

WriteUp Gemastik CTF 2022

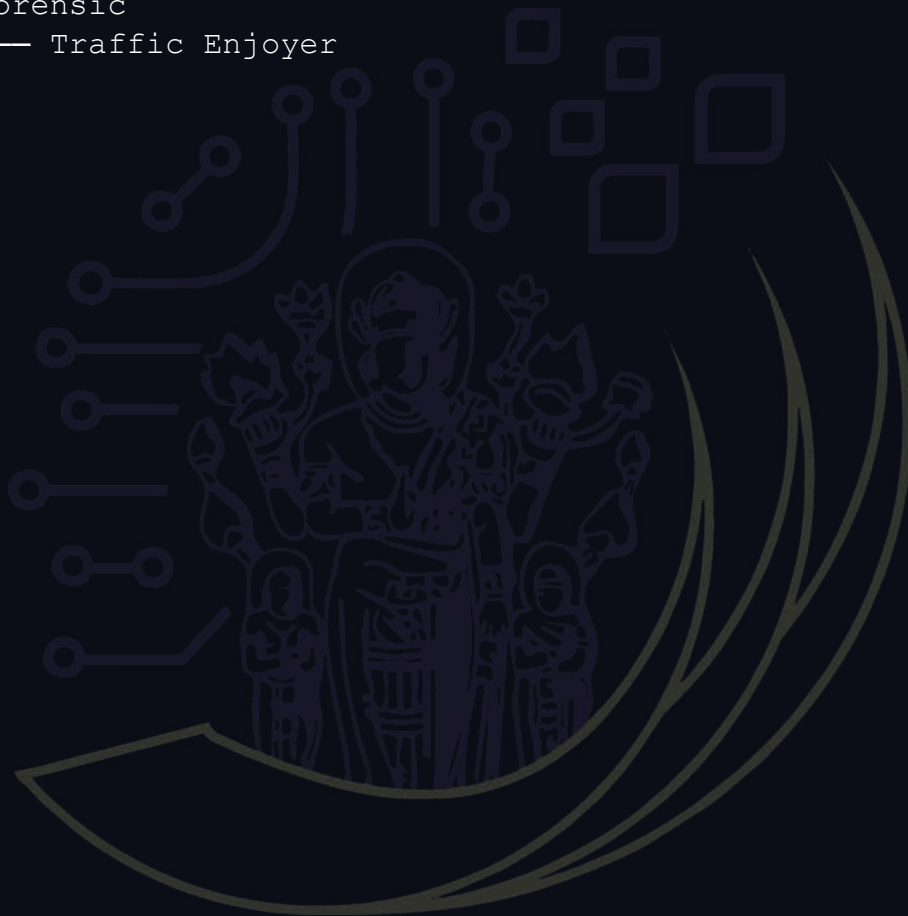
Team **AcRtf**



Table of Contents

Gemastik CTF 2022

- ├─ Cryptography
 - └─ Doublesteg
- ├─ Reverse Engineering
 - └─ CodeJugling
 - └─ Dino
 - └─ Rubyte
- └─ Forensic
 - └─ Traffic Enjoyer



Doublesteg

Cryptography

Deskripsi:

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

author - deomkicer#3362

Lampiran:

chall.py

```
#!/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))
```

Solusi:

Dapat dilihat pada proses enkripsi bahwa IV yang digunakan adalah `b"gemasteg" * 2` atau `b"gemasteggemasteg"`, dan *key* yang digunakan merupakan SHA256 dari `b"gemasteg"` yang susunan hurufnya diacak. Di sini, flag dienkripsi dua dengan sebuah *key*, kemudian hasil enkripsinya dienkripsi lagi menggunakan *key* yang lain.

Sebelumnya, kita dapat cek ada berapa kemungkinan *key* yang ada:

```
>>> from itertools import permutations
>>> len(set(permutations('gemasteg')))
10080
```

Dapat dilihat bahwa hanya terdapat 10080 ($8!/(2! * 2!)$) *key* yang mungkin.

Selain itu, kita juga bisa lihat pada potongan kode berikut:

```
6
7 FLAG = open("flag.png", "rb").read()
8 STEG = b"gemasteg"
9
```

Dapat dilihat bahwa flag merupakan gambar PNG, sehingga kita memiliki informasi mengenai blok pertama (16 *byte* pertama) dari *plaintext*, yaitu *signature* dari PNG ditambah *byte* awal *header* dari PNG, yaitu (dalam hex):

```
8950 4e47 0d0a 1a0a 0000 000d 4948 4452
```

Oleh karena itu, kita dapat melakukan *bruteforce* terhadap *key* dengan memanfaatkan Meet in the Middle *attack*, dengan menggunakan *known plaintext* di atas. MITM dilakukan dengan melakukan *pre-compute* enkripsi terhadap *known plaintext*, kemudian

melakukan *pre-compute* dekripsi terhadap *ciphertext*, kemudian membandingkan blok pertama dan mendapatkan potongan (*intersection*) dari kedua hasil tersebut. Dari perpotongan tersebut, bisa didapatkan *key* pertama dan *key* kedua.

solver.py

```
from itertools import permutations
from Crypto.Cipher import AES
from Crypto.Hash import SHA256
from Crypto.Util.Padding import *

STEG = b"gemasteg"
KEYS = []
IV = STEG * 2

def gen_keys():
    global KEYS
    keys = set(permutations(STEG))
    KEYS = [SHA256.new(bytes(i)).digest() for i in keys]

def encrypt(msg, key):
    global IV
    aes = AES.new(key, AES.MODE_CBC, IV)
    enc = aes.encrypt(msg)
    return enc

def decrypt(enc, key):
    global IV
    aes = AES.new(key, AES.MODE_CBC, IV)
    msg = aes.decrypt(enc)
    return msg

def main():
    gen_keys()
    enc = open('flag.enc', 'rb').read()

    pt =
pad(bytes.fromhex('89504e470d0a1a0a0000000d49484452'),
AES.block_size)
    ct = enc[:32]

    pt_map = {}
    ct_map = {}
```

```
for key in KEYS:
    enc_brute = encrypt(pt, key)[:16]
    pt_map[enc_brute] = key
    dec_brute = decrypt(ct, key)[:16]
    ct_map[dec_brute] = key

pt_keys = set(pt_map.keys())
ct_keys = set(ct_map.keys())
middle = list(pt_keys.intersection(ct_keys))[0]
pt_key = pt_map[middle]
ct_key = ct_map[middle]

msg_mid = decrypt(enc, ct_key)
msg = decrypt(msg_mid, pt_key)

with open('out.png', 'wb') as f:
    f.write(msg)

main()
```

Berikut adalah *output* gambar yang dihasilkan.



Flag: Gemastik2022{uji_nyali_encrypt_message_pakai_weak_key}

CodeJugling

Reverse Engineering

Deskripsi:

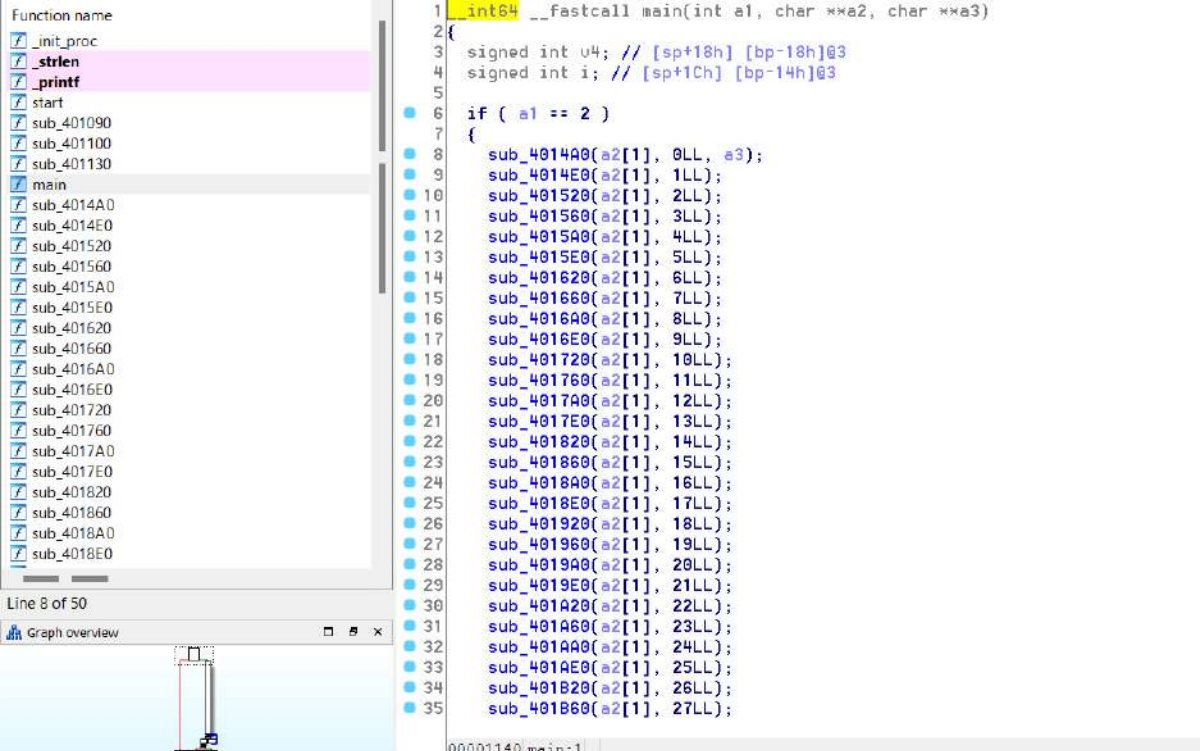
Find the flag!

Lampiran:

<binary file: reversing-itu-mudah>

Solusi:

Berikut adalah hasil *decompile* menggunakan IDA Pro.



```
1  __int64 __fastcall main(int a1, char **a2, char **a3)
2  {
3      signed int u4; // [sp+18h] [bp-18h]@3
4      signed int i; // [sp+1Ch] [bp-14h]@3
5
6      if ( a1 == 2 )
7      {
8          sub_4014A0(a2[1], 0LL, a3);
9          sub_4014E0(a2[1], 1LL);
10         sub_401520(a2[1], 2LL);
11         sub_401560(a2[1], 3LL);
12         sub_4015A0(a2[1], 4LL);
13         sub_4015E0(a2[1], 5LL);
14         sub_401620(a2[1], 6LL);
15         sub_401660(a2[1], 7LL);
16         sub_4016A0(a2[1], 8LL);
17         sub_4016E0(a2[1], 9LL);
18         sub_401720(a2[1], 10LL);
19         sub_401760(a2[1], 11LL);
20         sub_4017A0(a2[1], 12LL);
21         sub_4017E0(a2[1], 13LL);
22         sub_401820(a2[1], 14LL);
23         sub_401860(a2[1], 15LL);
24         sub_4018A0(a2[1], 16LL);
25         sub_4018E0(a2[1], 17LL);
26         sub_401920(a2[1], 18LL);
27         sub_401960(a2[1], 19LL);
28         sub_4019A0(a2[1], 20LL);
29         sub_4019E0(a2[1], 21LL);
30         sub_401A20(a2[1], 22LL);
31         sub_401A60(a2[1], 23LL);
32         sub_401AA0(a2[1], 24LL);
33         sub_401AE0(a2[1], 25LL);
34         sub_401B20(a2[1], 26LL);
35         sub_401B60(a2[1], 27LL);
```

Terlihat bahwa banyak *function calls* yang dilakukan. Setiap fungsi yang dipanggil memiliki struktur kode yang mirip, seperti berikut.

```

1  __int64 __fastcall sub_4014A0(__int64 a1, int a2)
2  {
3      __int64 result; // rax@1
4
5      result = a2;
6      dword_404050[a2] = (*(__BYTE *))(a1 + a2) ^ 0xEC) != 171;
7      return result;
8  }

```

Pada dasarnya, setiap *function call* melakukan pengecekan untuk setiap huruf pada *flag*, dengan melakukan fungsi XOR dan membandingkannya dengan sebuah nilai. Seperti contoh pada gambar di atas, fungsi tersebut dipanggil dengan variabel a1 merupakan *address* dari *flag*, dan variabel a2 bernilai 0. Jadi, fungsi diatas akan mengecek apakah nilai *flag* indeks ke-0 jika di-XOR dengan 0xEC akan menghasilkan 171. Untuk mendapatkan karakternya, cukup dilakukan XOR antara 171 dan 0xEC.

```
>>> chr(171 ^ 0xEC)
'G'
```

Didapatkan bahwa huruf pertama *flag* adalah G. Lakukan untuk setiap *function call*, maka *flag* pun akan didapatkan.

Flag: Gemastik2022{st45iUn_MLG_k07a_b4rU}

Dino

Reverse Engineering

Deskripsi:

```
Beat my highscore!  
author - vidner#6838
```

Lampiran:

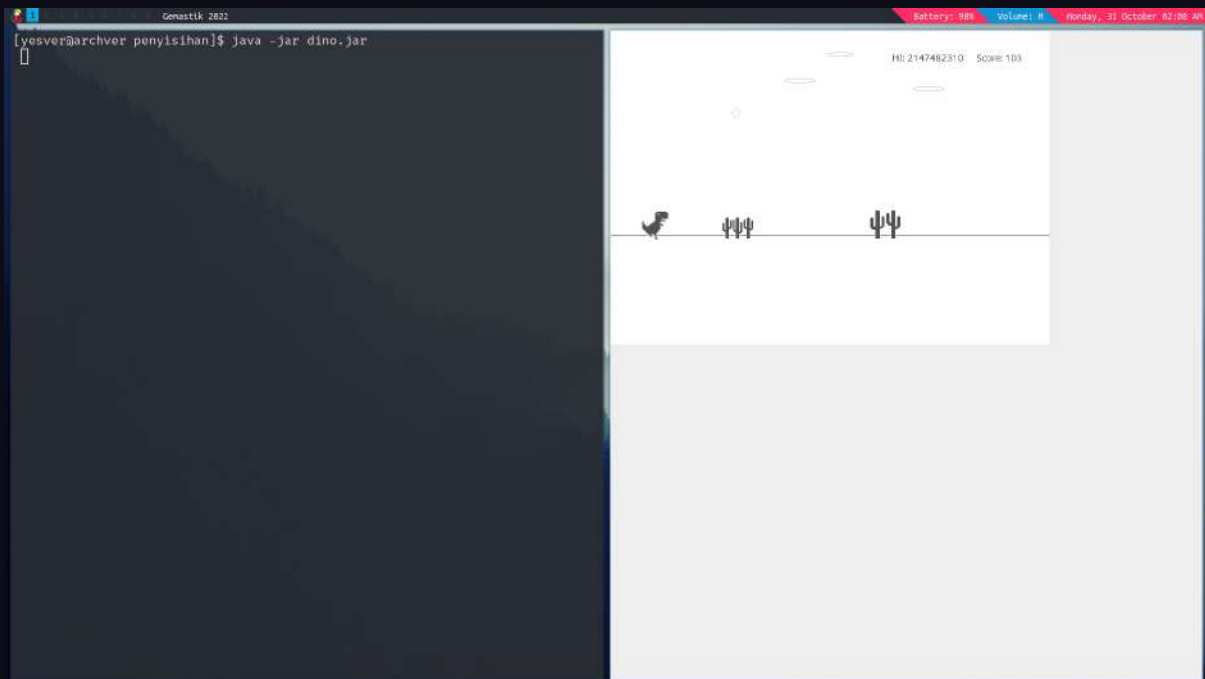
```
highscore.txt  
2147482310 21cb61a
```

```
<JAR file: dino.jar>
```

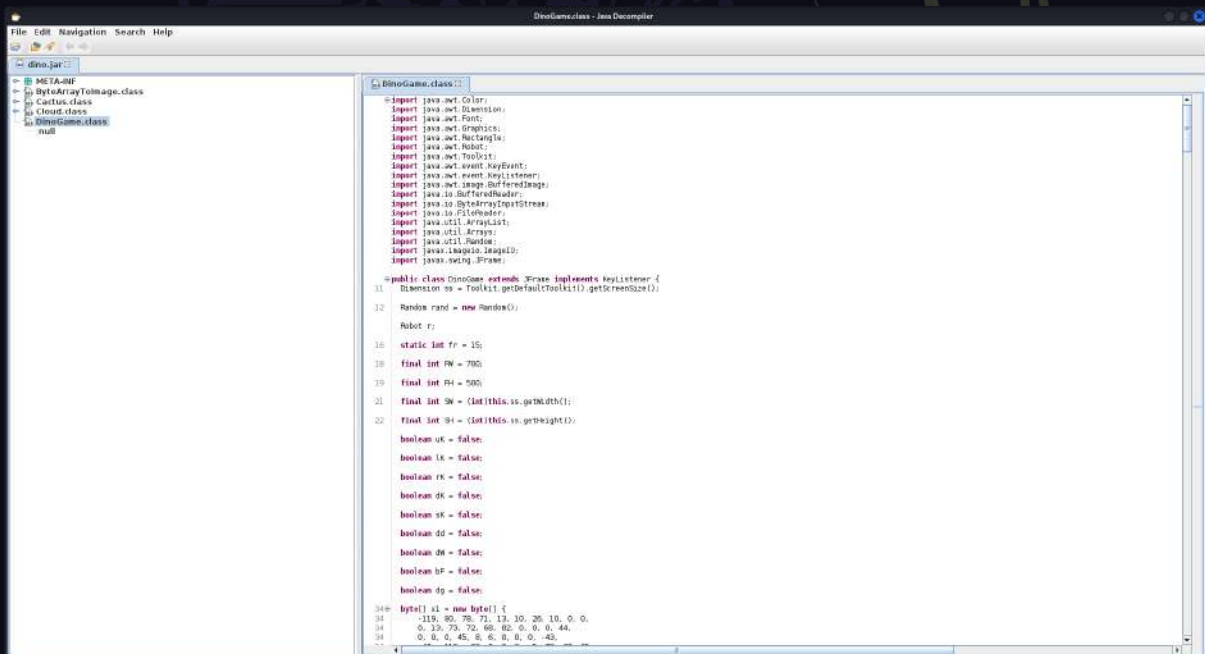
Solusi:

Diberikan sebuah file bernama dino.jar dan highscore.txt. File highscore berisi dua hal yaitu angka dan byte atau hex. File jar tersebut dapat dijalankan dengan menggunakan perintah `java -jar dino.jar` dan berisi permainan dino. Pada awal permainan, terdapat sebuah high score yang sangat tinggi. Sesuai dengan deskripsi soal, kita harus mengalahkan score tersebut untuk menang.

db



Setelah menjalankan program tersebut, dilakukan analisis pada kode program dengan menggunakan jd-gui.



Setelah melakukan analisis pada kode-kode tersebut, terdapat sebuah fungsi bernama rcr dan ls yang berkaitan dengan *scoring* pada game.

```

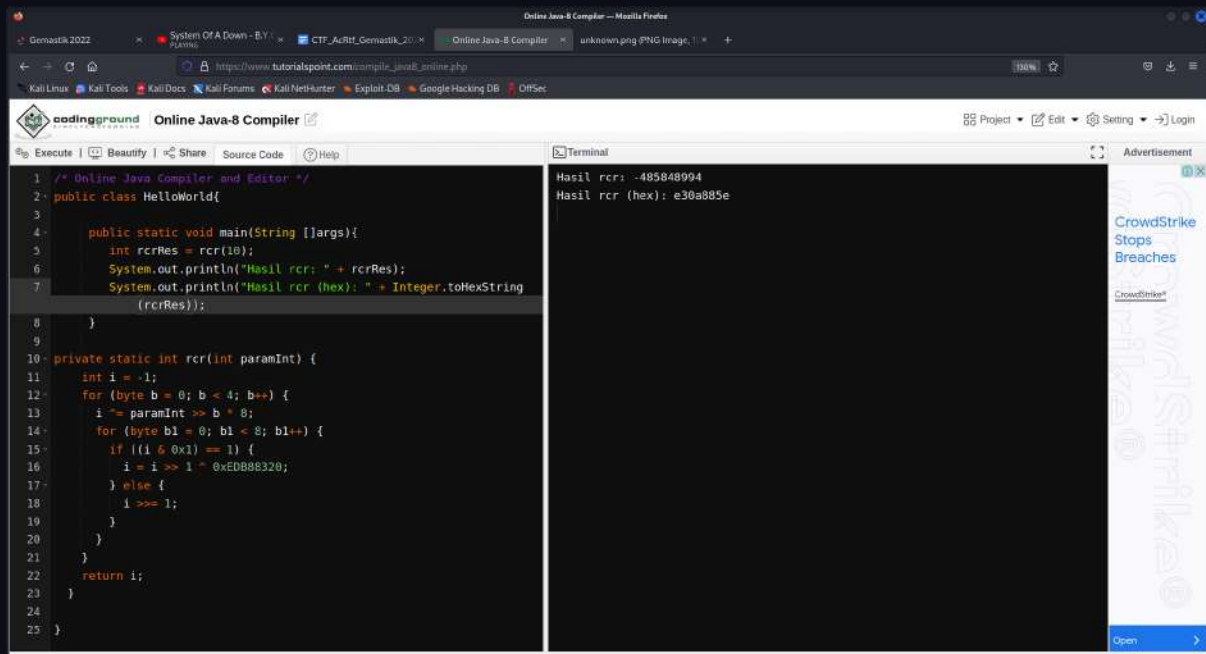
111 private int rcr(int paramInt) {
112     int j = this.x0[0] ^ this.x1[1] ^ this.x2[2] ^ this.x3[3] ^ this.x4[4] ^ this.x5[5] ^ this.x6[6] ^ this.x7[7];
113     this.ssss = this.ssss;
114 }
115
116 private int ls() {
117     gf();
118     try {
119         BufferedReader bufferedReader = new BufferedReader(new FileReader("highscore.txt"));
120         String str = bufferedReader.readLine();
121         bufferedReader.close();
122         String[] arrayOfString = str.split(" ");
123         int i = Integer.parseInt(arrayOfString[0]);
124         this.csss = arrayOfString[1];
125         int j = rcr(i);
126         if (!Integer.toHexString(j).equals(this.csss))
127             throw new Error("Invalid checksum");
128         this.ssss = rcr(rcr(j) ^ i);
129         return i;
130     } catch (Exception exception) {
131         System.out.println("Error loading highscore");
132         System.exit(0);
133     }
134 }
135
136 private int rcr(int paramInt) {
137     int i = 1;
138     for (byte b = 0; b < 4; b++) {
139         i = paramInt >> b * 8;
140         for (byte b1 = 0; b1 < 8; b1++) {
141             if ((i & 0x1) == 1) {
142                 i = i >> 1 ^ 0xEDB88320;
143             } else {
144                 i >>= 1;
145             }
146         }
147     }
148     return i;
149 }
150
151 public DinoGame(String paramString) {
152     super(paramString);
153     addKeyListener(this);
154 }

```

Dapat dilihat bahwa pada fungsi tersebut akan mengambil data dari file highscore.txt dan dimasukkan ke dalam array. Array index 0 berisi int angka dan index 1 berisi string hex. Dalam fungsi ls, terdapat pemanggilan fungsi rcr dengan menggunakan parameter dari array indeks 0 dan kemudian dimasukkan ke dalam variabel j. Setelah memanggil fungsi tersebut, dilakukan pengubahan variabel j ke hex string dan pengecekan apakah variabel j memiliki nilai yang sama dengan array index 1 yang berupa string hex atau tidak.

Berdasarkan analisis di atas, kita dapat merubah atau memperkecil *high score* dari permainan dengan syarat nilai dari skor sama dengan string hex setelah menggunakan fungsi rcr dan merubahnya ke dalam hex. Oleh karena itu, perubahan *high score* dilakukan dengan menggunakan parameter nilai yang kecil (10) pada fungsi rcr.

db



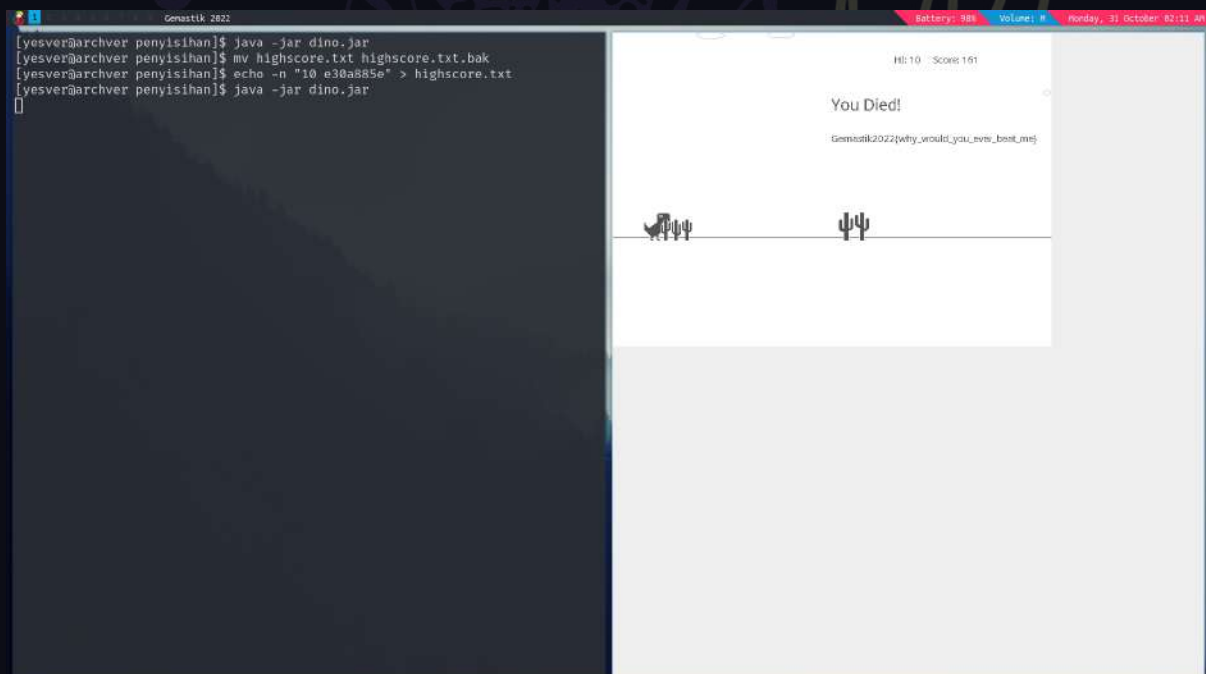
The screenshot shows a web browser window with the 'Online Java-8 Compiler' interface. The source code on the left is as follows:

```
1 // Online Java Compiler and Editor //
2 public class HelloWorld{
3
4     public static void main(String []args){
5         int rcrRes = rcr(10);
6         System.out.println("Hasil rcr: " + rcrRes);
7         System.out.println("Hasil rcr (hex): " + Integer.toHexString
8             (rcrRes));
9     }
10
11 private static int rcr(int paramInt) {
12     int i = -1;
13     for (byte b = 0; b < 4; b++) {
14         i ^= paramInt >> b * 8;
15         for (byte b1 = 0; b1 < 8; b1++) {
16             if ((i & 0x1) == 1) {
17                 i = i >> 1 ^ 0xEDB88320;
18             } else {
19                 i >>= 1;
20             }
21         }
22     }
23     return i;
24 }
25 }
```

The terminal output on the right shows:

```
Hasil rcr: -485848994
Hasil rcr (hex): e30a885e
```

Dengan menggunakan parameter 10 pada fungsi rcr, dihasilkan hex string e30a885e. Setelah mendapatkan hasil tersebut, dilakukan perubahan pada highscore.txt dengan mengganti isinya menjadi 10 dan e30a885e. Setelah merubah isi file tersebut, program kembali dijalankan dan *high score* telah berhasil diubah menjadi 10. Setelah skor melebihi 10, program mengeluarkan flag pada permainan tersebut.



Flag: Gemastik2022{why_would_you_ever_beat_me}

Rubyte

Reverse Engineering

Deskripsi:

Hope you find the hidden gem!

author - vidner#6838

Lampiran:

byte.txt

```
== disasm: #<ISeq:<compiled>@<compiled>:1 (1,0)-(1,99)>
(catch: FALSE)
0000 putself
( 1)[Li]
0001 opt_getinlinecache      8, <is:0>
0004 getconstant             :File
0006 opt_setinlinecache     <is:0>
0008 putstring               "flag"
0010 opt_send_without_block  <callinfo!mid:read, argc:1,
ARGS_SIMPLE>, <callcache>
0013 putstring               "H*"
0015 opt_send_without_block  <callinfo!mid:unpack,
argc:1, ARGS_SIMPLE>, <callcache>
0018 putobject_INT2FIX_0_
0019 opt_aref                <callinfo!mid:[], argc:1,
ARGS_SIMPLE>, <callcache>
0022 putobject              16
0024 opt_send_without_block  <callinfo!mid:to_i, argc:1,
ARGS_SIMPLE>, <callcache>
0027 opt_getinlinecache     34, <is:1>
0030 getconstant             :File
0032 opt_setinlinecache     <is:1>
0034 putstring               "flag"
0036 opt_send_without_block  <callinfo!mid:read, argc:1,
ARGS_SIMPLE>, <callcache>
0039 putstring               "H*"
0041 opt_send_without_block  <callinfo!mid:unpack,
argc:1, ARGS_SIMPLE>, <callcache>
0044 putobject_INT2FIX_0_
0045 opt_aref                <callinfo!mid:[], argc:1,
ARGS_SIMPLE>, <callcache>
```

db

```
0048 putobject 16
0050 opt_send_without_block <callinfo!mid:to_i, argc:1,
ARGS_SIMPLE>, <callcache>
0053 putobject_INT2FIX_1_
0054 opt_send_without_block <callinfo!mid:>>, argc:1,
ARGS_SIMPLE>, <callcache>
0057 opt_send_without_block <callinfo!mid:^, argc:1,
ARGS_SIMPLE>, <callcache>
0060 opt_send_without_block <callinfo!mid:puts, argc:1,
FCALL|ARGS_SIMPLE>, <callcache>
0063 leave
```

output.txt

```
2153997634379939228572579385071835718990334739880998312895779
21701237839559370177126393638370659139
```

Solusi:

Berikut adalah hasil analisis dari potongan-potongan *bytecode*.

```
0004 getconstant :File
0006 opt_setinlinecache <is:0>
0008 putstring "flag"
0010 opt_send_without_block <callinfo!mid:read, argc:1, ARGS_SIMPLE>, <callcache>
```

Kode di atas akan membaca isi dari *file* bernama "flag".

```
0013 putstring "H*"
0015 opt_send_without_block <callinfo!mid:unpack, argc:1, ARGS_SIMPLE>, <callcache>
```

Kode di atas akan mengubah isi dari *file* "flag" ke dalam bentuk hex.

```
0018 putobject_INT2FIX_0_
0019 opt_aref <callinfo!mid:[], argc:1, ARGS_SIMPLE>, <callcache>
0022 putobject 16
0024 opt_send_without_block <callinfo!mid:to_i, argc:1, ARGS_SIMPLE>, <callcache>
```

Kode di atas akan mengubah hex sebelumnya ke dalam sebuah bilangan *integer*.

```
0053 putobject_INT2FIX_1_
0054 opt_send_without_block <callinfo!mid:>>, argc:1, ARGS_SIMPLE>, <callcache>
```

Kode di atas akan melakukan *right bit-shift* sebanyak 1.

db

```
0057 opt_send_without_block      <callinfo!mid:^, argc:1, ARGS_SIMPLE>, <callcache>
```

Kode di atas akan melakukan XOR antara kedua buah *operand*, yaitu kedua kode berikut.

```
0004 getconstant                :File
0006 opt_setinlinecache          <is:0>
0008 putstring                    "flag"
0010 opt_send_without_block      <callinfo!mid:read, argc:1, ARGS_SIMPLE>, <callcache>
0013 putstring                    "H*"
0015 opt_send_without_block      <callinfo!mid:unpack, argc:1, ARGS_SIMPLE>, <callcache>
0018 putobject_INT2FIX_0_
0019 opt_aref                    <callinfo!mid:[], argc:1, ARGS_SIMPLE>, <callcache>
0022 putobject                    16
0024 opt_send_without_block      <callinfo!mid:to_i, argc:1, ARGS_SIMPLE>, <callcache>
```

```
0030 getconstant                :File
0032 opt_setinlinecache          <is:1>
0034 putstring                    "flag"
0036 opt_send_without_block      <callinfo!mid:read, argc:1, ARGS_SIMPLE>, <callcache>
0039 putstring                    "H*"
0041 opt_send_without_block      <callinfo!mid:unpack, argc:1, ARGS_SIMPLE>, <callcache>
0044 putobject_INT2FIX_0_
0045 opt_aref                    <callinfo!mid:[], argc:1, ARGS_SIMPLE>, <callcache>
0048 putobject                    16
0050 opt_send_without_block      <callinfo!mid:to_i, argc:1, ARGS_SIMPLE>, <callcache>
0053 putobject_INT2FIX_1_
0054 opt_send_without_block      <callinfo!mid:>>, argc:1, ARGS_SIMPLE>, <callcache>
```

Kurang lebih, kode awal program Ruby adalah sebagai berikut.

```
(inlandsche@kali)-[~]
$ cat code.rb
puts (read("flag").unpack("H*")[0].to_i(16))^(read("flag").unpack("H*")[0].to_i(16) >> 1)

(inlandsche@kali)-[~]
$ ruby --dump=insns code.rb
= disasm: #<ISeq:<main>@code.rb:1 (1,0)-(1,89)> (catch: FALSE)
0000 putself                      ( 1)[Li]
0001 putself
0002 putstring                    "flag"
0004 opt_send_without_block      <calldata!mid:read, argc:1, FCALL|ARGS_SIMPLE>
0006 putstring                    "H*"
0008 opt_send_without_block      <calldata!mid:unpack, argc:1, ARGS_SIMPLE>
0010 putobject_INT2FIX_0_
0011 opt_aref                    <calldata!mid:[], argc:1, ARGS_SIMPLE>
0013 putobject                    16
0015 opt_send_without_block      <calldata!mid:to_i, argc:1, ARGS_SIMPLE>
0017 putself
0018 putstring                    "flag"
0020 opt_send_without_block      <calldata!mid:read, argc:1, FCALL|ARGS_SIMPLE>
0022 putstring                    "H*"
0024 opt_send_without_block      <calldata!mid:unpack, argc:1, ARGS_SIMPLE>
0026 putobject_INT2FIX_0_
0027 opt_aref                    <calldata!mid:[], argc:1, ARGS_SIMPLE>
0029 putobject                    16
0031 opt_send_without_block      <calldata!mid:to_i, argc:1, ARGS_SIMPLE>
0033 putobject_INT2FIX_1_
0034 opt_send_without_block      <calldata!mid:>>, argc:1, ARGS_SIMPLE>
0036 opt_send_without_block      <calldata!mid:^, argc:1, ARGS_SIMPLE>
0038 opt_send_without_block      <calldata!mid:puts, argc:1, FCALL|ARGS_SIMPLE>
0040 leave
```

Dapat disimpulkan, bahwa kode melakukan enkripsi pada *flag* dengan mengubahnya ke *integer*, kemudian melakukan XOR antara *flag* dengan *flag* yang telah di *right bit-shift* sebanyak 1. Untuk mengembalikannya, di sini *solver* melakukan *brute-force* terhadap setiap karakter pada *flag*. Untuk setiap percobaan, akan dilakukan enkripsi pada *brute-forced flag*, dan jika hasilnya (yang telah di-convert ke hex) merupakan *prefix* dari output.txt yang juga telah di-convert ke hex, maka karakter yang di-bruteforce merupakan bagian dari *flag*.

```
solver.py
```

```
from Crypto.Util.number import *
from string import printable

enc_file = int(open('output.txt').read())
enc = long_to_bytes(enc_file)
enc_hex = hex(enc_file)

flag = b''
while not flag.endswith(b'}'):
    for i in printable:
        brute = flag + i.encode()
        brute_int = bytes_to_long(brute)
        enc_brute = brute_int ^ (brute_int >> 1)
        if enc_hex.startswith(hex(enc_brute)):
            flag = brute
            break

print(flag.decode())
```

```
rubyte> python .\solver.py
Gemastik2022{i_still_remember_30_october}
rubyte> |
```

Flag: Gemastik2022{i_still_remember_30_october}

Traffic Enjoyer

Forensic

Deskripsi:

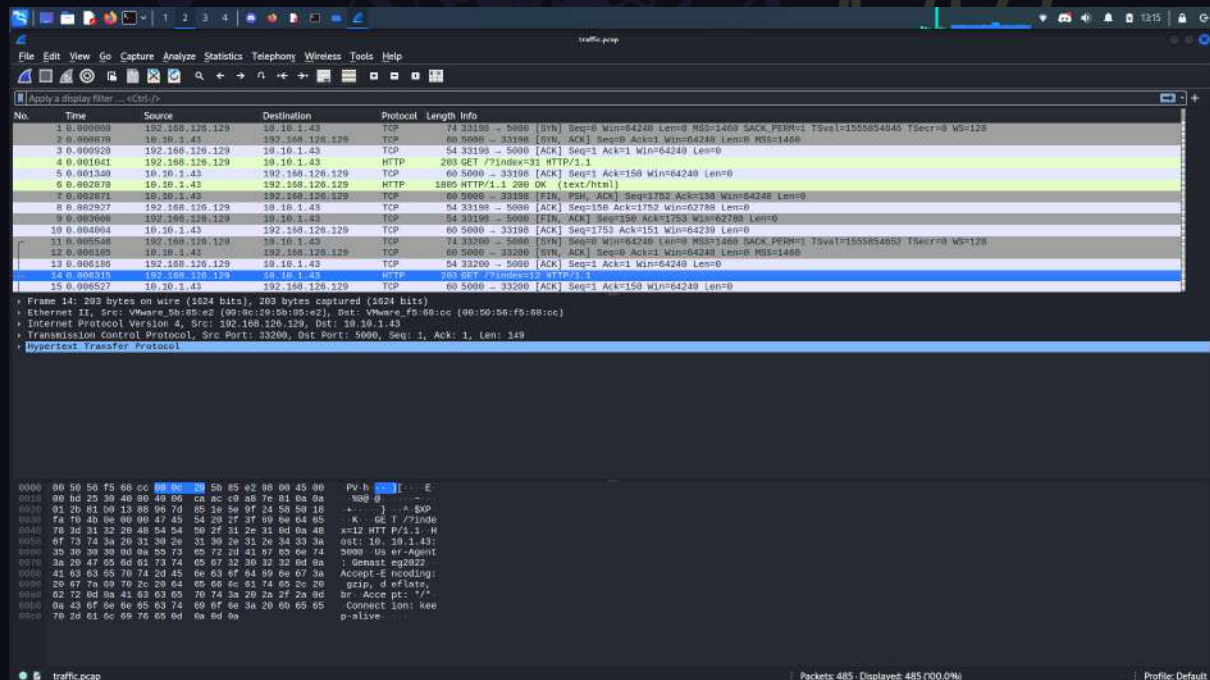
P balap first blood
author - deomkicer#3362

Lampiran:

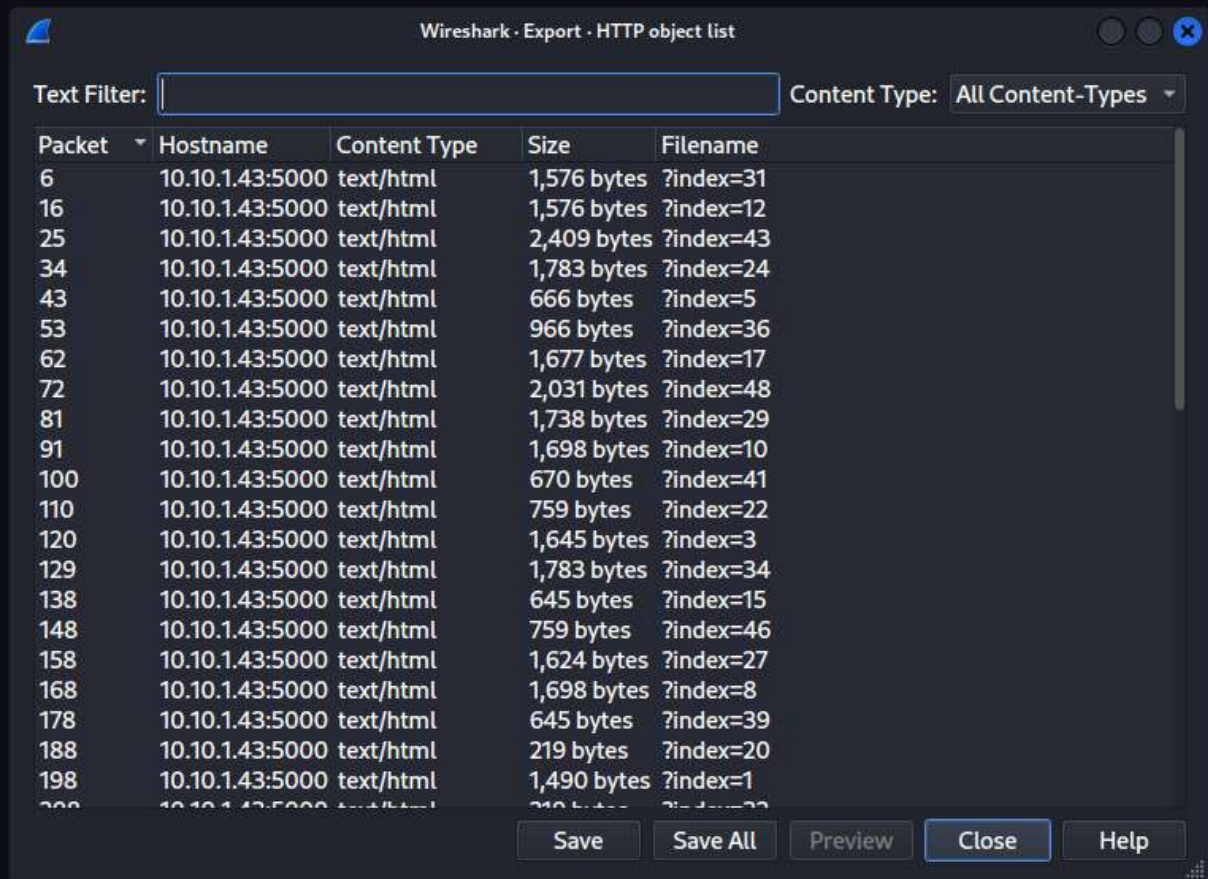
<pcap file: traffic.pcap>

Solusi:

Diberikan sebuah file pcap yang dapat dianalisis dengan menggunakan wireshark. Setelah melakukan analisis terhadap file yang diberikan, didapatkan bahwa file tersebut berisi request http dengan metode GET untuk melakukan request terhadap index dengan beberapa parameter angka.



Karena menggunakan protokol HTTP, jadi kita bisa melakukan export object HTTP dengan cara memilih menu File -> Export Object -> HTTP dan didapatkan hasil sebagai berikut.

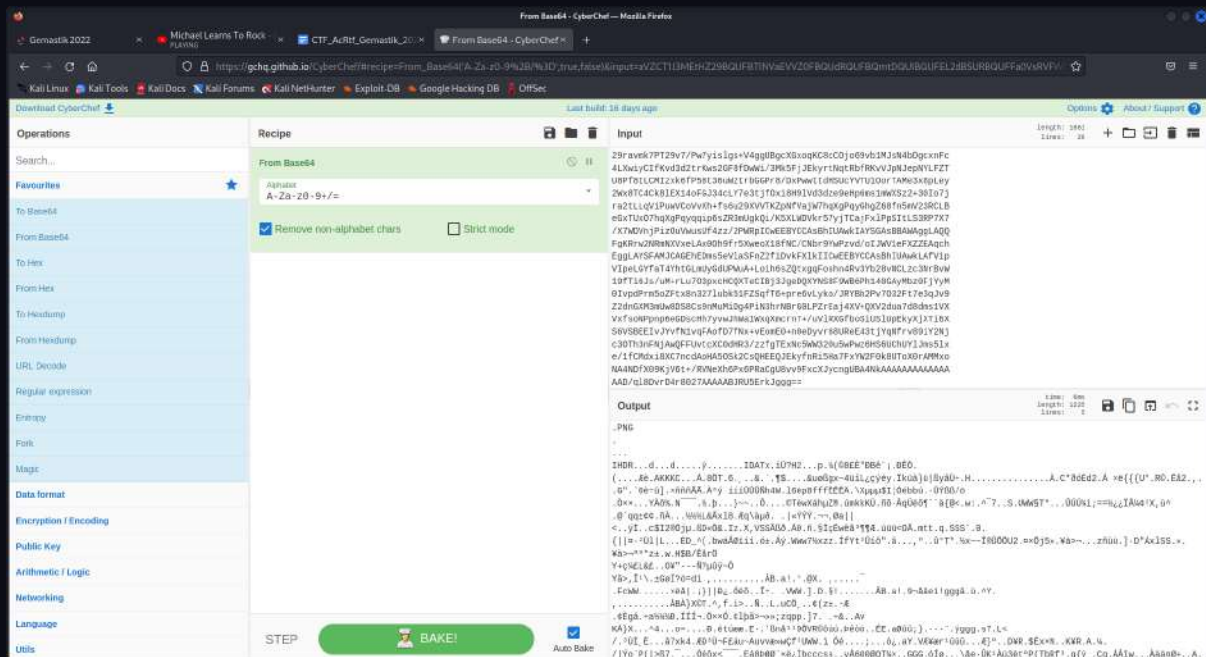


Seluruh object tersebut kemudian disimpan dengan memilih menu Save All dan tersimpan ke dalam folder yang dipilih. Hasilnya, didapatkan 50 file txt yang di dalamnya berisi sebuah hasil decode yang dicurigai hasil encode dengan base64.

```
~/Documents/Gemastik 15/Forensic/traffic, SOLVED/hasil_export/%3findex=0 - Mousepad
File Edit Search View Document Help
1 iVBORw0KGgoAAAANSUHEUGAAAGQAAABkCAIAAAD/gAIDAAAEkELEQVR4n03cP0gy
2 fxwHcI28KKk4o8gi0ELqqKGI0MrShigaGoKCxugPQUtLS0M0ra3BmTjWVBA2FLgE
3 EiaVqAS2JIKVFZkmdfjfZ3h+NPzuTL/n/el5ns9r/OB9/HzfeeLZ95RIAAAAAAAA
4 AAAAAAAAAAAAwJ9DKvDzyWQyg8Gg1+t7e3tVKpVSqZTL5TKZLB6PRYKRYDDo9/td
5 Ltfx8fHDw4PAs/0g7e3t29vb0Wg0V4RsNutwOGZmZsrKysQeXFi1tbUkSabT6WJi
6 +h+3293f3y/2CoTS19cXDAZZxPQlnU6vr6+LvQ7+jY2NfX5+lpLUL52dHaLU6HdY
7 4Wi1Wq6S+m1ra0vcFfH1t8Jx30v1trS05HtAPB53Op1erzccDLMUVDXp1Qqh4eH
8 29ravmk7PT29v7/Pw7yislgs+V4ggUBgcXGxoqKC8cC0jo69vb1MJsN4bDgcxnFc
9 4LXwiYCIkVd3d2trKws2GF8fDwWi/3Mk5FjJEkyrtNqtRbfRKvVJpNJepNYLFZT
10 U8Pf8ILCMIzXk6fP58t36uWztrbGGPr8/DxPwwttDHSUcYVTU10orTAME3x8pLey
11 2Wx8TC4Ck8lEX14oFGJ34cLY7e3tjf0xi8H9lVd3dze9eHp6ms1mWXSz2+30Io7j
12 ra2tLLqViPuwVCovVxH+fs6u29XVVTkZpNfVajW7hqXgPqyGhgZ68fn5mV23RCLB
13 eGxTUx07hqXgPqyqqip6sZR3mUgkQi/K5XLWDVkr57yjTCajFxlPpSiTlS3RP7X7
14 /X7WDVnjPizOuVwusUf4zz/2PWRpICwEEBYCCAsBhIUAwkIAYSGAsBBAWAggLAQQ
15 FgKRrw2NRmNXVxeLax00h9fr5XweoX18fNC/CNbr9YwPzvd/oIJWV1eFXZZEAqch
16 EggLAYSFAMJCAGEhEDms5eVlaSFnZ2fiDvkFXlkIICwEEBYCCAsBhIUAwkLafVip
17 VIpeLGYfaT4YhtGLmUyGdUPWuA+Loih6sZQtXgqFoshn4Rv3Yb28vNCLzc3NrBvW
18 19fTi6Js/uM+rLu703pxcHCQXTeCIBj3JgeDQXYNS8F9WB6Ph140GAYmbz0FjYyM
19 0IvpdPrm5oZFtx8n327lubk51FZSqfT6+pre6vLyko/JRYBh2Pv7032Ft7e3qJv9
20 Z2dnGXM3mUw8DS8Cs9nMuMiDg4PiN3hrNBrg0LPZrEaj4XV+QXV2dua7d8dms1VX
21 VxfsoNPpnp6eGDscHh7yvwJhWa1WxqXmcrn7+/uVlRXGfbosIUslUpEkyXjXTi6X
22 S6VSBEEIvJYvfN1vqFAofD7fNx+vEomE0+n0eDyvr68URRe43tjYqNfrv89iY2Nj
23 c30Th3nFNjAwQFFUvtcXC0dHR3/zzfgTExNc5WW320u5wPwz6HS6UChUYlJms5Lx
24 e/1fCMdxI8XC7ncdAoHA50Sk2CsQHEEQJEkyfnRi5Ha7FXYW2F0k8UT0X0rAMMxo
25 NA4NDfX09KjV6t+/RVNeXh6Px6PRaCgU8vv9FxcXJycngUBA4NkAAAAAAAAAAAA
26 AAD/ql8DvrD4r8027AAAAABJRu5ErkJggg==
```

Text yang dicurigai sebagai hasil encode base64 tersebut kemudian dilakukan decode dan ternyata benar merupakan string base64.

db



Dilihat dari hasil decode, dapat disimpulkan bahwa hasil tersebut merupakan string hex dari sebuah gambar dengan ekstensi png. Kemudian melakukan percobaan pada satu file untuk dijadikan gambar dengan command berikut pada terminal.

```
cat %3findex=0 | base64 -d > image.png
```

Hasil dari *command* tersebut berupa gambar huruf G. Artinya, dapat diasumsikan bahwa satu file text hasil decode base64 merepresentasikan string hex tiap huruf flag. Kemudian, dilakukan otomasi untuk merubah seluruh file menjadi gambar dengan menggunakan *command* bash berikut.

```
for i in {0..49}
do
    cat %3findex=$i | base64 -d > img$i.png
done
```

Dan asumsi sebelumnya benar dan dihasilkan gambar dengan jumlah 50 berupa huruf dari flag.

db



Flag: Gemastik2022{balapan_first_blood_is_real_f580c176}