**Writeup LycoReco**

**Gemastik 2022 - Penyisihan**



**Anggota tim:**

**Muhammad Garebaldhie Er Rahman (average kobo enjoyer)**

**Frederik Imanuel Louis (azuketto)**

**Rachel Gabriela Chen (chaerla)**

# **Daftar Isi**

[**Daftar Isi**](#_jtobumv3ewy0) **2**

[**Crypto**](#_dx0lkflrke9q) **3**

[Doublesteg](#_m36wkrp4fafi) 3

[Flag: Gemastik2022{uji\_nyali\_encrypt\_message\_pakai\_weak\_key}](#_5xahmfd9i27t) 6

[**Rev**](#_sd17a0qq89j2) **7**

[CodeJugling](#_b0celkyopunv) 7

[Flag: Gemastik2022{st45iUn\_MLG\_k07a\_b4rU}](#_84vqs1m55ebk) 11

[Dino](#_rj9rgunypg6i) 11

[Flag: Gemastik2022{why\_would\_you\_ever\_beat\_me}](#_efl4o870be5) 14

[Rubyte](#_wzglsll0i38z) 14

[Flag: Gemastik2022{i\_still\_remember\_30\_october}](#_i1je37h21fe8) 16

[**Foren**](#_421l36uhtqh) **17**

[Traffic Enjoyer](#_m6mj3ftoe9vq) 17

[Flag : Gemastik2022{balapan\_f1rst\_bl00d\_is\_real\_f580c176}](#_mf5cuc1ioaxy) 19

[Har](#_isikc1u0chkh) 20

[Flag : Gemastik2022{kinda\_wish\_this\_werent\_text}](#_1ijm6tx17bll) 22

# Crypto

# 

| Doubleste**g** 892  Single STEG encryption is weak, how about double STEG encryption?  author - deomkicer#3362 |
| --- |

| 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 |
| --- |

Pada chall, png pada dasarnya diencrypt menggunakan AES CBC mode sebanyak dua kali menggunakan key yang berbeda. Pada chall, iv yaitu STEG sudah diberikan, dan key diderive dari STEG yang hanya berukuran 8 byte yang dipermutasi. Perhatikan bahwa 8! = 4 \* 10^5. Karena enkripsi dilakukan dua kali, enkripsi rentan terhadap Meet in the Middle attack. Perhatikan pula bahwa file yang diencrypt adalah PNG, sehingga kita memiliki sebagian plaintext, yaitu header dari PNG. Dengan itu kita dapat melakukan mapping dari enkripsi header PNG dengan semua kemungkinan key yang mungkin. Kemudian, kita mendekripsi ciphertext dengan semua key yang mungkin dan mengecek apakah header ada pada mapping yang kita buat. Search tersebut dapat dilakukan dengan kompleksitas O(log(n)) dengan n = 8!, dan untuk semua key, kompleksitas total menjadi O(n log(n)) yang masih feasible untuk bruteforce. Code bruteforce diimplementasikan sebagai berikut:

**from Crypto.Cipher import AES**

**from Crypto.Hash import SHA256**

**from Crypto.Util.Padding import \***

**import random**

**import itertools**

**STEG = b"gemasteg"**

**import bisect**

**def find\_in\_sorted\_list(elem, sorted\_list):**

***# https://docs.python.org/3/library/bisect.html***

**'Locate the leftmost value exactly equal to x'**

**i = bisect.bisect\_left(sorted\_list, elem)**

**if i != len(sorted\_list) and sorted\_list[i] == elem:**

**return i**

**return -1**

**def fwrite(filename: str, data: bytes):**

**f = open(filename, "wb")**

**f.write(data)**

**f.close()**

**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 decrypt(msg: bytes, key: bytes):**

**key = SHA256.new(key).digest()**

**iv = STEG \* 2**

**aes = AES.new(key, AES.MODE\_CBC, iv)**

**enc = aes.decrypt(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 double\_decrypt(msg: bytes, keys: list[bytes]):**

**msg = pad(msg, AES.block\_size)**

**for key in keys:**

**msg = decrypt(msg, key)**

**return msg**

**FLAG = open("flag.enc", "rb").read()**

**def getrandsteg():**

**x = list(STEG)**

**random.shuffle(x)**

**return bytes(x)**

**print(getrandsteg())**

**sample = open("about\_img.png", "rb").read()**

**a = sample[:16]**

**print(len(encrypt(a, STEG))) *#16***

**head = []**

**mp = {}**

**for ls in list(itertools.permutations(STEG)):**

**key = [chr(c).encode() for c in ls]**

**key = b"".join(key)**

**ct = encrypt(a, key)**

**mp[ct]=key**

**head.append(ct)**

**head.sort()**

**for ls in list(itertools.permutations(STEG)):**

**key = [chr(c).encode() for c in ls]**

**key = b"".join(key)**

**semi = decrypt(FLAG[:32], key)**

**idx = find\_in\_sorted\_list(semi[:16], head)**

**if idx!=-1:**

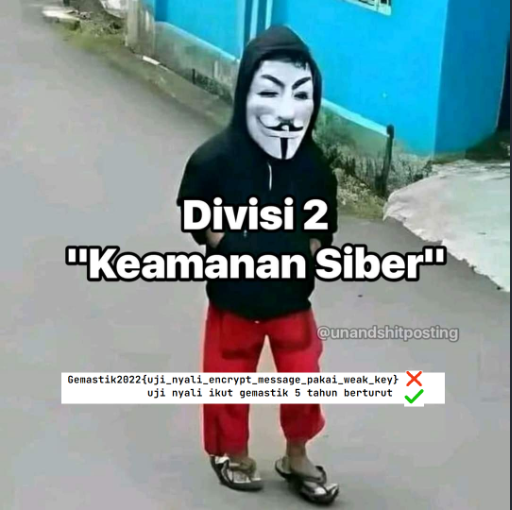
**mid = head[idx]**

**newkey = mp[mid]**

**buf = double\_decrypt(FLAG, [key, newkey])**

**fwrite("plain.png", buf)**

Kemudian, pada plain.png, diperoleh png sebagai berikut:

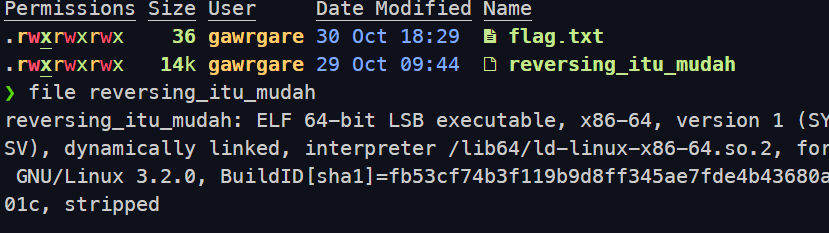
****

### Flag: Gemastik2022{uji\_nyali\_encrypt\_message\_pakai\_weak\_key}

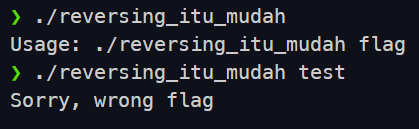
# **Rev**

| **CodeJugling** 500  Find the flag!  author - vidner#6838 |
| --- |

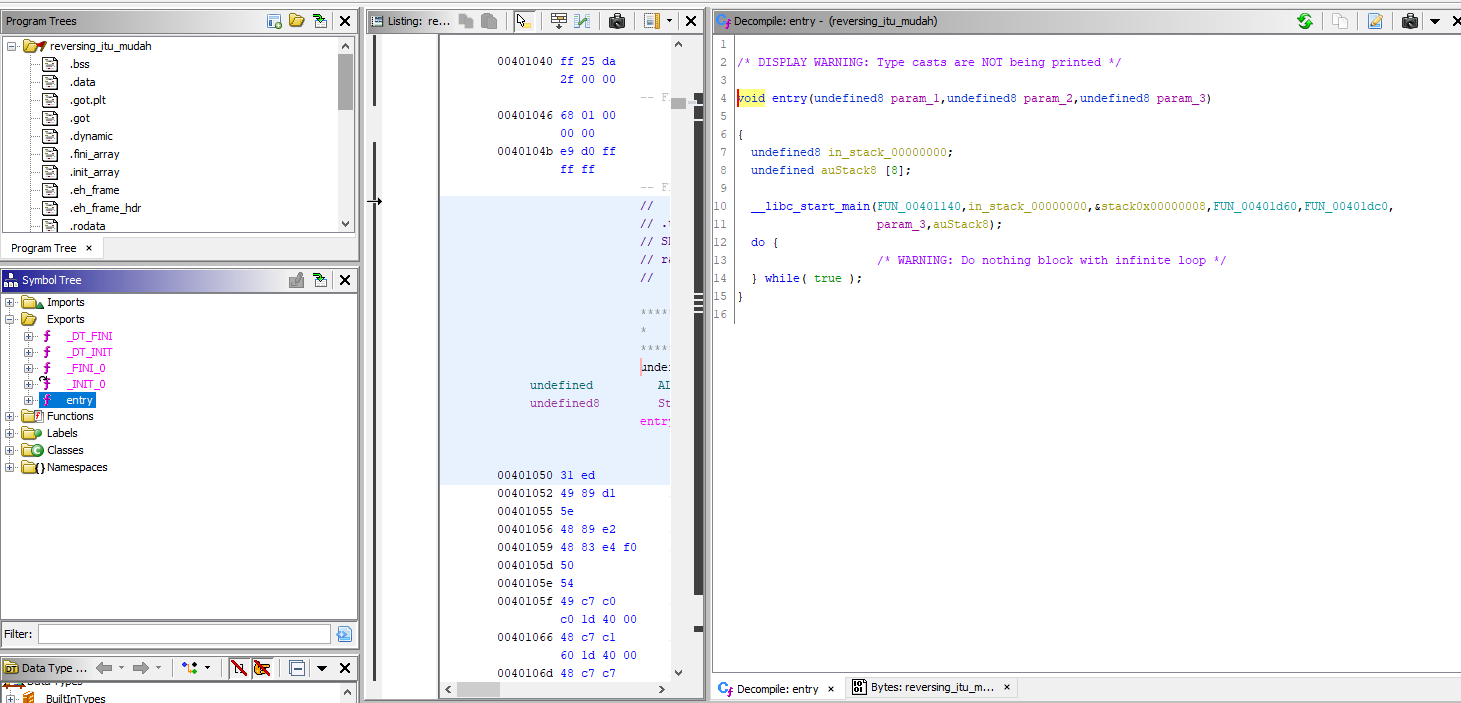
Kita diberikan sebuah binary file elf 64 bit stripped sebagai berikut



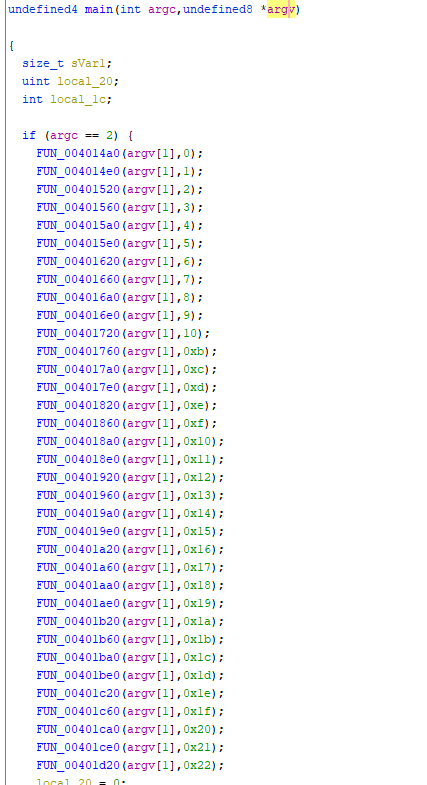
Ketika dijalankan program meminta sebuah argumen yang nantinya akan digunakan sebagai pengecekan flag

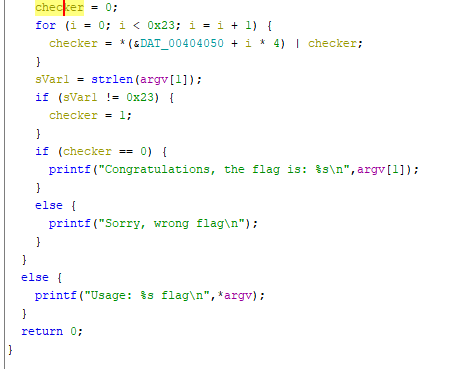


Langsung saja kami buka file tersebut dengan ghidra binary tersebut untuk dilihat dan dieksekusi lebih lanjut dan dengan mudah kami menemukan fungsi main dari binary

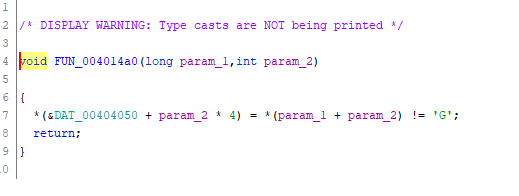


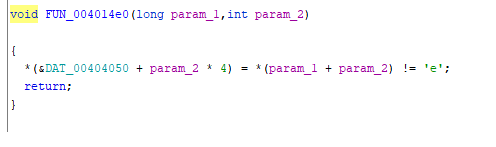
Dapat dilihat bahwa main memiliki beberapa fungsi yang dieksekusi ketika dijalankan

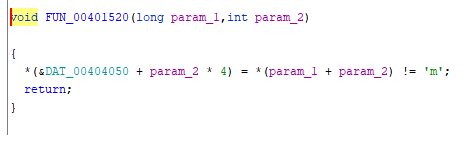




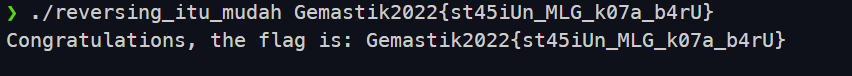
Dapat dilihat bahwa agar flagnya muncul maka nilai checker haruslah bernilai 1. Dan nilai checker diapat dengan melakukan bitwise or nilai 0 dengan nilai yang ada pada memory tersebut. Lalu mulailah kami melakukan pengechekan untuk setiap fungsi yang dijalankan







Ketiga fungsi yang dijalankan pertama akan melakukan pengechekan apakah huruf dari index ke i sama dengan huruf yang akan di check dan curiga bahwa setiap fungsi membentuk sebuah kata yang merupakan flag maka kami catat setiap huruf yang di check pada fungsi tersebut dan didapat flag



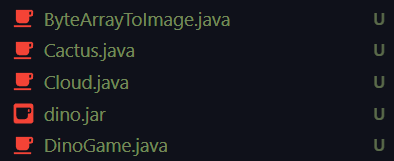
### Flag: Gemastik2022{st45iUn\_MLG\_k07a\_b4rU}

| **Dino** 500  Beat my highscore!  author - vidner#6838 |
| --- |

Diberikan sebuah jar file dan highscore.txt yang berisi

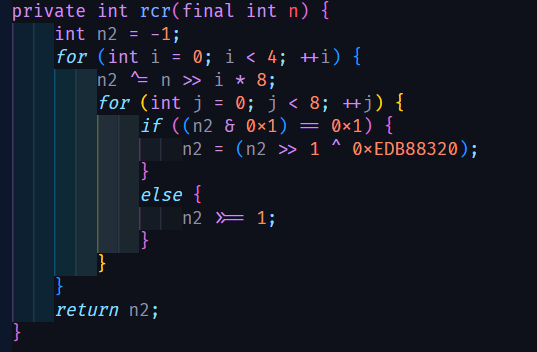


Lalu tanpa berpikir panjang kami pun melakukan decompilasi file Dino.jar tersebut dengan online decompiler dan mendapat empat buah file .java

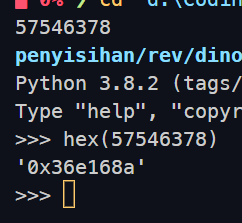


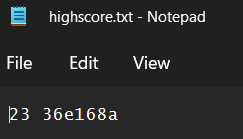
Pada dino game.java terdapat fungsi ls Yang berfungsi untuk membaca highscore.txt dan mencocokan dengan hasil crc yang terpisah oleh spasi. Probset meminta untuk kita mematikan program tersebut



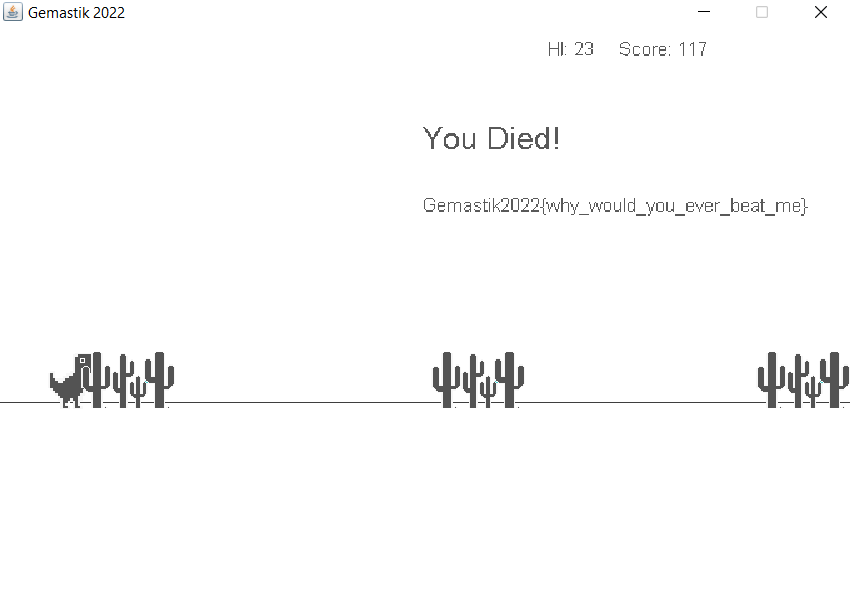


Lalu kami mencoba untuk merubah highscore.txt sekaligus mencocokan crc nya. Kami mengeset highscore dengan 23 sekaligus crc nya



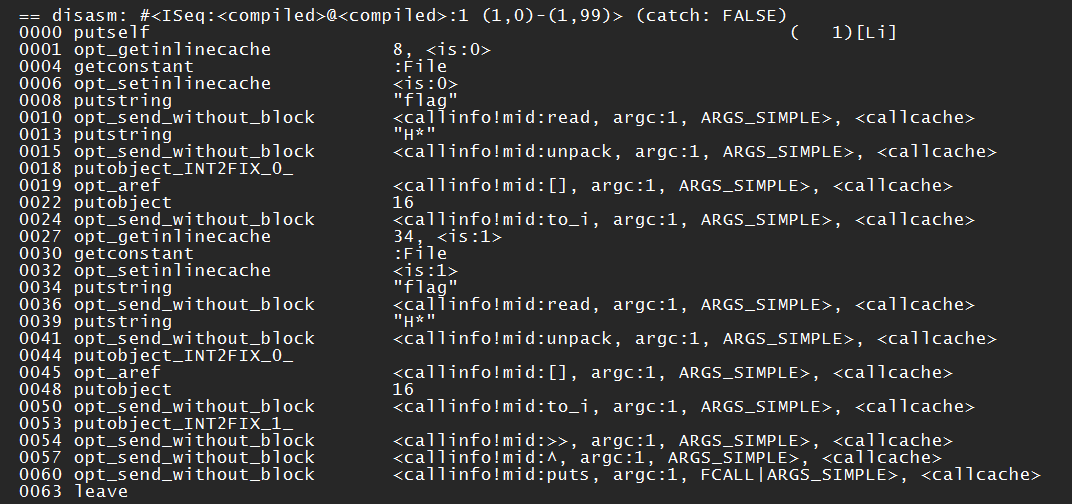


Lalu kami pun menjalankan game nya dan ketika melebihi 23 dan flag pun didapatkan



### Flag: Gemastik2022{why\_would\_you\_ever\_beat\_me}

| Rubyte 820  Hope you find the hidden gem!  author - vidner#6838 |
| --- |



Pada byte.txt, kita lihat bahwa dilakukan load flag sebanyak dua kali (line 4-24 dan line 30-50), dimana flag yang dibaca sudah dikonversi menjadi integer (pertama menjadi hex, kemudian menjadi integer). Setelah itu, flag kedua di-right shift sebanyak satu (line 53-54), kemudian di-xor dengan flag pertama.

Maka, kita cukup me-reverse hasil right-shift dan xor. Hal tersebut dapat dilakukan sebagai berikut:

Misalnya flag awal (per bit) adalah f[0],f[1],…,f[n]. Setelah right shift dan xor, ciphertext akan menjadi f[0], f[0]^f[1], f[1]^f[2],…,f[n-1]^f[n]. Maka, untuk mendecrypt ciphertext (c[0],… ,c[n]), kita lakukan flag = (c[0], c[0]^c[1], c[1]^c[2],…,c[n-1]^c[n]). Implementasi algoritma tersebut dilakukan sebagai berikut:

from Crypto.Util.number import \*

ct = 215399763437993922857257938507183571899033473988099831289577921701237839559370177126393638370659139 *# x ^ (x>>1)*

arr = []

temp = ct

while temp>0:

arr.append(temp%2)

temp//=2

arr.reverse()

for i in range(1, len(arr)):

arr[i]^=arr[i-1]

arr.reverse()

ans = 0

for i in range(len(arr)):

ans += pow(2, i)\*arr[i]

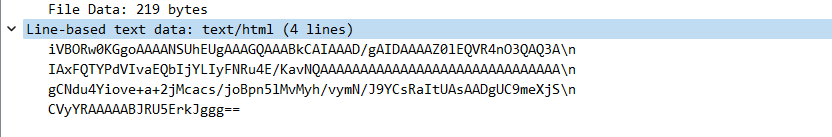
print(long\_to\_bytes(ans))

### Flag: Gemastik2022{i\_still\_remember\_30\_october}

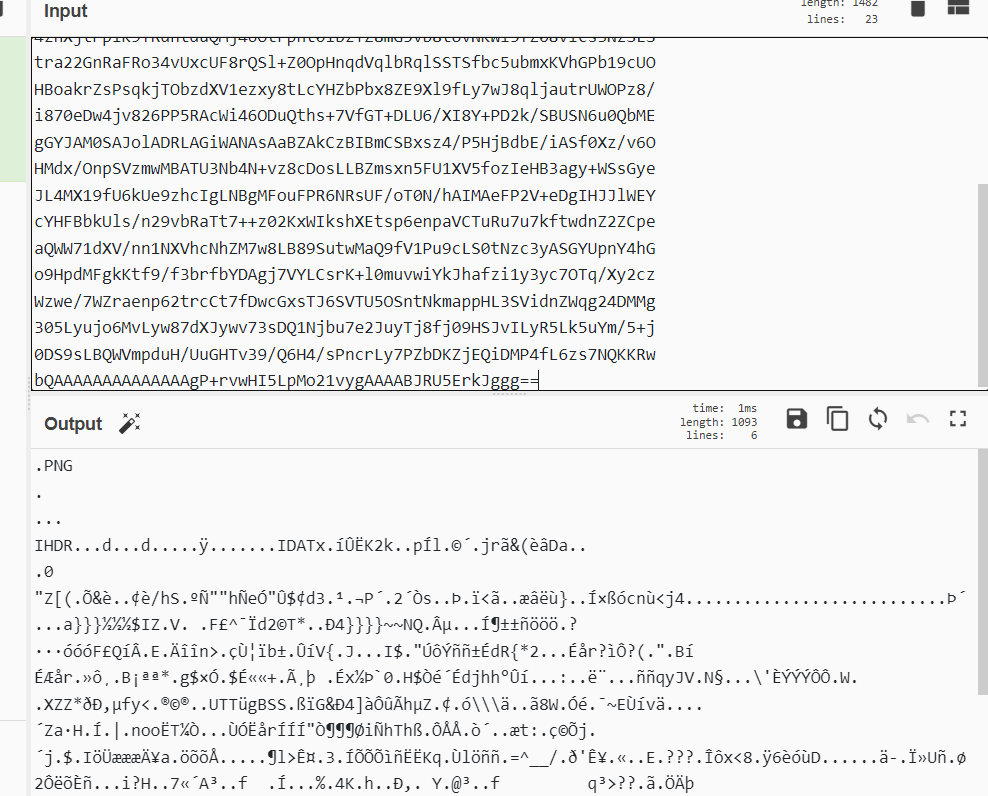
# **Foren**

| **Traffic Enjoyer** 500  P balap first blood  author - deomkicer#3362 |
| --- |

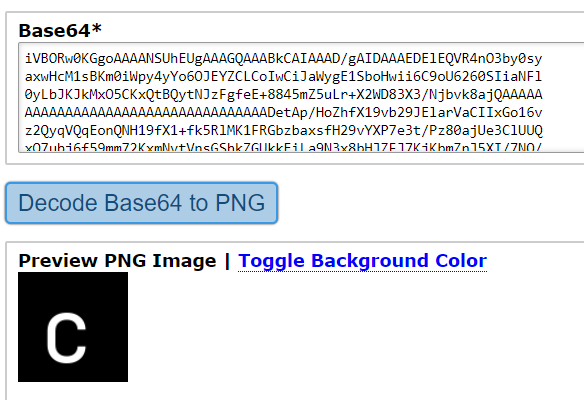
Kita diberikan file traffic.pcap. Dengan mengurutkan packet bytes berdasarkan length, kita mendapatkan sejumlah HTTP RESPONSE. Setelah mengecek packet bytesnya, kita mendapatkan file .txt yang diencode dengan base64:



Kita kemudian mencoba mendecode isi dari salah satu .txt tersebut:



Sehingga kita mengetahui bahwa setiap file .txt merupakan sebuah file .png yang diencode dengan base64. Kita kemudian menggunakan online tool <https://base64.guru/converter/decode/image/png> untuk mendecode semua file .txt. Kita mendapatkan bahwa setiap file .txt mewakili sebuah huruf pada flag dengan index yang bersesuaian dengan index yang di-GET.



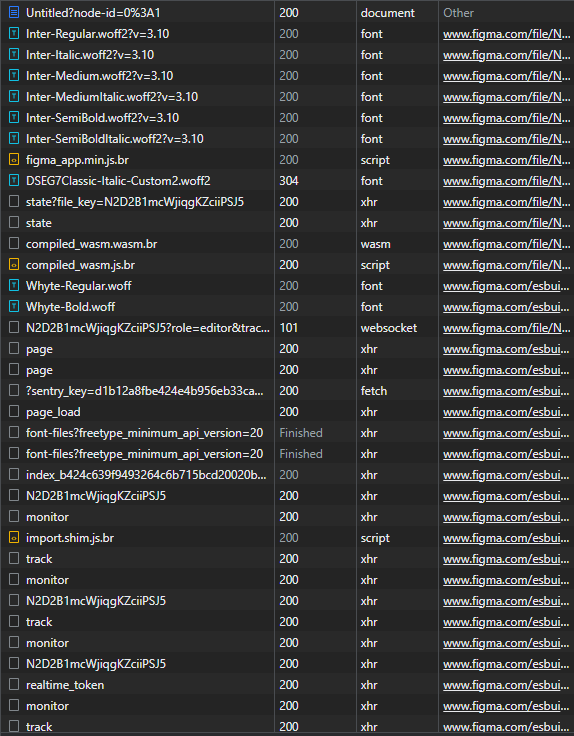
Kita kemudian mengurutkan semua karakter yang didapatkan dan mendapatkan:

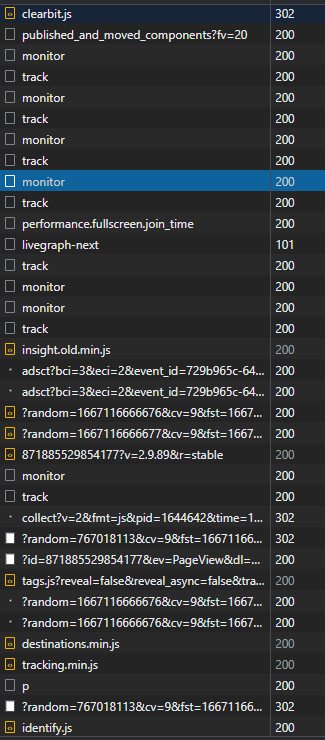
### Flag : Gemastik2022{balapan\_f1rst\_bl00d\_is\_real\_f580c176}

| **Har** 500  Har Har Har!  author - vidner#6838 |
| --- |

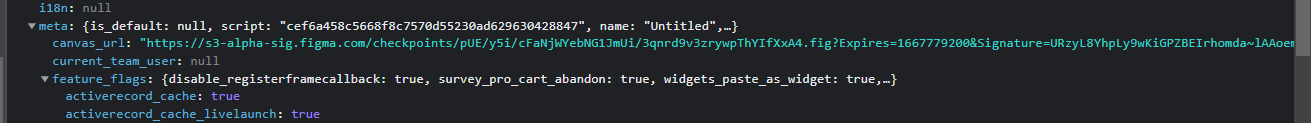
Diberikan sebuah har file yang di zip. Kami langsung membuka file tersebut di chrome dev tools untuk melakukan analysis terhadap request yang dijalankan.

Kami mendownload file tersebut dan membukanya di figma



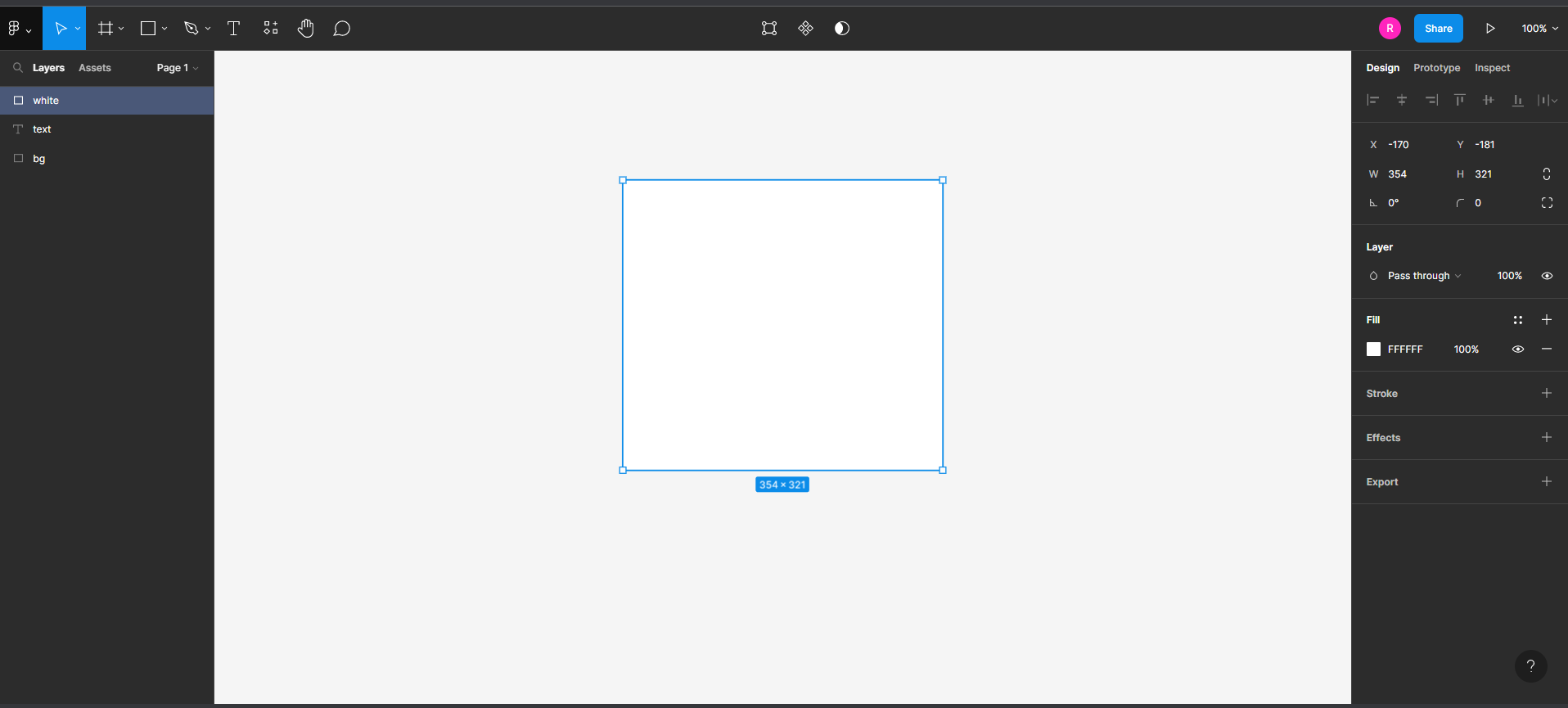


Dapat dilihat bahwa terdapat banyak request yang terjadi dan kami melakukan analysis untuk setiap request. Dan menemukan beberapa hal yang menarik.



Kami mendapatkan sebuah canvas\_url = [*https://s3-alpha-sig.figma.com/checkpoints/pUE/y5i/cFaNjWYebNG1JmUi/3qnrd9v3zrywpThYIfXxA4.fig?Expires=1667779200&Signature=URzyL8YhpLy9wKiGPZBEIrhomda~lAAoem5a9SVtu7lNvem2iswOXca9gzr69rSOu4ljLpPSPCrDT-kdYRBqY5p4jnKxjad5nwiG6KbbljZrzTmStHIMbgOwJlfQcd3w77UwL2fnNuUVl6MKYEO2I3qGH50M0YDrjOIrtyfhi471o26v~qDLB7pdrZn9ycRTKZbLGXtJyJ1Iq90nq0i5yx4i6NxkrTg4K5kxjZGL8VUzUXKBdXbQZANGlpsuEAQ4aALfqS5yCpko87QyTJUxR5bQLAP1Y0jTWWMoHoLM3eUyPNy2y~e77Wduk2o-vcMJzSKxoROk7xX1sWKlhqgspg\_\_&Key-Pair-Id=APKAINTVSUGEWH5XD5UA*](https://s3-alpha-sig.figma.com/checkpoints/pUE/y5i/cFaNjWYebNG1JmUi/3qnrd9v3zrywpThYIfXxA4.fig?Expires=1667779200&Signature=URzyL8YhpLy9wKiGPZBEIrhomda~lAAoem5a9SVtu7lNvem2iswOXca9gzr69rSOu4ljLpPSPCrDT-kdYRBqY5p4jnKxjad5nwiG6KbbljZrzTmStHIMbgOwJlfQcd3w77UwL2fnNuUVl6MKYEO2I3qGH50M0YDrjOIrtyfhi471o26v~qDLB7pdrZn9ycRTKZbLGXtJyJ1Iq90nq0i5yx4i6NxkrTg4K5kxjZGL8VUzUXKBdXbQZANGlpsuEAQ4aALfqS5yCpko87QyTJUxR5bQLAP1Y0jTWWMoHoLM3eUyPNy2y~e77Wduk2o-vcMJzSKxoROk7xX1sWKlhqgspg__&Key-Pair-Id=APKAINTVSUGEWH5XD5UA)

Kami mendownload file tersebut dan membukanya di figma:



Terdapat sebuah kotak putih. Dengan menggeser kotak putih tersebut, kita mendapatkan flag.

### Flag : Gemastik2022{kinda\_wish\_this\_werent\_text}