

Gemastik (Keamanan Siber) - Anya Haha Inakute Sabishii



Anggota:

Fauzan Aldi

Reverse Engineering

CodeJugling

Diberikan program, setelah decompile pada fungsi **main** terdapat hasil berikut:

```

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 )
    }
}

```

Pada intinya setiap fungsi melakukan cek dari parameter **argv[1]** sesuai index yang diberikan, setiap fungsi akan melakukan check dari byte ke index yang diberikan, seperti contohnya pada fungsi ini:

```

__int64 __fastcall sub_4014A0(__int64 a1, int a2)
{
    __int64 result; // rax

    result = a2;
    dword_404050[a2] = (*(char *)(a1 + a2) ^ 0xEC) != 0xAB;
    return result;
}

```

a1[a2] akan di xor dengan 0xEC lalu dicek apakah hasilnya 0xAB. Maka dari ini untuk kita perlu cukup melakukan: 0xEC *XOR* 0xAB menghasilkan char 'G'. Oke karena kita sudah mengerti bagaimana cara pengecekan program ini, maka kita langsung rekonstruksi setiap fungsi kedalam script yang kita buat:

```

#!/usr/bin/python3

flag = []

flag.append(0xEC ^ 0xAB)
flag.append(0x65)
flag.append(0x6D)
flag.append(0x61)
flag.append(0x6C ^ 0x1F)
flag.append(0xF8 ^ 0x8C) flag.append(0x58
^ 0x31)
flag.append(0x6F ^ 0x4)
flag.append(0x37 ^ 0x5)
flag.append(0xCD ^ 0xFD)
flag.append(0x3E ^ 0xC)
flag.append(0xCC ^ 0xFE)
flag.append(0x70 ^ 0xB) flag.append(0x73)

flag.append(0x24 ^ 80)
flag.append(0x60 ^ 84)
flag.append(0x10 ^ 37)
flag.append(105)

```

```

flag.append(0xC3 ^ 150)
flag.append(110)
flag.append(95)
flag.append(0x4d) flag.append(0x86
^ 202)
flag.append(0x80 ^ 199)
flag.append(0xD8 ^ 135)
flag.append(0x82 ^ 233)
flag.append(0x27 ^ 23)
flag.append(0x9B ^ 172)
flag.append(0x93 ^ 242) flag.append(0x7A
^ 37)
flag.append(98)
flag.append(52)
flag.append(114)
flag.append(0xD1 ^ 132)
flag.append(0xD ^ 112)

print("".join([chr(i) for i in flag]))

```

Jalankan script dan got flag!

```

(aimardcr@kuro)-[/mnt/c/Users/aimar/Desktop/gemastik/CodeJuggling]
$ python3 solve.py
Gemastik2022{st45iUn_MLG_k07a_b4rU}

```

FLAG: Gemastik2022{st45iUn_MLG_k07a_b4rU}

Dino

Diberikan file .jar dan .txt, yang dimana file .jar ini merupakan game dinosaurus yang bisa dimainkan ketika kita tidak punya internet di browser Chrome, lalu file highscore.txt yang dimana disini terdapat score tertinggi sebesar 2147482310 dan juga sebuah checksum untuk score tersebut supaya score tidak bisa diubah dengan mudah. Karena kita tidak tau tujuan awal dari chall ini, setelah membaca deskripsi chall ini maka kami asumsikan bahwa kami perlu mendapatkan score tertinggi,

namun karena ini tidak mungkin karena score tertingginya sangat tinggi, maka kita coba cari cara untuk mencari checksum untuk score 1. Setelah sedikit reversing, ditemukan fungsi yang kita duga merupakan fungsi checksum tersebut:

```
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;
}
```

Bisa dilihat bahwa fungsi **ls** merupakan fungsi untuk melakukan load highscore, yang dimana akan dilakukan check juga terhadap checksum dari scorenya. Oke karena fungsi checksum/**rcr** cukup simpel, tinggal kita jalankan fungsi tersebut seperti berikut:

```

private static 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;
}

public static void main(String[] args) {
    System.out.println(Integer.toHexString(rcr(1)));
}

```

Berhasil kita dapatkan checksum untuk score 1: **a06002d**. Ubah score tertinggi pada highscore.txt dan jangan lupa checksumnya dan jalankan gamenya.

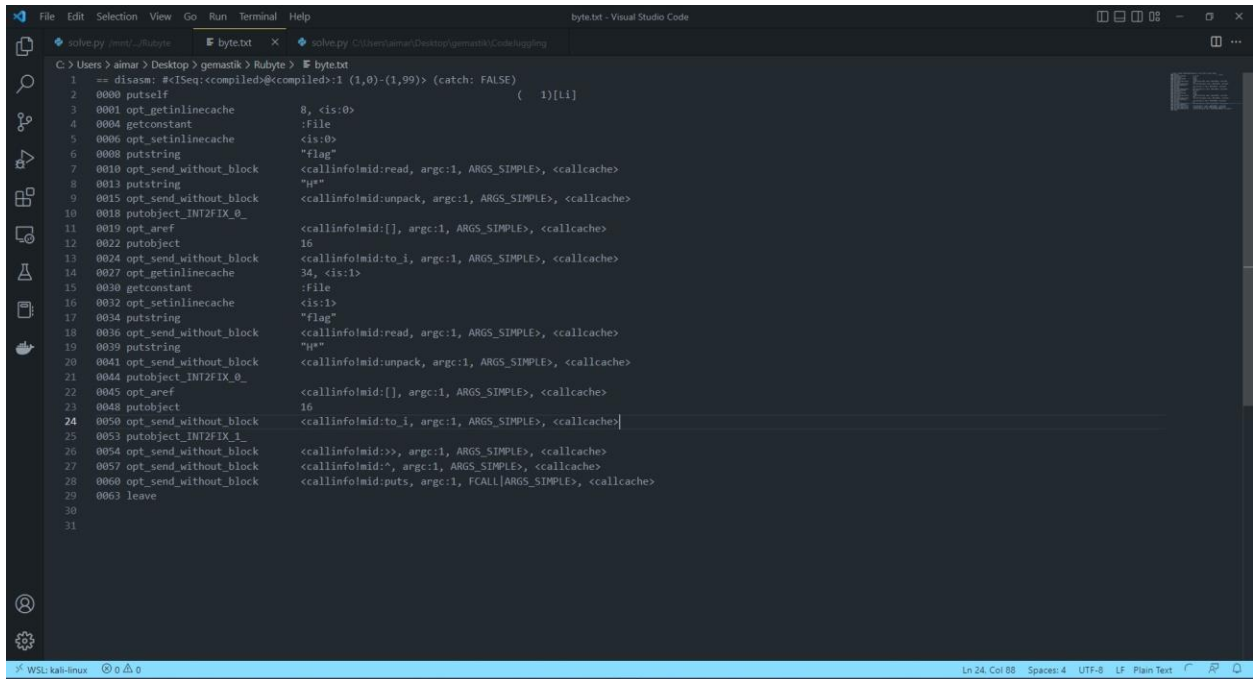


Got flag!

FLAG: Gemastik2022{why_would_you_ever_beat_me}

Rubyte

Diberikan bytes.txt yang merupakan hasil disassembly program ruby, dan juga output.txt yang merupakan hasil encrypt dari program tersebut.



Setelah kami analisa berjam-jam, ternyata program ruby ini bekerja cukup simpel, yaitu seperti:

```
File.read("flag").unpack("H*")[0].to_i(16) ^  
File.read("flag").unpack("H*")[0].to_i(16) >> 1
```

Pertama program akan membaca file **flag**, lalu melakukan unpack yang maksud dari unpack ini merupakan mengubah string yang telah dibaca dari file **flag** menjadi hex, lalu akan diubah menjadi int.

```
from re import X  
from Crypto.Util.number import *
```

```
9370177126393638370659139  
shiftamount = 1  
while x >> shiftamount:
```



```

x ^= x >> shiftamount
shiftamount <<= 1

print(long_to_bytes(x))
x =
21539976343799392285725793850718357189903347398809983128957792170123783955

```

Setelah sedikit penelusuran, ternyata program ruby ini mengimplementasikan encryption **Binary to Grey Code**. Oke karena kita sudah tipe encryptionnya, cukup kita implementasikan dalam script python:

Jalankan scriptnya dan got flag!

```

(aimardcr@kuro)-[/mnt/c/Users/aimar/Desktop/gemastik/Rubyte]
$ python3 solve.py
b'Gemastik2022{i_still_remember_30_october}'

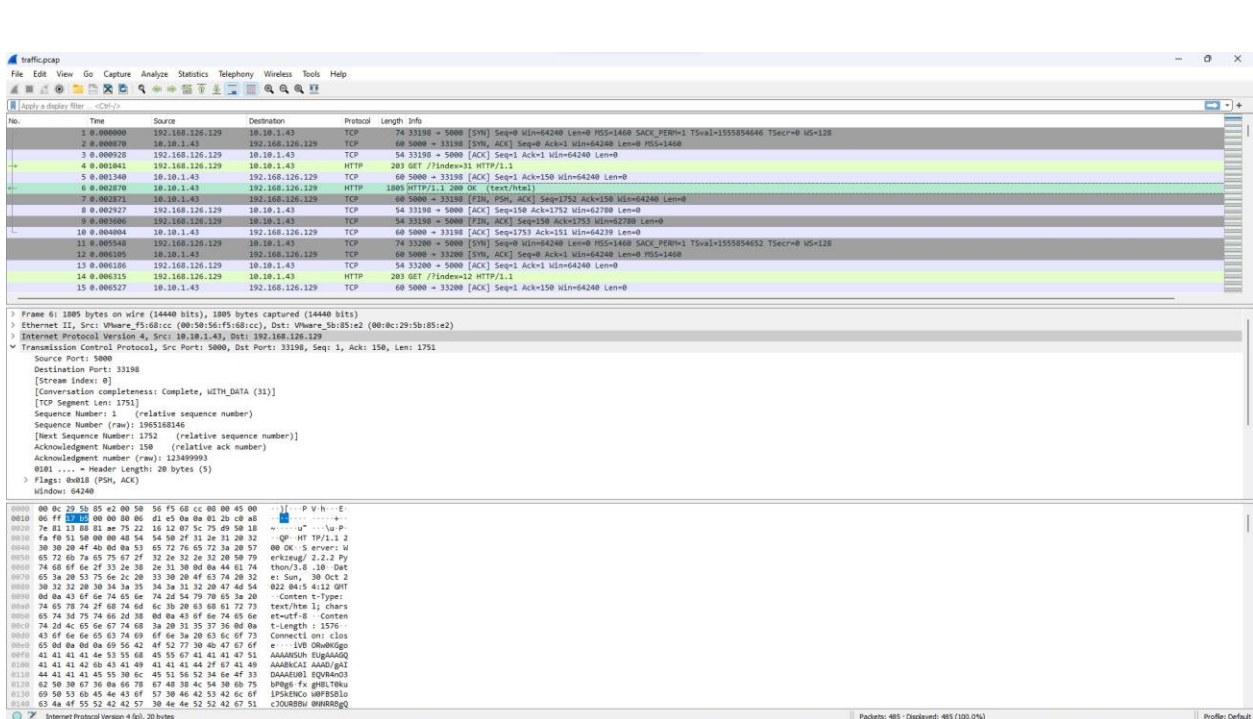
```

FLAG: **Gemastik2022{i_still_remember_30_october}**

Forensic

Traffic Enjoyer

Diberikan file .pcap, setelah kami analisis lebih lanjut ternyata isi dari file ini merupakan capture dari sebuah url yang dimana respons yang diberikan merupakan file PNG yang telah di-base64:



Namun setiap traffic hanya menyediakan 1 huruf dari flagnya, oke karena dari itu kami membuat script untuk melakukan parse. Pertama script ini akan mengumpulkan url dan response dari traffic <http://10.10.1.43:5000/>, lalu script akan melakukan sorting sesuai index yang diberikan lalu decode base64 nya dan write ke file.

```
from pyshark import *
import re
import base64

cap = FileCapture('traffic.pcap')

arr = []

for pkt in cap:
    if 'http' in pkt:
        http = pkt.http
```

```

keys = list(http._all_fields.keys())
values = list(http._all_fields.values())

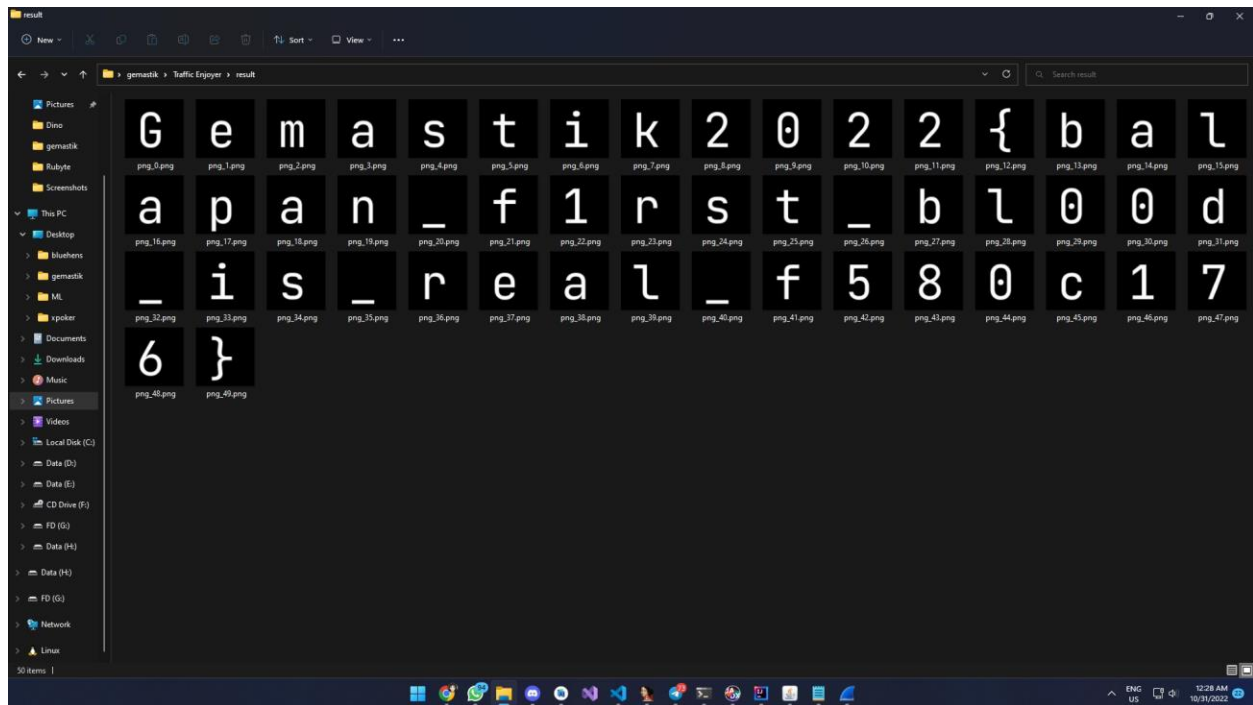
for key, value in zip(keys, values):
    if key == 'http.response_for.uri':
        if value.startswith('http://10.10.1.43:5000/?index='):
            arr.append([
                values[keys.index('http.response_for.uri')],
                values[keys.index('http.file_data')],
            ])

# sort url
arr.sort(key=lambda f: int(re.sub('\D', '', f[0])))

n = 0
for i in arr:
    data = i[1]
    with open('result/png_' + str(n) + '.png', 'wb') as f:
        f.write(base64.b64decode(data))
    n += 1

```

Jalankan script dan got flag!



FLAG: Gemastik2022{balapan_f1rst_bl00d_is_real_f580c176}

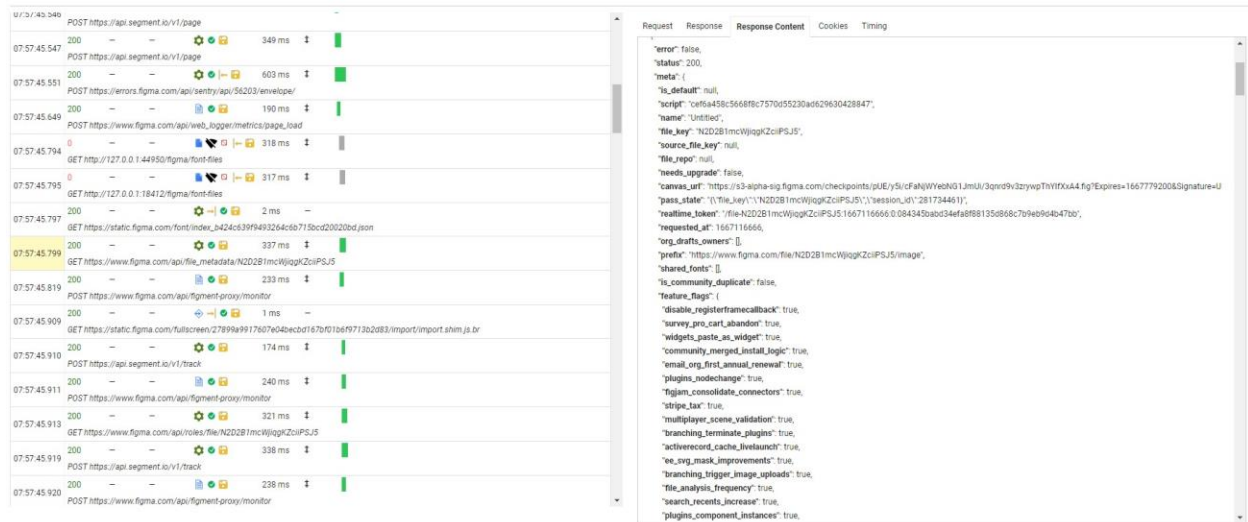
Har

Diberikan file .txt, yang dimana isinya merupakan HAR atau HTTP Archive.

Setelah kami analisis menggunakan tools milik google


https://toolbox.googleapps.com/apps/har_analyzer/.

Oke setelah dianalisa, ternyata ini berisi traffic capture dari website yang sedang dibuka yaitu figma. Jadi kami asumsikan bahwa flag akan berada pada gambar figma yang sudah dibuat. Oke percobaan pertama kami yaitu mencoba untuk mencuri cookie dari owner dari figma tersebut. Setelah kami coba beberapa kali namun gagal, kami analisa lebih lanjut dan terdapat traffic yang cukup menarik pada bagian responsnya:



Terdapat url menuju https://s3-alpha-sig.figma.com/checkpoints/pUE/y5i/cFaNjWYebNG1JmUi/3qnrD9v3zrywpThYIfXxA4.fig?Expires=1667779200&Signature=URzyL8YhpLy9wKiGPZBEIrhomda~lAAoem5a9SVtu7lNvem2iswOXca9gZR69rSOu4ljLpPSPCrDT-kdYR BqY5p4jnKxjad5nwiG6KbbLjZrzTmStHIMbgOwJlfQcd3w77UwL2fnNuUVl6MKYE O2I3qGH50M0YDrjOIrtYfhi471o26v~qDLB7pdrZn9ycRTKZbLGXtJyJlIq90nq0i5y x4i6NxkrTg4K5kxjZGL8VUzUXKBdXbQZANGlpsuEAQ4aALfqS5yCpko87Qy TJU xR5bQLAP1Y0jTWWMoHoLM3eUyPNy2y~e77Wduk2o-vcMJzSKxoROk7xX1s WKlhqgspg__&Key-Pair-Id=APKAINTVSUGEGWH5XD5UA

Setelah url tersebut diakses, terdapat file .fig terdownload. Setelah dibuka terdapat flag yang tersembunyi dibalik sebuah layer, got flag!



Gemastik2022{kinda_wish_this_werent_text}

FLAG: Gemastik2022{kinda_wish_this_werent_text}