# WRITE-UP CTF GEMASTIK 2022 KUALIFIKASI

30 Oktober 2022

## *anak kemaren sore*

*(IPB University)*


**Patar Isac Pardomuan (patsac)**

**Aulia Rochman (arai)**

**Muhammad Jundi Fathan (jedi)**

# Daftar Isi

# Forensic

## Traffic Enjoyer (500 pts)

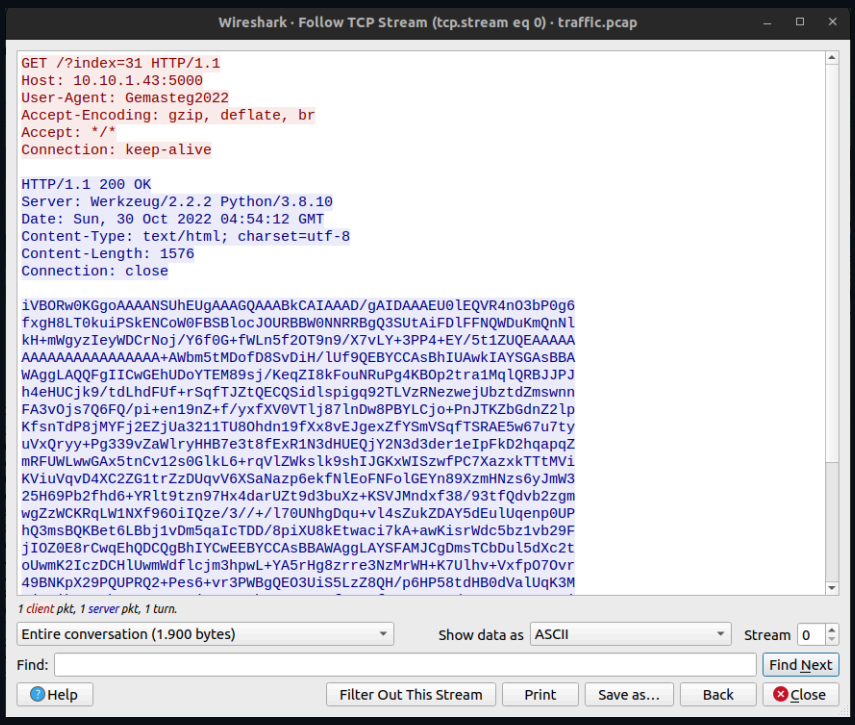
**Description**

P balap first blood

Diberikan sebuah file packet capture **traffic.pcap**.

**Solution**

Diberikan sebuah file packet-capture, Dengan menggunakan tools wireshark lakukan analisis terlebih dahulu terhadap TCP stream nya. Pada bagian analyze → Follow → TCP Stream terlihat encoding base64.




Dari base64 tersebut ketika di decode akan mendapatkan sebuah blob image png. Ketika melihat alur dari seluruh TCP stream data saya menyadari bahwa semua stream memiliki encoding base64 tersebut. Saya melakukan export HTTP object secara langsung terhadap http object atau stream dari HTTP.

```
→ traffic ls
'%3findex=0'   '%3findex=16'  '%3findex=23'  '%3findex=30'  '%3findex=38'  '%3findex=45'  '%3findex=8'
'%3findex=1'   '%3findex=17'  '%3findex=24'  '%3findex=31'  '%3findex=39'  '%3findex=46'  '%3findex=9'
'%3findex=10'  '%3findex=18'  '%3findex=25'  '%3findex=32'  '%3findex=4'   '%3findex=47'
'%3findex=11'  '%3findex=19'  '%3findex=26'  '%3findex=33'  '%3findex=40'  '%3findex=48'
'%3findex=12'  '%3findex=2'   '%3findex=27'  '%3findex=34'  '%3findex=41'  '%3findex=49'
```

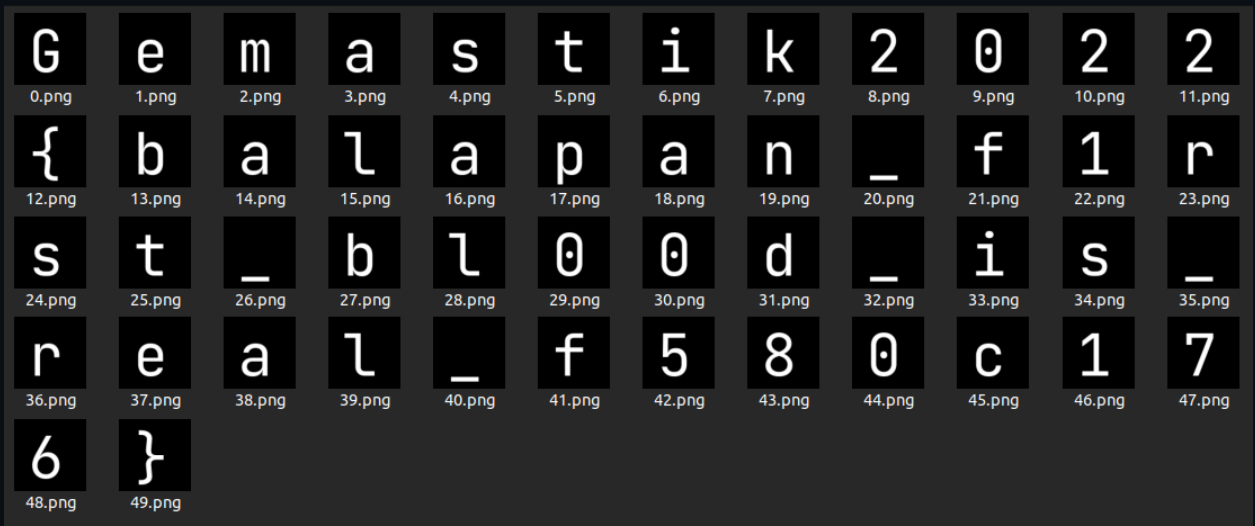
Semua hasil export tersebut merupakan encoding base64 dari setiap http object. Sehingga dapat dilakukan perulangan decode menggunakan  python os dengan memanfaatkan command

shell yaitu **“cat '%3findex=0' | base64 -d > 0.png”**. Sehingga saya membuat script sebagai berikut:

dc.py

```
import os


for i in range(50):
   cmd = f"cat '%3findex={i}' | base64 -d > {i}.png"
   os.system(cmd)
```

```
→ traffic python3 dc.py
→ traffic ls *.png
0.png   13.png  17.png  20.png  24.png  28.png  31.png  35.png  39.png  42.png  46.png  4.png  8.png
10.png  14.png  18.png  21.png  25.png  29.png  32.png  36.png  3.png   43.png  47.png  5.png  9.png
11.png  15.png  19.png  22.png  26.png  2.png   33.png  37.png  40.png  44.png  48.png  6.png
12.png  16.png  1.png   23.png  27.png  30.png  34.png  38.png  41.png  45.png  49.png  7.png
```




Hasil output akan mendapatkan file-file png yang ketika dilihat merupakan sebuah flag. Saya berfikir untuk menyambungkan semua gambar tersebut, namun menurut saya sepertinya akan sama saja hasilnya karena gambar yang tidak terlalu banyak. Lalu secara manual mengisi flag satu per satu menjadi flag yang lengkap.

**`Flag : Gemastik2022{balapan_f1rst_bl00d_is_real_f580c176}`**

# Cryptography

## Doublesteg (892 pts)

**Description**

Single STEG encryption is weak, how about double STEG encryption?

Diberikan file *chall.py* dan *flag.enc*.

*chall.py*

```python
#!/usr/bin/env python3
from Crypto.Cipher import AES
from Crypto.Hash import SHA256
from Crypto.Util.Padding import *
import random

FLAG = open("flag.png", "rb").read()
STEG = b"gemasteg"


def getrandsteg():
    x = list(STEG)
    random.shuffle(x)
    return bytes(x)


def encrypt(msg: bytes, key: bytes):
    key = SHA256.new(key).digest()
    iv = STEG * 2
    aes = AES.new(key, AES.MODE_CBC, iv)
    enc = aes.encrypt(msg)
    return enc


def double(msg: bytes, keys: list[bytes]):
    msg = pad(msg, AES.block_size)
    for key in keys:
        msg = encrypt(msg, key)
    return msg


def fwrite(filename: str, data: bytes):
    f = open(filename, "wb")
    f.write(data)
    f.close()


keys = [getrandsteg() for _ in range(2)]
```

```
fwrite("flag.enc", double(FLAG, keys))
```

Dari source *chall.py* dapat dilihat bahwa flag adalah file PNG dan dienkripsi dengan AES CBC dua kali dengan dua key berbeda. Untungnya, saya sudah pernah mengerjakan challenge serupa. Kita bisa menggunakan *Meet-in-the-middle attack* pada sistem seperti ini.

**Solution**

*Meet-in-the-middle attack* dapat dilakukan dengan melakukan bruteforce key untuk enkripsi *known plaintext*, dalam kasus disini adalah header PNG, dan disimpan ke dalam sebuah *dictionary* (enc_table) dengan hasil enkripsi sebagai *key dictionary* dan *encryption key* (key pertama) sebagai *value dictionary*-nya. Kemudian melakukan bruteforce key untuk dekripsi *ciphertext*, dalam kasus ini hanya block pertama saja, dan disimpan ke dalam sebuah *dictionary* (dec_table) dengan hasil dekripsi menjadi *key dictionary* dan *decryption key* (key kedua) sebagai *value dictionary*-nya. Setelah itu, kita mencari *intersection* atau *value dictionary key* yang sama dari kedua *dictionary* yang sudah dibentuk. Nah, jika sudah menemukan *key dictionary* yang sama, maka *encryption key* adalah *value* dari enc_table[*x*] dan *decryption key* adalah *value* dari dec_table[*x*], di mana *x* adalah *key dictionary* yang sama-sama ada di kedua *dictionary*.

Untuk generate semua kemungkinan key, saya membuat script agar jumlah keynya menjadi minimal. Berikut adalah scriptnya:

*key_gen.py*

```
#!/usr/bin/python3

from itertools import permutations

STEG = b"gemasteg"
ALL_K = permutations(STEG)
ALL_KEY = []
for k in ALL_K:
    if k not in ALL_KEY:
        ALL_KEY.append(k)

with open("all_key.py", "w") as f:
    f.write(f"{ALL_KEY = }")
```

Hasil dari permutasi key tersebut adalah sejumlah 10080 dan disimpan ke dalam *all_key.py*.

Hal ini cocok jika kita menggunakan rumus permutasi : $8! \div (2! * 2!) = 10080$.

```
>>> from all_key import ALL_KEY
>>> len(ALL_KEY)
10080
>>>
```

Selanjutnya setelah saya sudah generate key, tinggal melakukan *Meet-in-the-middle attack* seperti yang sudah saya jelaskan sebelumnya.

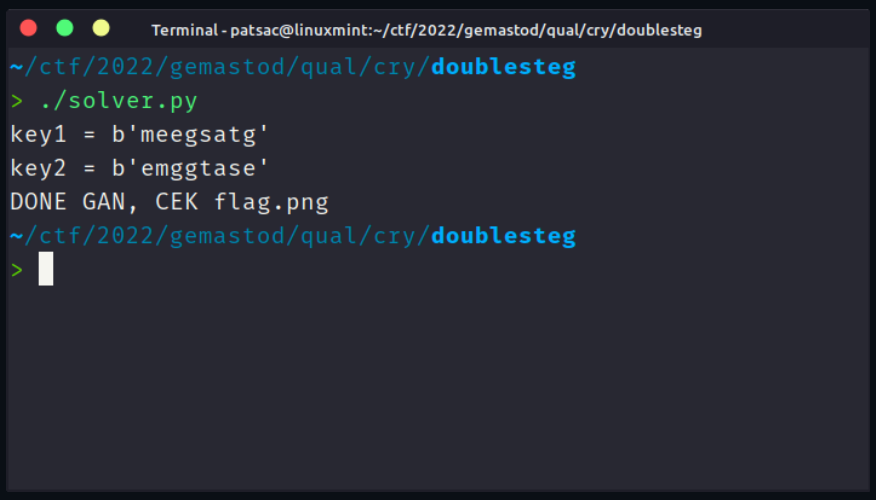
Berikut ini adalah full solvernya:

*solver.py*

```python
#!/usr/bin/python3
from Crypto.Hash import SHA256
from Crypto.Cipher import AES
from all_key import ALL_KEY

ENC = open("flag.enc", "rb").read()
STEG = b"gemasteg"
PLAIN = bytes.fromhex("89504E470D0A1A0A0000000D49484452")


def decrypt_cbc(msg: bytes, key: bytes):
    key = SHA256.new(key).digest()
    iv = STEG * 2
    aes = AES.new(key, AES.MODE_CBC, iv)
    plain = aes.decrypt(msg)
    return plain


def encrypt_cbc(msg: bytes, key: bytes):
    key = SHA256.new(key).digest()
    iv = STEG * 2
    aes = AES.new(key, AES.MODE_CBC, iv)
    enc = aes.encrypt(msg)
    return enc


def get_table(gas, isencrypt=False):
    table = {}
    msg = None
    if isencrypt:
        msg = PLAIN[:16]
    else:
        msg = ENC[:16]
    for key in ALL_KEY:
        custom = gas(msg, bytes(key))
        table[custom] = bytes(key)
    return table


def get_keys():
    block1 = ENC[:16]
    enc_table = get_table(encrypt_cbc, True)
    dec_table = get_table(decrypt_cbc)
    enc_table_set = set(enc_table.keys())
    dec_table_set = set(dec_table.keys())
    intersection = enc_table_set.intersection(dec_table_set).pop()
    key1 = enc_table[intersection]
    key2 = dec_table[intersection]
    return key1, key2
```

```python
def main():
    key1, key2 = get_keys()
    if key1 == None:
        print("KEYs NOT FOUND")
        exit(0)
    print(f"{key1 = }")
    print(f"{key2 = }")
    mid = decrypt_cbc(ENC, key2)
    flag = decrypt_cbc(mid, key1)

    with open("flag.png", "wb") as f:
        f.write(flag)
        f.close()
    print("DONE GAN, CEK flag.png")

    return 0


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

*Screenshot*



*flag.png*


`Flag : Gemastik2022{uji_nyali_encrypt_message_pakai_weak_key}`

## Relation (965 pts)

**Description**

Can you find the relation between these 2 encrypted files?

Hint :
If you have decrypted the PNG header but the image is still corrupted (e.g. invalid CRC), remember that the block size is 191 bytes (except the last block)

Diberikan *chall.py*, *flag1.enc*, *flag2.enc*, dan *key.pem*.

*chall.py*

```
#!/usr/bin/env python3
from Crypto.PublicKey import RSA

FLAG = open("flag.png", "rb").read()
NBIT = 1536
```

```python
def bytes_to_blocks(msg: bytes, size: int):
    return [msg[i : i + size] for i in range(0, len(msg), size)]


def blocks_to_bytes(blocks: list[bytes]):
    return b"".join(blocks)


def relation(pt: bytes, key):
    m1 = int.from_bytes(pt, "big")
    m2 = 7 * m1 + 7
    cts = []
    for m in [m1, m2]:
        assert m < key.n
        c = pow(m, key.e, key.n)
        ct = c.to_bytes(NBIT // 8, "big")
        cts.append(ct)
    return cts


def fwrite(filename: str, data: bytes):
    f = open(filename, "wb")
    f.write(data)
    f.close()


blocks = bytes_to_blocks(FLAG, (NBIT // 8) - 1)
rsa = RSA.generate(NBIT, e=7)

box1 = []
box2 = []

for block in blocks:
    cts = relation(block, rsa)
    box1.append(cts[0])
    box2.append(cts[1])

fwrite("flag1.enc", blocks_to_bytes(box1))
fwrite("flag2.enc", blocks_to_bytes(box2))
fwrite("key.pem", rsa.public_key().export_key())
```

Dari source *chall.py*, kita mengetahui bahwa flag adalah file PNG dan dienkripsi dengan RSA menghasilkan flag1.enc. Tetapi selain itu, flag juga dimodifikasi dengan persamaan linear kemudian dienkripsi dengan RSA menghasilkan flag2.enc. Setelah membacanya dan berpikir beberapa jam sambil googling. Saya akhirnya ingat saya pernah mengerjakan soal seperti ini juga. Kita bisa mendapatkan plaintext dengan melakukan *Franklin-Reiter Related Message Attack*.

**Solution**

Untuk solver *Franklin-Reiter Related Message Attack* saya menggunakan base solver dari salah satu sumber belajar cryptography terkenal, yaitu Crypton di halaman ini. Setelah memodifikasi sedikit untuk penyesuaian terhadap soal, akhirnya saya berhasil mendapatkan flag.png.

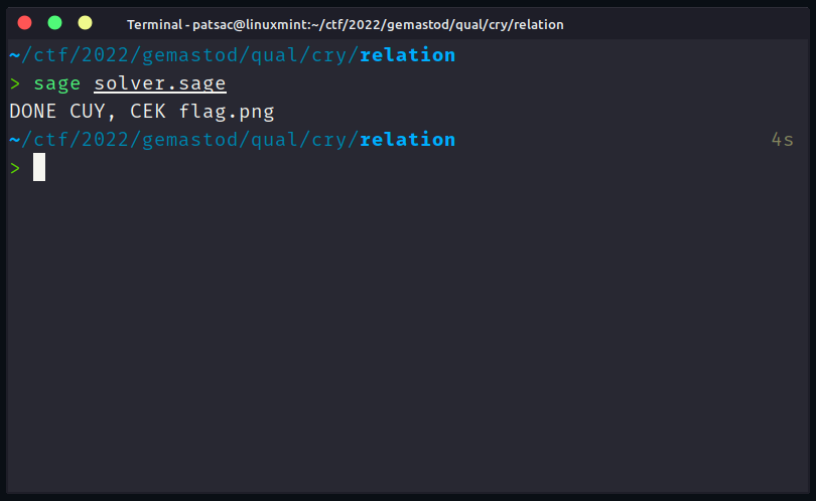
Berikut ini full solvernya:

*solver.py*

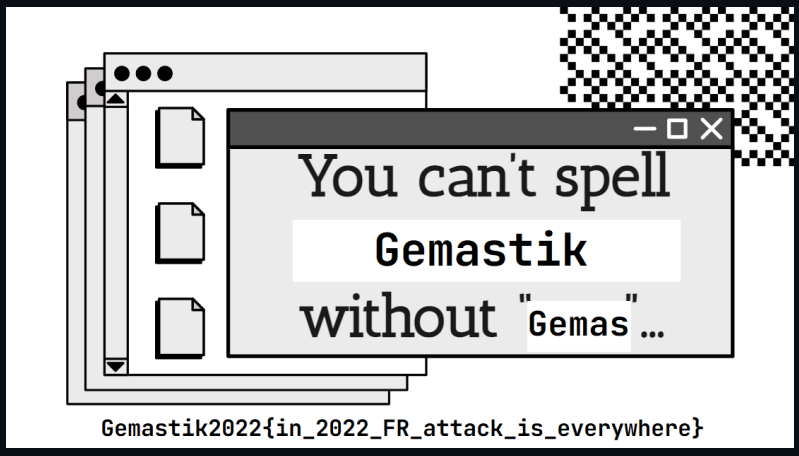
```
from Crypto.PublicKey import RSA
from Crypto.Util.number import bytes_to_long as b2l, long_to_bytes as l2b
from gmpy2 import iroot
from sage.all import *

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a.monic()

def franklinreiter(C1, C2, e, N, a, b):
    P.<X> = PolynomialRing(Zmod(N))
    g1 = (a*X + b)^e - C1
    g2 = X^e - C2
    result = -gcd(g1, g2).coefficients()[0]
    return hex(int(result))[2:]


def main():
    block = 192
    pubkey = open("key.pem", 'rb').read()
    pb = RSA.import_key(pubkey)
    enc1 = open("flag1.enc", 'rb').read()
    enc2 = open("flag2.enc", 'rb').read()
    png_bytes = b""
    for i in range(0, len(enc2), block):
        m = franklinreiter(b2l(enc2[i:i+block]), b2l(enc1[i:i+block]), pb.e, pb.n, 7, 7)
        if i < 67776:
            m = m.zfill(382)
        else:
            m = '0' + m
        png_bytes += bytes.fromhex(m)

    with open("flag.png", 'wb') as f:
        f.write(png_bytes)
        f.close()
    print("DONE CUY, CEK flag.png")


    return 0
```

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

*Screenshot*




*flag.png*




`Flag : Gemastik2022{in_2022_FR_attack_is_everywhere}`

# Reverse Engineering

## CodeJugling (500 pts)

**Description :**

"Find the flag!"

**Source :**

reversing-itu-mudah

Diberikan suatu file binary bernama *reversing-itu-mudah*. Ketika dicoba untuk dijalankan, ternyata dia meminta suatu input yang berupa flag dari soal ini, sehingga diketahui bahwa soal ini merupakan flag checker. Kemudian, file ini dibuka dengan menggunakan IDA

Ketika dibuka dan dicek fungsi main-nya, didapati bahwa flag merupakan suatu string dengan panjang 35 karakter dengan setiap karakternya dicek dengan menggunakan sebuah fungsi tersendiri

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  int v4; // [rsp+18h] [rbp-18h]
  int i; // [rsp+1Ch] [rbp-14h]

  if ( a1 == 2 )
  {
    sub_4014A0(a2[1], 0LL, a3);
    sub_4014E0(a2[1], 1LL);
    sub_401520(a2[1], 2LL);
    sub_401560(a2[1], 3LL);
    sub_4015A0(a2[1], 4LL);
    sub_4015E0(a2[1], 5LL);
    sub_401620(a2[1], 6LL);
    sub_401660(a2[1], 7LL);
    sub_4016A0(a2[1], 8LL);
    sub_4016E0(a2[1], 9LL);
    sub_401720(a2[1], 10LL);
    sub_401760(a2[1], 11LL);
    sub_4017A0(a2[1], 12LL);
    sub_4017E0(a2[1], 13LL);
```

```
  sub_401820(a2[1], 14LL);
  sub_401860(a2[1], 15LL);
  sub_4018A0(a2[1], 16LL);
  sub_4018E0(a2[1], 17LL);
  sub_401920(a2[1], 18LL);
  sub_401960(a2[1], 19LL);
  sub_4019A0(a2[1], 20LL);
  sub_4019E0(a2[1], 21LL);
  sub_401A20(a2[1], 22LL);
  sub_401A60(a2[1], 23LL);
  sub_401AA0(a2[1], 24LL);
  sub_401AE0(a2[1], 25LL);
  sub_401B20(a2[1], 26LL);
  sub_401B60(a2[1], 27LL);
  sub_401BA0(a2[1], 28LL);
  sub_401BE0(a2[1], 29LL);
  sub_401C20(a2[1], 30LL);
  sub_401C60(a2[1], 31LL);
  sub_401CA0(a2[1], 32LL);
  sub_401CE0(a2[1], 33LL);
  sub_401D20(a2[1], 34LL);
  v4 = 0;
  for ( i = 0; i < 35; ++i )
    v4 |= dword_404050[i];
  if ( strlen(a2[1]) != 35 )
    v4 = 1;
  if ( v4 )
    printf("Sorry, wrong flag\n");
  else
    printf("Congratulations, the flag is: %s\n", a2[1]);
}
else
{
  printf("Usage: %s flag\n", *a2);
}
```

```
  return 0LL;
}
```

Ketika dicek setiap fungsinya, terdapat perbandingan antara karakter tersebut secara utuh atau antara karakter tersebut setelah dilakukan operasi xor. Sehingga, bisa langsung di-reverse saja fungsi-fungsi tersebut untuk didapatkan flagnya

Solver yang saya buat adalah seperti berikut :

*solver.py*

```
karakter = [None]*35
karakter[0] = chr(171 ^ 236)
karakter[1] = chr(101)
karakter[2] = chr(109)
karakter[3] = chr(97)
karakter[4] = chr(31 ^ 108)
karakter[5] = chr(140 ^ 248)
karakter[6] = chr(49^88)
karakter[7] = chr(0x6f ^ 4)
karakter[8] = chr(0x37 ^ 5)
karakter[9] = chr(0xcd ^ 253)
karakter[10] = chr(0x3e ^ 12)
karakter[11] = chr(0xcc ^ 254)
karakter[12] = chr(0x70 ^ 11)
karakter[13] = chr(115)
karakter[14] = chr(0x24 ^ 80)
karakter[15] = chr(0x60 ^ 84)
karakter[16] = chr(0x10 ^ 37)
karakter[17] = chr(105)
karakter[18] = chr(0xc3 ^ 150)
karakter[19] = chr(110)
karakter[20] = chr(95)
karakter[21] = chr(77)
karakter[22] = chr(0x86 ^ 202)
karakter[23] = chr(0x80 ^ 199)
karakter[24] = chr(0xd8 ^ 135)
karakter[25] = chr(0x82 ^ 233)
karakter[26] = chr(0x27 ^ 23)
karakter[27] = chr(0x9b ^ 172)
karakter[28] = chr(0x93 ^ 242)
karakter[29] = chr(0x7a ^ 37)
```

```
karakter[30] = chr(98)
karakter[31] = chr(52)
karakter[32] = chr(114)
karakter[33] = chr(0xd1 ^ 132)
karakter[34] = chr(0xd ^ 112)
print(''.join(karakter))
```

```
jedi@DESKTOP-DTPA5CB:/mnt/d/CTF/gemastik/rev$ python3 solver.py
Gemastik2022{st45iUn_MLG_k07a_b4rU}
jedi@DESKTOP-DTPA5CB:/mnt/d/CTF/gemastik/rev$ ./reversing-itu-mudah Gemastik
2022{st45iUn_MLG_k07a_b4rU}
Congratulations, the flag is: Gemastik2022{st45iUn_MLG_k07a_b4rU}
jedi@DESKTOP-DTPA5CB:/mnt/d/CTF/gemastik/rev$
```

`Flag : Gemastik2022{st45iUn_MLG_k07a_b4rU}`

## Dino (500 pts)

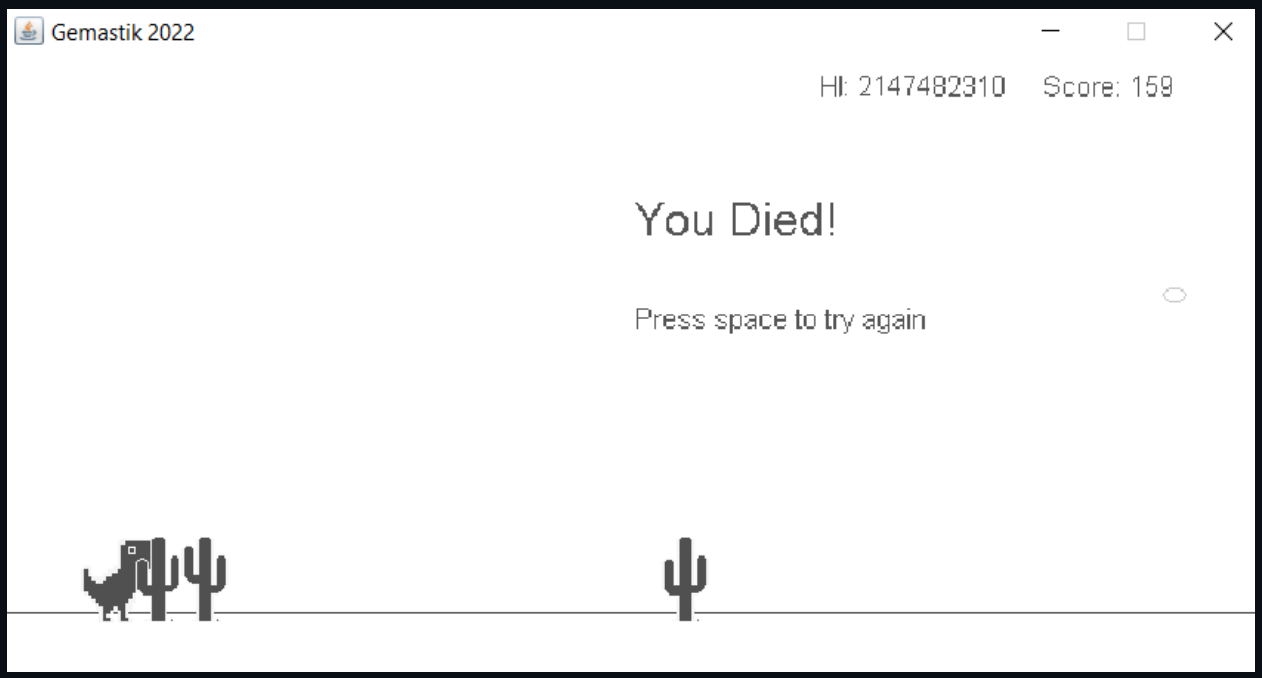
**Description :**

"Beat my highscore!"

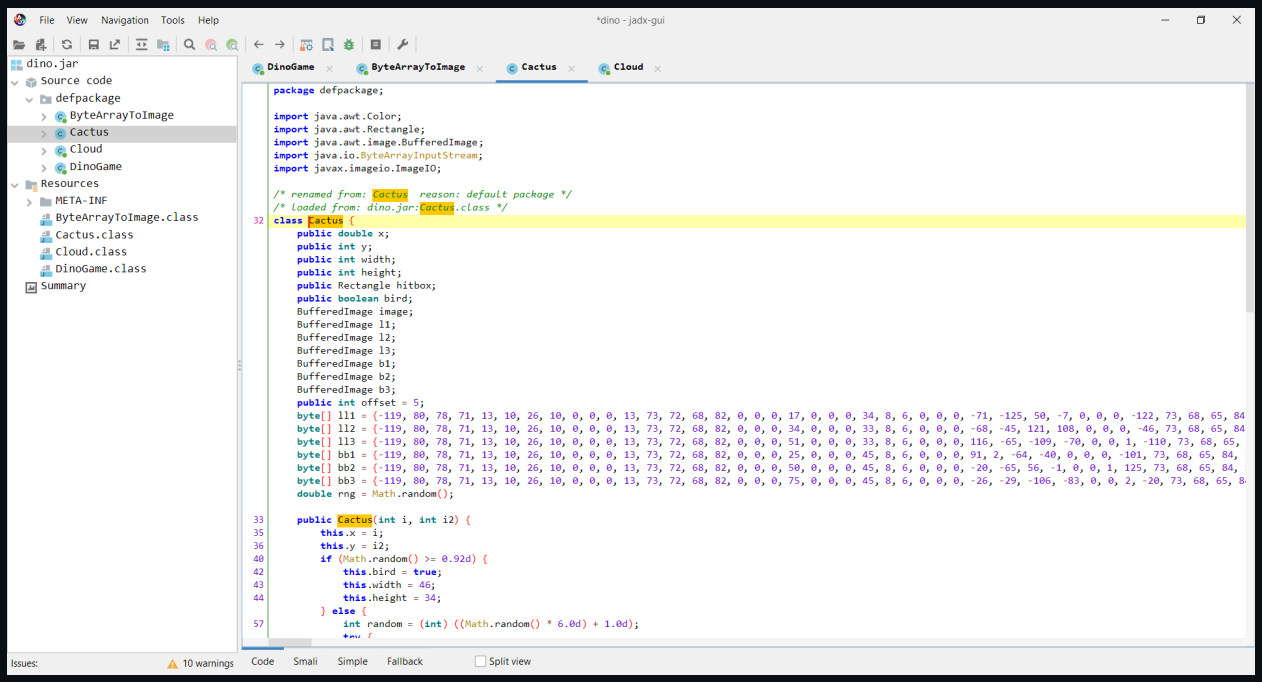
**Source :**

highscore.txt  dino.jar

Diberikan sebuah file jar dan sebuah file txt. Ketika file dino.jar di-run, ditampilkan game dino lost connection. Kita diminta untuk mengalahkan highscorenya, yaitu 2147482310, seperti yang tertera pada file highscore.txt



Kemudian, untuk mengecek file dino.jar lebih lanjut, digunakan jadx-gui. Diketahui terdapat 4 file source code, yaitu ByteArrayToImage.java, Cactus.java, Cloud.java, dan DinoGame.java. Pada file DinoGame.java




Pada file DinoGame.java, ada suatu bagian yang menarik, yaitu terdapat pemanggilan file highscore.txt. Ternyata, untuk highscore dari game tersebut diambil dari file highscore.txt. Namun, ada bagian yang cukup menarik, karena isi dari highscore.txt pada awalnya terdapat 2 buah angka, yaitu 2147482310 dan 21cb61a. Setelah dibaca-baca pada bagian berikut :

```
    private int ls() {
        gf();
        try {
            BufferedReader bufferedReader = new BufferedReader(new
FileReader("highscore.txt"));
            String readLine = bufferedReader.readLine();
            bufferedReader.close();
            String[] split = readLine.split(" ");
            int parseInt = Integer.parseInt(split[0]);
            this.csss = split[1];
            int rcr = rcr(parseInt);
            if (!Integer.toHexString(rcr).equals(this.csss)) {
                throw new Error("Invalid checksum");
            }
            this.ssss = rcr(rcr(rcr) ^ parseInt);
```

```
            return parseInt;
        } catch (Exception e) {
            System.out.println("Error loading highscore");
            System.exit(0);
            return 0;
        }
    }

    private int rcr(int i) {
        int i2;
        int i3 = -1;
        for (int i4 = 0; i4 < 4; i4++) {
            i3 ^= i >> (i4 * 8);
            for (int i5 = 0; i5 < 8; i5++) {
                if ((i3 & 1) == 1) {
                    i2 = (i3 >> 1) ^ (-306674912);
                } else {
                    i2 = i3 >> 1;
                }
                i3 = i2;
            }
        }
        return i3;
    }
```

ternyata angka kedua merupakan semacam checksum dari angka pertama. Sehingga, untuk mengubah nilai highscore, diperlukan juga bilangan checksum dari nilai highscore tersebut. Untuk mencarinya, bisa langsung kita run saja agar mendapat nilai checksumnya. Misalkan saya set nilai highscorenya menjadi 1, maka untuk mencari checksum dari 1 adalah seperti berikut :

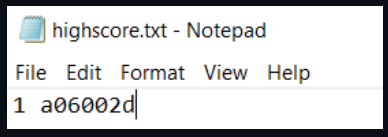
```
#include<bits/stdc++.h>
using namespace std;


int rcr(int i) {
        int i2;
        int i3 = -1;
        for (int i4 = 0; i4 < 4; i4++) {
```

```
            i3 ^= i >> (i4 * 8);
            for (int i5 = 0; i5 < 8; i5++) {
                if ((i3 & 1) == 1) {
                    i2 = (i3 >> 1) ^ (-306674912);
                } else {
                    i2 = i3 >> 1;
                }
                i3 = i2;
            }
        }
        return i3;
    }

int main(){
    cout << rcr(1) << endl;
}
```

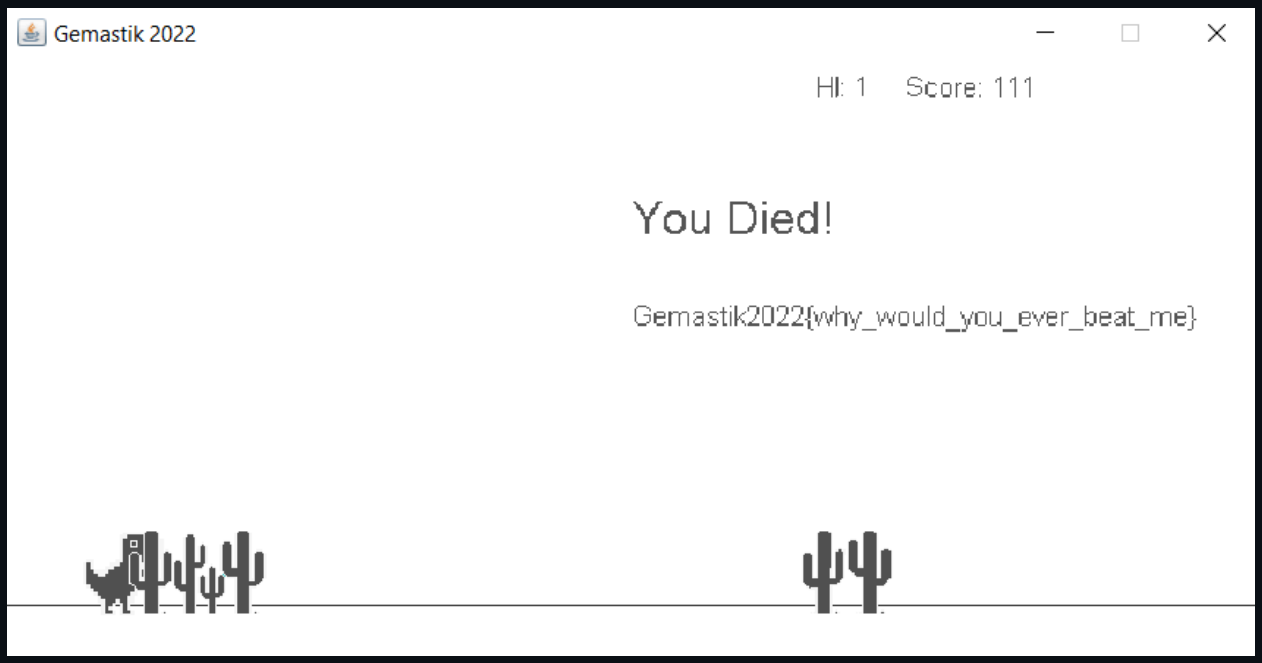
```
jedi@DESKTOP-DTPA5CB:/mnt/d/CTF/gemastik/rev$ g++ solv.cpp -o solv && ./solv
168165421
jedi@DESKTOP-DTPA5CB:/mnt/d/CTF/gemastik/rev$ printf "%x\n" 168165421
a06002d
jedi@DESKTOP-DTPA5CB:/mnt/d/CTF/gemastik/rev$
```

Karena bilangan kedua pada file highscore.txt asli menggunakan bilangan hex, maka kita perlu mengubah checksum milik kita menjadi bilangan hex juga

Kemudian, bisa kita ubah isi dari highscore.txt menjadi seperti berikut :




Dan kita jalankan kembali file dino.jar




**Flag : Gemastik2022{why_would_you_ever_beat_me}**