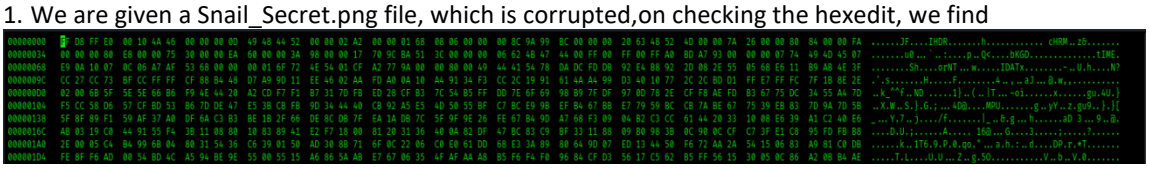
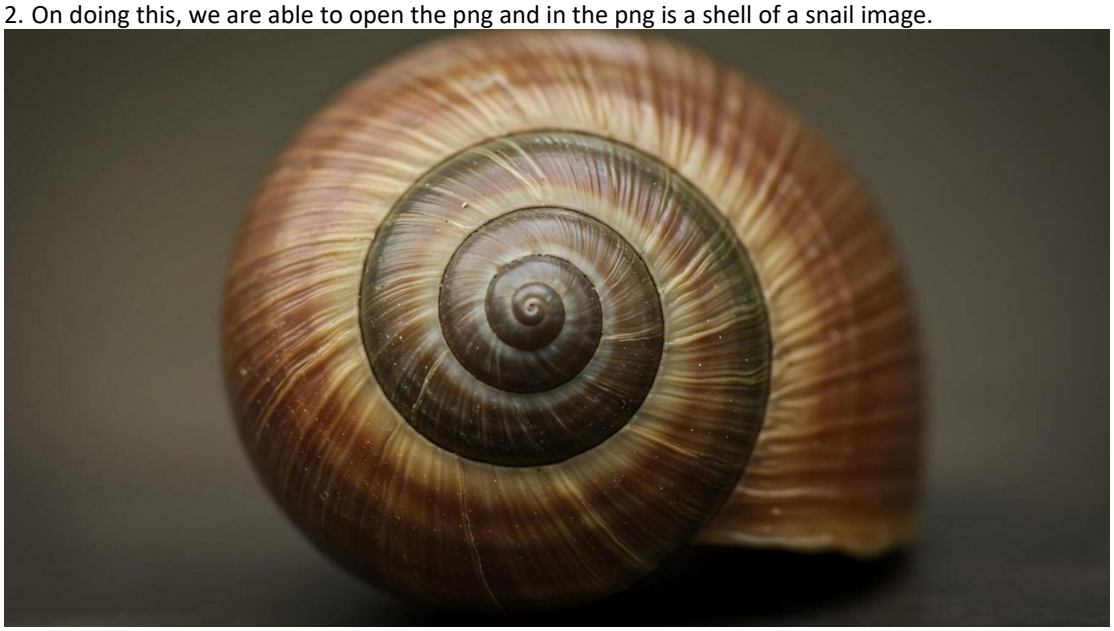
1. We are given a Snail_Secret.png file, which is corrupted,on checking the hexedit, we find



Which is the magic byte for jpeg file. We should change it to .png that is 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52.

2. On doing this, we are able to open the png and in the png is a shell of a snail image.



3. On increasing the height of the image,using this script,

```
#!/usr/bin/env python3
# set_png_height.py
# Usage: python3 set_png_height.py input.png output.png NEW_HEIGHT

import sys
import struct
import binascii
```

```
if len(sys.argv) != 4:
    print("Usage: python3 set_png_height.py input.png output.png NEW_HEIGHT")
    sys.exit(1)
```

```
infile, outfile, new_h_str = sys.argv[1], sys.argv[2], sys.argv[3]
```

```
try:
    new_h = int(new_h_str)
    if not (0 < new_h < 2**31):
        raise ValueError()
except ValueError:
    print("NEW_HEIGHT must be a positive integer (reasonable range).")
    sys.exit(1)
```

```
data = open(infile, "rb").read()
```

```
# Find IHDR chunk
pos = data.find(b'IHDR')
if pos == -1:
```

```
    print("IHDR chunk not found — not a valid PNG? Aborting.")
    sys.exit(1)

# IHDR data starts immediately after the 4-byte 'IHDR' type name
ihdr_data_start = pos + 4
ihdr_len = 13  # IHDR always has 13 bytes of data
ihdr_data = bytearray(data[ihdr_data_start:ihdr_data_start + ihdr_len])

# old width/height (big-endian)
old_width = struct.unpack(">I", ihdr_data[0:4])[0]
old_height = struct.unpack(">I", ihdr_data[4:8])[0]
print(f"Found IHDR: width={old_width}, height={old_height}")

# replace height bytes (bytes 4..7 of ihdr_data)
ihdr_data[4:8] = struct.pack(">I", new_h)
print(f"Setting new height = {new_h}")

# recompute CRC for the IHDR chunk: CRC is calculated over chunk type + chunk data
chunk_type_and_data = b'IHDR' + bytes(ihdr_data)
new_crc = binascii.crc32(chunk_type_and_data) & 0xffffffff
crc_bytes = struct.pack(">I", new_crc)

# Reassemble file:
before_ihdr = data[:ihdr_data_start]
after_ihdr_crc_offset = ihdr_data_start + ihdr_len + 4  # +4 for existing CRC
# keep everything after the old CRC
remaining = data[after_ihdr_crc_offset:]

new_file = before_ihdr + bytes(ihdr_data) + crc_bytes + remaining

with open(outfile, "wb") as f:
    f.write(new_file)

print(f"Written {outfile}. You should now inspect it (backup was not overwritten).")
```

we find some information:

4. To find more information, we do binwalk on the image and find a Troll.zip file which has a troll.png file.
5. On opening it, we find a clown_face image.
6. On some research, we find out that the snails shell shows Logarithmic spiral and the information provided to us is exactly the things we need to write a logarithmic spiral equation r=ae^(btheta)
7. So we try to find if there is anything embeded in the lsb of the troll.png. But we don't find anything
8. Doing the same for the 2nd lsb in the RGB, using this script,

```
from PIL import Image
import math

def bits_to_text(bits):
    chars = []
    for i in range(0, len(bits), 8):
        byte = 0
        for bit in bits[i:i+8]:
            byte = (byte << 1) | bit
        chars.append(chr(byte))
    return ''.join(chars)
```

```
def log_spiral_coords(width, height,
                      a=6.0, b=0.12,
                      theta_start=math.pi,
                      theta_end=8*math.pi,
                      theta_step=0.18):
    cx, cy = width // 2, height // 2
    coords, seen = [], set()
    theta = theta_start
    while theta < theta_end:
        r = a * math.exp(b * (theta - theta_start))
        x = int(round(cx + r * math.cos(theta)))
        y = int(round(cy + r * math.sin(theta)))
        if 0 <= x < width and 0 <= y < height and (x, y) not in seen:
            coords.append((x, y))
            seen.add((x, y))
        theta += theta_step
    return coords
```

```
def extract(stego_path):
    img = Image.open(stego_path).convert("RGBA")
    w, h = img.size
    pix = img.load()
    coords = log_spiral_coords(w, h)
```

```
    bits = []
    for (x, y) in coords:
        r, g, b, a = pix[x, y]
        for val in (r, g, b):
            bits.append((val >> 1) & 1)
    txt = bits_to_text(bits)
    print("🔓 Extracted message:")
    print(txt)
    return txt
```

```
if __name__ == "__main__":
    import sys
    if len(sys.argv) != 2:
        print("Usage: python decode_log_spiral.py <stego.png>")
        sys.exit(1)
    extract(sys.argv[1])
```

9. We find the something like REpTSVNBQ0F7bG9nX3NwMXI0bF9oM2xsfQ==..On decoding from base64, we get the flag.