**Grand Blue Challenge Solution:  
Faith See (1002851)**

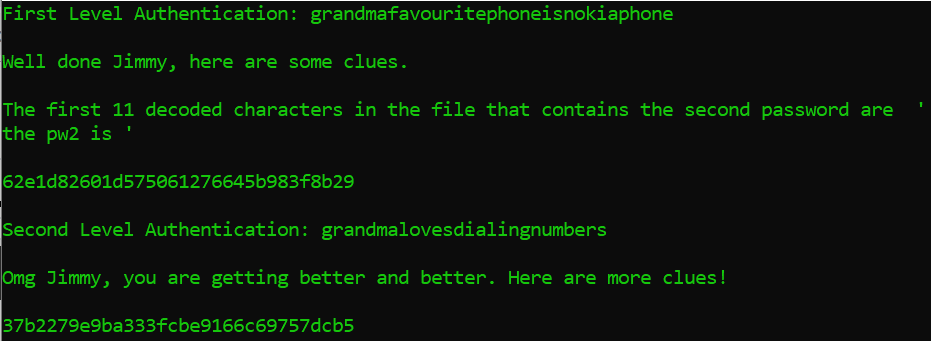
|  |
| --- |
| suspycous.exe |

**Puzzle #1 & Puzzle #2:**

We’re not entirely sure how we were supposed to actually obtain the answers the way the team had intended, cause instead of actually solving the puzzles, we used this [script](https://github.com/countercept/python-exe-unpacker/blob/master/pyinstxtractor.py) to decompile the executable.

This provided us with [this](https://drive.google.com/open?id=1uEW_2UMHiv3yD7z5BA4dj_9ia_qtmaQw), giving us the answers to puzzle #1 and #2.

|  |
| --- |
| grandmafavouritephoneisnokiaphone grandmalovesdialingnumbers |



This felt like a really good reminder to utilize online tools to simplify our job significantly.

**Puzzle #3:**

As well as providing us with the information needed for the first step of puzzle #3.

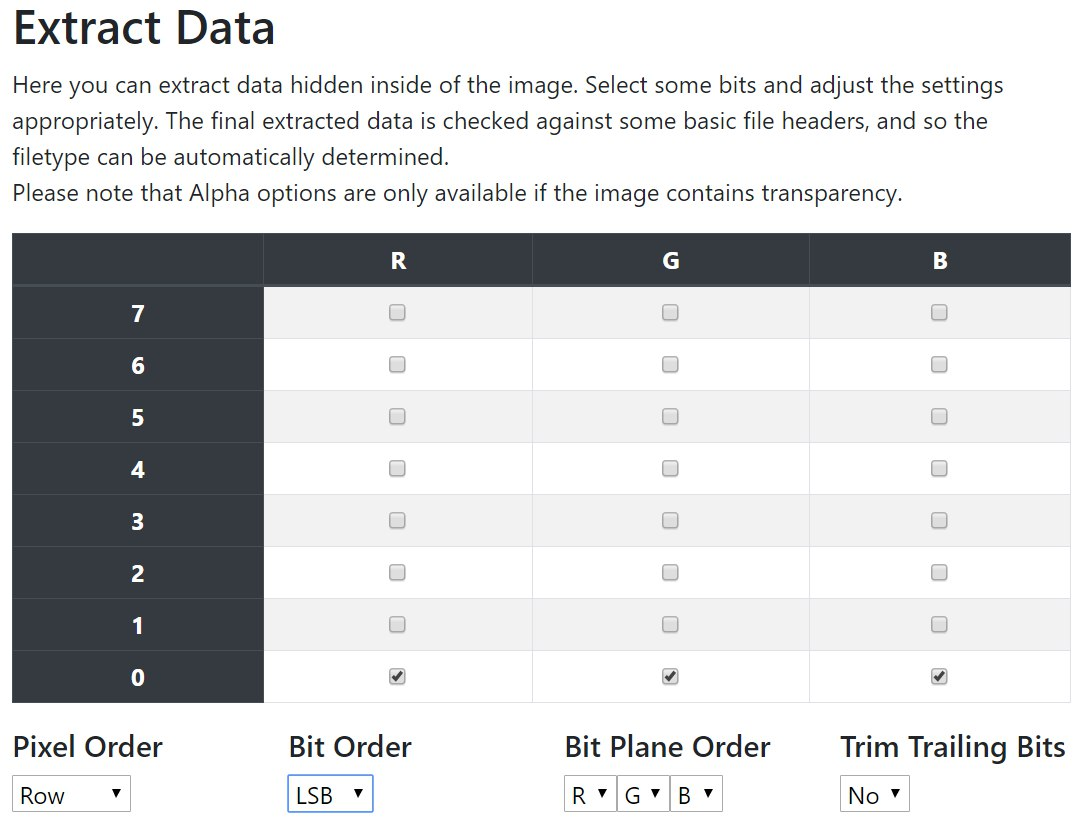
|  |
| --- |
| 62e1d82601d575061276645b983f8b29 37b2279e9ba333fcbe9166c69757dcb5aJ |

The numbers provided were used to find the 2 relevant pictures from the chopped ingredients folder.

Using Photoshop, we combined both images together to obtain the full image:



After obtaining the full image, we read up about [steganography using the LSB of each colour channel](http://datagenetics.com/blog/march12012/index.html). Using [StegOnline](https://georgeom.net/StegOnline), we extracted the LSB of each colour channel from the image of the butter with the following settings:



This gave us a [butter.dat](https://drive.google.com/open?id=1yNIzz794FtYSIVDKLUVX9K8zvUlI9tPf) file with all the extracted bits.



We tried using several .dat to .jpg / .png online file converters to obtain the image from the extracted bits, but none of them worked and they kept crashing. After speaking to the creators, they provided us with the image (of the cake) since they felt that we already knew what we were supposed to do and were only missing that step.   
*(It is worth noting that after this, they sent out a python file that would assist people with extracting the image via email.)*

Thus far I have only experienced steganography with files being embedded within other files as a whole, as opposed to only the LSB being used since the colour difference is hardly noticeable to the human eye. Really interesting...

**Puzzle #4:**

Looking at the full list of ingredients provided in suspycous.exe after solving puzzle #1 & #2 along with the answers to both puzzles, we obtained a list of values.

**Ingredient list:**

|  |
| --- |
| bokcHOY  rocKET  ladyfingERS  tomatOES  turNIP  lIME  sweetpotATO  goURD  onION  spinACH  mustARD   mushroOMS  radISH  blueberrIES  maNGO  kIWI  papAYA  strawbeRRY  watermeLON  olIVE  oraNGE  waterchesNUT  pEAR  lycHEE  cheESE  mILK  butTER  milkcondenSED  oATS  rICE  flOUR  mueSLI  beANS  graPES  shrIMP  rosemARY  lavenDER  cinnaMON  safRON  nutMEG |

**Puzzle #1 & #2 solutions:**

|  |
| --- |
| grandmafavouritephoneisnokiaphone grandmalovesdialingnumbers |

**List of values:**(obtained by taking the last 3 alphabets from the ingredient list and correlating them to the number pad on a Nokia phone)

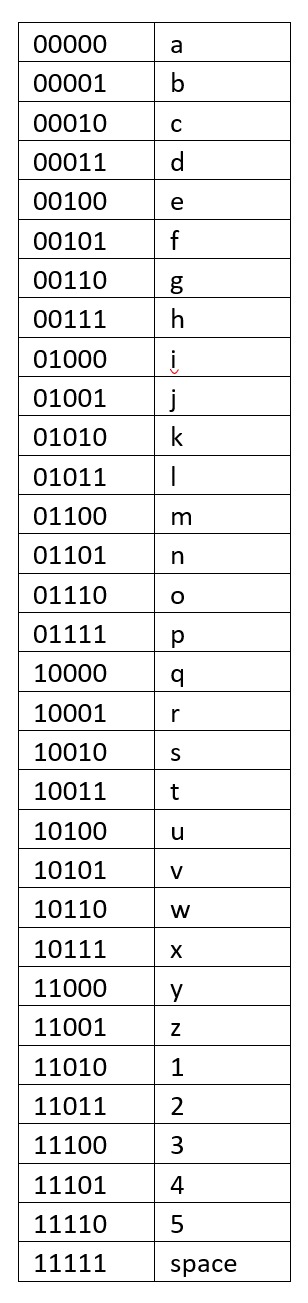
|  |
| --- |
| 469 538 377 637 647 463 286 873 466 224 273 667 474 437 646 494 292 779 566 483 643 688 327 433 373 455 837 733 287 423 687 754 267 737 467 279 337 666 766 634 |

**Solving for the flag:**

1. Read the image of the cake as bytes.
2. Obtain the bytes of the cake at the positions as enumerated in the list of values.
3. Convert the bytes to binary.

|  |
| --- |
| import numpy as np from PIL import Image  cake = np.array(Image.open('flagged\_cake.png')) flattened\_cake = cake.flatten()  values = [469, 538, 377, 637, 647, 463, 286, 873, 466, 224, 273, 667, 474, 437, 646, 494, 292, 779, 566, 483, 643, 688, 327, 433, 373, 455, 837, 733, 287, 423, 687, 754, 267, 737, 467, 279, 337, 666, 766, 634]  bits = "" for val in values:  bits += str(flattened\_cake[val] & 0b1)  print(bits) |

1. Obtain the string of bits 1100000000100010010011000000001000100100, split it into 5 bits at a time, 11000 00000 10001 00100 11000 00000 10001 00100.
2. Use Grandma’s favourite encoding to decrypt the binary to obtain the flag.



**Final flag:** CTF{yareyare}

Initially I was intending to just obtain the flattened image in bytes and manually obtain each value as necessary. As someone who is weaker in programming than most other people, I tend to work faster by manually computing the required information, but solving this puzzle gave me a good balance of working on both -- manual labour along with some coding to speed up my work.