Solution writeup
Hacky Easter 2019

brp64

May 29, 2019

Teaser

Requirements

the following video: `he2019_teaser.mp4`

Goal

Find the teaser Easter egg! Writing a solution document / hacking journal is optional for this case, the solution code is enough!

Solution

The video shows just some flickering of solid colours. Use `ffmpeg` to extract the individual frames:

```
ffmpeg -i he2019_teaser.mp4 teaser%5d.jpg
```

This gives us 230400 frames, which is conveniently 480*480. So use PIL to re-create the image and hope for the best:

```
from PIL import Image
import glob
import os

res = Image.new('RGB', (480,480))
for i in range(230400):
    ix = i % 480
    iy = i // 480

    fn = "teaser%05d.jpg" % (1+i)
    im = Image.open(fn)
    res.putpixel((ix,iy), im.getpixel((0,0)))
    im.close()
    if ix == 0:
        print(iy)

res.save("teaser.png")
```

This shows a beautiful egg with the QR code

`he19-th1s-isju-5tAt-Eazr`

01 - Twisted

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

Solution

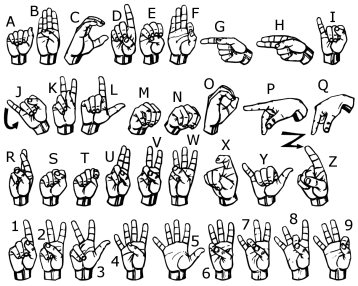
The egg has a swirl effect applied to it. Google for "online remove swirl effect" gives `https://www341.lunapic.com/editor/` as the top hit.

Upload the image and play with the swirl factor to get the egg:

02 - Just Watch

Just watch and read the password.
Then enter it in the egg-o-matic below. Lowercase only, and no spaces!

Solution



The picture contains symbols from the american sign language. Googling for pictures let us decode it easily using "Preview" on MacOS.

The egg is `GIVEMEASIGN`

03 - 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+UrBhm0=
```

`sloppy.rb`

Solution

The file is a simple Ruby script: convert the string into a hex number by converting each character to its ASCII value in hex and joining them. The resulted number is multiplied by a long constant, converted into hex and taking the two numbers at a time to create a byte string. This string is converted to base64 encoding.

Decode by reversing it (see attached script) and getting

`n00b_style_crypto`

Script:

```
enc = "K7sAYzGlYx0kZyXIIPrXxK22DkU4Q+rTGfUk9i9vA60C/ZcQOSWNfJLTu4RpIBy/27yK5CBW+UrBhm0="

factor = int('5'*101)

import base64

b = base64.b64decode(enc)

s = ''.join('{:02x}'.format(c) for c in b)

i = int(s,16)
print(i)
i = i // factor
s = '{:x}'.format(i)

res = ''.join(chr(int(s[2*i:2*i+2],16)) for i in range(len(s)//2))
print(res)
```

04 - Disco 2

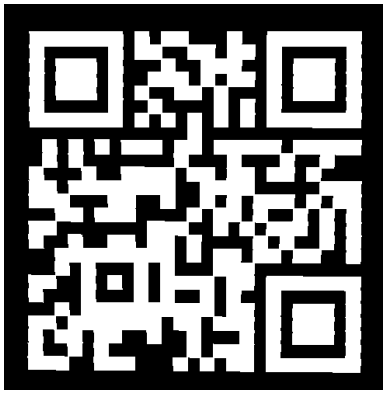
This year, we dance outside, yeaahh! See here.

Solution

The hint points us towards getting inside the sphere, which we can do by setting the variable `minDistance` to zero once we have downloaded the web page to a local directory. Once inside the sphere we see that there is set of mirrors inside the sphere, that form a QR code. So, remove the sphere from the scene and all mirrors on the sphere by filtering out only the mirrors less than 399 from the centre. As a last step, orient the mirrors towards z, make them white, and suppress loading of the background images.

Now we have a nice, inverted colour QR-code. Take a screen shot and use convert to negate the colours to get the egg.

Quick and dirty beats understanding of how to set the background to white :-)



05 - 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?

Solution

The hint about writing it not by himself leads to the document properties. The autor is 'SCIpher'; googling it leads to a MIT-website at `https://pdos.csail.mit.edu/archive/scigen/scipher.html`

This is clearly the author! Decoding the message gives `https://hackyeaster.hacking-lab.com/hackyeaster/images/eggs/5e171aa074f390965a12fdc240.png`

06 - Dots

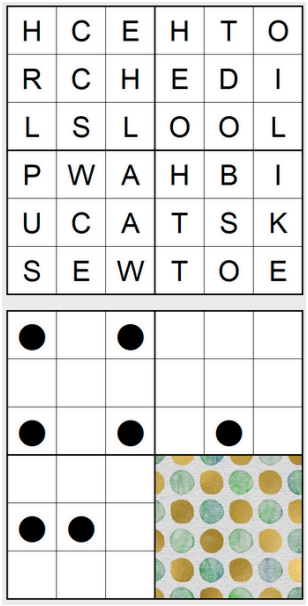
Uncover the dots' secret!
Then enter the password in the egg-o-matic below. Uppercase only, and no spaces!

Solution

Another nice "easy" challenge that was much more difficult to me than some of the "hard" ones. But very original.

The dots in the lower square indicate which of the letters in the upper square have to be read. The covered up square has the dots in the fields of the 3x3 square that were not taken by the other three squares. The solution can then be read from left to right and top to bottom. So it starts with 'HELLOBUCK', but this is not the solution.

Rotating the grid repeatedly by 90 degrees and reading out the letters gives the full solution 'HELLO BUCK THE PASSWORD IS WHITE CHOCOLATE' (spaces added for clarity). So 'WHITECHOCOLATE' is the solution sought.



07 - Shell we Argument

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

Solution

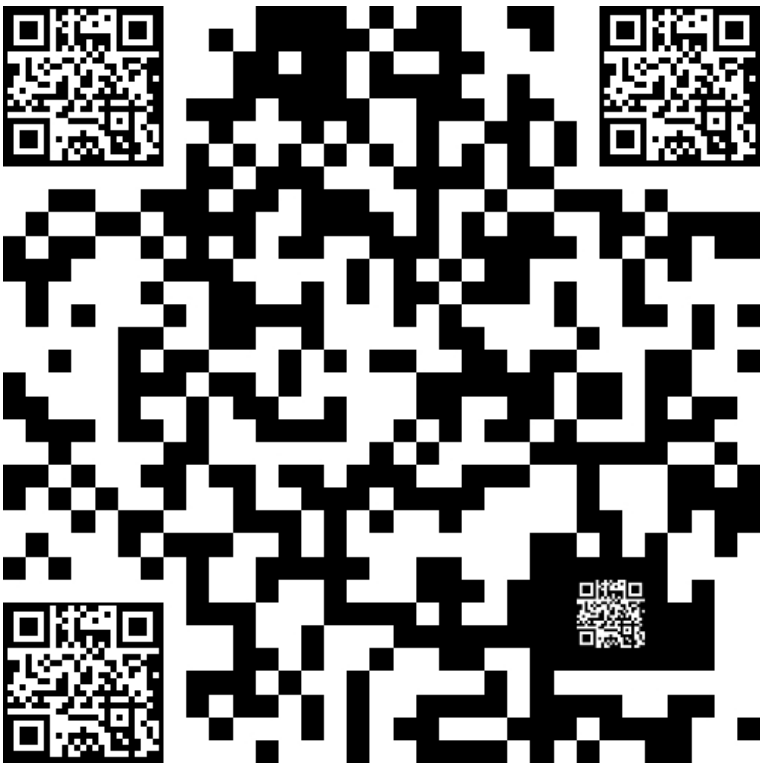
Look at the shell script and prepend the last line by `echo`. Now the script prints the cleartext of the shell script.

Calling the original script as
`sh eggi.sh -R 465 -a 333 -b 911 -I 112 -t 007`
gives the output

```
Ahhhh, finally! Let's discuss your arguments
...
Great, that are the perfect arguments. It took some time, but I'm glad, you see it now, too!
Find your egg at
https://hackyeaster.hacking-lab.com/hackyeaster/images/eggs/a61ef3e975acb7d88a127ecd6e156242c74af38c.png
```

08 - Modern Art

Do you like modern art?



Solution

The picture is of a QR-code with the alignment dots replaced by smaller QR codes that say `remove me`. Replacing them with the proper alignments using Gimp gets me another QR-code that says `Isn't that a bit too easy?`

After some racking my brain, I scanned the .jpg with Stegsolve and found that there is trailing garbage after the file: 611 bytes of unicode text that shows another QR-code. This time it says `AES-128`

So scan the file again for strings with

```
strings modernart.jpg | grep '^('
```

Figure 1: Small QR codes.

Figure 2: Main QR code.

and find two interesting candidates:

```
(E7EF085CEBFCE8ED93410ACF169B226A)
(KEY=1857304593749584)
```

So this could be an AES-128 encrypted string... Test it with `http://aes.online-domain-tools.com` and get

```
Ju5t_An_1mag3
```

(enter the secret in hex without brackets, key as text)

09 - rorriM rorriM

Mirror, mirror, on the wall, who's the fairest of them all?
`evihcra.piz`

Solution

The file given is kind of mirrored (archive.zip - evihcra.piz), together with the title of the challenge and inspection of the file using a hex editor, I quickly came to the conclusion that the file has to be byte-reversed to get a proper zip archive. I wrote a short python script:

```
with open('evihcra.piz', 'rb') as inF:
  b = inF.read()
  with open('archive.zip', 'wb') as outF:
    outF.write(b[::-1])
```

The resulting archive contains a file `90gge.gnp`, another hint for mirroring. Inspecting it in the hex editor, the magic number is compatible with a PNG file, except that it contains `GNP` instead of `PNG`. Changing these letters gives a valid PNG file that is mirrored and has funny colours. The QR code has black and white exchanged. Use convert to mirror and invert the image:

```
convert -flop 90gge.png -negate egg09.png
```

... and we're done.

10 - 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

Using ZAP to do some reconnaissance, I found this message when looking at `robots.txt`:

```
Maybe the_admin knows more about the flag.
```

So probably `the_admin` is a user.

The questions hint at the possibility that there is a NoSQL database in the background and that json is involved. Since the login page is the only place to send data, try a NoSQL injection.

See `https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/NoSQL%20Injection` for a helpful cheat sheet.

Use ZAP to intercept the login and replace it with a JSON version

```
{"username": "the_admin", "password": {"$ne": null}}
```

and we get the login! Under the question section we see that the password of `null` is really the flag. So get the password from the store!

Again, use an adapted script from the NoSQL injection site, we find the password of `null`:

```
NOSQL_injections_are_a_thing
```

Just for kicks, I tried to get all users on the system with their passwords:

```
{'null': 'NOSQL_injections_are_a_thing',
 'hax0r': '9vVVtFYBJPmk2wFw',
 'no_one': 'M5fnk5dVLtuXQ3NG',
 'the_bean': 'sAh3ZTREZggg2svM',
 'the_admin': '76eKxMEQFcfG3fPe'}
```

11 - 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.

Solution

The memory game has changed from last year in that picture x on the server is not the same each time. So in different rounds `http://whale.hacking-lab.com:1111/pic/80` will return a different image.

So the plan of attack is simple:

- download all images
- calculate the md5 of each image
- same md5 = pair
- click all the pairs

- go to next round

First approach: load page manually, save to local, run a script that creates JavaScript code to click all images in the right order. The JavaScript was stolen from evandrix' solution of Memeory 1.0 :-) A snippet of the code looks like this:

```
['12','85', ...].forEach((pic)=>
   {console.log(pic);$("img.boxFront").filter((i,el)=>$(el).attr("src")===pic).click();});
```

Unfortunately, this did not work reliably. So back to the programming desk to write a script to do everything in one go:

```
import hashlib
import base64

base_url = "http://whale.hacking-lab.com:1111/"

def getPics(s):
    md = {}
    for i in range(2,99):
        u = base_url+'pic/%d'%i
        print(i, end='')
        r = s.get(u)
        try:
            h = hashlib.md5()
            h.update(base64.b64encode(r.text.encode('utf-8')))
            d = h.hexdigest()
            if d in md:
                md[d].append(i)
            else:
                md[d] = [i]
        except ValueError:
            pass
    print()
    return(md)

def solveRound(session):
    solve_url = base_url+'solve'
    r = session.get(base_url)
    md = getPics(session)
    post_data = {'first': '1', 'second': '2'}
    for k in md:
        if len(md[k]) == 2:
            if md[k][0] != 1 and md[k][1] != 1:
                post_data['first'] = int(md[k][0])
                post_data['second'] = int(md[k][1])
                r = session.post(solve_url, post_data)
        else:
            last_pic = md[k][0]
    post_data['first'] = int(last_pic)
    post_data['second'] = int(1)
    r = session.post(solve_url, post_data)
    return(r.text)


def solve():
```

```
count = 0
ok = 'nextRound'
res = ok
import requests
session = requests.session()
while res == ok:
    count += 1
    print('round: ', count)
    res = solveRound(session)
```

The main function `solve` calls `solveRound` for each round, which in turn calls `getPics` to download all files and build a dictionary of all pairs. Picture 1 has to be handled separately, as every reference to `./pic/1` will create a HTML response of 500, so we just do this as the last pair.

In the end, the script prints the flag

```
ok, here is your flag: 1-m3m3-4-d4y-k33p5-7h3-d0c70r-4w4y
```

12 - 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.

Solution

Open Hopper and extract the hash-function as pseudo-c code. It seems really complicated, but can be simplified quite a bit:

- it uses a password of maximum 32 characters
- there is a long array in memory (845 bytes long)
- the password is copied repeatedly into an array of the same length.
- the two arrays are massaged, 4 byte at a time to write the output

Using some simple rules and some trial-and-error, I could narrow it down to a byte-wise operation between the two arrays. Not having any clue about the shape of the flag, I ran `strings decryptor` and found `XOR_Challenge.c` as one string.

So could it be that the complicated expression is really just an xor? Yes it is!

So decoding comes down to byte-wise xor of the hidden string with the password. Cracking XOR is easy if one has an idea of the cryptext. After some lengthy guesswork, starting with variations of 'XOR', I realised that the first word could be 'Hello' or 'Hell0' or 'He1ll0'. Starting with this educated guess, I found that shifting the password by 26 characters gave other good-looking text fragments. So the length of the password is probably 26 characters.

Now it is just a matter of finding characters around word-fragments and extending the password one by one. Finally, I arrived at the password `x0r_w1th_n4ndx0r_w1th_n4nd` and this produced the solution text

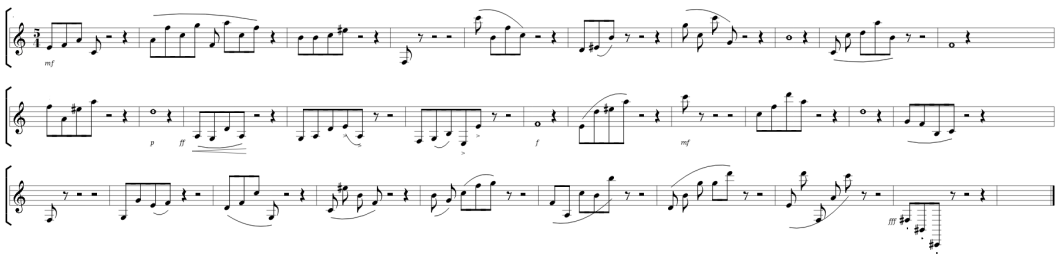
```
Hello,
congrats you found the hidden flag: he19-Ehvs-yuyJ-3dyS-bN8U.

'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)
```

Obviously, the real password is `x0r_w1th_n4nd`, but when it is late, one does not recognize it :)

It retrospect, calculating the index of coincidence would have given the key-length quickly and clearly.

13 – 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. Upper-case only, and no spaces!

Solution

A nice problem to be solved by pen and paper!

So it seems that we have to look at the number of notes and some thing about the full notes. After some head-scratching, I noticed that the bars always end in pauses and start with notes, except if there is a full note. The number of notes seems often to be four or five. Then it clicked, that it is hex notation of ASCII codes: for less than nine it is coded with quavers, the semibreves give the lettes a-h based on the frequency. Then it is easy

```
msg = [0x48, 0x41, 0x43, 0x4b, 0x5f, 0x4d, 0x45, 0x5f,
       0x41, 0x4d, 0x41, 0x44, 0x45, 0x55, 0x53]
for c in msg:
        print(chr(c), end='')
print()
```

or HACK_ME_AMADEUS

14 - White Box

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

```
9771a6a9aea773a93edc1b9e82b745030b770f8f992d0e45d7404f1d6533f9df348dbccd71034\
aff88afd188007df4a5c844969584b5ffd6ed2eb92aa419914e
```

Solution

After being mislead to thinking the white-box referred to knowing everything about the tool and having to reverse-engineer the binary, I was pointed into the right direction by keep3r. After a lot of searching and reading about white box cryptography, I finally started using Deadpool dfa. Some tweaking of the parameters found the key:

```
root@hlkali:306# python3 egg14_dfa.py
Press Ctrl+C to interrupt
Send SIGUSR1 to dump intermediate results file: $ kill -SIGUSR1 21291
Lvl 016 [0x0001F144-0x0001F145[ xor 0xE3 74657374746573747465737474657374 -> 6BDDDF1521A95452616187212F4C37D4 Go
Lvl 016 [0x0001F144-0x0001F145[ xor 0xF6 74657374746573747465737474657374 -> 60DDDF1521A954E9616116212FA637D4 Go
Lvl 016 [0x0001F144-0x0001F145[ xor 0x20 74657374746573747465737474657374 -> 98DDDF1521A954CD616139212F6037D4 Go
Lvl 016 [0x0001F144-0x0001F145[ xor 0xB4 74657374746573747465737474657374 -> A7DDDF1521A954316161D7212F2B37D4 Go
Lvl 016 [0x0001F145-0x0001F146[ xor 0xC7 74657374746573747465737474657374 -> 8DDDDFC021A96C5561D7CA2106D237D4 Go
Lvl 016 [0x0001F145-0x0001F146[ xor 0x1B 74657374746573747465737474657374 -> 8DDDDF5921A99F556165CA218DD237D4 Go
Lvl 016 [0x0001F145-0x0001F146[ xor 0x48 74657374746573747465737474657374 -> 8DDDDF2521A99355612CCA2114D237D4 Go
Lvl 016 [0x0001F145-0x0001F146[ xor 0x82 74657374746573747465737474657374 -> 8DDDDFB721A97455613ACA216CD237D4 Go
Lvl 016 [0x0001F146-0x0001F147[ xor 0x4A 74657374746573747465737474657374 -> 8DDDD715218954557161CA212FD23708 Go
Lvl 016 [0x0001F146-0x0001F147[ xor 0x3D 74657374746573747465737474657374 -> 8DDD051521065455E361CA212FD237EE Go
Lvl 016 [0x0001F146-0x0001F147[ xor 0xA0 74657374746573747465737474657374 -> 8DDD791521B554551B61CA212FD23783 Go
Lvl 016 [0x0001F146-0x0001F147[ xor 0x08 74657374746573747465737474657374 -> 8DDD7C1521DF54551161CA212FD2373D Go
Lvl 016 [0x0001F147-0x0001F148[ xor 0x33 74657374746573747465737474657374 -> 8D9ADF15F0A954556161CA092FD217D4 Go
Lvl 016 [0x0001F147-0x0001F148[ xor 0x86 74657374746573747465737474657374 -> 8D13DF15A4A954556161CABA2FD298D4 Go
Lvl 016 [0x0001F147-0x0001F148[ xor 0xF1 74657374746573747465737474657374 -> 8D60DF15EFA954556161CA762FD2EED4 Go
Lvl 016 [0x0001F147-0x0001F148[ xor 0x37 74657374746573747465737474657374 -> 8DB8DF157CA954556161CA7A2FD2C1D4 Go
```

```
Saving 17 traces in dfa_enc_20190519_113333-113405_17.txt
(['dfa_enc_20190519_113333-113405_17.txt'], [])
dfa_enc_20190519_113333-113405_17.txt
Last round key #N found:
FD83DB41AC158393CC291088B76F201A
```

Now use `inverse_aes.py` from `https://github.com/ResultsMayVary/ctf/blob/master/RHME3/whitebox/inverse_aes.py` to recover the key.

```
root@hlkali:308# python ../inverse_aes.py FD83DB41AC158393CC291088B76F201A
Inverse expanded keys = [
        fd83db41ac158393cc291088b76f201a
        91879460519658d2603c931b7b463092
        508d33cfc011ccb231aacbc91b7aa389
        a0c83a2a909cff7df1bb077b2ad06840
        9f60d8933054c5576127f806db6b6f3b
        96e8ff67af341dc451733d51ba4c973d
        f344af8e39dce2a3fe472095eb3faa6c
        473a36d7ca984d2dc79bc23615788af9
        5268bc628da27bfa0d038f1bd2e348cf
        b1aef4fcdfcac79880a1f4e1dfe0c7d4
        336d62336e6433645f6b33795f413335
]
Cipher key: 336d62336e6433645f6b33795f413335
As string: '3mb3nd3d_k3y_A35'
```

Using an on-line AES-decoder, the criptext can be decoded:

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

15 - 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

Easter 2018 = 01 April 2018

The Steem blockchain can be explored with the Steem explorer. With this I limited the blocks to within about 30'000 blocks. Then each block can be downloaded from `https://steemdb.com/block/123456789`.

So, wrote a script that downloads the blocks and then `grep` though them for `Hacky`. This gives:

```
21187964.text: "memo": "Hacky Easter 2019 takes place between April and
                                        May 2019. Take a note: nomoneynobunny"
```

... and we have the egg!

16 - Every-Thing

After the brilliant idea from here.

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

`EveryThing.zip`

Solution

I solved this challenge while off-line. In retrospect, it would have been better to wait until I had internet access again and install a docker container with MySQL... But so I wrote a Python script to read the dump and extract the data structure so that I could traverse it in Python.

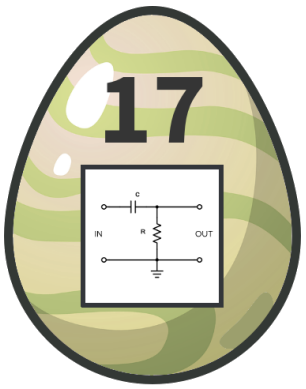
As a first step I built a new dictionary called children that lists all children of a given Id. Descending from the 'ROOT' node gets us to the 'galery', which in turn contains images.

Each image file is again split into different sections. The payload of the sections is in the data column and is base64 encoded. Ordering the fragments by the order-column `ord` makes the extraction straight forward:

```
def get_png(p, i):
    print('starting png ', p)
    with open('file%d.png' % i, 'wb') as outF:
        order = 0
        found = True
        while found:
            found = False
            for ch in children[p]:
                if int(db[ch]['ord']) == order:
                    found = True
                    if ch in children:
                        o2 = 0
                        found2 = True
                        while found2:
                            found2 = False
                            for ch1 in children[ch]:
                                if int(db[ch1]['ord']) == o2:
                                    found2 = True
                                    outF.write(base64.b64decode(db[ch1]['dta']))
                            o2 += 1
                    else:
                        outF.write(base64.b64decode(db[ch]['dta']))
            order += 1
```

Calling the above function for each child of 'galery' produces a number of pictures, number 7 contains the QR-code.

17 - 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

Following a hint by keep3r, we started to look into the PNG specification. As it turns out, there are filter fields for each scanline. While looking for a program to dissect a PNG file, I came across `pngtest.c`. Running it showed, that only two filter values were used in `eggdesign.png`. So why not dump them in order, hoping for a flag. Instead of writing my own program, I just changed one function of `pngtest.c` to print out all filters:

```
count_filters(png_structp png_ptr, png_row_infop row_info, png_bytep data)
 {
    if (png_ptr != NULL && row_info != NULL)
      {
       ++filters_used[*(data - 1)];
       printf("%d ", *(data - 1));
      }
 }
```

Interpreting the output as bits in the ASCII representation of characters and re-assembling them gives

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

18 - Egg Storage

Last year someone stole some eggs from Thumper.
This year he decided to use cutting edge technology to protect his eggs.
`Egg Storage`

Solution

Inspecting the source code of the page shows some web assembly code. Using a small script I converted the Uint8 array to a binary file and uploaded this to `https://webassembly.github.io/wabt/demo/wasm2wat/` to get an understandable file.

Analysing it, it becomes clear that the password is 24 characters long, starts with `Th3P` and the rest uses only the characters with ASCII codes in the set 32,48,49,51,52,53,72,76,88,99,100,102,114. A set of equations is also hidden in the source code and I could iteratively recover the password, only resorting to guessing the last character. My python code looks like this:

```
def searchPassword():
    vals = [48, 49, 51, 52, 53, 72, 76, 88, 99, 100, 102, 114]
    p = [' ' for x in range(24)]
    p[0] = chr(84) # 'T'
    p[1] = chr(104) # 'h'
    p[2] = chr( 51) # '3'
    p[3] = chr( 80) # 'P'

    # if (p[9] % p[8]) != 40:  # p[9] = 'X'/88, p[8] = '0'/48
    # only possible for one combination
    p[8] = '0'
    p[9] = 'X'
    # if (2 * (p[9] % p[4])) == p[13]: # 2*(88 % p4) == 72  p4 = 88 - 36 = 52/'4'
    # only possible for one combination
    p[4] = chr(52)
    p[13] = chr(72)
    # if (p[13] % p[6]) == 20:  # p[13] = 'L'/72, p[6] = '4'/52
    p[6]  = chr(52)

    # if (p[7] % p[6]) == p[10]: # 100 % 52 = 48, p[10] = '0'/48
    # if (p[5] - p[7]) != 14:  # p[5] = 'r'/114, p[7] = 'd'/100
    p[7]  = chr(100) # 48, 49, 51 for p7 do not fit the p5-p7 criterion
    p[10] = chr(48)
    # if (p[5] - p[9]) + p[19] != 79: # 114 - 88 = 26, 79 - 26 = 53, p[19] = '5'
    p[5]  = chr(114)
    p[19] = chr(53)
    # if (p[11] % p[13]) != (p[21] - 46):
    # p11 =  53 5 p21 =  99 c
    # p11 =  102 f p21 =  76 L
    # p11 =  114 r p21 =  88 X
    p[11] = chr(102)
    p[21] = chr(76)
    # if (p[7] - p[14]) == p[20]:  # 100 - p14 = p20
    # p14 =  48 0  p20 =  52 4
```

```
    # p14 =  52 4  p20 =  48 0
    p[14] = chr(48)
    p[20] = chr(52)
    # if (p[14]+1) == p[15]:
    p[15] = chr(ord(p[14]) + 1)

    p[23] = chr(51)
    p[22] = chr(49)
    # if p[23] != p[17]:
    p[17] = p[23]
    # if p[22] != p[15]:
    p[15] = p[22]
    # if p[12] == p[16]:
    p[12] = 'c'
    p[16] = p[12]
    # if (p[23] % p[22]) != 2:
    # p23 =  51 3  p22 =  49 1   --> 13 (ie)
    # p23 =  53 5  p22 =  51 3   --> 35 (es)
    # p23 =  100 d  p22 =  49 1  --> 1d (id)
    # p23 =  102 f  p22 =  100 d --> df

    # guess
    p[18] = '1'
    for c in vals:
        for c2 in vals:
            if (c % c2) == 2:
                p[23] = chr(c)
                p[22] = chr(c2)
                p[17] = p[23]
                p[15] = p[22]
                print('----:',''.join(p[i] for i in range(len(p))), 'XXXXX')
    return
```

In the end the password is `Th3P4r4d0X0fcH01c3154L13` or in non-1337 `TheParadoxOfChoiceIsALie`. Nice!

19 - CoUmpact DiAsc

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

`compudiasc`

Solution

I used Ghidra to analyse the binary. It is infact a CUDA-program, something that I only knew from literature. The analysis pointed to two buffers of 256 bytes each, containing values that were close to the AES sbox and inverted sbox. It also shows that the password is 16 bytes long. There is no hashing or similar going on to generate the key for AES, so the password is used as key itself. From the 16-byte key, we also know that we are likely dealing with AES-128. The scrambled egg is embedded in the binary and its size is very similar

to eggs from other challenges.

So what do we know so far:

- an AES-128 encrypted PNG egg is embedded in the binary.
- the password read from the command line is the key to AES-128

Checking all the other solution eggs reveals, that the first 16 bytes of the PNG-files are always the same:

```
root(kali)$ od -t x1 ../egg02/87340bc4296ff0f53b41c5ef2312139e1af818d4.png| head -1
0000000    89  50  4e  47  0d  0a  1a  0a  00  00  00  0d  49  48  44  52
```

So we know the plaintext of the first 16 bytes of the ciphertext, a perfect base to do brute-forcing.

The image shows a label that indicates that the password ends with 'THCUDA'. Assuming that the password ends with "WITH-CUDA" and that it is 16 characters long, I wrote a simple brute force cracker using `https://github.com/kokke/tiny-AES-c`.

This cracker finds the password `AESCRACKWITHCUDA` within a few minutes.

Then writing decoding the egg using python was easy :). I used gdb and the hackinglab-CD to write the memory from the binary to file:

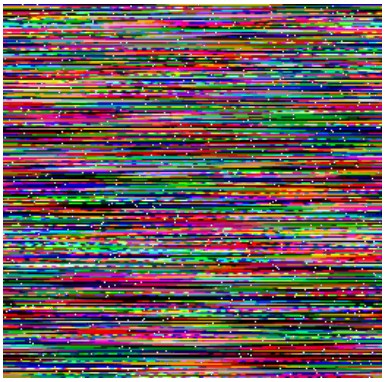
```
gdb-peda$ dump binary memory egg.bin 0x00687500 0x0069829f
```

Then this gets us the egg:

```
from Crypto.Cipher import AES

def decrypt(ciphertext,key, mode):
    encobj = AES.new(key,mode)
    return(encobj.decrypt(ciphertext))

pw = bytearray('AESCRACKWITHCUDA','ascii')
with open('egg.bin', 'rb') as enc:
    l = 69023
    pad = b'\x00'*((l // 16) * 16 + 16 - l)
    cript = enc.read(l)
    cript = cript + pad
    egg = decrypt(cript, bytes(pw), AES.MODE_ECB)
    with open('egg.png', 'wb') as dec:
        dec.write(egg)
```

20 – Scrambled Egg

This Easter egg image is a little distorted...
Can you restore it?



Solution

The image is really distorted, mostly it seems that the lines have been shuffled. My reasoning was, that the individual lines in eggs always have an alpha- channel of zero so that the background can be seen. So my first try was to look at the alpha channels for anything hidden. It turns out, that each line has exactly three pixels with alpha == 0, so this seems to be interesting.

Furthermore, when alpha == 0, then two of the three components (r,g,b) are zero and the third has a value v. This value v is the same on a line. Each value occurs only on one line, so this is probably the line number.

Creating an image with sorted lines does not really produce a better picture.

Looking at adjacent lines, I noticed, that around the pixels with alpha == 0, e.g. (r,g,b,a) = (128,0,0,0), the component that is non zero has a value of 1 in the pixels to the left and right of the alpha == 0 pixel. Low values correspond to dark pixels that are at the edge of the egg. So it could be that the individual colour-components are rotated to the left by this number of pixels. Within a line, each colour component is rotated by a different amount to the left and has to be rotated right by the same amount to untangle the picture.

Implement this rotation for the red components:

```
red[i] = red[(i+shift_red)%len(red)]
```

This does the trick and in the end we get a readable image. Realizing that some of the pixels will be off because the components are forced to zero, gives room for improvement: interpolate the colour channels around the alpha==0 pixels with the average of the horizontally neighbouring pixels, now the ugly speckles are gone, but there are still some artefacts from downscaling. Anyhow, this last step was optional, but rewarding :-)

Following up again on a chat with 0x1, I realised that I had it not completely correct: the picture should be square! So the pixels with alpha==0 have to be dropped and then shifted. Now the picture looks as it should.

21 - 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.

Entering the maze

Solution The user is presented with a quite simple interface. The only noticeable thing is the cryptic string “bqq‘vsm“0npwf0y0z

When the challenge is started, the screen stays empty. There is only a comment about knowing how to move. Inspecting the source of the page shows an unused JavaScript function:

```
<script type='text/javascript'>
      // add all this variables later
      let youCanTouchThis = "";
      let youCantTouchThis = "";
      let randomNumber = undefined;

      for (let i = 0; i < youCantTouchThis.length; i++) {
          if (youCantTouchThis.charCodeAt(i) === 28) {
              youCanTouchThis += '&';
          } else if (youCantTouchThis.charCodeAt(i) === 23) {
              youCanTouchThis += '!';
          } else {
              youCanTouchThis += String.fromCharCode(
                    youCantTouchThis.charCodeAt(i) - randomNumber);
          }
      }
      // document.write(m);
  </script>
```

This function transforms a string by a rotation (like a Caesar cipher). Checking all possible shifts, gives a likely possibility for the cryptic string:

```
__app_url__/move/x/y
```

So this must be the way to move around the maze. We can only move by one unit in either the x or the y direction. If we hit a wall, the user interface tells us. So walking around, we found challenge C11:

C11 – Warmup

Two images of a bunny wearing a crown. One is about double the size of the other. Since there are pixels sought, calculate the differ-

ence of the two files. There are only a few pixels that differ, so print them out:

```
[[6, 131], [7, 405], [169, 510], [250, 577], [276, 567], [328, 398],
[331, 99], [412, 248], [412, 361], [474, 385]]
```

First puzzle down.

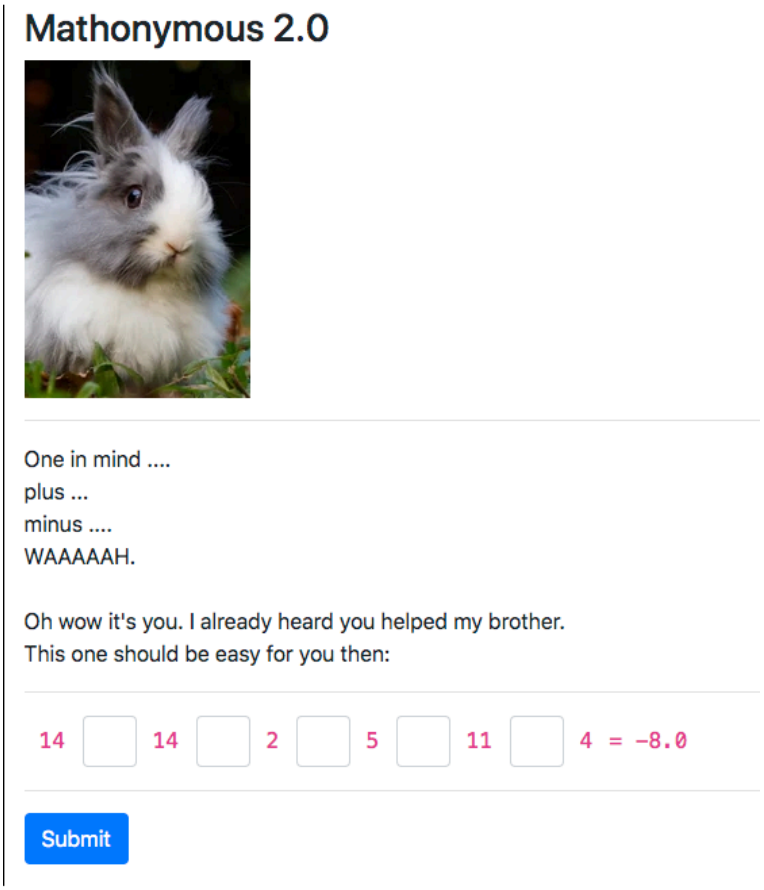
C12 – C0tt0nt4il Ch3ck V2.0 required

WARNING! C0tt0nt4il Ch3ck V2.0 required

You need 10 right answers in time!

You have to solve some math test. Fortunately, the result is part of the filename. For example in `63fed1bb-3d69-123-b64a-7b13bac2bee6.png`, the 123 is the result of the calculation.

C13 – Mathonymous



A simple math problem. Use python to find the solution and choose the one that has a floating point number as a result: 14 - 14 + 2 + 5 - 11 - 4 = -8 14 / 14 * 2 + 5 - 11 - 4 = -8.0

T01 – Myterious Circle

Myterious Circle

You step onto the circle and feel some kind of energy flowing through your body.
It feels like you could do some kind of giant jump through the map, but as soon as you raise your toes
and try to jump, you simply land onto them again without any special effect.

Maybe it's too early and something needs to be done before this circle works?

You see some scratched letters on a stone.

Read

Once all challenges C11–C13 are solved, we are transported to the second level of the maze.

C14 – Pumple's Puzzle

Hey I'm Pumple.

My Puzzle is very famous around here. Do you think you have what it takes to solve it?

No, you don't - haha! Noone solved it yet.

There are five bunnies.

The backpack of Bunny is green.

Midnight's star sign is capricorn.

The one-coloured backpack is also red.

The chequered backpack by Snowball was expensive.

The bunny with the red backpack sits next to the bunny with the yel-

low backpack, on the left.

The taurus is also handsome.

The scared bunny has a blue backpack.

The bunny with the dotted backpack sits in the middle.

Thumper is the first bunny.

The bunny with a striped backpack sits next to the funny bunny.

The funny bunny sits also next to the aquarius.

The scared bunny sits next to the pisces.

The backpack of the lovely bunny is camouflaged.

Angel is a attractive bunny.

Thumper sits next to the bunny with a white backpack.

	Bunny #1	Bunny #2	Bunny #3	Bunny #4	Bunny #5
Name	Thumper	Snowball	Bunny	Angel	Midnight
Color of backpack	Blue	White	Green	Red	Yellow
Characteristics	Scared	Funny	Handsom	Attractive	Lovely
Star sign	Aquarius	Pisces	Taurus	Virgo	Capricorn
Pattern on backback	Striped	Chequere	Dotted	One-colo	Camoufla

C15 – Punkt.Hase

Hey my friend, I found this one, on my journey.

Do you know what to do with it?

Look at the picture of the flashing dot in preview and decode the black and white frames into 0 and 1. There are 112 frames, so this is probably an ASCII encoded character.

Decoded: `xafcxdvhhopmhi` and the bunny signals back: `Congratulations, you solved it! Travel on, my friend!`

C16 – Pssst ...

```
... come over here, listen and answer me.

He: (?<=13)37
You:
Answer
```

When visiting again, the equation changes... (?<!13)37

If you read it aloud, it could be kind of an if statement:

'if challenge(?) <= 13 then return 37'

On the second visit it read if challenge(?) <! 13 then return 37', so entering 37 solves the riddle.

C17 – The Oracle

```
You didn't come here to make the choice. You've already made it.
You're here to try to understand why you made it.
```

```
Who I am you ask? Just call me "The Oracle". I know you want to help me with this.
The oracle has a hint for you!
Start with the number I gave you as seed, use the next random number in range as
    A NEW seed and after doing it 1336 times, you will get the right answer!!
import random
random.randint(-(133742), 133742)
-691083169870039031099715349702031846726761124816678481232010009688718318689017576672518649911065123018590450758951440750098384864\
  2
Guess the next 1337's number!
```

```
import random


random.seed(-691083169870039031099715349702031846726761124816678\
                      481232010009688718318689017576672518649911065123018\
                      5\
                      904507589514407500983848642)

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

```
  The Oracle says you didn't want to know, if it's right
or wrong.  You want to know, why you solved this.
```

```
11713164348900759365418636485086136015950689705433032672953646403385\
    45875897886242652817320998027472598653798767165107935526607987439
```

At a later visit we find a note:

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?.'

C18 – CLC32

Dreams ... just dreams, but today you have the chance to live a second life. Go! Start breathing and prove that you are worth this new beginning and don't make the same mistakes again. Hints for your restart: Do something when 3 or more sins tell you it is right.

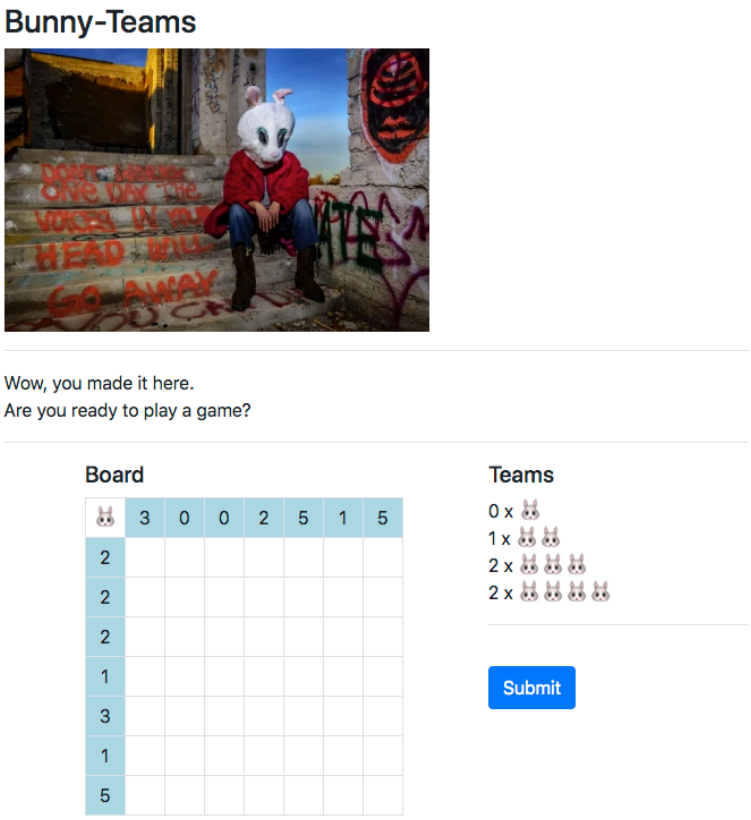
I found this challenge very challenging. It turns out, that the link leads to an URL that accepts GraphQL queries. With the help of `https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/NoSQL%20Injection` I was able to download the schema of the queries. Playing around, the query

```
{ In {see hear taste smell touch Out {see hear taste smell touch } } }
```

gives results back. These change with every retrieval and are dependent on the session-cookie. The hint points out, that we should restart the retrieval, if three of the five senses give the same answer. The letter should be written down. The four letter code is then the solution.

Very difficult.

C20 – Bunny-Teams



And the solution to battle ships:

F02 – Opa & CCrypto - Museum

You are too late for their famous story telling. The original story tellers left already several years ago.

Many people liked the stories they told, but they got kind of one-sided at the end of their career.

Today we know they used a specific formula to change their storiess and all the containing chapters in a magic way.

The notes we found have been implemented into this site.

The exit page holds another riddle: in the web-page is a snippet of js with very long arrays. Analysing the code, it seems that these arrays are the product of running the code with a starting vector. So the aim is to reverse the operation.

Throwing away all unneccessary code, one operation cycle is

```
def crank(s, box):
    s += 3
    # calculate the new s
    tmp = []
    ses = []
    for i in range(len(box)):
        s = box[i]['s'] + abs(math.floor(math.sin(s) * 20))
        l = list(box[i]['t'])
        l.append('%d' % i)
        tmp.append({'s':s,'t': l})
        ses.append(s)

    newBox = []
    ses = sorted(list(set(ses)))
    for i in ses:
        for j in range(len(tmp)):
            if tmp[j]['s'] == i:
                newBox.append(tmp[j])
    return newBox
```

The box consists of twenty arrays, each has a integer and a string. The string encodes the position of this array within the box with each operation. So there are 7331 positions per line.

Reversing this operation uses a few functions, but ultimately leads to the starting values.

```
def reverse_sort(box):
    newBox = []
    for i in range (len(box)):
        for j in range(len(box)):
            if int(box[j]['t'][-1]) == i:
                newBox.append(dict(s=box[j]['s'], t=box[j]['t'][:-1]))
                break

    return newBox

def get_last_s(box):
    if len(box[0]['t']) == 0:
        return 0
    else:
        for i in range(len(box)):
            if int(box[i]['t'][-1]) == 19:
                return box[i]['s']


def reverse_shuffle(box):
    newBox = reverse_sort(box)
    for i in range(len(newBox)-1,0,-1):
        s_old = newBox[i]['s'] - abs(math.floor(math.sin(newBox[i-1]['s'])*20))
        newBox[i]['s'] = s_old

    s = get_last_s(newBox) + 3
    s_old = newBox[0]['s'] - abs(math.floor(math.sin(s) * 20))
    newBox[0]['s'] = s_old
    return newBox

def unwind(box):
    nruns = len(box[0]['t'])
    for i in range(nruns):
        box = reverse_shuffle(box)
    return box
```

Within the js-script is also a string that can be used to print the starting values as text:

```
def print_box(box):
    a = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
    print('of a, unordered')
    for i in range(len(box)):
        print(a[box[i]['s']%len(a)], end='')
    print()
```

Results in `he19JfsMywiwmSxEyfYa`

22 - Muddy quagmire (not solved)

C01 – Old Rumpy

Timezone calculation: what time is it in Lusaka, when my current timezone is GMT?

I got the same result - we should be right yaaaah. Aw, I'm so excited for that travel. You know what? I'll just go now!

Bye!

C02 – Simon's Eyes

Hi, I'm Simon from the Security Team.

Do you really pay attention? I saw every step you made!

Tell me which moves you made till here from the beginning.

Took me a long time to get right: enter the path you took from the entrance to Simon, but pay special care to get the direction of the y-axis right.

```
?path=["3"%2C"3"%2C"3"%2C"1"%2C"1"%2C"4"%2C"4"%2C"4"%2C"1"%2C"1"%2C"3"%2C"3"%2C"3"\
% 2C"1"%2C"1"%2C"4"%2C"4"%2C"4"%2C"1"%2C"1"%2C"3"%2C"3"%2C"3"%2C"3"%2C"3"%2C"6"%2C"6"\
% 2C"6"%2C"6"%2C"6"%2C"6"%2C"6"%2C"6"%2C"6"%2C"6"%2C"4"%2C"4"%2C"4"%2C"4"%2C"4"%2C"4"\
% 2C"4"%2C"4"%2C"4"%2C"1"%2C"1"%2C"1"%2C"1"%2C"1"%2C"1"%2C"4"%2C"4"%2C"4"]
```

C03 – Mathoymous

Hmmmm.... one in mind .... plus ... minus .... WAAAAAH. Oh hey you came just right. Could you solve this until I search for my calculator? Who I am you ask? I think that would be a bit embarrassing because I can not solve this simple equation.

56-43+60*47 =

C04 – Randonacci

What a beautiful chain, but the last piece is missing.

Do you know what we need?

HINT

```
random.seed(1337)
sequence.append(1 % random.randint(1, 1))
sequence.append(1 % random.randint(1, 1))
sequence.append(2 % random.randint(1, 2))
```

Greetz, Leonardo F.

The chain has a row of elements with following numbers printed on:

```
[0, 0, 0, 1, 2, 3, 6, 0, 10, 6, 34, 41, 2, 3, 92, 271, 228, 1158, 874, 155,
  760, 161, 1377, 76, 12877, 561, 2654, 48507, 97042, 174104, 78347,
  260851, 993674, 1259337, 2483645, 5740505, 1575587, 3826257,
  21727529, 24850563, 673343, 15828943, 214735647, 338253, 94471517,
  385474364, 26496473, 2080231810, 162912664, 348797635, 117488414,
  10524736889, 12805435028, 19348307706, 8178002329, 25897469511,
  24880839358, 187779182313, 378924045099, 330018976494,
  57572802365, 1755769600244, 1787962449615, 1399589890939,
  4699103264099, 5537088097592, 799491845133, 10875107129731,
  41609657911621, 47938445436267, 6044678688289, 76354192309034,
  20146302444405, 50538003336545, 169286736998843, 140463926976638,
  1372753687833637, 2090937796081262, 3208841539011769,
```

```
  223403826756170, 18890057209817362, 15098281334975233,
  1101146015708157, 47550101433457787, 12050597934415257,
  128657844735176285, 169277061937864378, 330510651947427135,
  340288707202349036, 11709533245952345, 302127549822334661,
  2559809026849192761, 1568191551607991366, 3600013976164019953,
  7680536423553600728, 17111575771294704535, 29807283816584340074,
  53397101317854812084, 16641745710990072136, 1079100819585860413,
  125341458820724802472, 331958594176031667 42,
  ??????????????????????????????????????]
```

The name hints at the Fibonacci-Series that are used as the second argument to `randint` and as part of the dividend to the modulo operation. Taking the pseudo-code verbatim into python generates the series and produces the result.

```
def next_fib(a,b):
    return (b,a+b)

import random
random.seed(1337)

s = [0, 0, 0, 1, 2, 3, 6, ...]
myS = []
a,b = 1,1
for i in range(len(s)):
    if i>=2:
        a,b = next_fib(a,b)
    myS.append(b % random.randint(1, b))

for i in range(len(s)):
    print(s[i], myS[i], s[i]-myS[i])

a,b = next_fib(a,b)
print(b % random.randint(1, b))
```

C06 – C0tt0nt4il Ch3ck required

The difficulty is to read the coloured label because it moves so fast. Use a screen shot tool (see above) to read the label and to find the character following next. Enter it in 1337-notation and we're done.

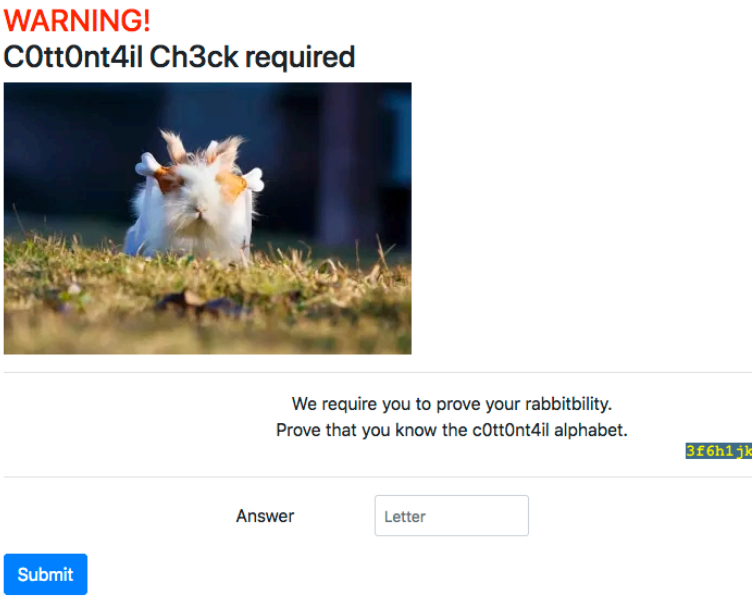
C08 – Sailor John

"Ahoy sailor, my name is John! Can you help me, solving this riddle?"

```
emirpx mod prime = c
p1 = 17635204117, c1 = 419785298 p2 = 1956033275219, c2 = 611096952820
```

Solve it using `https://www.alpertron.com.ar/DILOG.HTM`. Assume that the primes have to be reversed in the decimal notation.

So we find $x_1 = 1647592057 + 4408801029k$ and $x_2 = 305768189495 + 978016637609k$. Unfortunately, the solution as the concatenation of the two x_i was not accepted.



C09 – Ran-Dee's Secret Algorithm

"I just did my daily cryptotraining and found this one..."

Let's use a very small list of primes for RSA style encryption purposes. In fact their list is only the size of the smallest odd prime. One of the robots sent a message to three other robots. These are futuristic robots with the ability to use quantum computing and so they don't mind prime factoring huge numbers. You can't do that though. Find out what message the robot sent to his friends.

```
n0=561335866867161366556651038299444140721943206299883252 33\
      058401869707803703682716186831222740816157923491542101 68\
      307159475914213081021759597948038689876676892007399580 99\
      586826654330987218584372842942643082215621183907 3

c0=487084836230219003844477723778377376298913042405209702 06\
      578416605029721444459236143884842567305359715215194317 99\
      627785375106287039267168740424012477722145444567771983 73\
      054919273770400054114911622267689353043272237214 9

n1=106031991741228398087381693577060627325339667313238588 92\
      743816728206914395320609331466257631096646511986506501 27\
      203600766835807130436415615034513898364863087422048883 76\
      851187535744246862045959815145613432273162973178 99

c1=883895515518702990157008398945175622369348177479273727 66\
      618882238451527834609123122322863356228644312663496028 63\
      379622162995668522612751896796186394681006174093854867 57\
      117996512128227299052476236805574920658456448123

n2=431972268199954142508804890554135853905036810191805947 72\
      781599842207471693041753129885439403306011423063922105 54\
      155765819409217755814518415146092073267565213487633572 28\
      403310081855517062295331798029973666807878660835 23
```

```
c2=2818107200497394993854668960728013273351437664160516949 5\
      7549124283357041180880879789183447419376187061927283699 9\
      2365104786854427689675673204353839263196581517462813454 9\
      5464595656972154988757359459705335058503819578618 3
```

Since we know that there are only three primes in use and we know that n_i is the product of two of them, we can find the primes $p_0 = \sqrt{n_0 * n_1/n_2}$ and cyclic permutations. When we have the primes, we can calculate the secret keys $s_i = \text{lcm}(p_0 - 1, p_1 - 1)$. Now the only unknown is the exponent e, which we assume to be 65537 because it is a popular choice.

The decryption key d_i is calculated with the modular inverse of the exponent e and the secret key s_i. And the message $m_1 = c_1^{d_1} \text{mod} n_1$. Since all messages have to be the same, we can verify that our choice of exponent is correct. Then the message has to be converted into text by translating each byte into a character. So we arrive at the solution

```
RSA3ncrypt!onw!llneverd!e
```

23 - Maze (not solved)

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

`nc whale.hacking-lab.com 7331`

And a binary: `maze`

Solution

Since a binary is given, I first reverse-engineered it with Ghidra. It turns out, it is a simple maze game. The maze is created randomly every time the game is started. There are commands to move in all four directions, to search, to open, and to pick-up stuff.

When a chest is found, it can be opened with the proper key (also random per round) and the key can be picked up when seen. The key itself is a 32-byte value.

Do the old 'left hand on the wall'-trick: start with one assumed direction, check if there are walls north, east, south, west and decide which direction to go: if there is no wall to the left, turn counter clock wise. Otherwise, check if there is a wall ahead and if there is, turn clock wise until there is no wall.

The implementation uses telnetlib for python and is straightforward: do a search to check if there is a key or chest and then move. First we search for the key, if it is found, issue a 'pick up' command. Then continue through the maze until the chest is found, issue an 'open' command and then send the key.

This produces this string in return:

```
Congratulation, you solved the maze. Here is your reward:
X                 *****
X               ****   ****
X             ***         ***
X           ***             ***
X         ***                 ***
X       ***     ****   ****     ***
X     **      ** *** **  **      **
X    **           **   ***.       **
X   **          .***  **  **        **
X  **          ******  ****          **
X **                                   **
X**         +------------------+        **
X*          | +--+ *  *  +--+ |         *
X*          | |  |  ** * |  | |         *
X*          | +--+ ** ** +--+ |         *
X*          |  * **  ** *** * |         *
X*          | * *  ** *** * * |         *
X**         | +--+ * *  [] *  |        **
X *         | |  |  *** ** ** |        *
X **        | +--+ ** *** **  |       **
X  **       +------------------+      **
X   **                             **
X    ***                         ***
X      ***                     ***
X        ****               ****
X           *****       *****
X               *********
XPress enter to return to the menue
```

After spending way too much time to decipher the stars and blanks in the QR-code, I started looking for other vulnerabilities in the binary (at the hint by keep3r). And sure enough, there are two vulnerabilities:

1. when reading the key to open the chest, `fgets` reads up to 40 characters into a buffer that has only room for 32. Right ajacent to the key is a memory area that contains the pointer to the `error` function that is called when an incorrect command is entered. So by entering a 40-byte key we can jump to any function, just by giving an unexpected input.

2. When executing the hidden command `whoami`, the username entered is printed to the console using `printf(&user_name)`, so we have a format string attack.

The plan is to get a shell on the remote computer to search for the egg there. For this, we have to find out, where we should jump to when triggering an error. To do this, we can use the format string attack, to learn the address of a libc-function (the ASLR is switched on on the remote maching), then we can search for a gadget and correct for the ASLR.

Once we know all the address of the gadget, run the maze, get the key, open the chest with the key extended by the 8-byte address of

the gadget, and then trigger an error. Now we should have a root shell.

24 - 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 ;)

`CAPTEGpage`

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

Solution

As a first step I tried to understand the problem and quickly saw that a manual solution is not really feasible :)

So first step: whip up a script that gets us some trainining material by getting 1000 problems. At the same time, the script already chopped the picture up into nine smaller pictures that make up the riddle.

The plan is to find a way to recognize the smaller pictures and add up the eggs. With 9'000 pictures to train with, this should be possible. To match pictures, my algorithm compares the image to be matched with the training set. As a difficulty added, the images are jpegs with lossy compression. So a simple pixel-by-pixel comparison is not possible. I settled for the rms difference as a good measure to judge the match.

Since the images can be rotated, the unknown image is rotated four times by 90 degrees and the rms of these images compared with each image in the library. For a match, the rms will be much lower in the right orientation than in all the others. So use `(max(rms) - min(rms))/max(rms)` as the measure of fit. The threshold I just set to 0.8, without any further thought.

Now we just have to train the library, this was done in an interactive way. Look at all the files, compare against the corpus. If there is a match, continue. If there is no match, present the image to the user and have him enter the number of eggs. Add the image to the corpus with the proper number of eggs. When the corpus is large enough, for me about 540 images, then we start playing the game. This works OK, again have the user enter the number of eggs if no match is found (but for one image per round this is possible). After 42 rounds we get the flag

`he19-s7Jj-mO4C-rP13-ySsJ`

25 - Hidden Egg 1

I like hiding eggs in baskets :)

Solution

The hint points to the section listing the solved eggs: it has an image of a basket. Downloading the image and running it through binwalk shows this line

`/hackyeaster/images/eggs/f8f87dfe67753457dfee34648860dfe786.png`

This is part of the URL and then shows the egg.

26 - Hidden Egg 2

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

Solution

The hint "stylish" pointed me towards CSS-files. Inspecting the .css files using the web-developper tools of Chrome lead me to `source-sans-pro.css` that has the suspicious entry

```
@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');
}
```

Downloading the "font" and renaming it to `egg26.png` gives the egg.