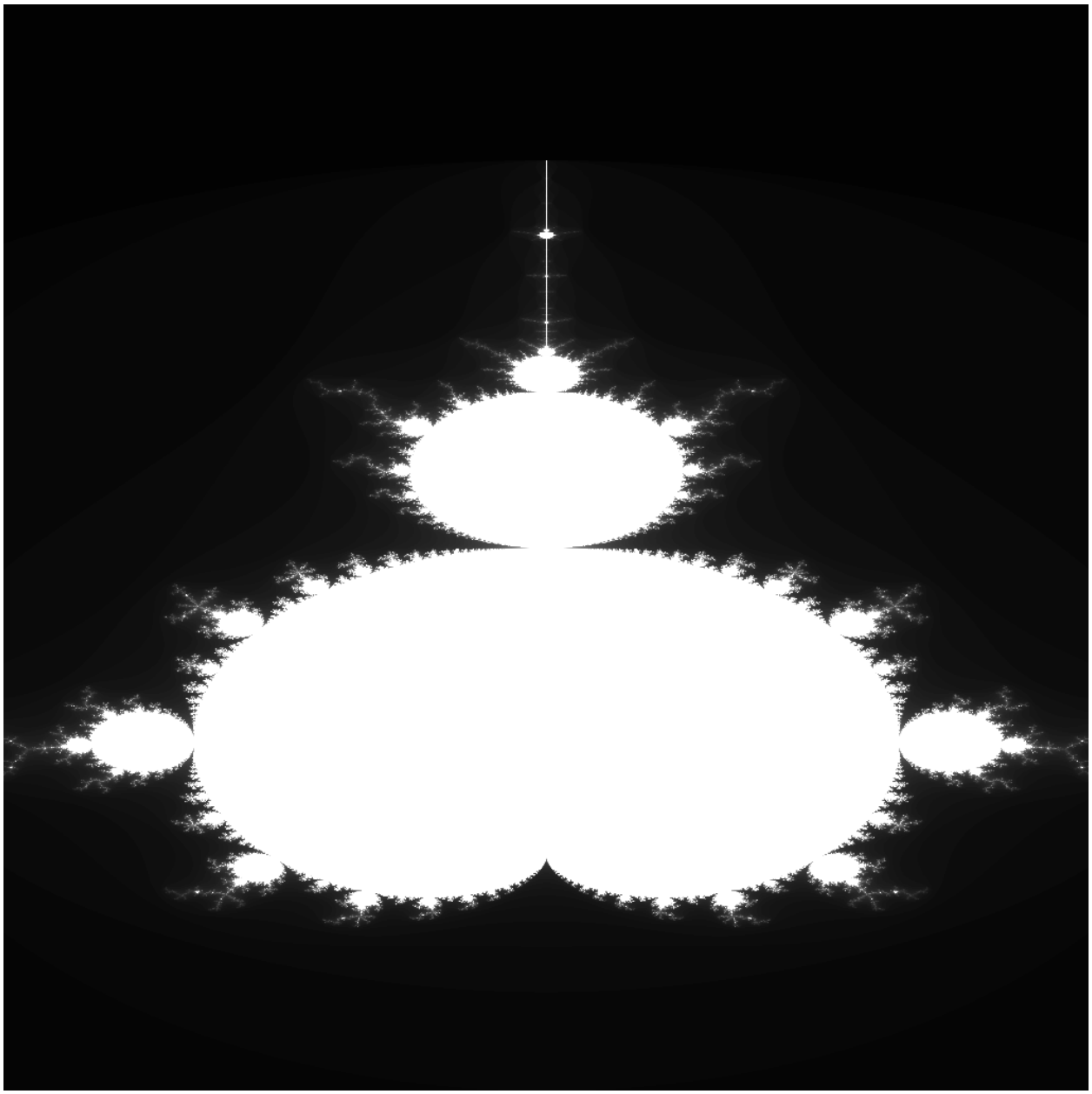
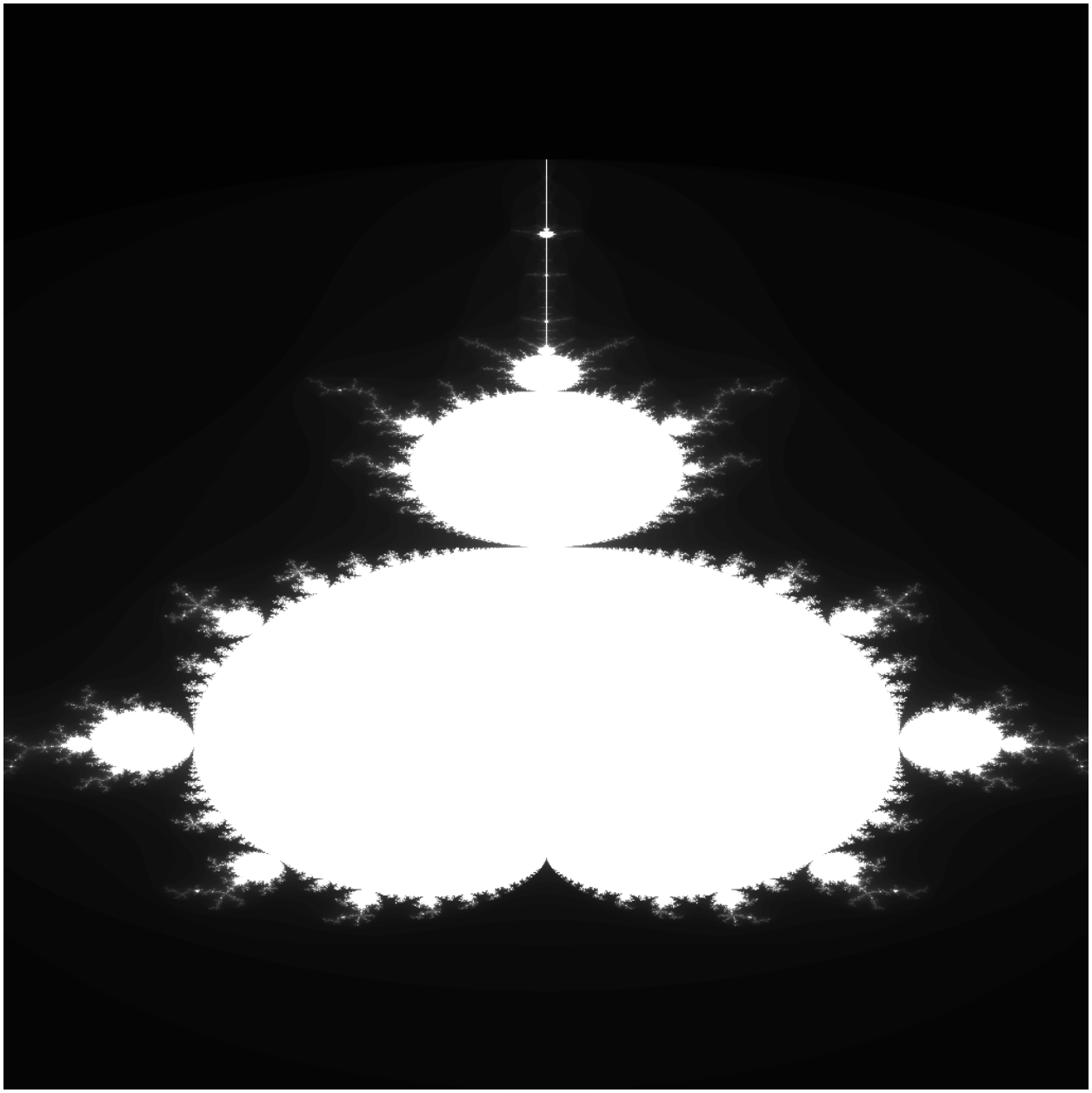
Writeup for Challenge Fractal

Gojo Satoru from Jujutsu Kaisen is known for his striking color palette. His signature colors include vibrant blue for his Limitless technique, radiant purple for Hollow Purple, and icy white for his hair, symbolizing his overwhelming power. His piercing blue eyes, revealed when he removes his blindfold, reflect the Six Eyes' immense precision and depth.



Okay, we have a beautiful fractal pattern that somehow includes the flag.

Since we also have `challenge.py`, which contains the code to generate the fractal pattern without the flag, let's use it to generate the pattern.



There seems to be no difference… *or is there?*

This is a clear indication of LSB steganography, where the least significant bits (LSB) are modified while the image appears unchanged.

Some of you might have tried using `zsteg -a` to extract the flag, but it's not that simple...

Let's convert both images into their respective NumPy array representations and spot the differences

```
import numpy as np
from PIL import Image

def generate_fractal(size=1024, max_iter=100):
    fractal = np.zeros((size, size), dtype=np.uint8)

    for x in range(size):
        for y in range(size):
            real = (x / size) * 3.5 - 2.5
            imag = (y / size) * 2.0 - 1.0
            c = complex(real, imag)
            z = complex(0, 0)
            iter_count = 0

            while abs(z) < 2 and iter_count < max_iter:
                z = z*z + c
                iter_count += 1

            fractal[x, y] = int((iter_count / max_iter) * 255)

    return fractal

def extract_numpy_array(image_path="fractal.png"):
    image = Image.open(image_path).convert('L')
    fractal = np.array(image)
    return fractal

fractal_with_flag = extract_numpy_array()
fractal = generate_fractal()
```

```
diff_array = []
for i in range(1024):
   for j in range(1024):
       if(fractal[i][j]!=fractal_with_flag[i][j]):
           diff_array.append((fractal_with_flag[i][j],fractal[i][j]))


print(diff_array)
```

```
$ python solution.py
[(254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255),
(254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (
254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (2
54, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (225, 224), (254, 255), (25
4, 255), (254, 255), (254, 255), (254, 255), (253, 252), (254, 255), (254, 255), (254, 255), (254, 255), (254
, 255), (254, 255), (254, 255), (236, 237), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254,
 255), (254, 255), (236, 237), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254,
255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 2
55), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 25
5), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255
), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255)
, (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255),
 (254, 255), (143, 142), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (133, 132),
(254, 255), (133, 132), (190, 191), (144, 145), (254, 255), (208, 209), (134, 135), (144, 145), (254, 255), (
254, 255), (254, 255), (254, 255), (254, 255), (159, 158), (254, 255), (146, 147), (233, 232), (141, 140), (2
54, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (141, 140), (254, 255), (254, 255), (25
4, 255), (254, 255), (254, 255), (254, 255), (254, 255), (144, 145), (254, 255), (254, 255), (254, 255), (254
, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254,
 255), (254, 255), (254, 255), (174, 175), (253, 252), (254, 255), (254, 255), (254, 255), (254, 255), (254,
255), (254, 255), (254, 255), (254, 255), (254, 255), (253, 252), (131, 130), (197, 196), (254, 255), (254, 2
55), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255), (254, 255)]
```

There are two things to observe here:

1. All the numbers are very large, which means the brighter side is used to encode the flag (though we don't know the threshold).
2. These numbers differ by only 1, indicating LSB steganography once again.

Let’s rewrite our diff array creation taking account this observations

```
def get_str(t,fractal,fractal_with_flag):
   bin_str = ''
   for i in range(1024):
       for j in range(1024):
           if(fractal[i][j]>t and fractal_with_flag[i][j]>t):
               bin_str+=(chr(ord('0') + (fractal_with_flag[i][j] & 1)))
   return bin_str[:496] # since 496 is divisble by 8

for i in range(0,255):
   print("Threshold : ",i,end=" ")
   binary_flag = get_str(i,fractal,fractal_with_flag)
   flag = ''.join(chr(int(binary_flag[i:i+8], 2)) for i in range(0,
len(binary_flag), 8))
   print(flag)
```

What we did was add a check to ensure that the pixel value is greater than a threshold.

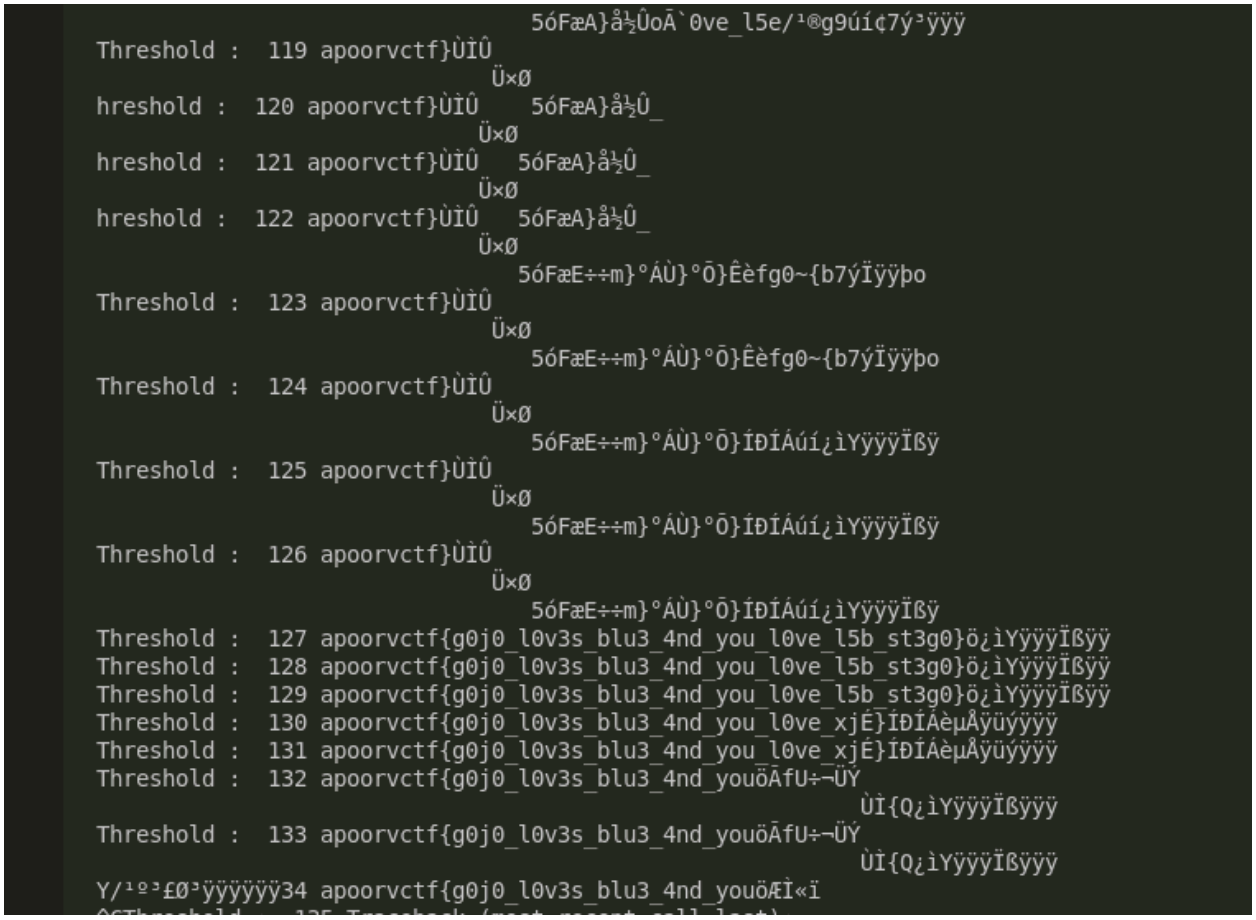
Then, we extracted the LSB from the given fractal image using:
`fractal_with_flag[i][j] & 1`

We also trimmed the extracted string to a length that is sufficiently divisible by 8, allowing it to contain the flag properly.

Finally, since the exact threshold is unknown, we can brute-force it by iterating from 0 to 255.

Output:

After running through various thresholds



We can see that threshold :127,128,129 gives our flag

Flag : apoorvctf{g0j0_l0v3s_blu3_4nd_you_l0ve_l5b_st3g0}