Forensics CTF 9

Platform: picoCTF 2019

Challenge Name: Investigative Reversing 3

Category: Forensics

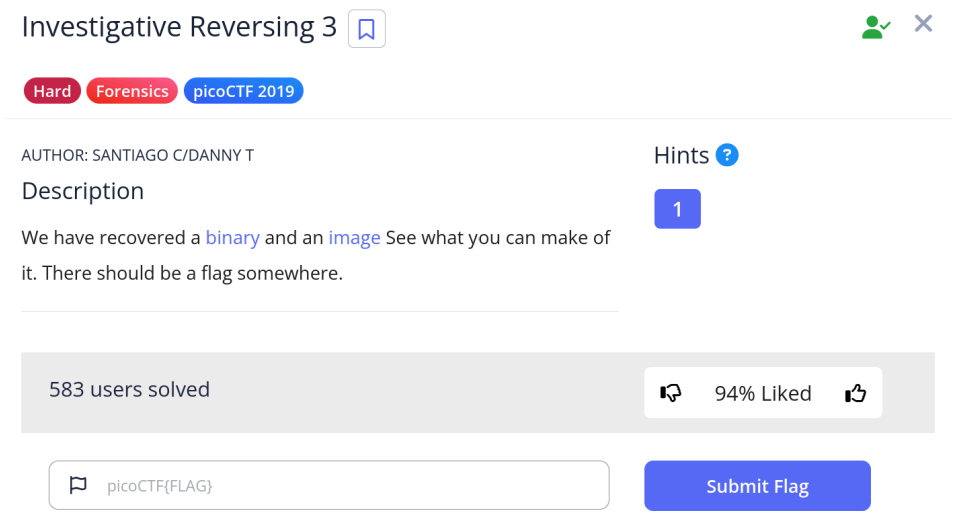
Difficulty: Hard

Submitted By: Gurleen Kaur Brar

Objective

The objective was to reverse engineer a custom binary and extract hidden data from an image file. The challenge involved disassembling the binary with Ghidra and implementing a decoding algorithm to recover the flag embedded using a Least Significant Bit (LSB) encoding method.

Challenge Description



Files and Tools Used

- **Files Provided:**

    - `mystery` (compiled ELF binary)
    - `encoded.bmp` (bitmap image file)
- **Tools Used:**
    - Ghidra (for binary decompilation)
    - Kali Linux Terminal
    - Python (for decoding script)

Step-by-Step Process

Step 1: Reverse Engineer the Binary with Ghidra

Loaded `mystery` into Ghidra and decompiled the `main()` function. The analysis revealed the following:

- File starts decoding at offset `0x2d3`
- Every **8 bytes' LSB** was used to encode one bit of the hidden message
- Every 9th byte was original data passed through

Decompile: main - (mystery)

```
undefined8 main(void)

{
  size_t sVar1;
  long in_FS_OFFSET;
  char local_7e;
  char local_7d;
  int local_7c;
  int local_78;
  uint local_74;
  int local_70;
  undefined4 local_6c;
  int local_68;
  int local_64;
  FILE *local_60;
  FILE *local_58;
  FILE *local_50;
  char local_48 [56];
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  local_6c = 0;
  local_60 = fopen("flag.txt","r");
  local_58 = fopen("original.bmp","r");
  local_50 = fopen("encoded.bmp","a");
  if (local_60 == (FILE *)0x0) {
    puts("No flag found, please make sure this is run on the server");
  }
  if (local_58 == (FILE *)0x0) {
    puts("No output found, please run this on the server");
  }
  sVar1 = fread(&local_7e,1,1,local_58);
  local_7c = (int)sVar1;
  local_68 = 0x2d3;
  for (local_78 = 0; local_78 < local_68; local_78 = local_78 + 1) {
    fputc((int)local_7e,local_50);
    sVar1 = fread(&local_7e,1,1,local_58);
    local_7c = (int)sVar1;
  }
  sVar1 = fread(local_48,0x32,1,local_60);
  local_64 = (int)sVar1;
  if (local_64 < 1) {
    puts("Invalid Flag");
                    /* WARNING: Subroutine does not return */
    exit(0);
  }
  for (local_74 = 0; (int)local_74 < 100; local_74 = local_74 + 1) {
    if ((local_74 & 1) == 0) {
      for (local_70 = 0; local_70 < 8; local_70 = local_70 + 1) {
        local_7d = codedChar(local_70,(int)local_48[(int)local_74 / 2],(int)local_7e);
        fputc((int)local_7d,local_50);
        fread(&local_7e,1,1,local_58);
      }
    }
    else {
      fputc((int)local_7e,local_50);
      fread(&local_7e,1,1,local_58);
    }
  }
  while (local_7c == 1) {
    fputc((int)local_7e,local_50);
    sVar1 = fread(&local_7e,1,1,local_58);
```

Step 2: Inspect Encoded Data in the BMP

Using `xxd`, viewed the relevant portion of the bitmap file:

```
xxd -g 1 -s $((0x2d3 - 32)) -l $((50*8 + 48 + 64)) encoded.bmp
```

This confirmed that data was encoded densely in the LSBs.

```
┌──(venv)─(gurleen㉿kali)-[~/ctf]
└─$ xxd -g 1 -s $((0×2d3 - 32)) -l $((50*8 + 48 + 64)) encoded.bmp
000002b3: 60 e0 00 80 80 00 00 80 80 20 00 80 80 40 00 80  `.. .. ..  . ..@..
000002c3: 80 60 00 80 80 80 00 80 80 a0 00 80 80 c0 00 80  .`. ... . ... . ...
000002d3: 80 e0 00 80 a1 01 01 80 a0 21 00 80 a1 40 01 81  .. ........!. ..@..
000002e3: a0 60 01 81 a0 80 00 81 a1 a0 00 81 a1 c1 01 80  .`..... ... .......
000002f3: a1 e1 00 80 c1 01 00 80 c0 20 01 80 c0 40 00 81  .. ... . ..  ...@..
00000303: c0 61 00 81 c0 80 00 81 c1 a0 00 80 c1 c0 00 81  .a. ... . ... . ...
00000313: c1 e0 01 81 e1 01 00 80 e0 20 01 80 e1 41 00 80  ........ ..  ...A..
00000323: e0 60 01 81 e1 80 01 81 e0 a0 00 80 e0 c0 01 81  .`..............
00000333: e0 e0 00 c0 00 01 00 c1 01 21 00 c0 00 40 00 c1  ..... ..  !..@..
00000343: 00 61 01 c0 00 81 01 c0 00 a1 01 c0 00 c0 00 c1  .a .. . ... . ...
00000353: 00 e0 01 c1 21 00 00 c1 21 21 01 c1 20 41 00 c0  . ...!.. .!.. A..
00000363: 20 60 01 c1 20 80 01 c0 20 a1 00 c1 20 c1 01 c0   `.. ... .. ... ...
00000373: 20 e0 00 c1 40 00 00 c1 41 20 00 c1 41 41 01 c1   ...@.. A ..AA..
00000383: 40 61 00 c0 40 80 00 c0 41 a1 01 c0 40 c0 01 c0  @a. @. . A ..@ ...
00000393: 40 e1 01 c1 60 00 00 c0 60 20 01 c1 60 40 00 c0  @ ... `.. `  ..`@..
000003a3: 61 60 00 c0 61 81 00 c0 60 a0 01 c1 60 c1 01 c0  a`. .a.. `  ... `...
000003b3: 60 e1 01 c0 80 01 01 c0 80 20 01 c0 81 41 00 c1  `........ ... A..
000003c3: 81 60 00 c1 81 81 01 c1 80 a1 00 c0 80 c0 00 c0  .`.........  ... ..
000003d3: 81 e1 00 c0 a0 00 00 c0 a0 21 01 c0 a0 40 00 c0  .. . .. .. ! ...@..
000003e3: a0 60 01 c1 a0 80 00 c0 a0 a0 00 c1 a1 c0 00 c0  .`..... ... . ... ..
000003f3: a0 e0 00 c0 c1 01 00 c0 c0 20 00 c0 c0 41 01 c0  .. ... . ..  . ..A..
00000403: c0 60 00 c0 c0 80 01 c1 c0 a0 00 f0 fa fe 00 a5  .`.........  ... ..
00000413: a1 a0 00 80 80 80 00 00 01 ff 00 00 ff 00 00 00  .. . ... .. .. . ...
00000423: fe ff 01 fe 00 00 00 fe 00 fe 01 ff fe 00 00 fe  . .. ... . .. . .. .
00000433: fe fe 00 e9 e9 e8 e8 e8 e8 e8 e8 e9 e9 e8 e8 e8  ..  ..............
00000443: e8 e9 e8 e8 e8 e9 e9 e8 e8 e8 e8 e9 e9 e8 e8 e9  ................
00000453: e9 e8 e8 e8 e8 e8 e9 e9 e9 e8 e8 e8 e9 e8 e8 e8  ................
00000463: e8 e9 e9 e8 e8 e9 e8 e9 e8 e8 e8 e9 e9 e8 e8 e9  ................
00000473: e8 e8 e9 e9 e8 e8 e8 e8 e8 e8 e9 e9 e9 e8 e8 e8  ................
00000483: e8 e8 e9 e8 e9 e9 e8 e8 e8 e9 e8 e9 e9 e9 e9 e9  ................
00000493: e8 e8 e8 e8 e8 e8 e8 e8 e8 e8 e8 e8 e8 e8 e8 e8  ................
000004a3: e8 e8 e8 e8 e8 e8 e8 e8 e8 e8 e8 e8 e8 e8 e8 e8  ................
```

Step 3: Decode with a Python Script

Wrote a script using to reconstruct the hidden message:

```
  GNU nano 8.3
from pwn import *

with open("encoded.bmp", "rb") as b:
    b.seek(0×2d3)
    bin_str = ""
    for j in range(100):
        if ((j & 1) == 0):
            for k in range(8):
                bin_str += str(ord(b.read(1)) & 1)
        else:
            b.read(1)

char_str = unbits(bin_str, endian='little')
print(char_str)
```

Step 4: Run the Decoder and Extract Flag

Executed the script and successfully retrieved the flag from the output.

```
┌──(venv)─(gurleen㉿kali)-[~/ctf]
└─$ python3 solve.py
b'picoCTF{4n0th3r_L5b_pr0bl3m_00000000000001f8ae184}'
```

Flag Submitted

picoCTF{4n0th3r_L5b_pr0bl3m_00000000000001f8ae184}

The flag was decoded and submitted successfully.