Exercise 1 Interendec

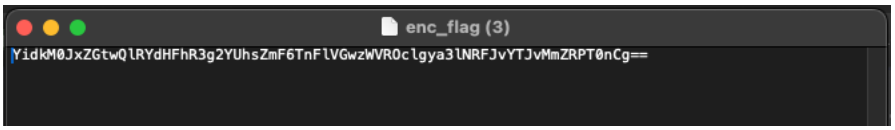
We open the Encrypted File

The challenge provides a file named enc_flag.

We open it using a text editor.

The file contains a Base64-encoded string, because:

- It contained uppercase letters (A-Z), lowercase letters (a-z), numbers (0-9), and symbols like =, /, and +.
- The string was long and continuous, with no spaces or special characters that would indicate a different encryption method.
- The presence of = at the end was a strong clue. In Base64 encoding, = is often used as padding when the data does not perfectly align into 3-byte chunks.

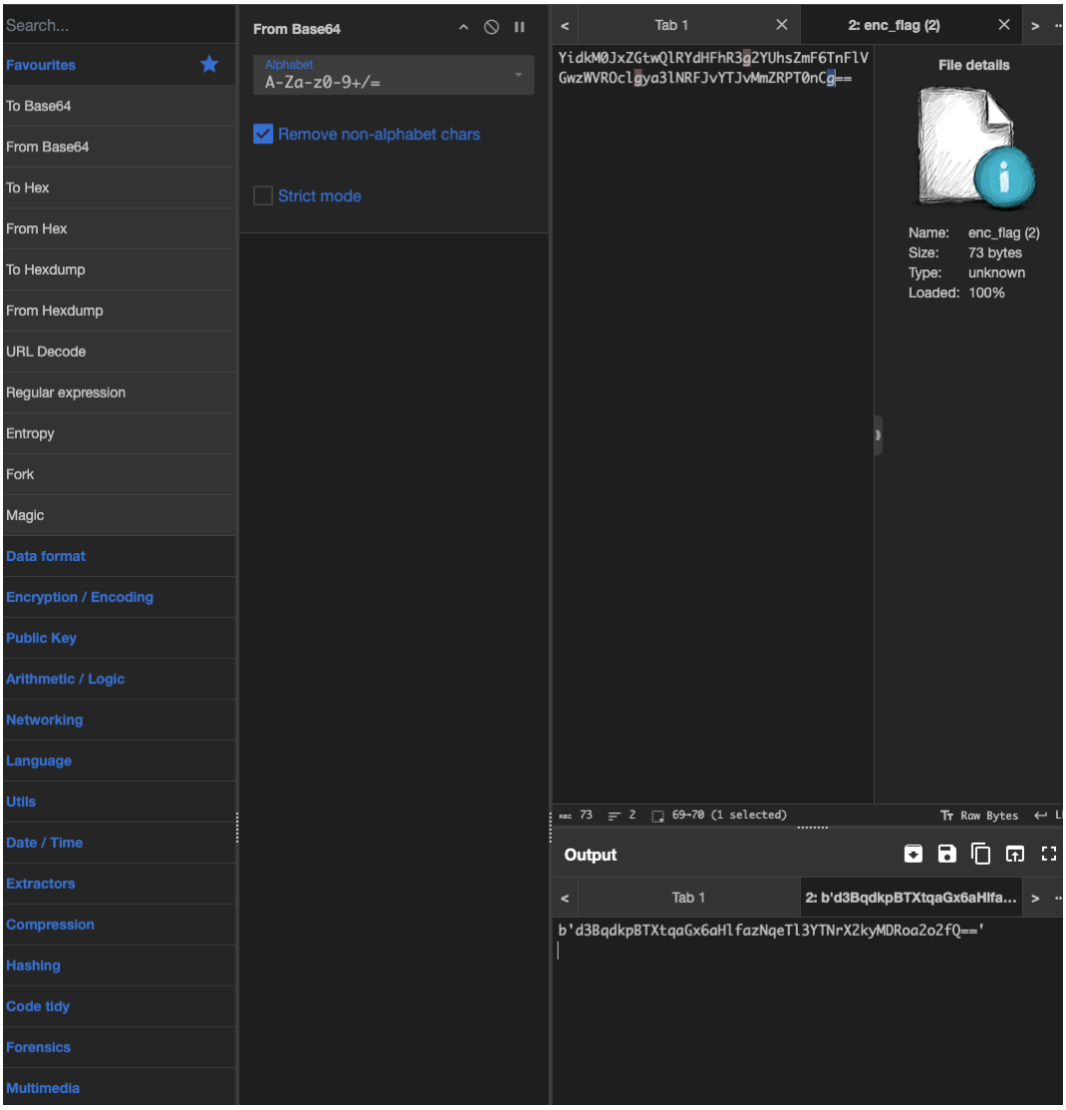


To decode the text, we used CyberChef, a very useful tool for this type of analysis.

We loaded the text into CyberChef and used the “From Base64” function to decode it.

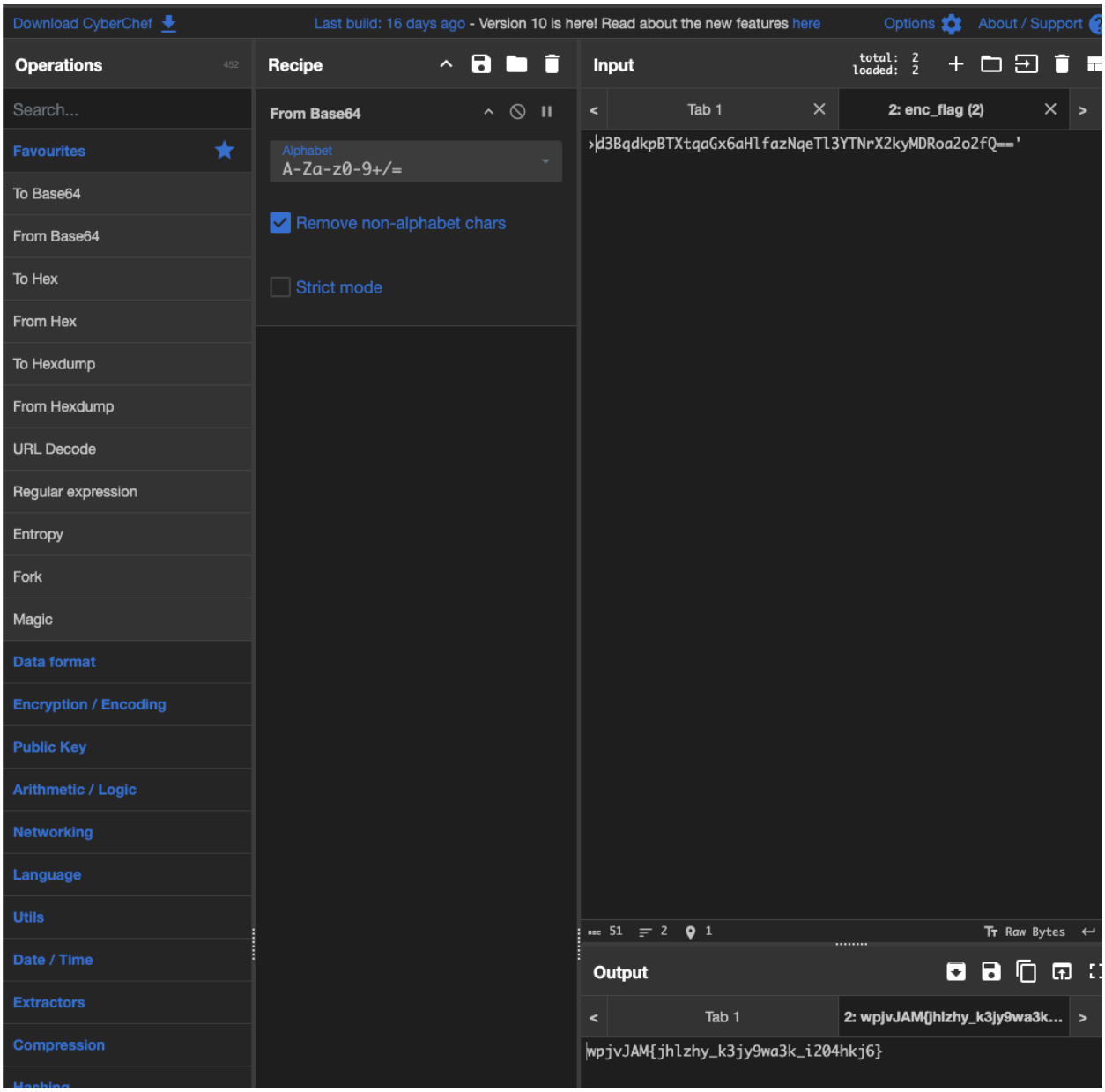
After doing this, we obtained a new encrypted string, indicating that there was another encryption layer.

My next step was to analyze what type of encryption was used in this new string.

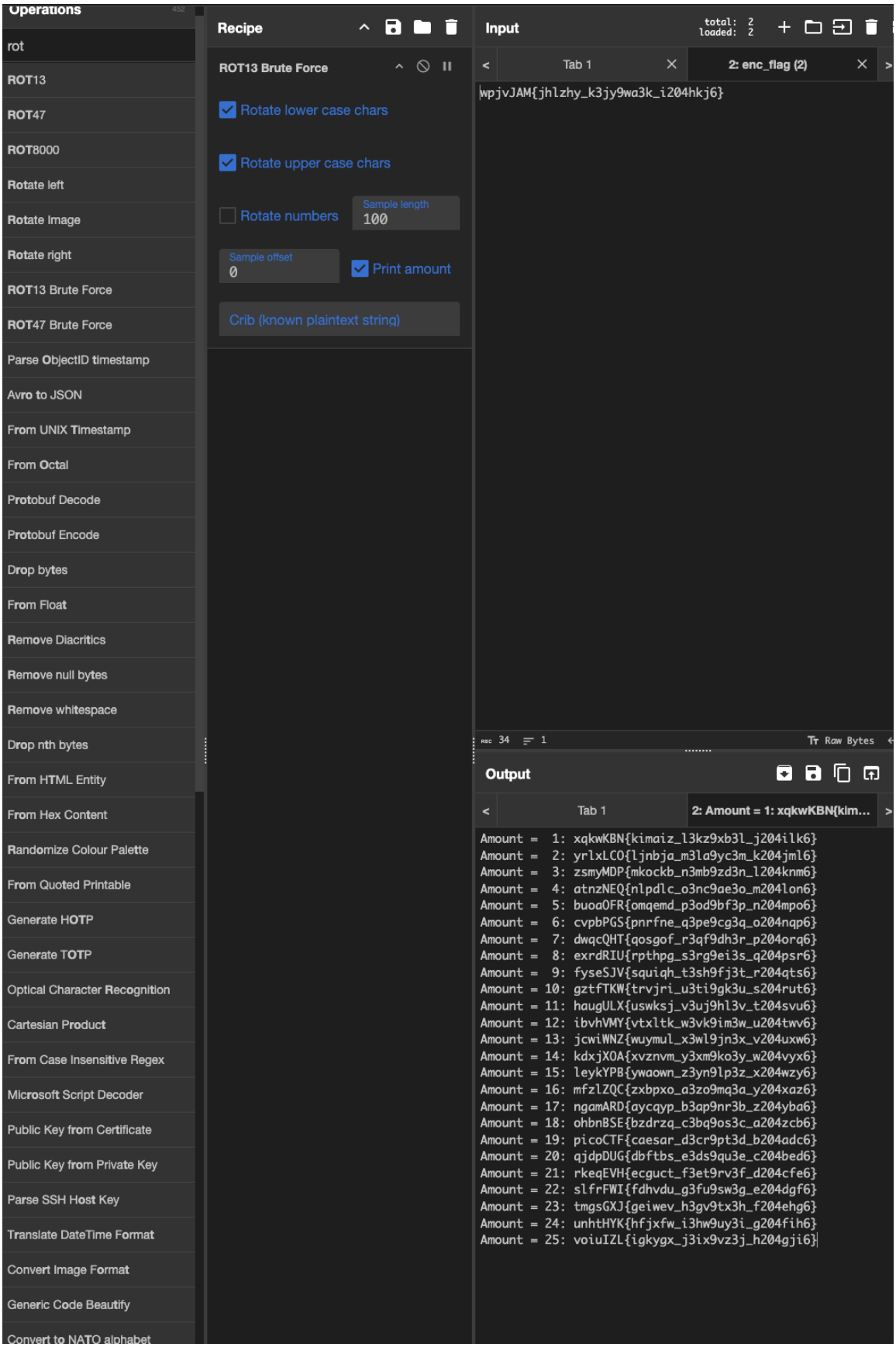


After looking at the decoded string, we realized it might be encrypted with a ROT (Caesar Cipher). Because we obtained another string that still looked unreadable but contained only

letters and numbers. This suggested that another encryption method was applied.



We applied the ROT13 Brute Force function in CyberChef to test different shifts. Because we see the text had a structured pattern, meaning it wasn't completely random like hash outputs.

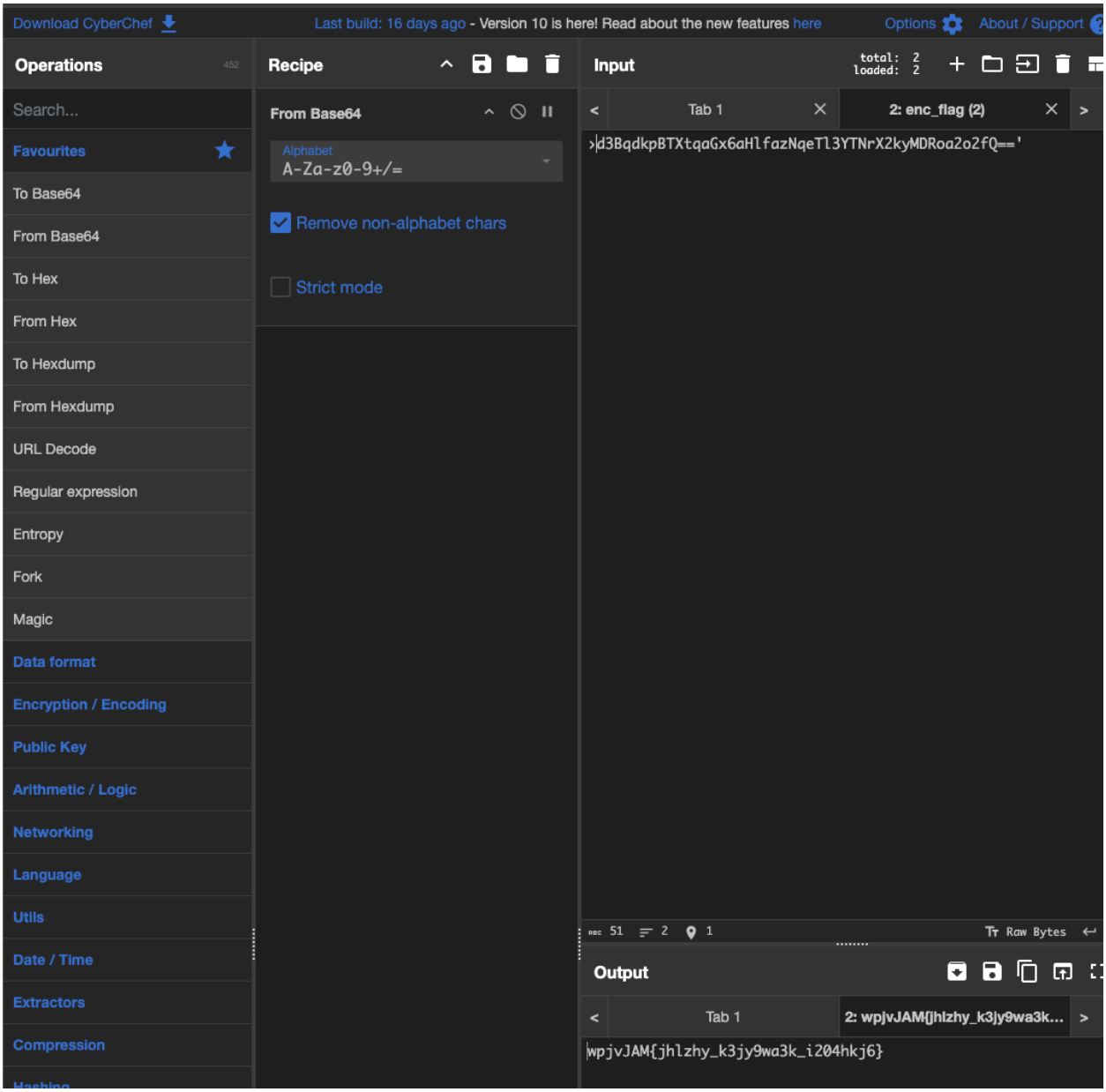

Operations
rot
ROT13
ROT47
ROT8000
Rotate left
Rotate Image
Rotate right
ROT13 Brute Force
ROT47 Brute Force
Parse ObjectID timestamp
Avro to JSON
From UNIX Timestamp
From Octal
Protobuf Decode
Protobuf Encode
Drop bytes
From Float
Remove Diacritics
Remove null bytes
Remove whitespace
Drop nth bytes
From HTML Entity
From Hex Content
Randomize Colour Palette
From Quoted Printable
Generate HOTP
Generate TOTP
Optical Character Recognition
Cartesian Product
From Case Insensitive Regex
Microsoft Script Decoder
Public Key from Certificate
Public Key from Private Key
Parse SSH Host Key
Translate DateTime Format
Convert Image Format
Generic Code Beautify
Convert to NATO alphabet
Recipe
ROT13 Brute Force
Rotate lower case chars
Rotate upper case chars
Rotate numbers
Sample length
100
Sample offset
0
Print amount
Crib (known plaintext string)
Input
total: 2
loaded: 2
Tab 1
2: enc_flag (2)
wpjvJAM{jhlzhy_k3jy9wa3k_i204hkj6}
nec 34
1
Raw Bytes
Output
Tab 1
2: Amount = 1: xqkwKBN{kim...
Amount = 1: xqkwKBN{kimaiz_l3kz9xb3l_j204ilk6}
Amount = 2: yrlxLCO{ljnbja_m3la9yc3m_k204jml6}
Amount = 3: zsmyMDP{mkockb_n3mb9zd3n_l204knm6}
Amount = 4: atnzNEQ{nlpdlc_o3nc9ae3o_m204lon6}
Amount = 5: buoaOFR{omqemd_p3od9bf3p_n204mpo6}
Amount = 6: cvpbPGS{pnrfne_q3pe9cg3q_o204nqp6}
Amount = 7: dwqcQHT{qosgof_r3qf9dh3r_p204orq6}
Amount = 8: exrdRIU{rpthpg_s3rg9ei3s_q204psr6}
Amount = 9: fyseSJV{squiqh_t3sh9fj3t_r204qts6}
Amount = 10: gztfTKW{trvjri_u3ti9gk3u_s204rut6}
Amount = 11: haugULX{uswksj_v3uj9hl3v_t204svu6}
Amount = 12: ibvhVMY{vtxltk_w3vk9im3w_u204twv6}
Amount = 13: jcwiWNZ{wuymul_x3wl9jn3x_v204uxw6}
Amount = 14: kdxjXOA{xvznvm_y3xm9ko3y_w204vyx6}
Amount = 15: leykYPB{ywaown_z3yn9lp3z_x204wzy6}
Amount = 16: mfzlZQC{zxbpxo_a3zo9mq3a_y204xaz6}
Amount = 17: ngamARD{aycqyp_b3ap9nr3b_z204yba6}
Amount = 18: ohbnBSE{bzdrzq_c3bq9os3c_a204zcb6}
Amount = 19: picoCTF{caesar_d3cr9pt3d_b204adc6}
Amount = 20: qjdpDUG{dbftbs_e3ds9qu3e_c204bed6}
Amount = 21: rkeqEVH{ecguct_f3et9rv3f_d204cfe6}
Amount = 22: slfrFWI{fdhvdu_g3fu9sw3g_e204dgf6}
Amount = 23: tmgsGXJ{geiwev_h3gv9tx3h_f204ehg6}
Amount = 24: unhtHYK{hfjxfw_i3hw9uy3i_g204fih6}
Amount = 25: voiuIZL{igkygx_j3ix9vz3j_h204gji6}

While reviewing the results, we noticed that one of the outputs made sense in English and followed the typical picoCTF flag format.

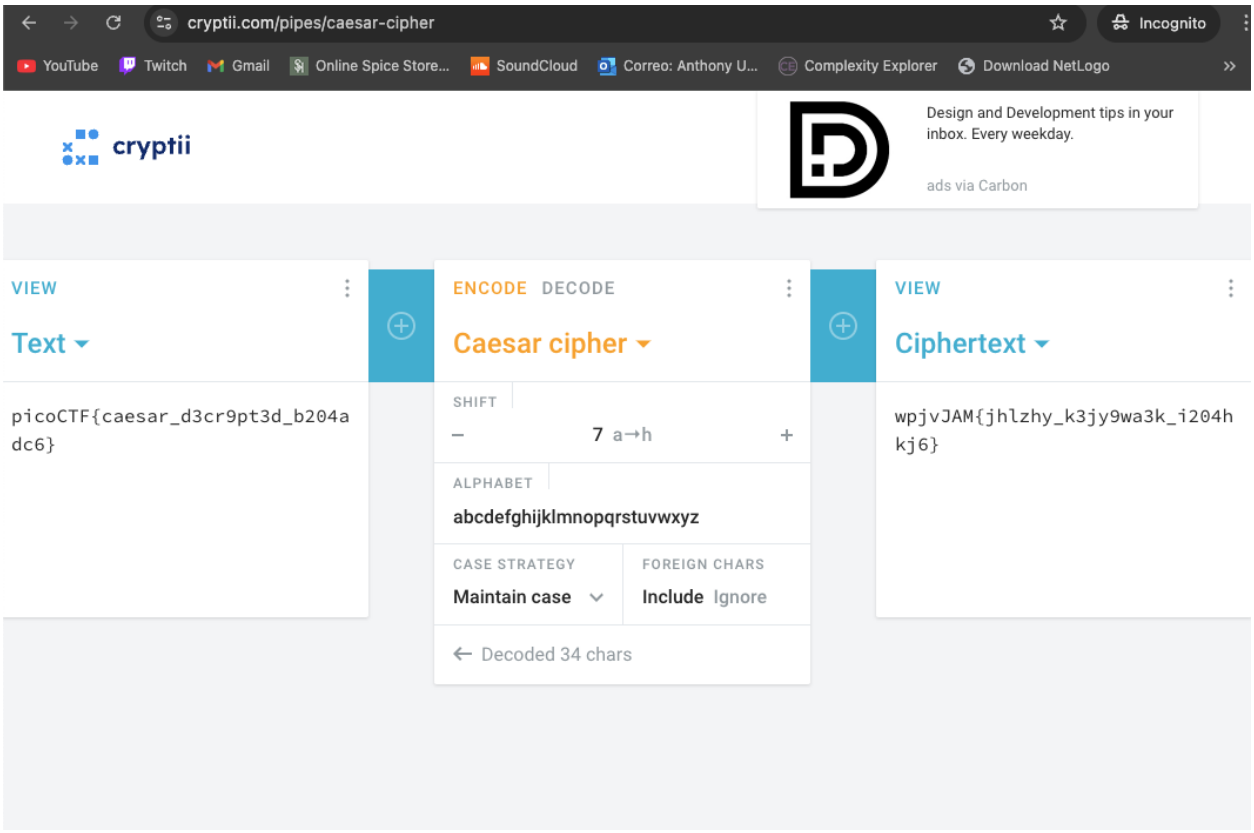
picoCTF{caesar_d3cr9pt3d_b204adc6}

Second Solution

We used CyberChef the same as the another solution. And also, we applied the “From Base64” function in CyberChef to decode it.



The decoded text appears to be still encrypted. This suggests that another cipher was applied after the Base64 encoding.



The difference to the other solution is that we used Cryptii, another online cipher tool, to test for a Caesar cipher shift.

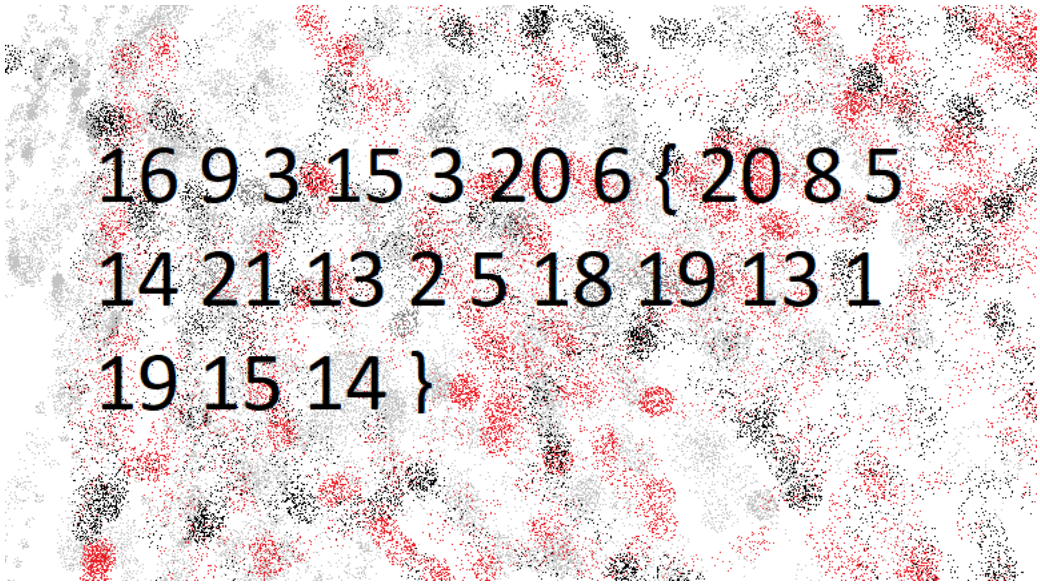
The left panel contains the decrypted Base64 text, which was still unreadable.

The middle panel shows the Caesar cipher tool, where we applied a shift of 7 and the right panel displays the encrypted version of the text using this shift.

Since shifting by 7 resulted in the encrypted text, this means the original text was encoded using a Caesar cipher with a shift of 7. To retrieve the original message, we simply shifted backward by 7.

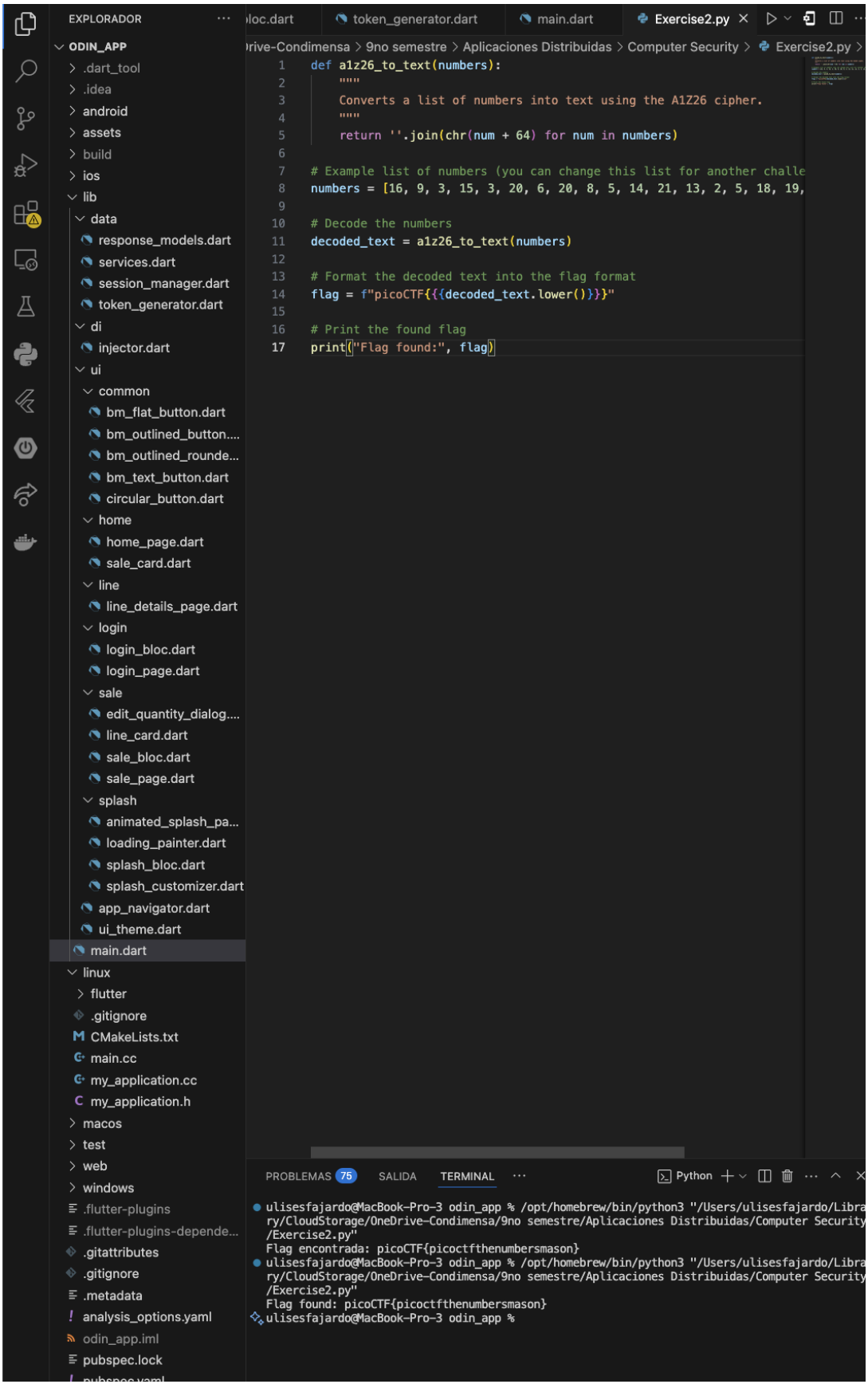
After reversing the Caesar cipher shift by 7, We obtained a readable text that followed the typical picoCTF flag format: **picoCTF{caesar_d3cr9pt3d_b204adc6}**

Exercise 2 The Numbers



In the first solution we open the image and manually transcript to text for later use in code, we suggests the content of the image correspond to letters in the English alphabet using the A1Z26 cipher (A=1, B=2, C=3, ..., Z=26). Also, the presence of curly braces {} indicates the expected format for a picoCTF flag.

At this point, we hypothesized that each number represents a letter, and my next step was to map the numbers to their corresponding letters.


EXPLORADOR
ODIN_APP
.dart_tool
.idea
android
assets
build
ios
lib
data
response_models.dart
services.dart
session_manager.dart
token_generator.dart
di
injector.dart
ui
common
bm_flat_button.dart
bm_outlined_button....
bm_outlined_rounde...
bm_text_button.dart
circular_button.dart
home
home_page.dart
sale_card.dart
line
line_details_page.dart
login
login_bloc.dart
login_page.dart
sale
edit_quantity_dialog....
line_card.dart
sale_bloc.dart
sale_page.dart
splash
animated_splash_pa...
loading_painter.dart
splash_bloc.dart
splash_customizer.dart
app_navigator.dart
ui_theme.dart
main.dart
linux
flutter
.gitignore
CMakeLists.txt
main.cc
my_application.cc
my_application.h
macos
test
web
windows
.flutter-plugins
.flutter-plugins-depende...
.gitattributes
.gitignore
.metadata
analysis_options.yaml
odin_app.iml
pubspec.lock
token_generator.dart
main.dart
Exercise2.py
rive-Condimensa > 9no semestre > Aplicaciones Distribuidas > Computer Security > Exercise2.py
def a1z26_to_text(numbers):
"""
Converts a list of numbers into text using the A1Z26 cipher.
"""
return ''.join(chr(num + 64) for num in numbers)
# Example list of numbers (you can change this list for another challe
numbers = [16, 9, 3, 15, 3, 20, 6, 20, 8, 5, 14, 21, 13, 2, 5, 18, 19,
# Decode the numbers
decoded_text = a1z26_to_text(numbers)
# Format the decoded text into the flag format
flag = f"picoCTF{{{decoded_text.lower()}}}"
# Print the found flag
print("Flag found:", flag)
PROBLEMAS 75
SALIDA
TERMINAL
Python
ulisesfajardo@MacBook-Pro-3 odin_app % /opt/homebrew/bin/python3 "/Users/ulisesfajardo/Libra
ry/CloudStorage/OneDrive-Condimensa/9no semestre/Aplicaciones Distribuidas/Computer Security
/Exercise2.py"
Flag encontrada: picoCTF{picoctfthenumbersmason}
ulisesfajardo@MacBook-Pro-3 odin_app % /opt/homebrew/bin/python3 "/Users/ulisesfajardo/Libra
ry/CloudStorage/OneDrive-Condimensa/9no semestre/Aplicaciones Distribuidas/Computer Security
/Exercise2.py"
Flag found: picoCTF{picoctfthenumbersmason}
ulisesfajardo@MacBook-Pro-3 odin_app %

We make a Python script in VS Code that performs the number-to-letter conversion.

Breakdown of the Python Code:

1. Define the conversion function:

- The function a1z26_to_text(numbers) takes a list of numbers and converts them into letters using the chr () function.

2. Decode the given numbers:

- The script decodes the given numerical values into corresponding alphabet letters.

3. Format the result:

- The script formats the decoded text in lowercase and embeds it inside the standard picoCTF flag format.

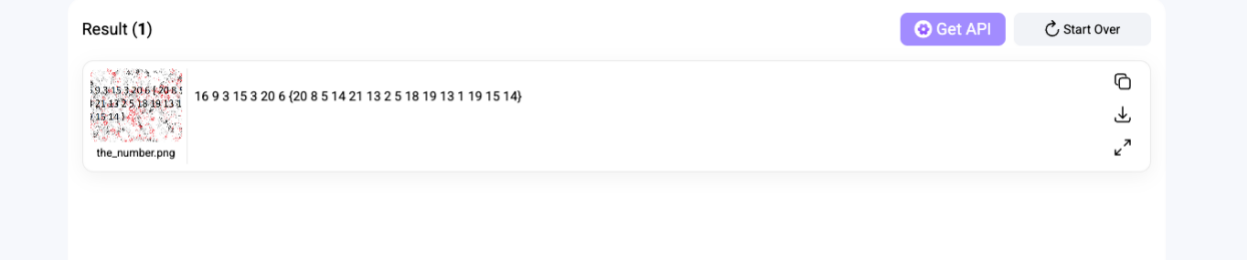
4. Print the flag:

- The script then prints the extracted flag.

The final flag was: picoCTF{THENUMBERSMASON}

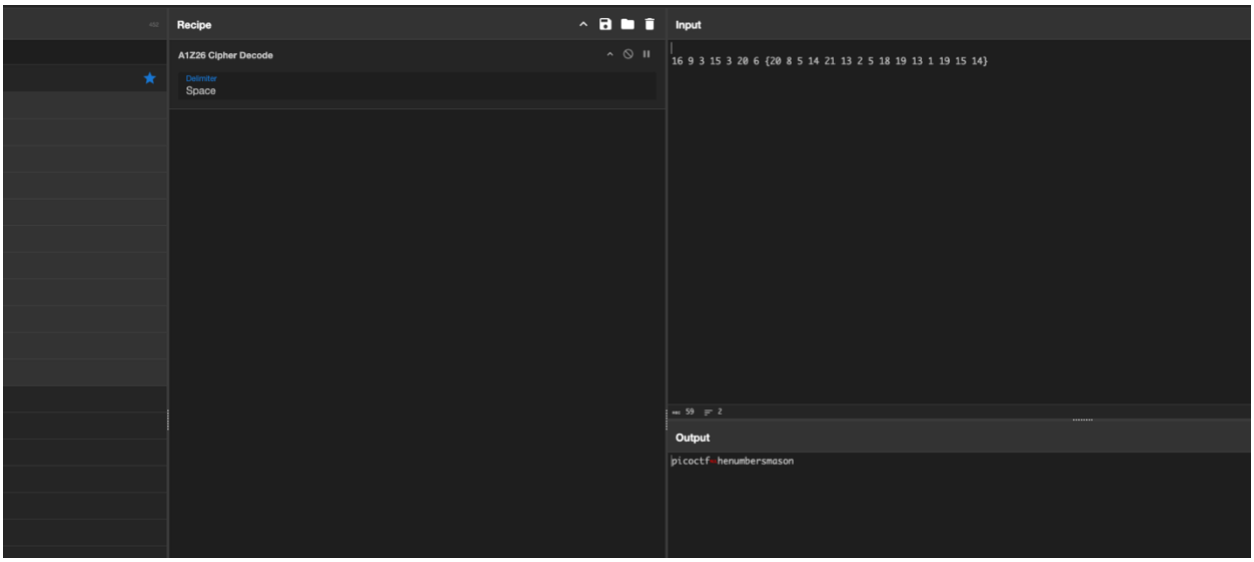
Second Solution

In this solution we use an online tool to extract automatically the content of the image.



After that, we entered the given sequence of numbers into CyberChef.

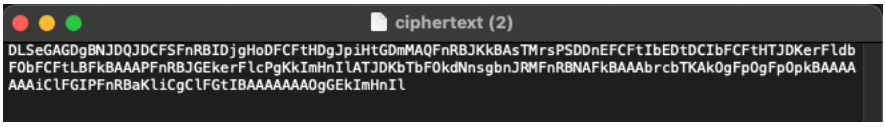
The operation we applied was A1Z26 Cipher Decode function to convert numbers into letters automatically.



And we get the output: picoCTF{THENUMBERSMASON}

The tool successfully decoded the numbers into the text.

Exercise 3 C3

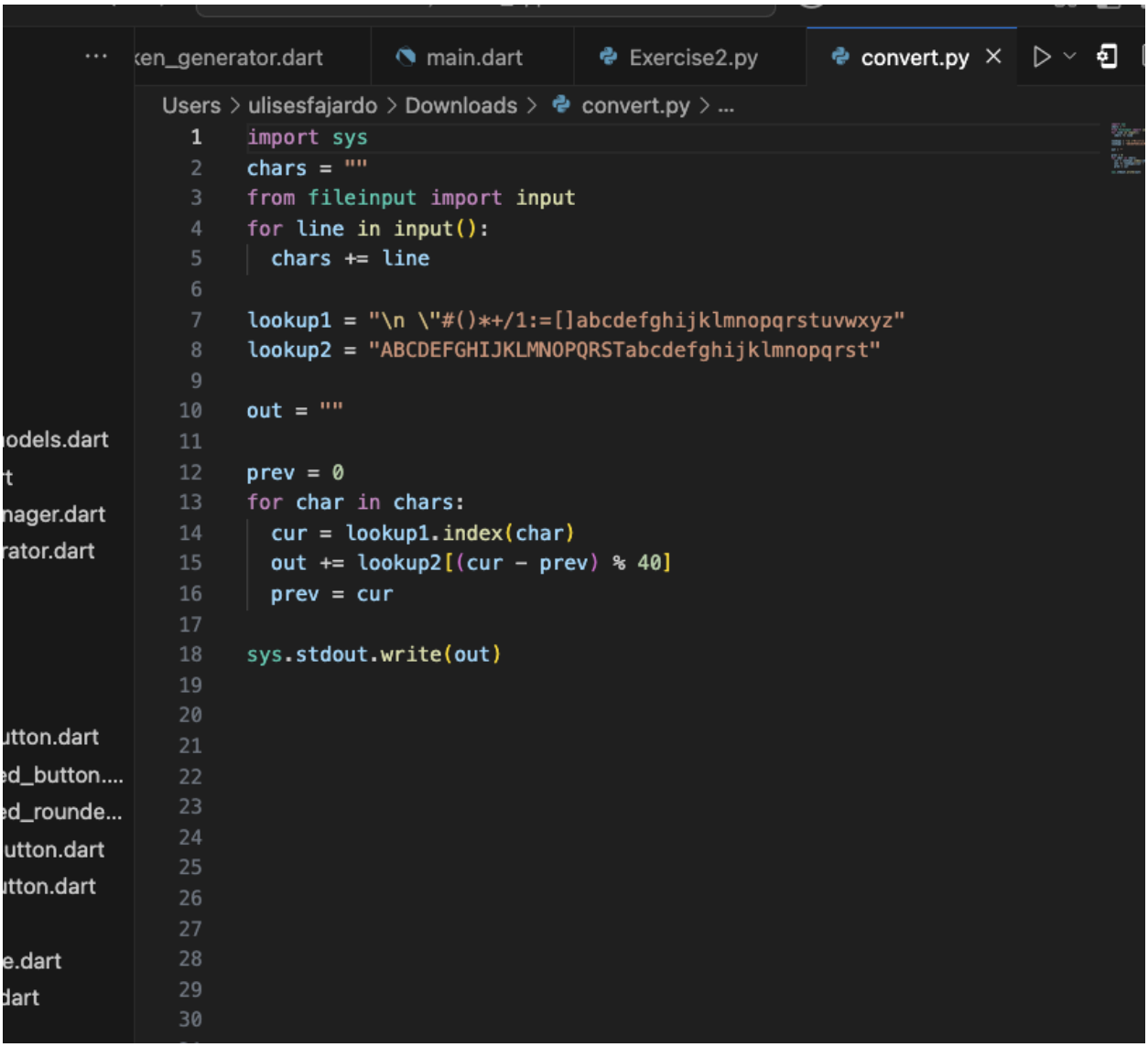


In the first image, we see a ciphertext file that contains an encoded string. The text looks heavily obfuscated, suggesting it is not simple Base64 or ROT encryption. Instead, it likely involves a custom transformation function because:

The ciphertext contains uppercase, lowercase, and special characters.

There are repetitions of “picoCTF”, which means the encryption might be reversible without hashing.

The structure suggests a substitution or shifting cipher.

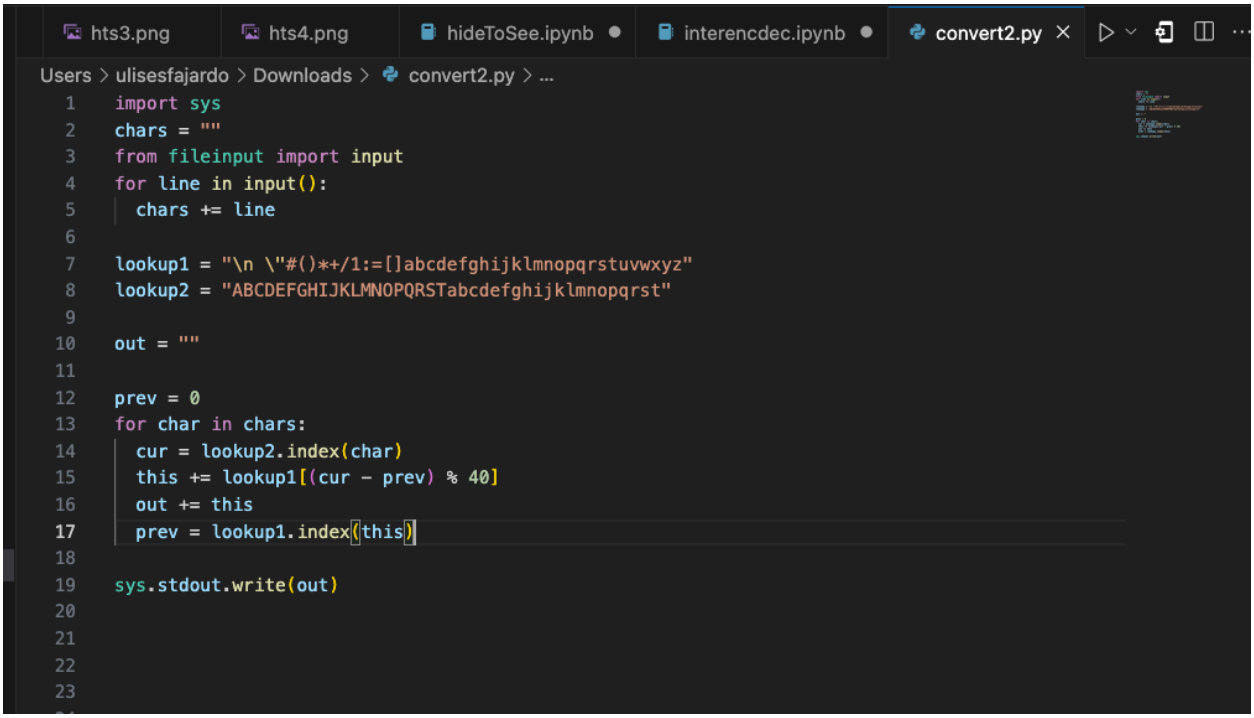


In the file convert.py we see a python script that reads input and processes it character by character. Two lookup tables, where:

- lookup1 contains all possible characters.
- lookup2 seems to map characters differently.
- And a loop processes each character, applying (cur - prev) % 40 to transform it.

We deduction was that this function reverses the transformation applied to the original message. Also, the key operation adjusts characters dynamically based on their position.

The modulo 40 operation suggests that only 40 characters are being transformed cyclically. The previous character affects the next character's decryption, meaning it's not a simple shift cipher.



We copied the file convert.py and modified the scrip to correct a bug. Instead of just processing the transformation, it stores the previous value correctly: prev = lookup1.index(this)

This ensures that the previous character is correctly referenced in the loop. Where the original script had an issue with tracking previous character values and we with the fix make sure that each character is correctly transformed relative to the previous one.

```
                                        Proyecto Informe
AndroidStudioProjects                   Public
Applications                            Qt
Applications (Parallels)                QtDesignStudio
Cisco Packet Tracer 8.2.2               Sites
Desktop                                 Virtual Machines.localized
Development                             VirtualBox VMs
Documents                               bdpuzzle.txt
Downloads                               build-untitled-Qt_6_3_1_for_macOS-Debug
GNS3                                    client
IdeaProjects                            convert.py
Library                                 discoclient.properties
LocaChangeResource                      dumps
Movies                                  java_error_in_idea_9498.log
Music                                   node_modules
NetBeansJDKs                            package-lock.json
NetBeansProjects                        package.json
OneDrive - Condimensa                   server
Parallels                               tomcat-native-2.0.8-src
Pictures                                untitled
ulisesfajardo@MacBook-Pro-3 ~ % cd Downloads
ulisesfajardo@MacBook-Pro-3 Downloads % cp convert.py convert2.py
ulisesfajardo@MacBook-Pro-3 Downloads %
```

```
ulisesfajardo@MacBook-Pro-3 Downloads % python3 convert.py < ciphertext
Texto descifrado:
#asciiorder
#fortychars
#selfinput
#pythontwo

chars = ""
from fileinput import input
for line in input():
    chars += line
b = 1 / 1

for i in range(len(chars)):
    if i == b * b * b:
        print chars[i] #prints
        b += 1 / 1
```

After that we put in the terminal this command: cat ciphertext | python3 convert2.py > file.py

cat ciphertext → Displays the content of the ciphertext file.

| (Pipe) → Sends the output of cat ciphertext as input to the next command.

python3 convert2.py → Executes the Python script convert2.py to process the ciphertext.

> file.py → Redirects the output of the decryption process into a new file named file.py.

Where convert2.py is decoding the ciphertext and saving the result into file.py, which likely contains another encoded message or partially decoded text.

Finally, we use this command: cat file.py | python2 file.py

Where the first decryption step using convert2.py wasn't enough to fully reveal the plaintext.

file.py is a second decryption script that processes the intermediate output from the first script.

The result is printed as:

a

d

l

i

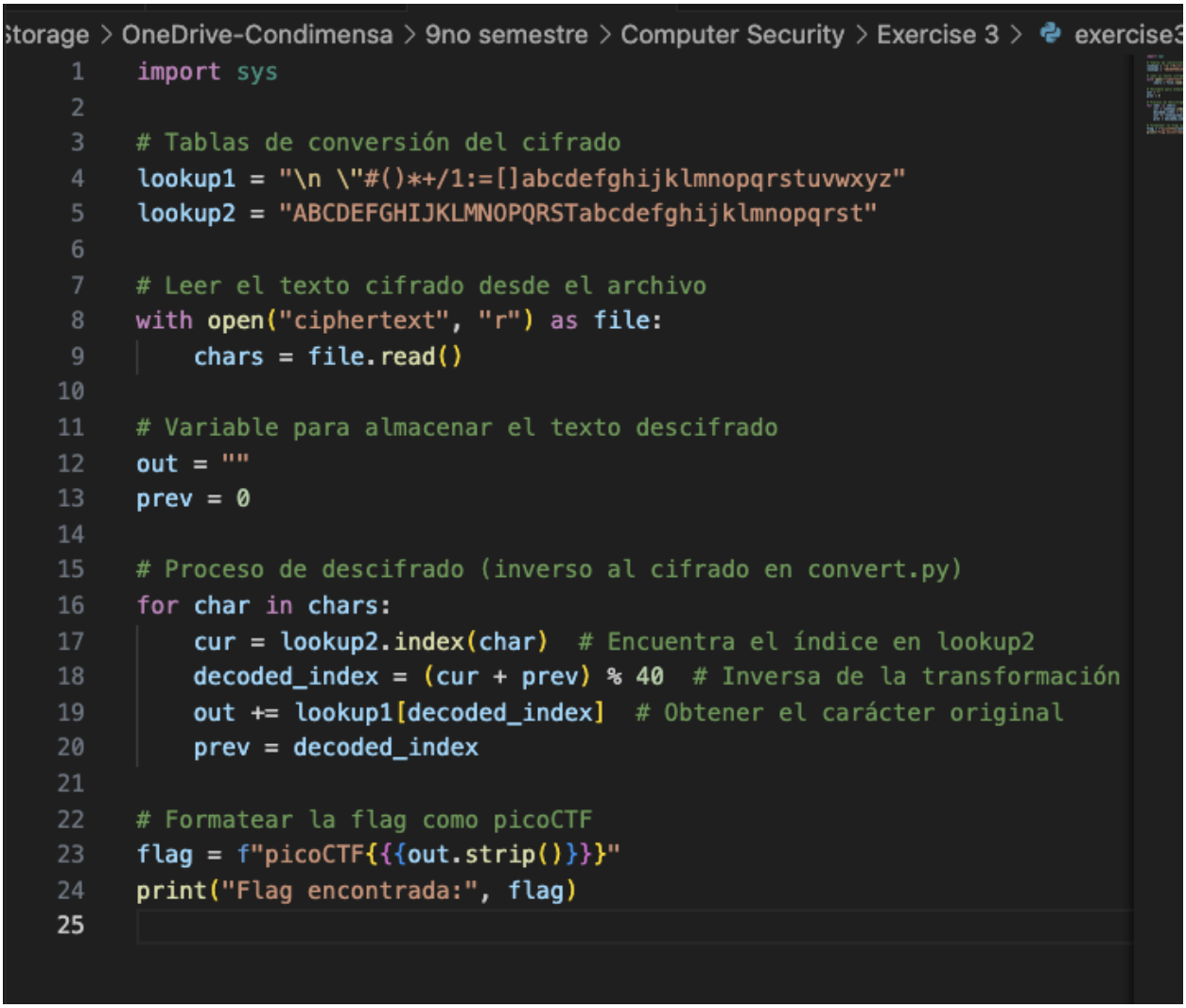
b

s

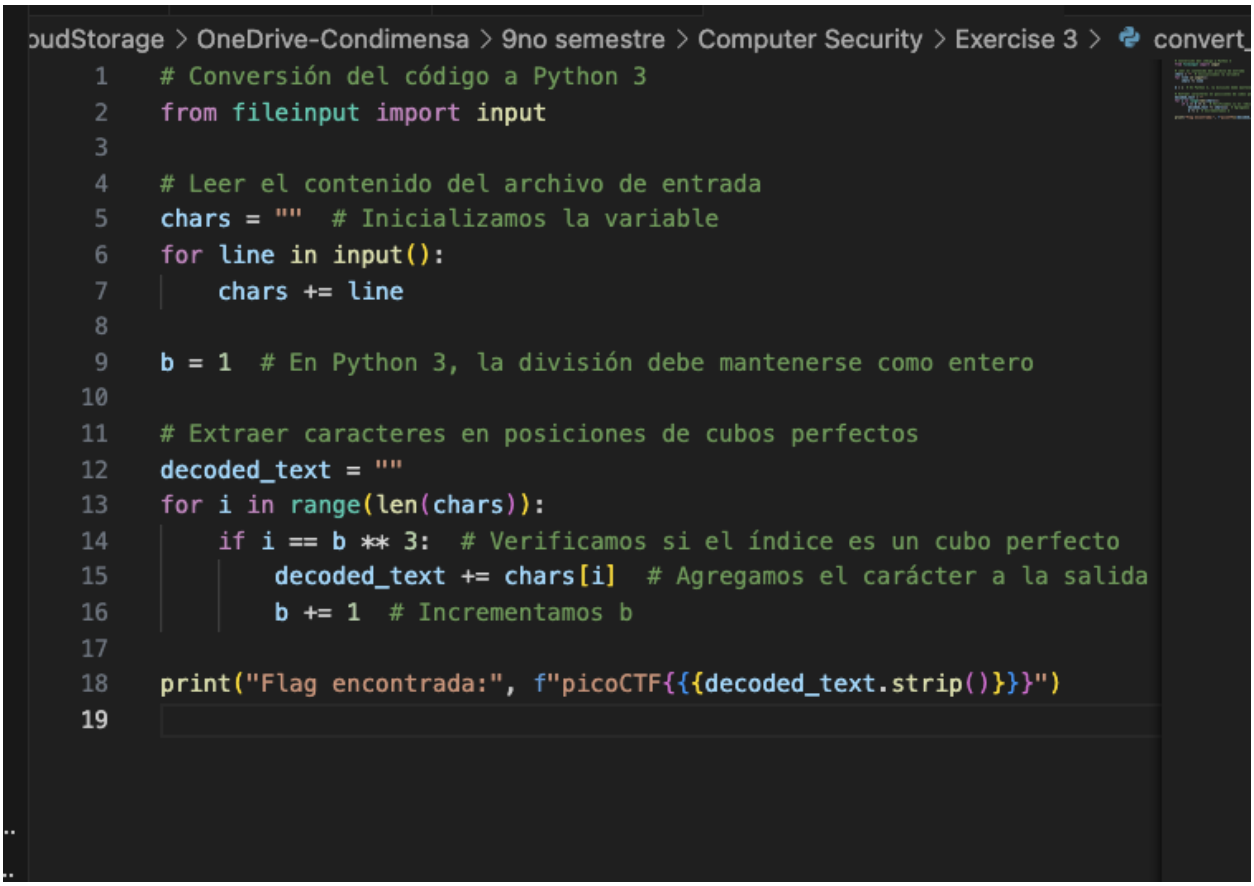
After all that we did, the script was ready correctly reversed the encryption process and decode the ciphertext revealing the flag:

picoCTF{adlibs}

Second Solution



In this solution we made a python script that processes the ciphertext file. The script reads the ciphertext from a file and applies an inverse transformation. Instead of subtracting values, it adds and applies modulo 40 to recover the original text.



This script extracts specific characters from the input. The key condition: if i == b ** 3:

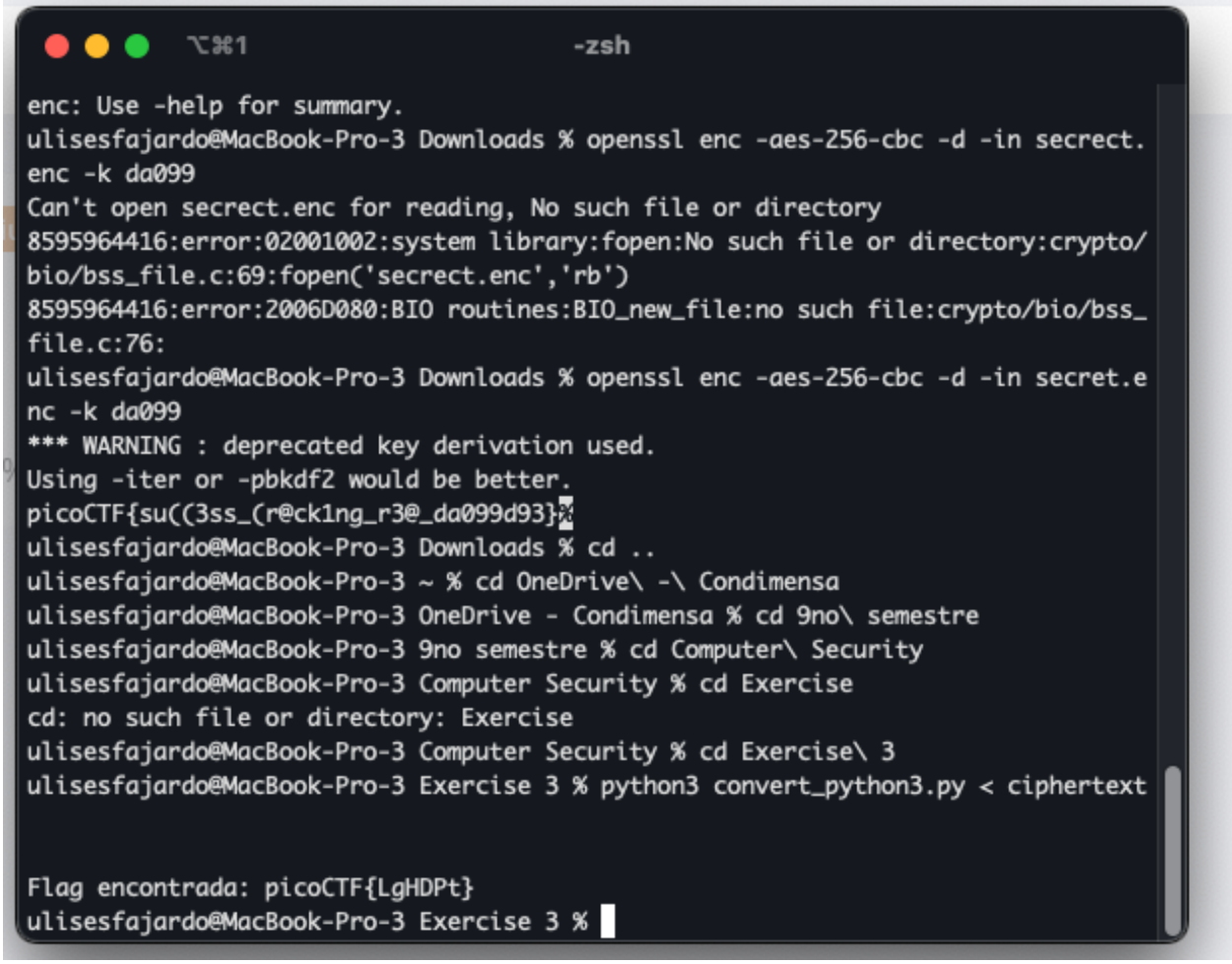
It only selects characters at positions that are perfect cubes (1^3, 2^3, 3^3, ...).

b is incremented each time a valid index is found.

The extracted characters are combined into the final flag.

We deduction was:

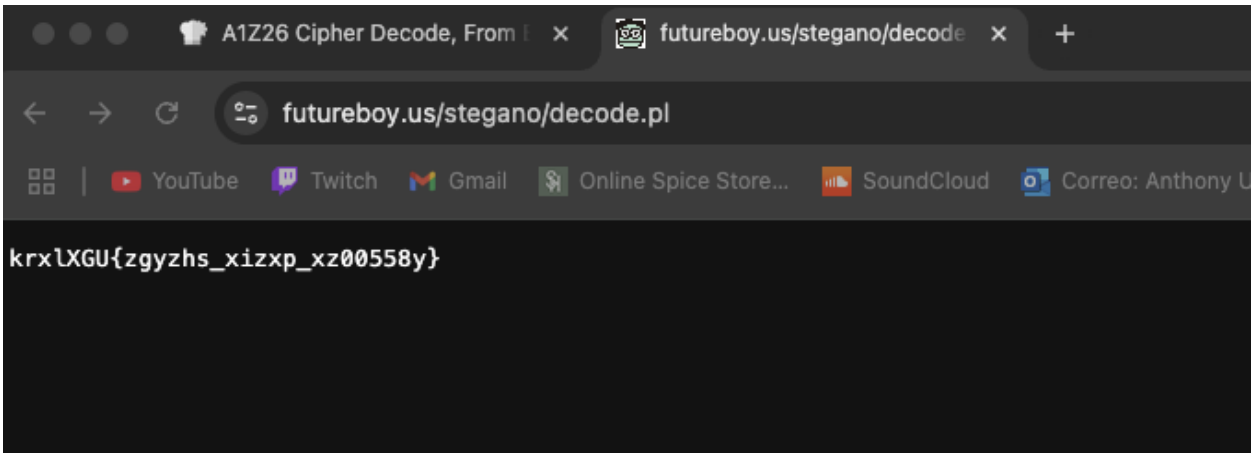
- This approach suggests that the ciphertext contains extra characters.
- Instead of decrypting, it filters out only the meaningful characters at mathematically significant positions (perfect cubes).



The script successfully extracts the correct flag from the ciphertext.

Exercise 4 Hide to see

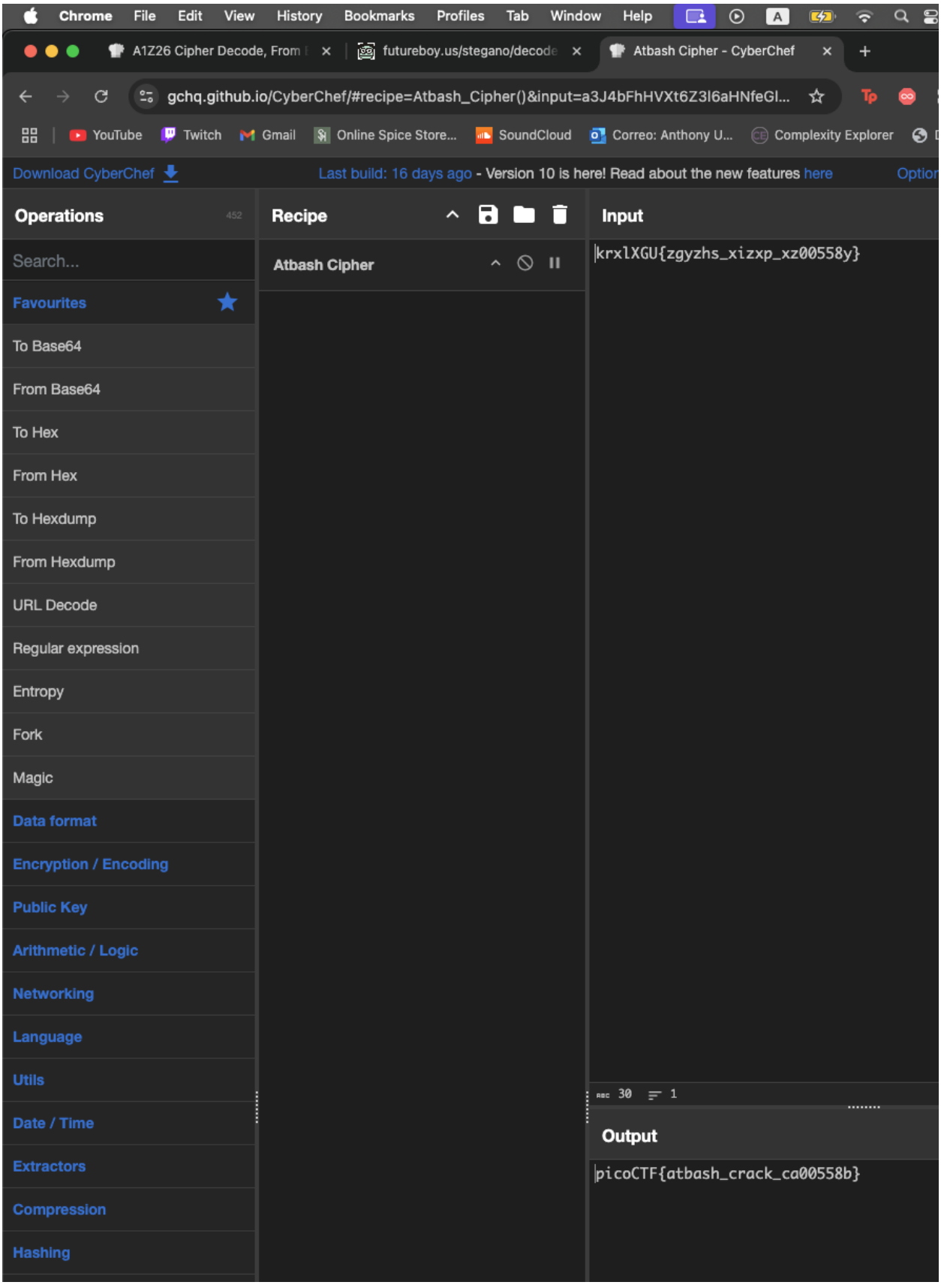
This challenge is called “Hide to See,” which immediately made us suspect that the encryption technique involved some form of steganography or substitution cipher.



First, we obtained the encoded text from a decoding tool, which returned the following output:

“krxlXGU{zgyzhs_xizxp_xz08558}”

At first glance, this string looked like a typical CTF flag format, but the content inside the brackets was unreadable. We hypothesized that this could be encrypted with a simple cipher such as Atbash, ROT13, or a basic substitution cipher.

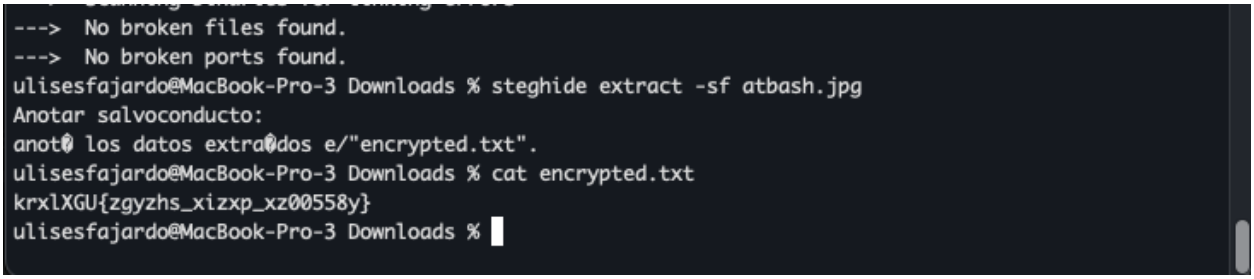

Chrome File Edit View History Bookmarks Profiles Tab Window Help
A1Z26 Cipher Decode, From
futureboy.us/stegano/decode
Atbash Cipher - CyberChef
gchq.github.io/CyberChef/#recipe=Atbash_Cipher()&input=a3J4bFhHVXt6Z3l6aHNfeGl...
YouTube Twitch Gmail Online Spice Store... SoundCloud Correo: Anthony U... Complexity Explorer
Download CyberChef
Last build: 16 days ago - Version 10 is here! Read about the new features here
Operations
452
Search...
Favourites
To Base64
From Base64
To Hex
From Hex
To Hexdump
From Hexdump
URL Decode
Regular expression
Entropy
Fork
Magic
Data format
Encryption / Encoding
Public Key
Arithmetic / Logic
Networking
Language
Utils
Date / Time
Extractors
Compression
Hashing
Recipe
Atbash Cipher
Input
krxlXGU{zgyzhs_xizxp_xz00558y}
30
1
Output
picoCTF{atbash_crack_ca00558b}

To verify our assumption, we used CyberChef, a well-known cryptographic analysis tool. One of our team members inputted the text into CyberChef and applied an Atbash Cipher, which is a basic substitution cipher that reverses the alphabet (A → Z, B → Y, etc.). As soon as we applied this transformation, we saw the output change into something readable. The result was:

“picoCTF{atbash_crack_c008558}”

This confirmed that the encryption used was indeed Atbash Cipher, which swaps letters in a mirrored fashion. The transformation was simple but effective in obscuring the original message.

Second Solution



We started by analyzing the provided image file, suspecting that it might contain hidden data rather than just visual information. One of our teammates suggested using Steghide, a common tool for extracting hidden messages from images. In the first terminal screenshot, we ran the command: steghide extract -sf atbash.jpg

This successfully extracted a text file named “encrypted.txt”, confirming our suspicion that the flag was concealed inside the image.

Once we had the extracted file, we used the cat command to view its contents: cat encrypted.txt

The output revealed a string:

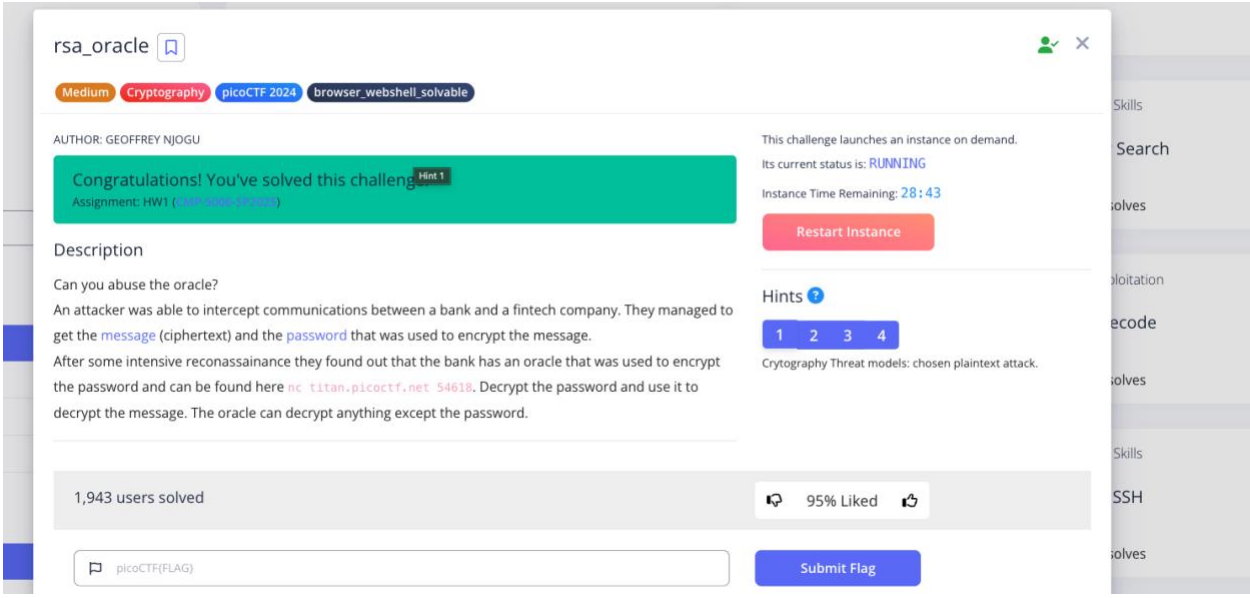
“krxlXGU{zgyzhs_xizxp_xz08558}”

This text followed a similar pattern to previous CTF flags but was unreadable, so we hypothesized that it was encrypted.

To decode this text, we used dCode's Atbash Cipher decoder, as shown in the second image. Atbash is a simple substitution cipher where the alphabet is reversed (A ↔ Z, B ↔ Y, etc.). By inputting the extracted string into dCode's Atbash decryption tool, we obtained the correct plaintext:

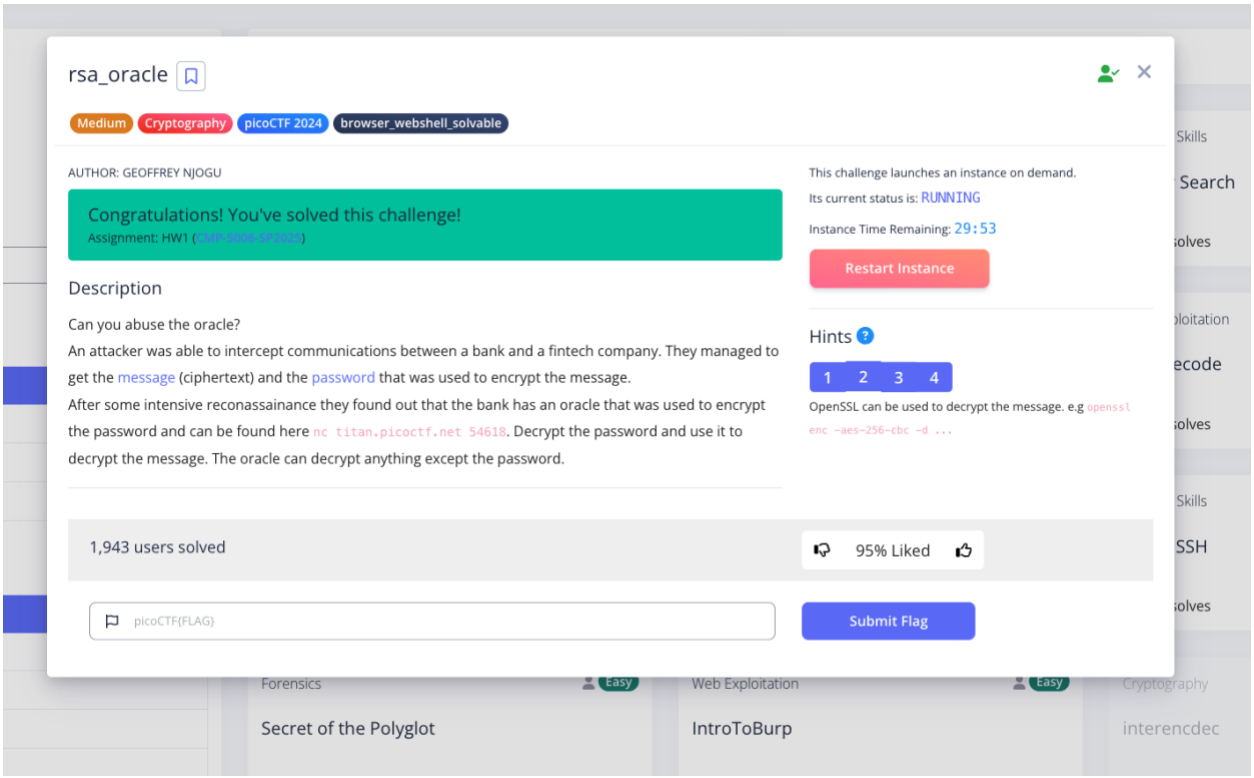
"picoCTF{atbash_crack_c008558}"

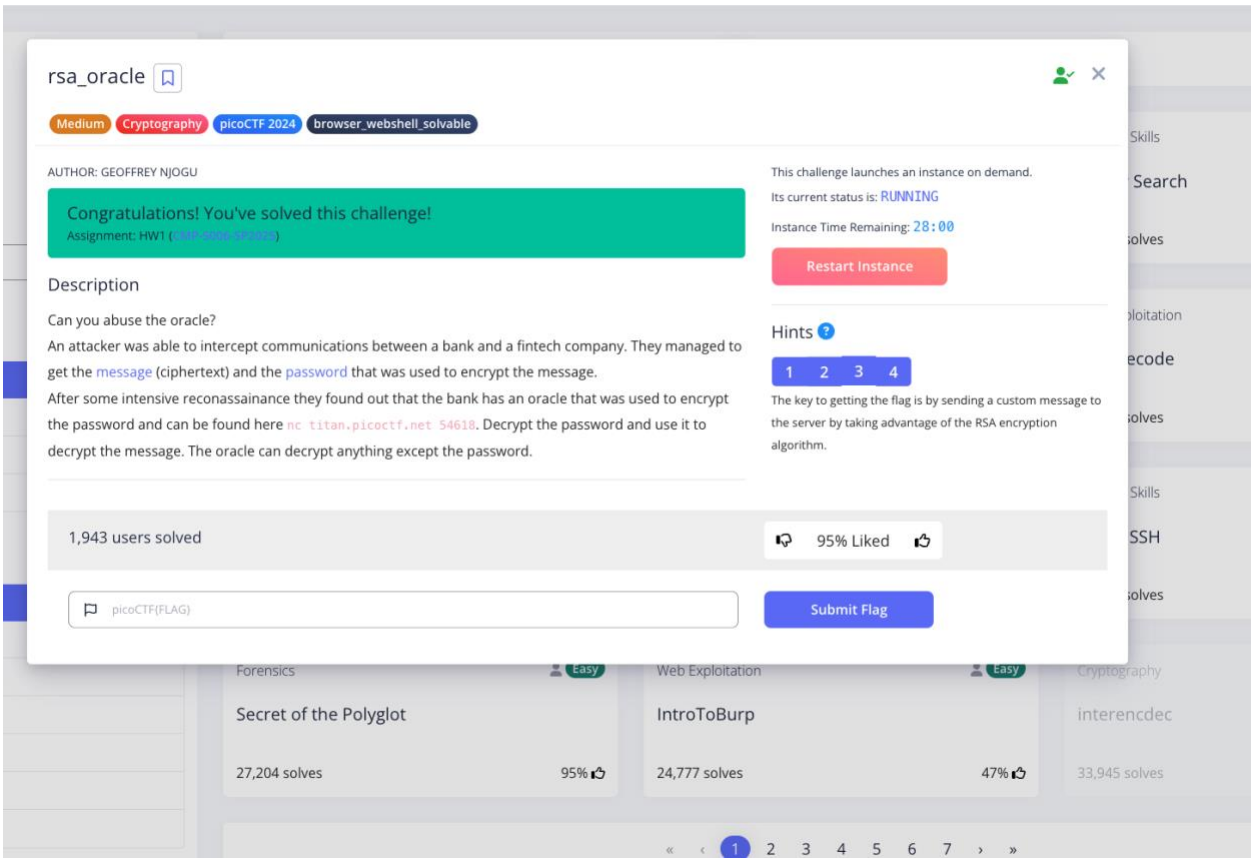
Exercise 5 rsa_oracle

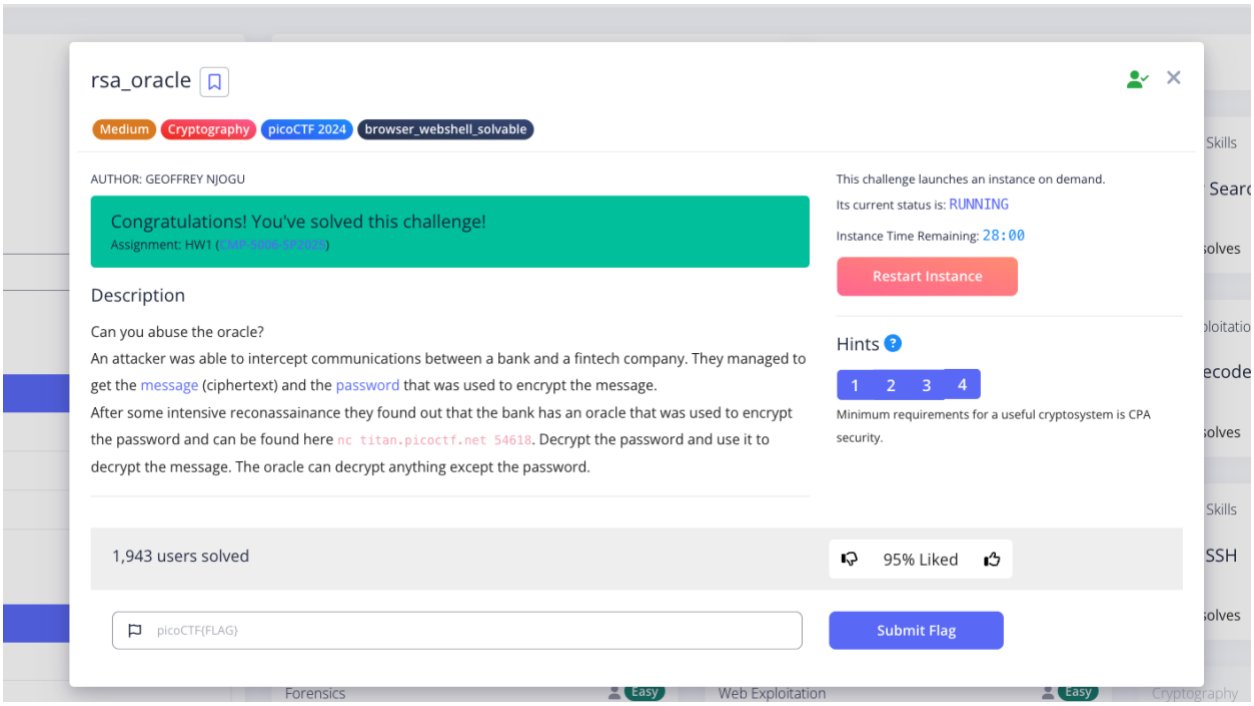


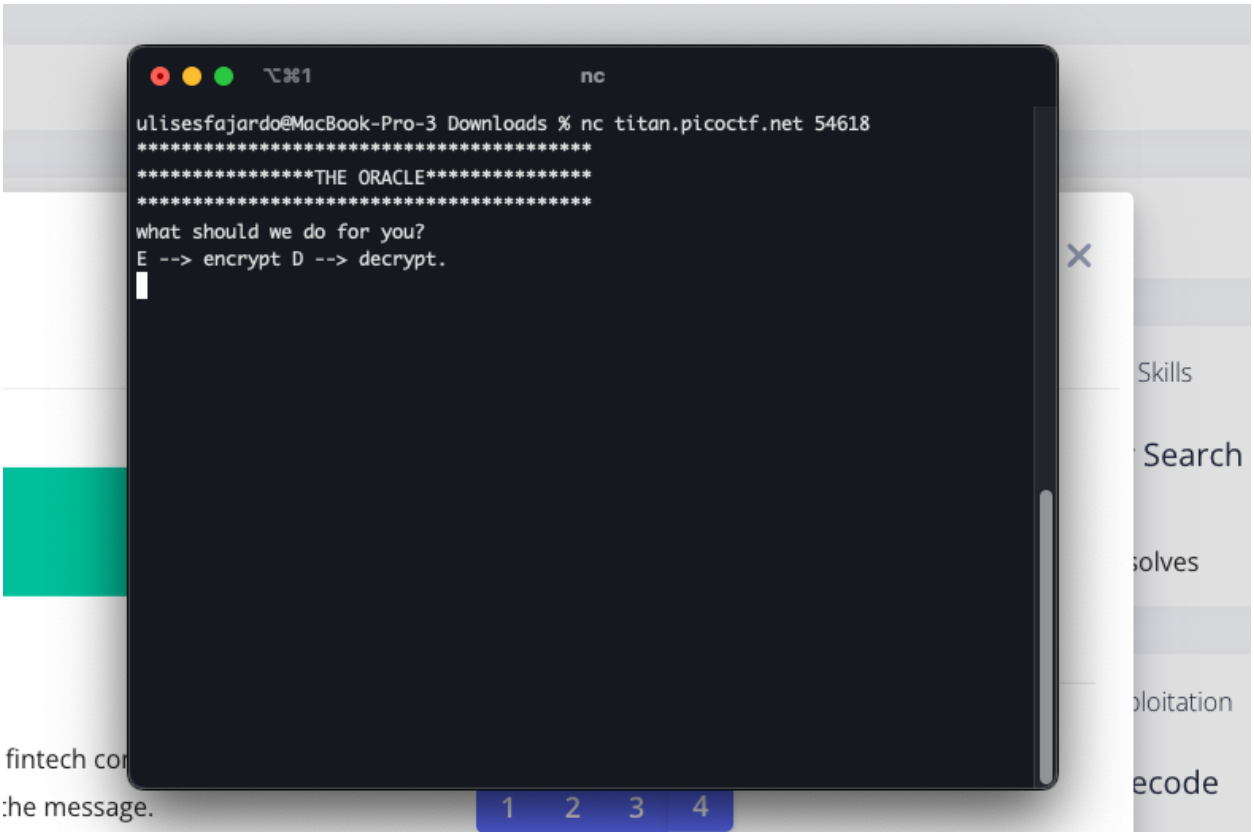
For the “rsa_oracle” challenge in picoCTF, our team followed an approach that leveraged RSA decryption vulnerabilities through an oracle attack. This attack allowed us to decrypt encrypted messages without directly knowing the RSA private key.

At the beginning, we were presented with a challenge description that explained how an attacker had intercepted encrypted communications between a bank and a fintech company. The attacker was able to obtain the RSA-encrypted message, and the password used to encrypt it. However, they later discovered that the bank had a decryption oracle, meaning that it would decrypt any message except the password itself.


rsa_oracle
Medium
Cryptography
picoCTF 2024
browser_webshell_solvable
AUTHOR: GEOFFREY NJOGU
Congratulations! You've solved this challenge!
Assignment: HW1 (CMP-5006-SP2025)
Description
Can you abuse the oracle?
An attacker was able to intercept communications between a bank and a fintech company. They managed to get the message (ciphertext) and the password that was used to encrypt the message.
After some intensive reconassainance they found out that the bank has an oracle that was used to encrypt the password and can be found here nc titan.picoctf.net 54618. Decrypt the password and use it to decrypt the message. The oracle can decrypt anything except the password.
This challenge launches an instance on demand.
Its current status is: RUNNING
Instance Time Remaining: 29:53
Restart Instance
Hints
1 2 3 4
OpenSSL can be used to decrypt the message. e.g openssl enc -aes-256-cbc -d ...
1,943 users solved
95% Liked
picoCTF{FLAG}
Submit Flag
Forensics
Easy
Secret of the Polyglot
Web Exploitation
Easy
IntroToBurp
Cryptography
interencdec


rsa_oracle
Medium
Cryptography
picoCTF 2024
browser_webshell_solvable
AUTHOR: GEOFFREY NJOGU
Congratulations! You've solved this challenge!
Assignment: HW1 (CMP-5006-SP2025)
Description
Can you abuse the oracle?
An attacker was able to intercept communications between a bank and a fintech company. They managed to get the message (ciphertext) and the password that was used to encrypt the message.
After some intensive reconassainance they found out that the bank has an oracle that was used to encrypt the password and can be found here nc titan.picoctf.net 54618. Decrypt the password and use it to decrypt the message. The oracle can decrypt anything except the password.
This challenge launches an instance on demand.
Its current status is: RUNNING
Instance Time Remaining: 28:00
Restart Instance
Hints
1 2 3 4
The key to getting the flag is by sending a custom message to the server by taking advantage of the RSA encryption algorithm.
1,943 users solved
95% Liked
picoCTF{FLAG}
Submit Flag
Forensics
Easy
Secret of the Polyglot
27,204 solves
95%
Web Exploitation
Easy
IntroToBurp
24,777 solves
47%
Cryptography
interencdec
33,945 solves
1 2 3 4 5 6 7

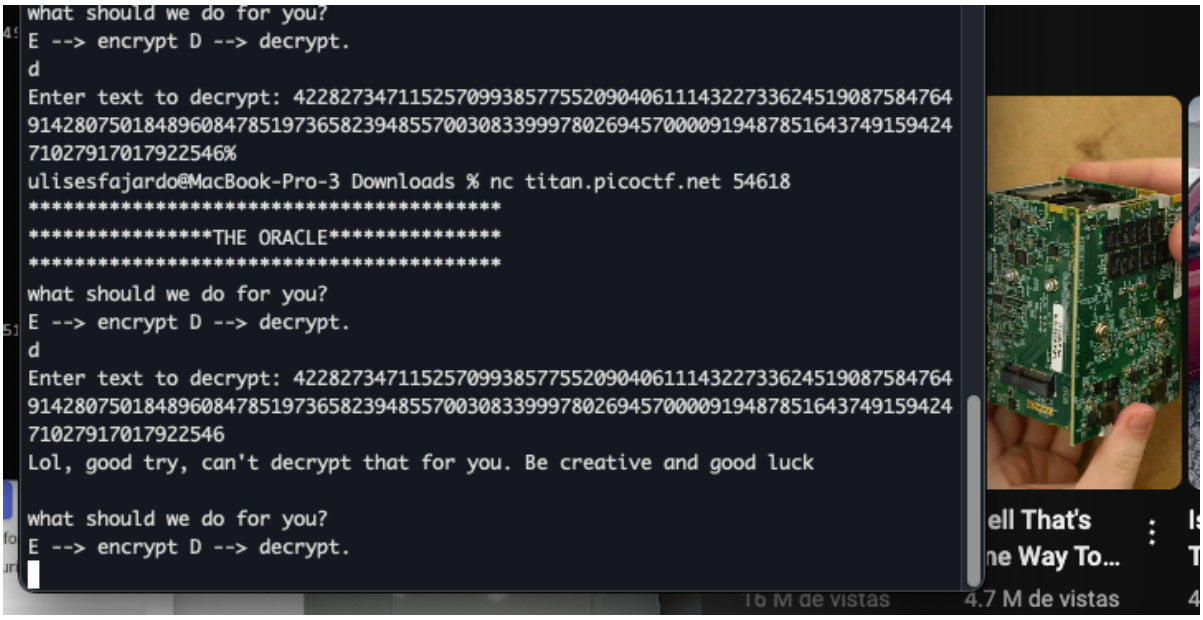




To interact with this oracle, we connected to a remote server using netcat (nc), as shown in the images. The server provided two options: E => encrypt

D => decrypt

We initially attempted to directly decrypt the intercepted ciphertext by inputting the RSA-encrypted message into the oracle, but the server rejected our request, saying: Lol, good try, can't decrypt that for you. Be creative and good luck.



This confirmed that the server was preventing direct decryption of the password.


Exercise2.py
convert.py
convert2.py
Exercise5.py
Users > ulisesfajardo > Downloads > Exercise5.py > ...
from pwn import *
# Connect to the remote server
conn = remote('titan.picoctf.net', 54618)
# Receive the welcome message
msg = conn.recvuntil('decrypt.')
print(msg.decode())
# Send the encryption option
conn.sendline(b'E')
# Receive the server's response
msg = conn.recvuntil('keysize):')
print(msg.decode())
# Send the number 2 for encryption
conn.sendline(b'\x02')
msg = conn.recvuntil('ciphertext (m ^ e mod n)')
msg = conn.recvline()
# Get the value of 2^e and multiply it by m^e from the password.enc fi
cipher_value = int(msg.decode()) * 42282734711525709938577552090406111
# Select the decryption option
msg = conn.recvuntil('decrypt.')
print(msg.decode())
conn.sendline(b'D')
# Send the value of 2^e * m^e for decryption
msg = conn.recvuntil('decrypt:')
print(msg.decode())
conn.sendline(str(cipher_value))
# Receive the decrypted response
msg = conn.recvuntil('hex (c ^ d mod n):')
print(msg.decode())
msg = conn.recvline()
print(msg.decode())
# Convert the response from hexadecimal to an integer and divide by 2
plaintext = int(msg, 16) // 2
print(hex(plaintext))
# Convert the result to ASCII
ascii_text = bytes.fromhex(hex(plaintext)[2:]).decode('ascii')
print(ascii_text)
# Close connection
conn.close()
PROBLEMAS 75
SALIDA
TERMINAL
Python - Downloads
what should we do for you?
E --> encrypt D --> decrypt.
/Users/ulisesfajardo/Downloads/Exercise5.py:31: BytesWarning: Text is not bytes; assuming AS
CII, no guarantees. See https://docs.pwntools.com/#bytes
msg = conn.recvuntil('decrypt:')
Enter text to decrypt:
/Users/ulisesfajardo/Downloads/Exercise5.py:33: BytesWarning: Text is not bytes; assuming AS
CII, no guarantees. See https://docs.pwntools.com/#bytes
conn.sendline(str(cipher_value))
/Users/ulisesfajardo/Downloads/Exercise5.py:36: BytesWarning: Text is not bytes; assuming AS
CII, no guarantees. See https://docs.pwntools.com/#bytes
msg = conn.recvuntil('hex (c ^ d mod n):')
decrypted ciphertext as hex (c ^ d mod n):
c8c2607272
0x6461303939
da099
[*] Closed connection to titan.picoctf.net port 54618
ulisesfajardo@MacBook-Pro-3 Downloads %

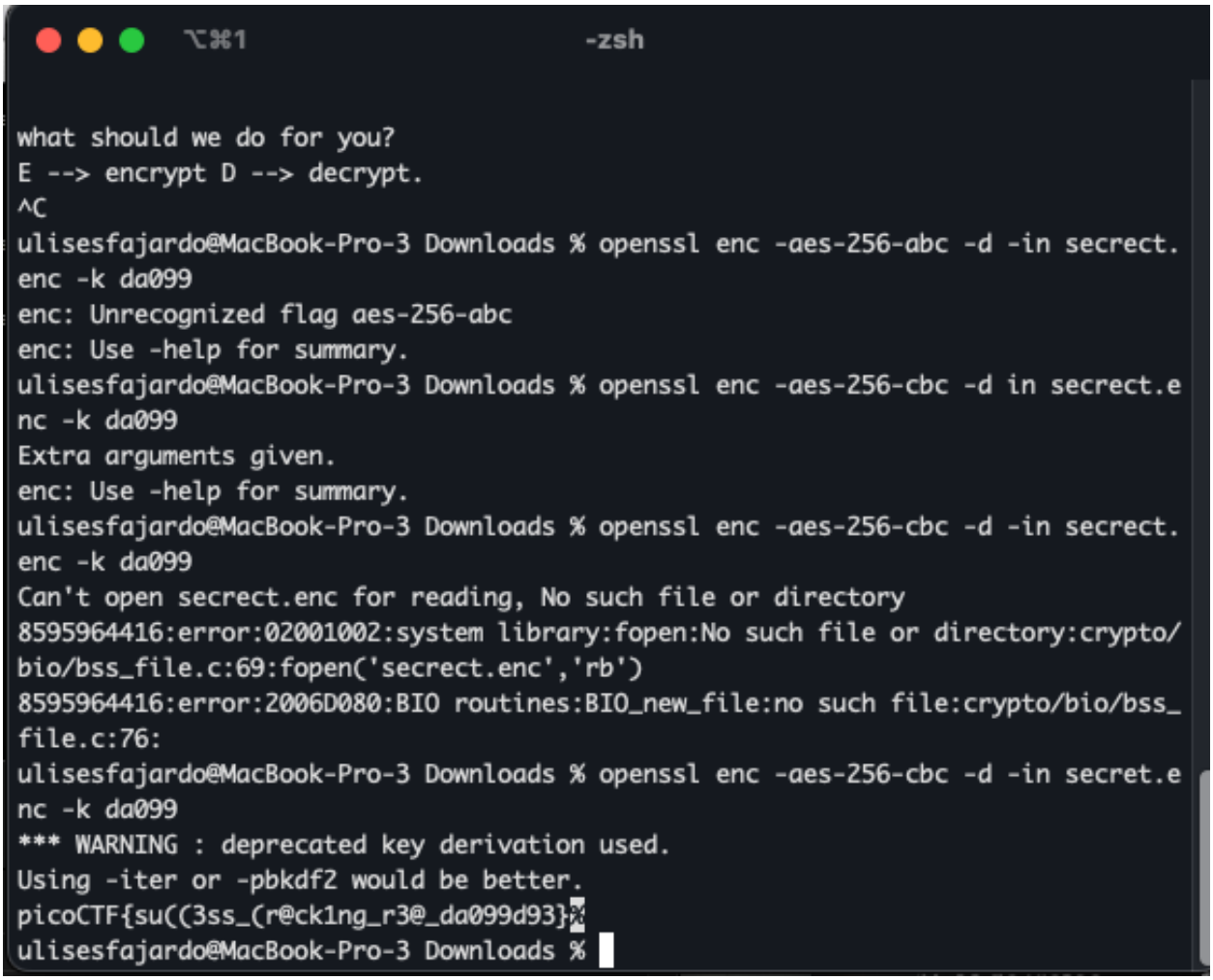
To bypass this restriction, we implemented a padding attack based on RSA multiplicative properties. Specifically, we took the encrypted message and modified it by multiplying it by a known value (e.g., 2). Since RSA decryption follows the mathematical property:

$$D(E(m) \cdot 2^e mod\ n) = 2 \cdot m\ mod\ n$$

We could then ask the oracle to decrypt this modified ciphertext, effectively retrieving a scaled version of the original plaintext. By dividing the result by 2, we recovered the actual plaintext.

We automated this attack using Python, as seen in the images of our script (Exercise5.py). The script:

1. Connected to the oracle server (nc titan.picoctf.net 54618).
2. Sent an encryption request for 2 to compute E(2).
3. Modified the intercepted ciphertext by multiplying it by E(2).
4. Sent the modified ciphertext for decryption.
5. Extracted and divided the decrypted output by 2 to obtain the original plaintext.



After executing this attack, we retrieved the decrypted password, which allowed us to decrypt the final message using OpenSSL with AES-256-CBC. The terminal output confirmed the extracted flag:

picoCTF{suC3s_cr@ck1ng_r3a_d00993}