

Write Up F3ngShu1 – TechTonicExpo



Team: Vincent, Rizqi, Julius

DAFTAR ISI

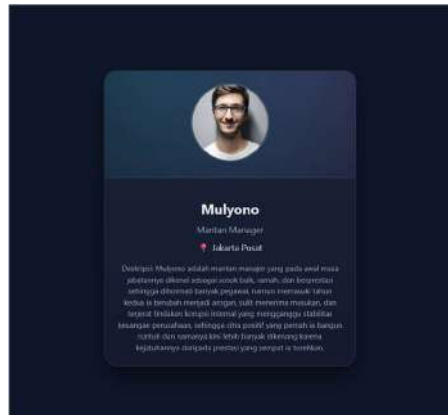
• WEB EXPLOITATION	3
• CRYPTOGRAPHY	10
• REVERSE ENGINEERING.....	18
• FORENSIC	22

- **WEB EXPLOITATION**

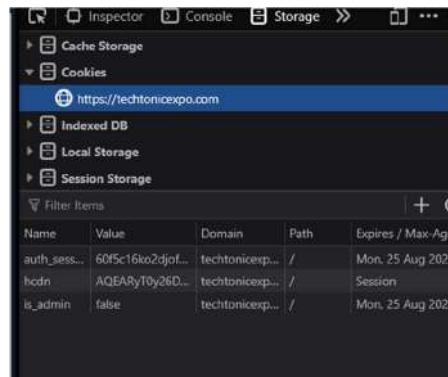
1. **Cookie Trickery**

Description:

Hint: *Terkadang, identitasmu ditentukan bukan dari login, tapi dari apa yang tersimpan di browser. 🍪 Coba ubah sedikit “isi”mu, dan lihat apa yang terbuka.*



Disajikan sebuah card seperti itu. Jika kita membuat dev tools dan pergi ke storage Dan di cookies akan terlihat seperti ini



Untuk bisa mendapatkan Flagnya_harus jadi admin. Jadi kita bisa ubah isi dari is_admin menjadi true dan refresh halamannya



Flag: TechtonicExpoCTF{C00K13_TR1CK3RY}

2. Bypass OTP

Descption:

Message (Deskripsi Challenge): Sistem OTP di Jupiter seolah-olah jadi benteng terakhir sebelum kamu bisa masuk. Tapi apakah benteng ini benar-benar kokoh, atau hanya ilusi yang bisa runtuh jika kamu berani mencoba menghapus sesuatu yang tidak seharusnya ada?

Temukan cara untuk menembusnya dan buktikan bahwa kamu tidak butuh angka keberuntungan untuk bisa lolos.

Cobalah berinteraksi dengan mekanisme OTP pada endpoint berikut:

Hint:

Kadang keamanan itu hanya ilusi. Apa jadinya jika verifikasi tidak benar-benar memeriksa apa yang kamu kirim? 📌 ✨

Pendaftaran

Nama Lengkap:
admin

Username:
admin

Nomor Telepon:
admin

Alamat:
admin

Password:

Daftar

Ketika dikirim website menanyakan auten 2 faktor

2FA Authentication

admin

Submit

Ketika di submit keluar tulisan ***Invalid! OTP***

Saya gunakan burp suite untuk melihat apa yang sebenarnya terjadi pada autentikasi 2 faktor ini. Ketika sampai di autentikasi 2 faktor ternyata ada sebuah req kode otp yang kita isi tadi

```
--
14 Accept: text/html,application/xhtml+xml,a
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer: https://techtonicexpo.com/jupite
20 Accept-Encoding: gzip, deflate, br
21 Priority: u=0, i
22
23 otp=admin
```

Kita bisa melakukan bypass dengan cara... ya hapus aja otpnya wkkw setelah itu klik forward

```
17 Accept-Ranges: bytes
18
19 Welcome to the Techtonic Expo CTF Challenge!, admin kamu berhasil nge bypass
   OTP nya. <br>
   Flag: TechtonicExpoCTF{BypasOTP_EzUbb777}
```

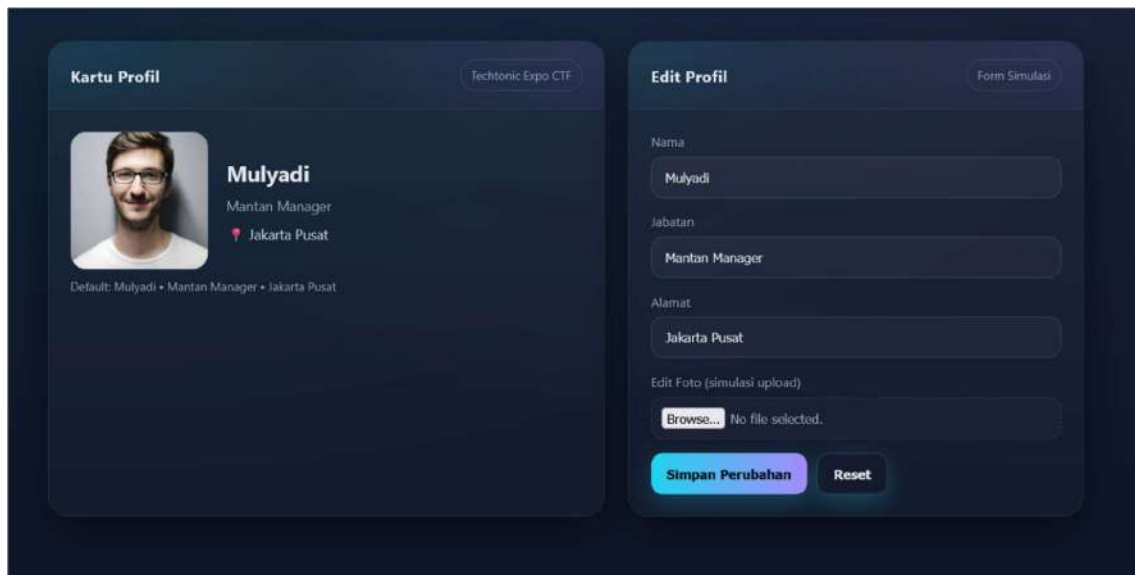
Dapet deh

Flag: TechtonicExpoCTF{BypasOTP_EzUbb777}

3. Disguised Upload (Change Extensions)

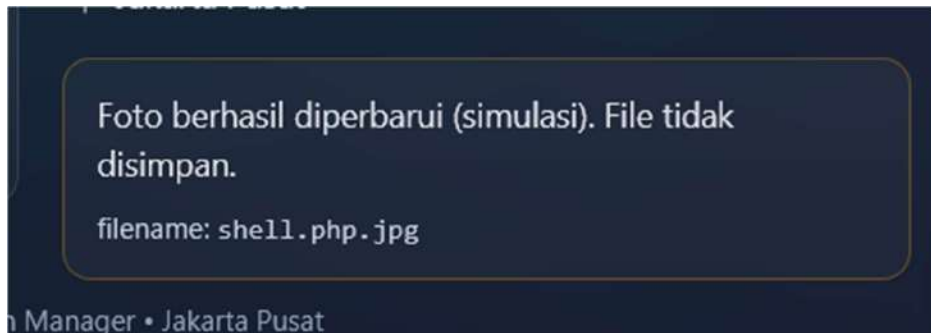
Description:

Hint: Kadang apa yang terlihat polos ternyata menyimpan rahasia. Coba ubah penampilan file sebelum kamu mengirimnya... apakah sistem masih bisa menebaknya? 😊

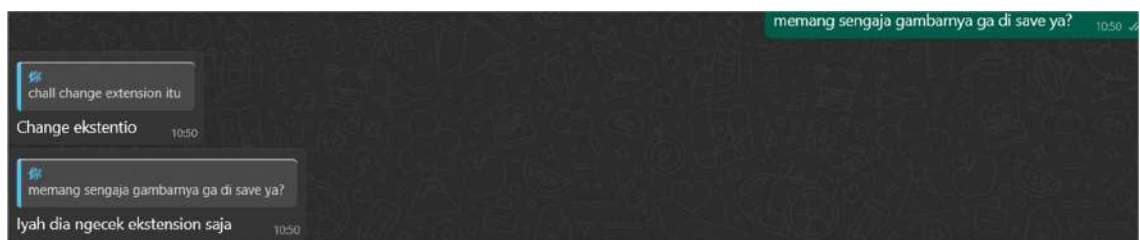


The screenshot shows a user profile interface. On the left, under 'Kartu Profil', there is a profile picture of a man with glasses, the name 'Mulyadi', title 'Mantan Manager', and location 'Jakarta Pusat'. Below this is a default string: 'Default: Mulyadi • Mantan Manager • Jakarta Pusat'. On the right, under 'Edit Profil', there is a 'Form Simulasi' button. The form contains input fields for 'Nama' (filled with 'Mulyadi'), 'Jabatan' (filled with 'Mantan Manager'), and 'Alamat' (filled with 'Jakarta Pusat'). Below these is a section for 'Edit Foto (simulasi upload)' with a 'Browse...' button and the text 'No file selected.'. At the bottom of the form are two buttons: 'Simpan Perubahan' and 'Reset'.

diberikan sebuah form edit profil dan ada file upload disana brarti ini ada lah file upload injection pikir saya awalnya. Saya mencoba mengupload **shell.php.jpg** pada gambar dan saya shock ituu memang berhasil membypass filter tapi file saya tidak tersimpan. Bahkan saya juga menggunakan shell.gif karna saya pikir itu mungkin bisa berhasil namun takdir berkata lain v:



Setelah satu jam mengalami stuck yang sama. Dan Karna saya sudah tidak tahan saya menanyakan kepada probset.



Saya disini speechless. Dantetikak saya mencoba melakukan bruteforce extension saya mendapatkan extension yang tepat **hiii.jpg.php** atau bisa di bilang **filename.jpg.php**

```
<div class="notice ok">
  Bypass terdeteksi! (filename
  berakhiran .jpg.php)

  <div class="
    file-name">
      filename: <code>
        hiii.jpg.php
      </code>
    </div>
  </div>

  <div class="flag">
    00FLAG: <strong>
      TechtonicExpoCTF{UPL04D_3X3CUT3}
    </strong>
  </div>
```

Flag: TechtonicExpoCTF{UPL04D_3X3CUT3}

4. JS Secret Hunter

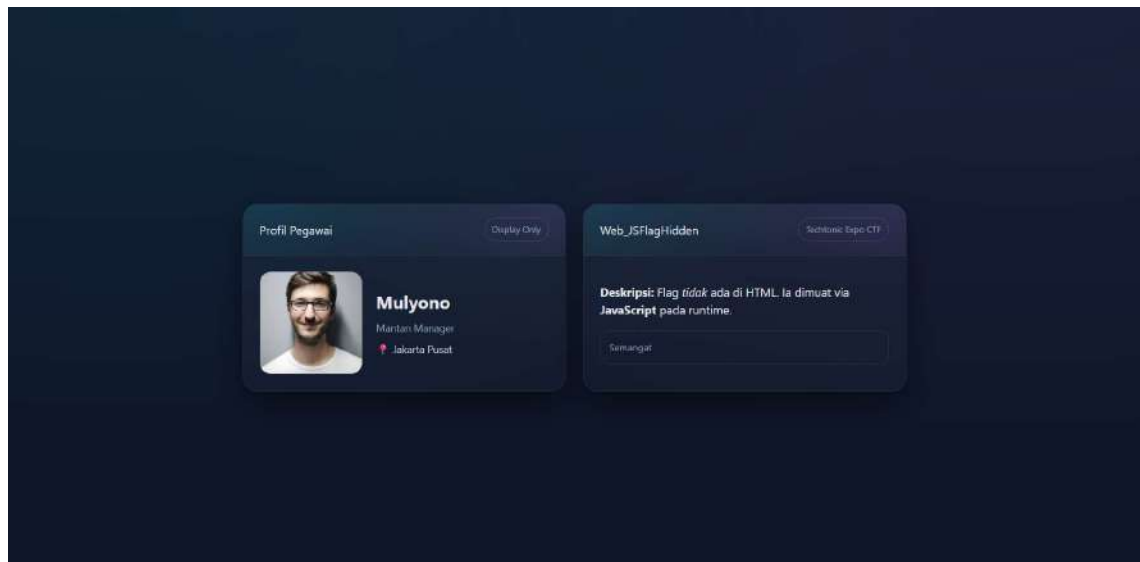


Disini kita di beri deskripsi

Flag tersembunyi di balik kode JavaScript. Kadang kamu harus menelusuri file terpisah dan menerjemahkan isinya sebelum bisa membacanya.

<https://techtonicexpo.com/uranus>

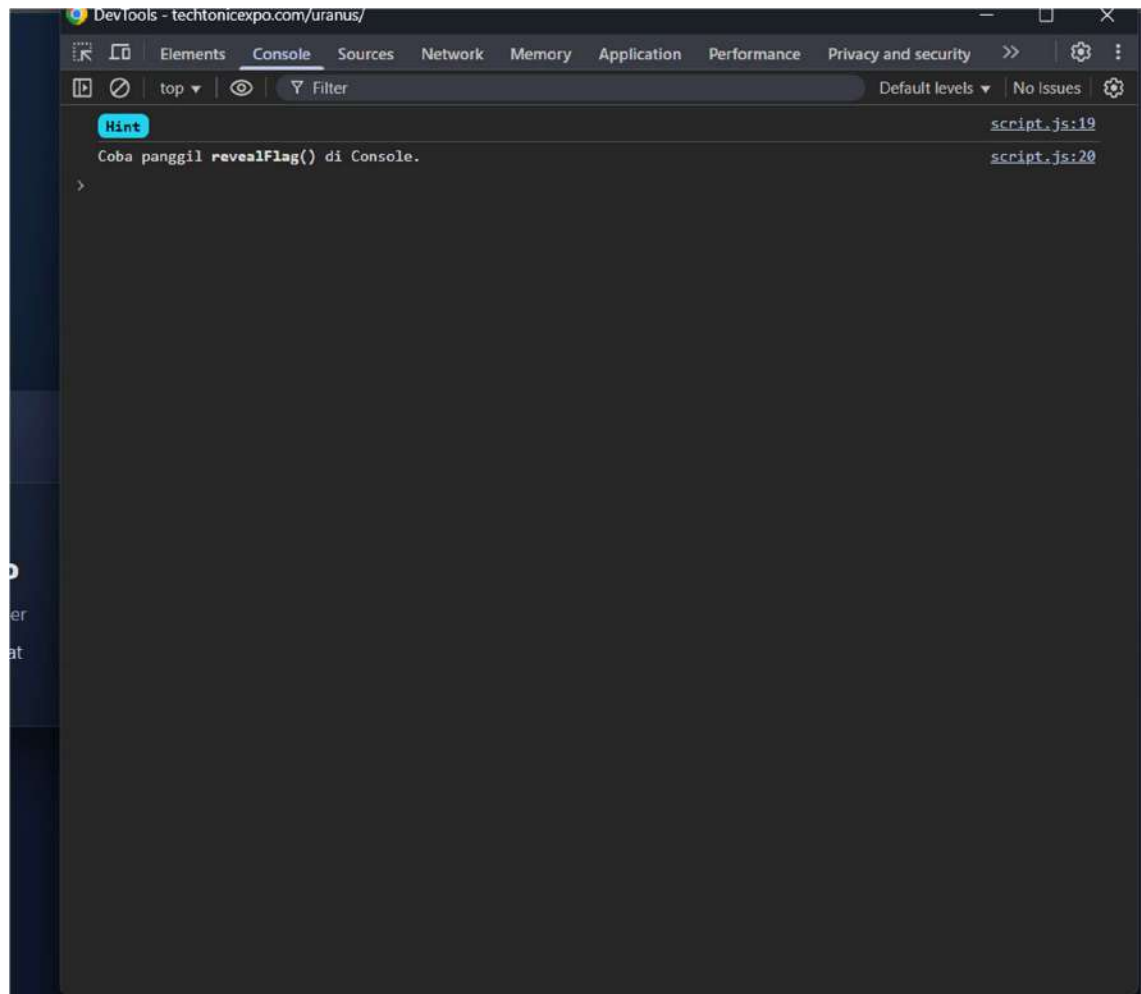
setelah ku buka websitenya pada halaman utama memunculkan



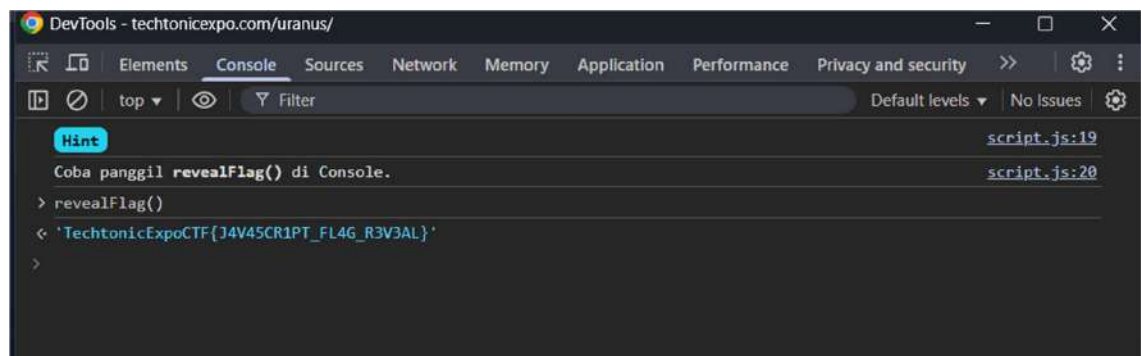
Huemm dia ada deskripsi disitu yaitu

Deskripsi: *Flag tidak ada di HTML. Ia dimuat via JavaScript pada runtime.*

Setelah itu aku inspeksi terus aku cek bagian console



Dan ya disitu ada hint berupa function `revealFlag()` terus saya coba ketik di console `revealFlag()`

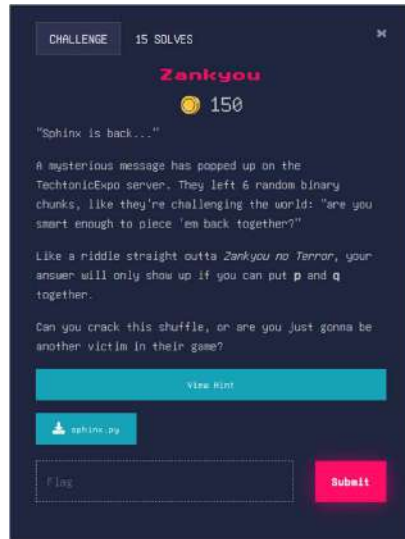


dan waw tidak saya duga ternyata dia beneran memberi kita flagnya dan bingo 🎉 🎉

Flag: *TechtonicExpoCTF{J4V45CR1PT_FL4G_R3V3AL}*

- CRYPTOGRAPHY

1. Zankyou



Di Challenge diatas kita mendapatkan deskripsi

"Sphinx is back..."

A mysterious message has popped up on the TechtonicExpo server. They left 6 random binary chunks, like they're challenging the world: "are you smart enough to piece 'em back together?"

*Like a riddle straight outta Zankyou no Terror, your answer will only show up if you can put **p** and **q** together.*

Can you crack this shuffle, or are you just gonna be another victim in their game?

Kalau kita perhatiin, jelas banget soal ini **main di area RSA**. Kata kuncinya “p and q”, ditambah ada file Python (sphinx.py) yang nyimpen potongan biner.

Setelah buka sphinx.py, isinya (potongan penting) kira-kira gini:

- $e = 65537 \rightarrow$ public exponent standar RSA.
- c =
408847730453539116793211358829625222749206768131859112882
579398340507305412986216412141862649412048726520625182302
643014627423020776951044649206908642994997304243377347499
547225547652903079220574779112662328758610740076090497312
675855317514361975326926212591891007671255586960785239851
322326433240392690798291782616738296426759620263703963011
898712245423044036561498927905205256302623374577628600735

288015210260856832776074193715903735595821669047884604213
758951 → ciphertext yang harus didekripsi.

- `pq_bit_shuffle = [chunk0, chunk1, ..., chunk5]` → list berisi **6 string biner**, masing-masing panjang 256 bit.

RSA modulus N biasanya dibentuk dari $p * q$, di mana p dan q adalah bilangan prima besar. Dari challenge ini, kita tahu p dan q sudah dipotong jadi 3 chunk biner masing-masing, lalu diacak jadi 6 potongan.

Tugas kita: cari cara nyusun ulang 6 potongan itu menjadi p dan q yang valid.

Kita tahu ada 6 chunk total. Berarti ada $6! = 720$ kemungkinan urutan. Angka segini relatif kecil, brute force masih gampang.

Untuk setiap permutasi, kita bisa:

1. Ambil 3 potongan pertama → gabung jadi kandidat p .
2. Ambil 3 potongan terakhir → gabung jadi kandidat q .
3. Konversi string biner jadi integer.
4. Tes apakah p dan q prima. Kalau ya → berarti kandidat valid.
5. Kalau valid, hitung:
 - $N = p * q$
 - $\phi = (p - 1)(q - 1)$
 - $d = \text{pow}(e, -1, \phi)$
6. Dekripsi ciphertext: $m = \text{pow}(c, d, N)$
7. Konversi m ke bytes → flag.

Dengan begini, kita tinggal nunggu kombinasi yang pas keluar.

Setelah itu aku membuat script python ku sendiri untuk menjalankan tugas diatas

```

import itertools, sympy

e = 65537

c =
408847730453539116793211358829625222749206768131859112882579398340507305412986216412141
86264941204872652062518230264301462742302077695104464920690864299499730424337734749954
72255476529030792205747791126623287586107400760904973126758553175143619753269262125918
91007671255586960785239851322326433240392690798291782616738296426759620263703963011898
71224542304403656149892790520525630262337457762860073528801521026085683277607419371590
3735595821669047884604213758951

pq_bit_shuffle =
['00111010011011100010001100010111110001001110100011001100011011101010111010001001100111
000111111000011011111011101100011100100101001011101000111100001000101101010011011001
0001100110111000111010111111101101000010110000001101001011001101011000100100001',
'11001100111011010101011010010001001010011101010111001011101110010000001011001000011001
00101000110000111011000100101001111010011010000010001100001101111111010111101000000010
00001011110110110101011100001111111111111101100111111101000010110000101001101',
'101010000100101010011010110001010011111110011111101011010001001110010100101101110000100
1001101110001100101111100111110000010110100010011100010110101010111010100001100100110001
01100101101100000100101001011000111001100101110010011111111110100000001111000111',
'10110101000111000100001001011010010011000011111100000111001010000100001101110010110010
001011101101100101011101100001010111001101100011111010000110011111011001101110001010111
001001010100000011101011010001101111101000100110100111001000010110111111011111',
'10100101000100010000100000011011101001000100010010110110010100001001101001010000101
11011010000111111000100011001011010000000101001110011111100000011010110011100010101001
0001101101110110011011001110011100100101110011101111010000101000101000110',
'0100000010111000011111000110100001111011111011010110011011000110011001001011001001010011
111110101001101101101110101011101000110101001011010110111110000001001000100001001011
11010110010101011110011001011111111011111001010111001000001011001111110000100']

for perm in itertools.permutations(chunks):

    p = int("".join(perm[:3]), 2)

    q = int("".join(perm[3:]), 2)

    if sympy.isprime(p) and sympy.isprime(q):

        N = p * q

        phi = (p - 1) * (q - 1)

        d = pow(e, -1, phi)

        m = pow(c, d, N)

        flag = m.to_bytes((m.bit_length() + 7)//8, "big")

        print("FLAG:", flag.decode())

        break

```

Waktu script jalan:

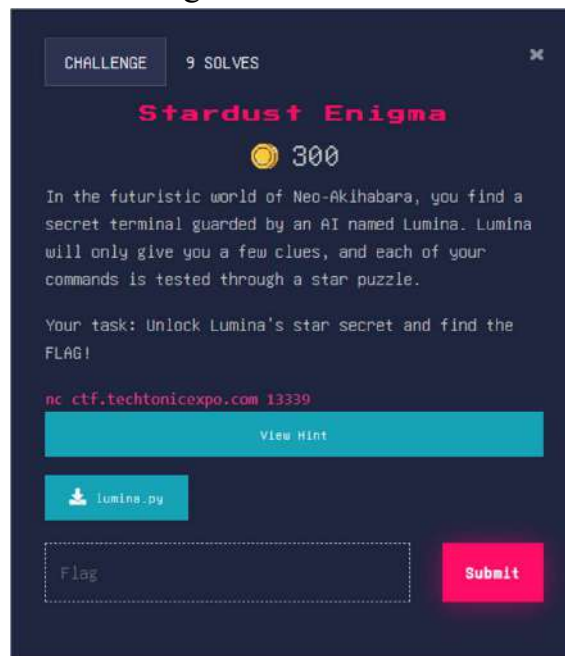
- Kebanyakan kombinasi langsung gagal karena p/q bukan prima.
- Tapi akhirnya ada 1 kombinasi yang valid.
- Susunan chunk yang benar ternyata:
 - **p = chunk3 || chunk4 || chunk2**
 - **q = chunk5 || chunk6 || chunk1**

Begitu ketemu, RSA private key d bisa dihitung dan ciphertext berhasil didekrips

Dan kita menemukan Flagnya Bingo 🎉 🎉

FLAG: *TechtonicExpoCTF{RS4_1S_0UR_F4M}*

2. Stardust Enigma



Pada challenge di atas kita di kasi sebuah deskripsi

In the futuristic world of Neo-Akihabara, you find a secret terminal guarded by an AI named Lumina. Lumina will only give you a few clues, and each of your commands is tested through a star puzzle.

Your task: Unlock Lumina's star secret and find the FLAG!

nc ctf.techtonicexpo.com 13339

dan kita di kasi sebuah hint yaitu:

- Lumina likes to hide what's actually real.
- The - and % operators have a significant meaning.
- The FLAG is encrypted with RSA, with $e = 65537$ and $n = p * q$.
- One prime is smaller than the other.

dan kita di kasi file python *lumina.py*

Saya mencoba melihat isi di dalam nc nya apa dan saya menemukan 4 opsi

```
1. Get new prime
2. Get leak
3. Get flag
4. Exit
>
```

dan aku mencoba memilih opsi nomor 1 dan dia memunculkan ini

```
1. Get new prime
2. Get leak
3. Get flag
4. Exit
> 1
which prime? (p/q)
>
```

saya pun memilih p setelah itu dia Kembali lagi ke opsi awal, saya bingung apa yang sebenarnya terjadi, dan saya cek file pythonnya dan saya menemukan bahwa opsinya ternyata dia sesuai dengan nama opsinya yaitu kita mengregenerate primenya

```
if choice == 1:
    if lock:
        print("You can't do that anymore!")
        continue

    print("which prime? (p/q)")
    print("> ", end="")
    prime = input()

    if prime == "p":
        p = getPrime(1024)
    elif prime == "q":
        q = getPrime(512)
    else:
        print("What?")
        continue

    n = p * q
    lock = True
```

Terus saya mulai memilih opsi kedua dan seperti source codenya dia mulai kasi kita sebuah leak yang sedikit membingungkan

1. Get new prime
2. Get leak
3. Get flag
4. Exit

> 2

choose leak p ? q (+-*/%)

> +

p + q =

634267640214061287745677345078807194443999779134577283749807628
026052800473447709571143390834393656459049327844062458947182463
759126891623198539123678578170013748442881850831505440172700171
156637840238141004349088298387660150815058931018682234734646800
007179296748279370908934585094818894258148474201846907274480025
316522358987231929011095249178121633336900187674418595015131471
661919429639142433628108786064062778424309489894330494116368951
901187312674439799572

dan terus aku terus bereksperimen

> 2

choose leak p ? q (+-*/%)

> %

p % q =

417769635590664872831345680431347892580610838164172761419736720

503003132720191974761921626224883405582249141250772711524081120
6386637647060687301179440063

Hint di soal:

- “Lumina likes to hide what’s real.”
- “The - and % operators have significant meaning.”
- RSA dengan $e=65537$, $n=p*q$.
- Salah satu prime lebih kecil.

Artinya: kita harus pakai leak - dan % (yang plaintext) untuk membongkar RSA.

Dari sesi nc, kita sudah dapat:

- $p + q$ (encrypted):

$c_{plus} =$

121597195462004034521711130924572902215593811693911327479157432
212766149767289511996833284291554830355456524215321418189215128
114326430479565674264313973280030252739675727275495446392726425
794083562788778897463278014465669132983065615592285978658334477
077768321589904023024305740320131849333287945473592686528900534
745911271567692805606546641879670526740324984695575308011802245
187460699996046483539428229913368563465014836646455955383990983
9727655416808474602545

- $p - q$ (plaintext):

$D =$

143571412260993633013628658707130598248292026511461058931489332
198772941241467330931441065897880960641834054494660493534399240
305811563758974592210783633210089706530669492957694268675608792
198389748197738973618334146985372569592477466449254756211674966
299386282969728937076961486864801572646790349816133771280

- $p \% q$ (plaintext):

$r =$

137963254347207148881265507691477879738854939000215767037255478
94364980128316608778312179227963645265117580768850703939392202
6614451156520365504772346597

- Ciphertext flag:

$c =$

361834684645290157144507529597744757892255552350754646073725150
124082994150161621421375218404525210543581661097354569208718894
757123061450155647171794423878051960331316124009091118223825412
373925612147768136453762650614556454958258409586048339460287812
008186170585105523483580064771489260585920928688493573312849025
162054212629561617447397533794138394386078148359379551428225772

871882244177548769704870536894871805671118472909237694963928642
123880494647417996748

Dari leak:

- $D = p - q$
- $r = p \% q$
- Kita tahu: $p = q + D$

Karena $p \equiv r \pmod{q}$, maka:

$$(p + q) \equiv r \pmod{q}$$

RSA server memberikan:

$$c_plus = (p + q)^e \bmod n$$

Sehingga $(r^e - c_plus)$ adalah kelipatan q .

Selain itu, $(D - r)$ juga kelipatan q , karena:

$$D = p - q$$

$$p \% q = r \rightarrow p = kq + r$$

$$\text{Maka } D - r = (kq + r - q) - r = (k-1)q$$

Jadi kita bisa dapatkan q dengan:

$$q = \gcd(D - r, r^e - c_plus)$$

Terus mari kita pulihkan Pulihkan RSA key

1. Hitung $q = \gcd(D - r, r^e - c_plus)$.
2. $p = q + D$.
3. $n = p * q$.
4. $\phi = (p-1)(q-1)$.
5. $d = e^{-1} \bmod \phi$.
6. Decrypt ciphertext flag:
7. $m = c^d \bmod n$
8. Convert ke bytes \rightarrow FLAG.

Jadi terus saya membuat script pythonnya

```

from math import gcd

D = 1435714122609936338136286587071305982482920265114610589314893321987729412414673309314410558978809606418340544946604933343992403058115637589745922107834
r = 1379632543472071488812655076914778797388549390002157670372554789436498012831660877831217922796364526511758076885070393933922026614451156520365504772344
c_plus = 12159719546200403452171113092457290221559381169391132747915743221276614976728951199683328429155483035545652421532141818921512811432643047956567426
c = 361834684645290157144507529597744757892255523507346460737251501240829941501616214213752184045252105435016610973545692087188947571230614501556471717944
e = 65537

S = D - r
t = (pow(r, e, S) - (c_plus * S)) % S
q = gcd(S, t)
p = q + D

n = p * q
phi = (p - 1) * (q - 1)

def invmod(a, m):
    def egcd(a, b):
        if b == 0: return (1, 0, a)
        x0, y0, g0 = 1, 0, a
        x1, y1, g1 = 0, 1, b
        while g1:
            q = g0 // g1
            x0, x1 = x1, x0 - q * x1
            y0, y1 = y1, y0 - q * y1
            g0, g1 = g1, g0 - q * g1
        return x0, y0, g0
    x, _, g = egcd(a, m)
    return x % m

d = invmod(e, phi)
m = pow(c, d, n)
flag = m.to_bytes((m.bit_length() + 7) // 8, 'big')

print(flag.decode())

```

Jadi dari script di atas

- $D \rightarrow$ hasil $p - q$
- $r \rightarrow$ hasil $p \% q$
- $c_plus \rightarrow$ hasil $p + q$, tapi diekspose sebagai RSA ciphertext $(p+q)^e \bmod n$ (ciphertext leak)
- $c \rightarrow$ ciphertext flag asli
- $e \rightarrow$ eksponen publik standar RSA

Dan saat di jalankan Bingo 🎉 🎉 kita dapat flagnya

Flag: *TechtonicExpoCTF{fu7ur3_i5_In_y0ur_h4nd5}*

Note: ini flagnya diliat-liat mirip ya sama yang challenge dari forensic tentang AES

• REVERSE ENGINEERING

1. Kirigaya Secret

Description:

*Kirito found a puzzle guarded by the AVL Tree, aka Locky **Lock**. Enter the binary **sequence** to navigate the tree and find the secret code. Be careful, one wrong move will reset the **level**!*

Hint:

*Anime inspiration: Sword Art Online, Kirito facing a digital puzzle. The random seed is **fixed**, so the sequence can be reversed. Check the **decrypt()** function in the .pyc file.*

1. *You might want to explore the AVL tree logic inside.*
2. *XOR decryption is used at the end.*
3. *Random seed is fixed: 199*
4. *Target Python version: 3.12*

Attachments:

Kiriya.pyc

Diberikan sebuah file **kirigaya.pyc** dan tertulis di hint bahwa itu terinspirasi dengan anime **sword art online** disitu dia bilang kita harus memahami **avl logic tree** dan ada xor decrypt di akhir. Dan seednya itu fix di angka 199 dan pythonnya 3.12. cara saya mendapatkan flagnya hanya dengan menggunakan seed yang diberikan.

Pertama tama sayaa mecoba melakukan [decompile](#) untuk melihat apa yang ada di balik program itu. Dan ya saya melihat itu bekerja dengan tkinter. Tapi bagian paling menarik bukan kodenya tapi ada sebuah variable array yang menampung flagnya pada line 130an

```
FLAG = [84, 101, 99, 104, 116, 111, 110, 105, 99, 69, 120, 112,
111, 67, 84, 70, 123, 115, 48, 109, 51, 111, 110, 51, 95, 116, 48,
108, 100, 95, 109, 51, 95, 116, 48, 95, 64, 95, 109, 48, 114, 51, 95,
99, 104, 52, 108, 108, 125]
```

Karna saya merasa yakin kalau itu pasti flag saya mencoba membalikan logikanya dengan membuat script python saya untuk mendecrypt xor itu dengan menggunakan seed yang diberikan

Ini kodenya:

```

>>> def dec(enc):
...     random.seed(199)
...     flag = []
...     for i, b in enumerate(enc):
...         r = random.randint(0,255)
...         flag.append(b^r)
...     return bytes(flag)
...
>>> flag_enc = [
...     84, 101, 99, 104, 116, 111, 110, 105, 99, 69, 120, 112, 111, 67, 84, 70,
...     123, 115, 48, 109, 51, 111, 110, 51, 95, 116, 48, 108, 100, 95, 109, 51,
...     95, 116, 48, 95, 64, 95, 109, 48, 114, 51, 95, 99, 104, 52, 108, 108, 125
... ]
>>> print(bytes(flag_enc).decode())
TechtonicExpoCTF{s0m3on3_t0ld_m3_t0 @_m0r3_ch4ll}
>>> |

```

Flag: TechtonicExpoCTF{s0m3on3_t0ld_m3_t0 @_m0r3_ch4ll}

2. Random XOR

Description:

Just a simple program for file encryption.

Hint:

No Hint

Attachments:

- Encrypt <- program executable
- Flag_enc.txt <- flag yang terenkripsi

Setelah saya mendownload kedua file saya mengecek jenis file dari mereka berdua

```

> file encrypt
encrypt: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=d044bd0f34a3960
1a6fald2a2d8f215653c47849, for GNU/Linux 3.2.0, not stripped
> file flag_enc.txt
flag_enc.txt: data

```

Dan juga pengecekan menggunakan xxd

```

> xxd -p flag_enc.txt | head -5
cc75ab6876ad7466299582037590c6ab2acc7a90b4d3975f482d72ae0f21
d868106862a59e8fba1ca3a0f101868ce2

```

Dan tampaknya harus melakukan reverse terhadap program enkripsi ini

Saya melakukan decompiler dengan menggunakan ghidra dan mendapati isi dari function main()

```

time_t tVar1;
undefined8 uVar2;
long lVar3;
long in_FS_OFFSET;
uint local_3c;
int local_38;
int local_34;
uint local_30;
int local_2c;
FILE *local_28;
void *local_20;
FILE *local_18;
long local_10;

local_10 = *(long *) (in_FS_OFFSET + 0x28);
tVar1 = time((time_t *) 0x0);
local_3c = (uint) tVar1;
srand(local_3c);
local_28 = fopen((char *) (param_2 + 8), "rb");
if (local_28 == (FILE *) 0x0) {
    puts("Error opening file.");
    uVar2 = 1;
}
else {
    fseek(local_28, 0, 2);
    lVar3 = ftell(local_28);
    local_34 = (int) lVar3;
    fseek(local_28, 0, 0);
    local_20 = malloc(0x40);
    fread(local_20, 1, (long) local_34, local_28);
    for (local_38 = 0; local_38 < local_34; local_38 = local_38 + 1) {
        local_30 = rand();
        local_30 = local_30 & 0xff;
        local_2c = rand();
        *(byte *) ((long) local_20 + (long) local_38) =
            *(byte *) ((long) local_20 + (long) local_38) ^ (byte) local_30;
    }
    local_18 = fopen((char *) (param_2 + 0x10), "wb");
    fwrite(local_20, 1, (long) local_34, local_18);
    printf("Encrypted flag: %s\n", local_20);
    fclose(local_28);
    fclose(local_18);
    free(local_20);
    uVar2 = 0;
}
if (local_10 != *(long *) (in_FS_OFFSET + 0x28)) {
    /* WARNING: Subroutine does not return */
}

```

yah bginilah ya bahasa alien. Wkwkkw

Setelah dicek alur program ini ternyata logiknya adalah:

Melakukan generate seed -> buka file input -> hitung ukuran dari file -> melakukan alokasi buffer(0x40) ini cuma 64 bytes -> ada loop enkripsi dan terakhir simpan hasilnya.

Dan ya ini mudah saja wkkw. Karna yang pertama seed nya ditulis di file output dan di program itu pakai time() <- ini masih bisa di predict

Jadi hal yang saya lakukan:

1. Ambil seed dari file
2. generate ulang seq xornya
3. decrypt pake xor lagi (karena A XOR B XOR B = A)

Kita bisa bikin solver kaya gni:

```

... import struct
... from ctypes import CDLL
...
... libc = CDLL("libc.so.6")
...
... def solver():
...     with open("flag_enc.txt", "rb") as f:
...         data = f.read()
...
...     seed = struct.unpack("<I", data[:4])[0]
...     encrypted_data = data[4:]
...
...     # print(f"Seed: {seed}")
...     # print(f>Data size: {len(encrypted_data)} bytes")
...
...     libc.srand(seed)
...
...     decrypted = bytearray()
...     for i in range(len(encrypted_data)):
...         xor_key = libc.rand() & 0xff
...         libc.rand()
...
...         decrypted_byte = encrypted_data[i] ^ xor_key
...         decrypted.append(decrypted_byte)
...
...     flag = decrypted.decode('utf-8', errors='ignore')
...     print(f"\nFlag: {flag}")
...
... print(solver())

```

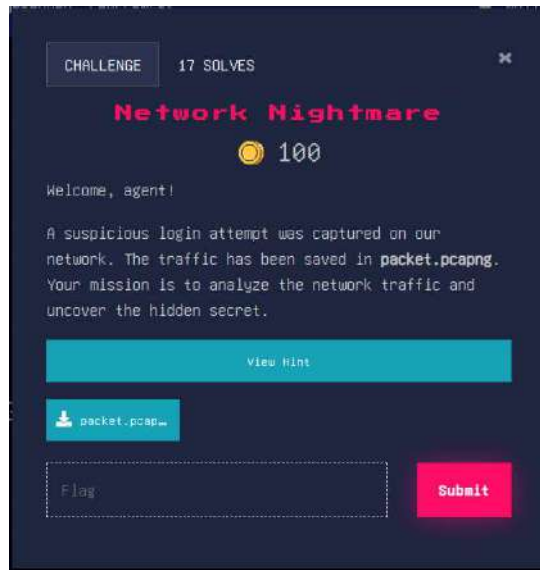
Dan ini flagnya

Flag: TechtonicExpoCTF{X0R_IS_N0t_R4nd0M_at_A11}

dan ya saya pun dapet **FIRST BLOOD** di challenge ini XIXIXIX

- **FORENSIC**

1. Network Nightmare



Pada challenge Network Nightmare kita di kasi deskripsi

Welcome, agent!

A suspicious login attempt was captured on our network. The traffic has been saved in packet.pcapng. Your mission is to analyze the network traffic and uncover the hidden secret.

Kita diberikan sebuah file *.pcapng*, format packet capture yang umum digunakan oleh Wireshark untuk analisis traffic jaringan. Ini langsung menunjukkan bahwa ini Adalah sebuah challenge network forensics.

Untuk tools yang saya gunakan Adalah

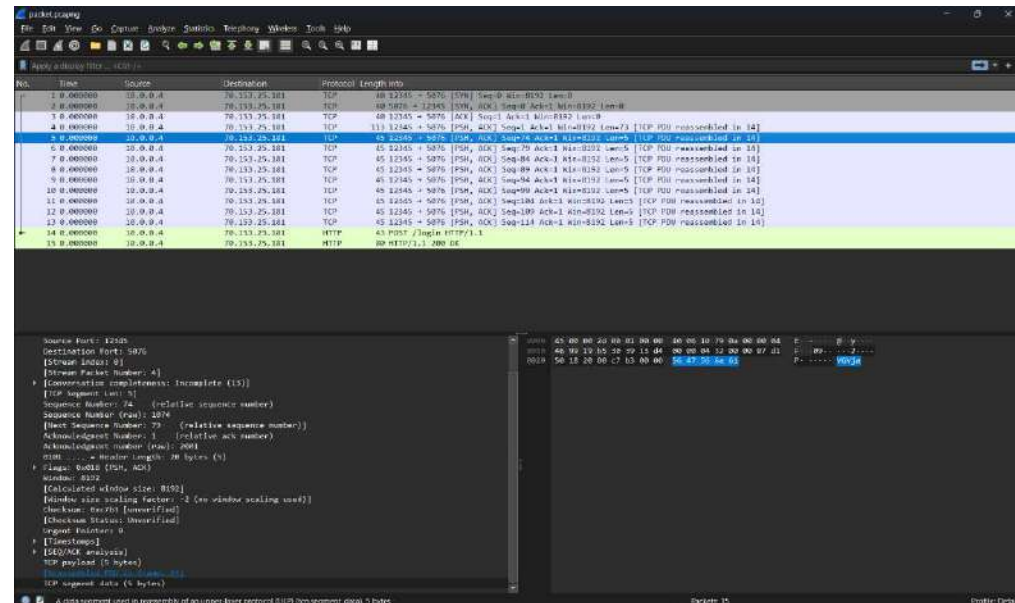
- **Wireshark** - Tool utama untuk analisis paket
- **Base64 decoder** - Untuk mendecode data terenkripsi yang ditemukan

Selanjutnya Saya memuat file *packet.pcapng* ke dalam **Wireshark** untuk memeriksa traffic jaringan. Capture menunjukkan serangkaian paket TCP dan HTTP antara dua endpoint:

- Source: 10.0.0.4:12345
- Destination: 70.153.25.181:5076

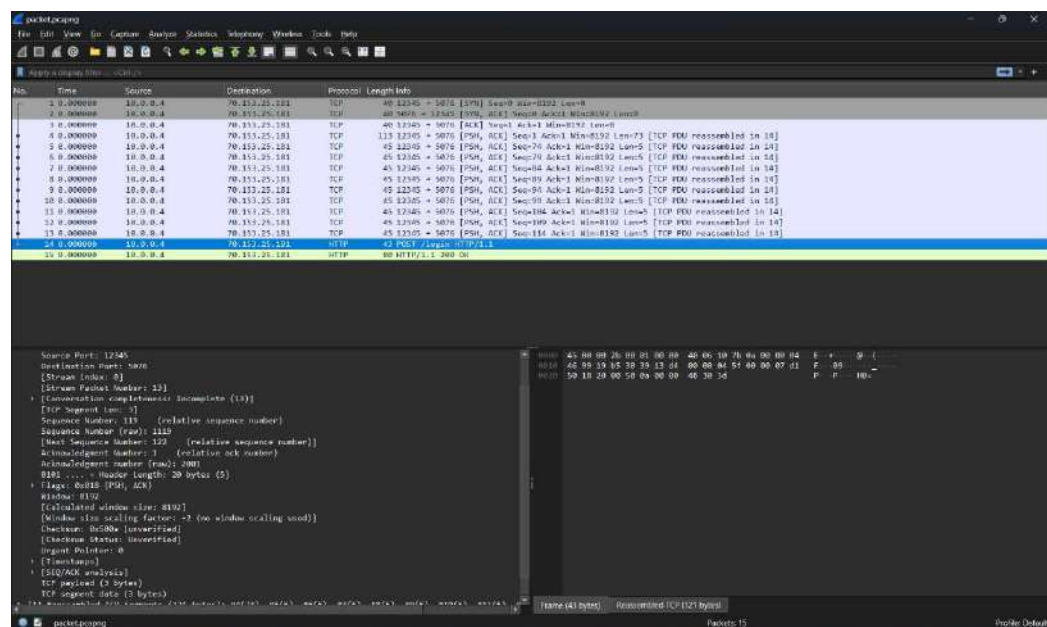
Dengan memeriksa daftar paket, saya memperhatikan pola menarik dalam komunikasi:

1. **TCP Handshake:** Pembentukan koneksi awal (SYN, SYN-ACK, ACK)
2. **Data Exchange:** Beberapa segmen TCP dengan flag PSH,ACK
3. **HTTP Traffic:** HTTP POST request dengan HTTP 200 OK response



Terus saya memeriksa kolom Info dengan lebih teliti. Saya memperhatikan apa yang tampak seperti data Base64 yang terfragmentasi tersebar di beberapa paket.

Dengan mengikuti TCP stream dan memeriksa paket individual, saya menemukan bahwa data Base64 dipecah menjadi beberapa segmen TCP.



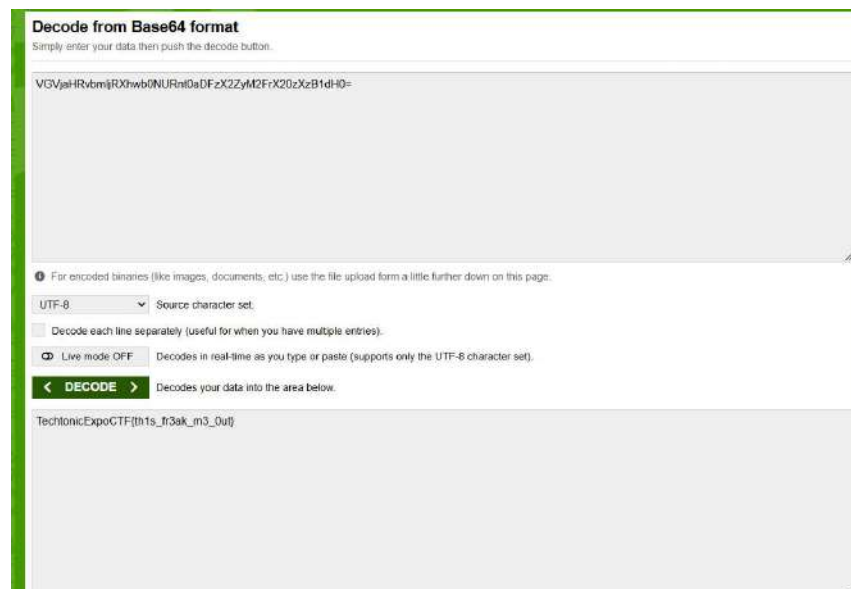
Dengan hati-hati memeriksa payload setiap paket dan mengikuti alur percakapan, saya berhasil merekonstruksi string Base64 lengkap yang telah terfragmentasi di seluruh traffic jaringan.

Data Base64 yang terfragmentasi muncul di bagian payload TCP, memerlukan perakitan ulang untuk mendapatkan pesan terencode yang lengkap.

ternyata data Base64 tersebut terpotong-potong di beberapa paket berbeda. Setelah saya gabungkan semua fragmen Base64 tersebut, saya mendapatkan string Base64 yang utuh yaitu:

VGVjaHRvbmljRXhwb0NURnt0aDFzX2ZyM2FrX20zXzBldH0=

Dan saya mendecodenya dengan [Base64Decoder](#) mengungkapkan flag tersembunyi yang sedang diekstraksi melalui komunikasi jaringan yang mencurigakan ini.



Dan Bingo 🎉🎉 kita mendapatkan flagnya
Flag: *TechtonicExpoCTF{th1s_fr3ak_m3_out}*

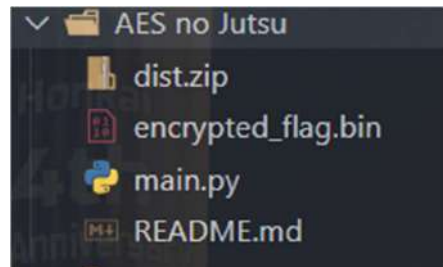
2. AES No Jutsu



Pada Challenge *AES No Jutsu* Kita di kasi sebuah deskripsi

Gulungan rahasia ini telah disegel dengan jutsu kuno. Bisakah kau membuka segel dan menemukan kebenaran di dalamnya?

Dan Kita di beri sebuah Zip file Bernama *dis.zip* dan isi dari file *dist.zip* tersebut Adalah sebuah file python Bernama *main.py* dan sebuah file bin yang bernama *encrypted_flag.bin* dan sebuah *Readme.md*



Mari kita analisis file *main.py* terlebih dahulu:

```
import argparse
from Cryptodome.Cipher import AES
from Cryptodome.Util.Padding import pad, unpad
import os

key =
bytes.fromhex('00112233445566778899aabbccddeeff00112233445566778899aabbccddeeff')
iv = bytes.fromhex('0102030405060708090a0b0c0d0e0f10')
```

BLOCK_SIZE = 16

Dari analisis kode, kita dapat melihat beberapa hal penting:

1. **Hardcoded Key & IV:** Kunci AES dan IV (Initialization Vector) sudah di-hardcode dalam script
2. **AES-256-CBC:** Menggunakan algoritma AES dengan mode CBC
3. **Key:**
00112233445566778899aabbccddeeff00112233445566778899aabbccddeeff (32 bytes untuk AES-256)
4. **IV:** 0102030405060708090a0b0c0d0e0f10 (16 bytes)

Script sudah menyediakan fungsi dekripsi. Kita tinggal menggunakannya:

```
python main.py --decrypt --input encrypted_flag.bin --output decrypt
```

setelah kita menjalankan script fungsi di atas kita mendapatkan sebuah file decrypt yang dia tidak ada extension nya terus saya mencoba mencari tahu dengan menggunakan script:

```
file decrypt
```

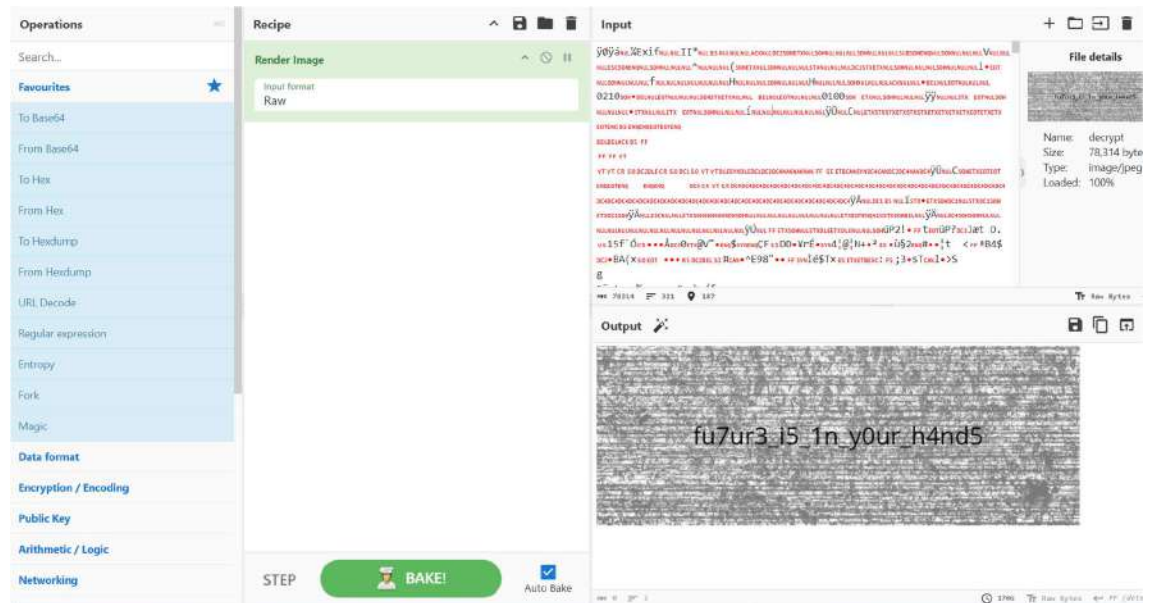
dan saya menemukan bahwa ini Adalah file jpeg

```
decrypt: JPEG image data, Exif standard: [TIFF image data, little-endian,
dentries=6, orientation=upper-left, xresolution=86, yresolution=94,
resolutionunit=2], progressive, precision 8, 640x237, components 3
```

Ternyata hasil dekripsi adalah file JPEG! Ini menunjukkan bahwa flag kemungkinan tersembunyi dalam gambar.

Karena file tersebut adalah gambar JPEG, saya menggunakan tools [CyberChef](#) terus saya Upload file decrypt ke [CyberChef](#) untuk melihat konten gambar

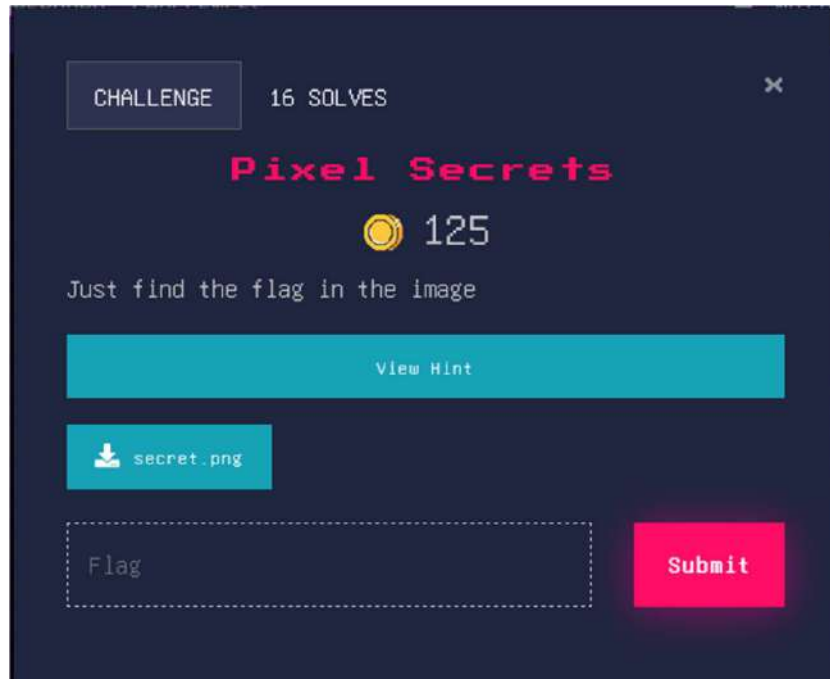
Setelah membuka gambar di [CyberChef](#), kita dapat melihat flag yang tersembunyi dalam gambar tersebut.



Dan yaps Bingo 🎉🎉 kita menemukan flagnya
 Flag: *TechtonicExpoCTF{fu7ur3_i5_1n_y0ur_h4nd5}*

- **STEGANOGRAPHY**

- Pixel Secrets



Pada challenge ini kita di berikan sebuah file yang bernama secret.png hal pertama yang perlu kita cek adalah file, file ini type apa.

```
pwd
kali@juliuswijaya: /mnt/c/user
(kali@juliuswijaya)-[/mnt/c/users/juliu/downloads]
$ file secret.png
secret.png: PNG image data, 50 x 50, 8-bit/color RGB, non-interlaced
(kali@juliuswijaya)-[/mnt/c/users/juliu/downloads]
$
```

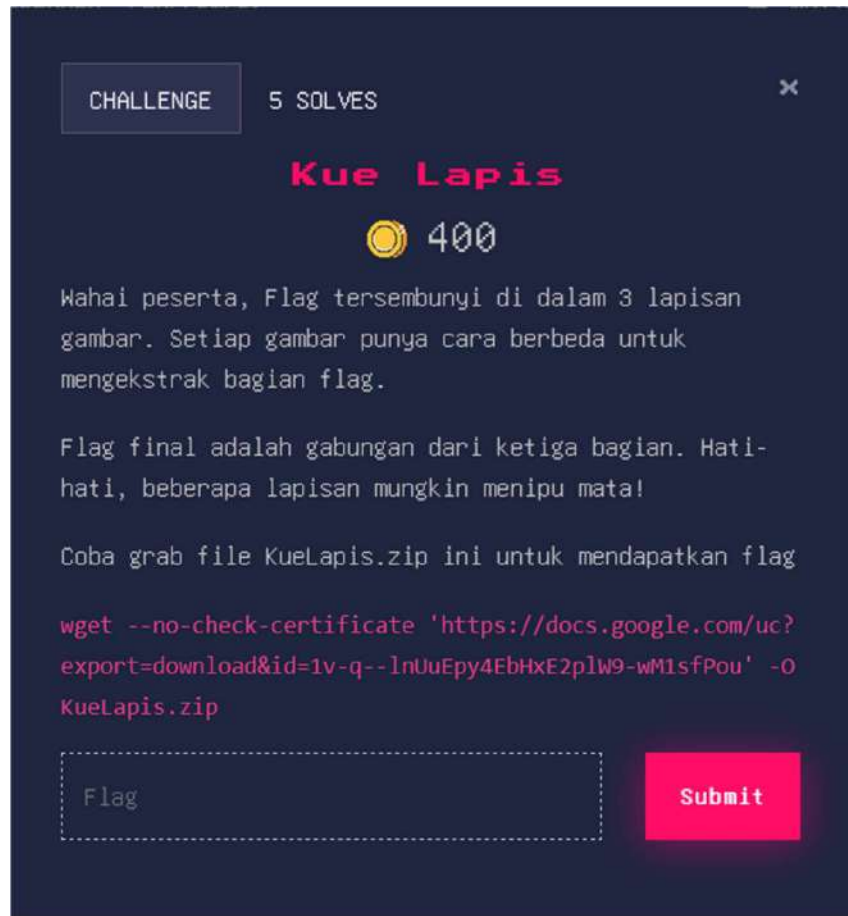
Nah dsni kita bisa lihat file nya berbentuk png image data yang berarti ada data tersembunyi di balik secret.png ini. Kalau sudah seperti ini kita bisa pake tools yang bernama zsteg

```
kali@juliuswijaya: /mnt/c/user  x  +  v
(kali@juliuswijaya)-[/mnt/c/users/juliu/downloads]
$ zsteg secret.png
b1,r,lsb,xy    .. text: "VGVjaHRvbmljRXhwb0NURntwMXgzbf9wMHczcl91bmwzNHNoM2R9"
b4,bgr,msb,xy  .. file: MPEG ADTS, layer I, v2, Monaural

(kali@juliuswijaya)-[/mnt/c/users/juliu/downloads]
$
```

Nah ketahuan dia ada code base64 kita langsung aja eksekusi code tersebut
VGVjaHRvbmljRXhwb0NURntwMXgzbf9wMHczcl91bmwzNHNoM2R9
FLAG : *TechtonicExpoCTF{p1x3l_p0w3r_unl34sh3d}*

- Kue Lapis



Kita liat challenge ini kue lapis, oke kita di beri code wget file zip nah setelah kita mendapatkan file zip tersebut kita langsung extract file tersebut dan kita mendapatkan 3 foto file png yang berarti kita dsruh cari 3 part flag

```

kali@juliuswijaya:~/mnt/c/users/juliu/downloads
$ wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=1v-q--lnUuEpy4EbHxE2plW9-wM1sfPou' -O KueLapis.zip
--2025-08-25 12:46:51-- https://docs.google.com/uc?export=download&id=1v-q--lnUuEpy4EbHxE2plW9-wM1sfPou
Resolving docs.google.com (docs.google.com)... 74.125.24.106, 74.125.24.138, 74.125.24.139, ...
Connecting to docs.google.com (docs.google.com)|74.125.24.106|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://drive.usercontent.google.com/download?id=1v-q--lnUuEpy4EbHxE2plW9-wM1sfPou&export=download [following]
--2025-08-25 12:46:52-- https://drive.usercontent.google.com/download?id=1v-q--lnUuEpy4EbHxE2plW9-wM1sfPou&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 172.253.118.132, 2404:6800:4003:c01::84
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|172.253.118.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1397333 (1.3M) [application/octet-stream]
Saving to: 'KueLapis.zip'

KueLapis.zip          100%[=====] 1.33M  474KB/s  in 2.9s

2025-08-25 12:46:57 (474 KB/s) - 'KueLapis.zip' saved [1397333/1397333]

kali@juliuswijaya:~/mnt/c/users/juliu/downloads
$ unzip KueLapis.zip
Archive: KueLapis.zip
  inflating: sanji.png
  inflating: ichiji.png
  inflating: ninji.png
  
```

Oke kita cek file tersebut

```
pwd x kali@juliuswijaya: /mnt/c/user + v
(kali@juliuswijaya)-[/mnt/c/users/juliu/downloads]
$ file sanji.png ninji.png ichiji.png
sanji.png: PNG image data, 1140 x 1568, 8-bit grayscale, non-interlaced
ninji.png: PNG image data, 1140 x 1568, 8-bit/color RGB, non-interlaced
ichiji.png: PNG image data, 924 x 1316, 8-bit/color RGB, non-interlaced
(kali@juliuswijaya)-[/mnt/c/users/juliu/downloads]
$
```

Oke kita dapat lihat disini bahwa semua file ini bentuk png image data, hal pertama yang saya lakukan adalah memakai tools zsteg karena tools ini untuk menganalisis data yang di sembunyikan dari si png ini

```
pwd x kali@juliuswijaya: /mnt/c/user + v
(kali@juliuswijaya)-[/mnt/c/users/juliu/downloads]
$ zsteg sanji.png
b1,r,lsb,xy .. text: "st3g4n0}"
(kali@juliuswijaya)-[/mnt/c/users/juliu/downloads]
$
```

Oke karena ini di akhir tutup kurung kurawal dapat di simpulkan ini part flag yang ke-3 yaitu *st3g4n0}*, kita lanjut lagi file png yang lain


```
pwd
kali@juliuswijaya: /mnt/c/user:
(kali@juliuswijaya)-[/mnt/c/users/juliu/downloads]
$ zsteg ichiji.png
imagedata      .. text: "f=AeBD`EC"
b1,b,lsb,xy    .. file: OpenPGP Public Key
b1,rgb,lsb,xy  .. text: "27:TechtonicExpoCTF{ku3_l4p1s_"
(kali@juliuswijaya)-[/mnt/c/users/juliu/downloads]
$ |
```

Oke kita menemukan part flag yang ke 1 yaitu *TechtonicExpoCTF{ku3_l4p1s_*

Oke kita lanjut lagi cari flag part ke-2

```
pwd
kali@juliuswijaya: /mnt/c/user:
(kali@juliuswijaya)-[/mnt/c/users/juliu/downloads]
$ zsteg ninji.png
imagedata      .. text: "&`%r\r\r\n\n"
b3,rgb,msb,xy  .. file: AIX core file fulldump 64-bit
b4,bgr,msb,xy  .. file: MPEG ADTS, layer I, v2, Monaural
(kali@juliuswijaya)-[/mnt/c/users/juliu/downloads]
$ |
```

Oke karena disini kita ada menarik, kita menggunakan zsteg –a agar keluar semua channel yang ada

```

kali@juliuswijaya:~/mnt/c/users/juliu/downloads
$ zsteg -a ninji.png
imagedata .. text: "G's\r\r\r\n\n"
b3,rgb,msb,xy .. file: AIX core file #ulldump 64-bit
b4,bgr,msb,xy .. file: MPEG ADTS, layer I, v2, Monaural
b5,bgr,lsb,xy .. file: MPEG ADTS, layer II, v1, Monaural
b6,bgr,msb,xy .. file: ddis/ddif
b2,bgr,msb,xy,prime .. file: MPEG ADTS, layer I, v2, 112 kbps, Monaural
b3,rgb,msb,xy,prime .. file: MPEG ADTS, layer I, v2, Monaural
b4,r,msb,xy,prime .. file: RDI Acoustic Doppler Current Profiler (ADCP)
b4,bgr,msb,xy,prime .. file: ddis/ddif
b5,r,msb,xy,prime .. file: MPEG ADTS, layer II, v1, 48 kHz, Monaural
b5p,r,msb,xy,prime .. file: RDI Acoustic Doppler Current Profiler (ADCP)
b5p,rgb,lsb,xy,prime .. file: MPEG ADTS, layer I, v2, Monaural
b6p,r,msb,xy,prime .. file: ddis/ddif
b6p,r,msb,xy,prime .. file: RDI Acoustic Doppler Current Profiler (ADCP)
b7,r,lsb,xy,prime .. file: AIX core file #ulldump 32-bit
b7p,r,msb,xy,prime .. file: RDI Acoustic Doppler Current Profiler (ADCP)
b7p,rgb,lsb,xy,prime .. file: MPEG ADTS, layer II, v1, Monaural
b4,bgr,msb,yx .. file: MPEG ADTS, layer I, v2, Monaural
b5,bgr,lsb,yx .. file: MPEG ADTS, layer II, v1, Monaural
b8,bgr,msb,yx .. file: ddis/ddif

```

Karena disini ga ada yang menarik, berarti data yang di sembunyikan itu bisa jadi di bit plane 1, karena biasanya bit plane 1 bisa jadi tempat data yang di sembunyikan jadi saya curiga kalau bagian flag nya berada di bit plane 1, jadi saya langsung mengekstrak zsteg -E karena ini kita mau mengekstrak bit plane 1 jadi parameter nya "b1, r, lsb, xy" mengapa saya ambil parameter ini karena saya ingin mengekstrak sebuah bit plane 1 dengan channel warna merah yang dalam parameter r, dan saya menggunakan extraction method dengan menggunakan lsb yang berarti bit paling terakhir dahulu, dan saya menggunakan parameter xy yang ini namanya pixel order karena saya menggunakan xy karena agar bisa membaca kiri ke kanan dan atas dan bawah

```

kali@juliuswijaya:~/mnt/c/users/juliu/downloads
$ zsteg -E b1,r,lsb,xy ninji.png > strings.txt

kali@juliuswijaya:~/mnt/c/users/juliu/downloads
$ strings strings.txt
d4r1_
21&m
Nizs5
rk5T
z5f2l
o*:7N'
Qu#D
{.:j?
t      ^G
uMU" |
V ~}
=yUp
EPHd
Phl.D
j?6k
xi'#
a|<?
zg5`
aw\!A
d sV
t74}
X'W0@
"iu;

```

Dan disini saya langsung buka file tersebut di bagian pertama itu ada bagian flag yang ke 2 yaitu `d4r1_`

FLAG : `TechtonicExpoCTF{ku3_l4p1s_d4r1_st3g4n0}`