Mark Seliternikov

picoCTF - Investigative Reversing 3 [400 points]

Challenge details:

Investigative Reversing 3 | 400 points

Tags: Category: Forensics

AUTHOR: SANTIAGO C/DANNY T

Description

We have recovered a binary and an image See what you can make of it. There should be a flag somewhere.

Hints

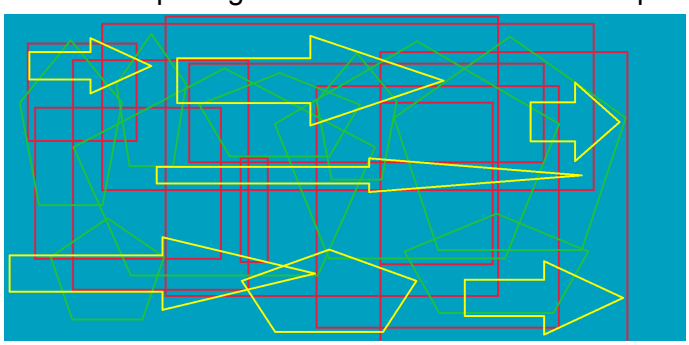
1

You will want to reverse how the LSB encoding works on this problem

So similarly to the previous challenge (Investigative Reversing 2), we get 2 files.
The file “mystery” which is an ELF file again:

```
└$ file mystery
mystery: ELF 64-bit LSB pie executable,
```

And a bitmap image which is called “encoded.bmp”:



So I’m pretty sure the idea is the same so I’m going to start by reversing “mystery” via ghidra (Reversing program made by the NSA).
After reversing this is what I found:

The first thing the program does is opening all necessary files, Which are “flag.txt”, “original.bmp” (original image) and “encoded.bmp” (encoded image which is about to be written to). Note that all the variables/parameters names have been altered in order to simplify understanding.

```
local_8c = 0;
flag_ptr = fopen("flag.txt","r");
orig_ptr = fopen("original.bmp","r");
enc_ptr = fopen("encoded.bmp","a");
if (flag_ptr == (FILE *)0x0) {
```

Once that is done it proceeds to write the first 0x2d3 (723) bytes to the encoded image.

```
local_7c = (int)sVar1;
local_68 = 0x2d3;
local_78 = 0;
while (local_78 < local_68) {
  fputc((int)(char)buf,enc_ptr);
  sVar1 = fread(&buf,1,1,orig_ptr);
  local_7c = (int)sVar1;
  local_78 = local_78 + 1;
}
```

Another important step that happens is reading the “flag.txt” contents into a buffer. It’ll be important later, also note that it must be 50 chars (0x32 = 50).

```
sVar1 = fread(flag_buf,0x32,1,flag_ptr);
local_64 = (int)sVar1;
if (local_64 < 1) {
  puts("Invalid Flag");
                    /* WARNING: Subroutine does
```

Then we get to the real deal, the part where the encoding happens.

```
i = 0;
while ((int)i < 100) {
  if ((i & 1) == 0) {
    j = 0;
    while (j < 8) {
      coded = codedChar(j,flag_buf[(int)i / 2],buf);
      fputc((int)(char)coded,enc_ptr);
      fread(&buf,1,1,orig_ptr);
      j = j + 1;
    }
  }
  else {
    fputc((int)(char)buf,enc_ptr);
    fread(&buf,1,1,orig_ptr);
  }
  i = i + 1;
}
```

As you can see it has 2 loops, the first loops 100 times and the second loops 8 times! (8 bits *wink*)
But there’s a twist! Note that the if condition is executed only when the ‘i’ value is even.

```
i = 0;
while ((int)i < 100) {
  if ((i & 1) == 0) {
    j = 0;
    while (j < 8) {
```

Otherwise it executes the ‘else’ statement, which is just reading the bytes from the original image normally.

```
else {
  fputc((int)(char)buf,enc_ptr);
  fread(&buf,1,1,orig_ptr);
}
```

What happens in the inner loop is encoding of 8 bytes. Using 1 byte from the flag and 8 bytes from the original image.

```
j = 0;
while (j < 8) {
  coded = codedChar(j,flag_buf[(int)i / 2],buf);
  fputc((int)(char)coded,enc_ptr);
  fread(&buf,1,1,orig_ptr);
  j = j + 1;
}
```

So basically what happens is 8 encoded bytes followed by one unencoded byte and the cycle repeats til the ‘i’ condition is met. (which is til 100)
Visual representation:
[encoded][encoded][encoded][encoded][encoded][encoded][encoded][encoded][UNENCODED][encoded][encoded][encoded][encoded][encoded][encoded][encoded][encoded][UNENCODED]

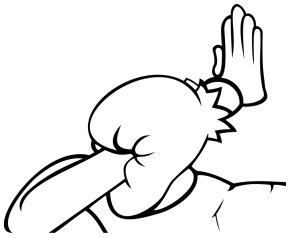
Now that actual encoding:

```
byte codedChar(int j,byte flagbuf,byte buf)

{
  byte local_20;

  local_20 = flagbuf;
  if (j != 0) {
    local_20 = (byte)((int)(char)flagbuf >> ((byte)j & 0x1f));
  }
  return buf & 0xfe | local_20 & 1;
}
```

They used the same encoding as in the last challenge!



Just kidding, it’s okay, I guess it wasn’t the main focus in this challenge.
In short:

The bits of the flag byte are shifted right according to the ‘j’ loop. (looping over each bit)

```
local_20 = (byte)((int)(char)flagbuf >> ((byte)j & 0x1f));
```

Then it creates a new byte, taking an original byte and making sure only the LSB is changed (the smallest bit)

```
return buf & 0xfe | local_20 & 1;
```

So in order to solve it all I need to do is construct the original bytes in reverse using their method of encoding :)
Here is the full python script:

```python
with open('encoded.bmp', 'rb') as f:

    for i in range(0, 0x2d3):
        cur = f.read(1)

    for i in range(0, 100):

        unencoded = 0

        if i % 2 == 0:

            for j in range(0, 8):

                cur = int.from_bytes(f.read(1), 'little')

                tmp = cur & 1

                tmp = tmp << j

                unencoded += tmp

            # print the constructed byte
            print(chr(unencoded), end='')

        else:

            cur = f.read(1)

print()
```

What basically happens in short is that I skip the first 0x2d3 bytes. Then every time the ‘i’ value is odd I skip it also. Otherwise, I loop for 8 times and collect the LSBs and reconstruct the original byte.
Very much like the previous challenge with a few tweaks. :)

And……

```
┌──(kali㉿kali)-[~/Desktop]
└─$ python3 test.py
picoCTF{4n0th3r_L5b_pr0bl3m_00000000000001f8ae184}
```

Yep that’s all folks!