Challenge #8 Solution

by Dimiter Andonov

The eighth challenge is about steganography. Before delving into the details, let's refresh our understanding about what steganography is and how can be used to conceal data within other data.

Steganography 101

Steganography is the art or practice of concealing a file, message, image, or video within another file, message, image, or video. The data being concealed is called a steganogram.

One of the popular media frequently used as a steganogram carrier is 24-bit bitmap image compressed as PNG. The reason for choosing the latter is that the PNG provides a loss-less compression and decompression of the underlying bitmap image. This means that the steganogram stored in a PNG is smaller than the original data, but can be reconstructed perfectly. For the same reasons, the JPEG format is not a good choice for a carrier media because of its lossy compression algorithm.

In a 24-bit color bitmap model, each pixel is represented by three color components – red (R), green (G), and blue (B) and each of them is one byte long. Instilling the steganogram into the RGB color components can be achieved by endless possible ways. In one example, a specific bit position can be used depending on the color, pixel coordinates, or some more complex algorithm. To minimize the original media color distortion, the instill process could be set to skip color components that would change their color in a more drastic way should the desired bit is used to carry the steganogram.

The steganogram instill process can be implemented by creating a bit stream around the data being concealed and use the bits to manipulate the carrying media's RGB color components. If the bits are chosen carefully the resulting steganogrammed image would not be visually different from the original image.

Overall View of the Steganography Challenge

For this challenge we wanted to keep the steganogram instillation process as simple as possible so we have chosen a basic algorithm where each color component in every pixel is considered as a target and the bit position in the carrying media is always zero. The steganogram itself is a small PE32 executable that when extracted and run, displays the email address that needs to be used to get the next challenge. The whole process can be divided

into two major steps - finding and decoding the steganogrammed PNG image and extracting the steganogram from the carrying media.

Step 1: Finding, decoding, and analyzing the PNG image

The challenge is implemented as a PE32 executable that when run displays the string "`the one who seeks finds...`" and then exits. When disassembled it is easy to spot the large Base64 encoded data between the `printf` API call displaying the string and the process termination with the `ExitProcess` Win32 API.

```
start     proc near
push offset Format  ; "the one who seeks finds…\n"
call ds:printf
add  esp,4
jmp  loc_4ccda5
start endp

;encoded Base64 data

loc_4ccda5:
          push 0
          call $+5
          jmp  ds:ExitProcess
```

Figure 1: Outer Layer Disassembly

The encoded data seems promising, so we extract it from the executable, decode it using the standard Base64 algorithm, and save it to the disk. Opening the file in a hex editor shows the familiar PNG signature at offset 0: `0x89, 0x50, 0x4e, 0x47`.

```
0000h: 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52  ‰PNG........IHDR
0010h: 00 00 02 58 00 00 01 E0 08 02 00 00 00 F6 75 84  ...X...à.....öu„
0020h: 0E 00 01 00 00 49 44 41 54 78 9C 84 FD 6F 90 24  .....IDATxœ„ýo
```

Figure 2: PNG Header

Looking into the structure of the PNG image does not show any anomalies and analyzing the PNG chunks does not provide us with any clues about what to do next. At this point we should consider that the image might have a steganogram, hopefully instilled by a basic algorithm.

Step 2: Extracting the steganogram

The simplest way to try first would be to represent the PNG image as a 24-bit color bitmap, enumerate all the pixels horizontally, and extract the bits at position zero of the RGB color

components. Because the least significant bit of a color generates such a small change to the actual color, it is easy to hide data in this position, one bit at a time. We can achieve that with the help of a small Python script using the PIL library. This script will iterate over each pixel's RGB components collecting the desired bits. Since we deal with one bit per color component we will be aggregating three bits per pixel.

```
from PIL import Image
from enum import Enum

# RGB color components
class ColorComponent(Enum):
    Red = 0
    Green = 1
    Blue = 2
    Guard = 3

def is_nth_bit_set(pixels, x, y, color_comp, pos):
    color_val = pixels[x, y][color_comp]
    return (color_val & (1 << pos)) != 0

def set_bit_in_buffer(steganogram, bit_counter, is_bit_set):
    byte = 0
    bit_index = bit_counter % 8
    if bit_index == 0:
        steganogram.append(byte)
    else:
        byte = steganogram[len(steganogram) - 1]
    mask = 1 << bit_index
    byte &= ~mask
    if is_bit_set:
        byte |= mask
    steganogram[len(steganogram) - 1] = byte
    return

def extract_steganogram(pixels, width, height):
    steganogram = []
    bit_counter = 0
    for y in range(0, height):
        for x in range(0, width):
            red = pixels[x,y][0]
            green = pixels[x,y][1]
            blue = pixels[x,y][2]
```

```
            for color_comp in range(ColorComponent.Red,
ColorComponent.Guard):
                is_bit_set = is_nth_bit_set(pixels, x, y, color_comp,
0)
                set_bit_in_buffer(steganogram, bit_counter,
is_bit_set)
                bit_counter += 1
    return steganogram
```

Figure 3: Extraction Script

After we run the Python script passing the pixel array to the `extract_steganogram` function, we should end up with an extracted steganogram that has the comforting MZ signature at the beginning of the file. The embedded steganogram size is not known during the extraction process so we continue until we reach the last pixel of the carrying media. In our case the embedded PE32 executable will have an overlay that represent the random data collected after the end of the steganogram. The overlay however can easily be cut away using a PE32 aware hex editor.

Running the extracted steganogram displays the email address needed for the next challenge:

```
Im_in_ur_p1cs@flare-on.com
```

Overall, the steganography challenge is one of the easier ones this year. The trick, however, is to figure out what to look for once the preliminary PNG analysis does not yield anything useful. At the same time we wanted to keep the instilling algorithm as simple as possible because our primary goal was to show how easy and efficient is to hide data inside other data using a steganography technique.