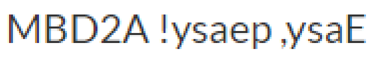
HACKY EASTER 2017 WRITEUP BY HARDLOCK

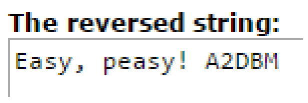
CONTENT

TEASER

PART 01

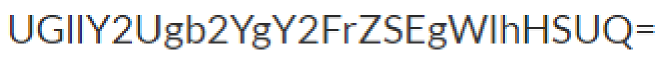

MBD2A !ysaep ,ysaE

this is obviously reverse and with the help of http://string-functions.com/reverse.aspx we can solve this quickly:

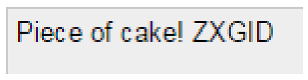

The reversed string:

```
Easy, peasy! A2DBM
```

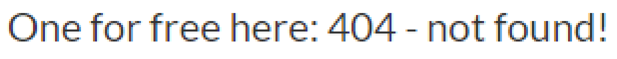
PART 02


UGlIY2Ugb2YgY2FrZSEgWlhHSUQ=

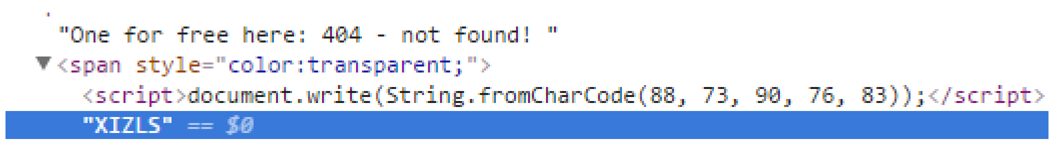
this looks very familiar (because of the = in the end) and is base64.
https://www.base64decode.org/ will help us here:


Piece of cake! ZXGID

PART 03

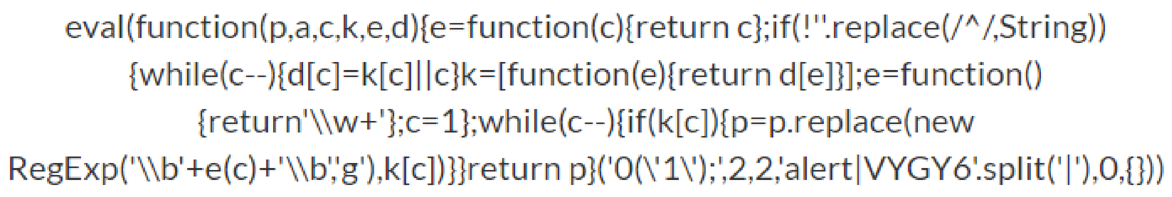

One for free here: 404 - not found!

ok, here we need to look behind the code it seems. using chrome and its developer tools will reveal the solution quickly:

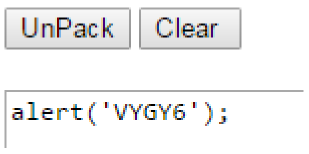

```
  "One for free here: 404 - not found! "
▼<span style="color:transparent;">
    <script>document.write(String.fromCharCode(88, 73, 90, 76, 83));</script>
    "XIZLS" == $0
```

we dont even have to convert it manually, chrome does it for us automatically. didnt know about that feature yet – nice!

PART 04


```
eval(function(p,a,c,k,e,d){e=function(c){return c};if(!''.replace(/^/,String)){while(c--){d[c]=k[c]||c}k=[function(e){return d[e]}];e=function(){return'\\w+'};c=1};while(c--){if(k[c]){p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c])}}return p}('0(\'1\');',2,2,'alert|VYGY6'.split('|'),0,{}))
```

well... this looks complicated, but its not. first i ran this in a .hta file and got the Alert box with the solution. but later i found a site which does clean (unpack) this code for us: http://matthewfl.com/unPacker.html



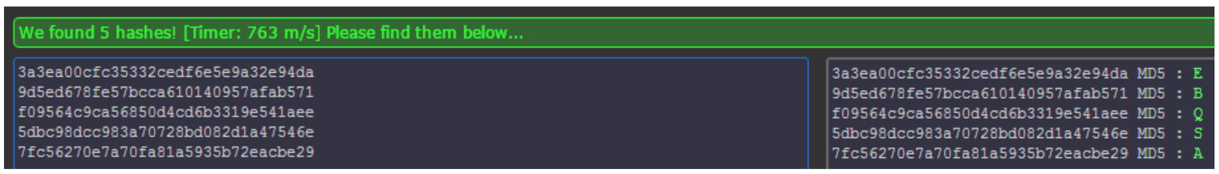
```
alert('VYGY6');
```

but as soon we have the solution, we will also notice that the solution was already cleartext in the challenge. whuuut?

PART 05

3a3ea00cfc35332cedf6e5e9a32e94da
9d5ed678fe57bcca610140957afab571
f09564c9ca56850d4cd6b3319e541aee
5dbc98dcc983a70728bd082d1a47546e
7fc56270e7a70fa81a5935b72eacbe29

this looks nice. i love hex. from the length these strings look like hashes and looking it up in google confirms this: its md5. using https://hashkiller.co.uk/md5-decrypter.aspx we can solve all at once:

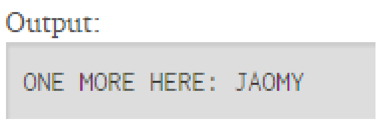


```
We found 5 hashes! [Timer: 763 m/s] Please find them below...

3a3ea00cfc35332cedf6e5e9a32e94da      3a3ea00cfc35332cedf6e5e9a32e94da MD5 : E
9d5ed678fe57bcca610140957afab571      9d5ed678fe57bcca610140957afab571 MD5 : B
f09564c9ca56850d4cd6b3319e541aee      f09564c9ca56850d4cd6b3319e541aee MD5 : Q
5dbc98dcc983a70728bd082d1a47546e      5dbc98dcc983a70728bd082d1a47546e MD5 : S
7fc56270e7a70fa81a5935b72eacbe29      7fc56270e7a70fa81a5935b72eacbe29 MD5 : A
```

PART 06

--- -. . / -- --- .-. . / .... . .-. . ---... / .--- .- --- -- -.--

yeah... looks like morse code. https://morsecode.scphillips.com/translator.html can help us here:



Output:

```
ONE MORE HERE: JAOMY
```

PART 07

Hwldp wx, Euxwh! QYAVL

looks like a substitution cipher. https://planetcalc.com/1434/ will show us all possible solutions and actually only one make sense:

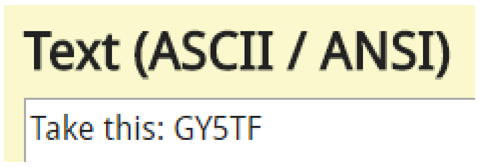
ROT23	Etiam tu, Brute! NVXSI

Etiam tu turns out to be latin. i dont get it, but we just need the solution anyway :-)

PART 08

84 97 107 101 32 116 104 105 115 58 32 71 89 53 84 70

these look like ascii numbers. http://www.asciitohex.com/ is a very good site to solve this (and hey... its even used in Mr. Robot series!)



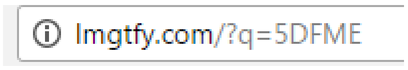
Text (ASCII / ANSI)

Take this: GY5TF

PART 09

Just a bit:
/2mi4AMj

this made me instantly think about bit.ly – which actually contains a redirection to:

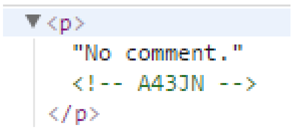


lmgtfy.com/?q=5DFME

PART 10

No comment.

well... im sure there is a comment in the source code then...

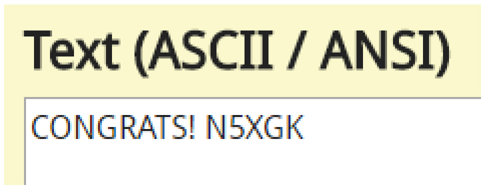


```
▼<p>
   "No comment."
   <!-- A43JN -->
 </p>
```

PART 11

👻👽👻👻👻👻👽👽👻👽👻👻👽👽👽👽👻👽👻👻👽👽👽👻👻👽👻👻👻👽👽👽👻👽
👻👽👻👻👽👻👻👽👻👻👻👻👻👽👻👽👻👽👻👽👻👻👻👽👻👽👻👻👽👽👻👻👽👻
👻👻👻👽👻👻👽👻👻👻👻👻👻👽👻👻👽👽👽👻👻👻👽👽👻👽👻👽👻👽👻👽👽👻
👻👻👻👽👻👻👻👽👽👽👻👽👻👻👽👻👽👽

aliens and ghosts – but we have only two different symbols which makes me think binary. i used ultraedit to search and replace these characters with 1 and 0. then i used http://www.asciitohex.com/ again to convert it:



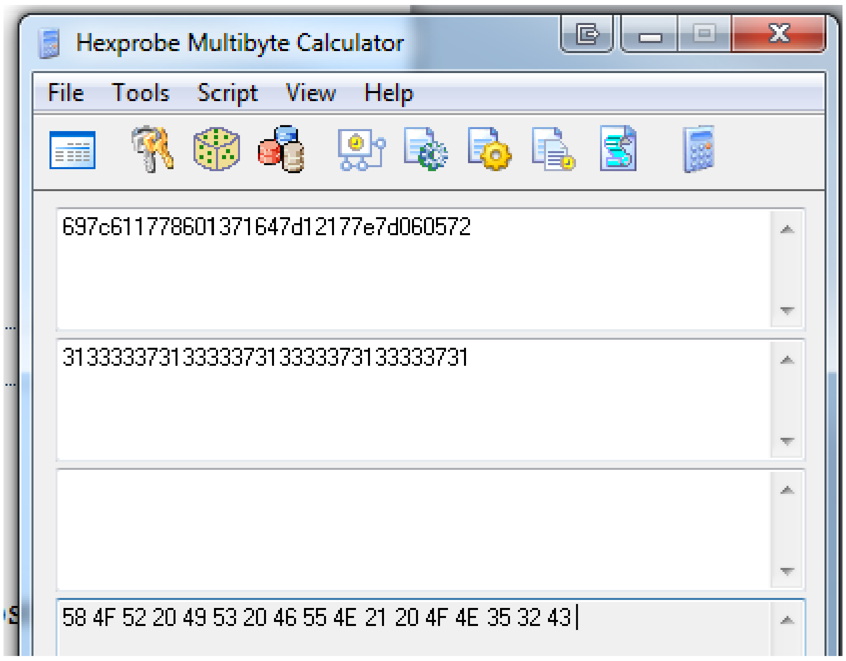
Text (ASCII / ANSI)

CONGRATS! N5XGK

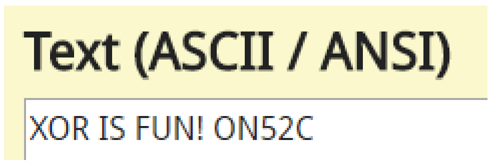
PART 12

697c611778601371647d12177e7d060572
31333337313333373133333731333337 31

two hex numbers without any hint. must be some sort of encryption then and the most simple one is XOR. using Hexprobe Calculator i xored the values and using asciitohex.com again to convert it to ascii.



```
697c611778601371647d12177e7d060572
31333337313333373133333731333337 31
58 4F 52 20 49 53 20 46 55 4E 21 20 4F 4E 35 32 43
```



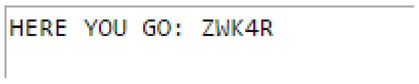
Text (ASCII / ANSI)

XOR IS FUN! ON52C

PART 13

URER LBH TB: MJX4E

i knew this will come... actually this is again caesar but shifted by 13 and therefore called Rot13 (ohwell, it is challenge 13, so this is sortof a hint). using http://www.rot13.com/ we can find the solution:

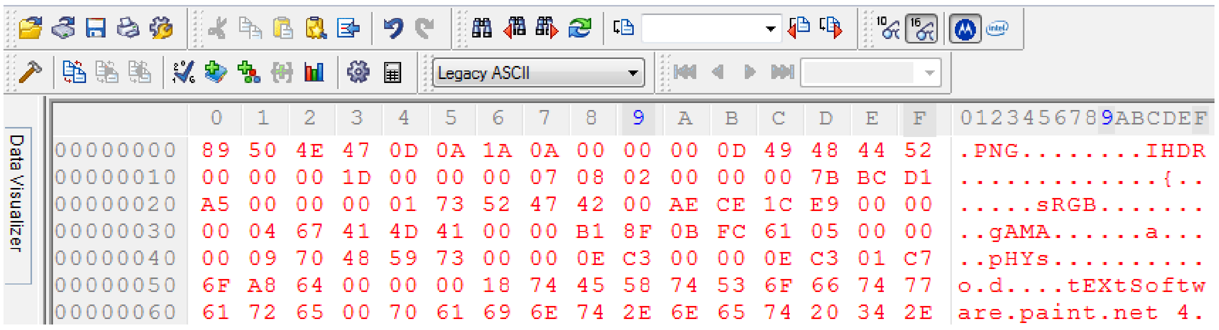


```
HERE YOU GO: ZWK4R
```

PART 14

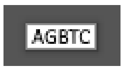
```
89504E470D0A1A0A0000000D49484452000001D000000070802000007BBCD1A500000001735247420 0AECE1CE90
```

nice hex. we should look at this in a hex editor (hex workshop is my tool of choice for that)



```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
00000000 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52  .PNG........IHDR
00000010 00 00 00 1D 00 00 00 07 08 02 00 00 00 7B BC D1  .............{..
00000020 A5 00 00 00 01 73 52 47 42 00 AE CE 1C E9 00 00  .....sRGB.......
00000030 00 04 67 41 4D 41 00 00 B1 8F 0B FC 61 05 00 00  ..gAMA......a...
00000040 00 09 70 48 59 73 00 00 0E C3 00 00 0E C3 01 C7  ..pHYs..........
00000050 6F A8 64 00 00 00 18 74 45 58 74 53 6F 66 74 77  o.d....tEXtSoftw
00000060 61 72 65 00 70 61 69 6E 74 2E 6E 65 74 20 34 2E  are.paint.net 4.
```

obviously a PNG file – an image. lets save it and check it out.



- small, but we got it!

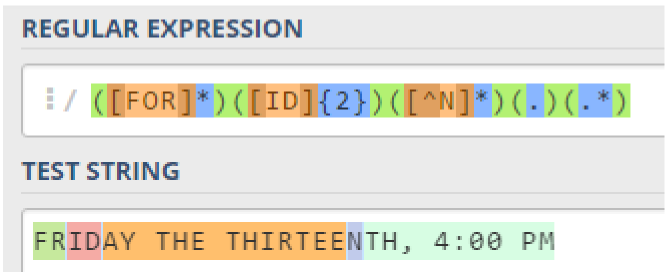
PART 15

FRIDAY THE THIRTEENTH, 4:00 PM

/([FOR]*)([ID]{2})([^N]*)(.)(.*)/g

$2E$44

this one took me some more time, but the /g in the end actually helped a lot. this is a regex expression! first i didnt get what to do with it, but looking at the numbers this means, take the second group, add E, add the 4. group and add 4. using https://regex101.com/ this is easy to solve:



REGULAR EXPRESSION

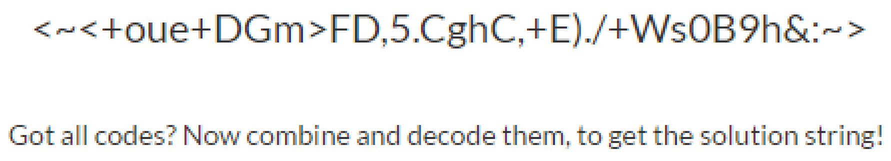
```
/ ([FOR]*)([ID]{2})([^N]*)(.)(.*)
```

TEST STRING

```
FRIDAY THE THIRTEENTH, 4:00 PM
```

we need ID and N. therefore **IDEN4** is the solution!

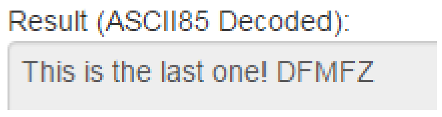
PART 16



```
<~<+oue+DGm>FD,5.CghC,+E)./+Ws0B9h&:~>
```

Got all codes? Now combine and decode them, to get the solution string!

this is base85 and we can use https://www.tools4noobs.com/online_tools/ascii85_decode/ to decode it:



Result (ASCII85 Decoded):

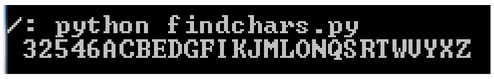
```
This is the last one! DFMFZ
```

FINAL SOLUTION

now we have got all the parts, but we still need to find the solution. these are all parts that we have found from the challenges.

A2DBM ZXGID XIZLS VYGY6 EBQSA JAOMY NVXSI GY5TF 5DFME A43JN N5XGK ON52C ZWK4R AGBTC IDEN4 DFMFZ

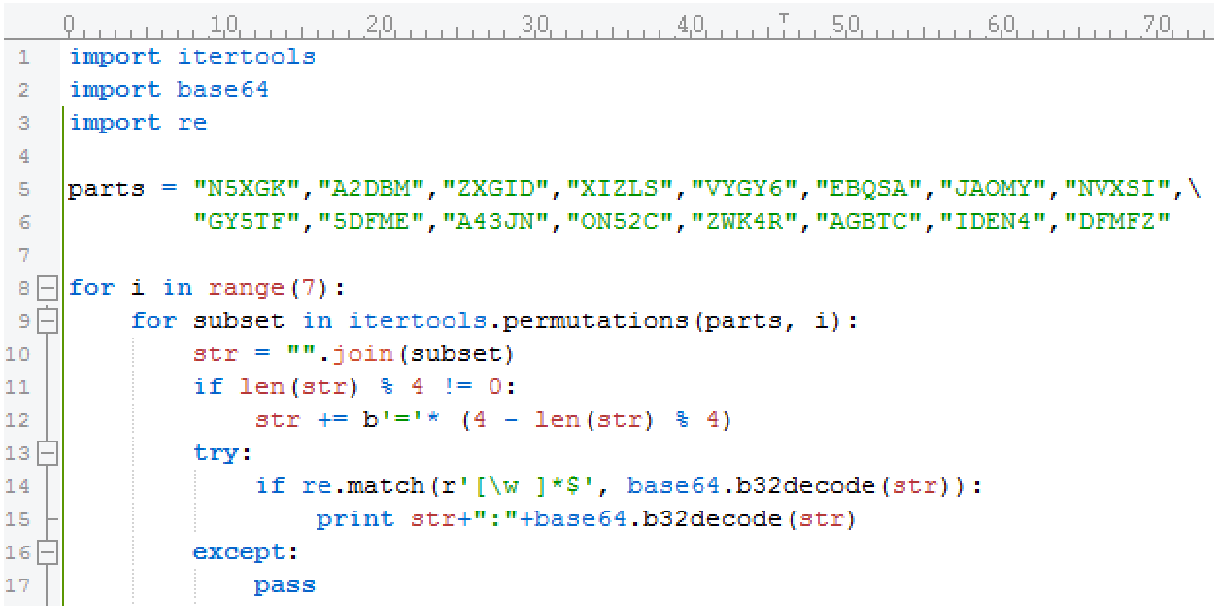
using python i listed all unique characters (print ''.join(set("ALL SOLUTIONS HERE")):



```
/: python findchars.py
 32546ACBEDGFIKJMLONQSRTWUYXZ
```

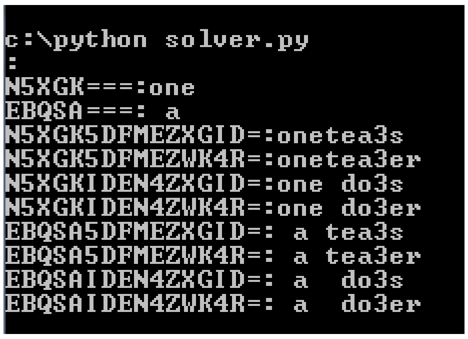
so this looks very suspicious and seems to be base32 –some of the parts do even decode to ascii.

i used python to code a bruteforce solver, which adds padding and then prints the result only when it contains ascii, numbers and/or spaces:



```python
import itertools
import base64
import re

parts = "N5XGK","A2DBM","ZXGID","XIZLS","VYGY6","EBQSA","JAOMY","NVXSI",\
        "GY5TF","5DFME","A43JN","ON52C","ZWK4R","AGBTC","IDEN4","DFMFZ"

for i in range(7):
    for subset in itertools.permutations(parts, i):
        str = "".join(subset)
        if len(str) % 4 != 0:
            str += b'='* (4 - len(str) % 4)
        try:
            if re.match(r'[\w ]*$', base64.b32decode(str)):
                print str+":"+base64.b32decode(str)
        except:
            pass
```

like this i was able to find some valid combinations and work on from there.

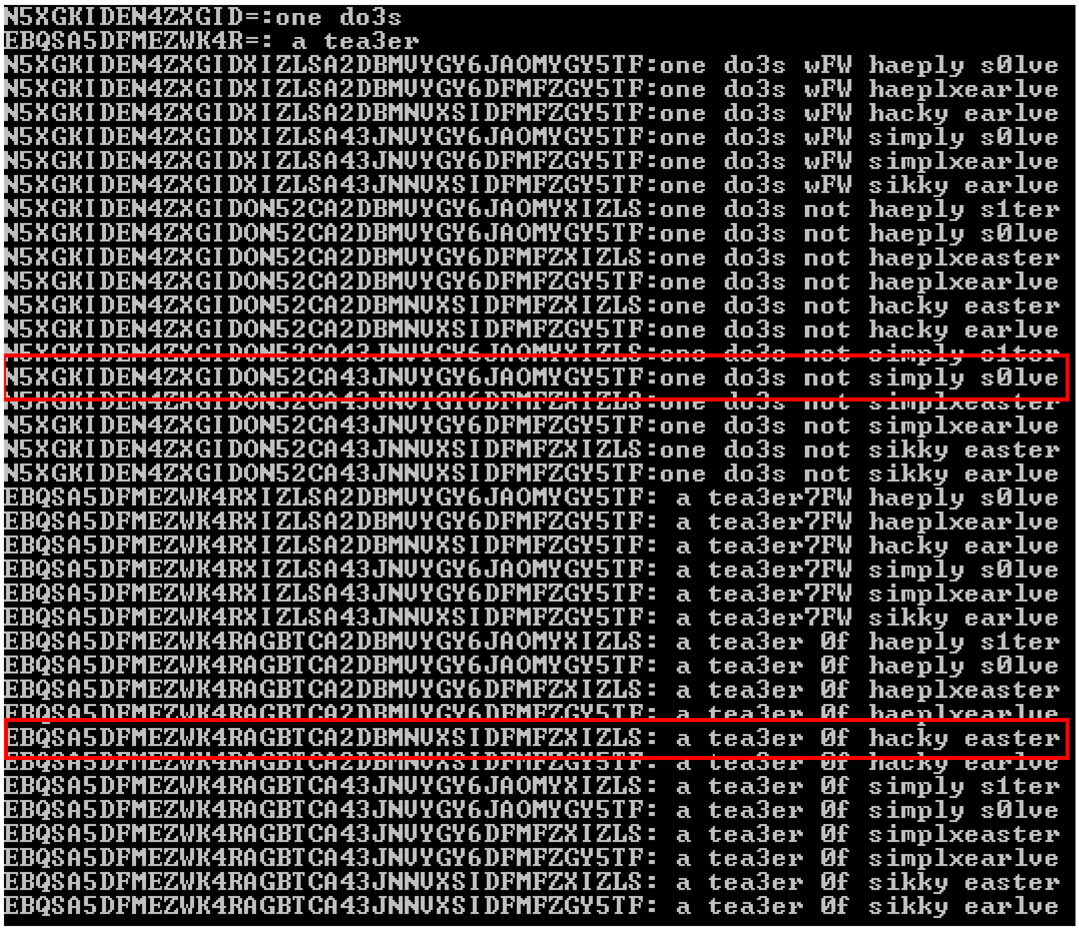


```
c:\python solver.py
:
N5XGK===:one
EBQSA===: a
N5XGK5DFMEZXGID=:onetea3s
N5XGK5DFMEZWK4R=:onetea3er
N5XGKIDEN4ZXGID=:one do3s
N5XGKIDEN4ZWK4R=:one do3er
EBQSA5DFMEZXGID=: a tea3s
EBQSA5DFMEZWK4R=: a tea3er
EBQSAIDEN4ZXGID=: a  do3s
EBQSAIDEN4ZWK4R=: a  do3er
```

"one do3s" and " a tea3er" look promising. to save time i put those already together as parts in my solver and ran it again.

and here we go. the solution is:

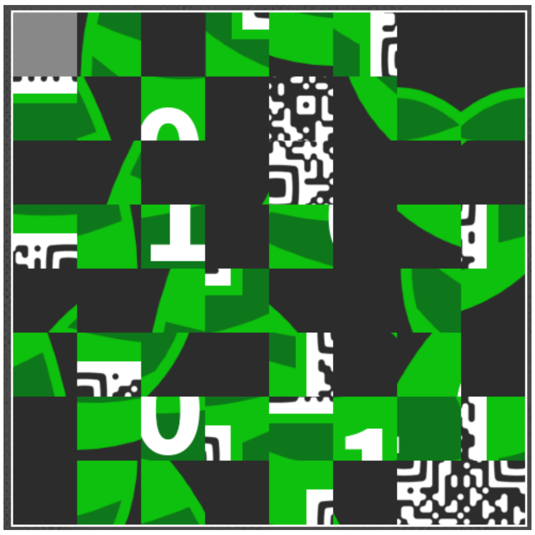
one do3s not simply s0lve a tea3er 0f hacky easter



```
N5XGKIDEN4ZXGID=:one do3s
EBQSA5DFMEZWK4R=: a tea3er
N5XGKIDEN4ZXGIDXIZLSA2DBMVYGY6JAOMYGY5TF:one do3s wFW haeply s0lve
N5XGKIDEN4ZXGIDXIZLSA2DBMVYGY6DFMFZGY5TF:one do3s wFW haeplxearlve
N5XGKIDEN4ZXGIDXIZLSA2DBMNVXSIDFMFZGY5TF:one do3s wFW hacky earlve
N5XGKIDEN4ZXGIDXIZLSA43JNVYGY6JAOMYGY5TF:one do3s wFW simply s0lve
N5XGKIDEN4ZXGIDXIZLSA43JNVYGY6DFMFZGY5TF:one do3s wFW simplxearlve
N5XGKIDEN4ZXGIDXIZLSA43JNNVXSIDFMFZGY5TF:one do3s wFW sikky earlve
N5XGKIDEN4ZXGIDON52CA2DBMVYGY6JAOMYXIZLS:one do3s not haeply s1ter
N5XGKIDEN4ZXGIDON52CA2DBMVYGY6JAOMYGY5TF:one do3s not haeply s0lve
N5XGKIDEN4ZXGIDON52CA2DBMVYGY6DFMFZXIZLS:one do3s not haeplxeaster
N5XGKIDEN4ZXGIDON52CA2DBMVYGY6DFMFZGY5TF:one do3s not haeplxearlve
N5XGKIDEN4ZXGIDON52CA2DBMNVXSIDFMFZXIZLS:one do3s not hacky easter
N5XGKIDEN4ZXGIDON52CA2DBMNVXSIDFMFZGY5TF:one do3s not hacky earlve
N5XGKIDEN4ZXGIDON52CA43JNVYGY6JAOMYXIZLS:one do3s not simply s1ter
N5XGKIDEN4ZXGIDON52CA43JNVYGY6JAOMYGY5TF:one do3s not simply s0lve
N5XGKIDEN4ZXGIDON52CA43JNVYGY6DFMFZXIZLS:one do3s not simplxeaster
N5XGKIDEN4ZXGIDON52CA43JNVYGY6DFMFZGY5TF:one do3s not simplxearlve
N5XGKIDEN4ZXGIDON52CA43JNNVXSIDFMFZXIZLS:one do3s not sikky easter
N5XGKIDEN4ZXGIDON52CA43JNNVXSIDFMFZGY5TF:one do3s not sikky earlve
EBQSA5DFMEZWK4RXIZLSA2DBMVYGY6JAOMYGY5TF: a tea3er7FW haeply s0lve
EBQSA5DFMEZWK4RXIZLSA2DBMVYGY6DFMFZGY5TF: a tea3er7FW haeplxearlve
EBQSA5DFMEZWK4RXIZLSA2DBMNVXSIDFMFZGY5TF: a tea3er7FW hacky earlve
EBQSA5DFMEZWK4RXIZLSA43JNVYGY6JAOMYGY5TF: a tea3er7FW simply s0lve
EBQSA5DFMEZWK4RXIZLSA43JNVYGY6DFMFZGY5TF: a tea3er7FW simplxearlve
EBQSA5DFMEZWK4RXIZLSA43JNNVXSIDFMFZGY5TF: a tea3er7FW sikky earlve
EBQSA5DFMEZWK4RAGBTCA2DBMVYGY6JAOMYXIZLS: a tea3er 0f haeply s1ter
EBQSA5DFMEZWK4RAGBTCA2DBMVYGY6JAOMYGY5TF: a tea3er 0f haeply s0lve
EBQSA5DFMEZWK4RAGBTCA2DBMVYGY6DFMFZXIZLS: a tea3er 0f haeplxeaster
EBQSA5DFMEZWK4RAGBTCA2DBMVYGY6DFMFZGY5TF: a tea3er 0f haeplxearlve
EBQSA5DFMEZWK4RAGBTCA2DBMNVXSIDFMFZXIZLS: a tea3er 0f hacky easter
EBQSA5DFMEZWK4RAGBTCA2DBMNVXSIDFMFZGY5TF: a tea3er 0f hacky earlve
EBQSA5DFMEZWK4RAGBTCA43JNVYGY6JAOMYXIZLS: a tea3er 0f simply s1ter
EBQSA5DFMEZWK4RAGBTCA43JNVYGY6JAOMYGY5TF: a tea3er 0f simply s0lve
EBQSA5DFMEZWK4RAGBTCA43JNVYGY6DFMFZXIZLS: a tea3er 0f simplxeaster
EBQSA5DFMEZWK4RAGBTCA43JNVYGY6DFMFZGY5TF: a tea3er 0f simplxearlve
EBQSA5DFMEZWK4RAGBTCA43JNNVXSIDFMFZXIZLS: a tea3er 0f sikky easter
EBQSA5DFMEZWK4RAGBTCA43JNNVXSIDFMFZGY5TF: a tea3er 0f sikky earlve
```

01 - PUZZLE THIS!

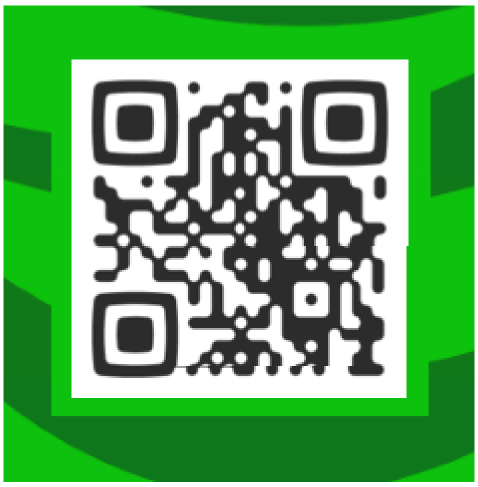
the first challenge is supposed to be an easy one, but i dont know if i just overcomplicated it or maybe i didnt get the idea how to solve it. i mean, yeah... i can obviously solve it by hand – but this takes forever. i was first looking for an automatic solver, because it was based on jqPuzzle, but the original image was already jqPuzzled and therefore this did not work.



so... the fastest way to solve this, was to cut the image from https://hackyeaster.hacking-lab.com/hackyeaster/images/challenge/egg01_shuffled.png into pieces. using imagemagick this was easy:

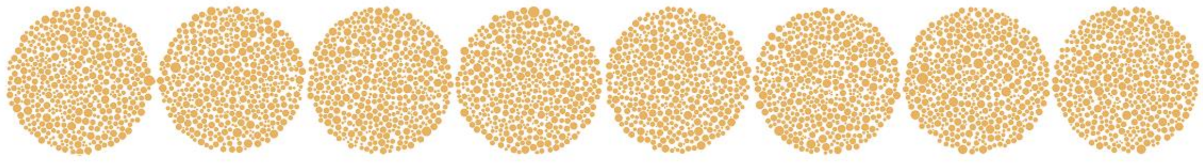
```
convert egg01_shuffled.png -crop 60x60 tiles/tile%03d.png
```

this gave me 64 pieces, which i then loaded into paint.net in different layers. i just focused on the parts with the QR code and deleted all the other layers. after that it was easy to move the parts around and rearrange the QR code:



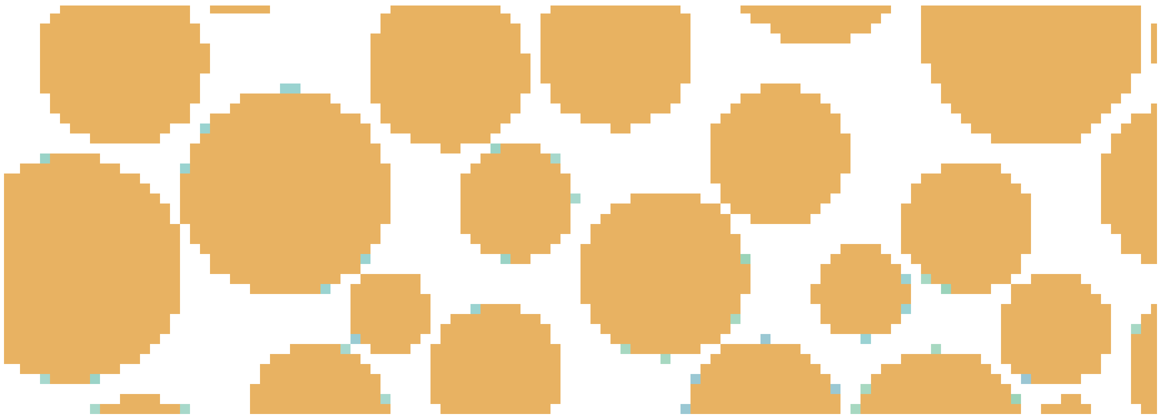
go shuffle yourself easterbunny!

02 - LOTS OF DOTS

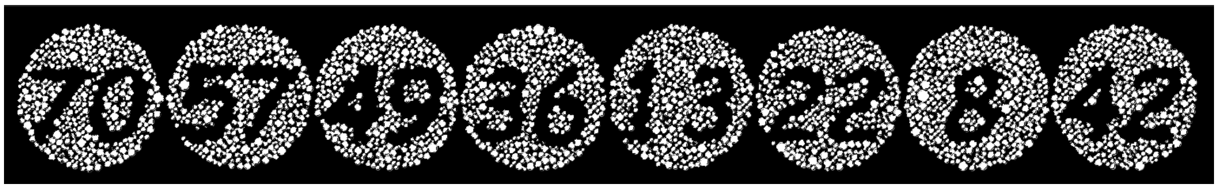


this challenge obviously looks like one of these color blindness tests, but i can only see one color with my eyes. should i visit an optician?

with the help of photoshop, we can actually find a different color in this image, but i cant barely see it.

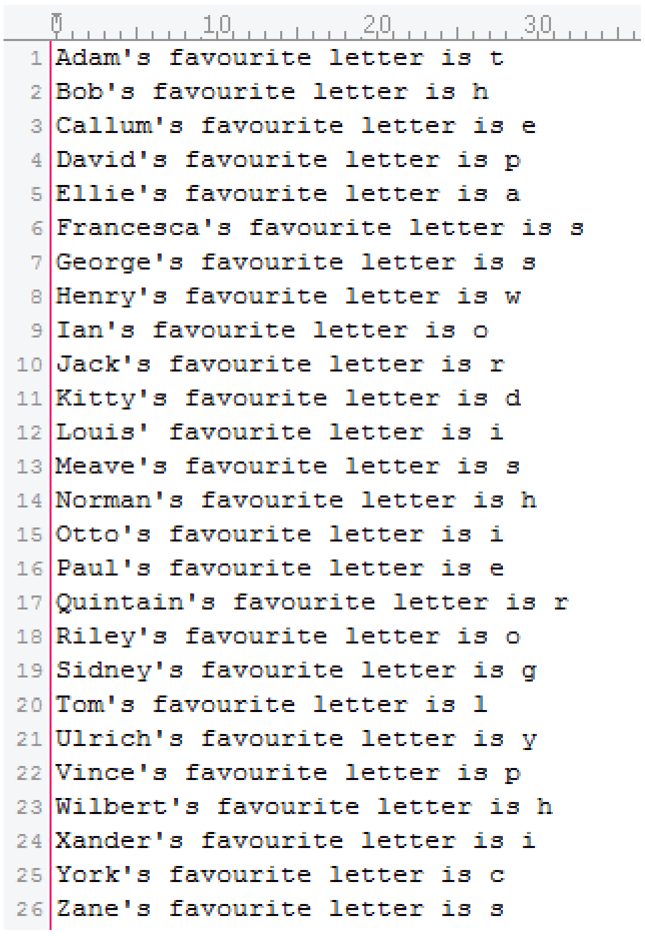


so, still using photoshop, i played with the "select", "color range" function and by setting tolerance to 0 and tweaking other settings, i was able to find the solution.



03 - FAVOURITE LETTERS

in this challenge, we have a list of people and their favorite letters. i noticed rather qickly, that every persons name started with a different letter and therefore i just tried to reorder the people in alphabetic order and then read their favorite letters. this was very easy with ultraedits sort function:



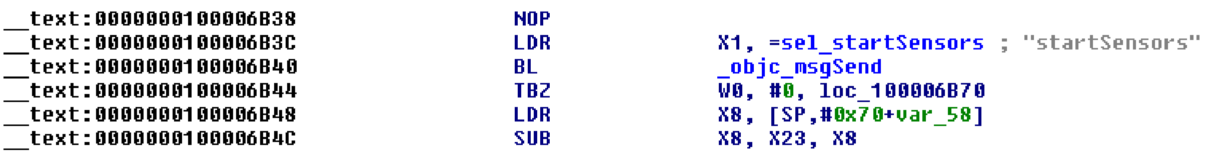
```
Adam's favourite letter is t
Bob's favourite letter is h
Callum's favourite letter is e
David's favourite letter is p
Ellie's favourite letter is a
Francesca's favourite letter is s
George's favourite letter is s
Henry's favourite letter is w
Ian's favourite letter is o
Jack's favourite letter is r
Kitty's favourite letter is d
Louis' favourite letter is i
Meave's favourite letter is s
Norman's favourite letter is h
Otto's favourite letter is i
Paul's favourite letter is e
Quintain's favourite letter is r
Riley's favourite letter is o
Sidney's favourite letter is g
Tom's favourite letter is l
Ulrich's favourite letter is y
Vince's favourite letter is p
Wilbert's favourite letter is h
Xander's favourite letter is i
York's favourite letter is c
Zane's favourite letter is s
```

now we can read from top to down: the password is **hieroglyphics**

04 - COOL CAR

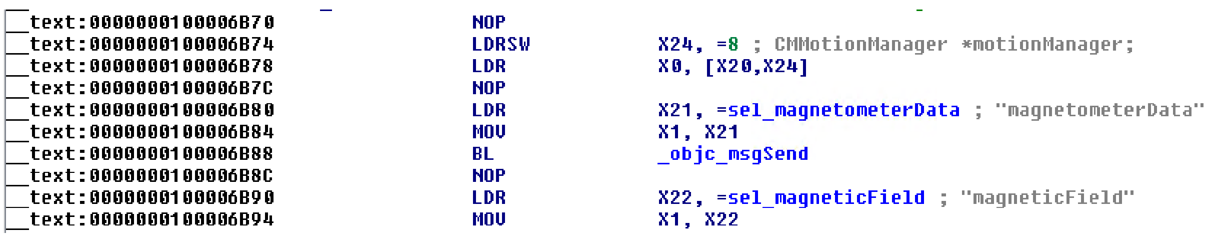
again, an easy challenge which turned out to be hard for me. something i have to do with my mobile phone – but i have really no idea why. by looking at the challenge in the mobile app, it was clear, that it was using some sensors.

i loaded my dumped iOS Hacky Easter App (clutch on jailbroken iphone) in IDA and looked for some sensors.



```
__text:0000000100006B38                 NOP
__text:0000000100006B3C                 LDR             X1, =sel_startSensors ; "startSensors"
__text:0000000100006B40                 BL              _objc_msgSend
__text:0000000100006B44                 TBZ             W0, #0, loc_100006B70
__text:0000000100006B48                 LDR             X8, [SP,#0x70+var_58]
__text:0000000100006B4C                 SUB             X8, X23, X8
```

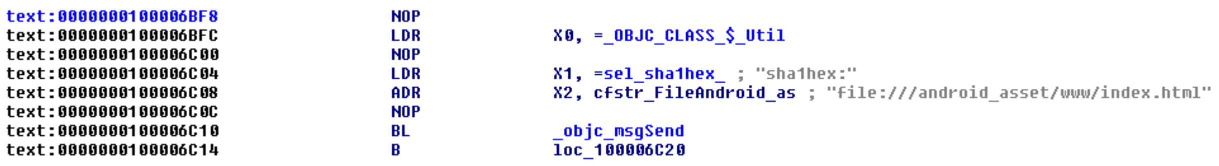
but which sensors are used?



```
__text:0000000100006B70                 NOP
__text:0000000100006B74                 LDRSW           X24, =8 ; CMMotionManager *motionManager;
__text:0000000100006B78                 LDR             X0, [X20,X24]
__text:0000000100006B7C                 NOP
__text:0000000100006B80                 LDR             X21, =sel_magnetometerData ; "magnetometerData"
__text:0000000100006B84                 MOV             X1, X21
__text:0000000100006B88                 BL              _objc_msgSend
__text:0000000100006B8C                 NOP
__text:0000000100006B90                 LDR             X22, =sel_magneticField ; "magneticField"
__text:0000000100006B94                 MOV             X1, X22
```

magneticField? magnetometerData? what the heck is this? im not magneto of x-men!?!

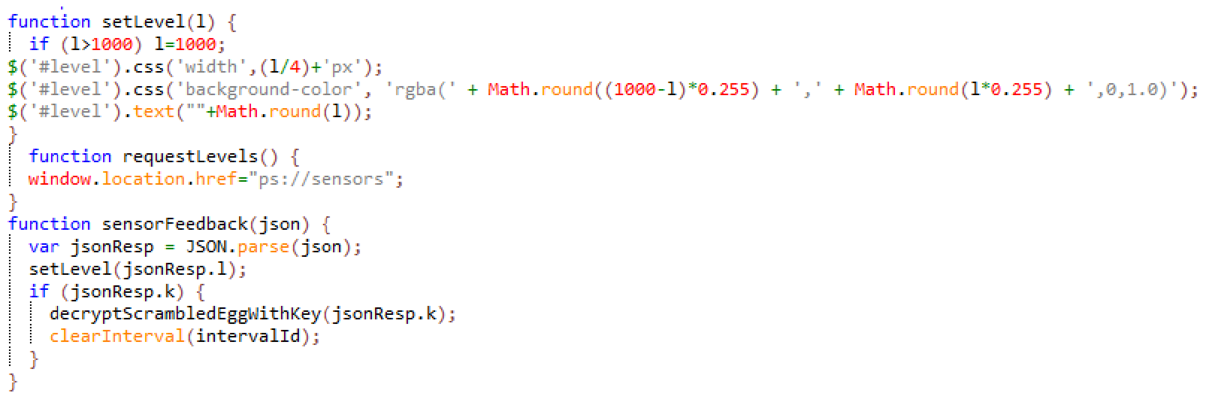
but i am nostradamus and can solve a challenge without knowing how it works. ha ha.



```
text:0000000100006BF8                 NOP
text:0000000100006BFC                 LDR             X0, =_OBJC_CLASS_$_Util
text:0000000100006C00                 NOP
text:0000000100006C04                 LDR             X1, =sel_sha1hex_ ; "sha1hex:"
text:0000000100006C08                 ADR             X2, cfstr_FileAndroid_as ; "file:///android_asset/www/index.html"
text:0000000100006C0C                 NOP
text:0000000100006C10                 BL              _objc_msgSend
text:0000000100006C14                 B               loc_100006C20
```

so... this actually does calculate a SHA1 of the STRING "file:///android_asset/www/index.html" – i mean... this is iOS and this path does not even exist. but now we know the solution of this challenge and the mobile app uses obviously this hash to decrypt the egg,

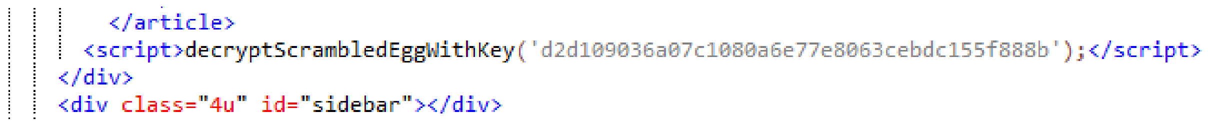
i checked the html files that are in the mobile app (in Hacky Easter.app\www) to display this challenge. there is some interesting javascript code which calls the mobile app sensor code and then uses the response key to decrypt the egg.



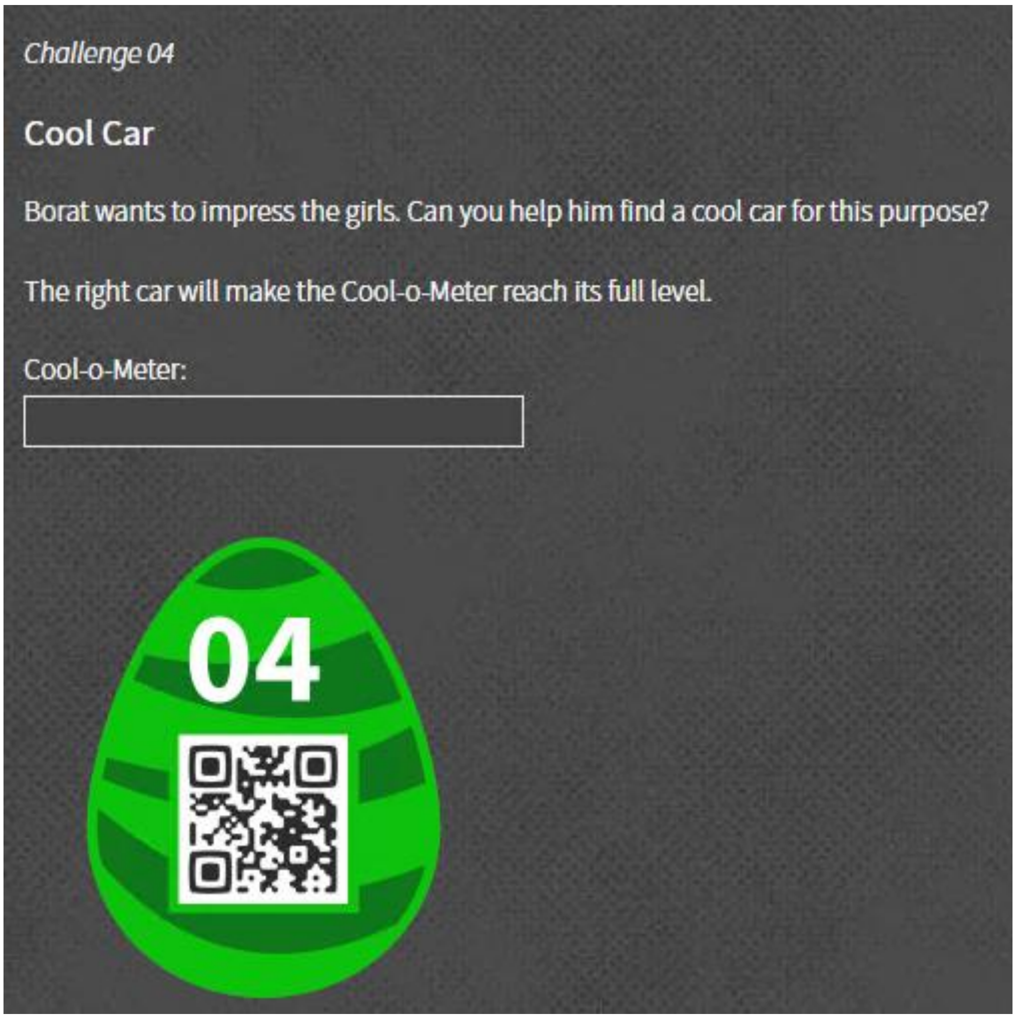
```
function setLevel(l) {
  if (l>1000) l=1000;
$('#level').css('width',(l/4)+'px');
$('#level').css('background-color', 'rgba(' + Math.round((1000-l)*0.255) + ',' + Math.round(l*0.255) + ',0,1.0)');
$('#level').text(""+Math.round(l));
}
  function requestLevels() {
  window.location.href="ps://sensors";
}
function sensorFeedback(json) {
  var jsonResp = JSON.parse(json);
  setLevel(jsonResp.l);
  if (jsonResp.k) {
    decryptScrambledEggWithKey(jsonResp.k);
    clearInterval(intervalId);
  }
}
```

jsonResp.k is what we actually need and from our disassembled code, we know that this is the SHA1 of the fake android path, which is d2d109036a07c1080a6e77e8063cebdc155f888b.

you know what? i just will call decryptScrambledEggWithKey with this hash from within this webpage and load it in my browser on my PC:



```
        </article>
      <script>decryptScrambledEggWithKey('d2d109036a07c1080a6e77e8063cebdc155f888b');</script>
    </div>
    <div class="4u" id="sidebar"></div>
```



Challenge 04

Cool Car

Borat wants to impress the girls. Can you help him find a cool car for this purpose?

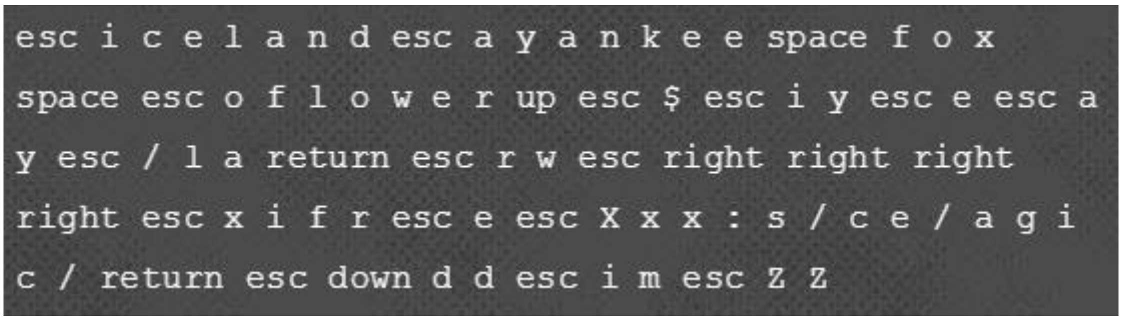
The right car will make the Cool-o-Meter reach its full level.

Cool-o-Meter:

haha. here you go with your magnetometer!

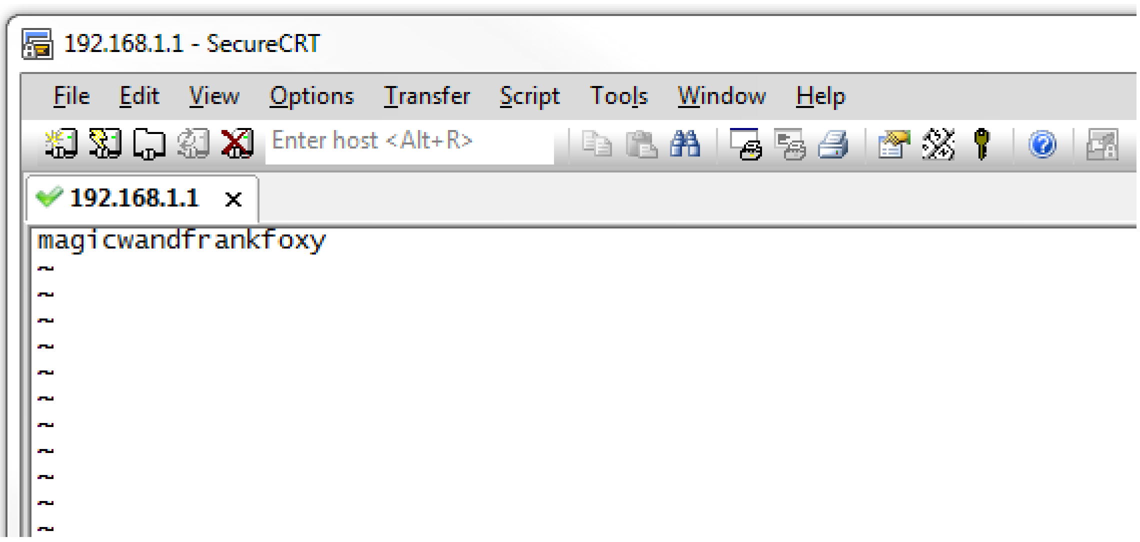
05 - KEY STROKES

we have this bunch of keystrokes for this challenge:



```
esc i c e l a n d esc a y a n k e e space f o x
space esc o f l o w e r up esc $ esc i y esc e esc a
y esc / l a return esc r w esc right right right
right esc x i f r esc e esc X x x : s / c e / a g i
c / return esc down d d esc i m esc Z Z
```

but keystrokes for what or where? the : sign actually reminded me of the vi editor syntax and i gave it a try:

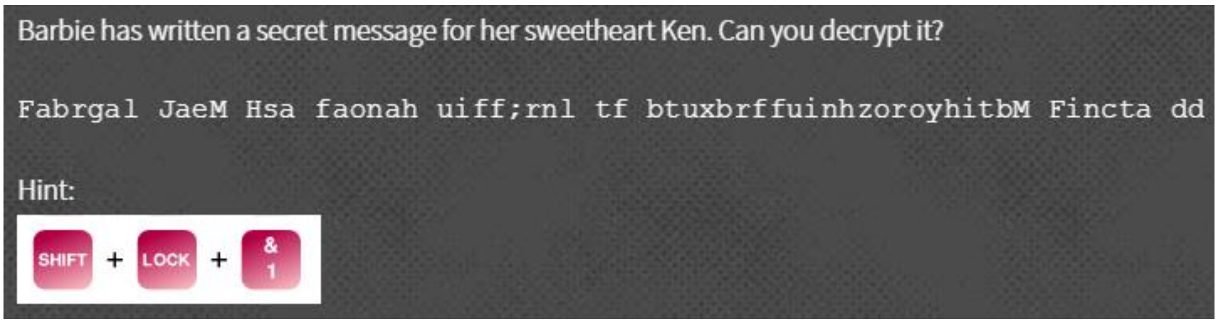


```
magicwandfrankfoxy
~
~
~
~
~
~
~
~
~
~
~
~
```

typing all these keystrokes in an empty vi editor, will give us the solution. we need to replace esc, space, return, up, right and down with the corresponding keys on the keyboard and the rest of the letters we can just type normally.

06 - MESSAGE TO KEN

so barbie sent a message to ken, but we cannot read it at all.


Barbie has written a secret message for her sweetheart Ken. Can you decrypt it?

```
Fabrgal JaeM Hsa faonah uiff;rnl tf btuxbrffuinhzoroyhitbM Fincta dd
```

Hint:

i first suspected a transition cipher, but all normal ones didnt work. i really was out of ideas, until i was surfing on the ipad when already in bed and just for fun i googled "barbie crypto" – this brought up the barbie typewriter

http://www.cryptomuseum.com/crypto/mehano/barbie/

Code	abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789
1	icolapxstvybjeruknfhqg;dzw >FAUTCYOLVJDZINQKSEHG<.1PB 523406789-
2	torbiudfhgzcvanqyepskx¢1w; RC>GHAPND<VUBLIKJETOYXM2QF 63405789-¨
3	hrnctqlpsxwogiekzaufyd+b;¢ SARYO>QIUX<GFDLJVTHNP1Z3KC 7405689-¨§
4	sneohkbufd;rxtaywiqpzl%c¢+ E>SPNRKLG1XYCUDV<HOIQ2B4JA 805679-¨§£

now, this makes sense and also the hint on the page was useful. time to get out of bed again to solve this challenge.

i was just too lazy to code something for this and converted only the obvious flag from the ciphertext (manually using the table above):

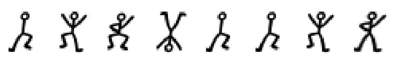
btuxbrffuinhzoroyhitbM

which is

lipglosspartycocktail

07 - CRYPTO FOR ROOKIES

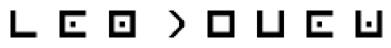
these are all encodings or ciphers and we can solve all of them with online tools. the tricky thing is, that some of these online converters give wrong results.



Qk9OVEVBT0s=

2 15 14 20 5 2 18 11

ONAGROBX



AOBETNOB

ERSWHERN

42 4f 4e 59 45 42 4f 4b

here are the ciphers in the corresponding order:

dancing men cipher: http://www.dcode.fr/dancing-men-cipher
base64: https://www.base64decode.org
letter numbers: http://rumkin.com/tools/cipher/numbers.php
rot13: http://rumkin.com/tools/cipher/rot13.php
pigpen cipher: http://www.dcode.fr/pigpen-cipher
reverse string: http://string-functions.com/reverse.aspx
caesar cipher (23): https://planetcalc.com/1434
hex: http://www.asciitohex.com

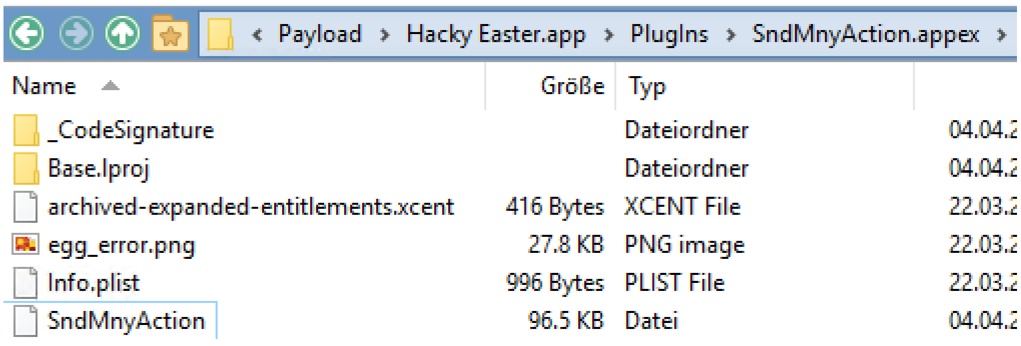
if done correctly, we will get these solutions:

BONTBBOK
BONTEAOK
BONTEBRK
BANTEBOK
CONTEBOK
BONTEBOA
BOPTEBOK
BONYEBOK

by taking every letter that is different on each row (from top to down), we can find the solution CAPYBARA

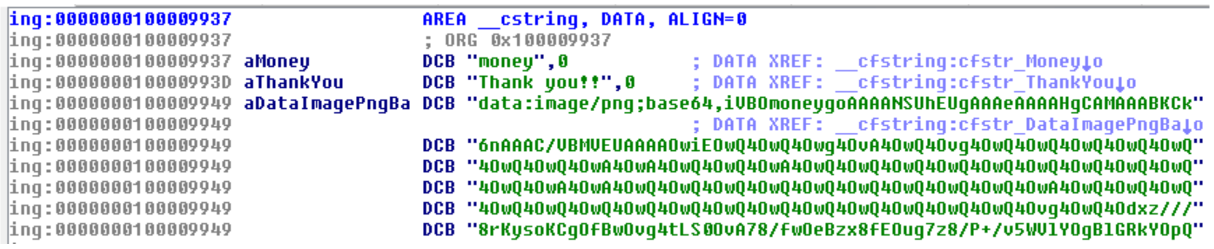
08 - SND MNY

i really dont have an idea, what this challenge was about. but like always, i dumped the iOS binary from my jailbroken iphone with clutch and inspected the binaries from the app. under the Plugins folder, i found this interesting binary:



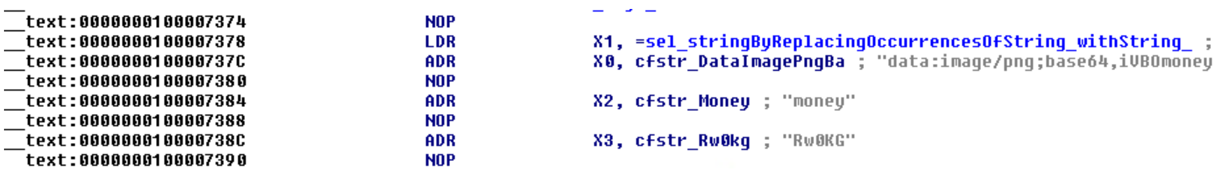
Name	Größe	Typ	
_CodeSignature		Dateiordner	04.04.2
Base.lproj		Dateiordner	04.04.2
archived-expanded-entitlements.xcent	416 Bytes	XCENT File	22.03.2
egg_error.png	27.8 KB	PNG image	22.03.2
Info.plist	996 Bytes	PLIST File	22.03.2
SndMnyAction	96.5 KB	Datei	04.04.2

looking at this in IDA made me find something interesting pretty quickly:



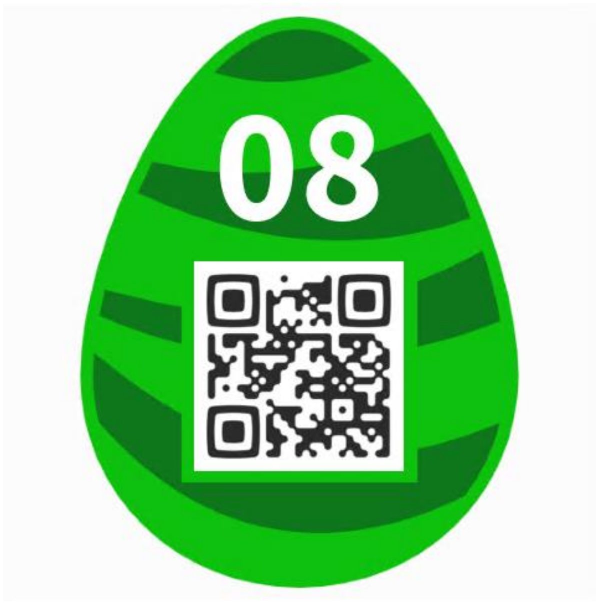
```
ing:0000000100009937                 AREA __cstring, DATA, ALIGN=0
ing:0000000100009937                 ; ORG 0x100009937
ing:0000000100009937 aMoney          DCB "money",0           ; DATA XREF: __cfstring:cfstr_Money↓o
ing:000000010000993D aThankYou       DCB "Thank you!!",0     ; DATA XREF: __cfstring:cfstr_ThankYou↓o
ing:0000000100009949 aDataImagePngBa DCB "data:image/png;base64,iVBOmoneygoAAAANSUhEUgAAAeAAAAHgCAMAAABKCk"
ing:0000000100009949                                         ; DATA XREF: __cfstring:cfstr_DataImagePngBa↓o
ing:0000000100009949                 DCB "6nAAAC/VBMVEUAAAA0wiE0wQ40wQ40wg40vA40wQ40vg40wQ40wQ40wQ40wQ40wQ"
ing:0000000100009949                 DCB "40wQ40wQ40wA40wA40wQ40wQ40wA40wQ40wQ40wQ40wQ40wQ40wQ40wQ40wQ"
ing:0000000100009949                 DCB "40wQ40wA40wA40wQ40wQ40wQ40wQ40wQ40wQ40wQ40wQ40wQ40wA40wQ40wQ40wQ"
ing:0000000100009949                 DCB "40wQ40wQ40wQ40wQ40wQ40wQ40wQ40wQ40wQ40wQ40wQ40wQ40vg40wQ40dxz///"
ing:0000000100009949                 DCB "8rKysoKCg0fBw0vg4tLS00vA78/fw0eBzx8fE0ug7z8/P+/v5WVlY0gBlGRkY0pQ"
```

of course i dumped this base64 string and tried to convert it, but it failed. by looking at the assembly code where this string was referenced, it was obvious what we have to do:



```
__text:0000000100007374                 NOP
__text:0000000100007378                 LDR             X1, =sel_stringByReplacingOccurrencesOfString_withString_ ;
__text:000000010000737C                 ADR             X0, cfstr_DataImagePngBa ; "data:image/png;base64,iVBOmoney
__text:0000000100007380                 NOP
__text:0000000100007384                 ADR             X2, cfstr_Money ; "money"
__text:0000000100007388                 NOP
__text:000000010000738C                 ADR             X3, cfstr_Rw0kg ; "Rw0KG"
  text:0000000100007390                 NOP
```

there is "money" in the base64, which should be "Rw0KG" – fixing this made the base64 correct and i was able to decode it into a PNG file.

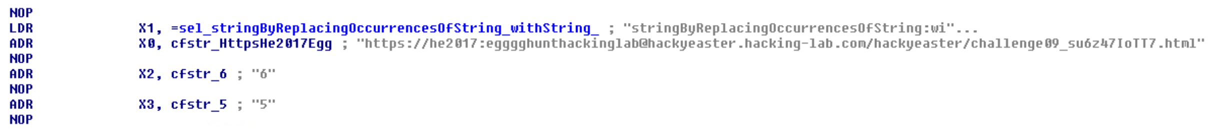


09 - MICROSCOPE

again a challenge, that i probably solved the "hard" way – instead of the easy one. when going through the disassembly of the Hacky Easter iOS Mobile App, i found references to the microscope challenge.

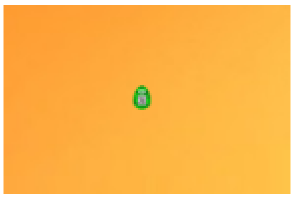
```
; MicroscopeViewController - (void)viewDidLoad

; void __cdecl -[MicroscopeViewController viewDidLoad](struct MicroscopeViewController *self, SEL)
__MicroscopeViewController_viewDidLoad_ ; DATA XREF: __objc_const:00000001000CBC98↓o
```

it turned out, that the mobile app loads an url from the hackyeaster server to display the image that we should use under a microscope.



```
NOP
LDR             X1, =sel_stringByReplacingOccurrencesOfString_withString_ ; "stringByReplacingOccurrencesOfString:wi"...
ADR             X0, cfstr_HttpsHe2017Egg ; "https://he2017:eggghunthackinglab@hackyeaster.hacking-lab.com/hackyeaster/challenge09_su6z47IoTT7.html"
NOP
ADR             X2, cfstr_6 ; "6"
NOP
ADR             X3, cfstr_5 ; "5"
NOP
```

the URL https://he2017:egggghunthackinglab@hackyeaster.hacking-lab.com/hackyeaster/challenge09_su6z47IoTT7.html is loaded, but trying this directly does not work. the code above shows us why – we need to replace "6" with "5" and doing this, lets us load the image.



its a little bit too small to scan, but we can just drag and drop it to the address bar of the browser, to see its full size:

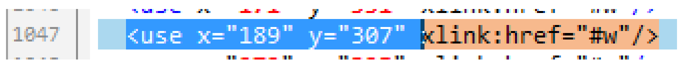
https://hackyeaster.hacking-lab.com/hackyeaster/images/challenge/egg09_fs0sYle2SN.png

10 - AN EGG OR NOT ...

oh yeah – this challenge gives us the egg straight away. but wait – of course this is NOT the correct egg. but how should we make a good egg out of this one?

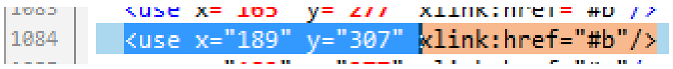
looking at the file itself in an editor like ultraedit, shows us, that the QR code is pained with coordinates in XML style. i dont know much about SVG file format, but it was logical, that the real egg must have been replaced by fake pixels using this coordinates. i checked my theory quickly with the ultraedit search function and indeed i found duplicates of coordinates:

here a white pixel is painted on line 1047:



```
1047    <use x="189" y="307" xlink:href="#w"/>
```

and here with the same coordinates a black pixel is painted on line 1084:



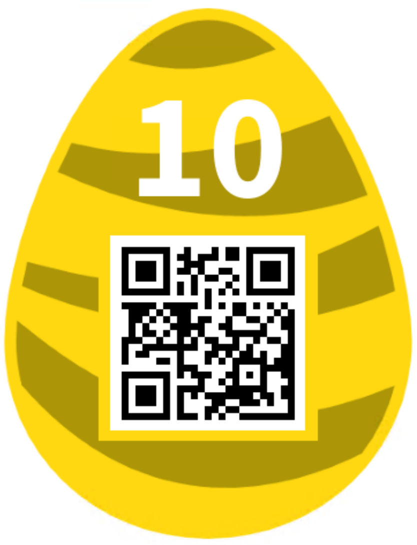
```
1084    <use x="189" y="307" xlink:href="#b"/>
```

this means, the good egg is being overwritten by the fake egg.

the easiest way to solve this is probably to turn around every pixel from white to black or the other way around, when its coordinates are being used twice. actually, this challenge could have been made more complicated by adding more than two references to the same coordinates, but like this it was rather easy to apply my logic.

```
f = open('aneggornot.svg','rb')
contents = f.read()

with open('aneggornot.svg','rb') as fp:
  for line in fp:
    coordinates=line.strip()[:20]
    if contents.count(coordinates)==2:
      l=line.strip()
      if '#w' in l:
        l=line.strip().replace('#w','#b')
      else:
        l=line.strip().replace('#b','#w')
      contents=contents.replace(l,'')

with open('solution.svg', 'wb') as s:
  s.write(contents)
```

my code simply checks if a line is available twice in the file and then it flips around its color and deletes the flipped line – which is in this case always the wrong one. the result i write into a new file and voila:



this is by far not the best way to solve this challenge, but it was easy and fast. nice one btw. great idea!

11 - TWEAKED TWEET

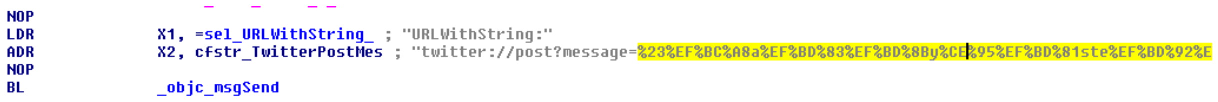
this one made me scratch my head for quite some time. but this is because i was expecting something completely different and the title is somehow misleading.

however... it turned out, to be some sort of steganography for twitter or in other words, secret messages on twitter.

if we google these terms, we will find a webpage which offers us exactly this.

http://holloway.co.nz/steg/

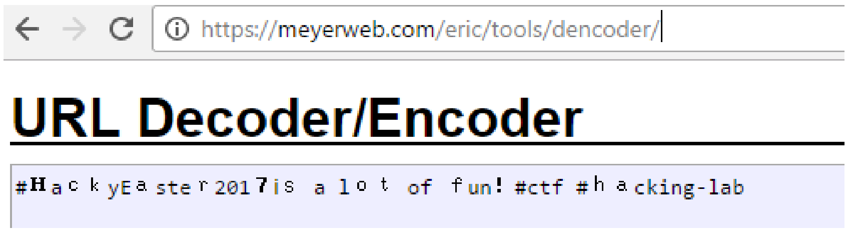
i grabbed the twitter http post data from the IDA disassembly of the Hacky Easter App:



```
NOP
LDR             X1, =sel_URLWithString_ ; "URLWithString:"
ADR             X2, cfstr_TwitterPostMes ; "twitter://post?message=%23%EF%BC%A8a%EF%BD%83%EF%BD%8By%CE%95%EF%BD%81ste%EF%BD%92%E
NOP
BL              _objc_msgSend
```

%23%EF%BC%A8a%EF%BD%83%EF%BD%8By%CE%95%EF%BD%81ste%EF%BD%92%E2%80%A9201%EF%BC%97%E2%80%A9%E2%85%B0%EF%BD%93%E2%80%80a%E2%80%84l%EF%BD%8F%EF%BD%94%E2%80%80%CE%BFf%E2%80%89%EF%BD%86un%EF%BC%81%E2%80%A8%23%D1%81tf%E2%80%88%23%EF%BD%88%EF%BD%81%CF%B2king-lab

this is obviously URLEncoded. with the help of an online decoder i converted it:

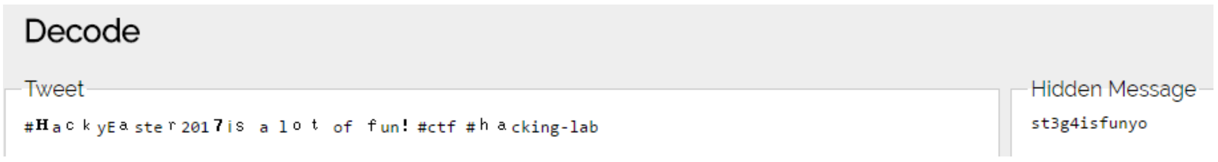


https://meyerweb.com/eric/tools/dencoder/

URL Decoder/Encoder

```
#Ｈａｃｋ yΕａ steｒ 201７ⅰｓ a l o t of ｆun！ #сtf #ｈ ａ ϲking-lab
```

and using the Twitter Secret Message decoder from the page above, i found the solution "st3g4isfunyo"



Decode

Tweet

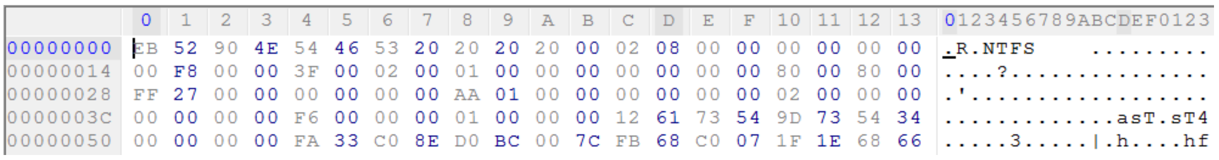
```
#Ｈａｃｋ yΕａ steｒ 201７ⅰｓ a l o t of ｆun！ #сtf #ｈ ａ ϲking-lab
```

Hidden Message

```
st3g4isfunyo
```

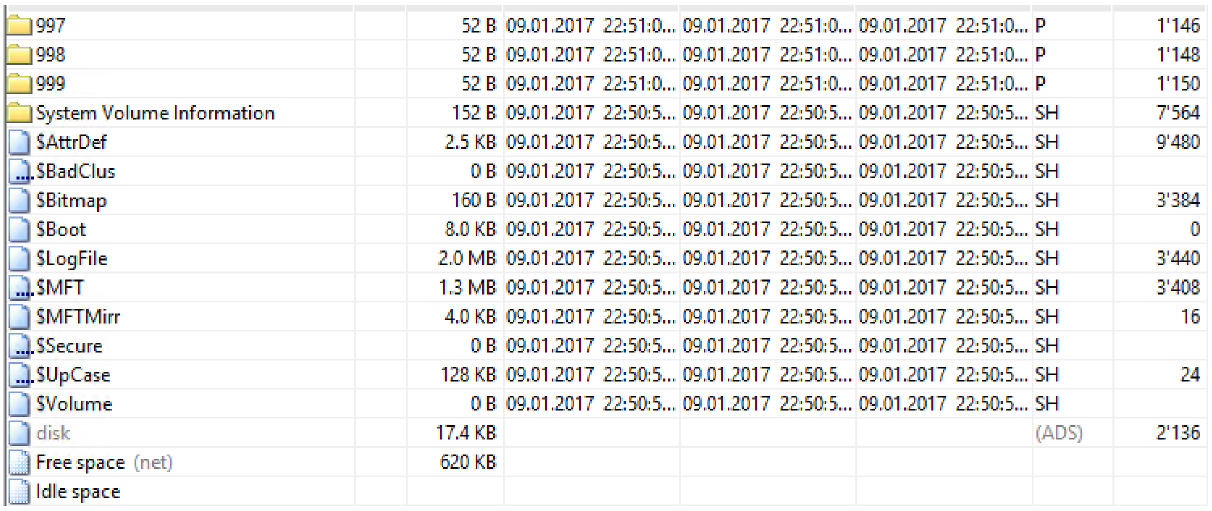
12 - ONCE UPON A FILE

the zipfile here contains another file, which seems to be an image of a disk.



```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  10 11 12 13  0123456789ABCDEF0123
00000000  EB 52 90 4E 54 46 53 20 20 20 20 00 02 08 00 00 00 00 00 00  _R.NTFS     .........
00000014  00 F8 00 00 3F 00 02 00 01 00 00 00 00 00 00 00 80 00 80 00  .....?..............
00000028  FF 27 00 00 00 00 00 00 AA 01 00 00 00 00 00 00 02 00 00 00  .'..................
0000003C  00 00 00 00 F6 00 00 00 01 00 00 00 12 61 73 54 9D 73 54 34  ..............asT.sT4
00000050  00 00 00 00 FA 33 C0 8E D0 BC 00 7C FB 68 C0 07 1F 1E 68 66  ......3.....|.h....hf
```

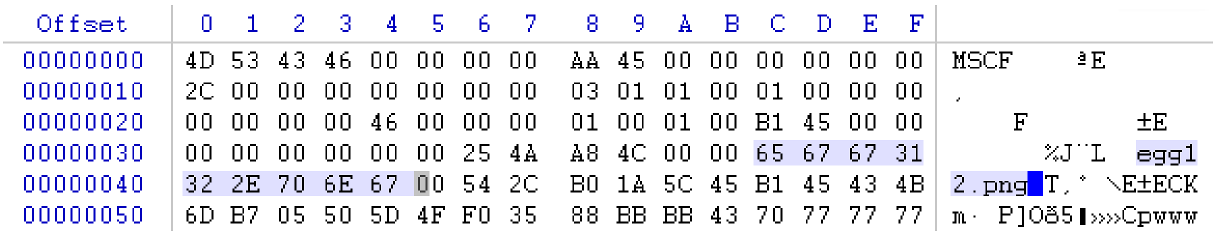
its time for winhex – the best tool for such forensic tasks. there we can select, interpret image as filedisk and like this we can browse the filesystem very easy.



997		52 B	09.01.2017 22:51:0...	09.01.2017 22:51:0...	09.01.2017 22:51:0...	P	1'146
998		52 B	09.01.2017 22:51:0...	09.01.2017 22:51:0...	09.01.2017 22:51:0...	P	1'148
999		52 B	09.01.2017 22:51:0...	09.01.2017 22:51:0...	09.01.2017 22:51:0...	P	1'150
System Volume Information		152 B	09.01.2017 22:50:5...	09.01.2017 22:50:5...	09.01.2017 22:50:5...	SH	7'564
$AttrDef		2.5 KB	09.01.2017 22:50:5...	09.01.2017 22:50:5...	09.01.2017 22:50:5...	SH	9'480
$BadClus		0 B	09.01.2017 22:50:5...	09.01.2017 22:50:5...	09.01.2017 22:50:5...	SH	
$Bitmap		160 B	09.01.2017 22:50:5...	09.01.2017 22:50:5...	09.01.2017 22:50:5...	SH	3'384
$Boot		8.0 KB	09.01.2017 22:50:5...	09.01.2017 22:50:5...	09.01.2017 22:50:5...	SH	0
$LogFile		2.0 MB	09.01.2017 22:50:5...	09.01.2017 22:50:5...	09.01.2017 22:50:5...	SH	3'440
$MFT		1.3 MB	09.01.2017 22:50:5...	09.01.2017 22:50:5...	09.01.2017 22:50:5...	SH	3'408
$MFTMirr		4.0 KB	09.01.2017 22:50:5...	09.01.2017 22:50:5...	09.01.2017 22:50:5...	SH	16
$Secure		0 B	09.01.2017 22:50:5...	09.01.2017 22:50:5...	09.01.2017 22:50:5...	SH	
$UpCase		128 KB	09.01.2017 22:50:5...	09.01.2017 22:50:5...	09.01.2017 22:50:5...	SH	24
$Volume		0 B	09.01.2017 22:50:5...	09.01.2017 22:50:5...	09.01.2017 22:50:5...	SH	
disk		17.4 KB				(ADS)	2'136
Free space (net)		620 KB					
Idle space							

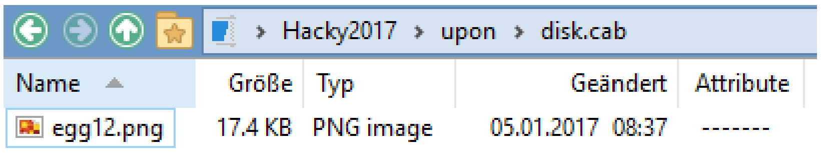
i noticed another file called disk, but it was deleted on this image, therefore it has this gray font.

looking at this "hidden" file, we can already see some information about the egg12 and also that this is a microsoft cabinet file (MSCF header) – lets save this as .cab then!



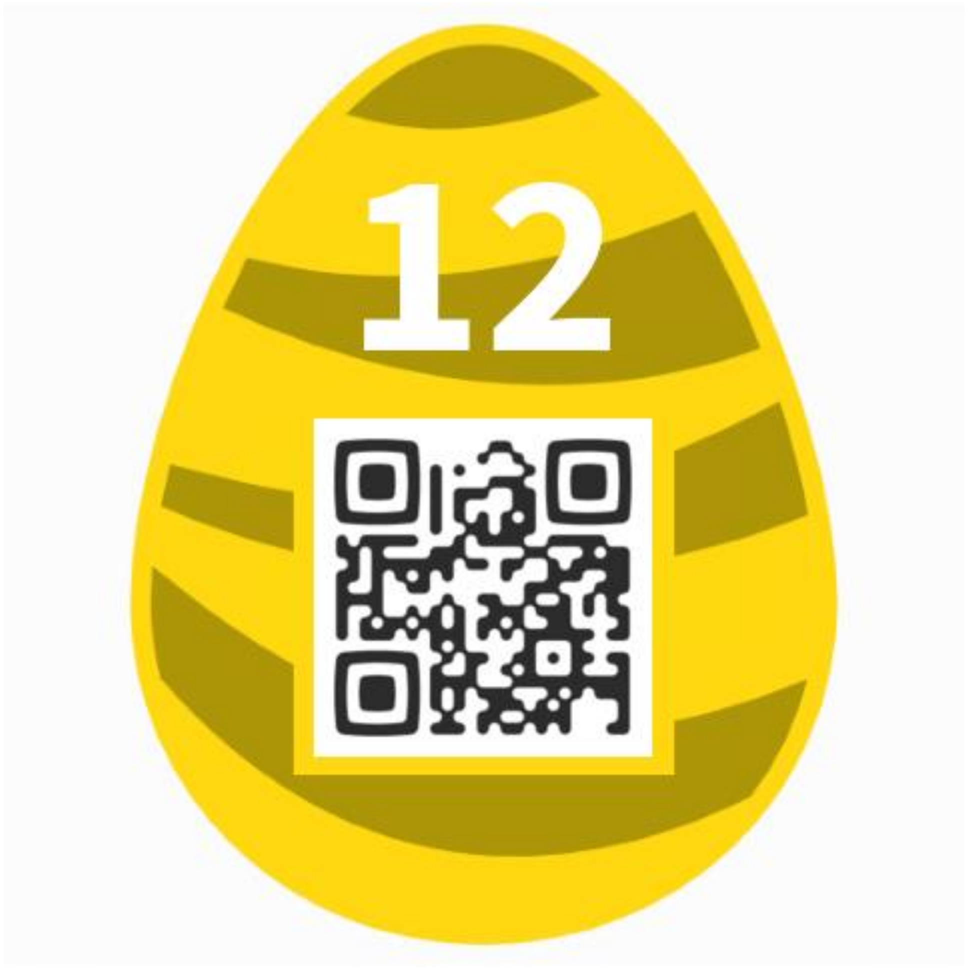
```
 Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000  4D 53 43 46 00 00 00 00  AA 45 00 00 00 00 00 00  MSCF    ªE
00000010  2C 00 00 00 00 00 00 00  03 01 01 00 01 00 00 00  ,
00000020  00 00 00 00 46 00 00 00  01 00 01 00 B1 45 00 00      F       ±E
00000030  00 00 00 00 00 00 25 4A  A8 4C 00 00 65 67 67 31        %J¨L  egg1
00000040  32 2E 70 6E 67 00 54 2C  B0 1A 5C 45 B1 45 43 4B  2.png T,° \E±ECK
00000050  6D B7 05 50 5D 4F F0 35  88 BB BB 43 70 77 77 77  m· P]Oð5 »»Cpwww
```

opening this cab file in normal explorer revealed the egg for this challenge:



Hacky2017 › upon › disk.cab

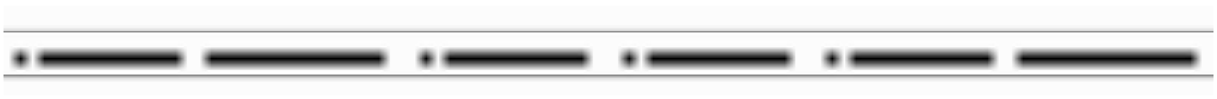
Name	Größe	Typ	Geändert	Attribute
egg12.png	17.4 KB	PNG image	05.01.2017 08:37	-------



13 - LOST THE THREAD

this challenge gives us an image, that looks like morse code – but obviously is not. actually i found the solution rather quickly, but didnt notice that it was the solution. selfpwn.

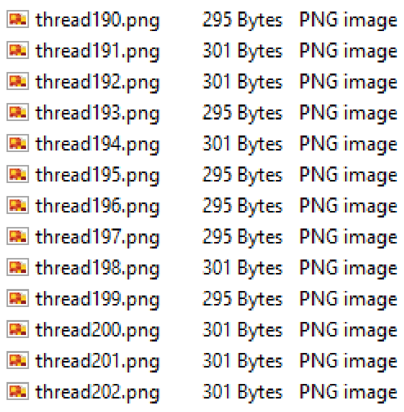
by analyzing the image, i noticed, that there are only two types of patterns available. one dash and a point-dash:



this can be interpreted as binary of course. i used python and PIL to read all pixels and output a binary string from it. funny thing is, this image was using the alpha channel to display the lines. every 17 pixel, a new dash started and therefore i used imagemagick to crop the image in 17 pixel sized pieces.

convert thread.png -crop 17x3 thread/thread%03d.png

this gave me 853 images and by looking at the sizes, it confirmed my theory about only having two patterns:



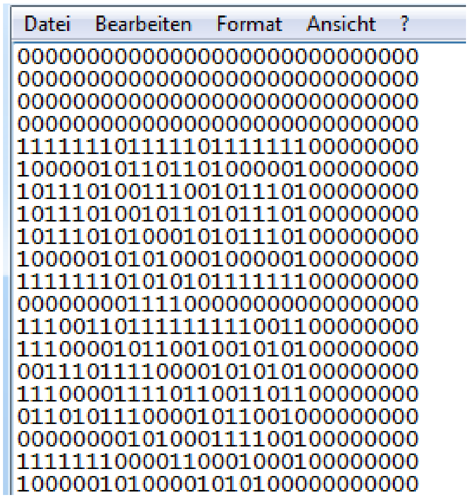
```
thread190.png   295 Bytes   PNG image
thread191.png   301 Bytes   PNG image
thread192.png   301 Bytes   PNG image
thread193.png   295 Bytes   PNG image
thread194.png   301 Bytes   PNG image
thread195.png   295 Bytes   PNG image
thread196.png   295 Bytes   PNG image
thread197.png   295 Bytes   PNG image
thread198.png   301 Bytes   PNG image
thread199.png   295 Bytes   PNG image
thread200.png   301 Bytes   PNG image
thread201.png   301 Bytes   PNG image
thread202.png   301 Bytes   PNG image
```

now i loaded every piece in python and got its pixels with PIL:

```python
from PIL import Image

bin=""
for j in range(0,852):
    s=""
    im = Image.open("thread"+str(j).zfill(3)+".png")
    rgb_im = im.convert('RGBA')
    for i in range(2,im.width):
        r, g, b, a = rgb_im.getpixel((i, 1))
        if a==0:
            s+="1"
        else:
            s+="0"
    if s=="100000000000011":
        bin+="0"
    elif s=="000000000000011":
        bin+="1"
    else:
        print "NOK:"+s
print bin
```

the non-matching patterns i skipped and the other ones i converted to binary.

i had really no idea what to do with the result, until i played with the binary string to find a pattern.

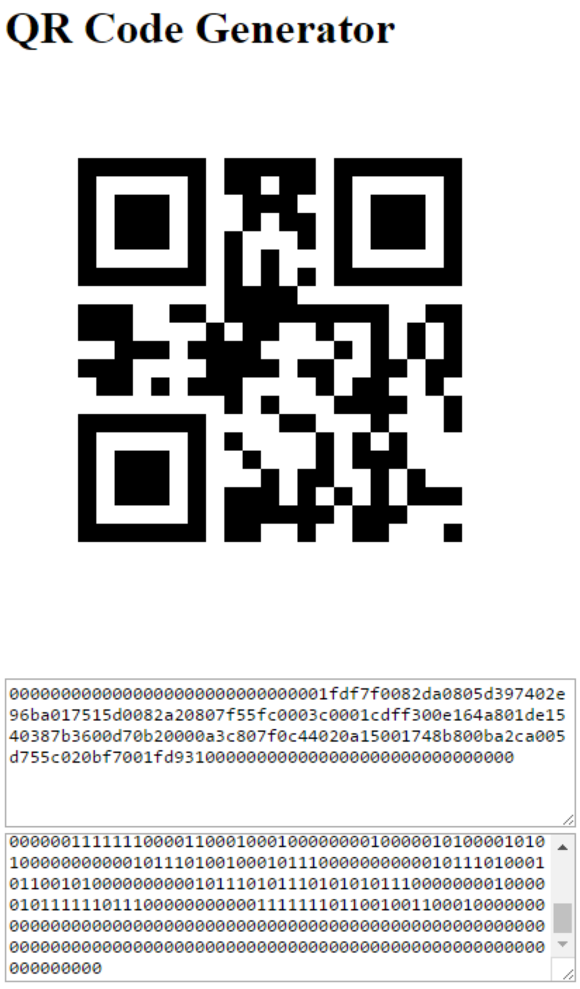


```
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
11111110111110111111100000000
10000010110110100000100000000
10111010011100101110100000000
10111010010110101110100000000
10111010100010101110100000000
10000010101000100000100000000
11111110101010111111100000000
00000000111100000000000000000
11100110111111111001100000000
11100001011001001010100000000
00111011110000101010100000000
11100001111011001101100000000
01101011100001011001000000000
00000000101000111100100000000
11111110000110001000100000000
10000010100001010100000000000
```

in notepad i manually tried to group the lines in a logical manner and somehow appeared this pattern. this looks like a QR code?!?!

i didnt know about that, but googling "binary to QR" brought up this site: https://bahamas10.github.io/binary-to-qrcode/

and using my result in this generator, gave me the solution!

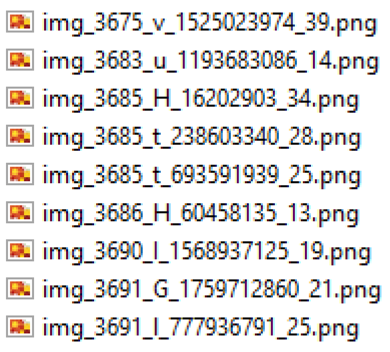


```
000000000000000000000000000001fdf7f0082da0805d397402e
96ba017515d0082a20807f55fc0003c0001cdff300e164a801de15
40387b3600d70b20000a3c807f0c44020a15001748b800ba2ca005
d755c020bf7001fd931000000000000000000000000000000
```

```
0000001111111000011000100010000000001000001010001010
1000000000001011101001000101110000000000010111010001
0110010100000000001011101011101010101110000000010000
0101111110111000000000001111111011001001100010000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
000000000
```

14 - SHARDS

shards. indeed. a lot of random looking image pieces.



```
img_3675_v_1525023974_39.png
img_3683_u_1193683086_14.png
img_3685_H_16202903_34.png
img_3685_t_238603340_28.png
img_3685_t_693591939_25.png
img_3686_H_60458135_13.png
img_3690_I_1568937125_19.png
img_3691_G_1759712860_21.png
img_3691_I_777936791_25.png
```

but are they really random? no. just some of the numbers in the name are random. the letters and numbers at the end are like coordinates of how we can reconstruct the image from this shards. i discovered that by trial and error in paint.net – but what is the easiest way to reconstruct this?

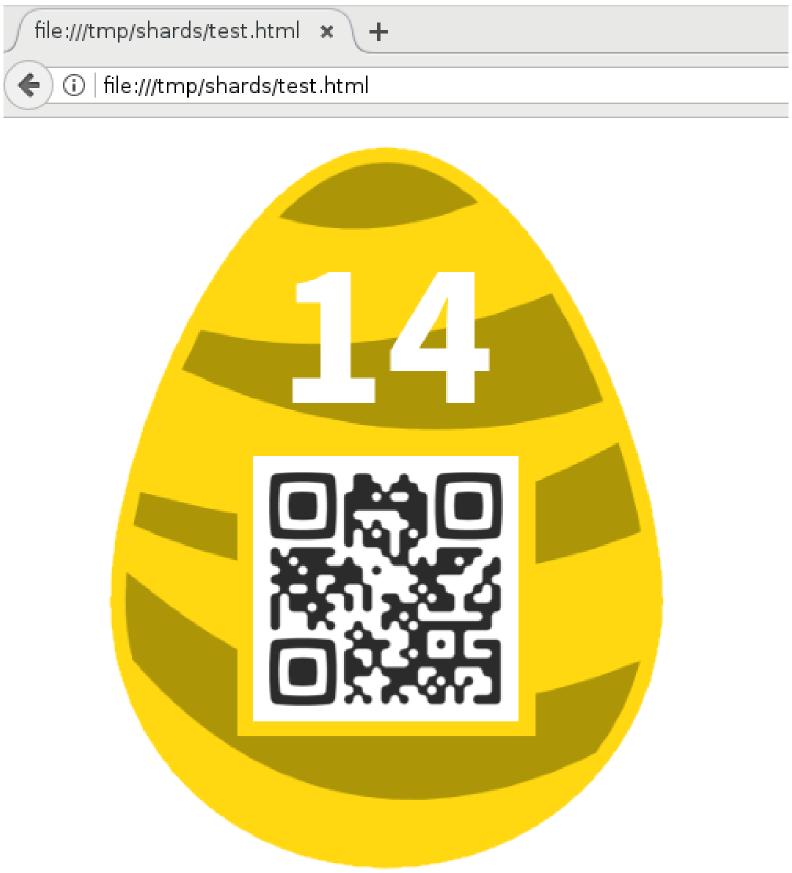
there is a problem on AppleOS and Windows – these operating systems cant handle lower and uppercase letters. means, they will not see a difference between these names, when i strip away the random numbers. lets move to linux then.

in a virtual box, i ran a kali linux and simply renamed the images – means stripping away the random numbers. then i created a html file, which does load these images in the correct order.

this was all pretty simple with this little python script:

```python
import glob
import os
import re

for i in range(0,40):
    images=glob.glob('img_*_'+str(i)+'.png')
    for image in images:
        print image
        newname=re.sub('_\d+_', '_', image)
        os.rename(image, newname)

base="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN"

for c in base:
    html=""
    for i in range(0,40):
        html+="<img src=img_"+c+"_"+str(i)+".png border=0 />"
    print html+"<br>""
```

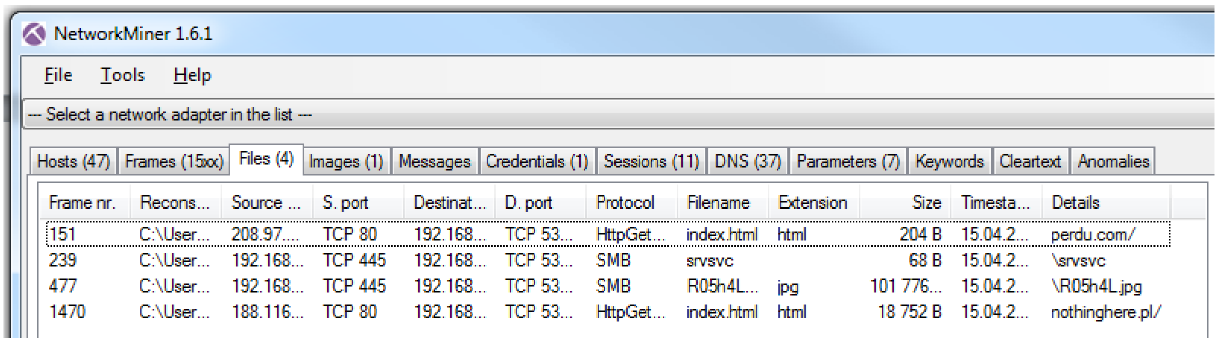
loading this html file then in the same directory resulted in the egg for this challenge. cool idea for a challenge – i really enjoyed it.



15 - P CAP

this challenge gave us a pcapng file, which is obviously a network trace. but there was really a lot of information in it and wireshark file export did not reveal much. i was googling then for a better analysis tool and found this here: http://www.netresec.com/?page=NetworkMiner

but damn, the free version cannot handle pcapng files – another great google search brought up this online converter: http://pcapng.com/ and like this i was able to load a pcap into this tool.

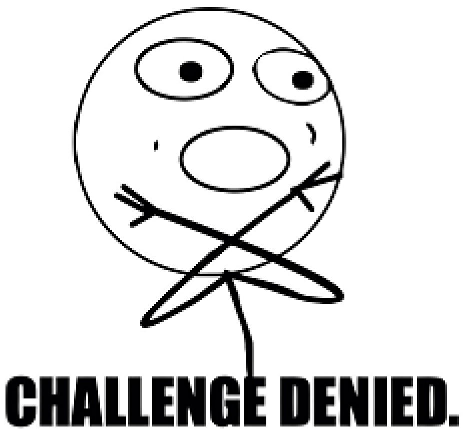

NetworkMiner 1.6.1
File Tools Help
--- Select a network adapter in the list ---
Hosts (47) Frames (15xx) Files (4) Images (1) Messages Credentials (1) Sessions (11) DNS (37) Parameters (7) Keywords Cleartext Anomalies

Frame nr.	Recons...	Source ...	S. port	Destinat...	D. port	Protocol	Filename	Extension	Size	Timesta...	Details
151	C:\User...	208.97....	TCP 80	192.168...	TCP 53...	HttpGet...	index.html	html	204 B	15.04.2...	perdu.com/
239	C:\User...	192.168...	TCP 445	192.168...	TCP 53...	SMB	srvsvc		68 B	15.04.2...	\srvsvc
477	C:\User...	192.168...	TCP 445	192.168...	TCP 53...	SMB	R05h4L...	jpg	101 776...	15.04.2...	\R05h4L.jpg
1470	C:\User...	188.116...	TCP 80	192.168...	TCP 53...	HttpGet...	index.html	html	18 752 B	15.04.2...	nothinghere.pl/

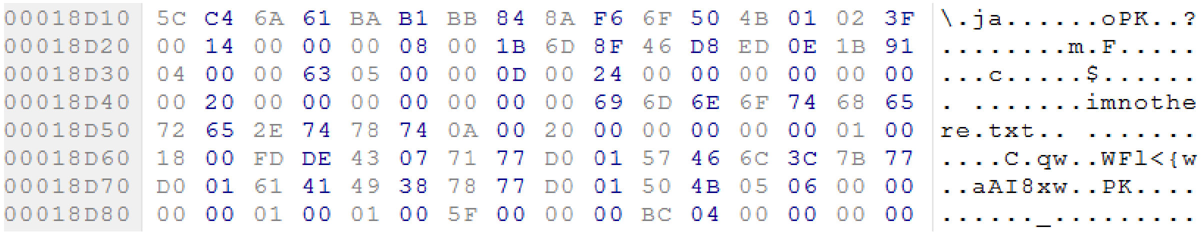
this is an amazing software. it gives us all information in a very good overview. very impressive!

it even does extract all files from the capture automatically in the AssembledFiles folder!

but from all these files, there was only one really interesting one – the image.

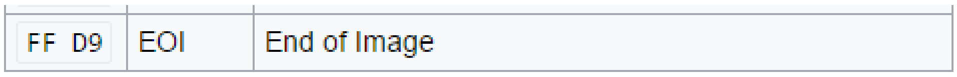


here i was stuck for a moment, until i looked at this file with a hex editor:

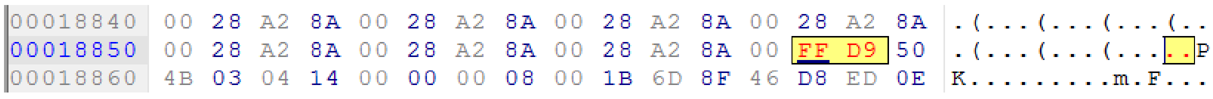


```
00018D10  5C C4 6A 61 BA B1 BB 84 8A F6 6F 50 4B 01 02 3F  \.ja......oPK..?
00018D20  00 14 00 00 00 08 00 1B 6D 8F 46 D8 ED 0E 1B 91  ........m.F.....
00018D30  04 00 00 63 05 00 00 0D 00 24 00 00 00 00 00 00  ...c.....$......
00018D40  00 20 00 00 00 00 00 00 00 69 6D 6E 6F 74 68 65  . .......imnothe
00018D50  72 65 2E 74 78 74 0A 00 20 00 00 00 00 00 01 00  re.txt.. .......
00018D60  18 00 FD DE 43 07 71 77 D0 01 57 46 6C 3C 7B 77  ....C.qw..WFl<{w
00018D70  D0 01 61 41 49 38 78 77 D0 01 50 4B 05 06 00 00  ..aAI8xw..PK....
00018D80  00 00 01 00 01 00 5F 00 00 00 BC 04 00 00 00 00  ......_.........
```

this looke like a zip file at the end of the image (PK header). steganography, yay. we can locate the real end of the jpg file by searching for FF D9 (https://en.wikipedia.org/wiki/JPEG_File_Interchange_Format)

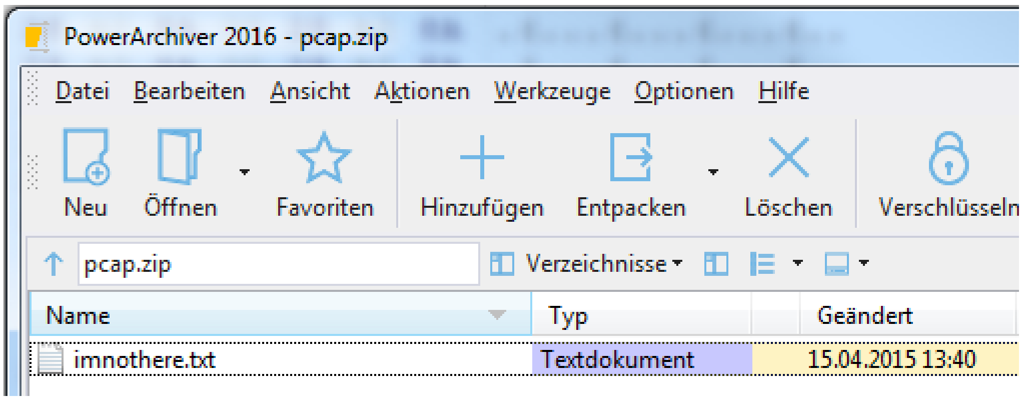


FF D9	EOI	End of Image



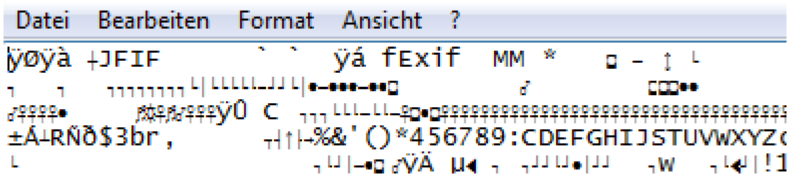
```
00018840  00 28 A2 8A 00 28 A2 8A 00 28 A2 8A 00 28 A2 8A  .(...(...(...(..
00018850  00 28 A2 8A 00 28 A2 8A 00 28 A2 8A 00 FF D9 50  .(...(...(.....P
00018860  4B 03 04 14 00 00 00 08 00 1B 6D 8F 46 D8 ED 0E  K.........m.F...
```

from here to the end, we can copy the bytes into a new file and give it a zip extension. now we can it open in any unarchiver:


PowerArchiver 2016 - pcap.zip
Datei Bearbeiten Ansicht Aktionen Werkzeuge Optionen Hilfe
Neu Öffnen Favoriten Hinzufügen Entpacken Löschen Verschlüsseln
pcap.zip
Verzeichnisse

Name	Typ	Geändert
imnothere.txt	Textdokument	15.04.2015 13:40

it turned out, that this is not a txt file, but another image:



```
ÿØÿà +JFIF        ` `   ÿá fExif   MM *
±Á┴RÑð$3br,        ┤!|-%&'()*456789:CDEFGHIJSTUVWXYZ
```

which contained only this little string:

```
7061n.php
```

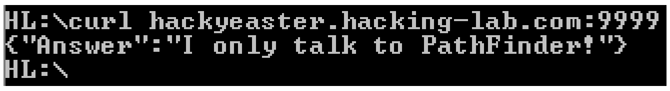
since we are playing hacky easter, we probably should try to load this on the challenges page:

https://hackyeaster.hacking-lab.com/hackyeaster/7061n.php

which showed me then the egg for this level.

16 - PATHFINDER

i tried netcat to connect to the port of this challenge and it didnt even give me an answer. also a browser didnt work and i was out of options for a while. but then, using curl and even wget, i got finally an answer from the challenge and from there i could work on.

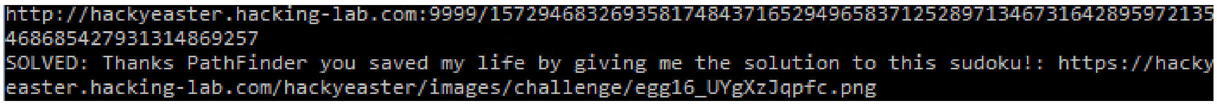


```
HL:\curl hackyeaster.hacking-lab.com:9999
{"Answer":"I only talk to PathFinder!"}
HL:\
```

obviously we have to use a different user-agent.

this riddle gave a headache for a while. i knew, there must be a recursive function to solve it rather easy, but first i tried to write all possible urls to a file – which is not a good idea for this challenge. in the end i implemented a recursive function like this:

```python
import requests, sys

url="http://hackyeaster.hacking-lab.com:9999/"

def getanswer(url):
  result = requests.get(url, headers={'User-Agent': 'PathFinder'}).json()
  answer=result['Answer']
  if answer=="Follow one of the possible paths":
    return result['paths']
  if answer=="Go on! Follow one of the possible paths":
    return result['paths']
  elif answer=="This leads to nowhere, so turn around!":
    return 0
  elif answer=="You've left the path!":
    return 0
  else:
    print "SOLVED: "+answer+": "+result['Secret']
    sys.exit()

def pathfinder(url):
  print url
  result=getanswer(url)
  if result!=0:
    for path in result:
      pathfinder(url+str(path))

pathfinder(url)
```



```
http://hackyeaster.hacking-lab.com:9999/157294683269358174843716529496583712528971346731642895972135
468685427931314869257
SOLVED: Thanks PathFinder you saved my life by giving me the solution to this sudoku!: https://hacky
easter.hacking-lab.com/hackyeaster/images/challenge/egg16_UYgXzJqpfc.png
```

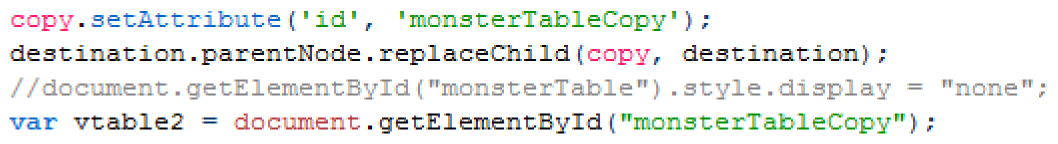
that was a really nice challenge. i learned a lot while solving this. great stuff!

17 - MONSTER PARTY

this was the last challenge that i solved. i dont know, why this is rated medium. for me this was harder than the hard ones. first, it wasnt really clear, what we have to do and i tried a lot of nonsense stuff. but also, it was clear, that we had to produce a QR code in the end and looking at the grid of 27x27, this probably means, we have to move the monsters until the borders are empty. thats how we can get a 25x25 QR code.

since the whole thing was coded in javascript, i decided to solve it in javascript aswell and i must admit – i suck at javascript and did a lot of mistakes (reusing variables, copy paste code errors, etc).

i copied the page locally and implemented a jump button, which will move the monsters according their patterns. like this i was able observe each jump with a click. i noticed, that there is an index table, which is used to place the monsters on the grid. i used this table, to jump with the monsters and displayed it on my local copy by commenting the "hide" function out.



```
copy.setAttribute('id', 'monsterTableCopy');
destination.parentNode.replaceChild(copy, destination);
//document.getElementById("monsterTable").style.display = "none";
var vtable2 = document.getElementById("monsterTableCopy");
```

for every color, i implemented a corresponding jump pattern function (dont blame me for the coding style. this is not optimized at all)

```
<script type="text/javascript">
 function red(x,y,i){
  i=i%4;
  switch (parseInt(i)) {
   case 0:
     break;
   case 3:
     x-=2;
     y+=1;
     break;
   case 1:
     x+=2;
     y+=1;
     break;
   case 2:
     y+=1;
     break;
  }
  if (x<0){
   x+=27;
  }
  if (x>26){
   x=x%27;
  }
  if (y<0){
   y+=27;
  }
  if (y>26){
   y=y%27;
  }
 return [x,y];
 }
```

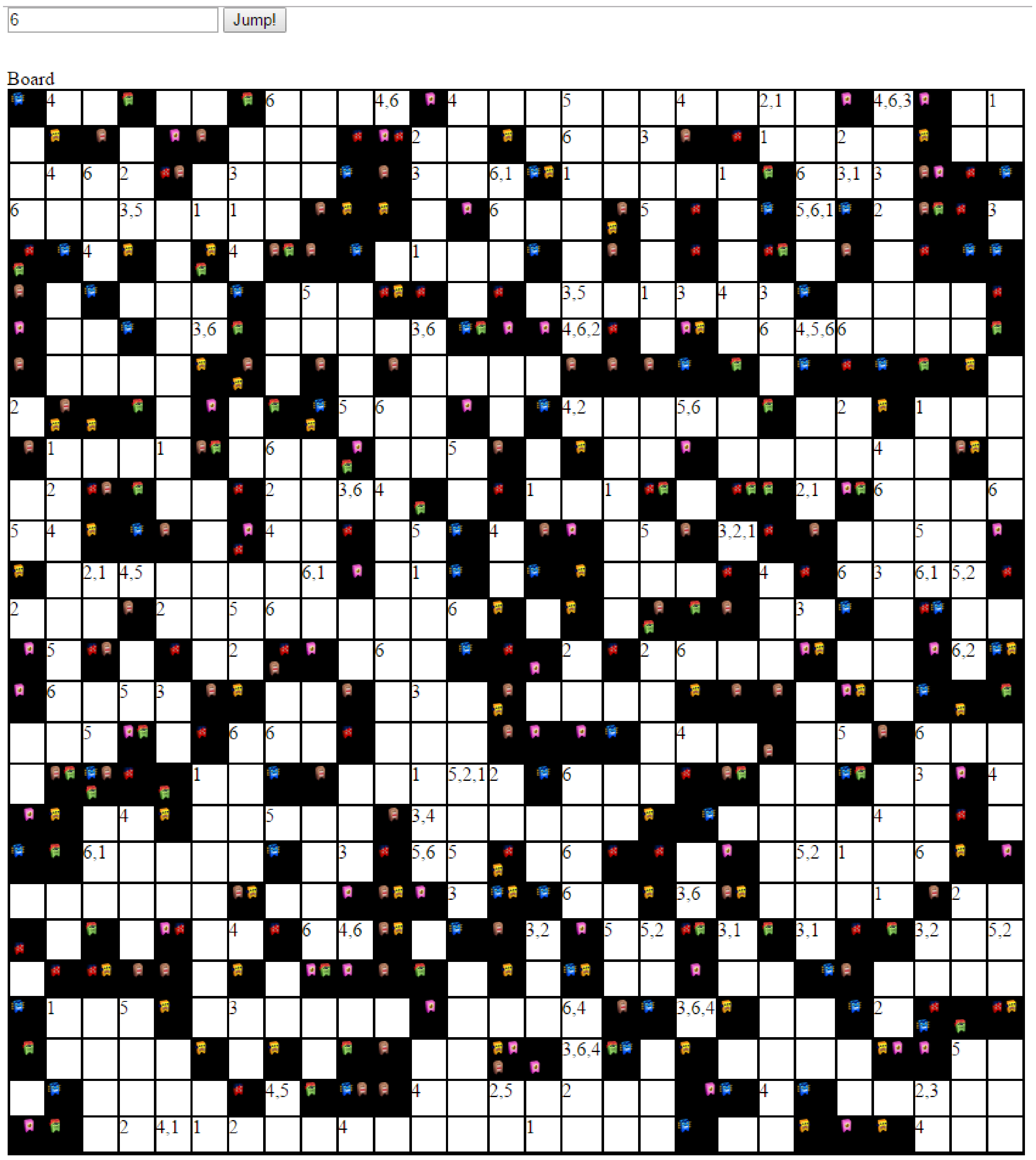
then i adjusted the existing code, to place the monsters to their new location and paint the cell black.

```
case 1:
    c=red(x,y,a);
    for (j = 0; j <= jumps; j++) {
     c=red(c[0],c[1],j);
    }
    vtable2.rows[c[0]].cells[c[1]].innerHTML += "<img src='images/challenge/ico_red.png'>";
    vtable2.rows[c[0]].cells[c[1]].style.backgroundColor  = 'black';
    break;
```

and here is the code for the jump button:

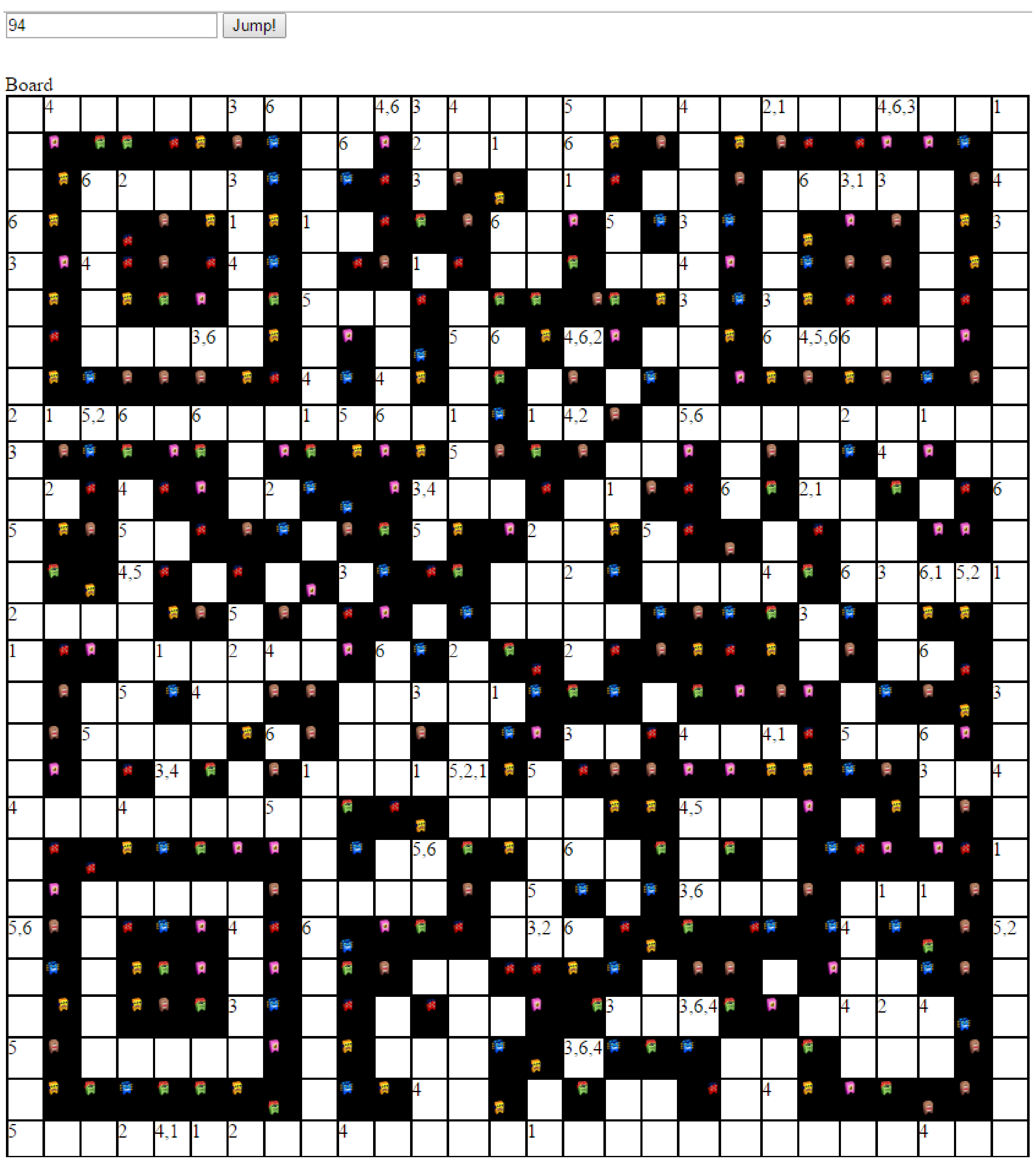
```
<script>
 function incrementValue()
 {
     var value = parseInt(document.getElementById('number').value, 10);
     value = isNaN(value) ? 0 : value;
     value++;
     document.getElementById('number').value = value;
     copyTable(value);
 }
</script>
<form>
   <input type="text" id="number" value="0"/>
   <input type="button" onclick="incrementValue();" value="Jump!" />
</form>
```

now, i was able to jump with all the monsters together and visually display the result:



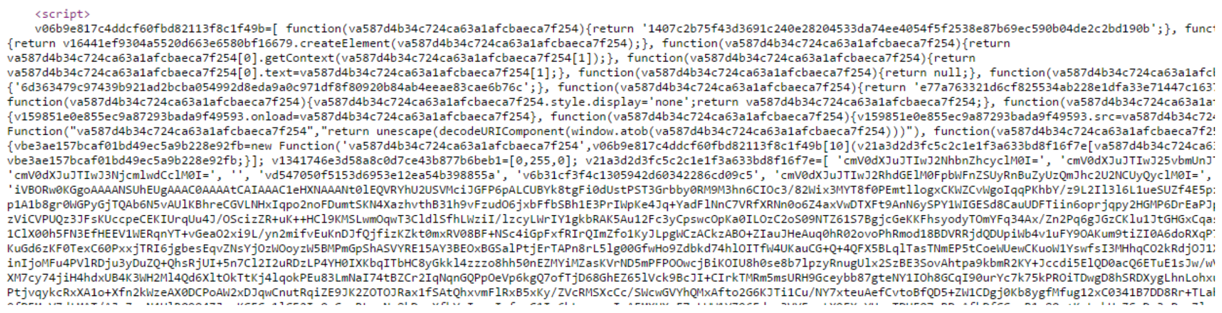
this of course not worked like desired and something must be wrong. i double checked everything and the patterns were correct. in the end it turned out, to be a twist in the game. the monsters did not start from the initial position of their jumping pattern – they already jumped once and i had to adjust my code to jump accordingly.

i almost gave up, but after almost 100 jumps i finally saw a QR code!



18 - NITWIT'S DOORMAT KEY

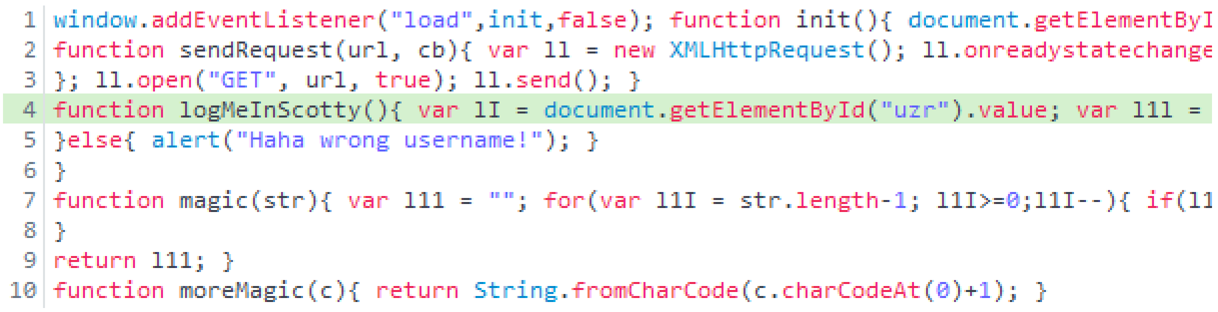
a loginpage to hack. should be easy. but wait – checking the source code of the page shocked me:



```
    <script>
    v06b9e817c4ddcf60fbd82113f8c1f49b=[ function(va587d4b34c724ca63a1afcbaeca7f254){return '1407c2b75f43d3691c240e28204533da74ee4054f5f2538e87b69ec590b04de2c2bd190b';}, func
{return v16441ef9304a5520d663e6580bf16679.createElement(va587d4b34c724ca63a1afcbaeca7f254);}, function(va587d4b34c724ca63a1afcbaeca7f254){return
va587d4b34c724ca63a1afcbaeca7f254[0].getContext(va587d4b34c724ca63a1afcbaeca7f254[1]);}, function(va587d4b34c724ca63a1afcbaeca7f254){return
va587d4b34c724ca63a1afcbaeca7f254[0].text=va587d4b34c724ca63a1afcbaeca7f254[1];}, function(va587d4b34c724ca63a1afcbaeca7f254){return null;}, function(va587d4b34c724ca63a1afc
{'6d363479c97439b921ad2bcba054992d8eda9a0c971df8f80920b84ab4eeae83cae6b76c';}, function(va587d4b34c724ca63a1afcbaeca7f254){return 'e77a763321d6cf825534ab228e1dfa33e71447c163
function(va587d4b34c724ca63a1afcbaeca7f254){va587d4b34c724ca63a1afcbaeca7f254.style.display='none';return va587d4b34c724ca63a1afcbaeca7f254;}, function(va587d4b34c724ca63a1a
{v159851e0e855ec9a87293bada9f49593.onload=va587d4b34c724ca63a1afcbaeca7f254}, function(va587d4b34c724ca63a1afcbaeca7f254){v159851e0e855ec9a87293bada9f49593.src=va587d4b34c72
Function("va587d4b34c724ca63a1afcbaeca7f254","return unescape(decodeURIComponent(window.atob(va587d4b34c724ca63a1afcbaeca7f254)))"), function(va587d4b34c724ca63a1afcbaeca7f2
{vbe3ae157bcaf01bd49ec5a9b228e92fb=new Function('va587d4b34c724ca63a1afcbaeca7f254',v06b9e817c4ddcf60fbd82113f8c1f49b[10](v21a3d2d3fc5c2c1e1f3a633bd8f16f7e[va587d4b34c724ca6
vbe3ae157bcaf01bd49ec5a9b228e92fb;}]; v1341746e3d58a8c0d7ce43b877b6beb1=[0,255,0]; v21a3d2d3fc5c2c1e1f3a633bd8f16f7e=[ 'cmV0dXJuJTIwJ2NhbnZhcyclM0I=', 'cmV0dXJuJTIwJ25vbmUnJ
'cmV0dXJuJTIwJ3NjcmlwdCclM0I=', '', 'vd547050f5153d6953e12ea54b398855a', 'v6b31cf3f4c1305942d60342286cd09c5', 'cmV0dXJuJTIwJ2RhdGElM0FpbWFnZSUyRnBuZyUzQmJhc2U2NCUyQyclM0I=',
'iVBORw0KGgoAAAANSUhEUgAAAC0AAAAtCAIAAAC1eHXNAAANt0lEQVRYhU2USVMciJGFP6pALCUBYk8tgFi0dUstPST3Grbby0RM9M3hn6CIOc3/82Wix3MYT8f0PEmtllogxCKWZCvWgoIqqPKhbY/z9L2Il3l6L1ueSUZf4E5p
p1A1b8gr0WGPyGjTQAb6N5vAUlKBhreCGVLNHxIqpo2noFDumtSKN4XazhvthB31h9vFzudO6jxbFfbSBh1E3PrIWpKe4Jq+YadFlNnC7VRfXRNn0o6Z4axVwDTXFt9AnN6ySPY1WIGESd8CauUDFTiin6oprjqpy2HGMP6DrEaPJ
zViCVPUQz3JFsKUccpeCEKIUrqUu4J/OScizZR+uK++HCl9KMSLwmOqwT3CldlSfhLWziI/lzcyLWrIY1gkbRAK5Au12Fc3yCpswcOpKa0ILOzC2oS09NTZ61S7BgjcGeKKFhsyodyTOmYFq34Ax/Zn2Pq6gJGzCKlu1JtGHGxCqa
1ClX00h5FN3EfHEEV1WERqnYT+vGeaO2xi9L/yn2mifvEuKnDJfQjfizKZkt0mxRV08BF+NSc4iGpFxfRIrQImZfo1KyJLpgWCzACkzABO+ZIauJHeAuq0hR02ovoPhRmod18BDVRRjdQDUpiWb4v1uFY9OAKum9tiZI0A6doRXqP
KuGd6zKF0TexC60PxxjTRI6jgbesEqvZNsYjOzWOoyzW5BMPmGpShASVYRE15AY3BEOxBGSalPtjErTAPn8rL5lg00GfwHo9Zdbkd74hlOITfW4UKauCG+Q+4QFX5BLqlTasTNmEP5tCoeWUewCKuoW1YswfsI3MHhqCO2kRdjOJ1
inIjoMFu4PVlRDju3yDuZQ+QhsRjUI+5n7Cl2I2uRDzLP4YH0IXKbqITbHC8yGkkl4zzzo8hh50nEZMYiMZasKVrND5mPFPOOwcjBiKOIU8h0se8b7lpzyRnugUlx2SzBE3SovAhtpa9kbmR2KY+Jccdi5ElQD0acQ6ETuE1sJw/w
XM7cy74jiH4hdxUB4K3WH2Ml4Qd6XltOkTtKj4lqokPEu83LmNaI74tBZCr2IqNqnGQPpOeVp6kgQ7ofTjD68GhEZ65lVck9BcJI+CIrkTMRm5msURH9Gceybb87gteNY1IOh8GCqI90urYc7k75kPROiTDwgD8hSRDXygLhnLohx
PtjvqykcRxXA1o+Xfn2kWzeAX0DCPoAW2xDJqwCnutRqiZE9JK2ZOTOJRax1fSAtQhxvmFlRxB5xKy/ZVcRMSXcCc/SWcwGVYhQMxAfto2G6KJTi1Cu/NY7xteuAefCvtoBfQD5+ZW1CDgj0Kb8ygfMfug12xC0341B7DD8Rr+TLa
```

argh. very obfuscated javascript code. no idea how this was made up and no deobfuscator was able to help here. i copied the page to my local PC and used a javascript beautifier to make it a little better, but still it was a mess.

i used firefox and the debugger from the inspector to place some breakpoints on almost every function, because i wanted to check the values during runtime. funny thing happened: i ran it and entered some dummy username and password and firefox showed me the deobfuscated javascript code in the debugger console. nice!



```
window.addEventListener("load",init,false); function init(){ document.getElementByI
function sendRequest(url, cb){ var l1 = new XMLHttpRequest(); l1.onreadystatechange
}; l1.open("GET", url, true); l1.send(); }
function logMeInScotty(){ var lI = document.getElementById("uzr").value; var l1l =
}else{ alert("Haha wrong username!"); }
}
function magic(str){ var l11 = ""; for(var l1I = str.length-1; l1I>=0;l1I--){ if(l1
}
return l11; }
function moreMagic(c){ return String.fromCharCode(c.charCodeAt(0)+1); }
```

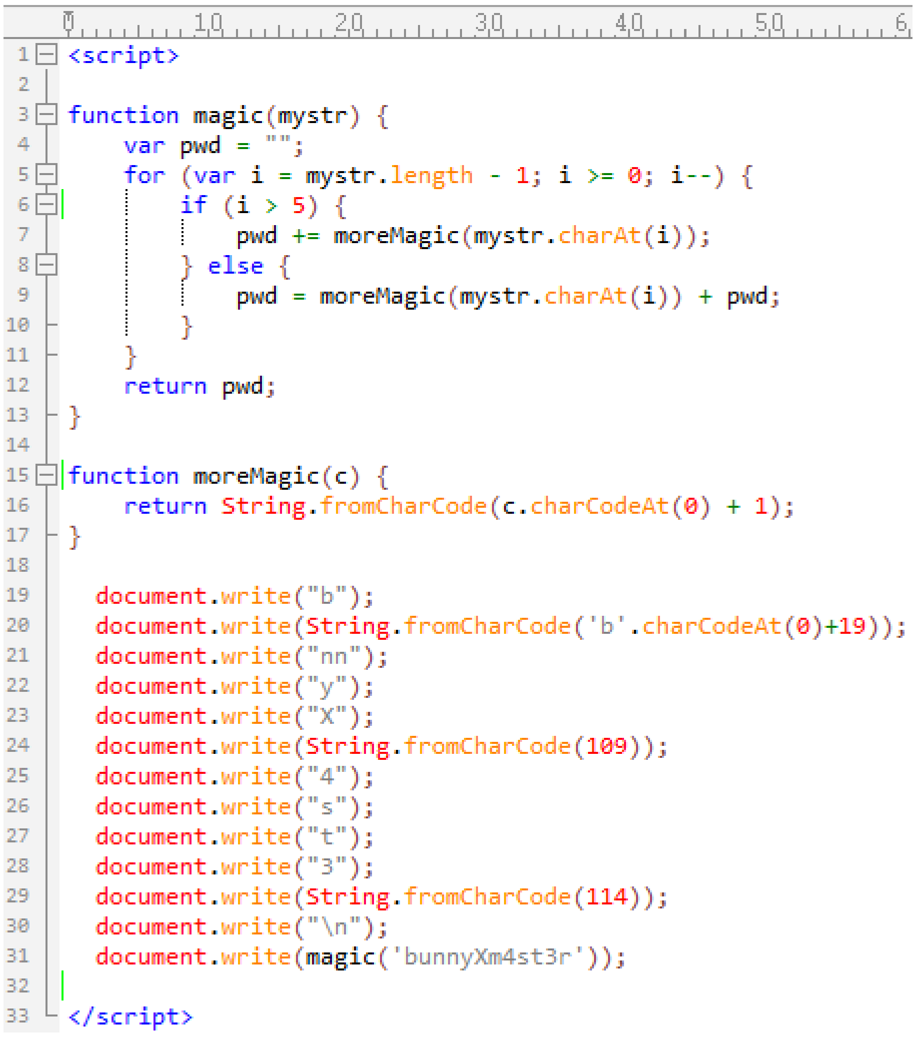
using a javascript beautifier again, i finally got clean code:

```
function logMeInScotty() {
    var lI = document.getElementById("uzr").value;
    var l1l = document.getElementById("puzzwerd").value;
    if (lI.length == 12 && (lI[0] == "b") && (lI.charCodeAt(0) == lI.charCodeAt(1) - 19)
        if (l1l == magic(lI)) {
            dataUrl = 'https:' + String.fromCharCode(47, 47) + 'hackyeaster.hacking-lab.
            sendRequest(dataUrl, function(lIl) {
                document.getElementById("egg").src = "data:image/png;base64," + lIl;
            });
        } else {
            alert("Haha wrong password!");
        }
    } else {
        alert("Haha wrong username!");
    }
}
```

now it was just a matter of interpreting the code, to find the username and then throw it into the magic function:

```
if (username.length == 12 &&
  (username[0] == "b") &&
  (username.charCodeAt(0) == username.charCodeAt(1) - 19) &&
  (String.fromCharCode(username.charCodeAt(3) & 0x7F) == "n") &&
  (username[3] == username[2]) &&
  (username.charCodeAt(4) == username.charCodeAt(1) + username[7] * 1) &&
  (username[5] == "X!&)=" [0]) &&
  (username[6] == String.fromCharCode(109)) &&
  (username[7] == (1 << 2)) &&
  (username[8] == "s") &&
  (username.charCodeAt(8) == username.charCodeAt(9) - 1) &&
  (username[10] == username[7] - 1) &&
  (username[11] == String.fromCharCode(114))) {
   if (password == magic(username)) {
```

this was runtime javascript and didnt work like this. i had to adjust some things – also in the magic function, but in the end i came up with this javascript to solve it:

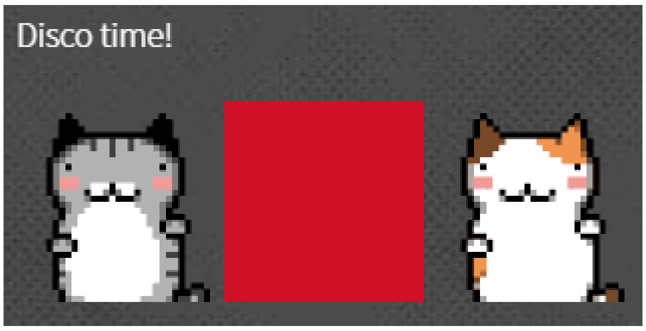


```
<script>

function magic(mystr) {
    var pwd = "";
    for (var i = mystr.length - 1; i >= 0; i--) {
        if (i > 5) {
            pwd += moreMagic(mystr.charAt(i));
        } else {
            pwd = moreMagic(mystr.charAt(i)) + pwd;
        }
    }
    return pwd;
}

function moreMagic(c) {
    return String.fromCharCode(c.charCodeAt(0) + 1);
}

  document.write("b");
  document.write(String.fromCharCode('b'.charCodeAt(0)+19));
  document.write("nn");
  document.write("y");
  document.write("X");
  document.write(String.fromCharCode(109));
  document.write("4");
  document.write("s");
  document.write("t");
  document.write("3");
  document.write(String.fromCharCode(114));
  document.write("\n");
  document.write(magic('bunnyXm4st3r'));

</script>
```

Username: bunnyXm4st3r
Password: cvoozYs4ut5n

19 - DISCO TIME

funny. lets disco!



obviously we have one interesting image in this challenge. its an animated gif and its made of a lot of frames. its not really visible, but this image contains thousands of frames!

with imagemagick we can split this image into its frames again using this simple command:

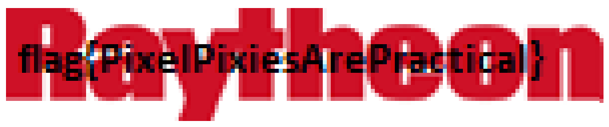
convert disco2.gif disco.png

whuuut? this gave me 4172 files! those are almost all red or black pixels. what the heck should we do with that?

the only thing that makes sense, is to put them together into a new image. like they were a puzzle. now i could have used a python image library, but this would also mean a lot of code. because im lazy, i just made up a html file, which loads all these images and orders them correctly (of course using python)

```
html="<table border=0 cellspacing=0 cellpadding=0><tr><td>"
for i in range(1,4172):
    html+="<img src=disco20"+str(i).zfill(4)+".png border=0 /><br>"
    if i%28==0:
        html+="</td><td>"
print html+"</td></tr></table>"
```

i dont know, if there was a way to find out how many pixels there should be in a row. i just noticed some sort of pattern and with trial and error i found out, that i should start a new row after 28 pixels. with the html solution, this was rather easy to do and here is the solution:

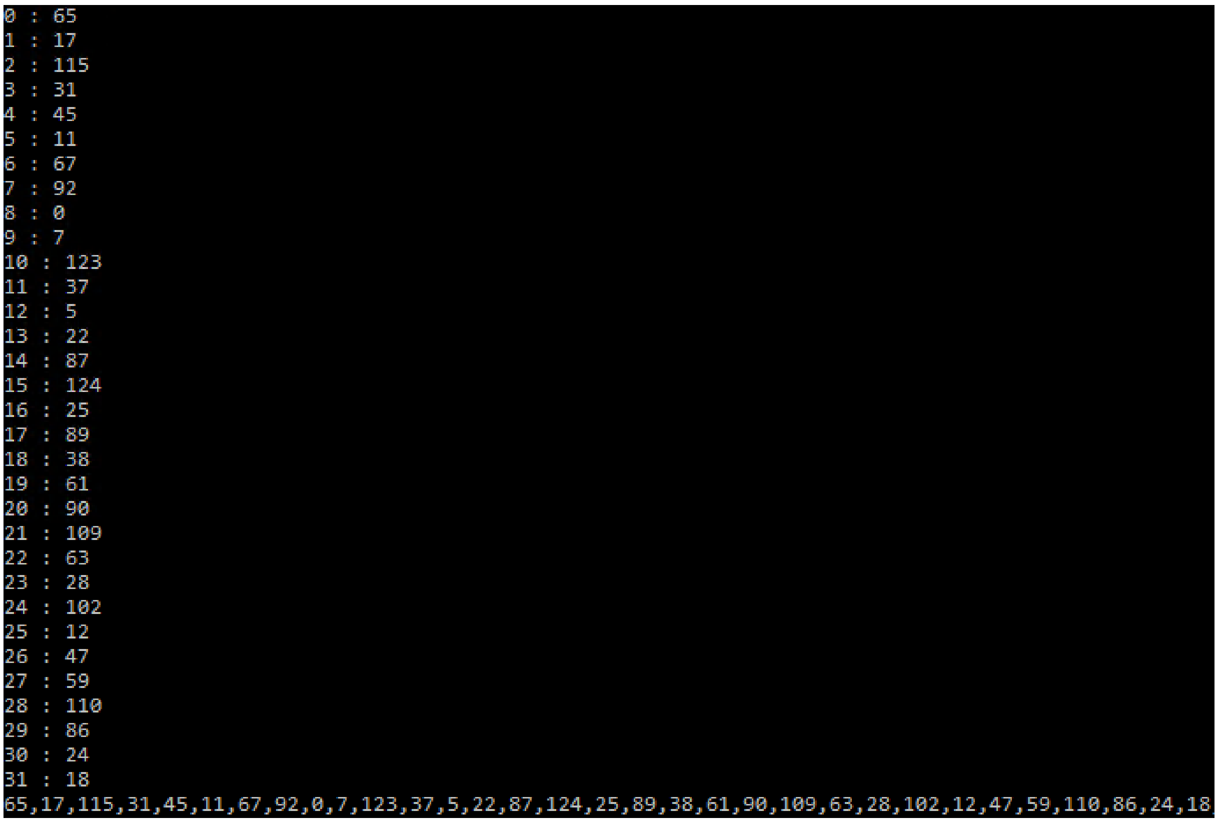


20 - SPAGHETTI HASH

so... we have again a hash collision or bad implementation of hashes challenge here. we know, that the short hash is made up of characters of the big hash (SHA-512). the first step is therefore, to find out, which chars of the bigger hashes are used. for this i made up a little python script using some random outputs from the testing page:

```
hash1="ee26b0dd4af7e749aa1a8ee3c10ae9923f618980772e473f8819a5d4940e0db27ac185f8a0e1d5f84f88bc887fd67b143732c304cc5fa9ad8e6f57f50028a8ff"
short1="aaf27717ed890e8a1f8dd92e0efea8c1"
hash2="39ca2b1f97c7d1d223dcb2b22cbe20c36f920aeefd201d0bf68ffc08db6d9ac608a0a202fb536d944c9d1f50cf9bd61b5bc84217212f0727a8db8a01c2fa54b7"
short2="83b3d70d3faabb05cfea97621dbd252d"
hash3="9e7befeb12c45c31259344585530362301477b18cbe73de23419fae50c0f86f665594d5816bba4d0a86850aaea4d41296e3d668d6fcb4030523ced0dcab3840f"
short3="55c3d4949b3bf5a85a164063852f3a59"

x=""

for i in range(0,32):
  a=short1[i]
  b=short2[i]
  c=short3[i]
  for j in range(0, 128):
    if hash1[j]==a and hash2[j]==b and hash3[j]==c:
      print i,":",j
      x+=str(j)+","
      break
print x
```

checking three hashes was enough to find the pattern.


```
0 : 65
1 : 17
2 : 115
3 : 31
4 : 45
5 : 11
6 : 67
7 : 92
8 : 0
9 : 7
10 : 123
11 : 37
12 : 5
13 : 22
14 : 87
15 : 124
16 : 25
17 : 89
18 : 38
19 : 61
20 : 90
21 : 109
22 : 63
23 : 28
24 : 102
25 : 12
26 : 47
27 : 59
28 : 110
29 : 86
30 : 24
31 : 18
65,17,115,31,45,11,67,92,0,7,123,37,5,22,87,124,25,89,38,61,90,109,63,28,102,12,47,59,110,86,24,18
```

now i just had to use the found indexes (positions of the chars used from SHA-512) with a bruteforcer to recover the passwords.

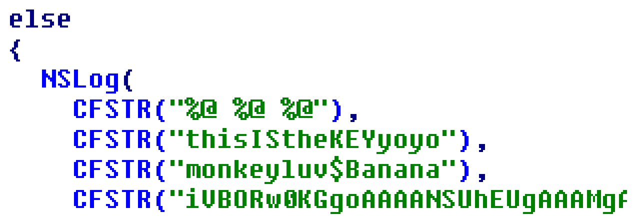
```
import hashlib

indexes=[65,17,115,31,45,11,67,92,0,7,123,37,5,22,87,124,25,89,38,61,90,109,63,28,102,12,47,59,110,86,24,18]

hash1="87017a3ffc7bdd5dc5d5c9c348ca21c5"
hash2="ff17891414f7d15aa4719689c44ea039"
hash3="5b9ea4569ad68b85c7230321ecda3780"
hash4="6ad211c3f933df6e5569adf21d261637"

with open(r"C:\Users\thumper\Downloads\rockyou.txt") as fp:
  for line in fp:
    h=""
    mypass=line.strip()
    hash=hashlib.sha512(mypass).hexdigest()
    for number in indexes:
      h+=hash[number]
    if h==hash1 or h==hash2 or h==hash3 or h==hash4:
      print h+":"+mypass

#87017a3ffc7bdd5dc5d5c9c348ca21c5:Prodigy
#ff17891414f7d15aa4719689c44ea039:Cleveland
#5b9ea4569ad68b85c7230321ecda3780:benchmark
#6ad211c3f933df6e5569adf21d261637:12345678
```

21 - MONKEY

hey, this is an iphone binary! very nice! immediately loading in IDA and checking out the strings. we are even lucky and this was compiled in 32bit and 64bit mode – therefore we can use hex-rays for pseudo code.

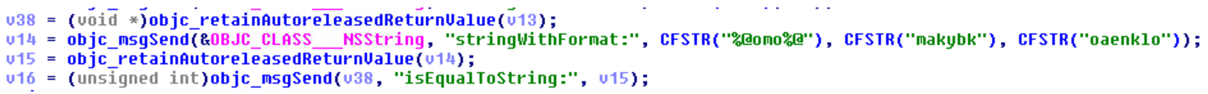
i also installed the app itself on my iphone, using cydia impactor from:
http://www.cydiaimpactor.com
and my developer certificate (but it works also with the free developer certificate)

the binary contained some honeypot with wrong keys and dummy images:



```
else
{
  NSLog(
    CFSTR("%@ %@ %@"),
    CFSTR("thisIStheKEYyoyo"),
    CFSTR("monkeyluv$Banana"),
    CFSTR("iVBORw0KGgoAAAANSUhEUgAAAMgf
```

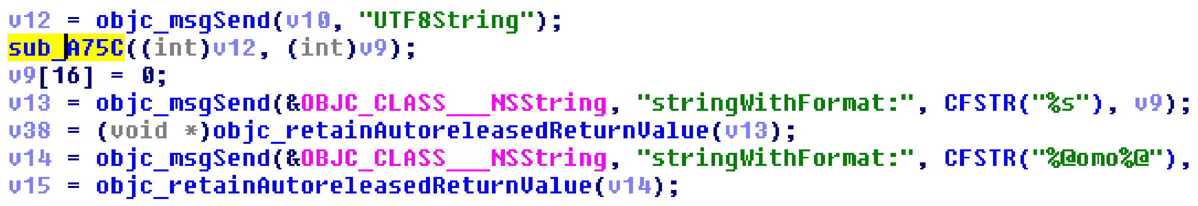
of course, it was not that easy. the key itself seems to be checked here:



```
v38 = (void *)objc_retainAutoreleasedReturnValue(v13);
v14 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("%@omo%@"), CFSTR("makybk"), CFSTR("oaenklo"));
v15 = objc_retainAutoreleasedReturnValue(v14);
v16 = (unsigned int)objc_msgSend(v38, "isEqualToString:", v15);
```

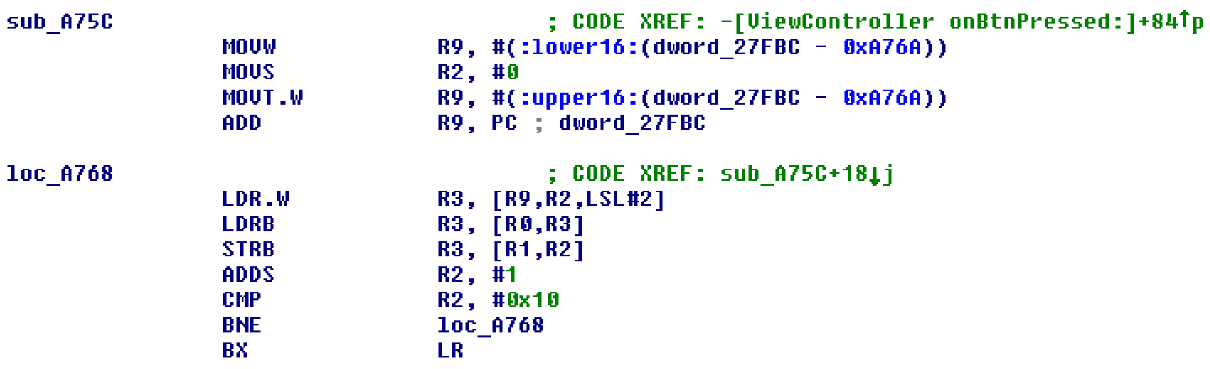
but whatever i tried, this key "makybkomooaenklo" didnt work on the iphone.

i really did overlook something here and when i saw it – facepalm. oh no.



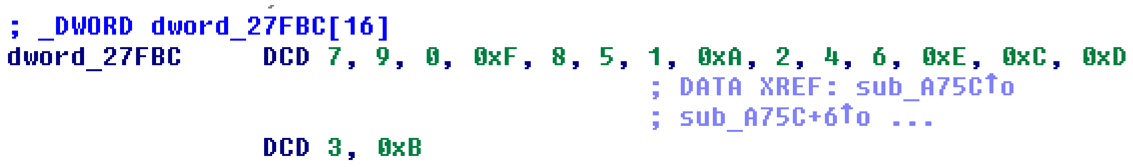
```
v12 = objc_msgSend(v10, "UTF8String");
sub_A75C((int)v12, (int)v9);
v9[16] = 0;
v13 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("%s"), v9);
v38 = (void *)objc_retainAutoreleasedReturnValue(v13);
v14 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("%@omo%@"),
v15 = objc_retainAutoreleasedReturnValue(v14);
```

there is a sub before the check!!!!



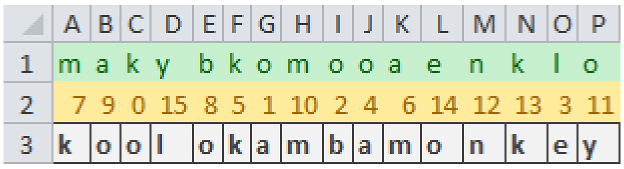
```
sub_A75C                                ; CODE XREF: -[ViewController onBtnPressed:]+84↑p
                MOVW            R9, #(:lower16:(dword_27FBC - 0xA76A))
                MOVS            R2, #0
                MOVT.W          R9, #(:upper16:(dword_27FBC - 0xA76A))
                ADD             R9, PC ; dword_27FBC

loc_A768                                ; CODE XREF: sub_A75C+18↓j
                LDR.W           R3, [R9,R2,LSL#2]
                LDRB            R3, [R0,R3]
                STRB            R3, [R1,R2]
                ADDS            R2, #1
                CMP             R2, #0x10
                BNE             loc_A768
                BX              LR
```

this is a lookup function, which does change the order of the string based on the table referenced in this sub.



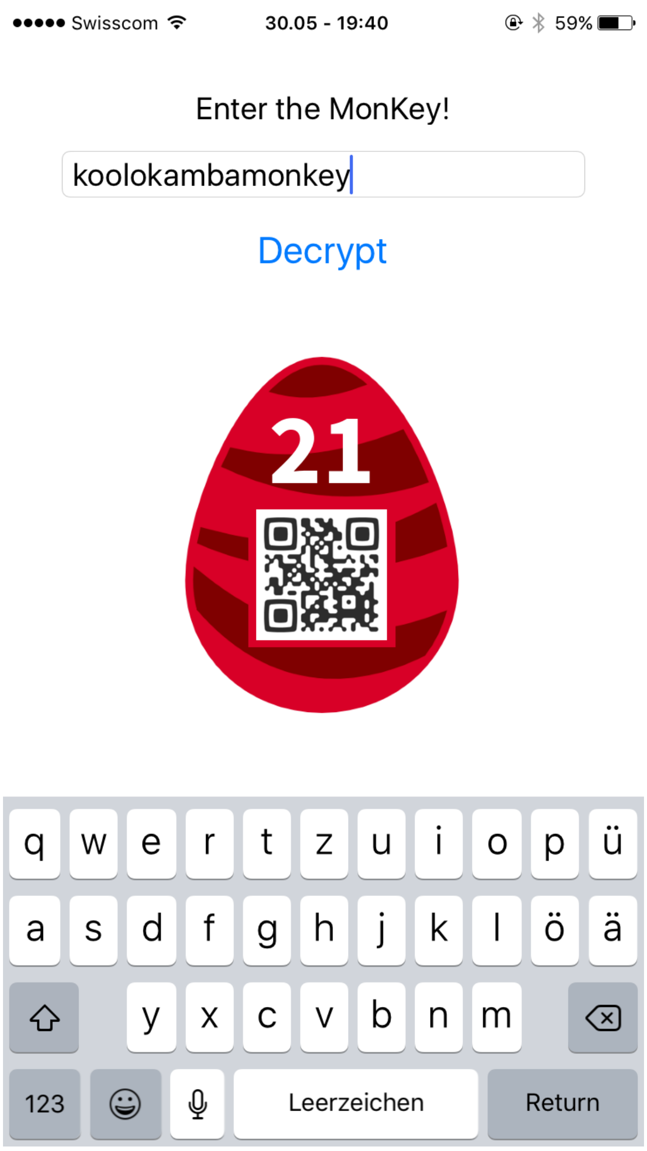
```
; _DWORD dword_27FBC[16]
dword_27FBC     DCD 7, 9, 0, 0xF, 8, 5, 1, 0xA, 2, 4, 6, 0xE, 0xC, 0xD
                                        ; DATA XREF: sub_A75C↑o
                                        ; sub_A75C+6↑o ...
                DCD 3, 0xB
```

makybkomooaenklo must be reordered according to this table. i made this manually in excel:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	m	a	k	y	b	k	o	m	o	o	a	e	n	k	l	o
2	7	9	0	15	8	5	1	10	2	4	6	14	12	13	3	11
3	k	o	o	l	o	k	a	m	b	a	m	o	n	k	e	y

and here we go with our egg:



22 - GAME, SET AND HASH

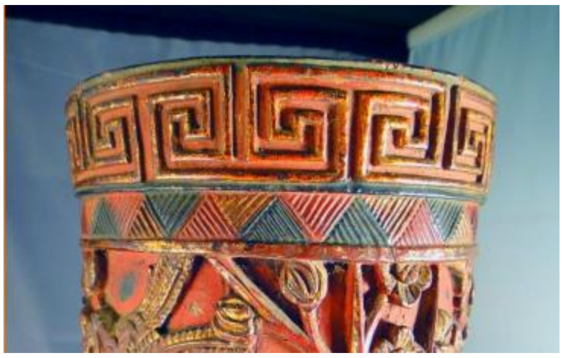
so in this game-type challenge, we connect to a service, which gives us some hashes to crack. thing is, they are rather big and we dont have the time to bruteforce them and i dont have a rainbow table of sha-256 locally.

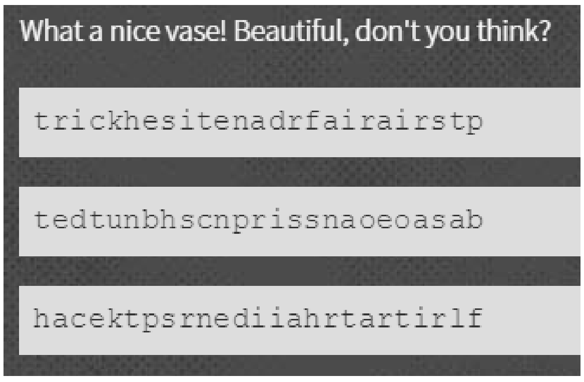
every hash that i tried to solve with google, did actually give a hit on some hash cracker sites. there was one which offered obviously all of them: http://hashtoolkit.com/reverse-sha256-hash

i decided to use this "service" with a python script in realtime. with every hash that the challenge gave me, i call this website and using regex, i get the "decrypted" password from it. easy peasy.

```
import socket
import requests
import re

TCP_IP = "hackyeaster.hacking-lab.com"
TCP_PORT = 8888
url="http://hashtoolkit.com/reverse-sha256-hash/"
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
print "[]connecting..."
data = s.recv(100)
print data
print "[]getting banner..."
print "[]sending y..."
s.send("y\n")
print "[]getting go..."
go=s.recv(100)
print go

for x in range(0, 100):
  print "[]getting hash..."
  resp=s.recv(200)
  print "[hash response]:"+ resp
  hash=re.search(r'[0-9a-f]{64}',resp).group()
  print "[]cracking hash"
  r = requests.get(url+hash)
  password=re.search(r'decrypted sha256 hash.*?</span>', r.content)\
  .group().replace('decrypted sha256 hash">','').replace('</span>','')
  print "[]sending password: "+password
  s.send(password)
  s.send("\n")
  print "[]getting response"
  print s.recv(200)

s.close()
```

23 - LOVELY VASE





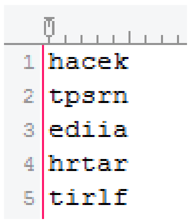
What a nice vase! Beautiful, don't you think?

```
trickhesitenadrfairairstp

tedtunbhscnprissnaoeoasab

hacektpsrnediiahrtartirlf
```

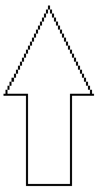
so. this is obviously a manual cipher based on the signs on this vase. its not so obvious how it works and i did get the solution by good guessing in the end.

we have 25 chars on each string and i just tried to find a logic behind them.

i played a little with these strings in ultraedit and added a line break after 5 characters. then i noticed something in the last string:

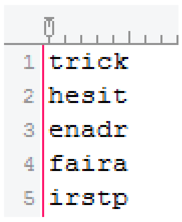


```
hacek
tpsrn
ediia
hrtar
tirlf
```

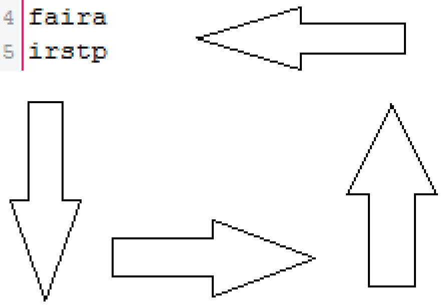


reading from bottom to top will give "the third part is claire frank"

nice! this was easy, but what about the other parts? like i said, i used a lot of guessing and trying to find the correct solution. but i knew, that the sentences have to start with "the first part is" and "the second part is" – this helped a lot to find the pattern for the other two strings.

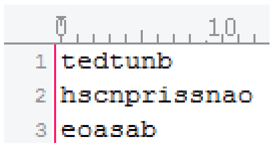


```
trick
hesit
enadr
faira
irstp
```



the first part was using the symbol from the vase. from top down, then right, then up, then left and so on. i think you get the idea. this will give us: "the first part is adriane rick"

the second part, was harder to solve, but when i reordered the characters like this:



```
tedtunb
hscnprissnao
eoasab
```

i was able to read from top to bottom and then always up and down:

"the second part is susanna bob"

now, we have three names: adriane rick, susanna bob and claire frank. to get the QR code we just needed to enter all these together like this: **adrianericksusannabobclairefrank**

24 - YOUR PASSPORT, PLEASE

i knew, there will be a java challenge. every year, i pull my hair because of this challenges. and even worse – this time we NEED to use eclipse.

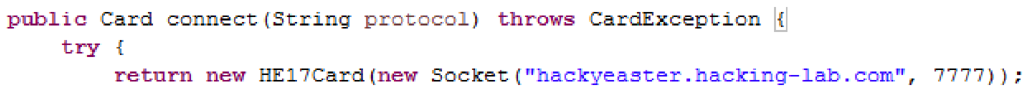
i was really out of ideas about this challenge and the topic itself shocked me. not the challenge itself, but the language and the environment. yeah, we got the project template, but opening this in eclipse threw so many errors, that i closed it straight away.

later, when my hair grew again, i gave this challenge another try and with google i found out, that these errors can be solved by configuring eclipse correctly. wtf. really?

anyway – this challenge is about how to read an epassport. with a lot of research, i have managed to copy/paste some github example code together that in the end really worked. this here was a good source for example:

https://github.com/tananaev/passport-reader/blob/master/app/src/main/java/com/tananaev/passportreader/MainActivity.java

but first of all, i had to adjust the code to really connect to the challenge server instead of localhost:



```
public Card connect(String protocol) throws CardException {
    try {
        return new HE17Card(new Socket("hackyeaster.hacking-lab.com", 7777));
```

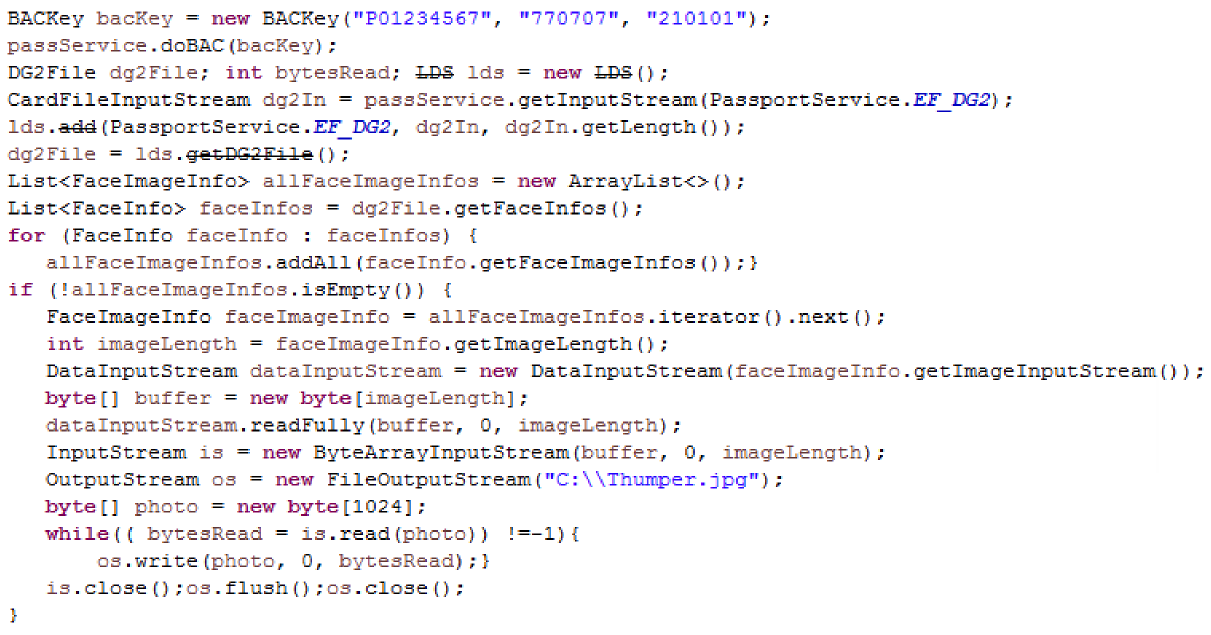
of course i had to read through a lot of specifications and forums, to understand what we really have to do.

here for example was exactly explained how we can read the image from an epassport: https://stackoverflow.com/questions/21849601/android-nfc-read-data-from-epassport

You need the make a BAC (Basic Access Control) against your epassport to be able to read the basic informations printed on the passport

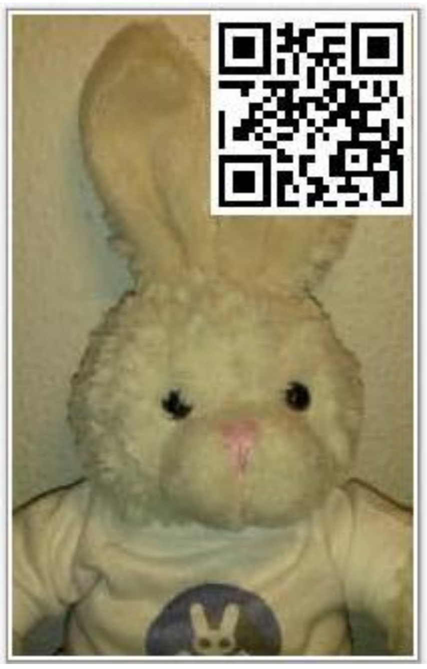
All this information is located in the DG (Data Group) 1, the photo is located in the DG2.

So, we obviously need to get DG2 after a BAC. it felt like an endless time of google and copy pasting source codes, but i finally came up with a working java code:



```
BACKey bacKey = new BACKey("P01234567", "770707", "210101");
passService.doBAC(bacKey);
DG2File dg2File; int bytesRead; ~~LDS~~ lds = new ~~LDS~~();
CardFileInputStream dg2In = passService.getInputStream(PassportService.EF_DG2);
lds.~~add~~(PassportService.EF_DG2, dg2In, dg2In.getLength());
dg2File = lds.~~getDG2File~~();
List<FaceImageInfo> allFaceImageInfos = new ArrayList<>();
List<FaceInfo> faceInfos = dg2File.getFaceInfos();
for (FaceInfo faceInfo : faceInfos) {
   allFaceImageInfos.addAll(faceInfo.getFaceImageInfos());}
if (!allFaceImageInfos.isEmpty()) {
   FaceImageInfo faceImageInfo = allFaceImageInfos.iterator().next();
   int imageLength = faceImageInfo.getImageLength();
   DataInputStream dataInputStream = new DataInputStream(faceImageInfo.getImageInputStream());
   byte[] buffer = new byte[imageLength];
   dataInputStream.readFully(buffer, 0, imageLength);
   InputStream is = new ByteArrayInputStream(buffer, 0, imageLength);
   OutputStream os = new FileOutputStream("C:\\Thumper.jpg");
   byte[] photo = new byte[1024];
   while(( bytesRead = is.read(photo)) !=-1){
       os.write(photo, 0, bytesRead);}
   is.close();os.flush();os.close();
}
```

it seems, that i was using deprecated functions and eclipse didnt like it, thats why some of the code is strikethrough (wtf). however, this magically worked and i was able to load the photo of thumper of his epassport.



END

this year was harder than the last years and also the range of challenges was different, because of volunteers supplying their ideas. like every year, this was a lot of fun and i learned again new topics and techniques. thanks a lot to PS for his awesome work creating this event and thanks aswell for the people who helped!

see you next easter!

hardlock