ESTEGANOGRAFÍA

FWHIBBIT CTF TEAM

Stego Cipher

Points: 230
Country: Germany
Attachment: https://mega.nz/#!R8d1ULqK!8MKhlD1Cde9CvsFWnGOVVrJ1WE6K66V4KM4daJ1NnWU
Description: Our spies found this image. They think something is ciphered in it... what could it be?
Also, they could steal a suspicious script, we are not sure but maybe it was used for ciphering some message.
The future of our team depends on you!

Link: https://mega.nz/#!R8d1ULqK!8MKhlD1Cde9CvsFWnGOVVrJ1WE6K66V4KM4daJ1NnWU

Tenemos dos ficheros: *encrypted.png* y *stego_cipher.py*. El primero es una imagen y lo segundo es un pequeño script en python.

(Imagen proporcionada)

```python
from PIL import Image
import random

FLAG = '^__^'

img = Image.open('original.png')
img_pix = img.convert('RGB')

x = random.randint(1,255)
y = random.randint(1,255)

img_pix.putpixel((0,0),(len(FLAG),x,y))

for l in FLAG:
    x1 = random.randint(1,255)
    y1 = random.randint(1,255)
    img_pix.putpixel((x,y),(ord(l),x1,y1))
    x = x1
    y = y1

img_pix.save('encrypted.png')
```

(Script de python proporcionado)

Empezamos por la imagen. Buscando la imagen original en internet podemos ver notables diferencias.

(Imagen original)

Después de esto inspeccionamos el script que tenemos. Primero abre la imagen *original.png* y lo convierte a RGB (líneas 6,7). Luego genera dos números random, *x y*, que pueden tener un valor entre 1 y 255 (líneas 9,10). Ahora viene lo importante, la función *Image.putpixel*, buscando en internet obtenemos:

> "***im.putpixel(xy, colour)***
> *Modifies the pixel at the given position. The colour is given as a single numerical value for single-band images, and a tuple for multi-band images.*"

Fuente:http://effbot.org/imagingbook/image.htm

La línea de código que estamos intentando entender es *img_pix.putpixel((0,0),(len(FLAG),x,y))* (línea 12), entonces lo que hace es poner en el pixel 0,0 en cada canal un número, el primero siendo la longitud del flag que queremos sacar, y luego los dos números "aleatorios" generados anteriormente, *x y*.

Después viene un bucle for (que tiene tantas iteraciones como la longitud del flag, línea 14) que genera otros dos números random, *x1 y1*, desde 1 hastas 255 (líneas 15,16) y pone/escribe estos dos números en el pixel *x y* anteriores (línea 17). Ahora escribe en la imagen en el pixel x y tres cosas: el ordinal del carácter del flag y los valores de x1 y1. A continuación actualiza los valores *x*,*y* a *x1*,*y1* respectivamente (líneas 18,19).

Finalmente guarda la imagen como *encrypted.png* (línea 21).

Una vez entendido el código, vamos a escribir un script para sacar la flag de la imagen haciendo las operaciones “en el orden inverso”.

Primero importamos lo necesario, abrimos la imagen *encrypted.png* y la convertimos a RGB, igual que en el script original. En el mismo recurso donde encontramos la definición de la función de *putpixel* vemos que existe una función *getpixel*:

> “**im.getpixel(xy)** ⇒ value or tuple
> Returns the pixel at the given position. If the image is a multi-layer image, this method returns a tuple.”

Fuente:http://effbot.org/imagingbook/image.htm

Con esta función podemos obtener los valores que hay en un determinado pixel. En el script original lo primero que se hace es escribir en el pixel 0,0 información útil. Lo leemos con *img_pix.getpixel((0,0)),* donde el primer elemento es la longitud de la flag, el segundo elemento es el valor de x y el tercer elemento es el valor de y.

Con esta información podemos empezar nuestro bucle for porque sabemos el número de iteraciones (la longitud del flag). Dentro del bucle leemos la información contenida en el pixel x y (valores que hemos obtenido anteriormente). El primer elemento que obtenemos es el ordinal del carácter de la flag, por lo que lo convertimos a chr; y los valores de x y siguientes (la actualización de los índices del siguiente pixel que vamos a leer). El carácter que vamos leyendo lo vamos guardando y el resultado será la flag. A continuación el código resultante:

```
1.  from PIL import Image
2.
3.  img = Image.open('encrypted.png')
4.  img_pix = img.convert('RGB')
5.
6.  primtuple = img_pix.getpixel((0,0))
7.
8.  lengflag = primtuple[0]
9.  x = primtuple[1]
10. y = primtuple[2]
11.
12. almc = ''
13.
14. for l in xrange(lengflag):
15.     seguntuple = img_pix.getpixel((x,y))
16.
17.     almc += chr(seguntuple[0])
18.
19.     x = seguntuple[1]
20.     y = seguntuple[2]
21.
22. print almc
```

El resultado obtenido gracias al script es:
fwhibbit{st3g4n0gr4phy_1n_1m4g3_p1x3ls}

-SoA aka ruso