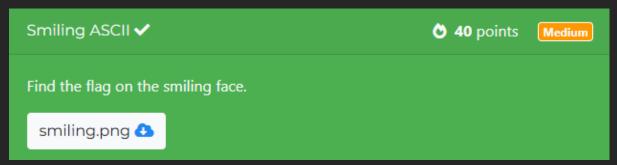
Smiling ASCII CTF Writeup

This document is a walkthrough on one way to solve the **Smiling ASCII CTF** on **CTFLearn**. The objective is to explain how I was able to solve this CTF to my future self.

General Information

- Difficulty: Medium
- Category: Forensics
- Link: Challenge Smiling ASCII CTFlearn CTF Practice

Introduction



We're given a PNG file, which looks like this:

```
100003833383386000
             >>>>$$$$$$$$$$$$$$$$$$$$$$$$$$$$
           O$ ($3 O$)
 e # eo
         . ,20022004932020202020202020202020202020
                                    $$ 33 3303
oc 1 1 1
                8888688888
                          383833333
3838833383
                                     33500303
        ~33333388B
33333365
        333333333
                 3033333333
                                    32333333
       300000000000
                 00339333003
                           333993333333333333333333
 3353553
 28123033381230333812301
                           30333833303332
                                    2303388833031
 .003368333033683330336833303368333033683330336833303
 633°
                                       3336
     333
2003388303033388303
                                -5333"
           5000
                                333
                21 .... 338331 ......
        3860
                              ~383"·
        $$$$$0
                     10.00
              33030
                             @0338
          2000000000000
                     2383333333
                      3$3"...
```

First and foremost, let's do a simple strings command on the file to see if there's anything hidden:

```
(alexandre vbox)-[~/Downloads]
$ strings smiling.png | tail
eddd:
eddd:
/###
_FFF
2222
eddd:
/###
,/>I
IEND
```

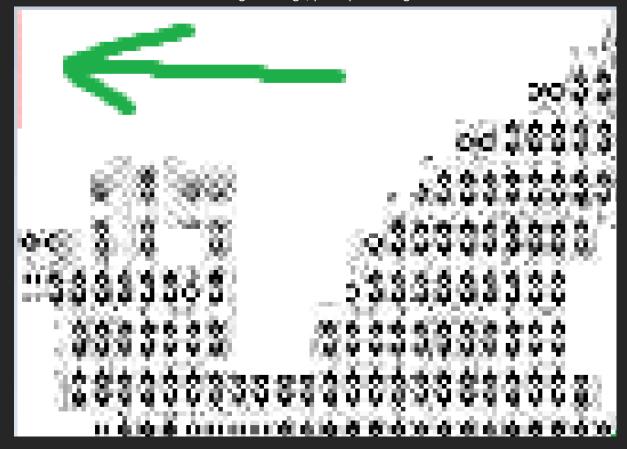
RGlkIHlvdSBrbm93IHRoYXQgcGl4ZWxzIGFyZSwgbGlrZSB0aGUgYXNjaWkgdGFibGUsIG51bWJlcmVkIGZyb20gMCB0byAyNTU/Cg=

It looks like base64-encoded string was added at the end of the file

```
—(alexandre⊕ vbox)-[~/Downloads]
$\$\$\ \ext{echo} \text{RGlkIHlvdSBrbm93IHRoYXQgcGl4ZWxzIGFyZSwgbGlrZSB0aGUgYXNjaWkgdGFibGUsIG51bWJlcmVkIGZyb20gMCB0byAyNTU/Cg= | base64 --decode
Did you know that pixels are, like the ascii table, numbered from 0 to 255?
```

It looks like we're going to have to examine the pixels on the image.

Effectively, upon closer look, red pixels can be seen at the top left of the image, which suggests that information has been added to the original image, perhaps the flag.



Using the PIL (for pillow) library in Python, we're able to convert the image into an array of numbers, each indicating the respective RGBA value of each pixel. The output confirms that the Green, Blue, and Alpha values of the first few pixels in the first row of the image are different to the others, perhaps their values are hiding a message?

```
File Edit Selection Find View Goto Tools Project Preferences Help
♦ text.py
                program2.py
    from PIL import Image
    import numpy as np
 4 img = Image.open('smiling.png')
 5 data = np.array(img)
 6 print(data)
[[255]
         67
             67
                 671
  [255 255 255 255]
  [255 255 255 255]
  [255 255 255 255]
  [255 255 255 255]
  [255 255 255 255]]
         84
 [[255
             84
                  841
  [255 255 255 255]
  [255 255 255 255]
  [255 255 255 255]
  [255 255 255 255]
  [255 255 255 255]]
 [[255
        70
             70
                  701
  [255 255 255 255]
  [255 255 255 255]
```

Let's extract the blue value of each pixel in the row of the image (the same can be done for green and alpha values), we get the following array:

```
~/Dow
File Edit Selection Find View Goto Tools Project Preferences Help
       × program2.py
  from PIL import Image
  import numpy as np
  img = Image.open('smiling.png')
  data = np.array(img)
  modified pixels = data[:, 0, 2]
8 print(modified_pixels)
[ 67 84
    70 108 101 97 114 110 123 97 115 99 105 105
                          95 112 105 120
101 108
    95 102 108 97 103 125
                0 255 255 255 255 255 255
                              255 255
228 237
255 255 255 255
255
                          255
                            255
                              255
                                255
255 255 255 255 255 255 255 255 255
                     255
                   255
                       255
                         255
                          255 255
                              255
255 255
255 255 255 255
255 255
255 255 255 255 255
         255
           255 255 255 255
                   255 255
                       255 255
                          255
                            255
                              255
255 255 255 255 255 255 255 255 255
                   255 255
                       255 255 255 255 255 255
[Finished in 574ms]
```

And there you have it, by converting the first 26 values of the array into ASCII characters, we find the following flag: CTFlearn{ascii_pixel_flag}.

Note: We could've used a tool called **zsteg** to solve this automatically