!FILTER_PATTERN.matcher(sql).matches()) {
            return null;
        }
        return sql;
```

```
    }

    /**
     * 将非<code>0-9a-zA-Z/-.</code>的字符替换为空
     *
     * @param str 字符串
     * @return 被过滤的字符串
     */
    public static String replaceSpecialStr(String str) {
        StringBuilder sb = new StringBuilder();
        str = str.toLowerCase();
        for(int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            // 如果是0-9
            if (ch >= 48 && ch <= 57 ){
                sb.append(ch);
            }
            // 如果是a-z
            else if(ch >= 97 && ch <= 122) {
                sb.append(ch);
            }
            else if(ch == '/' || ch == '.' || ch == '-'){
                sb.append(ch);
            }
        }

        return sb.toString();
    }

    public static void main(String[] args) {
    }

}
```