HACKY EASTER 2018 WRITEUP

by Christoph Kolbicz

CONTENTS

00 TEASER

we are given a game and the challenge description states:"Beat the boss and get the Easter egg! Submit the solution code of the egg" - but we are hackers and will beat the boss in a different way!

i simply started the game, saved it once and exited again. then i used a savegame editor (RpgMakerSaveEdit) to give myself the egg item.

in the game again i was able to find the secret key - just needed to convert it from hex to ascii.

Password: 47ru3h3r0

01 PRISON BREAK

this challenge gives us two phone numbers and an image. a quick google with "prison break origami code" brought me to this page with the solution: http://www.givememyremote.com/remote/2006/09/26/prison-break-origami-code-broken/

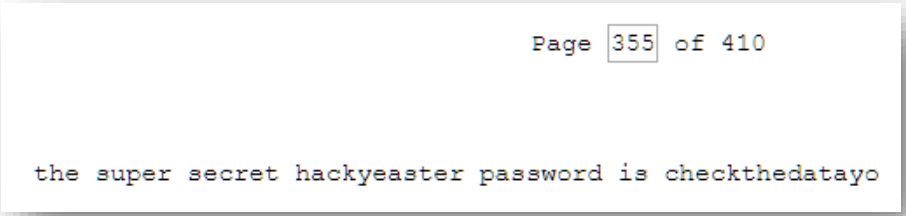
following these steps, i discovered the password: prisonerisking

02 BABYLON

after hours of google (i simply didnt read good enough) i finally found the library of babel homepage. using the codes from the challenge, we can search the library on this link:

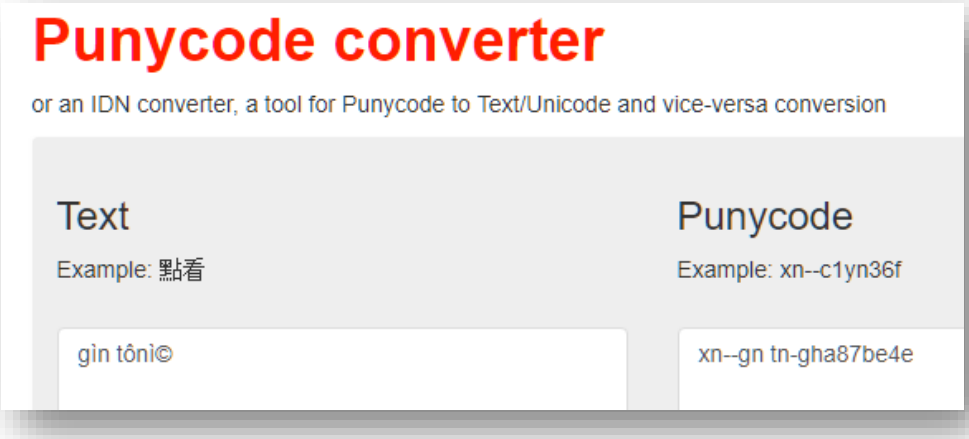
https://libraryofbabel.info/browse.cgi

using the hex from the challenge and the other parameters we can discover the challenges key:



03 PONY CODER

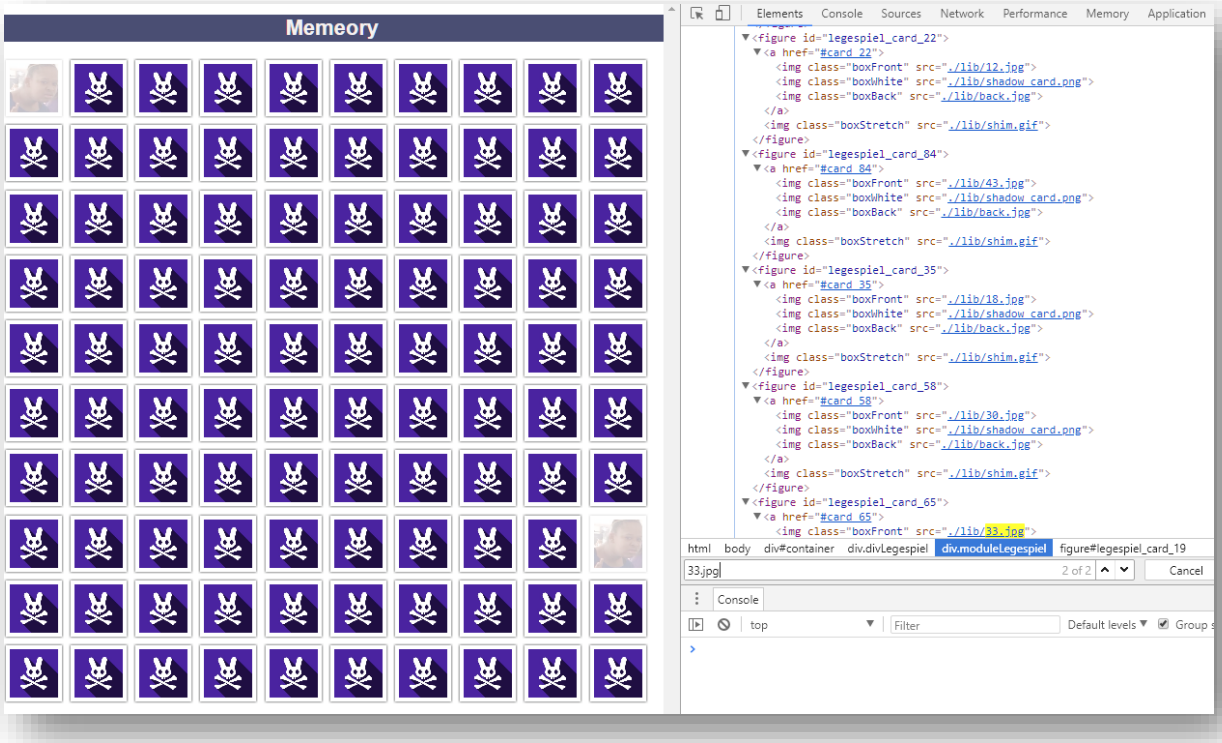
pony code google fu. i stumbled over punycode after a while - but the given text is missing the prefix. when we add it, we can decode it:



pass: gin tonic

04 MEMEORY

using the developer tools in chrome, we can simply discover which image is behind the memory cards:



i expanded all nodes recursively and then used the search to highlight the correct cards. this was a manual process, which i did during a tv session. i was too lazy to code something for that.

05 SLOPPY & PASTE

in this mobile challenge we should just copy paste the code in the moble app, but the text gets truncated.

like every year, i dumped the app with Clutch2 on my jailbroken iphone and checked out the files from the app.

in the www folder i opened the file challenge05.html in ultraedit, copied the base64 string to cryptool, decoded and saved it as png. task solved.

06 COOKING FOR HACKERS

after we decode all strings from base64 to ascii, we get a new string.

saltoilt7w2gntdo.onion

onion is a TOR url.

using the tor browser we can open this address and find the hidden egg.

07 JIGSAW

i googled for tools to solve jigsaws automatically, but didnt find something good until i added "python" to my search query:

https://github.com/nemanja-m/gaps

this script will solve the jigsaw almost 100% for us. enough to read the solution.

password: goodsheepdontalwayswearwhite

08 DISCO EGG

this is a nice challenge - looks really fancy. but we need somehow a QR code, which is black and white.

using the developer tools i discovered, that the cells either have the class white or black, but never both:

```
▼<tr>
  <td class="darkgreen orange purple black lightgrey green cyan red blue
  yellow" style="background-color: rgb(0, 100, 18);"></td>
  <td class="lightgrey orange mediumgrey yellow white tan" style=
  "background-color: rgb(255, 100, 3);"></td>
  <td class="yellow orange tan darkgrey black purple" style="background-
  color: rgb(204, 186, 28);"></td>
  <td class="black red purple tan darkgrey" style="background-color:
  rgb(71, 0, 165);"></td>
  <td class="lightgrey yellow tan green red black darkgreen mediumgrey"
  style="background-color: rgb(221, 9, 7);"></td>
  <td class="lightgrey yellow magenta darkgrey green white mediumgrey
  tan" style="background-color: rgb(144, 113, 58);"></td>
```

i created a local copy of this webpage with wget and then search replaced all colors until i only had black and white left. again with search replace i changed the "class" to "bgcolor" and then viewed the webpage.

using cellspacing=0 and cellpadding=1 for the html table will give a nice QR.

09 DIAL TRIAL

i did of course reverse engineer the mobile app and found out, that it just plays an mp3 file which contains DTMF tones.

using this webpage i converted them to numbers:

https://unframework.github.io/dtmf-detect/

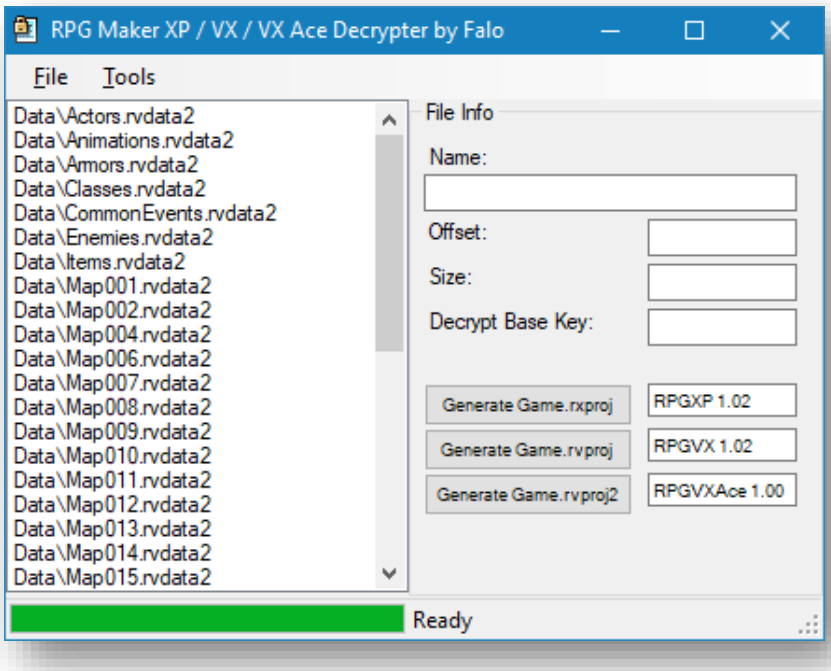
4*7#2*6#1*2#2*5#2*3#3*6#2*6#2*6#3*6#2*5#3*4#1*2

this is the same encoding like in challenge 1. just use the numbers on the keypad and read the corresponding letters, which gives: **snakeonnokia**

10 LEVEL TWO

i tried to hack the save game like i did in the teaser, but that only gave me one password which didnt solve the challenge. then i tried to decrypt the rgss3a file with this decrypter:

http://www.mediafire.com/file/3ps81u5opyasn07/All+decrypter.rar



with sysinternals strings utillity i extracted all strings of these rvdata2 files:

strings *.rvdata2 >strings.txt

i noticed multiple hex strings:

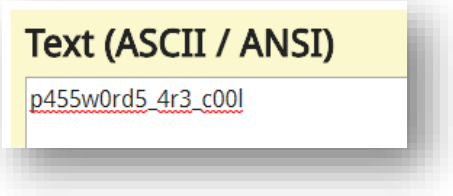
7034353577307264355f3472335f6330306c

7034353577307264355f3406033b5749114c

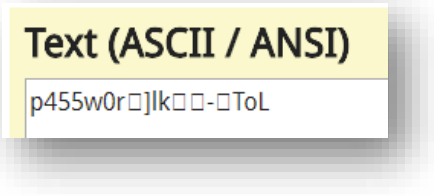
7034353577307264355f052d066b15035433

7034353577307210 5d6c6b05032d0f546f4c

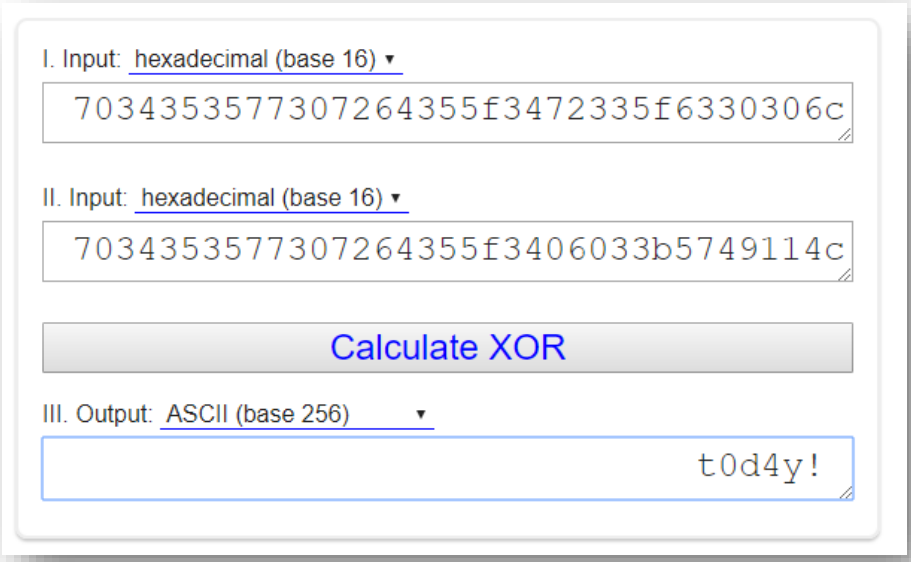
but only one was good ascii (that is the password that you can get by cheating the savegame egg)



the others looked somehow broken



here i was stuck for some time, because i didnt understand the hint "you must combine them". but then suddenly i did what a hacker does - xoring the values! http://xor.pw/ helps a lot and even displays ascii.



password: **1_54v3d_th3_w0rld_t0d4y!**

11 DE EGG YOU MUST

a password protected zip. ok - lets crack it. using zip2john and then john, i discovered the password really quickly: **thumper**

now we have multiple files that seems to be splitted - lets copy them together. we can use cat (on linux or MacOS) or copy /b egg* >egg.bin on windows.

using the file command we can find out that it is a m4v file, lets rename it then.

but its only a video of a cat playing piano. "de egg we must". smells like steganography, but what the heck is de egg?

if you google deegg steganography (without the space) you will find it: Deegger Embedder.
http://www.zasi.org/DeEgger-Embedder.php

using the m4v file with it, will reveal the egg. without the proper extension it didnt work for me.

12 PATIENCE

from the mobile app again i checked the html file of this challenge:

```
<script>
  hash = 'genesis';
  count = 100000;
  setTimeout( function() { document.location.href = 'ps://count?h=' + hash + '&c=' + count; } , 1000);
  function countFeedback(jsonString) {
      var json = JSON.parse(jsonString);
        if (json) {
          hash = json.h;
          count = json.c;
          $('#count').text(count);
          if (count == 0) {
              document.getElementById('flag').setAttribute('src', 'https://hackyeaster.hacking-lab.com/hackyeaster/images/eggs/'+hash+'.png');
          } else if (count > 0) {
              setTimeout( function() { document.location.href = 'ps://count?h=' + hash + '&c=' + count; } , 3000);
          }
        }
    }
  </script>
```

it calculates a hash on every timer event. of course i tried to trick the counter, but this didnt work. lowering the timeout also didnt really make it faster.

from the javascript we can see that it sends a hash and the counter to the app.

```
id __cdecl -[MainViewController handleCount:forHash:](MainViewController *self, SEL a2, id a3, id a4)
{
  id v4; // x21
  id v5; // x19
  id result; // x0
  int v7; // w20
  struct objc_object *v8; // x0
  id v9; // x0

  v4 = a4;
  v5 = a3;
  result = 0LL;
  if ( a3 && a4 )
  {
    v7 = (unsigned __int64)objc_msgSend(a3, "intValue");
    v8 = (struct objc_object *)objc_msgSend(v4, "stringByAppendingString:", v5);
    v9 = ((id (__cdecl *)(Util_meta *, SEL, id))objc_msgSend)((Util_meta *)&OBJC_CLASS___Util, "sha1hex:", v8);
    if ( v7 & 0x80000000 )
      result = 0LL;
    else
      result = (id)objc_msgSend(
                     &OBJC_CLASS___NSString,
                     "stringWithFormat:",
                     CFSTR("{ \"h\":\"%@\", \"c\":\"%d\" }"),
                     v9,
                     (unsigned int)(v7 - 1));
  }
  return result;
}
```

and in the app it does calculate sha1hex based on the input from the javascript. so we have genesis+100000 hashed and then this hash+99999 hashed and so on.

with a simple python script we can find the correct path to the egg:

```python
import hashlib

h = hashlib.sha1('genesis'+'100000')
print h.hexdigest()

for i in range(99999,0,-1):

    h = hashlib.sha1(h.hexdigest()+str(i))


print 'https://hackyeaster.hacking-lab.com/hackyeaster/images/eggs/'+h.hexdigest()+'.png'
```

13 SAGITTARIUS...

this was the last challenge i have solved. it was rather tricky, because it required some different knowhow - not really computer related.

the file itself can be opened in google earth and it will show some stars at the sky.

using the coordinates from the xml file, i recreated the circle in excel - see screenshot below.

to get a QR code, we need to map the coordinates from the circle to a square. some google searches gave me this interesting page:

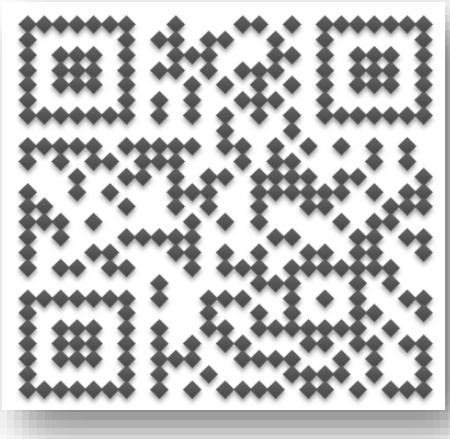
http://squircular.blogspot.ch/2015/09/elliptical-arc-mapping.html

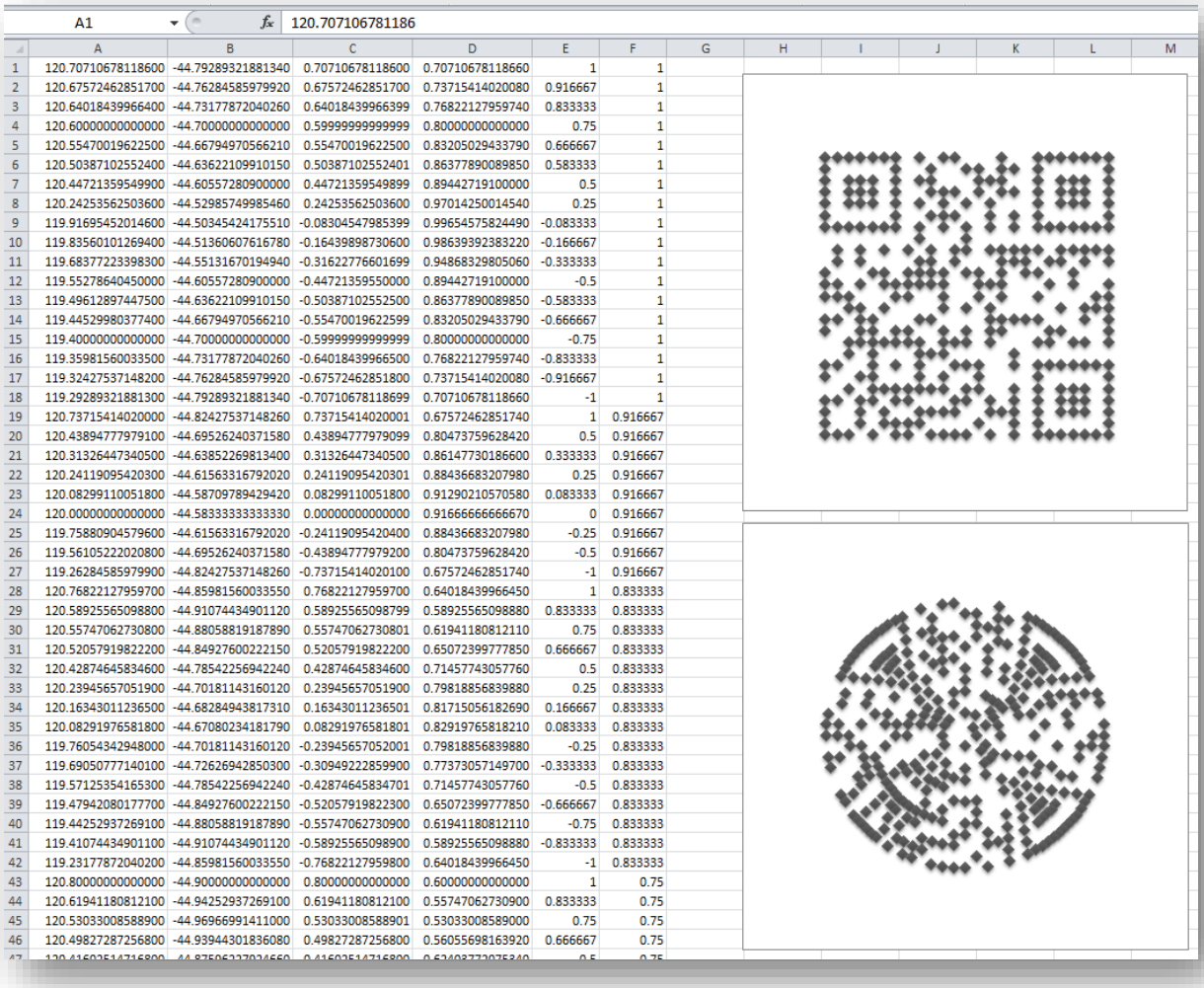
i tried this code with the coordinates from the kml, but it didnt work. reading the paper https://arxiv.org/ftp/arxiv/papers/1509/1509.06344.pdf reveals, that we need the coordinates in the range -1 to 1.

we can achieve that by finding the center of the circle and then substract the values from the coordinates. the middle is 120.0, -45.5.

i did this in excel with a formula and then used these coordinates in the C code from the article above - but yeah, i could have implemented the formula in VBA directly.

now ive got new coordinates and using those in excel to draw a point diagram, will give us a QR code. but its mirrored. i had to tweak excel datapoint format a little and then flip the qr code in paint.net to finally scan it. phew.





A1 | 120.707106781186

	A	B	C	D	E	F
1	120.70710678118600	-44.79289321881340	0.70710678118600	0.70710678118660	1	1
2	120.67572462851700	-44.76284585979920	0.67572462851700	0.73715414020080	0.916667	1
3	120.64018439966400	-44.73177872040260	0.64018439966399	0.76822127959740	0.833333	1
4	120.60000000000000	-44.70000000000000	0.59999999999999	0.80000000000000	0.75	1
5	120.55470019622500	-44.66794970566210	0.55470019622500	0.83205029433790	0.666667	1
6	120.50387102552400	-44.63622109910150	0.50387102552401	0.86377890089850	0.583333	1
7	120.44721359549900	-44.60557280900000	0.44721359549899	0.89442719100000	0.5	1
8	120.24253562503600	-44.52985749985460	0.24253562503600	0.97014250014540	0.25	1
9	119.91695452014600	-44.50345424175510	-0.08304547985399	0.99654575824490	-0.083333	1
10	119.83560101269400	-44.51360607616780	-0.16439898730600	0.98639392383220	-0.166667	1
11	119.68377223398300	-44.55131670194940	-0.31622776601699	0.94868329805060	-0.333333	1
12	119.55278640450000	-44.60557280900000	-0.44721359550000	0.89442719100000	-0.5	1
13	119.49612897447500	-44.63622109910150	-0.50387102552500	0.86377890089850	-0.583333	1
14	119.44529980377400	-44.66794970566210	-0.55470019622599	0.83205029433790	-0.666667	1
15	119.40000000000000	-44.70000000000000	-0.59999999999999	0.80000000000000	-0.75	1
16	119.35981560033500	-44.73177872040260	-0.64018439966500	0.76822127959740	-0.833333	1
17	119.32427537148200	-44.76284585979920	-0.67572462851800	0.73715414020080	-0.916667	1
18	119.29289321881300	-44.79289321881340	-0.70710678118699	0.70710678118660	-1	1
19	120.73715414020000	-44.82427537148260	0.73715414020001	0.67572462851740	1	0.916667
20	120.43894777979100	-44.69526240371580	0.43894777979099	0.80473759628420	0.5	0.916667
21	120.31326447340500	-44.63852269813400	0.31326447340500	0.86147730186600	0.333333	0.916667
22	120.24119095420300	-44.61563316792020	0.24119095420301	0.88436683207980	0.25	0.916667
23	120.08299110051800	-44.58709789429420	0.08299110051800	0.91290210570580	0.083333	0.916667
24	120.00000000000000	-44.58333333333330	0.00000000000000	0.91666666666670	0	0.916667
25	119.75880904579600	-44.61563316792020	-0.24119095420400	0.88436683207980	-0.25	0.916667
26	119.56105222020800	-44.69526240371580	-0.43894777979200	0.80473759628420	-0.5	0.916667
27	119.26284585979900	-44.82427537148260	-0.73715414020100	0.67572462851740	-1	0.916667
28	120.76822127959700	-44.85981560033550	0.76822127959700	0.64018439966450	1	0.833333
29	120.58925565098800	-44.91074434901120	0.58925565098799	0.58925565098880	0.833333	0.833333
30	120.55747062730800	-44.88058819187890	0.55747062730801	0.61941180812110	0.75	0.833333
31	120.52057919822200	-44.84927600222150	0.52057919822200	0.65072399777850	0.666667	0.833333
32	120.42874645834600	-44.78542256942240	0.42874645834600	0.71457743057760	0.5	0.833333
33	120.23945657051900	-44.70181143160120	0.23945657051900	0.79818856839880	0.25	0.833333
34	120.16343011236500	-44.68284943817310	0.16343011236501	0.81715056182690	0.166667	0.833333
35	120.08291976581800	-44.67080234181790	0.08291976581801	0.82919765818210	0.083333	0.833333
36	119.76054342948000	-44.70181143160120	-0.23945657052001	0.79818856839880	-0.25	0.833333
37	119.69050777140100	-44.72626942850300	-0.30949222859900	0.77373057149700	-0.333333	0.833333
38	119.57125354165300	-44.78542256942240	-0.42874645834701	0.71457743057760	-0.5	0.833333
39	119.47942080177700	-44.84927600222150	-0.52057919822300	0.65072399777850	-0.666667	0.833333
40	119.44252937269100	-44.88058819187890	-0.55747062730900	0.61941180812110	-0.75	0.833333
41	119.41074434901100	-44.91074434901120	-0.58925565098900	0.58925565098880	-0.833333	0.833333
42	119.23177872040200	-44.85981560033550	-0.76822127959800	0.64018439966450	-1	0.833333
43	120.80000000000000	-44.90000000000000	0.80000000000000	0.60000000000000	1	0.75
44	120.61941180812100	-44.94252937269100	0.61941180812100	0.55747062730900	0.833333	0.75
45	120.53033008588900	-44.96966991411000	0.53033008588901	0.53033008589000	0.75	0.75
46	120.49827287256800	-44.93944301836080	0.49827287256800	0.56055698163920	0.666667	0.75

14 SAME SAME...

this challenge got me lost quite some time, because i thought its about magic hashes. actually magic hashes would work (because of the == operator), but i was not able to create a file that gave me one.

but recently google showed that they can create a sha1 collision https://shattered.io/

with this attack we can create two pdf's that will have the same sha1 and that contain QR codes for Hacky Easter and Hackvent: here is a website that does it for us: https://alf.nu/SHA1

its not obvious, but this php qr library can also read PDF's.

15 MANILA GREETINGS

cards? deck? sounds pretty much like the solitaire cipher. i just spent too much time using the wrong tools.

first i converted the deck into numbers using this simple python script (JK is Joker A and JB Joker B)

```
he18 — -bash — 111×35
MacBook-Pro:he18 kolbicz$ python solitaire.py
21,42,20,16,2,44,14,6,46,19,53,26,38,50,11,33,29,35,48,47,9,40,30,8,3,39,27,45,32,49,52,1,23,25,12,54,51,43,22,
41,5,37,36,4,10,18,34,28,15,24,7,13,31,17
MacBook-Pro:he18 kolbicz$
```

then its easy with this webpage: http://www.phys.uconn.edu/~aldridge/solitaire/solitaire.htm#fields

Passkey

Key Shuffle

Starting Deck

11,25,22,24,46,10,28,34,49,27,42,2,19,31,7,44,37,36,15,52,1,23,38,54,17,53,33,3,20,14,21,51,43,16,18,50,13,35,6,5,39,40,29,41,4,48,47,9,26,45,32,30,8,12

Message Length: 26 Generate Keystream

Keystream

MLDPKYCOPRPRCBGVVJUAVPTLVB

Cleartext

THEPASSWORDISCRYPTONOMICON

Ciphertext

GTIFLRVLEJTAVEYULDJOKCCOKP

Encrypt Decrypt GO GO GADGET

16 GIT CLOAK --HARD

oh jeez - again a git challenge. im really a n00b at git and i have tried a lot of things - including a lot of gui tools and scripts, but i only found fake flags.

this made me think, that maybe the flag isnt even referenced in this repo, but only available in the binary files. but how can we extract it?

after some google fu i found out, that we can display the contents of git objects with "git cat-file -p":

https://githowto.com/git_internals_working_directly_with_git_objects

now we just need to supply the correct hashes to extract all the files. we can find these by listing all objects with ls-R. a quick ultraedit search/replace/copy job created all possible combinations and the egg was hidden in this command:

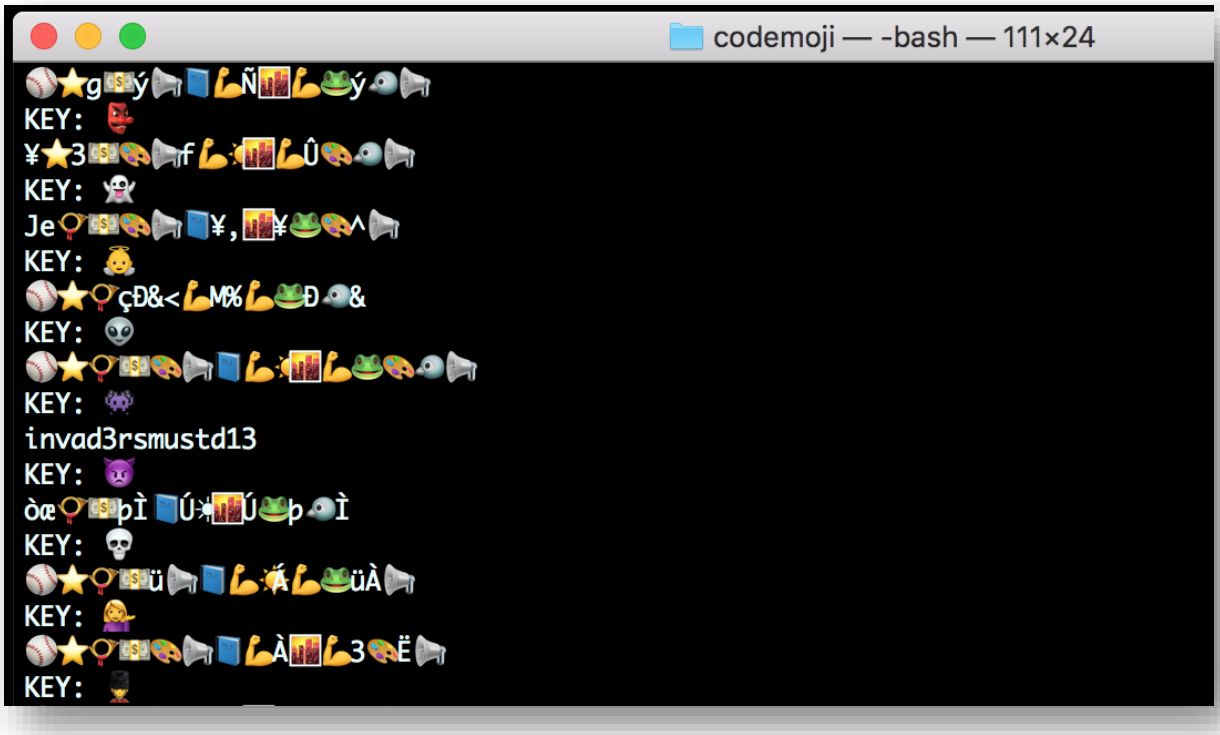
```
git cat-file -p dbab6618f6dc00a18b4195fb1bec5353c51b256f >egg.png
```

17 SPACE INVADERS

codemoji encryption? what? i had no idea how to decrypt this, but i have found the source code of it here

https://github.com/mozilla/codemoji

using node, i ran a bruteforcer coded in js, which decrypted the ciphertext with all emoji's that i found in another example js file in this project.



oh no. the key was already visible on the challenge page(alien emoji) - a good guess already would have solved it. ha ha.

18 EGG FACTORY

we are given the file A.8xp which turns out to be a program for the TI-series calculators. using this online decompiler/compiler/emulator we can analyze and solve this challenge:-

https://www.cemetech.net/sc/

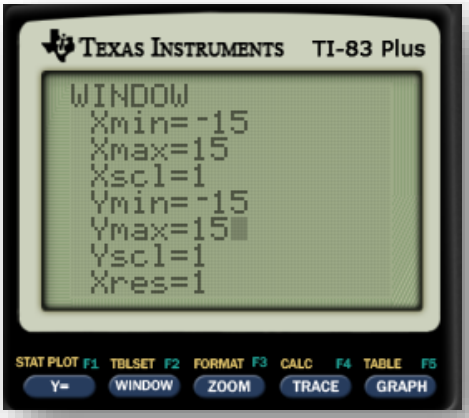
i just lost quite some time figuring out how the calculator works and which ROM i have to use.

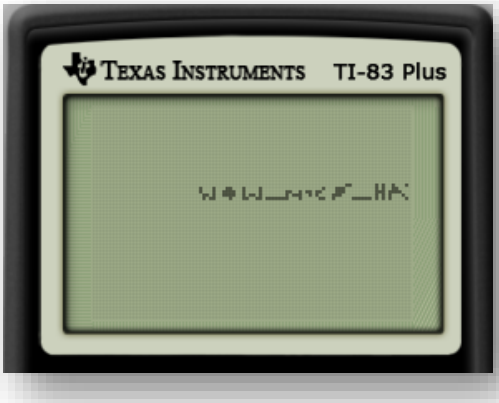
following the basic code showed a program that asks for numbers and lets you login or check a passcode. the number 3 reveals a secret, but it will fail, because not all variables were set. we need to find the password - which is based on the code only a 8956 xor 9191 = 0283

```
If Str0="2":Then
    Disp "ENTER PASSWORD"
    Input "",Str5
    DelVar E1->C
    8956->N
    expr(Str5)->M
    While N>0 or M>0
        .5int(N->N
        .5int(M->M
        E+C(fPart(N) xor fPart(M->E
        C2->C
    End
    If E=9191:Then
        Disp "Successful :)"
```

how to make it print the flag:

- load a TI-83/84 ROM
- select `TI-83+/TI-84+ TI-BASIC in the source code view`
- `click Test`
- `turn calculator On (left bottom)`
- `press PRGM`
- `press 1, enter`
- `press 2, enter`
- `0283`
- `press again PRGM`
- `press 1, enter`
- `press 3, enter`
- it will print a message, but its too big for the screen
- use WINDOW to change the Xmin, Xmax, etc





almost not readable, but i ran it in a different emulator and a good guess also helped. WOW_N1CE_HAX

19 VIRTUAL HEN

this challenge was somehow weird, because its blind brute force with no hints. from the algorithm we can detect TEA encryption and the keyspace is limited. also the passsword length is fixed - but still - on my PC i only got about one character changed over night for all eight chars!

bruteforcing all possible values would take multiple DAYS - in the worst case even a MONTH.

if you good guess here and start with G as beginning letter (thinking about EGG) then its solveable in just a few minutes. i dont really got the idea of this challenge - bruteforce such a long time or a lucky punch?

anyway, i coded a C program, which decrypts one block and compares it to a valid PNG header, because bruteforce with the program itself would take even longer. wordlists didnt work either.

check the source code for more information.

```
STHBEGGGSTHBEGGG
SGALTGGGSGALTGGG
S[KEUGGGS[KEUGGG
SKN[HGGGSKN[HGGG
SM[YKGGGSM[YKGGG
SSPUQGGGSSPUQGGG
SAGI@GGGSAGI@GGG
S^Y^AEGGS^Y^AEGG
SXOVNEGGSXOVNEGG
SBJPDEGGSBJPDEGG
SCUTFEGGSCUTFEGG
key found : H@CKYEGG
```

20 ARTIST: NO NAME YET

this challenge gives us two files. in the pdf we can find a hint using http://www.extractpdf.com

Bilder | Text | Fonts | Metadata

Ergebnis als Datei herunterladen

```
Okay, let's do the information exchange as we coordinated. First let me
tell you: hiding informations in a MIDI file will be popular soon! We should
only do it this way to stay covered. MIDI hiding is just next level - wow! So,
here are all informations you need to find the secret: Trackline: Can't remember now,
but you'll find it. It's kinda quiet this time, because of the doubled protection
algorithm! Characters: 0 - 127 (by the way: we won't need the higher ones
ever…)Let's go!
I'm very exited for the lyrics that you will create
for this masterpiece.
Best wishes, your friend

LuckyTail
```

ok - so we know its about hiding information in midi files. this is steganography. the hint about 0-127 tells us its about the volume midi parameter (good i did produce a lot of electronic music in my life).

we most likely need to extract midi events - but whats the best way to do that? i was too lazy to learn about all the details of the midi format and just used https://www.anvilstudio.com/ to save the midi events as txt and then regex the volume out of it.

i coded a python script that gets all midi events from the different tracks (that i saved manually to txt) and converted them to ascii.

```python
import re

x=""

with open("C:\\Users\\administrator\\Desktop\\hacky2018\\3.txt") as f:
    for line in f:
        s = re.compile("vol:\s\d{1,3}")
        m = s.search(line)
        if m:
            x += chr(int(m.group(0).replace("vol: ", "")))
print x
```

i ran this for all files and got one interesting hit:

```
C:\Users\administrator\Desktop\hacky2018>python solver.py
-.-. --- -- .--. --- ... . -.. -... -.-- -.. .--- ... .--. ----- ----- -. -.--
```

this is morse code and gives:

Input:

```
-.-. --- -- .--. --- ... . -.. -... -.-- -.. .--- ... .--. ----- ----- -. -.--
```

Output:

```
COMPOSEDBYDJSP00NY
```

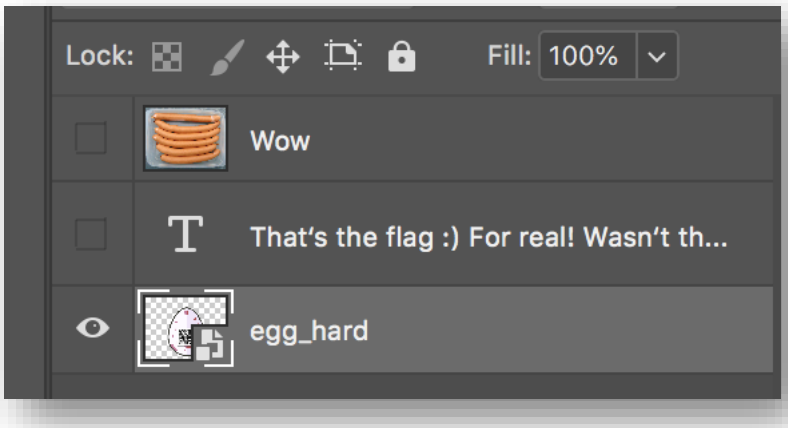
nice challenge!

21 HOT DOG

the zip file contains a jpg image, but something is not correct about that. using the file command reveals, that it is actually a tiff.

```
5k-iMac:Downloads christoph$ file hotdog.jpg
hotdog.jpg: TIFF image data, little-endian, direntries=27, height=2067, bps=338, compression=none, PhotometricIntepretation=RGB, desc
ription=*Don't forget to delete this*, manufacturer=Panasonic, model=DMC-FZ18, orientation=upper-left, width=2700
```

when we rename it to .tiff and then open it in photoshop, we get additional layers.



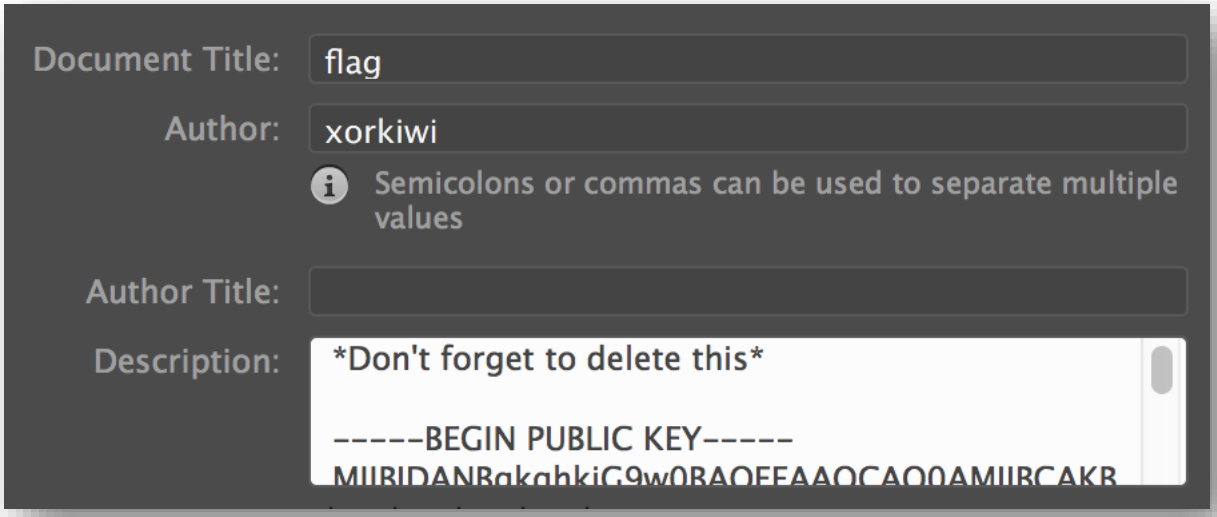
sadly the egg is not the flag, but it contains ciphercode:

Decode Succeeded

Raw text	Arf3ThIY8VQg2GUd249wzDYi7CXqTST+9g4Q7bbT2eF+mD2KB+6oi3rVSY/eZ6/onNBNYPo2BPqIVEbL35G62pIHvab
Raw bytes	42 d4 17 26 63 35 46 84 95 93 85 65 16 73 24 75 56 43 23 43 97 77 a4 45 96 93 74 35 87 15 45 35 42 b3 96 73 45 13 76 26 25 43 26 54 62 b6 d2 03 24 b7 1f c3 90 71 bd a4 cd c9 59 4d 64 bd 95 68 d8 bd bd b9 39 09 39 65 41 bc c9 09 41 c5 25 59 15 88 80 ce d0 78 91 04 58 70 49 48 76 61 62 47 63 72 59 6f 73 47 43 70 59 68 69 7a 36 45 59 6e 61 6d 6e 4e 50 72 48 64 7a 6d 45 4f 73 38 6c 43 52 77 31 63 32 50 65 38 6b 6c 34 31 46 48 30 75 64 37 74 42 6e 36 71 44 2f 73 74 6e 5a 66 47 6b 63 62 65 49 72 6a 61 53 69 49 59 53 76 65 48 53 0a 00 ec 11 ec 11 ec 11 ec 11 ec 11 ec 11 ec 11 ec 11

Arf3ThIY8VQg2GUd249wzDYi7CXqTST+9g4Q7bbT2eF+mD2KB+6oi3r
VSY/eZ6/onNBNYPo2BPqIVEbL35G62pIHvabGcrYosGCpYhiz6EYnam
nNPrHdzmEOs8lCRw1c2Pe8kl41FH0ud7tBn6qD/stnZfGkcbeIrjaSi
IYSveHS

in the metadata we can also find a public key:



Don't forget to delete this

-----BEGIN PUBLIC KEY-----
MIIBIDANBgkqhkiG9w0BAQEFAAOCAQ0AMIIBCAKBgQTMleqB9nvRKhTnR4/2BDDU
g5hkjbRQygvrZWDATbC9rXxCAqaegim2XUlD8yVxYkyzJZxmAYba7qLTe3bctocM
L7GXdMf3kQiVLPigN2auEiPFreWZvZ/b4FzcvOhh+SprypAkYn9SapTyGzLdpYdD
TyoWFRT7QgEhlsDGcncsXQKBgQCVbdUZa5uQ7O9bgu2WPvUwwvul+ZK5gOZCF299
1QRa/rdDHKyYiUxxZXjemxGICxvoC698wVvmVqzG/sCT+iLArlh4OmSHgyd1yjcA
CWmsffHYLvsl3tnN9Jiu5qzN6aGthHjK/424NK0RkfjUdmnQydYN/MqfS7c+AkfJ
QWV/9w==
-----END PUBLIC KEY-----

ok - we have a public key and ciphertext. what we need now is the private key, but its not in the challenge. lets see what we can extract from this public key with openssl:

```
5k-iMac:he18 christoph$ openssl rsa -pubin -inform PEM -text -noout < public.key
Public-Key: (1027 bit)
Modulus:
    04:cc:95:ea:81:f6:7b:d1:2a:14:e7:47:8f:f6:04:
    30:d4:83:98:64:8d:b4:50:ca:0b:eb:65:60:c0:4d:
    b0:bd:ad:7c:42:02:a6:9e:82:29:b6:5d:49:43:f3:
    25:71:62:4c:b3:25:9c:66:01:86:da:ee:a2:d3:7b:
    76:dc:b6:87:0c:2f:b1:97:74:c7:f7:91:08:95:2c:
    f8:a0:37:66:ae:12:23:c5:ad:e5:99:bd:9f:db:e0:
    5c:dc:bc:e8:61:f9:2a:6b:ca:90:24:62:7f:52:6a:
    94:f2:1b:32:dd:a5:87:43:4f:2a:16:15:14:fb:42:
    01:21:22:c0:c6:72:77:2c:5d
Exponent:
    00:95:6d:d5:19:6b:9b:90:ec:ef:5b:82:ed:96:3e:
    f5:30:c2:fb:88:f9:92:b9:80:e6:42:17:6f:7d:d5:
    04:5a:fe:b7:43:1c:ac:98:89:4c:71:65:78:de:9b:
    11:88:0b:1b:e8:0b:af:7c:c1:5b:e6:56:ac:c6:fe:
    c0:93:fa:22:c0:ac:88:78:3a:64:87:83:27:75:ca:
    37:00:09:69:ac:7d:f1:d8:2e:fb:25:de:d9:cd:f4:
    98:ae:e6:ac:cd:e9:a1:ad:84:78:ca:ff:8d:b8:34:
    ad:11:91:f8:d4:76:69:d0:c9:d6:0d:fc:ca:9f:4b:
    b7:3e:02:47:c9:41:65:7f:f7
```

1027bit? can we crack it? not really, but i have an idea. lets convert the Modulus to decimal and then feed it to factordb.com.

Search | Sequences | Report results | Factor tables | Status | Downloads

8627421546422318392454906523054478110039391888898791606019168250989193764992396580832864 Factorize! (?)

Result:

status (?)	digits	number
FF	309 (show)	$8627421546...77_{<309>} = 2178799522...13_{<155>} \cdot 3959713345...29_{<155>}$

we have a hit! we just cracked RSA-1027. here are P and Q:

2178799522695817282946788820649068111400321304485606703112899813574211262513425563577235208574330894946656793478545800265281621740859513523358040060627841 3

3959713345148733427795095053000386195288511240450061829870229990556683111766647009803589047757206821068397128010430418458046941744065644356719673321695092 9

now that we know that, we even can use RsaCtfTool to make a proper private key:

```
5k-iMac:RsaCtfTool christoph$ python RsaCtfTool.py --publickey ../public.key --private
-----BEGIN RSA PRIVATE KEY-----
MIICOgIBAAKBgQTMleqB9nvRKhTnR4/2BDDUg5hkjbRQygvrZWDATbC9rXxCAqae
gim2XUlD8yVxYkyzJZxmAYba7qLTe3bctocML7GXdMf3kQiVLPigN2auEiPFreWZ
vZ/b4FzcvOhh+SprypAkYn9SapTyGzLdpYdDTyoWFRT7QgEhIsDGcncsXQKBgQCV
bdUZa5uQ709bgu2WPvUwwvuI+ZK5gOZCF2991QRa/rdDHKyYiUxxZXjemxGICxvo
C698wVvmVqzG/sCT+iLArIh4OmSHgyd1yjcACWmsffHYLvsl3tnN9Jiu5qzN6aGt
hHjK/424NK0RkfjUdmnQydYN/MqfS7c+AkfJQWV/9wIgcpt4HD9KQu5nAtq5QcXb
grmSvMFpX3nlUY5uWNkI1QcCQQGgAYCTfd8+2Gu44jZ007I4qUHjj7yrSXimPyvJ
ep9WrUhXkM5fuwVW7XIozQN1AjVAbuJ48vTO5zzdfWH4PXMNAkEC9ArBsCYtbG8Z
Nb6mpS0JXAzYPfUn3pId4RPiq8ib+6lU1b0j0rWgy9KGQBSpXsSc2VF2aIgZBneL
+3JLfTSKkQIgcpt4HD9KQu5nAtq5QcXbgrmSvMFpX3nlUY5uWNkI1QcCIHKbeBw/
SkLuZwLauUHF24K5krzBaV955VGOljZCNUHAkEBCfcw78XS1mJbJ+DjtNfD+sAx
ARp2YCiBWeoWbvRDzxTl5nAHgC8EvHqc8/ffHSSaUywFHM18kYCnDvx1MMaNEA==
-----END RSA PRIVATE KEY-----
```

22 BLOCK

this challenge

information a

but actually, we can use RsaCtfTool to decrypt the file directly, without even looking at the private key.

the flag is base64 and we need it as binary first. i ran "echo Arf3ThIY8VQg2GUd249wzDYi7CXqTST+9g4Q7bbT2eF+mD2KB+6oi3rVSY/eZ6/onNBNYPo2BPqIVEbL35G62pIHvabGcrYosGCpYhiz6EYnamnNPrHdzmEOs8lCRw1c2Pe8kl41FH0ud7tBn6qD/stnZfGkcbeIrjaSiIYSveHS | base64 --decode >flag.bin" and then we can decrypt the flag with:

python ./RsaCtfTool/RsaCtfTool.py --publickey public.key --uncipherfile flag.bin

```
5k-iMac:he18 christoph$ python ./RsaCtfTool/RsaCtfTool.py --publickey public.key --uncipherfile flag.bin
[+] Clear text : Great job haxxor, here's your flag: {b3w4r3_0f_c0n71nu3d_fr4c710n5}
```

22 BLOCK JANE

this challenge gives us a service which only responds with yes or no. we also have a ciphertext and information about AES and blocks.

must be about the padding oracle then - the only difference for me is that its not http based. ive done such challenges in the past, but not socks based.

for this kind of attack there are already multiple tools and libraries. my google fu even gave me a full solution in python, which we only need to adapt a little.

https://losfuzzys.github.io/writeup/2015/10/06/tumctfteaser2015-pudel/

this is exactly what we need. only the ciphertext, return code and the server:port has to be changed. then we can run it for a while (it really takes its time) and we will see the cleartext:

```
HOST = "whale.hacking-lab.com"
PORT = 5555

encdata = [0xe3, 0x43, 0xf4, 0x26, 0x04, 0xca, 0x58, 0xa7, 0x31, 0xad, 0xbf, 0x10, 0xb3, 0x76, 0xee, 0x33,
           0xaa, 0x94, 0x49, 0x26, 0xcd, 0xf9, 0x54, 0x40, 0x0d, 0x86, 0xee, 0x4f, 0x6e, 0x35, 0x77, 0x4e,
           0xc5, 0x10, 0xfe, 0x57, 0x67, 0xba, 0xba, 0x99, 0xa3, 0xed, 0x28, 0xfa, 0x26, 0xdc, 0x99, 0xb6,
           0xc1, 0xda, 0xdd, 0x08, 0x7e, 0x4c, 0xee, 0x27, 0xe4, 0x55, 0x07, 0x00, 0x52, 0x76, 0xc1, 0x0f,
           0xd9, 0xc1, 0x5f, 0x27, 0xd3, 0x48, 0x1a, 0x92, 0xf3, 0x4d, 0xd4, 0x64, 0x77, 0xf7, 0xbe, 0x3c]
```

```
Terminal - root@HLKali: /home/hacker/blockjane
File  Edit  View  Terminal  Tabs  Help
    00000020
[*] 00000000  df c0 c3 16  60 52 f0 39  fa 4b 19 1e  4c 68 df 11  |····|`R 9| K··|Lh··|
    00000010  d9 c1 5f 27  d3 48 1a 92  f3 4d d4 64  77 f7 be 3c  |··_'|·H··| M d|w· <|
    00000020
[*] 00000000  de c0 c3 16  60 52 f0 39  fa 4b 19 1e  4c 68 df 11  |····|`R 9| K··|Lh··|
    00000010  d9 c1 5f 27  d3 48 1a 92  f3 4d d4 64  77 f7 be 3c  |··_'|·H··| M d|w· <|
    00000020
[*] 00000000  dd c0 c3 16  60 52 f0 39  fa 4b 19 1e  4c 68 df 11  |····|`R 9| K··|Lh··|
    00000010  d9 c1 5f 27  d3 48 1a 92  f3 4d d4 64  77 f7 be 3c  |··_'|·H··| M d|w· <|
    00000020
[*] 00000000  dc c0 c3 16  60 52 f0 39  fa 4b 19 1e  4c 68 df 11  |····|`R 9| K··|Lh··|
    00000010  d9 c1 5f 27  d3 48 1a 92  f3 4d d4 64  77 f7 be 3c  |··_'|·H··| M d|w· <|
    00000020
[*] 00000000  db c0 c3 16  60 52 f0 39  fa 4b 19 1e  4c 68 df 11  |····|`R 9| K··|Lh··|
    00000010  d9 c1 5f 27  d3 48 1a 92  f3 4d d4 64  77 f7 be 3c  |··_'|·H··| M d|w· <|
    00000020
[*] assword is: oracl3in3delphi

    See you soon!

    Jane

    \x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e
```

23 RAPBID LEARNING

this challenge was different and i spent quite some time to do my researches. it cannot be solved manually - believe me, i have tried it.

google for machine learning and python revealed some complicated articles and some libraries which can help on this topic. the main idea is:

we have data to train with. then we are given 5000 datasets, which are missing one attribute (g00d) - we must use machine learning, to predict these values.

i have made two scripts - one that does get the training data and saves it to a file (in the correct order).

then my learning script will get the 5000 datasets and tries to predict the missing values with the help of sklearn.ensemble RandomForestRegressor (because LinearRegression had a rather bad rate) algorithm. most of the code i have found here:

https://www.dataquest.io/blog/machine-learning-python/

i played with some sample datasets to understand how it works and then adapted it for this challenge. sometimes you need not a lot of sample data to reach 100% predictions, but in this case i had sampled 150 datasets to reach 100%:

```
session_id: 43e26bef6ec5bdd98f67853c9fd330ef85c2f4fc
train.shape:
(150, 9)
test.shape:
(5000, 9)
head(20):
    w31ght  l3ngth  g3nd3r   c0l0r      n4m3  t411  sp00n  ag3  g00d
0        2      52  female   brown    Evelyn    11     13    3  True
1        2      52    male     red      John    11     13    2  True
2        5      53  female     red      Mary     9     10    7  True
3        4      49    male   brown      John    11      8    1  True
4        2      50    male   brown   Bradley    11     10    7  True
5        2      44    male     red    Wesley    11     14    2  True
6        2      52  female     red   Rosalyn    11     13    5  True
7        2      46  female   black       Hue    11     13    4  True
8        3      54    male     red   Charles    11     10    4  True
9        2      52  female     red  Kathleen    10     14    4  True
10       2      40  female   brown    Billie    11     13    5  True
11       2      40    male     red    Dennis    10     14    6  True
12       2      41  female   brown   Latoria    11     14    1  True
13       2      41  female     red     Pearl    10     10    3  True
14       2      42    male   brown   Michael    10     13    4  True
15       2      49  female     red  Charisse    10     12    5  True
16       2      53  female   black       Meg    11     11    4  True
17       4      46  female   brown   Nichole     8      8    1  True
18       4      53    male  purple     James    11     10    4  True
19       2      47  female   brown   Frances    10     14    0  True
mean_squared_error:
0.09683872
200 OK
SCORE: MTAwLjA1 - lolnice! - I'll tell my guys to set up your reward for this shift at
 /reward, don't forget to bring your cookie!
200 OK
```

after we have the results, we must make sure that we send it in a json integer array (with correct json encoding) to the webserver. if we pass 99%, we will get the egg. check out the source code- the machine learning part is commented pretty well.

24 ELF

this elf binary does only show the correct flag, when we enter the correct pin. first i didnt get how this works and i tried to patch the binary, but that didnt show a proper qr. then i noticed in IDA that the qr1 is referenced more than once.

```
.text:000006B4 A1 4B 20 00 00                 mov     eax, dword ptr unk_204B
.text:000006B9 8B 1D AF 20 00+                mov     ebx, dword ptr unk_20AF
.text:000006BF 39 D8                          cmp     eax, ebx
.text:000006C1 7D 07                          jge     short mp_1
.text:000006C3 68 3F 20 00 00                 push    offset qr1
.text:000006C8 EB 05                          jmp     short mp_2
.text:000006CA                ; ---------------------------------------------------------------
.text:000006CA
.text:000006CA
.text:000006CA 68 A
.text:000006CF
.text:000006CF
.text:000006CF E8 0
.text:000006D4 83 C
.text:000006D7 89 E
.text:000006D9 5D
.text:000006DA C3
.text:000006DA
```

xrefs to qr1

Directio	Ty	Address	Text	
	o	checkpin+F	push	offset qr1
D...	o	.text:0000075B	mov	esi, offset qr1
D...	o	.text:0000076D	mov	eax, offset qr1
D...	o	.text:00000779	mov	esi, offset qr1

i used gdb to check whats going on, but sadly i was not able to set a breakpoint directly to the desired address. but i found the code by inspecting the disassembly in gdb (x/60i $eip):

```
0x5655575b <qr_next_line+39>:        mov    esi,0x5655703f
0x56555760 <qr_next_line+44>:        xor    ebx,ebx
0x56555762 <qr_next_line+46>:        xor    ecx,ecx
0x56555764 <qr_next_line+48>:        add    ebx,DWORD PTR [esi+ecx*4]
0x56555767 <qr_next_line+51>:        inc    ecx
0x56555768 <qr_next_line+52>:        cmp    ecx,0x19
0x5655576b <qr_next_line+55>:        jne    0x56555764 <qr_next_line+48>
0x5655576d <qr_next_line+57>:        mov    eax,0x5655703f
0x56555772 <qr_next_line+62>:        mov    eax,DWORD PTR [eax-0x4]
0x56555775 <qr_next_line+65>:        cmp    eax,ebx
0x56555777 <qr_next_line+67>:        jne    0x56555793 <qr_next_line+95>
0x56555779 <qr_next_line+69>:        mov    esi,0x5655703f
0x5655577e <qr_next_line+74>:        add    esi,0x64
```

here is an interesting check. during the nextline painting, there is a compare that will manipulate the drawing of the lines. we can set a breakpoint to 0x*56555775* and check the registers

```
Breakpoint 2, 0x56555775 in qr_next_line ()
gdb-peda$ info r
eax            0x1e240  0x1e240
ecx            0x19     0x19
edx            0xf7fa7870       0xf7fa7870
ebx            0x4179dce2       0x4179dce2
esp            0xffffd608       0xffffd608
ebp            0xffffd608       0xffffd608
esi            0x5655703f       0x5655703f
edi            0xf7fa6000       0xf7fa6000
eip            0x56555775       0x56555775
```

in eax we have my fake code (dec 123456) and in ebx 0x4179dce2 which is 1098505442 in decimal. lets try it on the binary. yes! this shows the correct qr!

25 HIDDEN EGG #1

this egg is hidden in the response headers of the webpage. its called Content-Eggcoding and is a base64 encoded link to https://hackyeaster.hacking-lab.com/hackyeaster/images/eggs/ba0c74ed439ab4795fc36999f542ba50b326e109.png

26 HIDDEN EGG #2

to reveal this egg you must browse on the page with the edge browser and then pin the page to the startmenu. to see the egg, just resize the new icon:

of you can simply copy the link from this xml https://hackyeaster.hacking-lab.com/browserconfig.xml

27 HIDDEN EGG #3

this hidden egg is an image in the mobile app - its called logosmall.png on iOS.