WriteUp Dark

Challenge: You have the dark.exe binary. This program is not what it seems. It finds its type and contents, extracts the files it contains, finds the seed, and extracts the encrypted flag. The flag was encrypted with a keystream derived from SHA-256 (seed + counter).
The encrypted bits were scattered in the BMP LSBs using a pseudo-random permutation seeded with the same seed. "Search LSB" or "Look for blue-channel LSBs using a seeded PRNG"

```
sha256sum dark.exe
6dd8ceb2a6e887bb53afcef4963677935aaa3cc3a909fb64355a75edf25625a3  dark.exe
```

The file dark.exe is a jpg file and inside have 2 more files: another jpg file and a bmp file

```
file dark.exe
dark.exe: JPEG image data, JFIF standard 1.01, aspect ratio, density 1x1, segment length 16, baseline, precision 8, 2048x2048, components 3
```

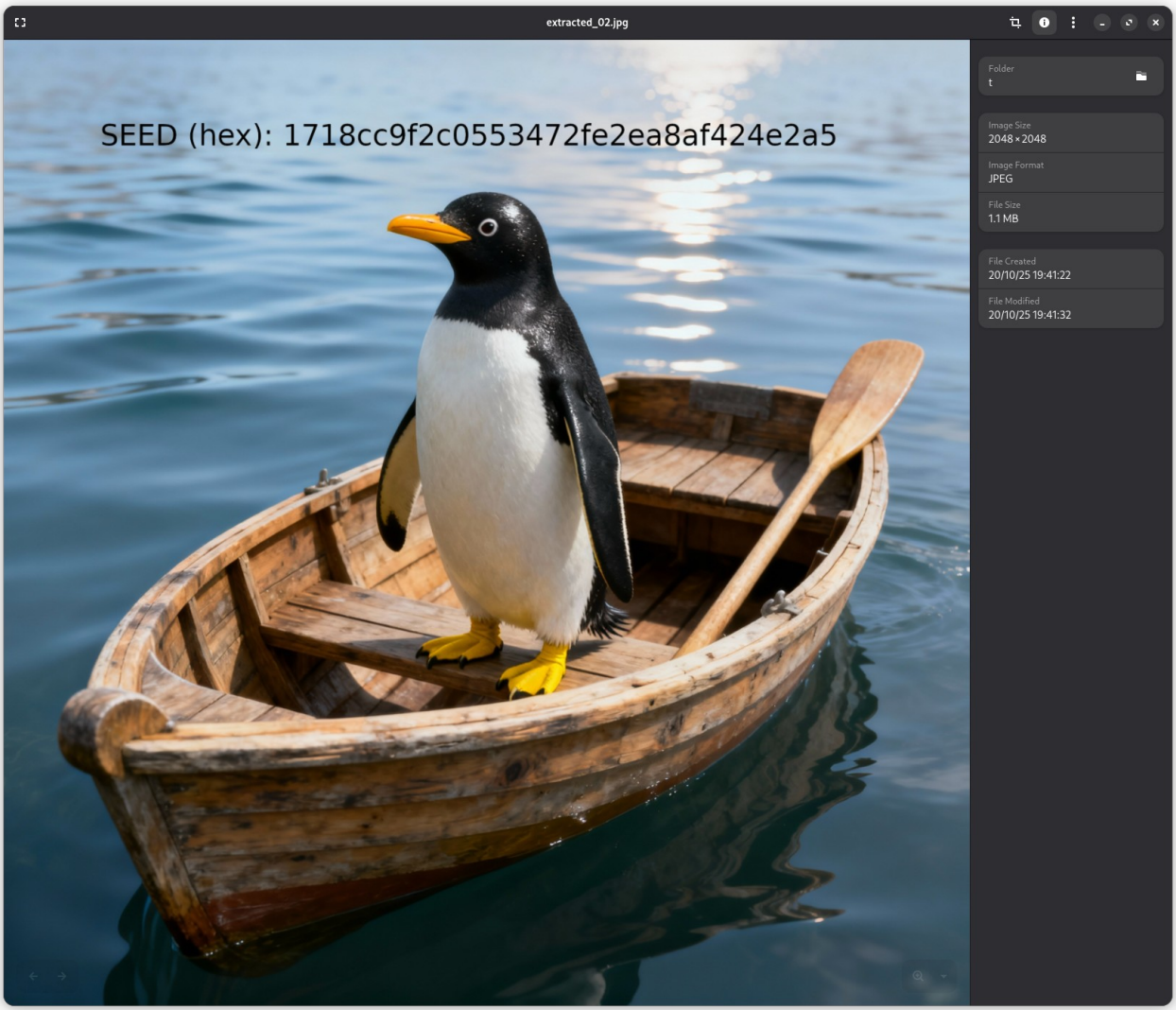
```
binwalk dark.jpg

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             JPEG image data, JFIF standard 1.01
1137096       0x1159C8        JPEG image data, JFIF standard 1.01
2243252       0x223AB4        PC bitmap, Windows 3.x format,, 800 x 400 x 24
```

Extract the files

```
dd if=dark.jpg of=extracted_01.jpg bs=1 count=1137096 status=progress
dd if=dark.jpg of=extracted_02.jpg bs=1 skip=1137096 count=1106156 status=progress
dd if=dark.jpg of=extracted_03.bmp bs=1 skip=2243252 status=progress
```

If open the file extracted_02.jpg show the sed: SEED (hex):

```
1718cc9f2c0553472fe2ea8af424e2a5
```



The flag was encrypted with a keystream derived from SHA-256 (seed + counter). The encrypted bits were scattered in the BMP LSBs using a pseudo-random permutation seeded with the same seed. "Search LSB" or "Look for blue-channel LSBs using a seeded PRNG"

Using a program in python could extract the flag from extracted_03.bmp file:
python3 extractor.py 1718cc9f2c0553472fe2ea8af424e2a5 extracted_03.bmp
magic: b'STG1' length: 40
Recovered flag: flagmx{Steganografia_TecnicaAntiforense}

Ths extractor.py file

```
#!/usr/bin/env python3
# extractor.py
from PIL import Image
import struct, hashlib, random
import sys
```

```
def keystream_bytes(seed_hex, length):
    out = bytearray()
    counter = 0
    while len(out) < length:
        h = hashlib.sha256()
        h.update(seed_hex.encode('utf-8'))
        h.update(struct.pack(">I", counter))
        out.extend(h.digest())
        counter += 1
    return bytes(out[:length])

if len(sys.argv) != 3:
    print("Usage: python3 extractor.py <seed_hex> <bmp_path>")
    sys.exit(1)

seed = sys.argv[1]
bmp_path = sys.argv[2]
img = Image.open(bmp_path)
w, h = img.size
total_pixels = w*h
pixels = img.load()

# Recreate permutation
rng = random.Random(seed)
positions = list(range(total_pixels))
rng.shuffle(positions)

# Primero extraemos 8 bytes (header=8 bytes => 64 bits) para descifrar magic+length
bits = []
for i in range(64):
    idx = positions[i]
    x = idx % w; y = idx // w
    r,g,b = pixels[x,y]
    bits.append(b & 1)

header_bytes = bytearray()
for i in range(0,64,8):
    byte = 0
    for j in range(8):
        byte = (byte << 1) | bits[i+j]
    header_bytes.append(byte)

ks_header = keystream_bytes(seed, 8)
dec_header = bytes([a ^ b for a,b in zip(header_bytes, ks_header)])
magic = dec_header[:4]
length = struct.unpack(">I", dec_header[4:8])[0]
print("magic:", magic, "length:", length)

# Ahora extraemos total_bits = (8 + length) * 8
total_bytes = 8 + length
total_bits = total_bytes * 8
bits = []
for i in range(total_bits):
    idx = positions[i]
    x = idx % w; y = idx // w
    r,g,b = pixels[x,y]
    bits.append(b & 1)

payload_enc = bytearray()
```

```
for i in range(0,total_bits,8):
    byte = 0
    for j in range(8):
        byte = (byte << 1) | bits[i+j]
    payload_enc.append(byte)

ks_full = keystream_bytes(seed, total_bytes)
payload = bytes([a ^ b for a,b in zip(payload_enc, ks_full)])
magic2 = payload[:4]
length2 = struct.unpack(">I", payload[4:8])[0]
flag = payload[8:8+length2]
print("Recovered flag:", flag.decode('utf-8'))
```