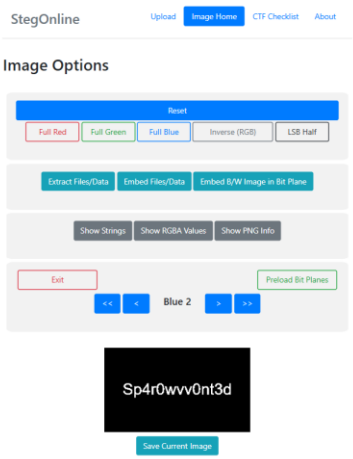
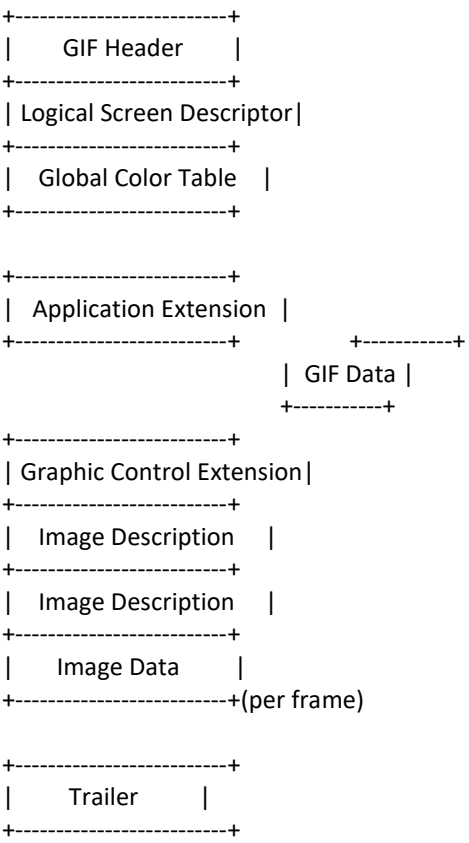
1.We are given with the zip file in which we find wanted.png and esacpe.zip which is locked.
2. On investigating the png file, we find a password in the 2nd blue bit planes of the png.



3. Use the password: "Sp4r0wvv0nt3d" to unzip the locked escape.zip.
4. In escape.zip file, we get locked Last_horizon.zip and SPARROWSPOTTED!!.gif file.
5. The gif appears black and white and we get no such hint or information from any known tools.
6. Now let us dive deep into what are exactly gifs and how do they work.



A.this was the main structure of the gif file , there are mainly two version of GIF87a and GIF89a , the difference is that GIF89a supports transparency and animation
B.Logical Screen Descriptor - it contains the width and height of the gif , and some flags to indicate if there is a global color table or not
C.Global Color Table - it contains the colors used in the gif , each color is represented by 3 bytes (RGB)
D.Application Extension - it contains some metadata about the gif like the loop count , block size etc

E.Image Descriptor - this is the per frame data , it contains the position of the frame in the logical screen , and some flags to indicate if there is a local color table or not
F.Local Color Table - it contains the colors used in the frame , each color is represented by 3 bytes (RGB)
G.Graphics Control Extension - It tells each frame how long to stay on screen.
H.Image Data - This containts the actual picture data for each frame.
I.Trailer - this contains a single byte to indicate the end of the gif file containing hexadecimal 0x3B which is ';' in ascii

7. The next step was to check if there is a global color table or not , and if there is a local color table or not
8. On carefully studying the gif file structure, we get to know that there is Global Color Table and also Local color table with 256 entries per frame
9. We will use the script:

```
def format_palette_bytes(palette_bytes):
    if not palette_bytes:
        return "N/A"
    colors = [tuple(palette_bytes[i:i+3]) for i in range(0, len(palette_bytes), 3)]
    return ", ".join(map(str, colors))

def skip_data_sub_blocks(file_handle):
    while True:
        block_size = file_handle.read(1)
        if not block_size or block_size == b'\x00':
            break
        file_handle.read(int.from_bytes(block_size, 'little'))
```

```
def find_all_lcts(file_path, output_path):
    with open(file_path, 'rb') as f, open(output_path, 'w') as log:
        log.write(f"Local Color Table Analysis for: {file_path}\n")
        log.write("=" * 50 + "\n\n")
```

```
        # --- Header and Logical Screen Descriptor ---
        f.read(10) # Skip header and screen dimensions
        packed_fields = int.from_bytes(f.read(1), 'little')
        gct_flag = (packed_fields & 0x80) >> 7
        gct_size_val = packed_fields & 0x07
        f.read(2) # Skip background color index and pixel aspect ratio
```

```
        # --- Skip Global Color Table (GCT) ---
        if gct_flag:
            gct_length = 3 * (2 ** (gct_size_val + 1))
            f.read(gct_length)
```

```
        # --- Main Loop to Find All Frames ---
        frame_counter = 0
        while True:
            block_type = f.read(1)
            if not block_type:
                log.write("End of file reached.\n")
                break
```

```
            # Extension Block
            if block_type == b'\x21':
                f.read(1)
                skip_data_sub_blocks(f)
```

```
            # new frame
            elif block_type == b'\x2C':
```

```
                log.write(f"--- Frame {frame_counter} ---\n")
                f.read(8)

                packed_field = int.from_bytes(f.read(1), 'little')
                lct_flag = (packed_field & 0x80) >> 7
                lct_size_val = packed_field & 0x07
```

```
                if lct_flag:
                    log.write("Status: Local Color Table (LCT) FOUND\n")
                    lct_length = 3 * (2 ** (lct_size_val + 1))
                    lct_data = f.read(lct_length)
                    log.write(f"Size: {lct_length} bytes ({lct_length // 3} colors)\n")
                    log.write(f"Values: {format_palette_bytes(lct_data)}\n\n")
                else:
                    log.write("Status: No LCT found (uses GCT)\n\n")

                #Skip the actual image data to get to the next block
                f.read(1) # LZW Minimum Code Size
                skip_data_sub_blocks(f)
                frame_counter += 1
```

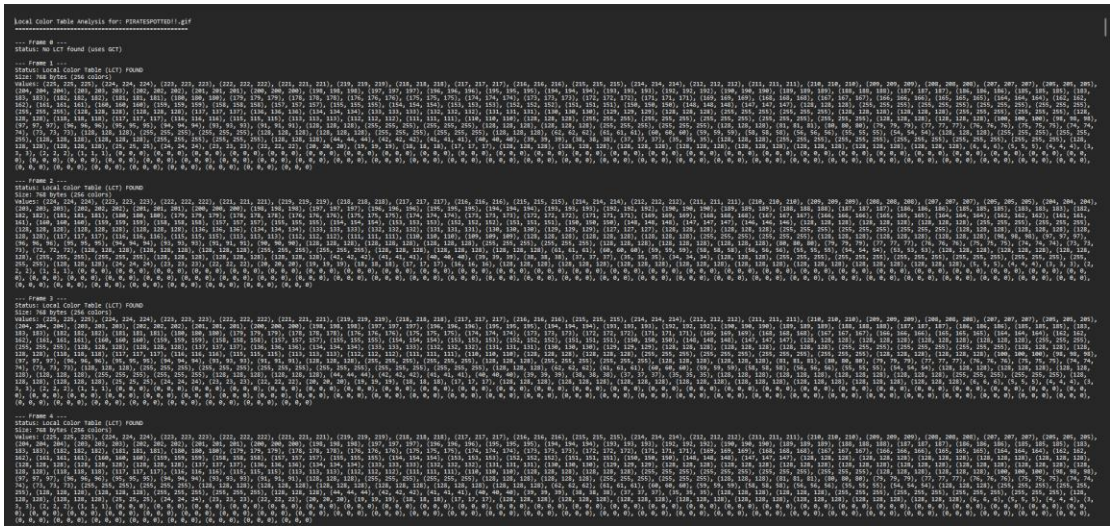
```
            # Trailer
            elif block_type == b'\x3B':
                log.write("GIF Trailer found. End of image data.\n")
                break

            else:
                log.write(f"Unknown block type {block_type.hex()} found. Stopping parse.\n")
                break

    print(f"[SUCCESS] palette analysis complete. See '{output_path}'.")
```

```
if __name__ == "__main__":
    gif_filename = "PIRATESPOTTED!!.gif"
    log_filename = "LCT.txt"
    find_all_lcts(gif_filename, log_filename)
```

Explanation of LCT code

A.format_palette_bytes(palette_bytes) - this function takes a flat list of palette bytes and formats them into a readable list of (R, G, B) tuples
B.Skip_data_sub_blocks(file_handle) - this function skips over GIF data sub-blocks
C.find_all_lcts(file_path, output_path) - this function skips GIF header and Global Color Table if present , then iterates through the blocks to find Image Descriptor blocks (frames) and checks for
D.Local Color Tables (LCTs) , then the trailer block and then writes the results to file

10. This outputs:



11.As we can see that there is a local color table for each frame and it is unqiue and a gct for the whole gif , we try to visualize colors in the gif
12.Using per frame local color table i created a image of 16x16 pixels (256 colors) and filled each pixel with the color from the local color table using this script

```
import re
import os
from PIL import Image

def parse_log_file(log_path):
    """
    Parses the detailed log file to extract the LCT for each frame.
    Returns a list of palettes.
    """
    all_palettes = []
    try:
        with open(log_path, 'r') as f:
            content = f.read()
    except FileNotFoundError:
        print(f"[ERROR] The log file '{log_path}' was not found.")
        print("[INFO] Please make sure it's in the same directory as this script.")
        return None

    # Use regular expressions to find all blocks of LCT data
    lct_blocks = re.findall(r"Status: Local Color Table \(LCT\) FOUND.*?Values: (.*?)\n\n",
content, re.S)
    print(f"[INFO] Found {len(lct_blocks)} LCTs in the log file.")
```

```
    for values_str in lct_blocks:
        try:
            # Safely evaluate the string "(r, g, b), ..." into a list of tuples
            palette = list(eval(values_str))
            all_palettes.append(palette)
        except Exception as e:
            print(f"[ERROR] Could not parse palette data: {values_str[:50]}... Error: {e}")

    return all_palettes
```

```
def visualize_lcts(palettes):
    output_dir = "flag_characters1"
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
```

```
        print(f"[INFO] Created directory: {output_dir}")
```

```
    BLOCK_SIZE = 20  # Size of each color block in pixels
    GRID_SIZE = 16   # The grid is 16x16
```

```
    char_count = 0
    for idx, lct in enumerate(palettes):
        # A 16x16 grid requires exactly 256 colors. Filter for these palettes.
        if len(lct) != 256:
            continue
```

```
        # Create a new image for this character
        char_image = Image.new("RGB", (BLOCK_SIZE * GRID_SIZE, BLOCK_SIZE * GRID_SIZE))
```

```
        # Fill the image with 16x16 blocks of color from the LCT
        for color_idx, color in enumerate(lct):
            row = color_idx // GRID_SIZE
            col = color_idx % GRID_SIZE

            # Top-left corner of the block
            start_x = col * BLOCK_SIZE
            start_y = row * BLOCK_SIZE

            # Draw the block
            for x in range(BLOCK_SIZE):
                for y in range(BLOCK_SIZE):
                    char_image.putpixel((start_x + x, start_y + y), color)
```

```
        # Save the resulting character image
        output_filename = os.path.join(output_dir, f"char_{char_count:03d}.png")
        char_image.save(output_filename)
        char_count += 1
```

```
    if char_count > 0:
        print(f"\n[SUCCESS] Created {char_count} character images in the '{output_dir}'
folder.")
        print("[INFO] Open the folder and view the images in order by filename to reveal the
flag.")
    else:
        print("\n[FAIL] No 256-color LCTs were found to visualize.")
```

```
if __name__ == "__main__":
    log_filename = "LCT.txt"
    all_lcts = parse_log_file(log_filename)

    if all_lcts:
        visualize_lcts(all_lcts)
```

13.Explanation of visualization code

A.parse_log_file(log_path) - this function reads the log file using regex “Values: (…)” , then converts each block of RGB to a list of tuples and retunrs it
B.visualize_lcts(palettes) - this function first makes a directory to store images , then loops through all the palettes with 256 colors , then for each palette it creates a blank image then fills it 16x16 grid with each color from the palette and saves it as char_000.png , char_001.png etc\
C.main - this calls the two function

14. After running this script, we get

15. Using this as the password for the next file,We get Last_Horizon.gif.
16. For this we can use a gift tool available on github https://github.com/dtmsecurity/gift
17. Using the gather function of this tool using "python3 gifttool-cli.py --source Last_Horizon.gif gather file ”,
18. The file our flag in it..DJSISACA{HI_JACK_SPARROW_HERE}