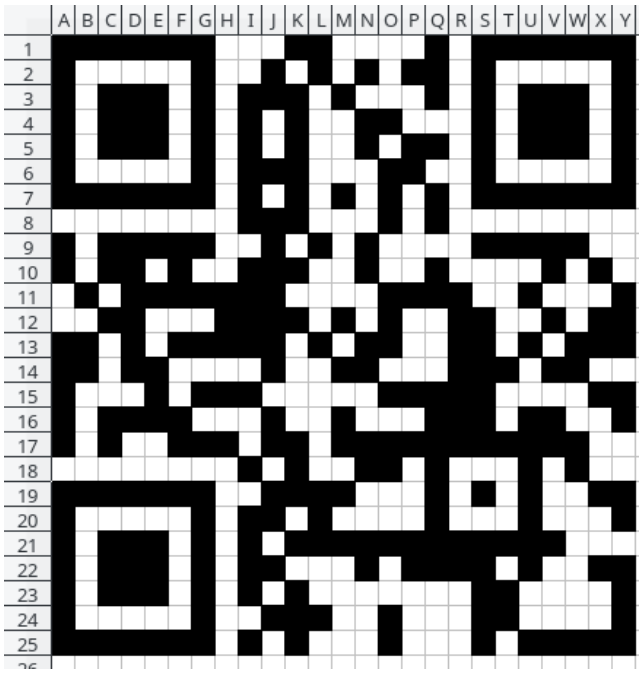
he2019.md

Egg 1

For this egg, I simply copied the QR code to a grid.



Egg 7

After I searched for the two variables `Ax2` and `xTT`, it became clear that the last line means `eval "..."`. Therefore, I replaced the `eval` with `echo` to get the actual script. There, I could just read the parameters `./eggi.sh -R 465 -a 333 -b 911 -I 112 -t 007` and get the flag URL https://hackyeaster.hacking-lab.com/hackyeaster/images/eggs/a61ef3e975acb7d88a127ecd6e156242c74af38c.png.

Egg 9

First of all, the provided file had it's bytes in reverse order, which can be fixed using `< evihcra.piz xxd -p -c1 | tac | xxd -p -r > archive.zip`. The resulting zip file contained a png file with part of its magic number reversed (GNP instead of PNG). In this image, all that was left to do was to mirror it and invert its colors.

Egg 15

The password for this egg is hidden somewhere in the Steem blockchain; therefore, I figured the simplest way to find it was to download the complete block log and grep it. After waiting for 15 hours for the download to complete and another 20 hours for `grep 'Hacky Easter'` (raspberry pi server...) I got (amongst some more lines) `Hacky Easter 2019 takes place between April and May 2019. Take a note: nomoneynobunny`, which is the correct password for the egg.

Egg 18

FIrst, I converted the web assembly code to WAT using wasm2wat. Then, I analyzed the validatePassword function: first, it calls `validateRange`, which checks whether all characters except for the first four are one of `01345HLXcdfr`. Then, the individual characters are checked: the first four have an absolute value, the others are compared to each other (including operations like addition, subtraction, multiplication, modulo, etc.). Using just these checks, I was able to reconstruct almost all of the password: `Th3P4r4d0X0{5|f|r}{m1}H01{m1}{m2}{m3}54{c|L|X}1{m2}`, where `{m*}` are missing characters and `{5|f|r}` and `{c|L|X}` share the same index (if the first one is `5`, the second one is `c` etc.).

The next check is for two things: the values of all but the first four characters are added and compared to 1352, and XORed and compared to 44. Using this information, I brute forced the remaining characters:

```
def chck(a):
    for x in poss:
        for y in poss:
            if (a - 2*x - 2*y) in poss:
                print(x, y, '->', a - 2*x - 2*y)

chck(1352 - sum(chars)) # before that, enter 5|f|r manually and set all missing chars to 0
```

For all possiblities, calculate the character XOR and compare to 44. This yields a few valid combinations, the correct one is `Th3P4r4d0X0fcH01c3154L13`.

Egg 24

To detect eggs in the image, I wrote a python script using OpenCV:

```
#!/usr/bin/python3
import cv2
import numpy as np

def getEggs(im):
    par = cv2.SimpleBlobDetector_Params()
    par.filterByColor = True
    par.blobColor = 0
    par.filterByArea = True
    par.minArea = 100
    par.filterByCircularity = False
    par.filterByConvexity = False
    det = cv2.SimpleBlobDetector_create(par)

    # edge detection, a bit of filtering
    grad = 220
    im2 = cv2.Canny(im, grad, grad)

    kernel = np.array([
        [0, 0, .25, 0, 0],
        [0, 0, 0, 0, 0],
        [.25, 0, 0, 0, .25],
        [0, 0, 0, 0, 0],
        [0, 0, .25, 0, 0],
    ])
    im2 = cv2.filter2D(im2, -1, kernel)
    _, im2 = cv2.threshold(im2, 50, 255, cv2.THRESH_BINARY)

    radius = 17
    kernel = np.ones((radius,radius), np.uint8)
    im3 = cv2.dilate(im2, kernel)

    # blob detection
    keypoints = det.detect(im3)

    im_with_keypoints = cv2.drawKeypoints(im, keypoints, np.array([]), (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW
    cv2.imshow("Keypoints", im_with_keypoints)
```

```
    cv2.waitKey(10)
    return len(keypoints)
```

While this gives pretty good results, it's still too inaccurate. Therefore, I implemented the client as follows:

```
#!/usr/bin/python3
import numpy as np
import cv2
import requests
from captegg import getEggs

s = requests.Session()

while 1:
    # loading website
    print(s.get('http://whale.hacking-lab.com:3555/').text)
    z = s.get('http://whale.hacking-lab.com:3555/picture')
    imarr = np.asarray(bytearray(z.content), dtype=np.uint8)
    im = cv2.imdecode(imarr, cv2.IMREAD_COLOR)

    # counting eggs
    eggs = getEggs(im)

    # correcting error
    err = input()
    if not len(err):
        err = 0
    err = int(err)
    eggs = eggs + err
    print(s.post('http://whale.hacking-lab.com:3555/verify', data={'s':eggs}).text)
```

Since the only important info is how many eggs there are (no egg positions etc.), I use this to manually correct the errors introduced by the simple detection algorithm. Every detected egg is overlayed with a red circle, making it really easy to see eggs without a circle or ones with too many circles (or other objects with circles). Therefore, I only have to count these errors and enter the corresponding number, which is easily doable in seven seconds. After a few attempts, I got the flag `he19-s7Jj-mO4C-rP13-ySsJ`.