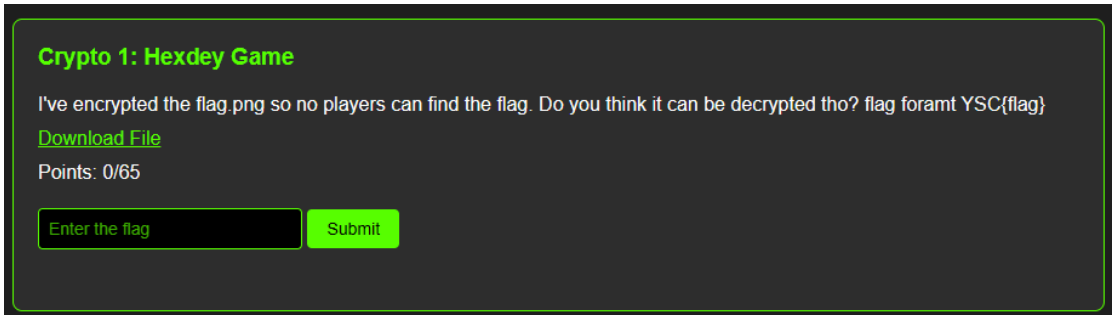


This challenge indicates that the **flag.png** file is encrypted, and it tasks us with decrypting it to uncover the hidden flag.

```
┌──(3BOOD㉿H4CK3RB0Y)-[~/Desktop/ysc-ctf/CRYPTO/Hexdey game -CRYPTO]
└─$ ls
 flag_encrypted.png  'hexdey game.py'

┌──(3BOOD㉿H4CK3RB0Y)-[~/Desktop/ysc-ctf/CRYPTO/Hexdey game -CRYPTO]
└─$ cat hexdey\ game.py
import os

def encrypt_image(input_file, output_file, offset=10):
    """
    Encrypt an image by modifying its hexadecimal data.
    """
    try:
        with open(input_file, "rb") as f:
            data = f.read()

        encrypted_data = bytes((byte + offset) % 256 for byte in data)

        with open(output_file, "wb") as f:
            f.write(encrypted_data)

        print(f"Image encrypted successfully and saved as {output_file}")
    except Exception as e:
        print(f"Error during encryption: {e}")

def decrypt_image(input_file, output_file, offset=10):
    """
    Decrypt an image by reversing the encryption process.
    """
    try:
        with open(input_file, "rb") as f:
            data = f.read()

        decrypted_data = bytes((byte - offset) % 256 for byte in data)

        with open(output_file, "wb") as f:
            f.write(decrypted_data)

        print(f"Image decrypted successfully and saved as {output_file}")
    except Exception as e:
        print(f"Error during decryption: {e}")

if __name__ == "__main__":
    original_image = "flag.png"
    encrypted_image = "flag_encrypted.png"
    decrypted_image = "flag_decrypted.png"

    # Encrypt the image
    if os.path.exists(original_image):
        encrypt_image(original_image, encrypted_image, offset=10)
    else:
        print(f"Error: {original_image} not found in the current directory.")

    # Decrypt the image
    if os.path.exists(encrypted_image):
        decrypt_image(encrypted_image, decrypted_image, offset=10)
    else:
        print(f"Error: {encrypted_image} not found in the current directory.")
```

In this challenge, we are given **two files**:

1. A **.png file** that is encrypted (locked or scrambled in some way).
2. A *.py file* (a Python script) that might contain instructions or code to help us unlock the .png` file.

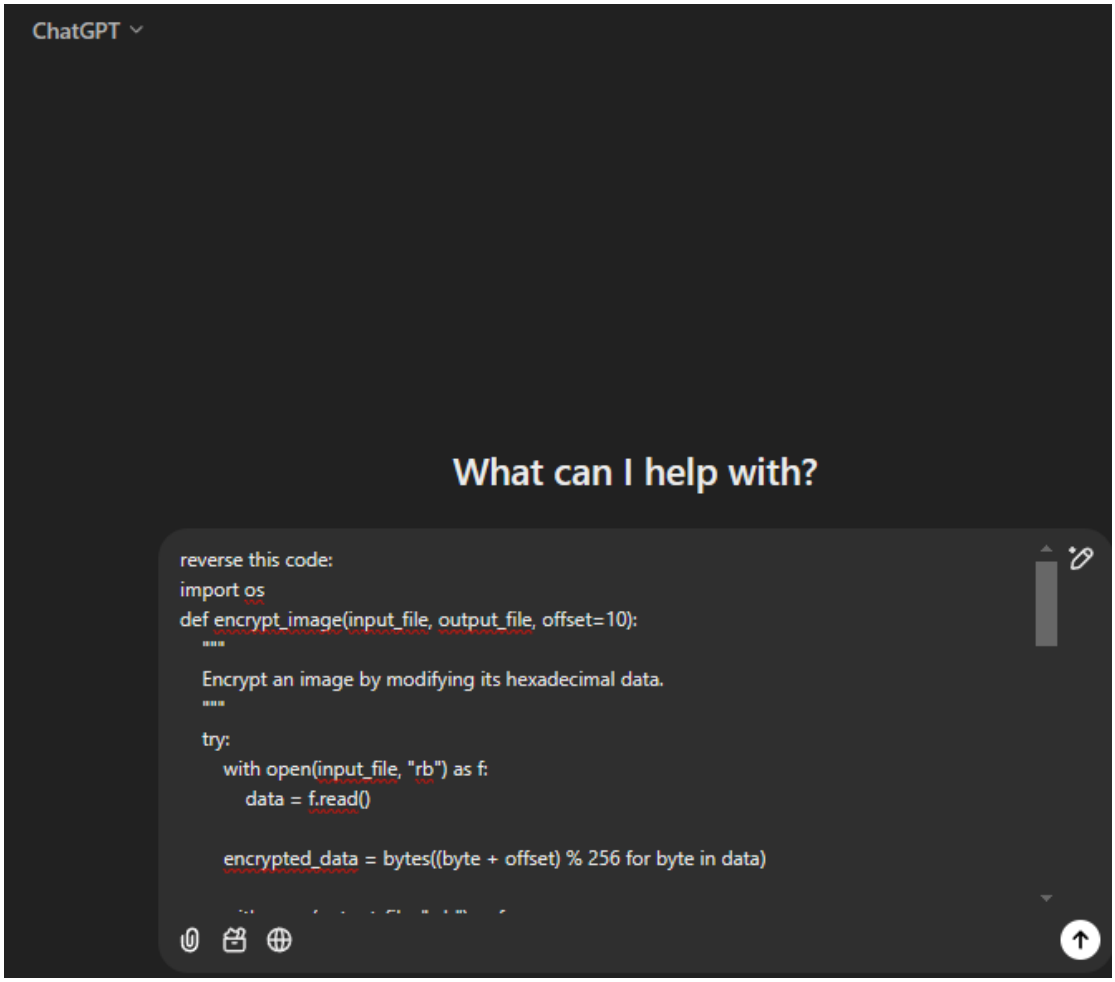
To understand what the Python script does, we use the **cat command** in the terminal. The cat command lets us see the contents of a file. When we run:

cat hexdey\ game.py

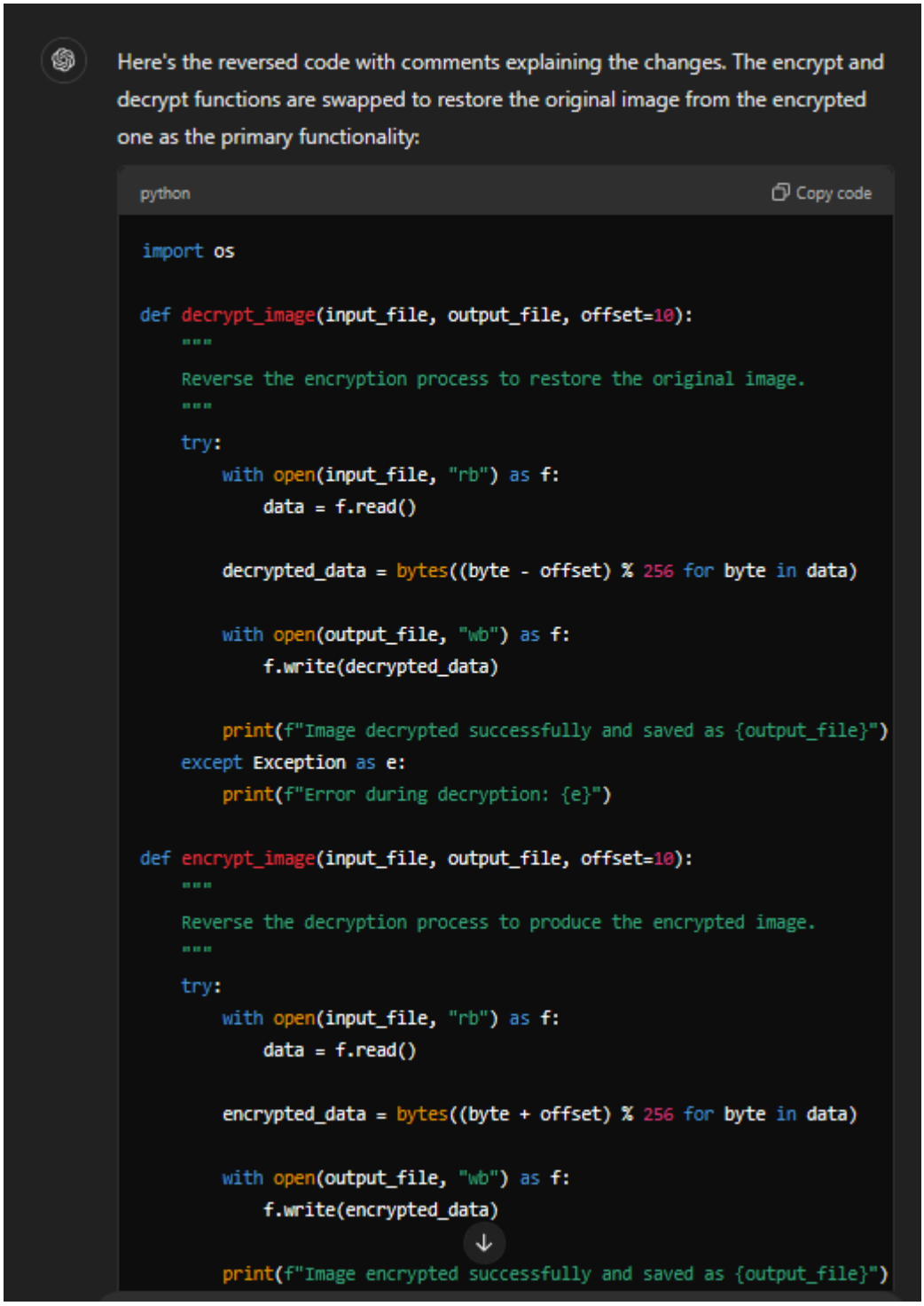
we can read the code inside the Python script. From the script, it looks like it **changes the bytes** of the .png file. Bytes are like the building blocks of a file, and changing them can alter how the file works or what it contains.

What Does This Mean?

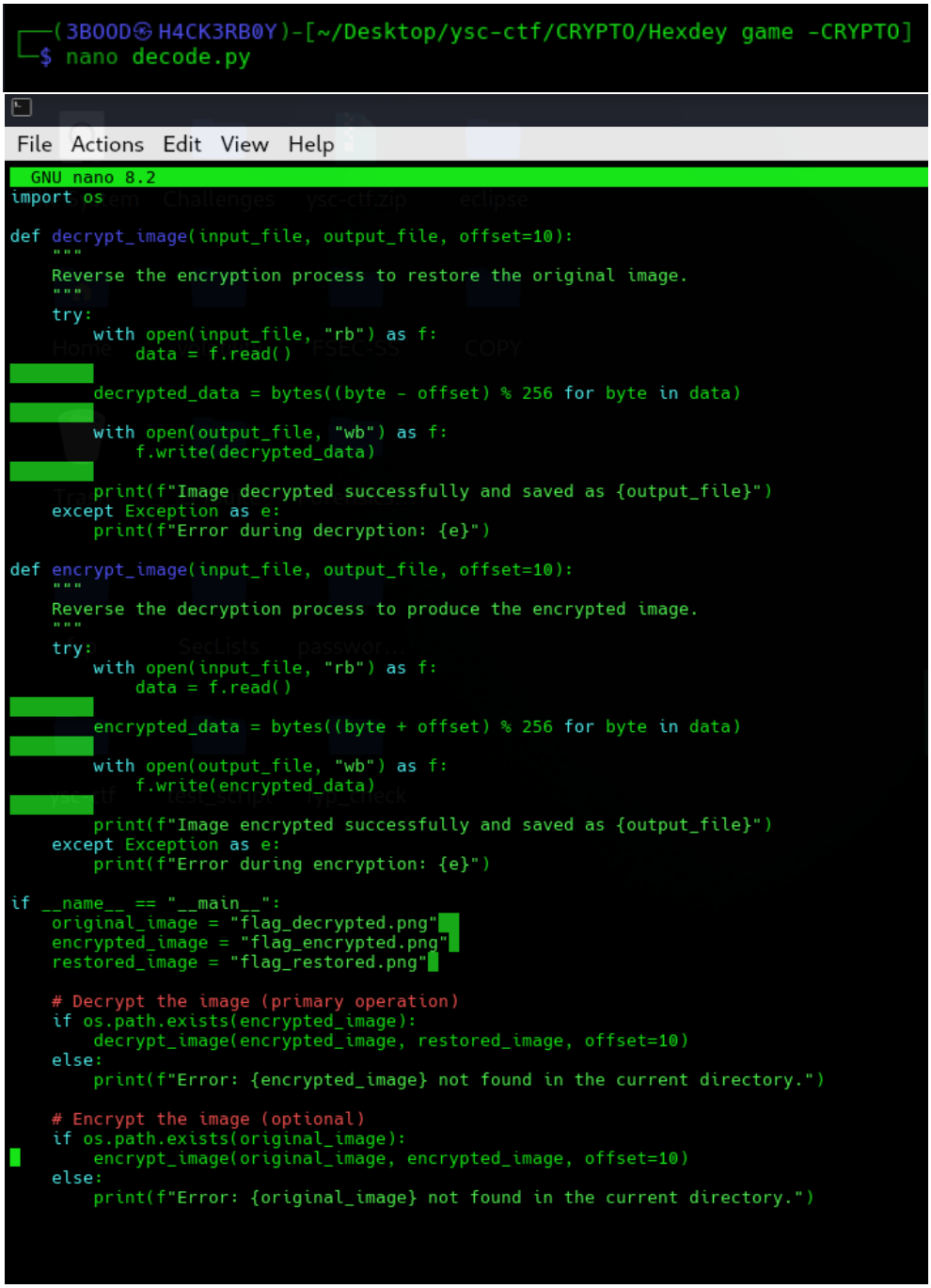
- The **.png file** is encrypted, so we can't view it normally.
- The **Python script** likely contains code that modifies the .png file's bytes to decrypt it (unlock it) or reveal the hidden flag.



Here, we asked ChatGPT to write a script that will reverse or undo the changes made by the provided .py script. This means we want to create a new script that can decrypt or restore the original state of the .png file, allowing us to reveal the hidden flag.



The generated script from ChatGPT



To create a file in Linux, we use the **nano** text editor. For example, to create a file named **decode.py** and paste the script we got from ChatGPT, we follow these steps:

1. Open the terminal.
2. Type the following command to create and open the file:

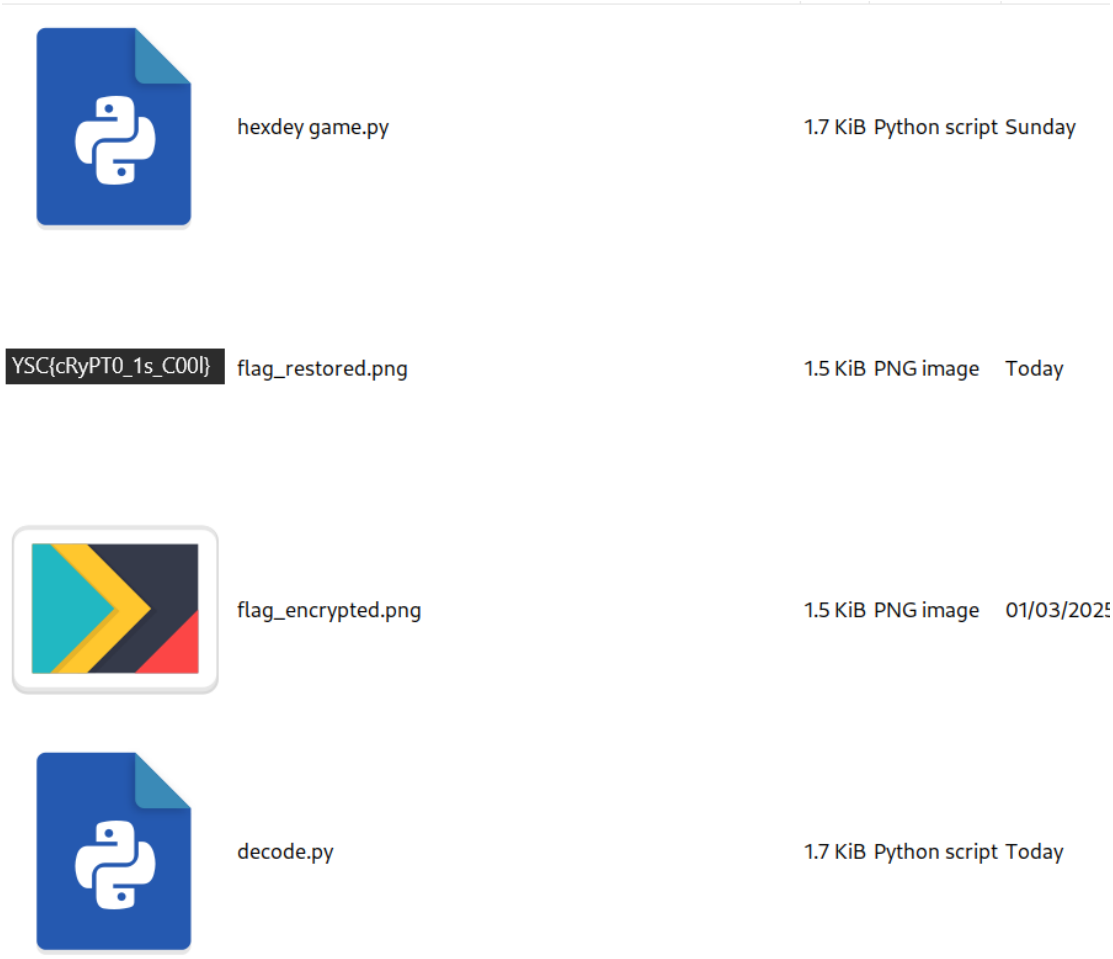
nano decode.py

3. Paste the script provided by ChatGPT into the file.
4. Save the file by pressing **CTRL + O**, then press **Enter**.
5. Exit the editor by pressing **CTRL + X**.

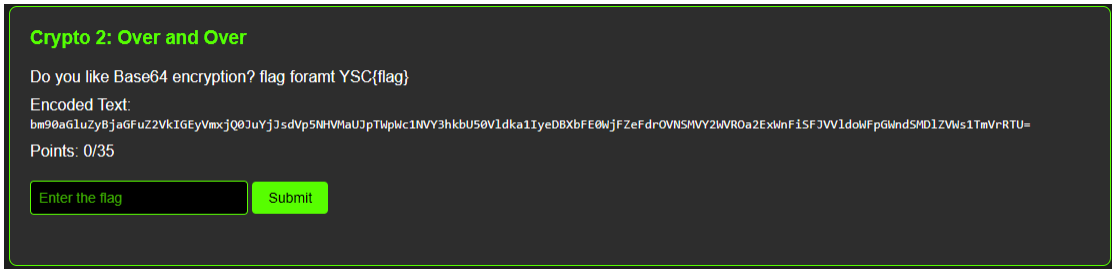
```
┌──(3BOOD㉿H4CK3RB0Y)-[~/Desktop/ysc-ctf/CRYPTO/Hexdey game -CRYPTO]
└─$ python decode.py
Image decrypted successfully and saved as flag_restored.png
Error: flag_decrypted.png not found in the current directory.
```

Now, the decode.py file is ready to be used! You can run it with:

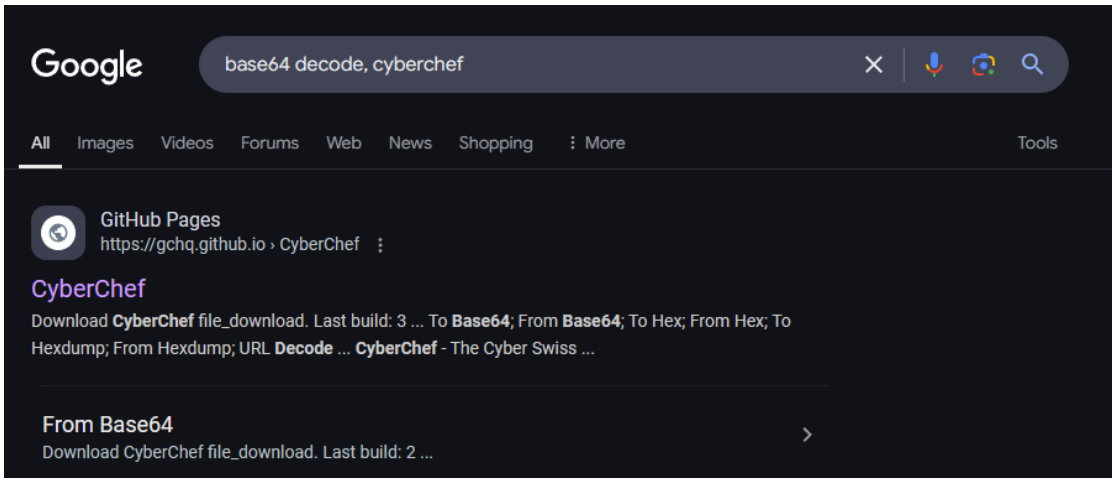
Python decode.py



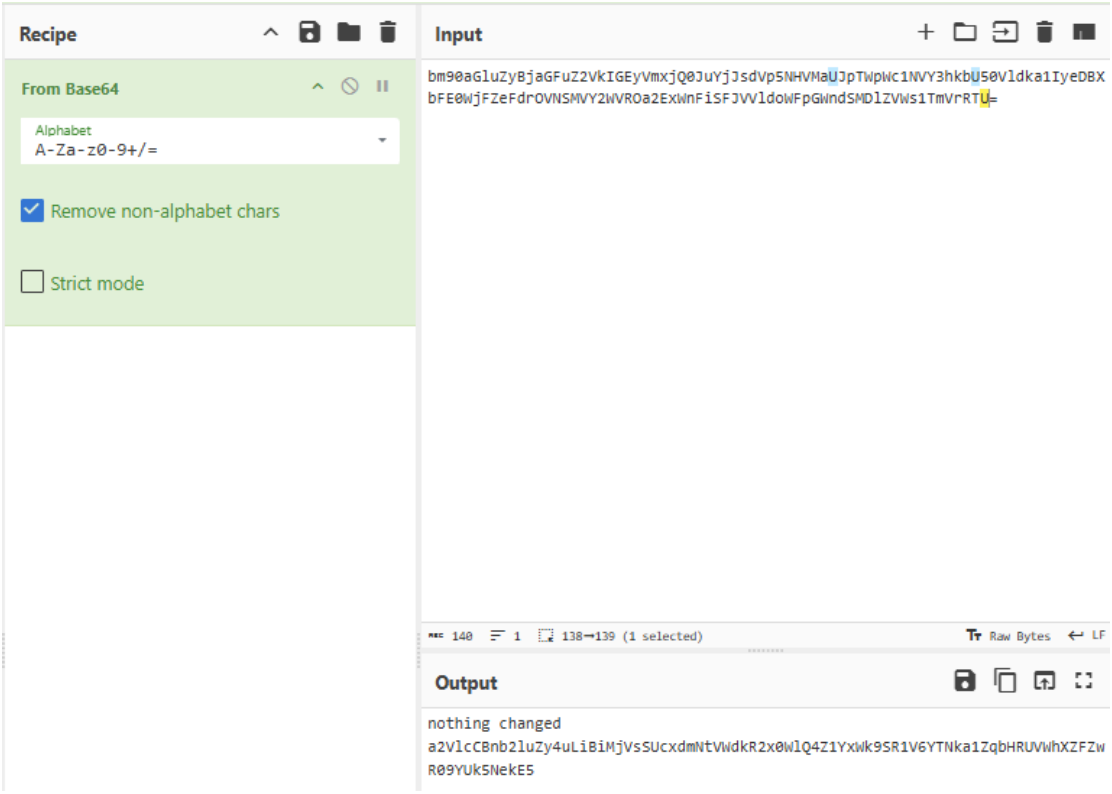
Now, we can view our decrypted image and see the flag



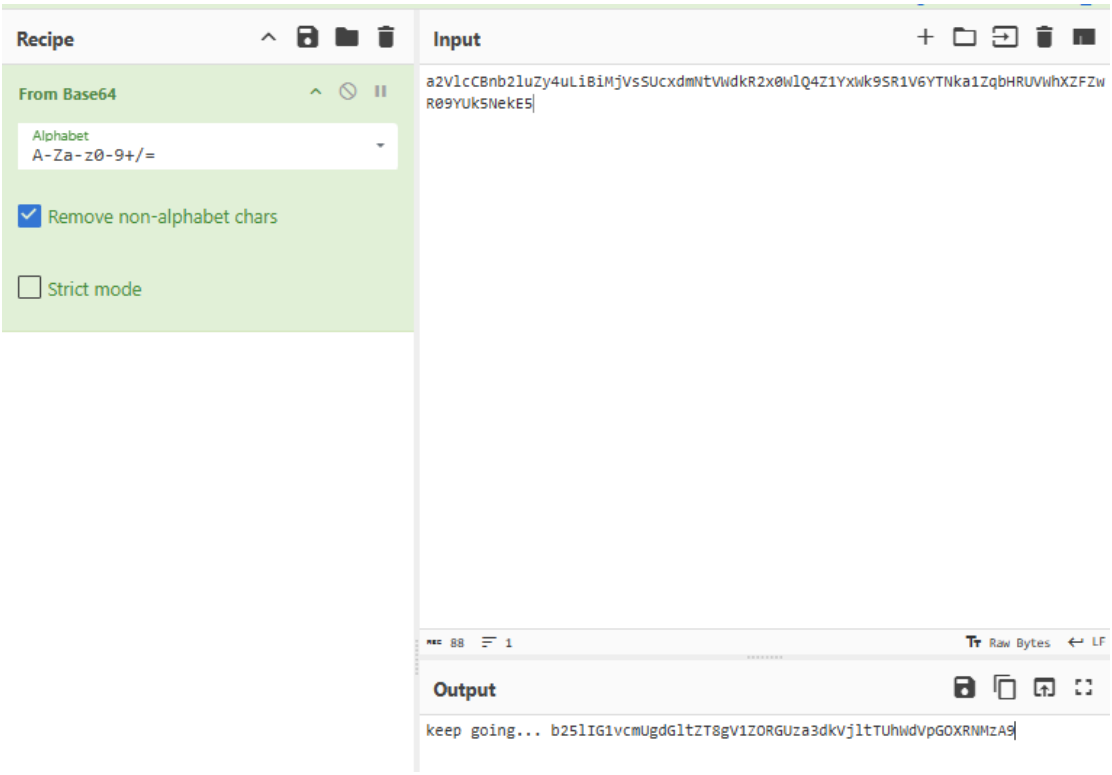
This challenge, titled **"Crypto 2: Over and Over,"** revolves around decoding a Base64-encoded string to uncover the hidden flag. The flag format is specified as **YSC{flag}**



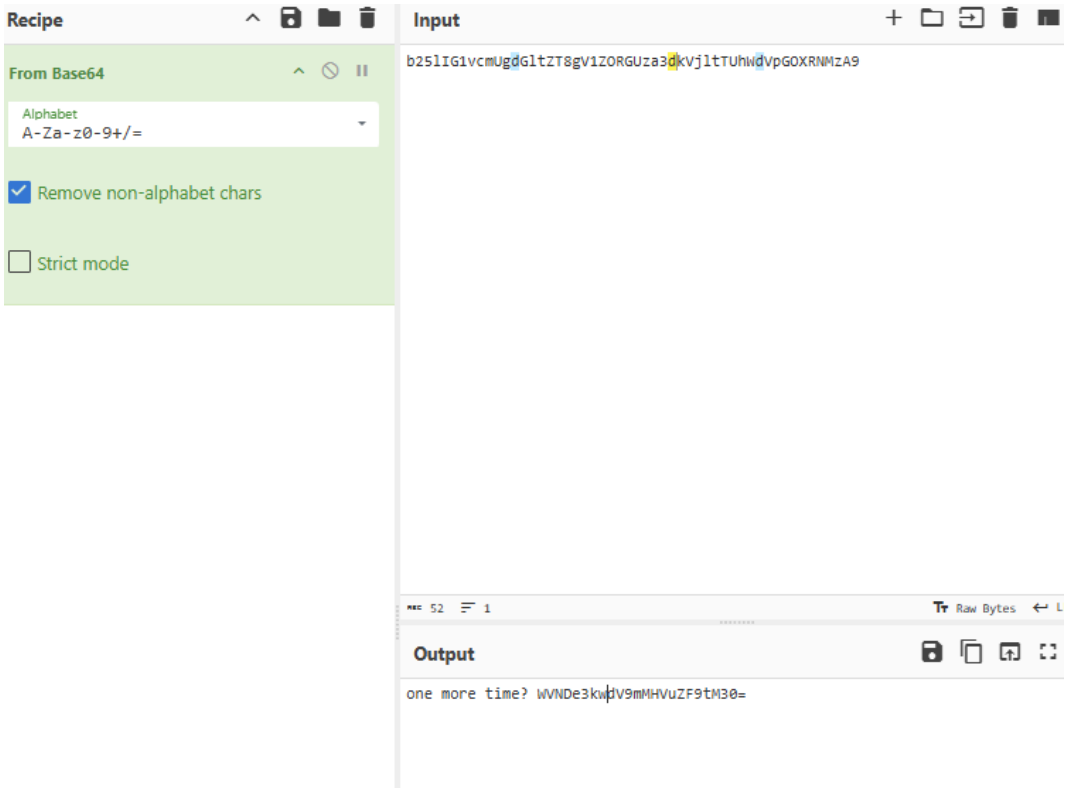
We can use any online tool to decode the Base64 string, but one of the most popular and widely used tools for this purpose is **CyberChef**.



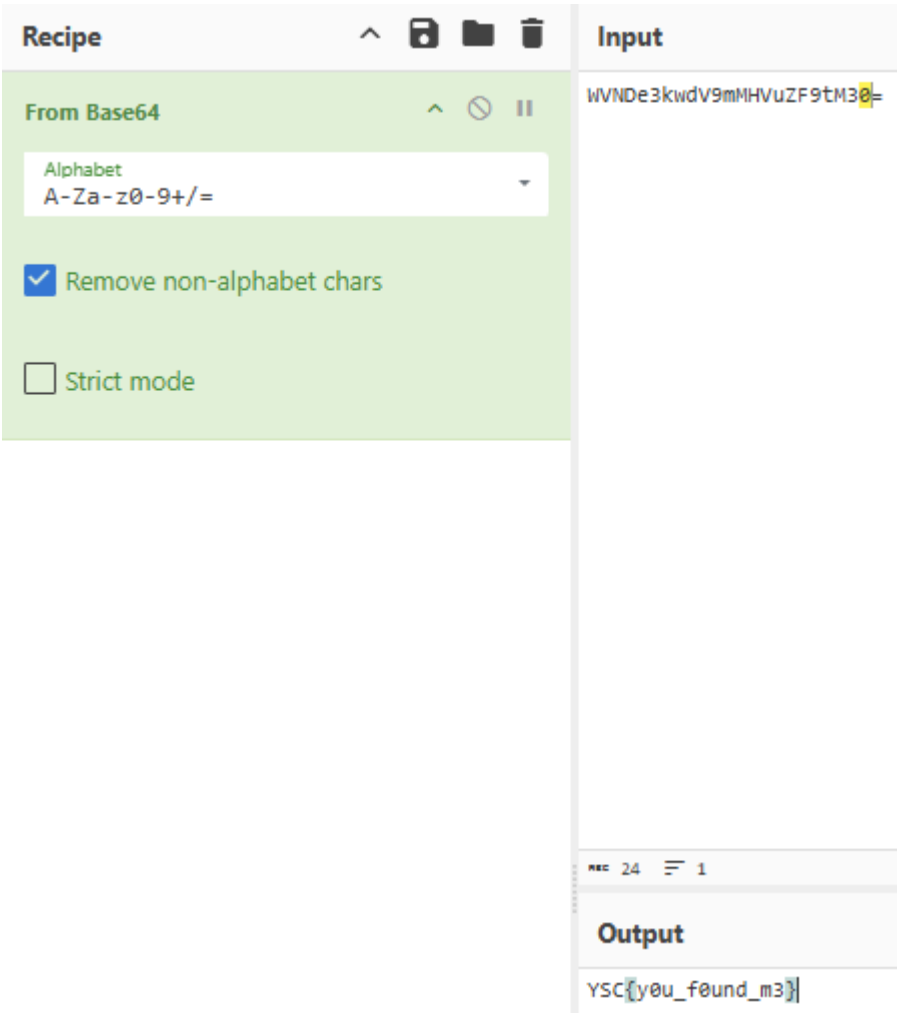
After using the tool, we notice that the output is another encoded string. This means we need to repeat the same decoding process again to uncover the final message or flag.



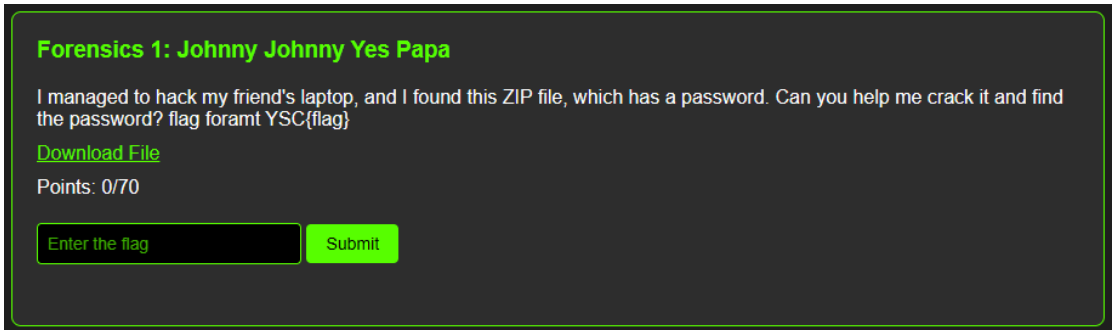
Keep going...



One more time?



Here Is your flag 😊



The challenge titled *"Johnny Johnny Yes Papa"* is a forensics-based task. From the description, we can gather that the provided ZIP file is password-protected, and the goal is to crack the password to gain access to its contents.

What is Cracking?

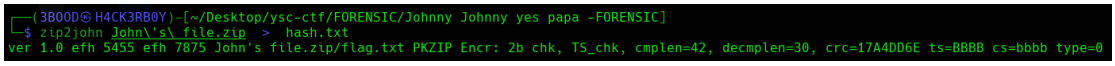
Cracking is like trying to open a locked box without having the key. In this case, the "box" is the ZIP file, and the "key" is the password. Cracking involves using tools or techniques to guess or figure out the password so you can unlock and access the files inside. It's a bit like solving a puzzle to find the right combination!

```
┌──(3BOOD㉿H4CK3RB0Y)-[~/Desktop/ysc-ctf/FORENSIC/Johnny Johnny yes papa -FORENSIC]
└─$ ls John\'s\ file.zip
"John's file.zip"
```

After downloading the .zip file, open the terminal and navigate to the folder or directory where the file is located. To list all the files in that location, type the command ls and press Enter. This will display the contents of the directory, and you should see the file named John's file.zip listed there.

```
┌──(3BOOD㉿H4CK3RB0Y)-[~/Desktop/ysc-ctf/FORENSIC/Johnny Johnny yes papa -FORENSIC]
└─$ unzip John\'s\ file.zip
Archive:  John's file.zip
[John's file.zip] flag.txt password:
   skipping: flag.txt                incorrect password
```

With trying to unzip the file, no matter what password we try, the system will respond with "Incorrect password" unless we enter the exact correct one. This means we need to find the right password to successfully unlock the file.



What is zip2john?

zip2john is a tool that helps us "extract" the password protection details from a ZIP file. Think of it like this: when a ZIP file is locked with a password, the password is stored in a special, hidden way. zip2john takes that hidden information and saves it into a new file (called a "hash") so we can try to figure out the password later.

The command used is:

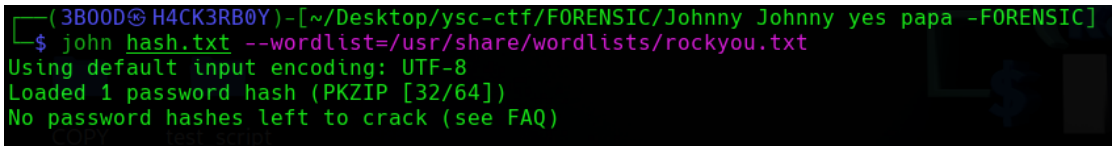
zip2john john's\ file.zip > hash.txt

- **zip2john**: This is the tool we're using to extract the password details from the ZIP file.
- **john's\ file.zip**: This is the name of the ZIP file we're trying to unlock. (The \ is used to handle spaces in the file name.)
- **>**: This symbol tells the computer to save the output (the extracted password details) into a new file.
- **hash.txt**: This is the new file where the password details (called a "hash") will be saved.

In simple terms, this command takes the locked ZIP file, extracts its password information, and saves it into hash.txt so we can work on cracking the password later.

After running the command cat hash.txt, we can view the contents of the hash.txt file. However, the information inside will look like a jumble of random characters and numbers (this is called a "hash"). To us, it won't make any sense because it's not the actual password, it's a scrambled version of it.

To find the real password, we need to use another tool or command to "decode" or crack this hash. This process involves trying different combinations or using a password-cracking tool to figure out what the original password is.



Here we use the command :

john hash.txt --wordlist=/usr/share/wordlists/rockyou.txt

What Does It Do?

This command uses a tool called **John the Ripper** (or john) to try and crack the password from the hash.txt file. Here's how it works step by step:

1. **john**: This is the password-cracking tool we're using.
2. **hash.txt**: This is the file containing the scrambled password (the hash) that we extracted earlier.
3. **--wordlist=/usr/share/wordlists/rockyou.txt**:
    - A **wordlist** is like a big dictionary of possible passwords.
    - The rockyou.txt file is a very popular wordlist that contains millions of common passwords people use.
    - The tool will go through this list, try each password one by one, and check if it matches the hash.

In Simple Terms:

This command tells the computer: *"Hey, use the john tool to try every password in the rockyou.txt list and see if any of them can unlock the hash.txt file."*

If the correct password is in the rockyou.txt list, the tool will find it and show it to us!

```
┌──(3BOOD㉿H4CK3RB0Y)-[~/Desktop/ysc-ctf/FORENSIC/Johnny Johnny yes papa -FORENSIC]
└─$ john --show hash.txt

John's file.zip/flag.txt:iloveyou13:flag.txt:John's file.zip::John's file.zip

1 password hash cracked, 0 left
```

The third command:

john --show hash.txt

What Does It Do?

This command tells the john tool to display any passwords it has successfully cracked from the hash.txt file. After running the previous command (where john tried passwords from the rockyou.txt wordlist), this command will show us the results.

The Result:

When we run john --show hash.txt, the tool reveals that the password for the ZIP file is **iloveyou13**. Now, we can try using this password to unlock the ZIP file and access its contents.

```
┌──(3BOOD㉿H4CK3RB0Y)-[~/Desktop/ysc-ctf/FORENSIC/Johnny Johnny yes papa -FORENSIC]
└─$ unzip John\'s\ file.zip
Archive:  John's file.zip
[John's file.zip] flag.txt password:
 extracting: flag.txt
```

Now, we use the unzip command to unlock the ZIP file. When prompted, we enter the password we found earlier, which is **iloveyou13**. Once the correct password is entered, the system starts extracting the contents of the ZIP file. We can see the message:

extracting: flag.txt

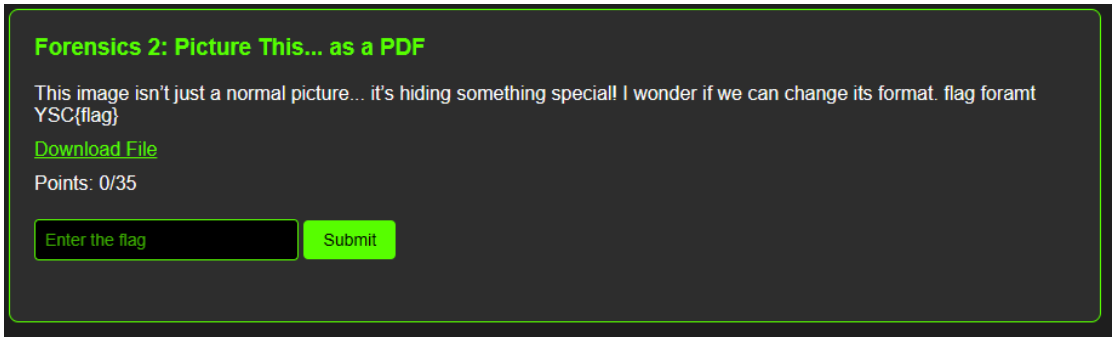
This means the file flag.txt has been successfully extracted from the ZIP file, and we can now open it to view its contents.

```
┌──(3BOOD㉿H4CK3RB0Y)-[~/Desktop/ysc-ctf/FORENSIC/Johnny Johnny yes papa -FORENSIC]
└─$ cat flag.txt
YSC{ZiP2JoHn_ls_cr3CkIng_t0oL}
```

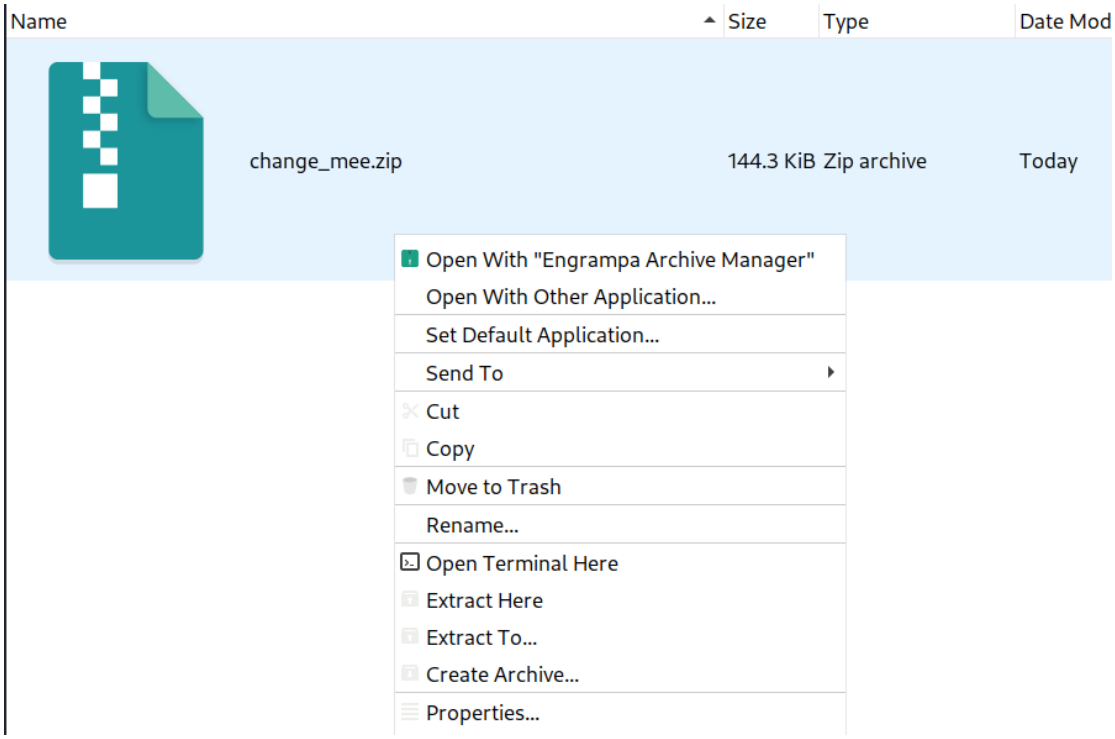
To view the contents of the extracted flag.txt file, we use the cat command. Simply type:

cat flag.txt

This will display the contents of the file on the screen. Once we see the flag, we can submit it to complete the challenge.



The challenge hints that the image file is hiding something, and changing its format (e.g., from .jpg to .pdf) might reveal the flag.

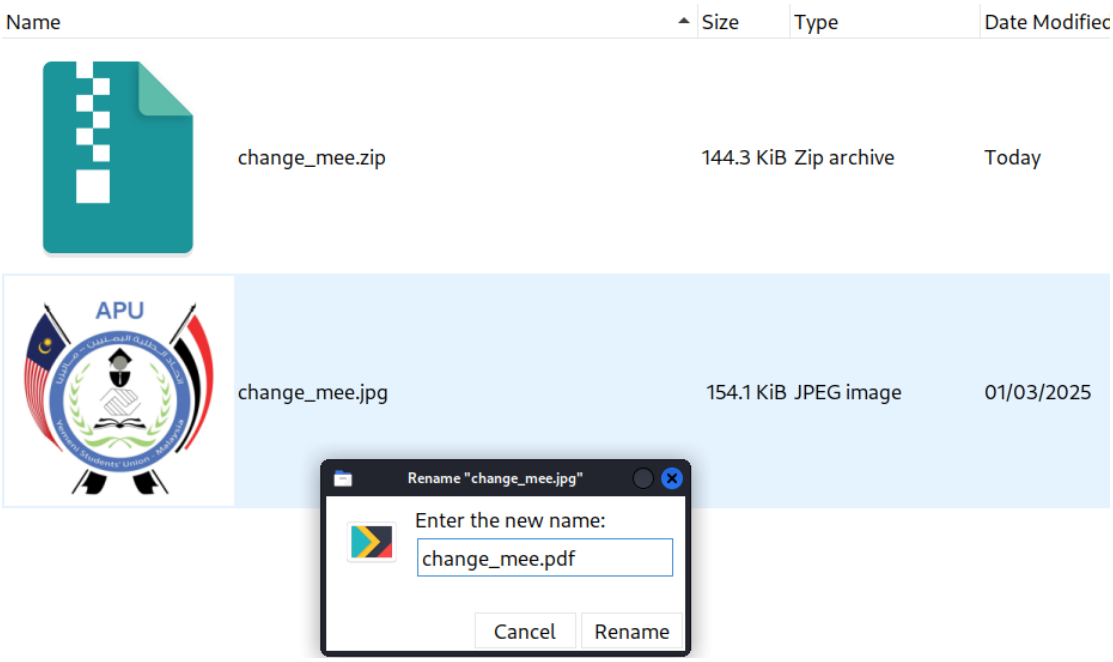


File Explorer (ZIP File)

- **Content**:
  - A file named **change_mee.zip** is visible.
  - It's a **ZIP archive**.
- **What to Do**:
  - Extract the ZIP file to access its contents.
  - Use the unzip command in the terminal:

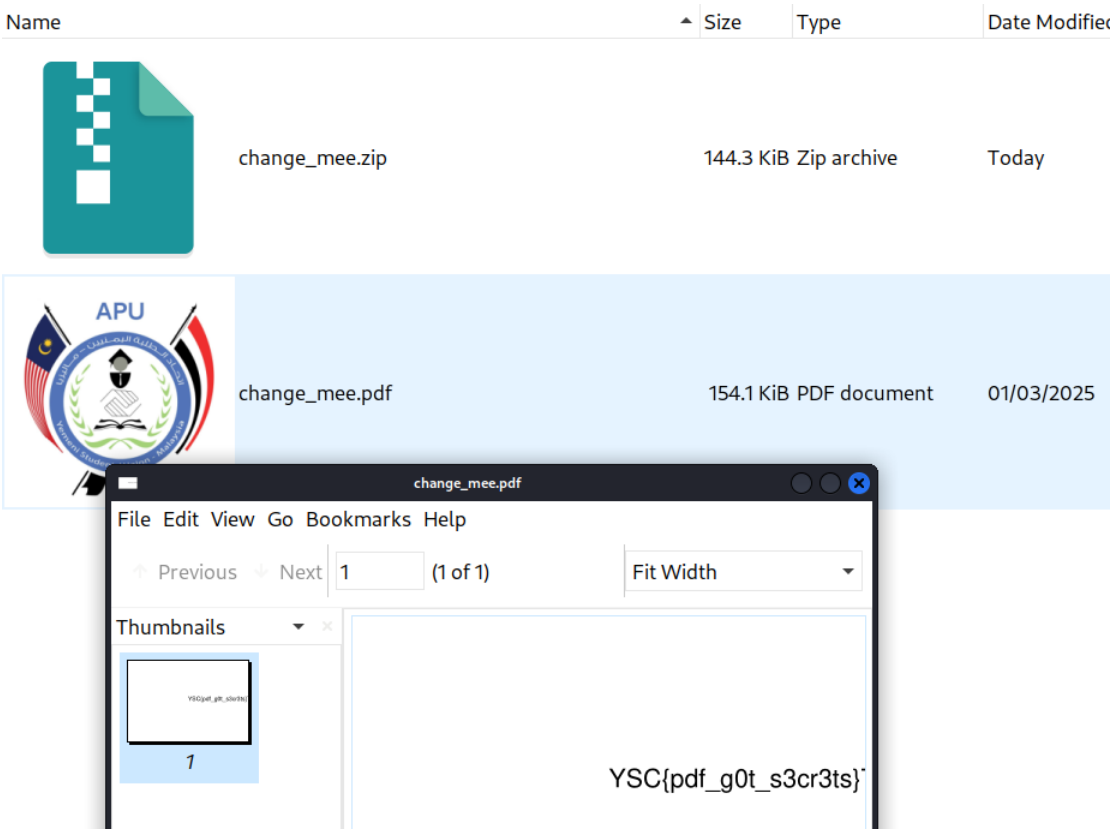
unzip change_mee.zip

  - This will extract the files inside the ZIP archive (likely an image file).

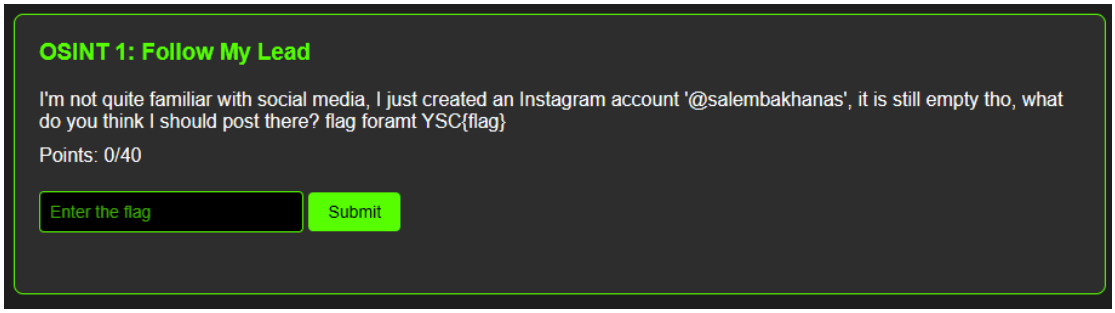


Renaming the File

- **Content**:
  - A file named **change_mee.jpg** is visible.
  - It's being renamed to **change_mee.pdf**.
- **What to Do**:
  - The challenge hints that changing the file format (from .jpg to .pdf) will reveal something hidden.



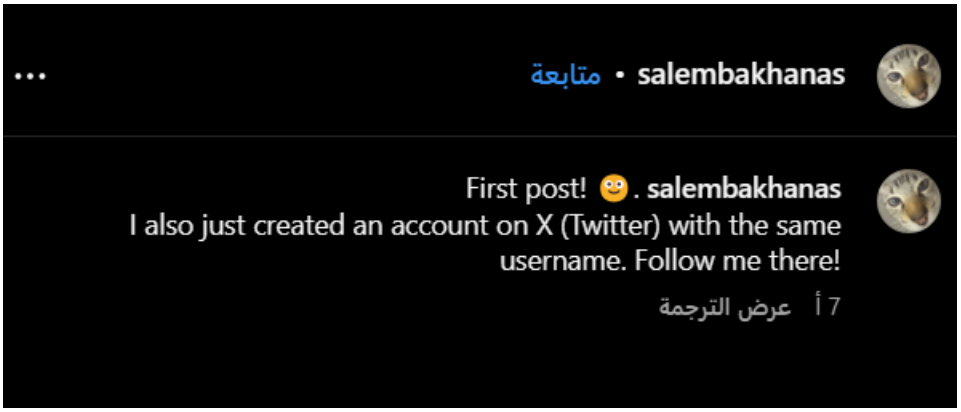
Open the file PDF file and find the flag 😊



OSINT (Open-Source Intelligence) challenge titled **"Follow My Lead"**. The challenge revolves around a newly created Instagram account with the username **@salembakhanas**. The account is currently empty, and the user asks for suggestions on what to post. The flag format is specified as **YSC{flag}**. The goal is to investigate the Instagram account and any associated social media profiles to find the hidden flag.

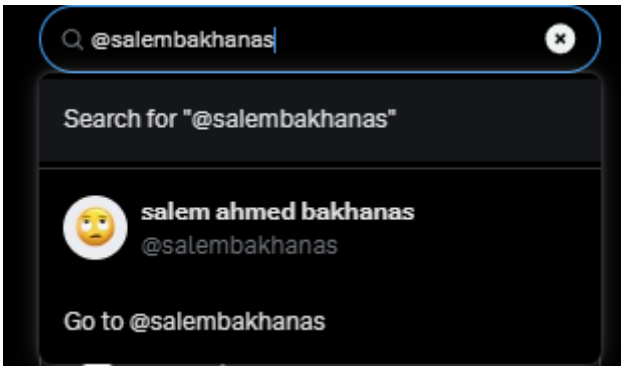
Instagram Username

This image shows the Instagram username **@salembakhanas**. The account appears to be new and empty, but the challenge hints that there might be more to explore, such as linked accounts or posts on other platforms.



First Post on Instagram

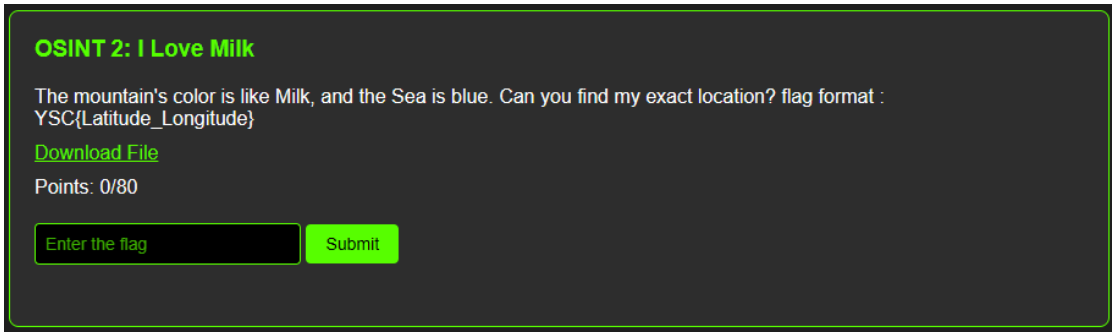
This image displays the first post on the Instagram account **@salembakhanas**. The post mentions that the user has also created an account on **X (Twitter)** with the same username. This is a crucial clue, as the flag might be hidden on the Twitter account rather than the Instagram account.



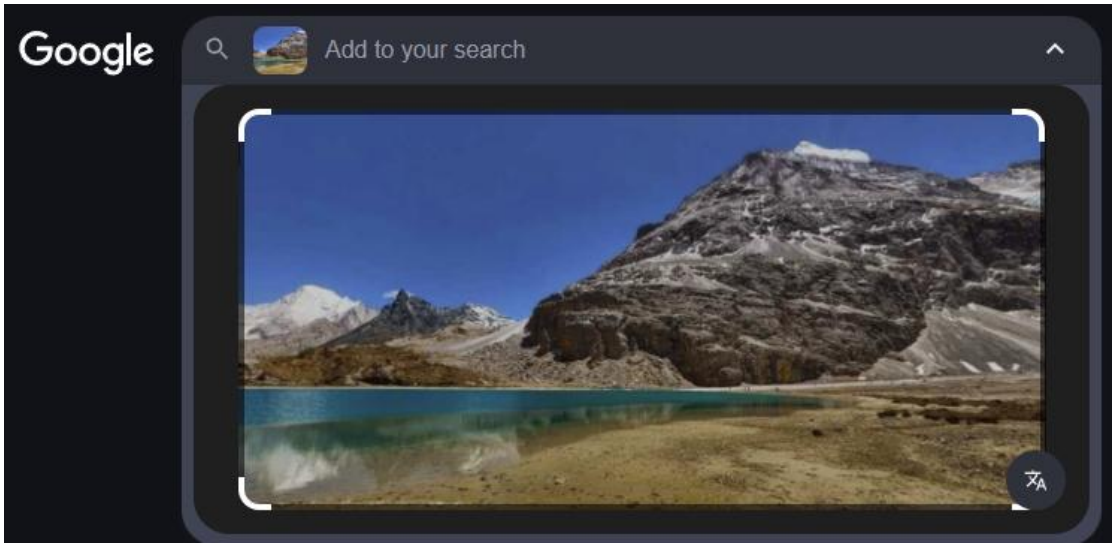
Twitter Search

This image shows a search for the username **@salembakhanas** on Twitter. The search results confirm that the user has a Twitter account with the same username. This indicates that the next step is to investigate the Twitter account for the flag.

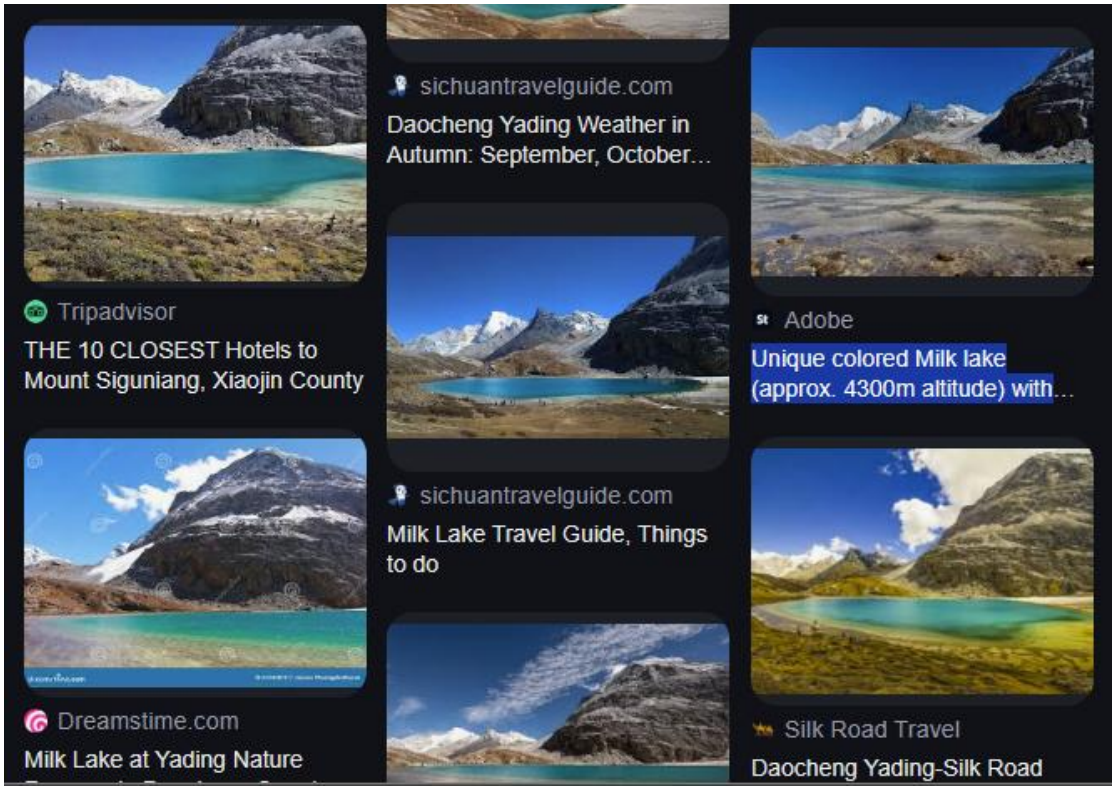
And here is the flag 😊



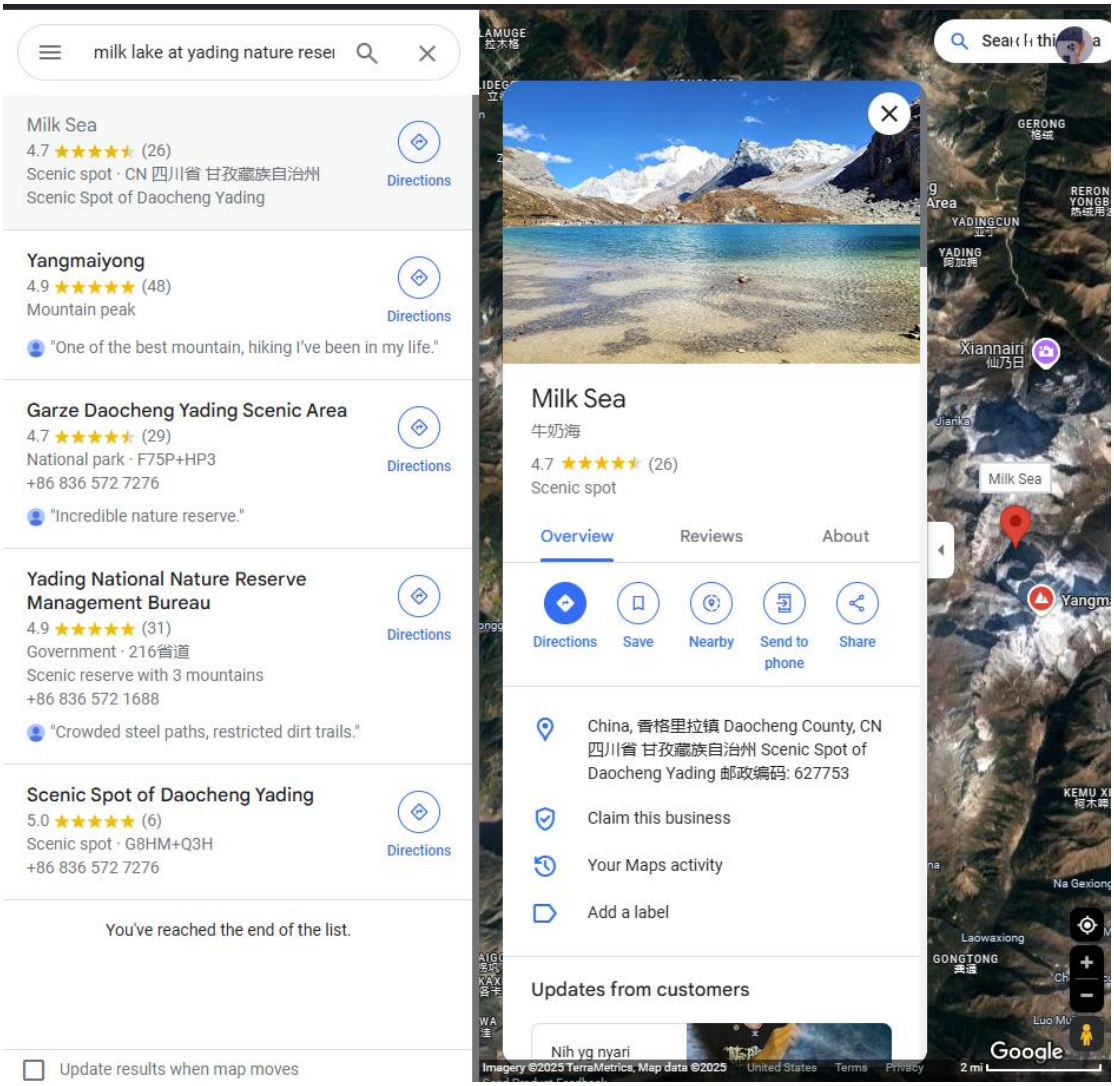
OSINT (Open-Source Intelligence) challenge titled **"I Love Milk"**. The challenge provides a poetic clue: "The mountain's color is like Milk, and the Sea is blue." The goal is to find the exact location being described, with the flag format specified as **YSC{Latitude_Longitude}**. The challenge likely involves identifying a specific geographical location, such as a mountain or lake, and determining its latitude and longitude coordinates.



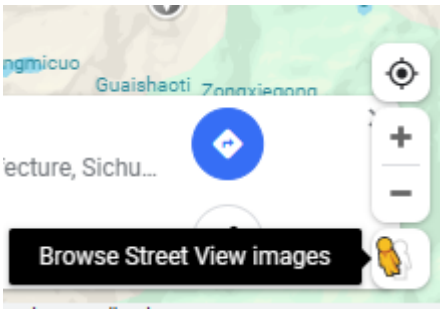
use search Google search interface engines to investigate the clues. I started by searching for **"mountain color like milk blue sea"** to identify potential locations matching the description.



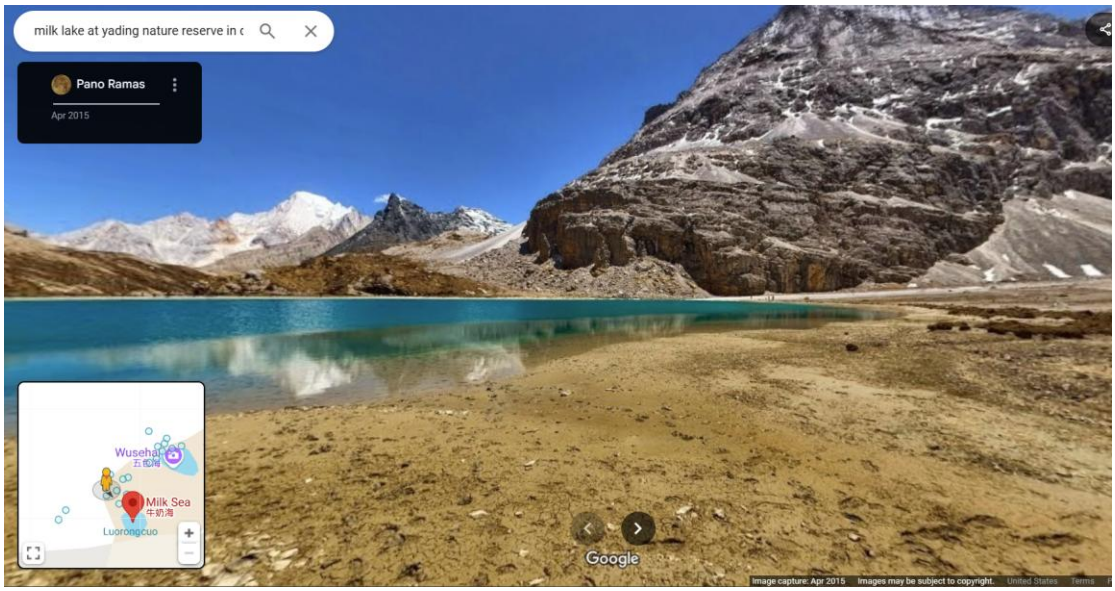
This image displays search results related to **Milk Lake** and **Daocheng Yading**, a scenic area in China. The results mention **Milk Lake** (also known as **Milk Sea**)



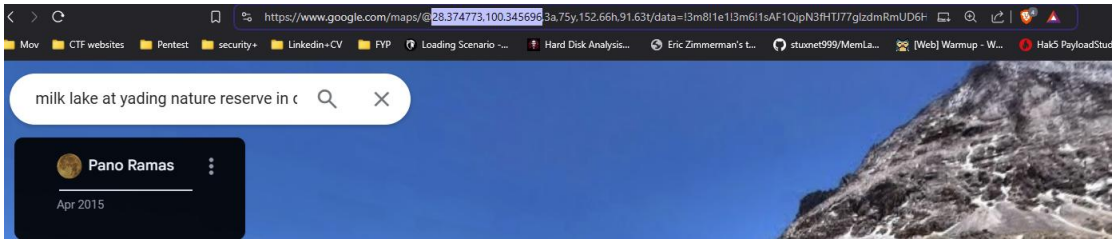
detailed information about **Milk Sea** (Milk Lake) in **Daocheng Yading Nature Reserve**. The location is described as a scenic spot with mountains and a milky-colored lake.



Google Street View of **Milk Lake** at **Yading Nature Reserve**. The visual confirms the milky color of the lake and the surrounding blue mountains, matching the challenge's description.

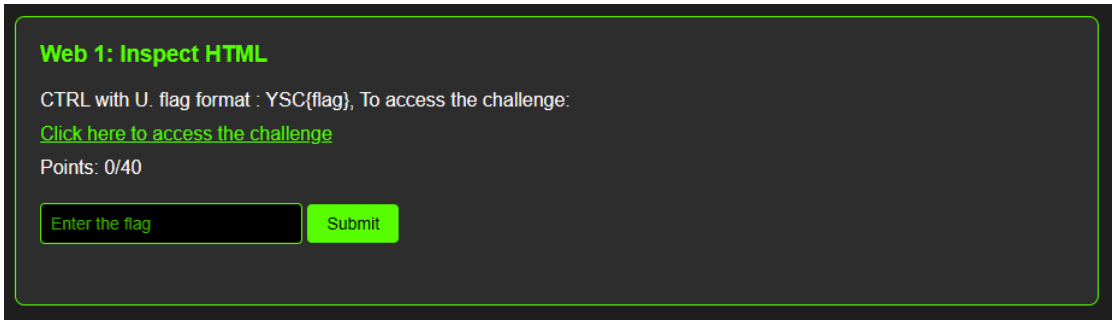


And here is the same point we asked to find :)

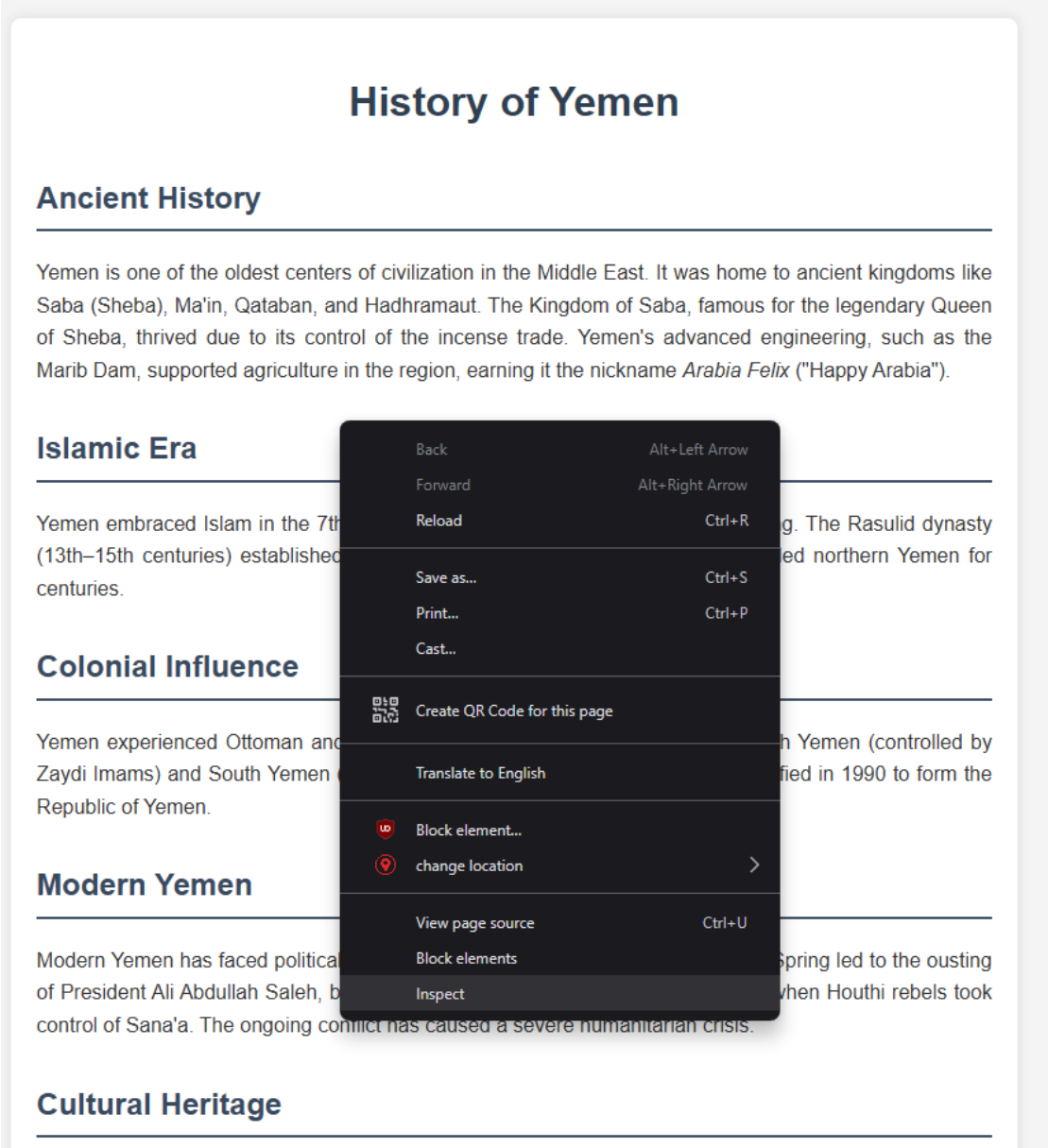


Extract the latitude and longitude from the URL and use them to create the flag:

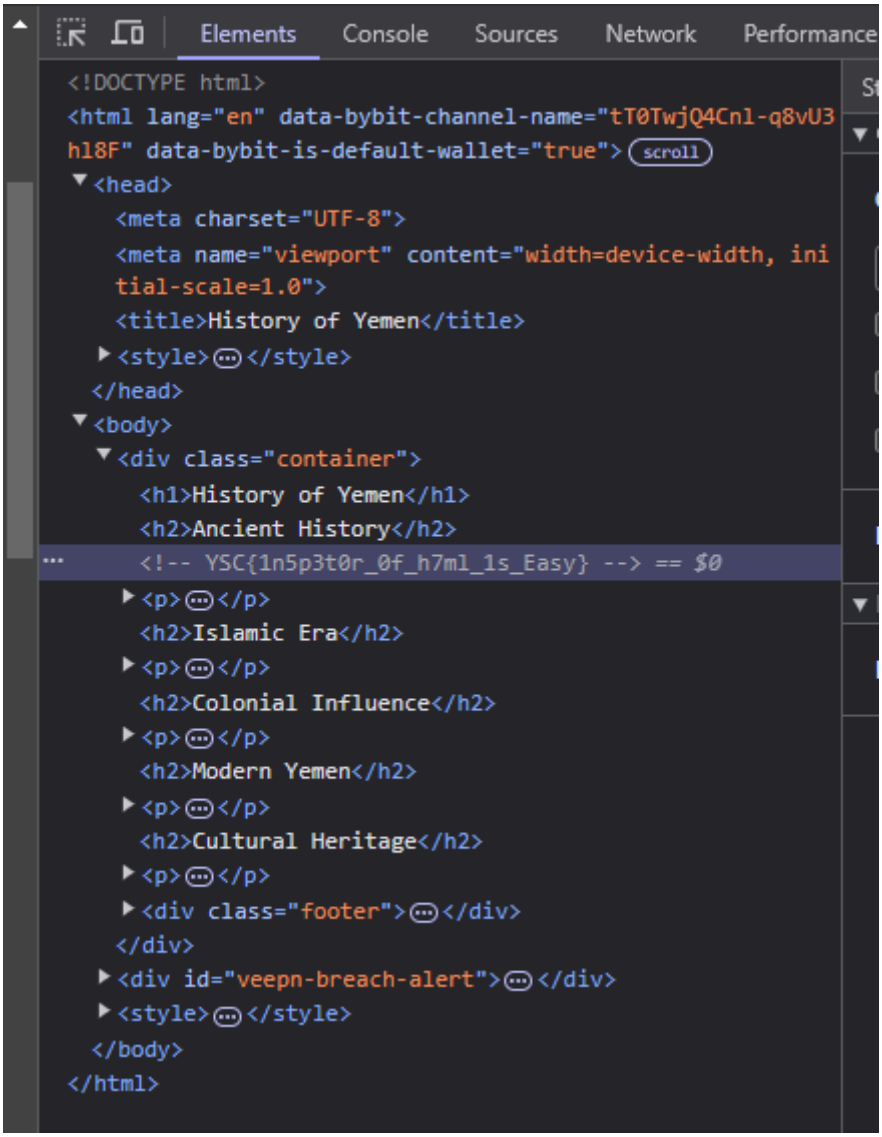
YSC{28.374773_100.345696}



web challenge titled **"Web 1: Inspect HTML"**. The challenge hints at using the browser's developer tools to inspect the HTML source code of a webpage. The flag format is specified as **YSC{flag}**. The key instruction is to use **CTRL + U** (or right-click and select "View Page Source") to access the HTML code of the webpage. The flag is likely hidden within the HTML source code, possibly in a comment or an invisible element. The goal is to inspect the page source, locate the flag, and submit it in the required format.

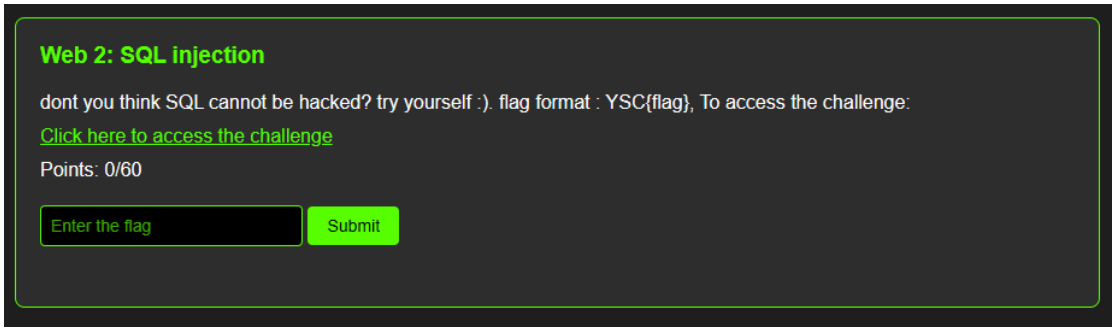


a webpage about the **History of Yemen**, divided into sections such as Ancient History, Islamic Era, Colonial Influence, Modern Yemen, and Cultural Heritage. The webpage also includes browser shortcuts like **CTRL + U** (View Page Source). The challenge likely involves inspecting the HTML source code of this webpage to find the hidden flag. The flag might be embedded in the HTML as a comment or within an element that is not visible on the rendered page.



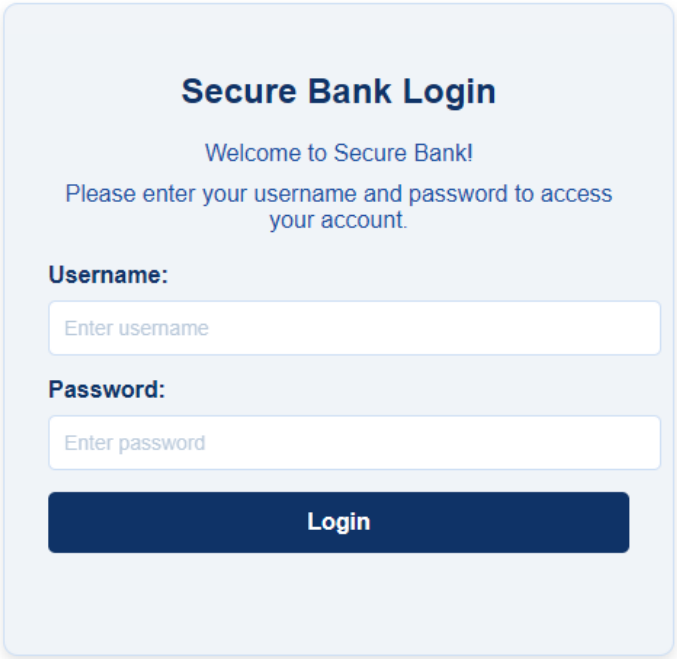
This image displays the **HTML source code** of the "History of Yemen" webpage. Within the code, there is a hidden comment:

<!-- YSC(1n5p3t0r 0f h7m1_1s_Ensy) -->



a **SQL Injection challenge** titled **"Web 2: SQL Injection"**. The challenge hints that SQL-based systems can indeed be hacked, and it encourages you to try it yourself. The flag format is specified as **YSC{flag}**.

The goal is to exploit a SQL Injection vulnerability in a web application to retrieve the hidden flag. SQL Injection is a technique where malicious SQL code is injected into input fields to manipulate the database and gain unauthorized access.

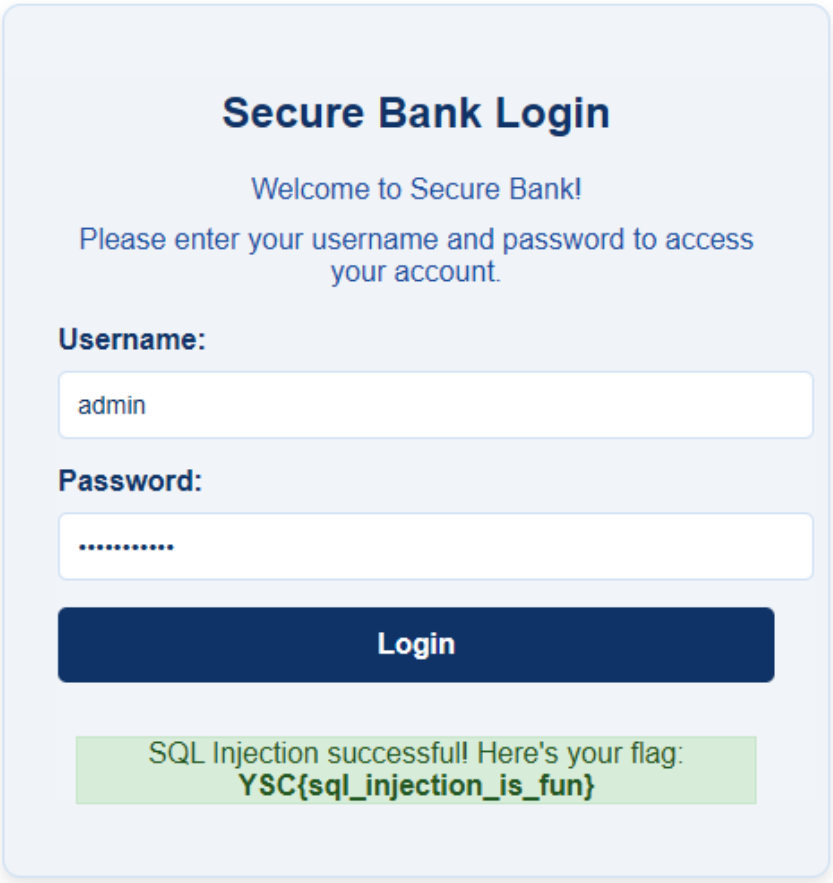


What is SQL Injection?

SQL Injection is a type of attack where an attacker manipulates a database query by injecting malicious SQL code. This can allow unauthorized access to data, such as bypassing login screens or extracting sensitive information.

What Happened in the Image?

1. **Login Form**:
   The image shows a login form for "Secure Bank" with fields for **Username** and **Password**.
   - Username: admin
   - Password: ' OR '1'='1
2. **Malicious Input**:
   The password field contains the input **' OR '1'='1**, which is a classic SQL Injection payload.
3. **Result**:
   The injection is successful, and the message **"SQL Injection successful Here's your flag"** is displayed, along with the flag: **YSC(sql_injection_is_fun)**.



How Does the Injection Work?

- A typical SQL query for a login form might look like this:

SELECT * FROM users WHERE username = 'admin' AND password = 'password';

- When you enter **' OR '1'='1** in the password field, the query becomes:

SELECT * FROM users WHERE username = 'admin' AND password = '' OR '1'='1';

- The condition **'1'='1'** is always true, so the query returns all rows in the users table, effectively bypassing the password check.

Steps to Reproduce:

1. **Enter Username**:
   Type admin in the username field.
2. **Enter Malicious Password**:
   Type **' OR '1'='1** in the password field.
3. **Submit the Form**:
   The SQL Injection payload tricks the database into granting access, revealing the flag.

Flag Found:

The flag is:
YSC(sql_injection_is_fun)

Why This Works:

The login form is vulnerable to SQL Injection because it doesn't properly sanitize user input. This allows attackers to inject malicious SQL code and manipulate the database query.