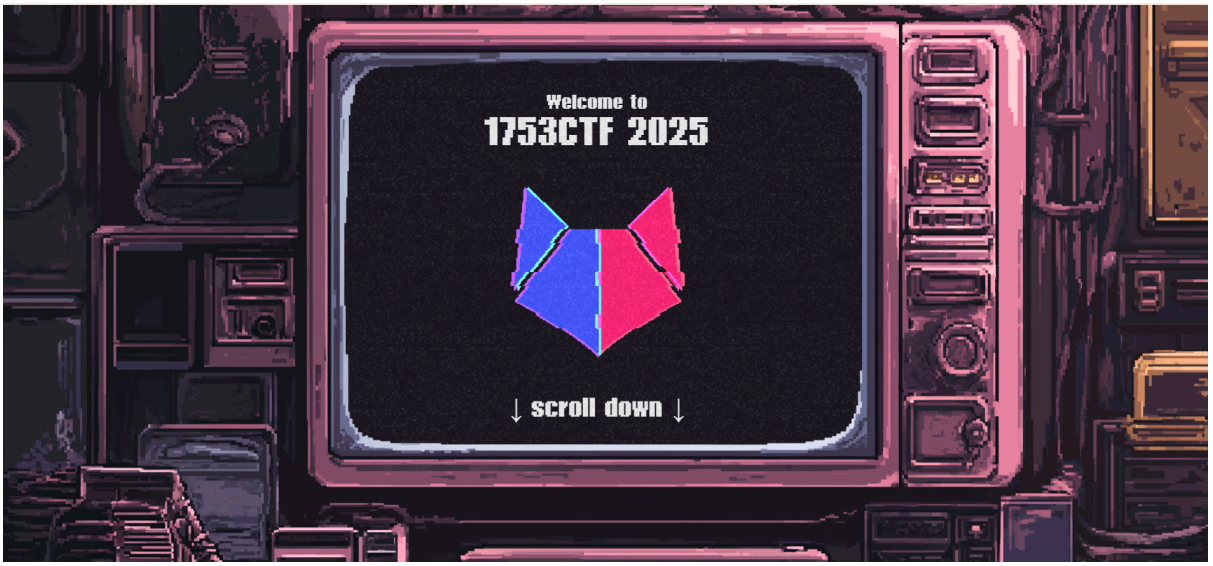
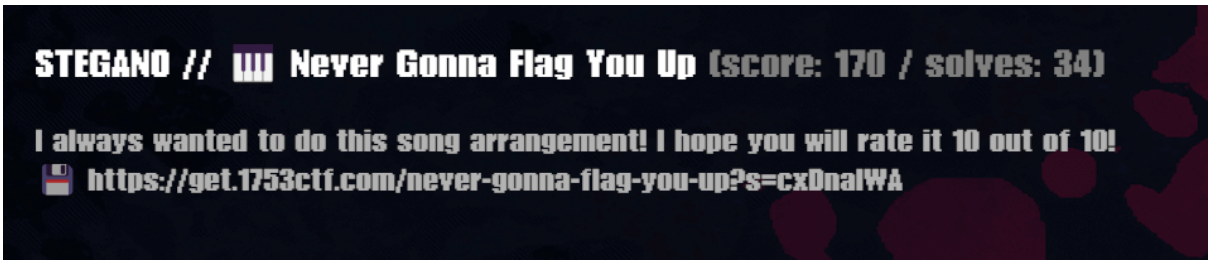


1. Never Gonna Flag You Up (Steganography)



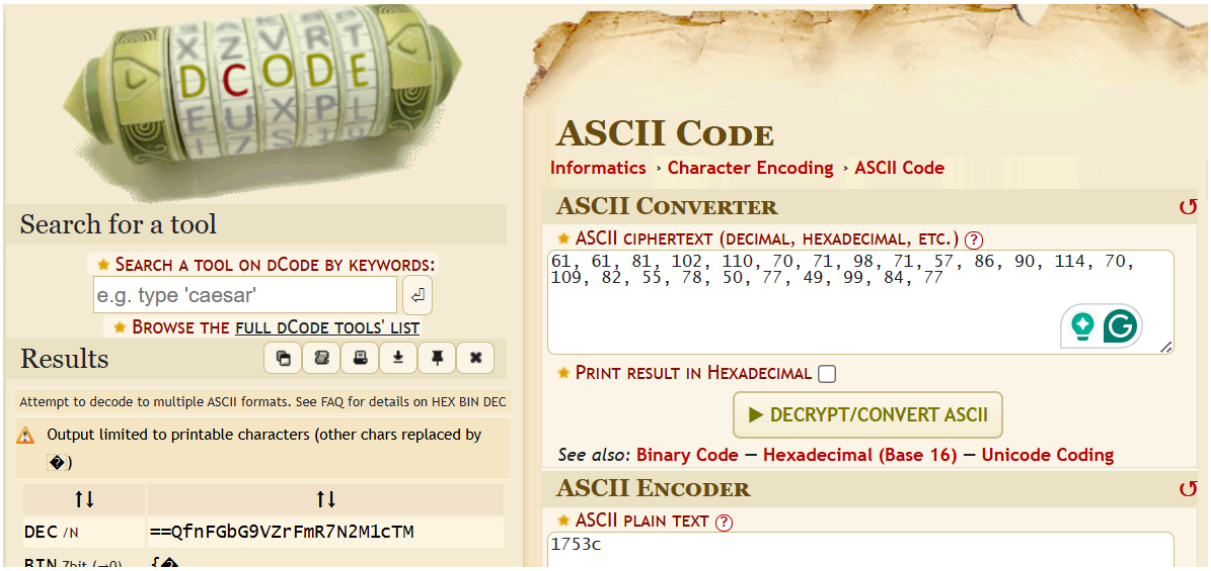
I was given a midi file (final_version_v199237.mid). The first thing I tried is to open it in audiotonic software to inspect. However, there is nothing interesting in it.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ file final_version_v199237.mid
final_version_v199237.mid: Standard MIDI data (format 0) using 1 track at 1/480

┌──(kali㉿kali)-[~/Desktop]
└─$ midicsv final_version_v199237.mid > output.csv
```

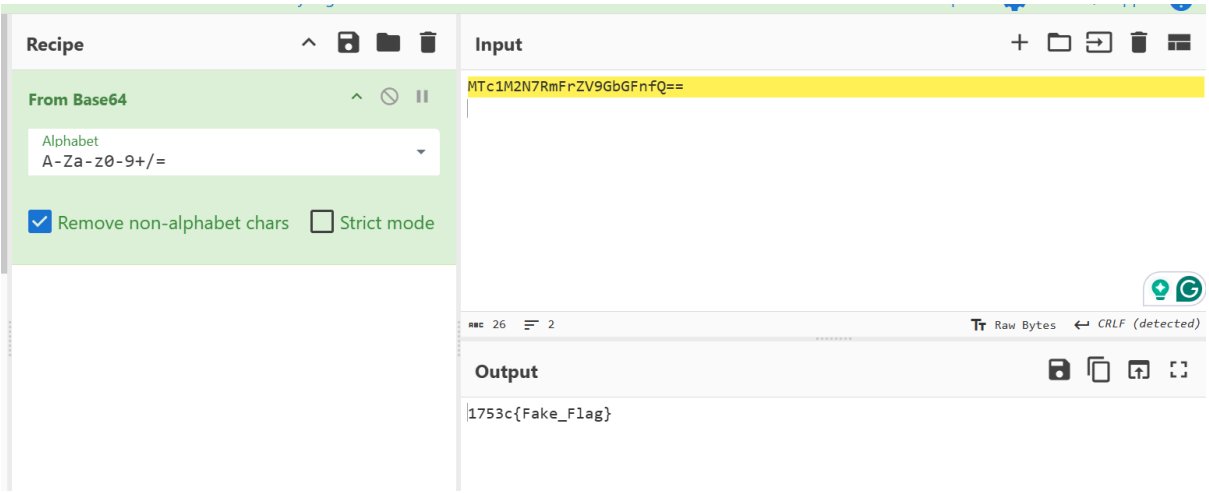
Next, I search on the internet on how to view midi data and apparently, we could turn the file into .csv to view the details of the music score. Therefore, I use the command **midicsv** to turn the midi file into csv.

```
0, 0, Header, 0, 1, 480
1, 0, Start_track
1, 0, Note_on_c, 0, 55, 64
1, 0, Program_c, 0, 61
1, 0, Program_c, 0, 61
1, 0, Program_c, 0, 81
1, 0, Program_c, 0, 102
1, 0, Program_c, 0, 110
1, 0, Program_c, 0, 70
1, 0, Program_c, 0, 71
1, 0, Program_c, 0, 98
1, 0, Program_c, 0, 71
1, 0, Program_c, 0, 57
1, 0, Program_c, 0, 86
1, 0, Program_c, 0, 90
1, 0, Program_c, 0, 114
1, 0, Program_c, 0, 70
1, 0, Program_c, 0, 109
1, 0, Program_c, 0, 82
1, 0, Program_c, 0, 55
1, 0, Program_c, 0, 78
1, 0, Program_c, 0, 50
1, 0, Program_c, 0, 77
1, 0, Program_c, 0, 49
1, 0, Program_c, 0, 99
1, 0, Program_c, 0, 84
1, 0, Program_c, 0, 77
1, 0, Note_off_c, 0, 55, 64
1, 0, Channel_prefix, 0
1, 0, Title_t, "final_version_v199237"
1, 0, Instrument_name_t, "Inst 1"
1, 0, Time_signature, 4, 2, 24, 8
1, 0, Key_signature, 0, "major"
1, 0, SMPTE_offset, 33, 0, 0, 0, 0
1, 0, Tempo, 631579
1, 1454, Note_on_c, 0, 56, 81
1, 1510, Channel_aftertouch_c, 0, 0
1, 1570, Note_on_c, 0, 58, 81
1, 1617, Note_off_c, 0, 56, 60
1, 1687, Note_on_c, 0, 61, 81
1, 1698, Note_off_c, 0, 58, 90
1, 1802, Note_on_c, 0, 58, 81
1, 1821, Channel_aftertouch_c, 0, 0
```

By inspecting the csv file, I found some suspicious text like Program_c and I tried to combine the numbers at the end of Program_c together and tried ascii code on dcode to decode it.



From the results, it looks like a base64 strings in reverse. Therefore, I reversed the string and throw it in cyberchef to decipher using base64.



However, this turns out to be a fake flag! So I start to play around to see if there is a pattern of 1753c in ascii code in the csv file.

The ascii code of 1753c is 49 55 53 51 99. So I started looking for these pattern in the csv file. Surprisingly, I noticed that the line Note_on_c is on mostly ends with 81 and for those that didn't end with 81, I notice the pattern that I was looking for, 49 55 53 51 99 …

I created a python script to help me combine all of the numbers that is in the same line as Note_on_c but don't end with 81 and convert them to ascii strings.

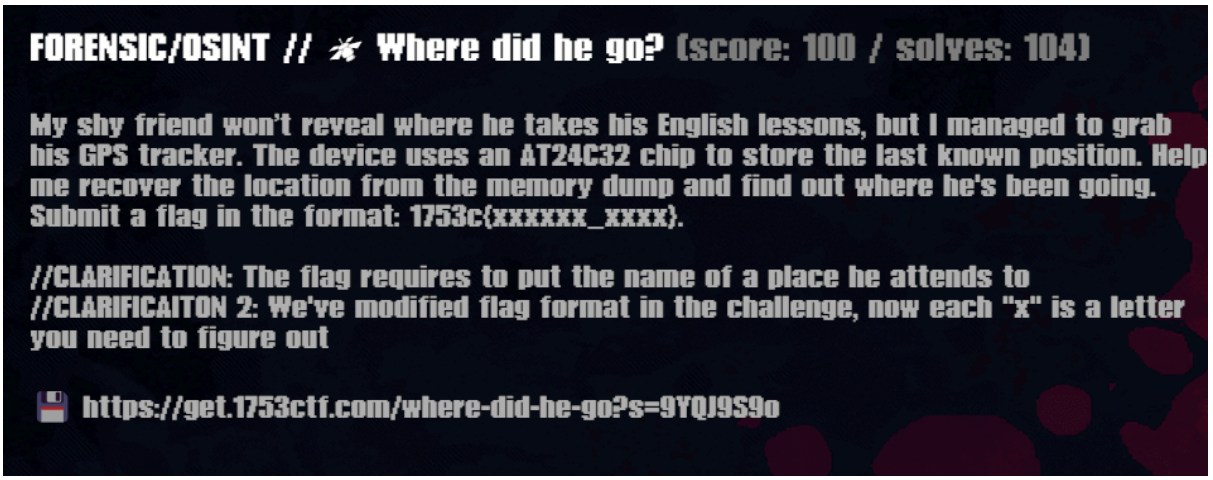
```
# Path to your midi_stego.txt file
input_file = 'midi stego.txt'

# Store the note numbers that meet the criteria
suspicious_notes = []

# Open the file and process each line
```

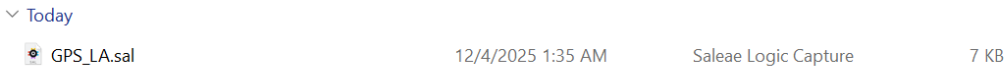
```
with open(input_file, 'r') as file:
    for line in file:
        line = line.strip()  # Remove leading/trailing whitespace
        # Check if the line contains 'Note_on_c'
        if 'Note_on_c' in line:
            parts = line.split(', ')  # Split by comma and space
            if len(parts) >= 6:
                velocity = parts[-1]  # The last element (velocity)
                # If velocity doesn't end with 81, capture the note
number
                if velocity != '81':
                    note_number = parts[5]  # The note number is in the
5th position (index 4)
                    suspicious_notes.append(note_number)

# Print the suspicious note numbers
print("Suspicious note numbers (Note_on_c not ending with 81):")
print(suspicious_notes)

# Optionally, you could convert these numbers to ASCII if needed
ascii_text = ''.join(chr(int(n)) for n in suspicious_notes if 32 <=
int(n) <= 126)
print("\nPossible flag in ASCII:")
print(ascii_text)
```

```
Suspicious note numbers (Note_on_c not ending with 81):
['64', '49', '55', '53', '51', '99', '123', '82', '105', '99', '107', '95', '65', '115', '116', '108', '101', '121', '95'
['64', '49', '55', '53', '51', '99', '123', '82', '105', '99', '107', '95', '65', '115', '116', '108', '101', '121', '95'
, '73', '115', '95', '80', '114', '111', '117', '100', '95', '79', '102', '95', '89', '111', '117', '125']

Possible flag in ASCII:
@1753c{Rick_Astley_Is_Proud_Of_You}
```

Flag: 1753c{Rick_Astley_Is_Proud_Of_You}

2. Where Did He Go (Forensic / OSINT)

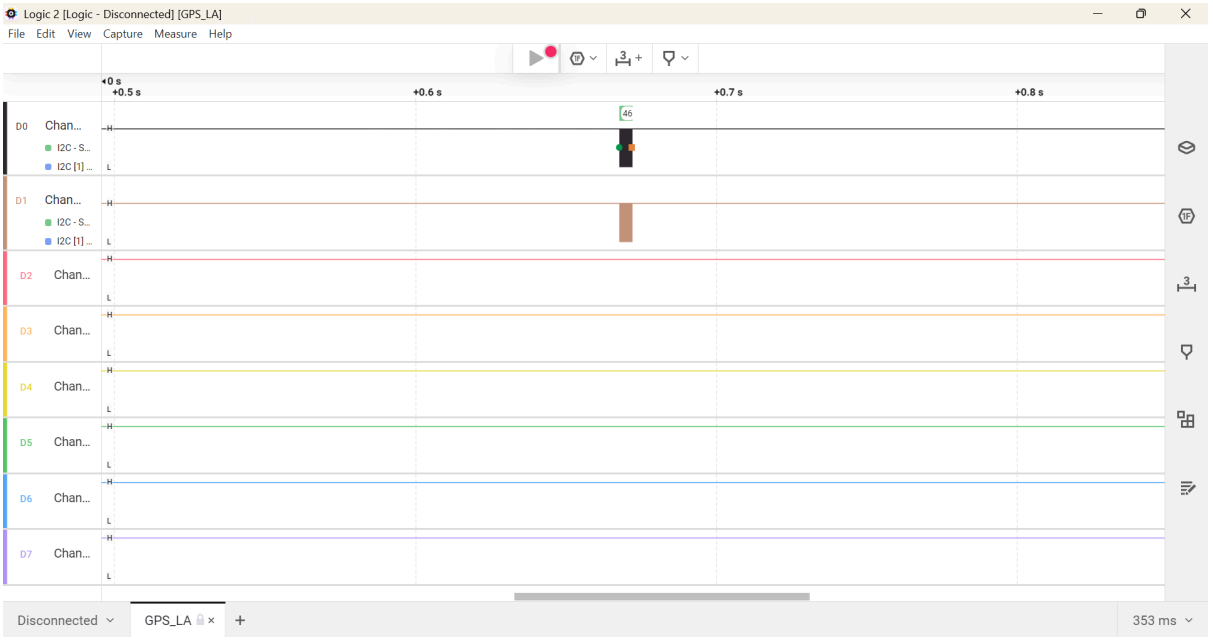




I am given a file name GPS_LA.sal. I've never heard of a .sal file so I search for the internet to look for ways to open a .sal file.

A .sal file, in the context of Saleae Logic Analyzers, is the file format used to store captured digital data. It's essentially a compressed archive containing the captured data along with metadata specific to the analyser's software, such as acceleration rendering information.

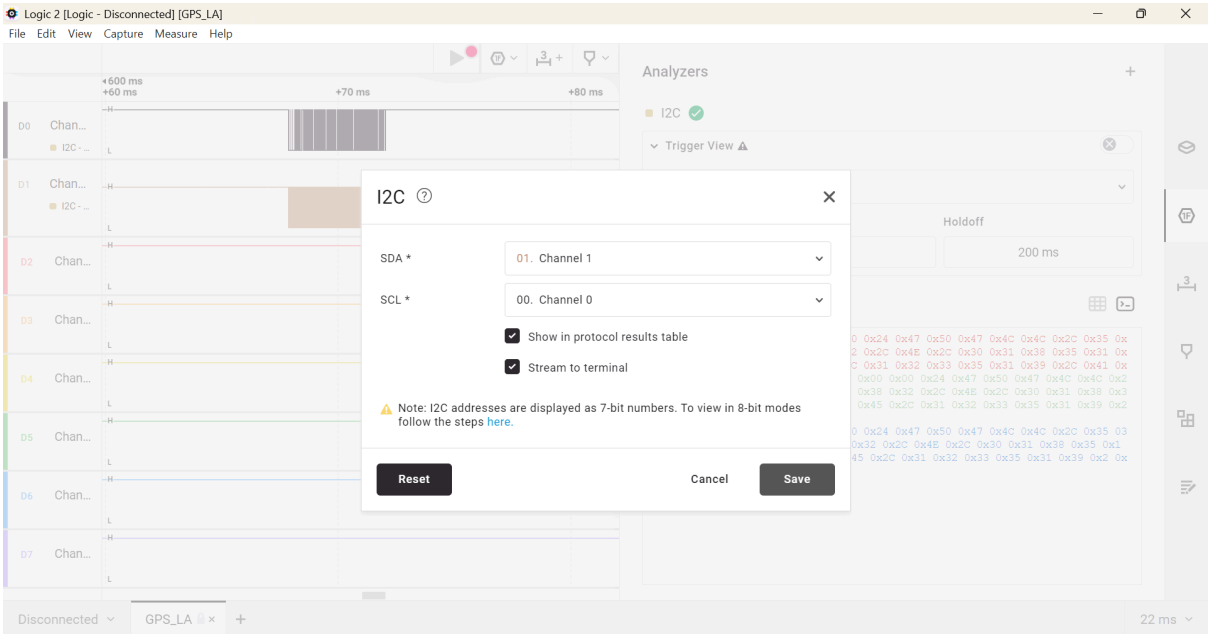
I also found out that there is a software call Logic 2 that can open and analyse a .sal file.



The first thing I saw is that there are 7 channels and there are multiple analysers that I can choose to use from the sidebar so I decided to search for more information on the internet on which analyser is more suitable to be used. I found that:

The I2C analyser was used to interpret communication between the GPS module and the microcontroller. Since the raw electrical signals on these lines are not human-readable, the I2C analyser decodes them into structured data frames showing device addresses, transmitted bytes, and acknowledgements.

In this case, the I2C analyser might helped to identify the GPS module's address and extract NMEA sentence data (e.g., $GPGLL,...) byte by byte.



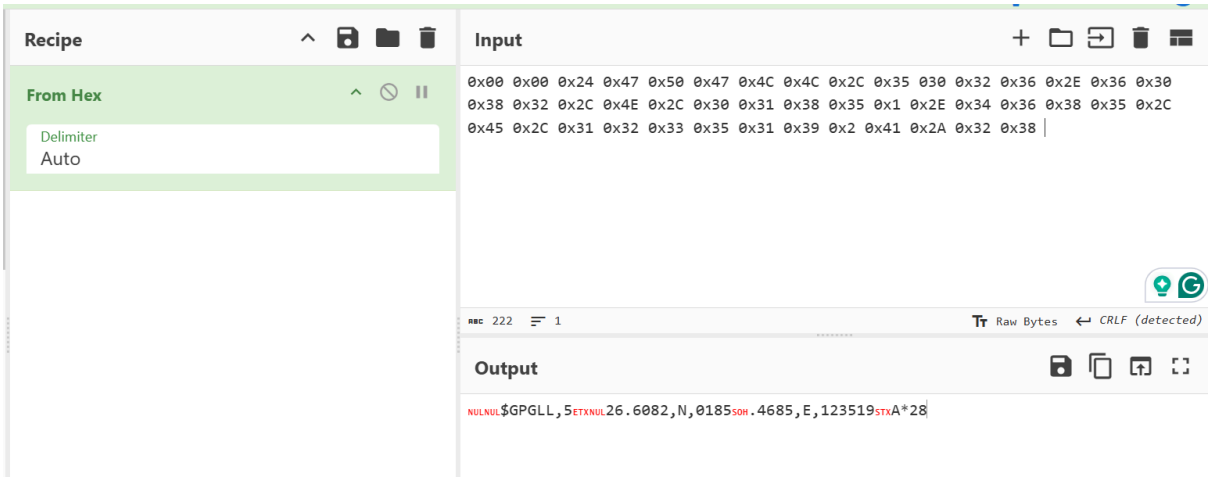
I2C uses two lines:

- **SDA (Serial Data Line)** – for data transfer
- **SCL (Serial Clock Line)** – for timing and synchronization

Data

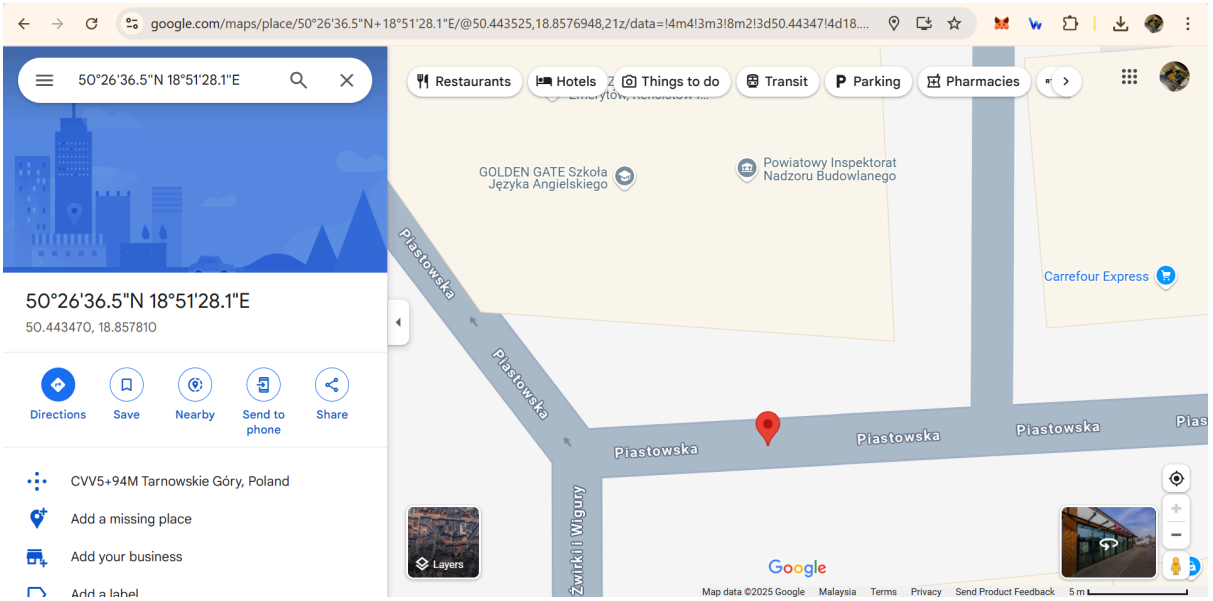
```
write to 0x50 ack data: 0x00 0x00 0x24 0x47 0x50 0x47 0x4C 0x4C 0x2C 0x35 0x
x32 0x36 0x2E 0x36 0x30 0x38 0x32 0x2C 0x4E 0x2C 0x30 0x31 0x38 0x35 0x31 0x
x34 0x36 0x38 0x35 0x2C 0x45 0x2C 0x31 0x32 0x33 0x35 0x31 0x39 0x2C 0x41 0x
x32 0x38 write to 0x50 ack data: 0x00 0x00 0x24 0x47 0x50 0x47 0x4C 0x4C 0x2
35 0x30 0x32 0x36 0x2E 0x36 0x30 0x38 0x32 0x2C 0x4E 0x2C 0x30 0x31 0x38 0x3
31 0x2E 0x34 0x36 0x38 0x35 0x2C 0x45 0x2C 0x31 0x32 0x33 0x35 0x31 0x39 0x2
41 0x2A 0x32 0x38
write to 0x50 ack data: 0x00 0x00 0x24 0x47 0x50 0x47 0x4C 0x4C 0x2C 0x35 03
0 0x32 0x36 0x2E 0x36 0x30 0x38 0x32 0x2C 0x4E 0x2C 0x30 0x31 0x38 0x35 0x1
0x2E 0x34 0x36 0x38 0x35 0x2C 0x45 0x2C 0x31 0x32 0x33 0x35 0x31 0x39 0x2 0x
41 0x2A 0x32 0x38
```

After running the I2C analyser, this is the result that it shows. I then copied the ack data and put it in Cyber Chef to convert from hex to text.



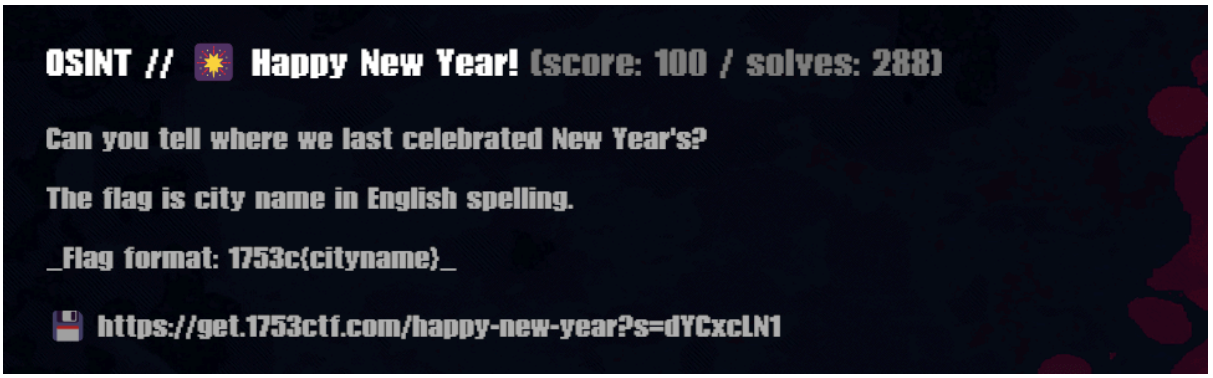
After running the cipher, I can determine that:
Latitude: 50.44347° N
Longitude: 18.85781° E

I then put it in google maps to search for the location and found a language school name GOLDEN GATE.

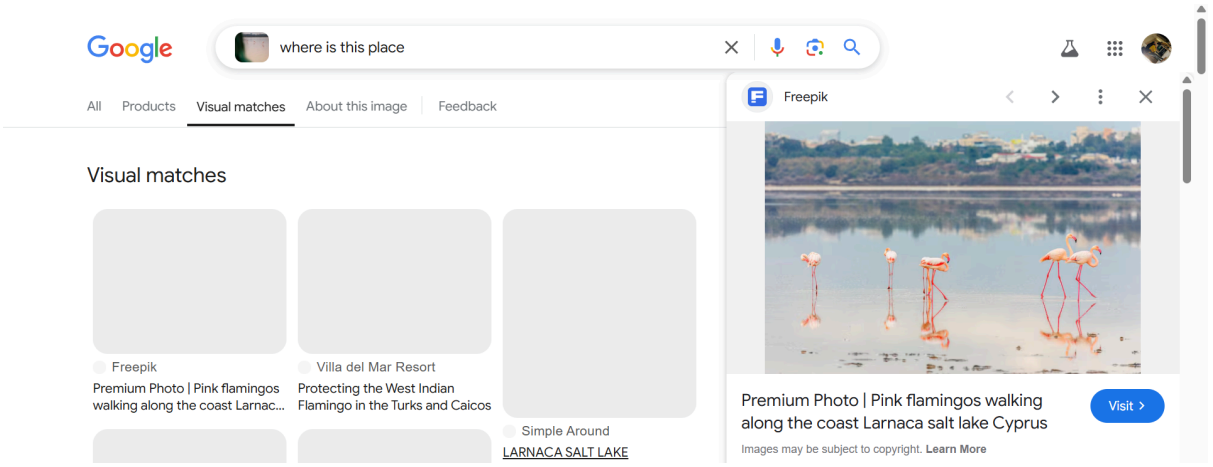


Flag: 1753c{GOLDEN_GATE}

3. Happy New Year! (OSINT)



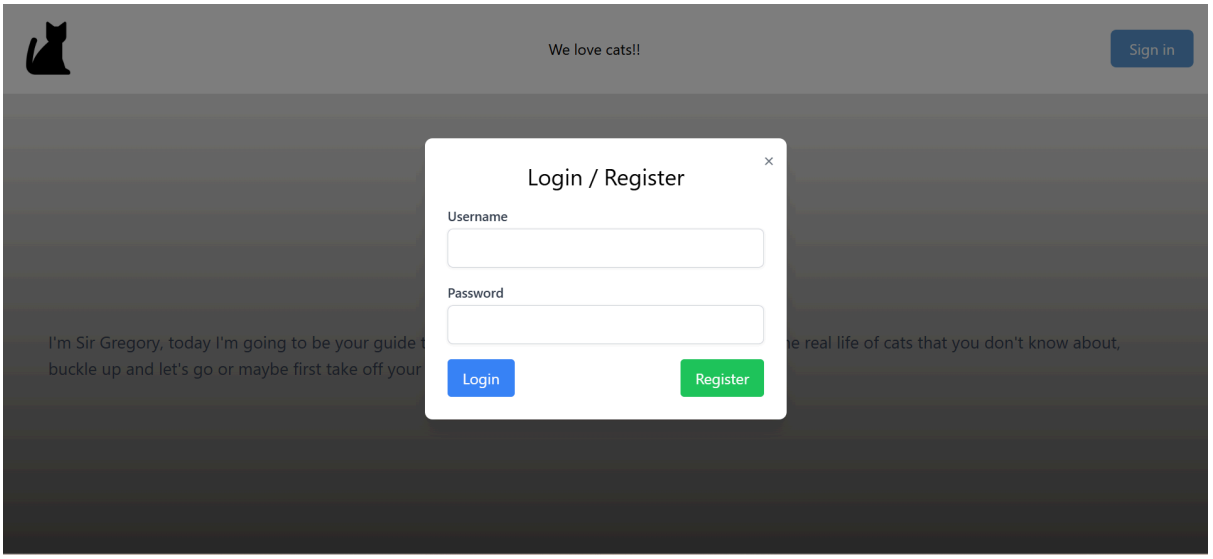
This challenge is pretty easy. I'm given an image to analyse and the goal is to find out which city the picture is taken in. I just put it into Google lens to identify similar places and found a picture that looks similar to it. The picture is taken in the city of Larnaca.



Flag: 1753c{Larnaca}

4. Escatlate flag #1 (Web)

I am given the link to the website and also a source code folder to be downloaded. This is what the website looks like:



In the source code folder, Index.js caught my eye because of a part of the code:

Index.js

```
app.post('/api/register', (req, res) => {

    const existingUser = users.find(u => u.username ==
req.body.username);
    if(existingUser)
        return res.status(400).send('User already exists');

    if(req.body.role?.toLowerCase() == 'admin')
        return res.status(400).send('Invalid role');

    const user = {
        username: req.body.username.substring(0, 20),
        password: req.body.password.substring(0, 20),
```

```
        token: crypto.randomBytes(32).toString('hex'),
        role: req.body.role.substring(0, 20) || 'user'
    }

    users.push(user);

    res.json(user);
})
app.get('/api/message', (req, res) => {
    if(req.user.role.toUpperCase() === 'ADMIN')
        return res.json({ message: `Hi Admin! Your flag is
${process.env.ADMIN_FLAG}` });

    if(req.user.role.toUpperCase() === 'MODERATOR')
        return res.json({ message: `Hi Mod! Your flag is
${process.env.MODERATOR_FLAG}` });

    res.json({ message: `Hello ${req.user.username}` });
})
```

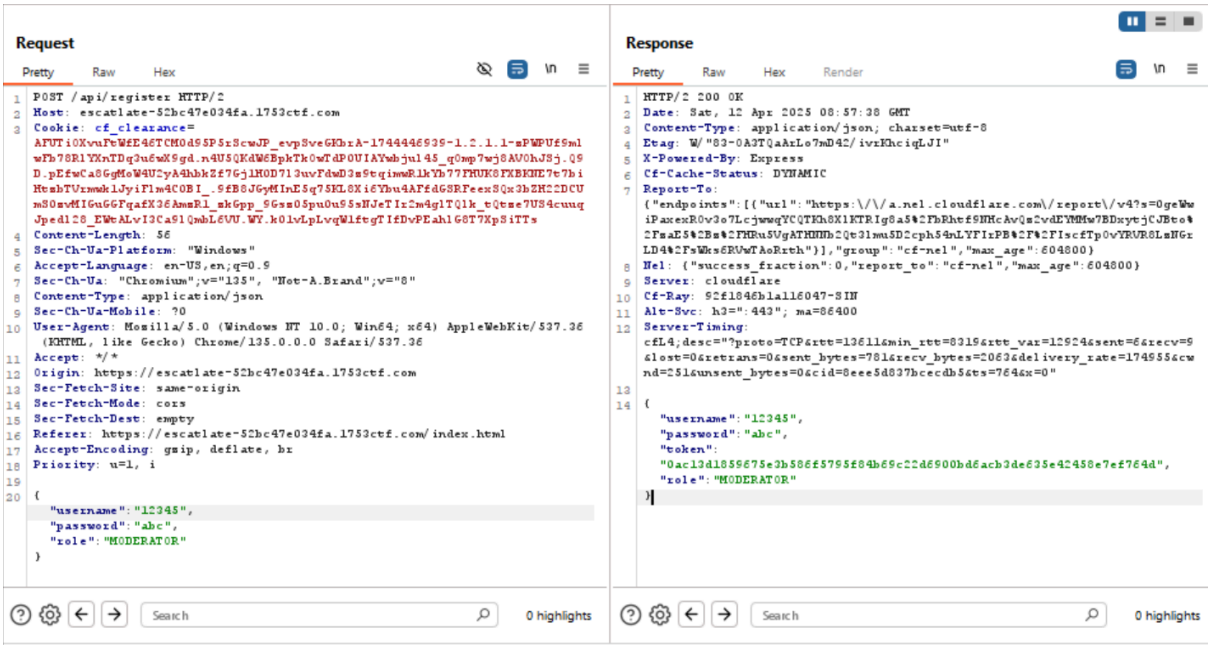
This shows that I either have to be an “ADMIN” or “MODERATOR” role to get the flag. Since there is a role checking for admin, I will try registering as a moderator first to try get the flag.

However, according to Auth.js, every user that registers will be automatically assigned a role called “user”. Therefore, I will use Burp Suite to try to be the man in the middle and manipulate the requested data.

Auth.js

```
// Register
registerBtn.addEventListener("click", async function () {
    const username = usernameInput.value.trim();
    const password = passwordInput.value.trim();

    if (!username || !password) {
        showError("Please enter both username and password!");
        return;
    }

    try {
        const response = await fetch("/api/register", {
            method: "POST",
            headers: {
```

```
                "Content-Type": "application/json"
            },
            body: JSON.stringify({
                username,
                password,
                role: "user"
            })
        });

        const data = await response.json();

        if (response.ok) {
            storeUserData(data);
            redirectToDashboard();
        } else {
            showError(data || "Registration failed!");
        }
    } catch (error) {
        showError("Server error! Please try again later.");
    }
});
```

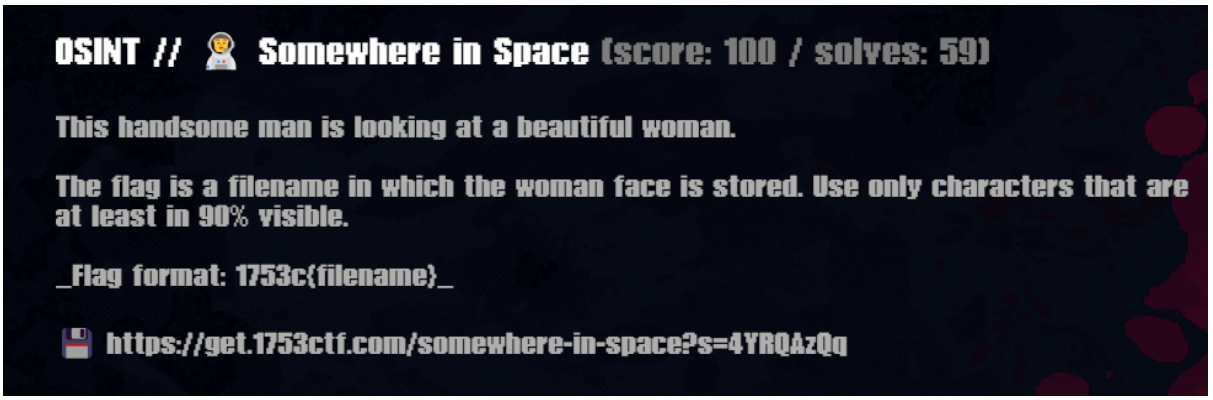


After modifying the registration request by changing **"role": "user"** to **"role": "MODERATOR"**, I successfully created an account with the moderator role. Using the token returned upon successful registration, I then sent a **curl** request to the server, authenticated as a moderator, and was able to retrieve the flag.

```
└─$ curl -X GET https://escatlate-52bc47e034fa.1753ctf.com/api/message \
  -H "x-token: 0ac13d1859675e3b586f5795f84b69c22d6900bd6acb3de635e42458e7ef764d"

{"message":"Hi Mod! Your flag is 1753c{0h_my_g0d_h0w_c0uld_1_m1ss_thi1_r0l3_ch3ck}"}
```

Flag: 1753c{0h_my_g0d_h0w_c0uld_1_m1ss_thi1_r0l3_ch3ck}

5. Somewhere in Space (OSINT)



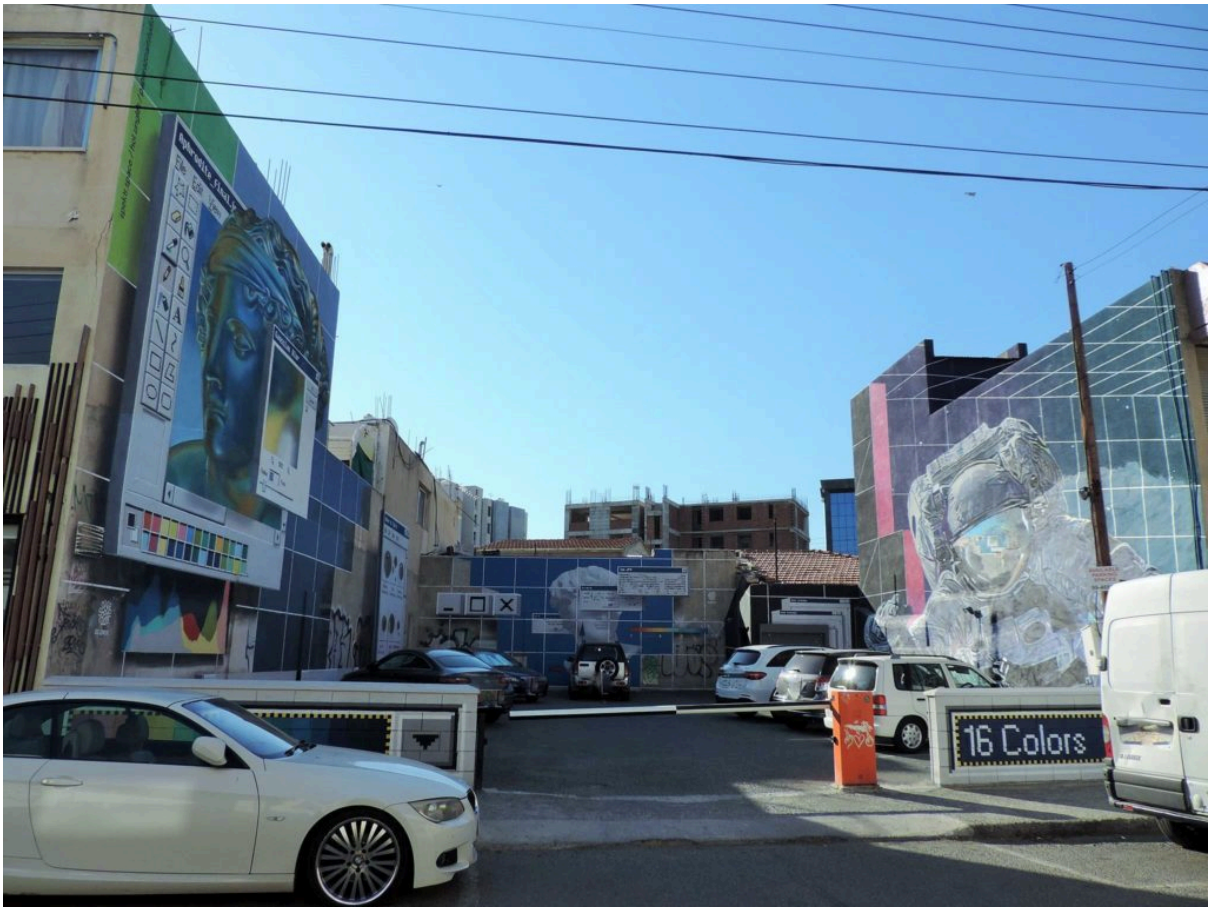
I'm given an image:

After putting it in Google Lens, I found the picture in the link below.

https://vagabundler.com/cyprus/streetart-map-limassol/agiou-andreou-253/

On the website, I found an image that shows the astronaut wall art as the opposite of another wall art that looks like a woman and it fits the description of the challenge

where the astronaut is the handsome man and the woman wall art across from it is te beautiful woman he is looking at.



Scrolling down on the pictures, I found a larger and clearer image of the wall art of the woman and the woman's face is stored on the file name Aphrodite_final.

Since the challenge requires us to include the name that is 90% visible, the flag is: **1753c{Aphrodite_final.j}**