Hacky Easter 2018

Summary

PS, www.hacking-lab.com

HACKING-LAB

Table of Contents

Intro

Outro

Hacky Easter 2018 is history!

More than 2'400 hackers participated. A new record, after there was a little drop last year. What pleases me most is the fact that 14 (!) volunteers helped making the event such a success, by providing one or more challenges. This is just great and keeps Hacky Easter running! In case you want to provide a challenge, too, just let me know!

Thank you and stay tuned for HACKvent 2018 and Hacky Easter 2019!

PS
ps@hacking-lab.com

Volunteers

A big thank you to the volunteers who provided challenges (in alphabetical order):

- 3553x
- AppleStuff
- brp64
- CoderKiwi
- darkstar
- Dingo
- Dykcik
- explo1t
- inik
- jcel
- Kiwi.wolf
- opasieben
- otaku
- trolli101

Credits

Credits for the solutions go to (in alphabetical order):

- 0x90v1
- Buge
- Darkice
- Eydis
- Floxy
- HaRdLoCk
- IVlike
- Kiwi.wolf
- L3O
- LlinksRechts
- Lukasz_D
- MaZeWindu
- Meliver
- Mitsch
- S0Kala
- Seppel
- Spitzbua
- TheVamp
- beewasp
- blaknyte0
- darkstar
- daubsi
- eash
- enigma69
- evandrix
- explo1t
- horst3000
- inik
- jcel
- khae
- markie
- mezuru
- muzido
- opasieben
- pjslf
- scryh
- sym
- veganjay

Awards

Perfect Solvers

Congrats to the following 30 hackers who solved all Easter eggs (in order of time)! Well done!

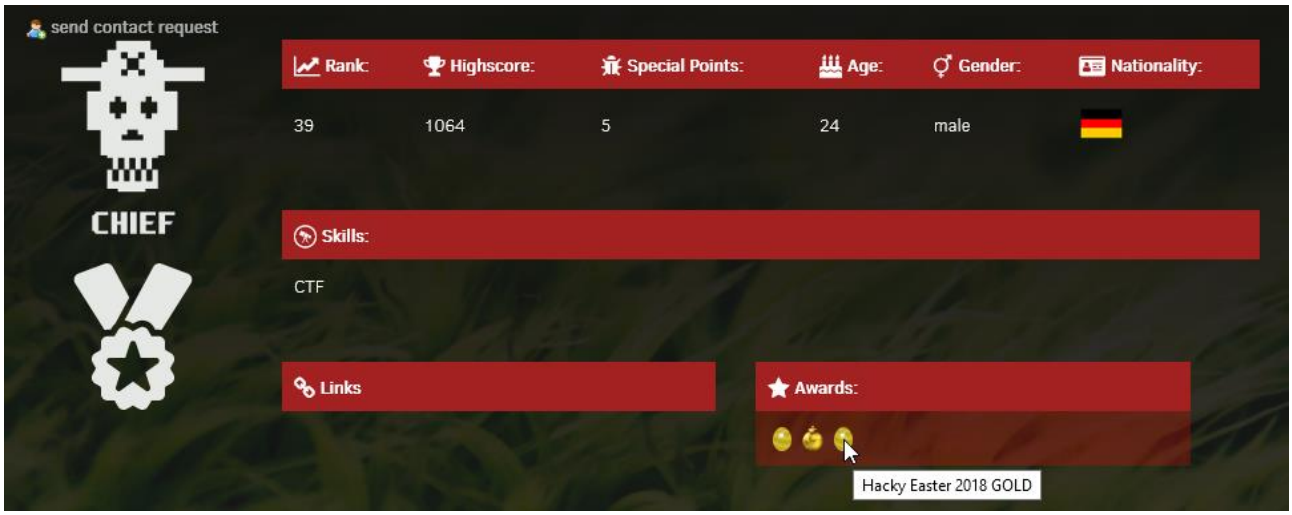
darkstar
Darkice
ikarus31415
explo1t
pjslf
applemarkus
resjoc13
GianfriAur
DrSchottky
sunscan
evandrix
0x90v1
eash
armfish
DeathsPirate
Eydis
0e85dc6eaf
LlinksRechts
Floxy
daubsi
veganjay
opasieben
HaRdLoCk
ptu
khae
MaZeWindu
TheVamp
vernjan
Buge
Seppel

Hacking-Lab Awards

As usual, we've created awards in Hacking-Lab for this competition. You got one of them, in case you reached the following total scores (Easter eggs, write-up, and teaser challenge).

- 130 points  GOLD
- 110 points  SILVER
- 90 points  BRONZE

Your awards are shown on the profile page:



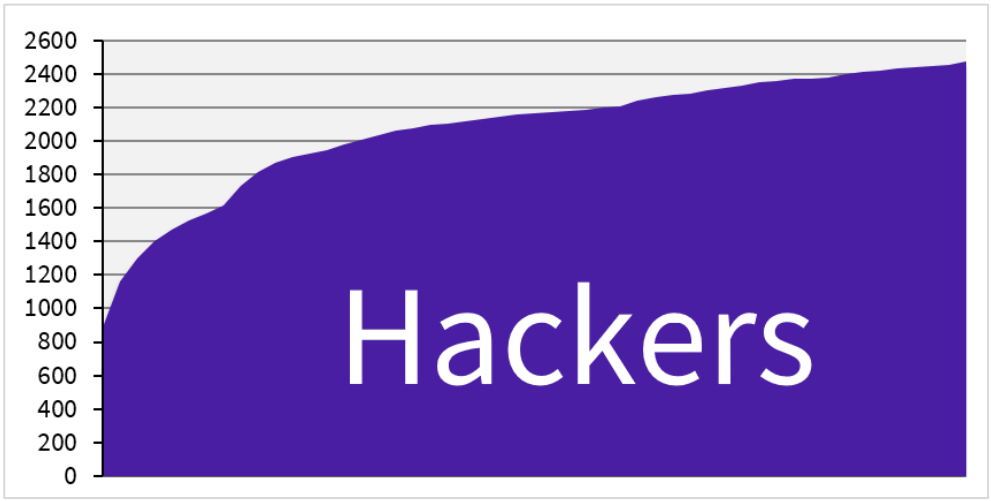
Statistics

General

	2018	2017	2016	2015	2014
Hackers	2'475	1'735	2'154	1’313	728
Points total	17'126	21'374	28'672	25’170	13’992
Points per hacker	6.92	12.32	13.31	19.17	19.22
Perfect solvers	30	53	54	55	-
Eggs solved	6'240	7'458	10'050	7’698	4’140
Nations	86	78	104	86	-

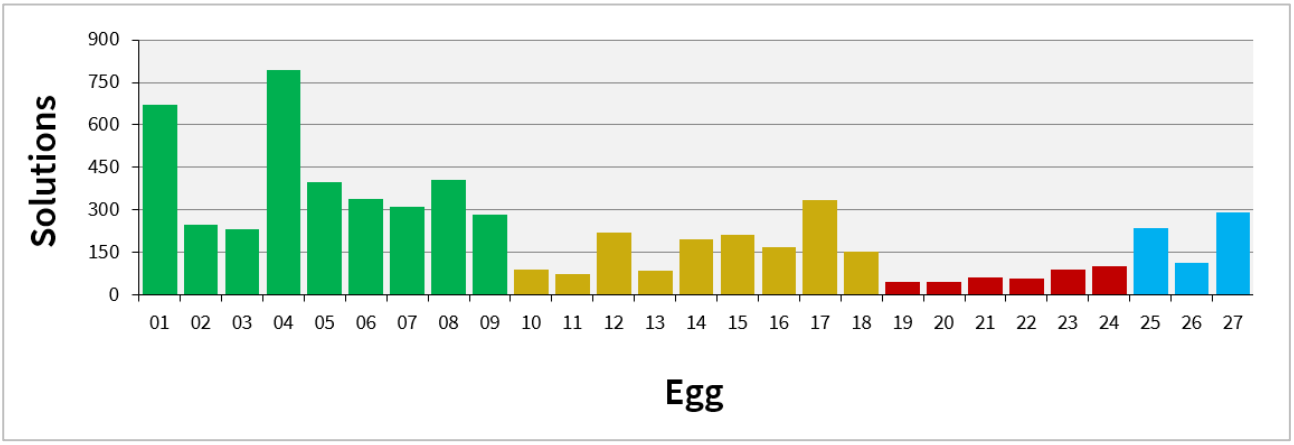
Event Activity

Number of hackers and solutions, growing with time.



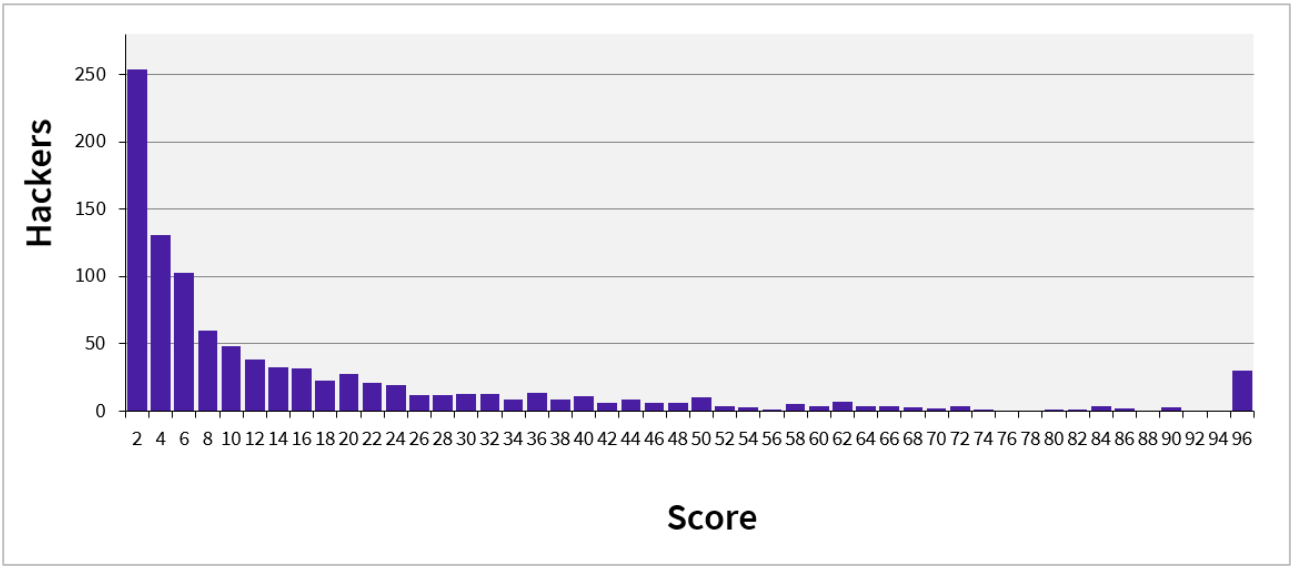
Solutions per Egg

Number of solutions, per egg. Seems "easy" eggs 2 and 3 were a bit too hard…



Score Distribution

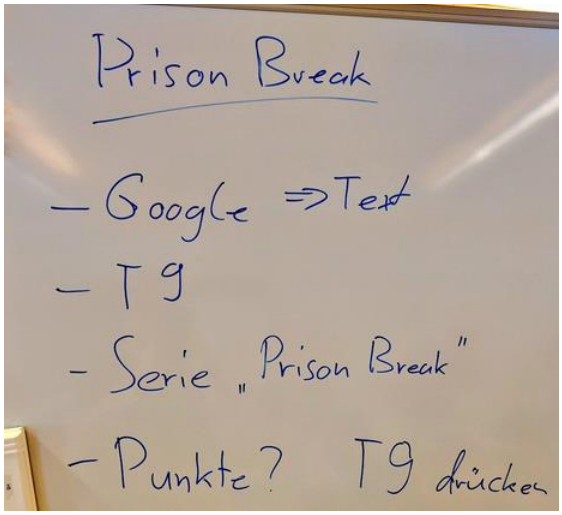
Number of users, for each score.



Fun

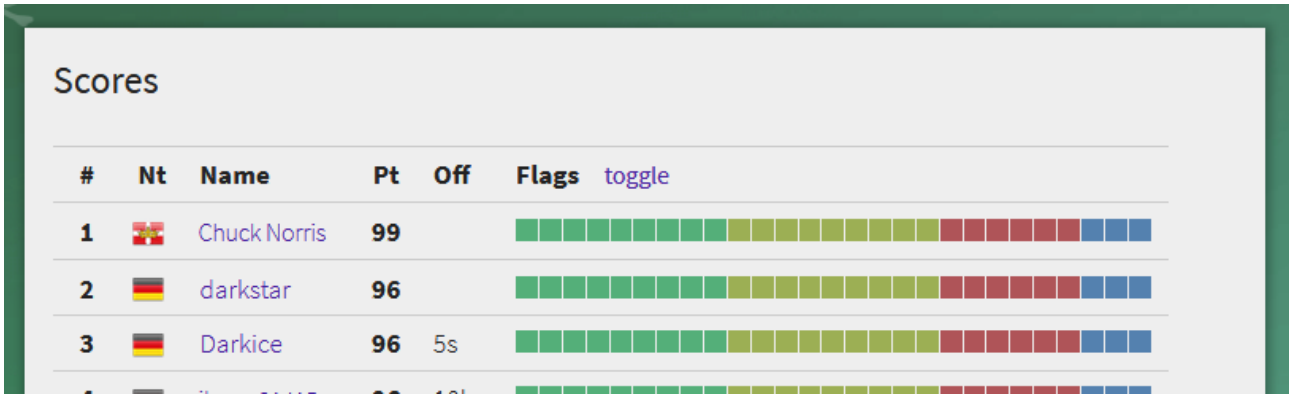
Images

Found online and in solution documents provided.



Chuck Norris

No worries, we weren't hacked. It was just a little joke by us.



Solutions

Teaser Challenge

Level: medium
Solutions: 187
Author: PS

Challenge

Play the teaser game, and beat the boss to get an Easter egg. But we warned - the boss is very powerful! You'll need some trickery to be victorious!

Beat the boss and get the Easter egg! Submit the solution code of the egg.

Solution by HaRdLoCk

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!

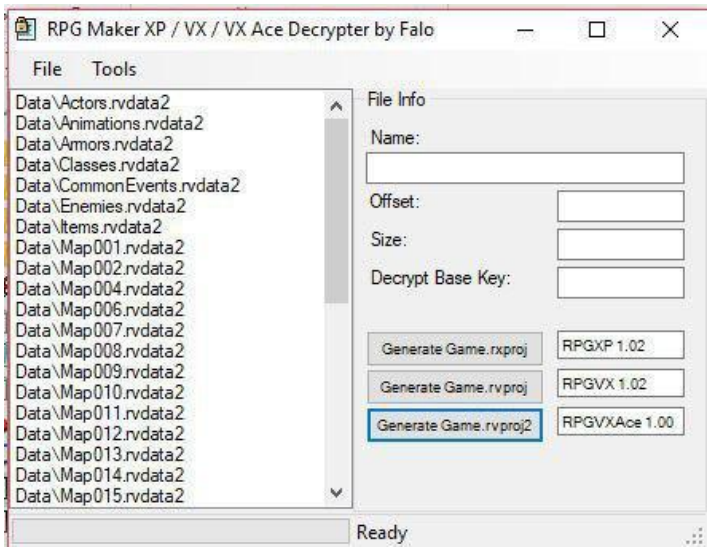
i simply started the game, saved it once and exited again. then i used a savegame editor (RpgMakerSaveEdit) to give myself the egg item.

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

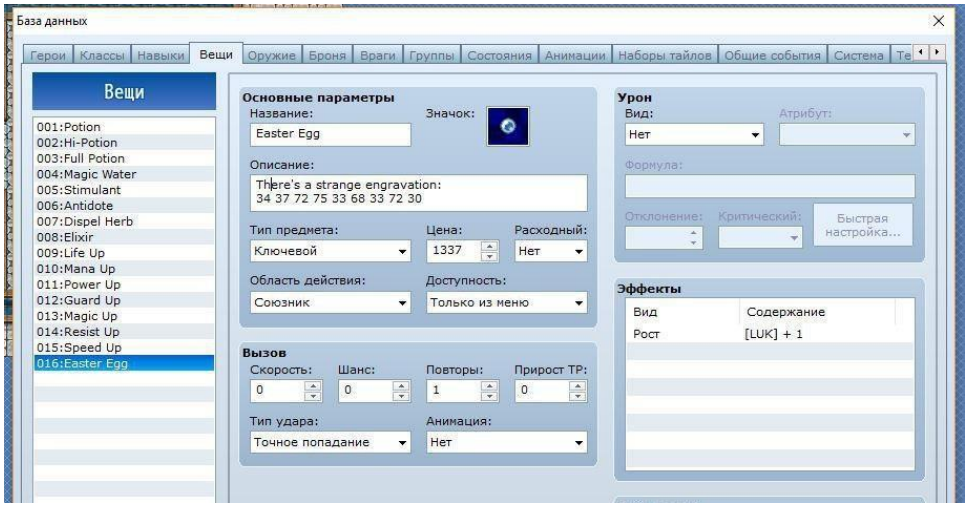
Password: 47ru3h3r0

Solution by Eydis

First I extracted game data with RPG Maker XP / VX VX Ace Decrypter and generated the Game.rvproj2 file.

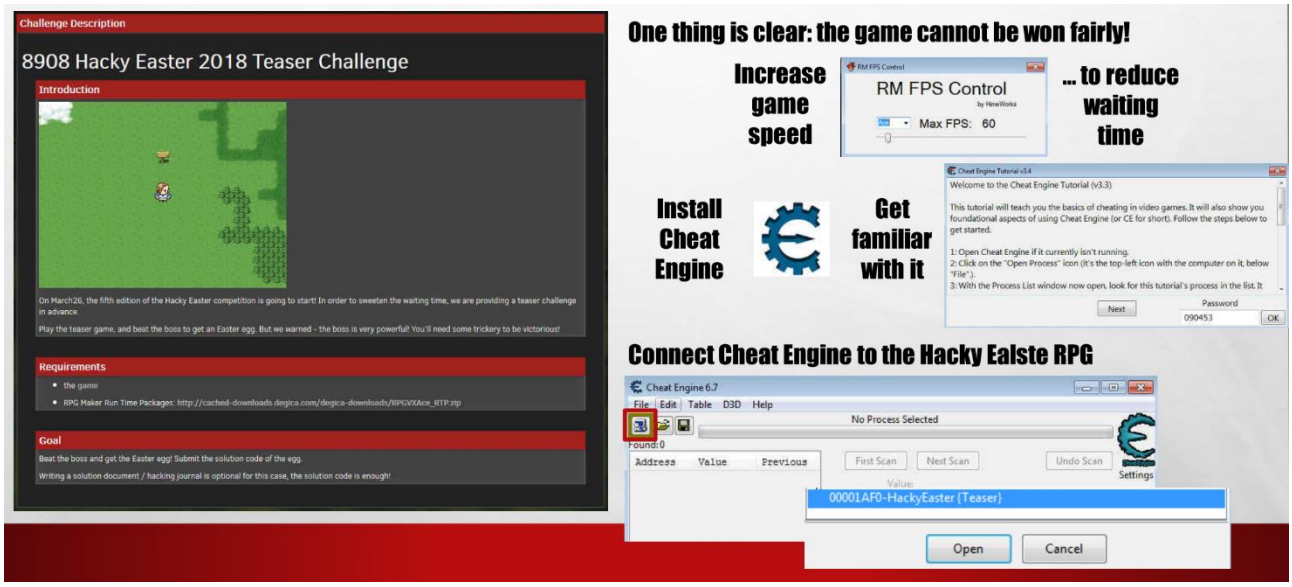


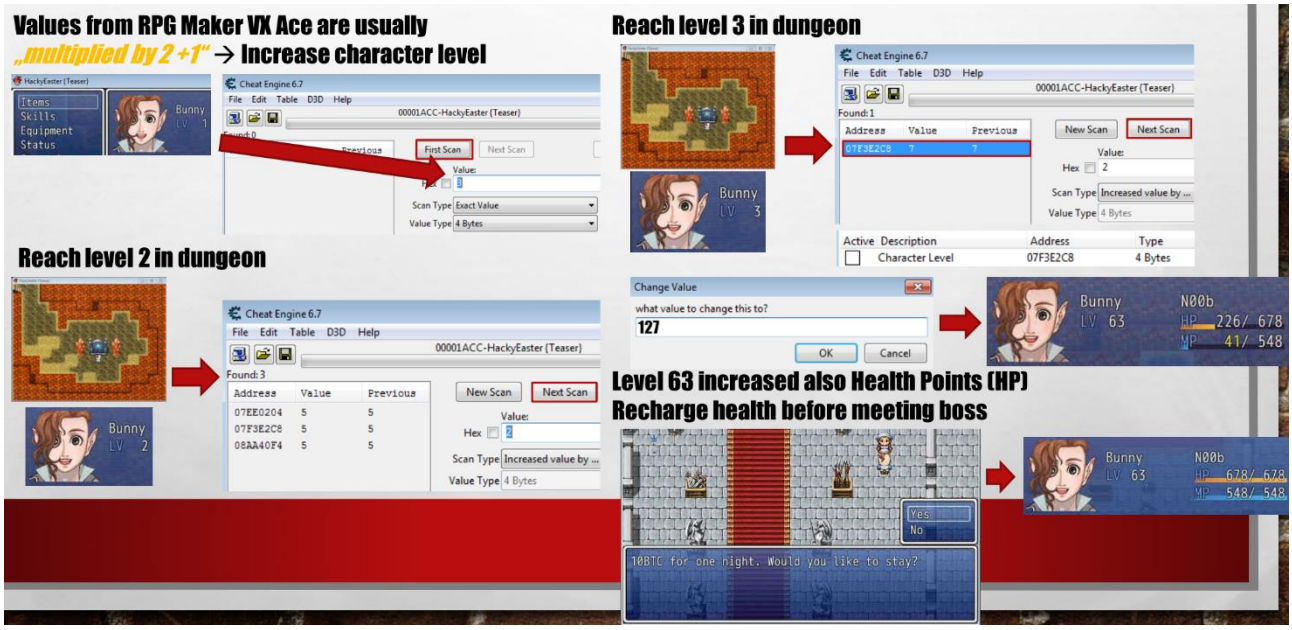
I opened the game file with RPGVXAce RPG Maker and looked through items and locations.
I found an item called 'Easter Egg':

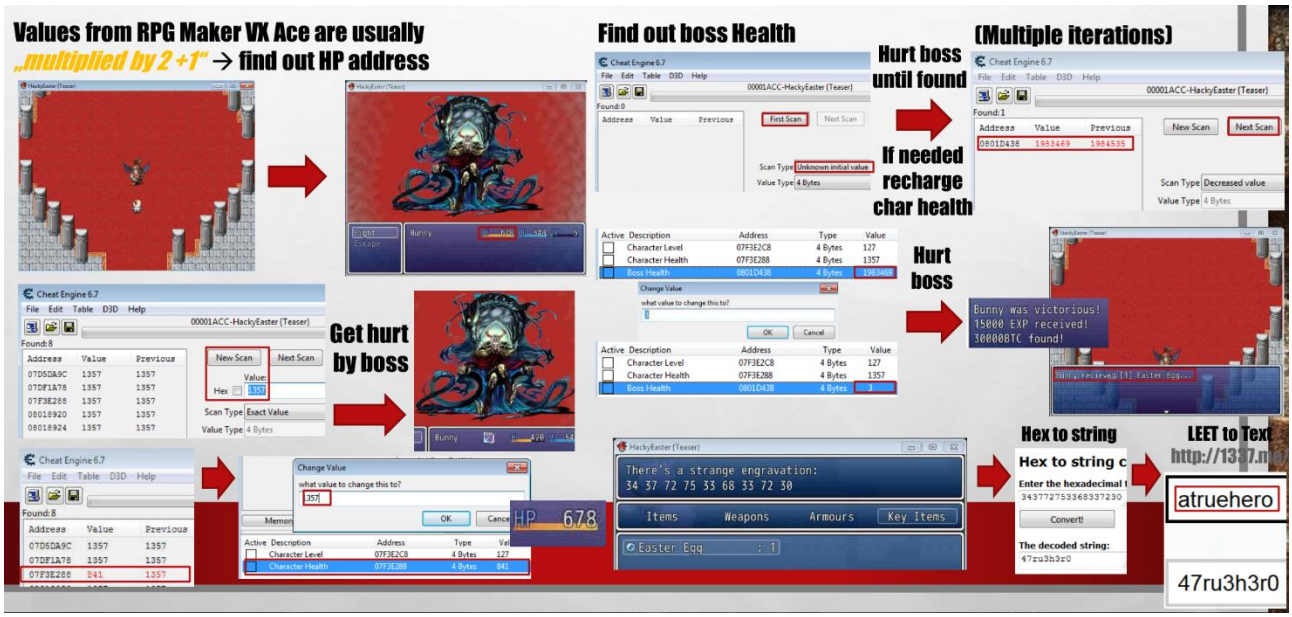


In the description there was a following number sequence: *34 37 72 75 33 68 33 72 30*. Converted to ASCII: 47ru3h3r0.

Solution by Seppel







Egg 01 – Prison Break

Level: easy
Solutions: 670
Author: Dingo

Challenge

Your fellow inmate secretly passed you an old cell phone and a weird origami. The only thing on the phone are two stored numbers.

```
555-7747663 Link
555-7475464 Sara
```

Find the password and enter it in the Egg-o-Matic below. **lowercase only, no spaces!**

origami.png

Solution by opasieben

Reference

This refers to the encryption used in prison break to send Sara a secret message. The Numbers refer to the keys of a T9 numpad, where the dots means the character on the button. e.g. Number 7, 3 dots = R

Encryption

```
Number: 555-7747663 Link
. ... ... .... ... .. ..
7 7   4   7    6   6  3
P R   I   S    O   N  E
```

```
Number: 555-7475464 Sara
... ... .... .. ... .. .
7   4   7    5  4   6  4
R   I   S    K  I   N  G
```

Password

```
prisonerisking
```

Solution by IVlike

We can take a look at the attached picture (origami.png). There we can see some dots. If we transfer the dots to numbers we get:

```
.       ...     ...     ....    ...     ..      ..
1       3       3       4       3       2       2
```

Taking these numbers in combination with the provided numbers from above we can write it up like:

```
7   7   4   7   6   6   3
1   3   3   4   3   2   2
```

Now looking up for an old phone number pad we can use these numbers to get the wanted word. So we take the first letter of button 7, then the third and so on:

```
p r i s o n e
```

now the same for the second on

```
...     ...     ....    ..      ...     ..      .
3       3       4       2       3       2       1
```

Taking these numbers in combination with the provided numbers from above we can write it up like:

```
7   4   7   5   4   6   4
3   3   4   2   3   2   1
r   i   s   k   i   n   g
```

The answer is:

prisonerisking

Solution by darkstar

Solved by hand using an old mobile phone.

Egg 02 – Babylon

Level: easy
Solutions: 246
Author: CoderKiwi

Challenge

The tower is not the only thing in Babylon which has walls and shelves.

```
4 - 4 - 28 - 355
```

babylon.txt

Solution by Spitzbua

Visit the library of babel online:
https://libraryofbabel.info/browse.cgi

Solution by pjslf

I followed the hint from description. After some googling I found out that it could be a reference to the Library of Babel.

"The Library of Babel" (Spanish: La biblioteca de Babel) is a short story by Argentine author and librarian Jorge Luis Borges (1899–1986), conceiving of a universe in the form of a vast library containing all possible 410-page books of a certain format and character set.

It brought me to the libraryofbabel.info page.

I selected browse option from the menu, inserted the ciphertext content of the provided file into the *hex name* textarea and submitted the form.

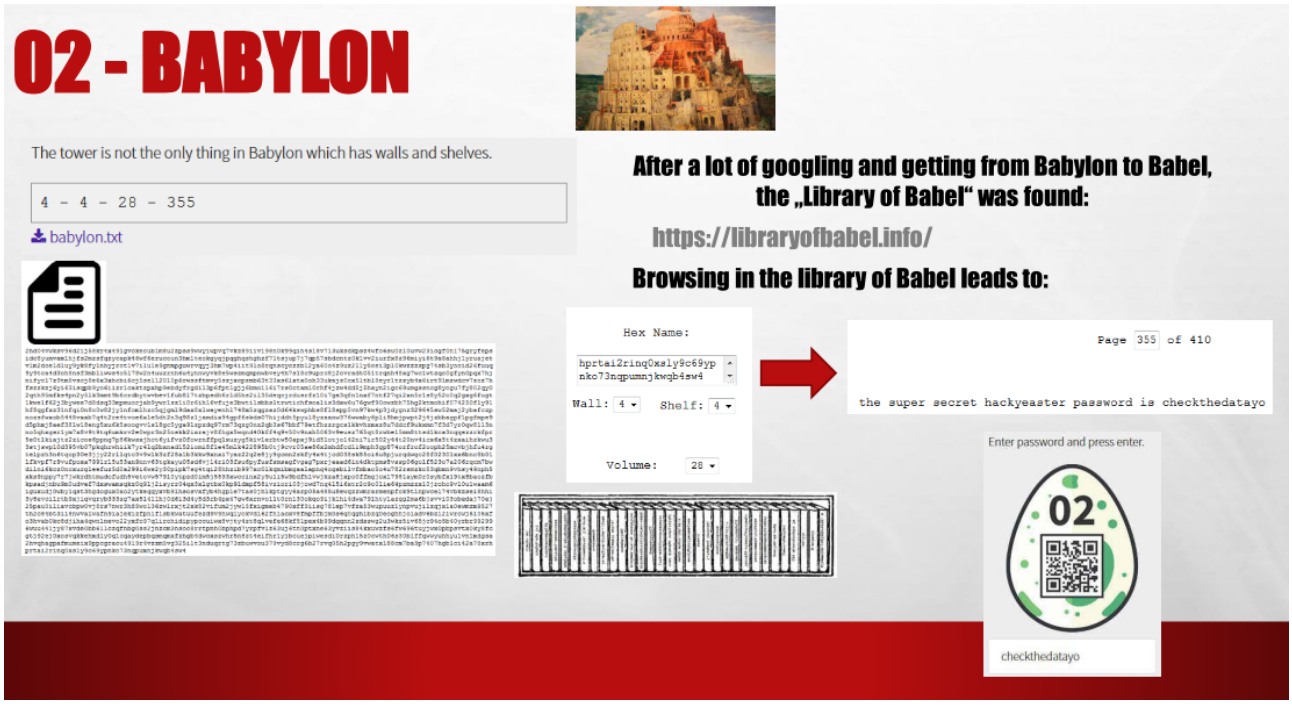
Then I simply applied the addressing from description:

```
Wall: 4
Shelf: 4
Volume: 28
Page: 355
```

This was the content of that page:

```
the super secret hackyeaster password is checkthedatayo
```

Solution by Seppel



Egg 03 – Pony Coder

Level: easy
Solutions: 233
Author: PS

Challenge

Tony the pony has encoded something for you. Decode his message and enter it in the *egg-o-matic* below! **Lowercase and spaces only, and special characters!**

```
gn tn-gha87be4e
```

Solution by beewasp

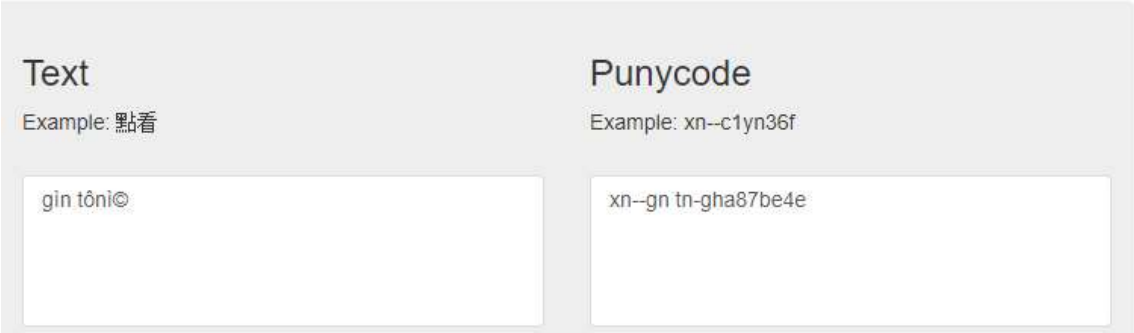
This is punycoder
https://www.motobit.com/util/punycode-decoder-encoder.asp

Source data:
gn tn-gha87be4e
FromPUNYCODE : Unicode string :
gìn tônì©
ToIDN : IDN representation of 'gn tn-gha87be4e' string :
gn tn-gha87be4e
ToPUNYCODE : Punycode representation of 'gn tn-gha87be4e' string :
gn tn-gha87be4e-
ToUTF7 : utf-7 representation of 'gn tn-gha87be4e' string :
gn tn-gha87be4e
Input text:
gn tn-gha87be4e
To IDN >> | To punycode >> | From punycode >> | From IDN >>

It didn't take gìn tônì©
As the password, so standard characters: gin tonic gave the egg.

Solution by eash

After some Google search I reach the “PunyCoder” site https://www.punycoder.com/.



The password is “gin tonic”

Solution by muzido

A google search for "pony coder decode". Then I found some pages about punycode or idn (Internationalized Domain Names). Then I installed an idn converter tool as below;

```
sudo apt-get install idn
```

then run this command to decode Punycode

```
->> idn -d "gn tn-gha87be4e"
gìn tônì©
```

I found the password

Solution:

gin tonic

Egg 04 – Memeory

Level: easy
Solutions: 793
Author: otaku

Challenge

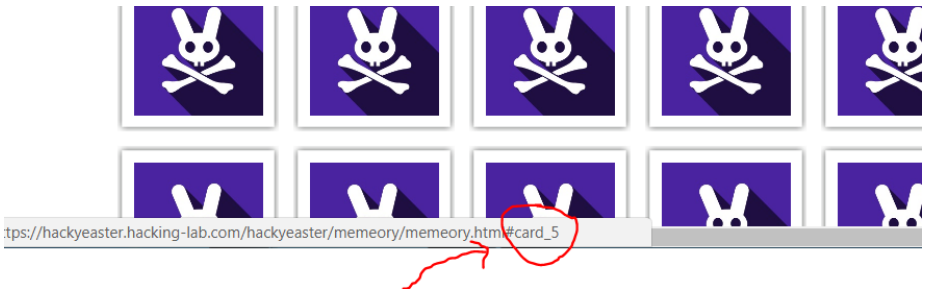
Fancy a round of memeory?

Click here to play.

Solution by enigma69

I know, my solution maybe is not the fastest way - but it is simple!

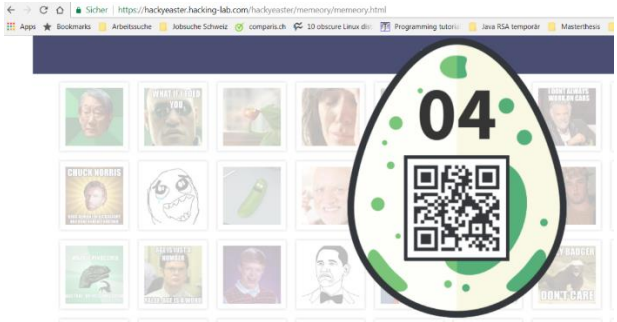
When I opened the game, I saw a card set with 100 concealed cards. If I hovered with the mouse over a card, I could see the card_number at the bottom of the screen:



I found out, that cards which belong together, have adjacent card numbers (card number 0 and 1 have the same picture, card number 2 and 3 have the same picture and so on).

So I hovered with my mouse over all concealed cards, until I found 2 adjacent card numbers - after I found these 2 cards, I revealed these cards.

When I revealed the last card, I got my easter egg:



Solution by evandrix

```
(new Array(50).fill(0)).forEach((el,i)=>{xs =
`./lib/${i+1}.jpg`;$("img.boxFront").filter((i,el)=>$(el).attr("src")===xs).clic
k();});
```

Solution by jcel

The task was to successfully play memory with meme pictures in a 100x100 grid. Inspecting the source code of the page revealed that the tiles were initiallay arranged in the correct order, i.e. two matching tiles (identified by the image file) were next to each other:

```
      <figure id="legespiel_card_0">
        <a href="#card_0">
          <img class="boxFront" src="./lib/1.jpg" />
          <img class="boxWhite" src="./lib/shadow_card.png" />
          <img class="boxBack" src="./lib/back.jpg"  />
           </a>
          <img class="boxStretch" src="./lib/shim.gif" />
      </figure>
      <figure id="legespiel_card_1">
        <a href="#card_1">
          <img class="boxFront" src="./lib/1.jpg" />
          <img class="boxWhite" src="./lib/shadow_card.png" />
          <img class="boxBack" src="./lib/back.jpg"  />
           </a>
          <img class="boxStretch" src="./lib/shim.gif" />
      </figure>
```

In the Firefox's Inspector tool, the tiles were rearranged randomly. A simple manual (but somewhat tedious) method to solve this was thus to simply use the Inspector's drag-and-drop function to restore the initial order. Then, simply clicking pairs of neighboring tiles revealed the egg.

A much more elegant solution would have been to sort and click the tiles using JavaScript on the console, but the simple task of manually sorting an array was kind of relaxing ;-)

Egg 05 – Sloppy & Paste

Level: easy
Solutions: 396
Author: Lukasz_D

Challenge

The egg is right here. Just copy it!

Copy to Clipboard

iVBORw0KGgoAAAANSUhEUgAAAeAAAAHgCAMAAABKCk6nAAACQ1BMVEUAAAA0NDQzNTozNTozNTkzNTo1MjgzNTo0NDo1NzkzN
To1Mzw0NDo0NTk0NDkzNDo0NDo1NDo1NDo0NDo0NDo0NDk0NDo0NDo0NDo0NDk0NDo0NDk1NTo1NDk0NTs1NDn///r5+eQzN
jr///9Ur3k1NTV7xnswNDhMTlLz8/Pw8PBdX2luMTWZm53HyMne3t9rbnA4Oz5TVllQU1aOj5LU1NWEhoj8/Pz9/vji4+JCRUh6fH/29va
qq6y4urs9QEPS09Tp6un6+vpGSEpfYWWKjI6ur7D19+FAQkalpqfLy8zs7e1IS05Ysnr5+/ZWWFldtH1oam3g4eHP6MpbXF3Gx8fX2Nd8f
nigoaFkZ2plt4Pl5eXV7NF+yH7z+u684Lvg4M59w5GOj4d2d3bC47+k16Lz9ODm8dfw8dynqZ7j899UVVTr89vK5sav2rCPz47CwsGCyYJ
tbmrv9d7c8Ni8vby0tbTo6dWV0ZRjZGL2+/J0v4xxc3X29vHu7ulUsHnb7NCq2qnp9eW4uauYzqJ+glHH5cPOzr6z2rac1JuWmJiDhH6Sl
JWKzImYmZCGyoaIiYLa2tm237Sc0KeQyp3t9+m9vrCm1K1DQ0I9PT3y8u1tuolnaWbP0M/X2MfGx7eKyJnf39uExZbt7dmvr6Ntu4STk
43h7tTc3MvS0sK4ubeoqank5NGys6adnpRzc29Om27Cw7RSqHVPT048TkU5R0J3vndytHNFXUw9VUqgopdjlWVlhWNYf1tDdFpJaVN
rp236gmvtAAAAIHRSTlMABfn16e8P29UZ4iOBLq2iSmhBzcV2tFK8l4mQN1lvYEWTttgAADbgSURBVHja7J3rT5NXHMcfCw5LedJAuioTLed0

Solution by veganjay

This is a mobile challenge. Get the Android APK from the device:

```
$ adb shell pm list packages -f | grep hacky
package:/data/app/ps.hacking.hackyeaster.android-
1/base.apk=ps.hacking.hackyeaster.android
$ adb pull /data/app/ps.hacking.hackyeaster.android-1/base.apk
[100%] /data/app/ps.hacking.hackyeaster.android-1/base.apk
```

Use apktool to extract the contents

```
$ apktool d base.apk
```

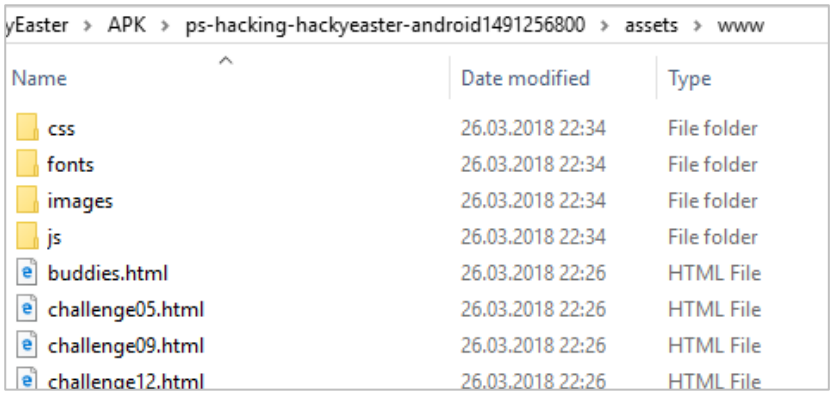
Inside the file base/assets/www/challenge05.html is the base64 encoded image for the egg.

```
$ cp challenge05.html egg05.txt
<Edit the file and retain only the base64 code>
$ base64 -d egg05.txt > egg05.png
```

Solution by sym

When copying the base64 string on the smartphone and then decoding it, the following image is displayed:

So I unpacked the APK and found the HTML-Files in the www dir.

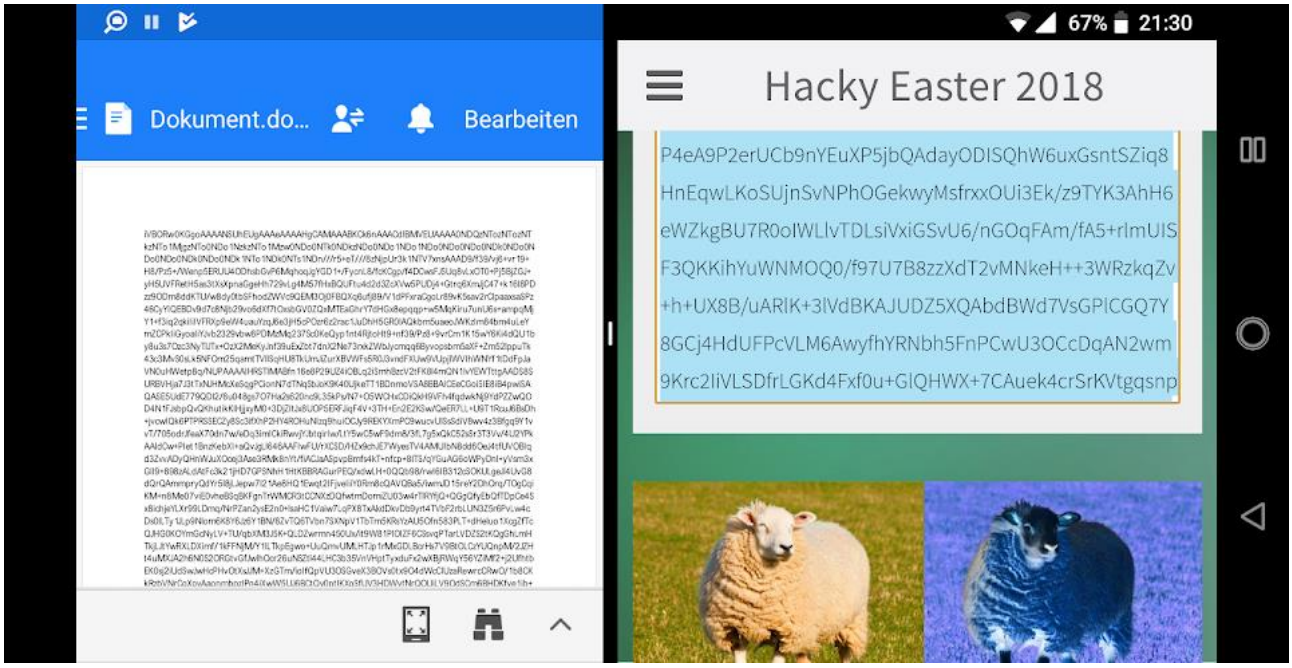


I checked the source of challenge05.html. As I expected, I could see the Base64 string which was different than the previous one from the clip board.

Decoding this one gave me the correct egg.

Solution by blaknyte0

Activate split screen (Android) and start the Hacky Easter App in one screen and a text editor in the other one. Drag the text from the app into the text editor (Polaris Office).



Copy text into a Base64 decoder, remove one equals symbol (=), decode and the easter egg shows up.

Egg 06 – Cooking for Hackers

Level: easy
Solutions: 337
Author: AppleStuff

Challenge

You've found this recipe online:

1 pinch: c2FsdA==

2 tablesspoons: b2ls

1 teaspoon: dDd3Mmc=

50g: bnRkby4=

2 medium, chopped: b25pb24=

But you need one more secret ingredient! Find it!

Solution by L3O

Looking at the encoded text seems like base 64 which gives:

```
salt
oil
t7w2g
ntdo.
onion
```

3rd and 4th didn't make any sense so I tried to keep decoding into Hex, base32 so on. After a while when I was just staring the decoded ingredients onion catch my eye and I thought it might be related to TOR then I started googling with the challenge title and tor then I bumped into this website: https://thehiddenwiki.org/ Finally I was able to solve the challenge. It was an onion URL.

Solution by Seppel

Solution by eash

I decoded the receipt from base64 that pointed me to a .onion site.

Recipe	Base64 decoded
1 pinch: c2FsdA==	salt
2 tablesspoons: b2ls	oil
1 teaspoon: dDd3Mmc=	t7w2g
50g: bnRkby4=	ntdo.
2 medium, chopped: b25pb24=	onion

https://saltoilt7w2gntdo.onion.to/ingredient_egg06.png

The secret ingredient is:

Egg 07 – Jigsaw

Level: easy
Solutions: 311
Author: darkstar

Challenge

Thumper was probably under time pressure and jumped around a bit too wild. As a result, his picture has broken. Can you write a program to put it back together?

jigsaw.png

Solution by markie

So the scrambled jigsaw is 32 tiles x 18. The file is 1280 x 720 so each square is 40 x 40 pixels.
Use image magic to chop jigsaw.png into its component parts.

```
magick convert jigsaw.png -crop 40x40 C:\Users\mark\Desktop\hacky\parts-%02d.png
```

Now it's just a case of putting it back together in MS Paint 3d.

Password is: goodsheepdontalwayswearwhite

Solution by Meliver

I tried to do it with a script, but did not see an easy solution. As the GAF (Girlfriend Acceptance Factor) of hacky easter is not very high, I used girlfriend.solve(jigsaw) to get the solution of this challenge. The script focused on putting together the pieces with letters on:

goodsheepdontalwayswearwhite

Solution by evandrix

@ https://github.com/alexey-tsvetkov/jigsaw-puzzle

```
src/run.py --generate -i jigsaw.png -o jigsaw-pieces --piece_size 40
```

...or...

```
convert -crop 40x40+0+0 jigsaw.png piece-0-0.png
```

```
[py]
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import sys
import operator
from PIL import Image

def go(root,direction,niter):
    ns = []
    for i in xrange(niter):
        # print>>sys.stderr, "#%d: %d"%(i,root)
        vals = {}
        ima = Image.open("jigsaw-pieces/%d.png"%root)
        da = ima.load()
        sa = ima.size
        for j in xrange(576):
            if j == root: continue
            imb = Image.open("jigsaw-pieces/%d.png"%j)
            sb = imb.size
            db = imb.load()
            z = 0
            for x in xrange(sa[1]):
                if direction == "l":
                    a = da[0,x]
                    b = db[sa[0]-1,x]
                elif direction == "t":
                    a = da[x,0]
                    b = db[x,sa[1]-1]
                elif direction == "r":
                    a = da[sa[0]-1,x]
                    b = db[0,x]
                else: # [b]ottom
                    a = da[x,sa[1]-1]
                    b = db[x,0]
                y = abs(a[0]-b[0])+abs(a[1]-b[1])+abs(a[2]-b[2])
                z += y
            vals[j] = z
        sorted_vals = sorted(vals.iteritems(), key=operator.itemgetter(1))
        # print>>sys.stderr, root,direction,niter, sorted_vals[:8]
        root, _ = sorted_vals[0]
        if direction == "l": ns.insert(0,root)
        else: ns.append(root)
```

```
    return ns

if __name__ == "__main__":
    for root in
[4,8,15,19,25,28,30,33,34,39,40,57,59,66,71,99,126,133,145,157,198,199,201,205,2
11,240,252,259,260,285,292,298,305,367,372,376,377,389,432,441,461,466,476,478,5
16,517,524,536,550,561,562]:
        print>>sys.stderr, root
        img_out = Image.new("RGBA", (40*32+1,40*18+1), (255,255,255,255))
        idxss = [

    go(go(root,"t",1)[0],"l",8)+[go(root,"t",1)[0]]+go(go(root,"t",1)[0],"r",8),
            go(root,"l",8)+[root]+go(root,"r",8),

    go(go(root,"b",1)[0],"l",8)+[go(root,"b",1)[0]]+go(go(root,"b",1)[0],"r",8)
        ]
        for i,idxs in enumerate(idxss):
            for j,x in enumerate(idxs):
                img = Image.open("jigsaw-pieces/%d.png"%x, "r")
                img_w, img_h = img.size
                img out.paste(img, (j*40,40*(1+i)))
        img_out.save("output-%d.png"%root)
```

password: goodsheepdontalwayswearwhite
(ref: Bon Jovi - Good Guys Don't Always Wear White)

Egg 08 – Disco Egg

Level: easy
Solutions: 407
Author: inik

Challenge

> *Make things as simple as possible but no simpler.*
>
> -- Albert Einstein

Click here

Solution by explo1t

First I stored the html site locally. After some analysis I found out, that every QR-code pixel has multiple classes. It randomly changes color out of its available colors (derived from classes). So I insert following js code, before the pixel change animation:

```
if (classes.indexOf("black") >= 0) {
       color = cellcolors["black"]
}
else {
       color = cellcolors["white"]
}
```

After some short waiting time, the egg was revealed.

Solution by eash

I have replaced the Javascript code that loads initially with the following code:

```
$(document).ready(function() {
  $("td").each(function() {
    var classList = $(this).attr("class");
    if (classList.indexOf("black") !== -1) {
      $(this).css("background-color", "black");
    } else if (classList.indexOf("white") !== -1) {
      $(this).css("background-color", "white");
    }
  });
});
```

Solution by muzido

There are multiple colors in disco.html source code as below.

```
...
<td class="cyan red brown blue black green darkgrey" style="background-
color:#FBF305;"></td>
<td class="cyan black blue red lightgrey" style="background-
color:#FBF305;"></td>
<td class="darkgreen black tan cyan green blue" style="background-
color:#FBF305;"></td>
..
```

I used the following shell code to remove all the other colors from the class except for "black" or "white" from disco.html

GetEgg.sh

```
#!/bin/bash
# to insert newline before "<td " and after "</td>" from html source code.
sed "s/<td /\n<td /g;s|</td>|</td>\n|g" disco.html |
while read -r line    # to read line by line
do
    # to change only contain "td class "  in the line
    if [[ $line = *"td class"* ]]; then
        # if the line contains white then delete other colors
        if [[ $line = *"white"* ]]; then
            echo '<td class="white" style="background-color:#ffffff;"></td>';
        else
        # if the line contains black then delete other colors
            if [[ $line = *"black"* ]]; then
                echo '<td class="black" style="background-color:#000000;"></td>';
            fi
        fi
    else
        # if the line not contains "white" or "black then just print the line
        echo $line;
    fi
done
```

Then I run the shell code as below.

```
./GetEgg.sh > Egg8.html
```

I found the egg in Egg8.html file.

Egg 09 – Dial Trial

Level: easy
Solutions: 283
Author: trolli101

Challenge

Dial the phone!

Dial!

Enter the password in the Egg-o-Matic below. Lowercase only! Make sure to be online!

Solution by veganjay

Having extracted the Android program in a previous challenge, find an mp3 file in base/res/raw/dial.mp3

Convert that to a WAV file:

```
$ fmpeg -i dial.mp3 dial.wav
```

Upload the WAV file to an online DTMF decoder

Copy and paste the results to a file. The results contain the tone offset, end offset and length, which we do not care about, so remove it with:

```
$ grep "^.$" decoded.txt > numbers.txt
```

This decodes to the pattern:

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

Separate the numbers by pairs, delimited by the "#" sign. For example:

```
4 * 7 #
2 * 6 #
```

At this point, the puzzle is very similar to challenge 1. The second number in each pair representing the number on a telephone keypad, and the first number corresponds to the index of the letter on the key. For example, "4

* 7" means look at the number 7, which has the letters "PQRS", and take the 4th letter, which is "S". Repeat this for all values:

```
4 * 7 #  s
2 * 6 #  n
1 * 2 #  a
2 * 5 #  k
2 * 3 #  e
3 * 6 #  o
2 * 6 #  n
2 * 6 #  n
3 * 6 #  o
2 * 5 #  k
3 * 4 #  i
1 * 2    a
```

The password is "snakeonnokia".

Solution by muzido

I found apk file from https://apkpure.com/hacky-easter/ps.hacking.hackyeaster.android. Then decompile this apk file by using http://www.javadecompilers.com/apk

I found dial.mp3 file in <apk_source>res/raw. Then run the following code to convert mp3 to wav file.
ffmpeg -i dial.mp3 dial.wav

I uploaded this wav file the following page.
http://dialabc.com/sound/detect/index.html

I found these;

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

Then I found the password like challenge 1.

```
Phone Numbers  :                 7   6   2   5   3   6   6   6   6   5   4   2
Letter position on the Keypad :   4   2   1   2   2   3   2   2   3   2   3   1
                                  S   N   A   K   E   O   N   N   O   K   I   A
```

Solution:
snakeonnokia

Solution by TheVamp

Another mobile Challenge. The first look goes into “assets/www/challenge09.html”. There is only a call into “ps://dial”. So I need to reverse the .dex file. The following part Triggers the Dial. You find this in the Activity.class:

```
124
125    private void handleDial() {
126        try {
127            this.mediaPlayer = MediaPlayer.create(this.getBaseContext(), 2131165184);
128            this.mediaPlayer.setAudioStreamType(3);
129            this.mediaPlayer.setLooping(false);
130            this.mediaPlayer.start();
131        } catch (Exception var2) {
132            var2.printStackTrace();
133        }
134    }
135
```

So it just plays a music file. A look up into “res/raw” shows a dial.mp3. It plays some DTMF Tones. I used the following site to make my first try: http://dialabc.com/sound/detect/

For this I converted the mp3 with audacity to a wav file. I got the following Input:

```
“4 * 7 # 2 * 6 # 5 # 2 * 3 # 3 * 6 # 2 * 6 # 2 * 6 # 3 * 6 # 2 * 5 # 3 * 4 #”
```

It looks like the prison break cipher. Which results into SNJEONNOKI and that’s wrong. After another research I found a tool called “DTMFChecker” and this give me the output:

```
“4*7#2*6#1*2#2*5#2*3#3*6#2*6#2*6#3*6#2*5#3*4#” which is SNAKEONNOKI
```

The looks really close, but it looks like that the last char is missing “snake on nokia” should be the solution (without spaces):

Egg 10 – Level Two

Level: medium
Solutions: 90
Author: Kiwi.wolf

Challenge

So you managed to beat the boss in the teaser game? This one won't be that easy!

You'll need RPG Maker Run Time Packages to run the game.

Hints: there are several parts to be found. Combine them, and enter the final flag in the *egg-o-matic* below, **without spaces**! Saving the game from time to time certainly helps.

game.zip

Solution by Mitsch

use "RGSSAD - RGSS2A - RGSS3A Decrypter.exe" the extract the game definitions from Game.rgss3a
convert it into the a readable YAML format with

```
./rvpacker -a unpack -d ../../HackyEaster\ RPG/ -t ace
```

the parts of the flag can be found in

```
Map014.yaml: Prison         703435357730726435 5f052d066b15035433
Map015.yaml: Garden         703435357730721056 c6b05032d0f546f4c
Map024.yaml: Dimension Rift 703435357730726435 5f3406033b5749114c
Items.yaml
  description: "703435357730726435 5f3472335f6330306c\r\n"
  name: Egg
```

XOR each flag from the Maps with the Egg Item

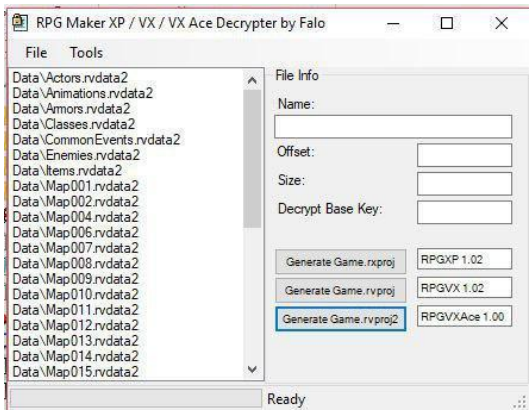
```
00  00  00  00  00  00  00  00  00  00  31  5F  35  34  76  33  64  5F
00  00  00  00  00  00  00  74  68  33  5F  77  30  72  6C  64  5F  20
00  00  00  00  00  00  00  00  00  00  00  74  30  64  34  79  21  20
```

ignoring resulting 0x00 and 0x20 and you get

```
1_54v3d_th3_w0rld_t0d4y!
```

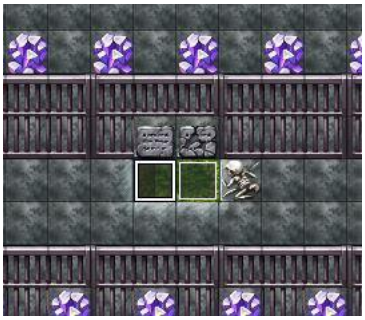
Solution by Eydis

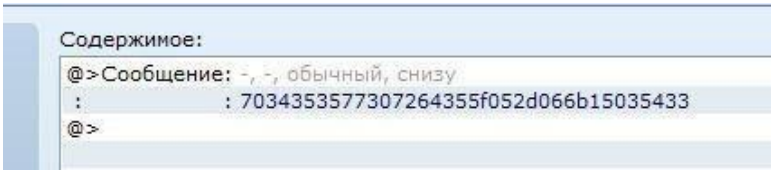
First I extracted game data with RPG Maker XP / VX VX Ace Decrypter and generated the Game.rvproj2 file.



Then I opened the Game file with RPGVXAce RPG Maker and looked through maps and items. I found the following pieces:

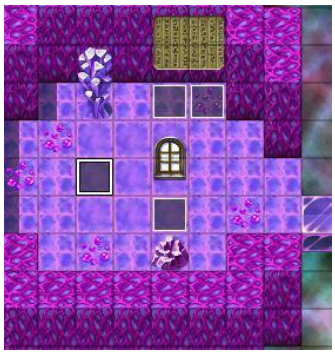
1. 703435357730726435 5f052d066b15035433 (p455w0rd5_-k————————————————T3) – event in the Prison location:

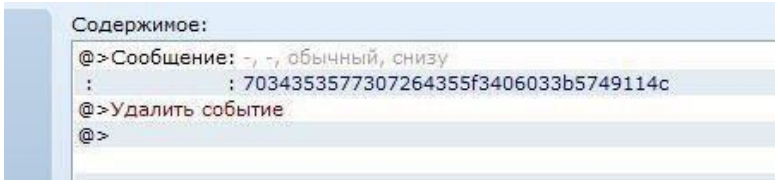




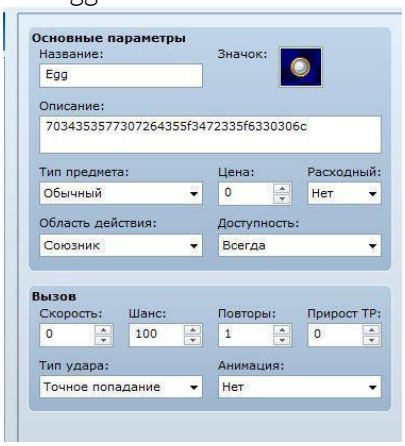
2. 70343535773072105d6c6b05032d0f546f4c (p455w0r]lk————————————————ToL) – event in the Garden location:

3. 703435357730726435 5f3406033b5749114c (p455w0rd5_4————————————————;WIL) – event in the Dimension Rift:





4. The Egg item: 7034353577307264355f3472335f6330306c (p455w0rd5_4r3_c00l)



5. An important hint:

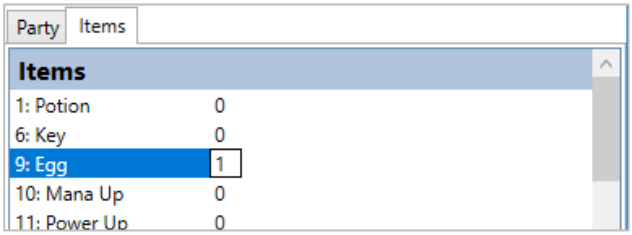
When I xored first three pieces with the last one I got these pieces of the password:

```
1. 7034353577307264355f052d066b15035433 ^ 7034353577307264355f3472335f6330306c =
315f35347633645f (1_54v3d_)

2. 70343535773072105d6c6b05032d0f546f4c ^ 7034353577307264355f3472335f6330306c =
7468335f7730726c645f20 (th3_w0rld_ )

3. 7034353577307264355f3406033b5749114c ^7034353577307264355f3472335f6330306c =
74306434792120 (t0d4y! )
```

Password: 1_54v3d_th3_w0rld_ t0d4y!

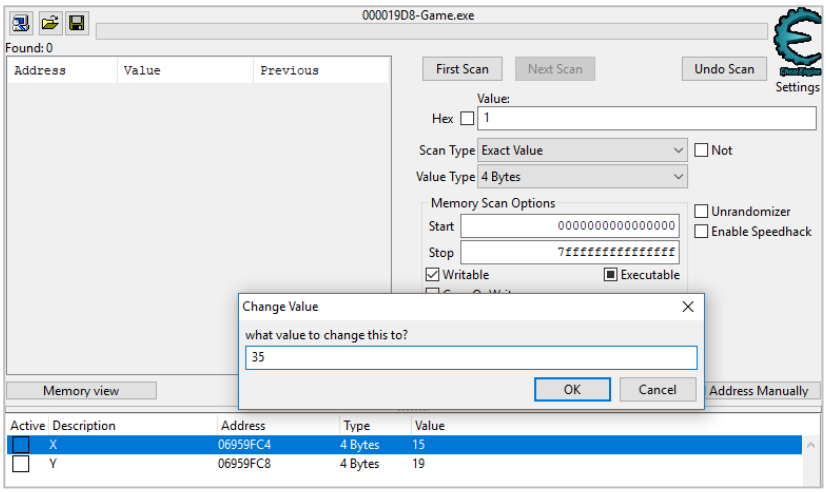
Solution by Lukasz_D

Similarly, as in the teaser challenge, let's modify the saved state of the game first. Using the RPG editor from https://f95zone.com/threads/rpg-maker-save-editors.51/ we see that there is an item called Egg available, so we can change the state such that the Bunny will have an Egg.



After relaunching the game, we see that there is a code in the Egg

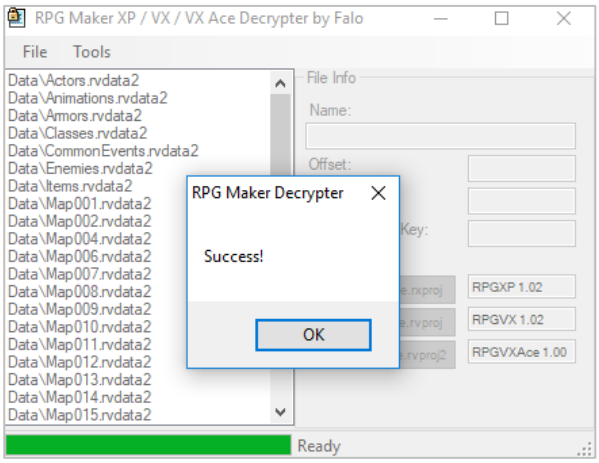
Decoding it gives: p455w0rd5_4r3_c00l, but this is not the correct password yet, as the description of the challenge says, there are several parts that needs to be combined. To find other parts we would need to explore maps of the game, unfortunately, the Bunny is trapped in a prison and cannot move freely. To change Bunny's location, I used Cheat Engine http://www.cheatengine.org/. In the Cheat Engine I found addresses in the game's memory that store X and Y coordinates of the Bunny and simply changed the X coordinate to move Bunny out of the prison



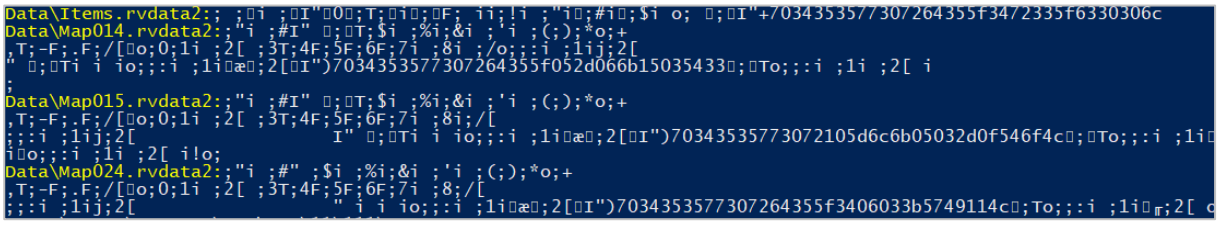
Immediately after escaping the prison, I encountered another code in the similar format as the one in the Egg.

However, this code does not decode to a printable string. Its beginning is p455w0rd5_ but the remaining part seems to be encrypted. As there will probably be more such codes in the game, searching for each of them manually would take too much time.

All data of the game are stored in Game.rgss3a file. The file is not readable, but its content can be extracted using an RPG Maker Decrypter found at https://www.reddit.com/r/FNaFBFangames/comments/3o0a7j/if_you_want_to_decrypt_any_rpg_maker_games_heres



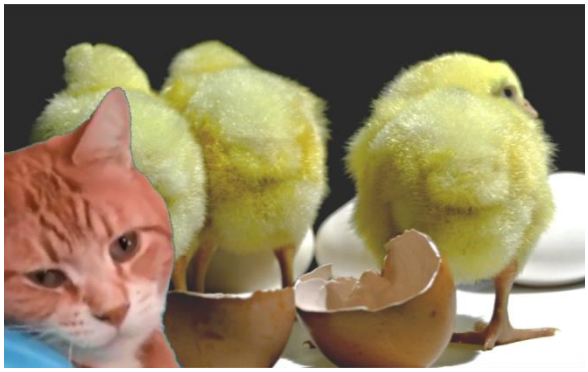
Now using PowerShell command findstr /sic:"703435" *.* in the directory where the rgss3a file was extracted we can find all codes:



So, in total we have four codes: 7034353577307264355f3472335f6330306c, 7034353577307264355f052d066b15035433, 70343535773072105d6c6b05032d0f546f4c, 7034353577307264355f3406033b5749114c but only the first one is entirely printable.

Xoring the first code with the remaining three gives: "1_54v3d_", "th3_w0rld_ ", "t0d4y! ".
Joining these three words together and removing spaces gives the correct password:
1_54v3d_th3_w0rld_t0d4y!

Egg 11 – De Egg you must



Level: medium
Solutions: 73
Author: explo1t

Challenge

Who was first, the cat or the egg?

basket.zip

Solution by pjslf

The zip archive was protected by a password so the first step was to crack it using a suitable dictionary.

```
$ fcrackzip -D -u -p ./dictionary/top_10000.txt basket.zip
PASSWORD FOUND!!!!: pw == thumper

$ unzip basket.zip
Archive:  basket.zip
[basket.zip] egg1 password: thumper
  inflating: egg1
  inflating: egg2
  inflating: egg3
  inflating: egg4
  inflating: egg5
  inflating: egg6
```

Then I looked at what I got.

```
$ file egg?
egg1: ISO Media, Apple iTunes Video (.M4V) Video
egg2: data
egg3: data
egg4: data
egg5: data
egg6: data
```

The media file looked corrupted or incomplete. The challenge description was talking about a cat and an egg so I immediately tried to concatenate those egg files using cat command.

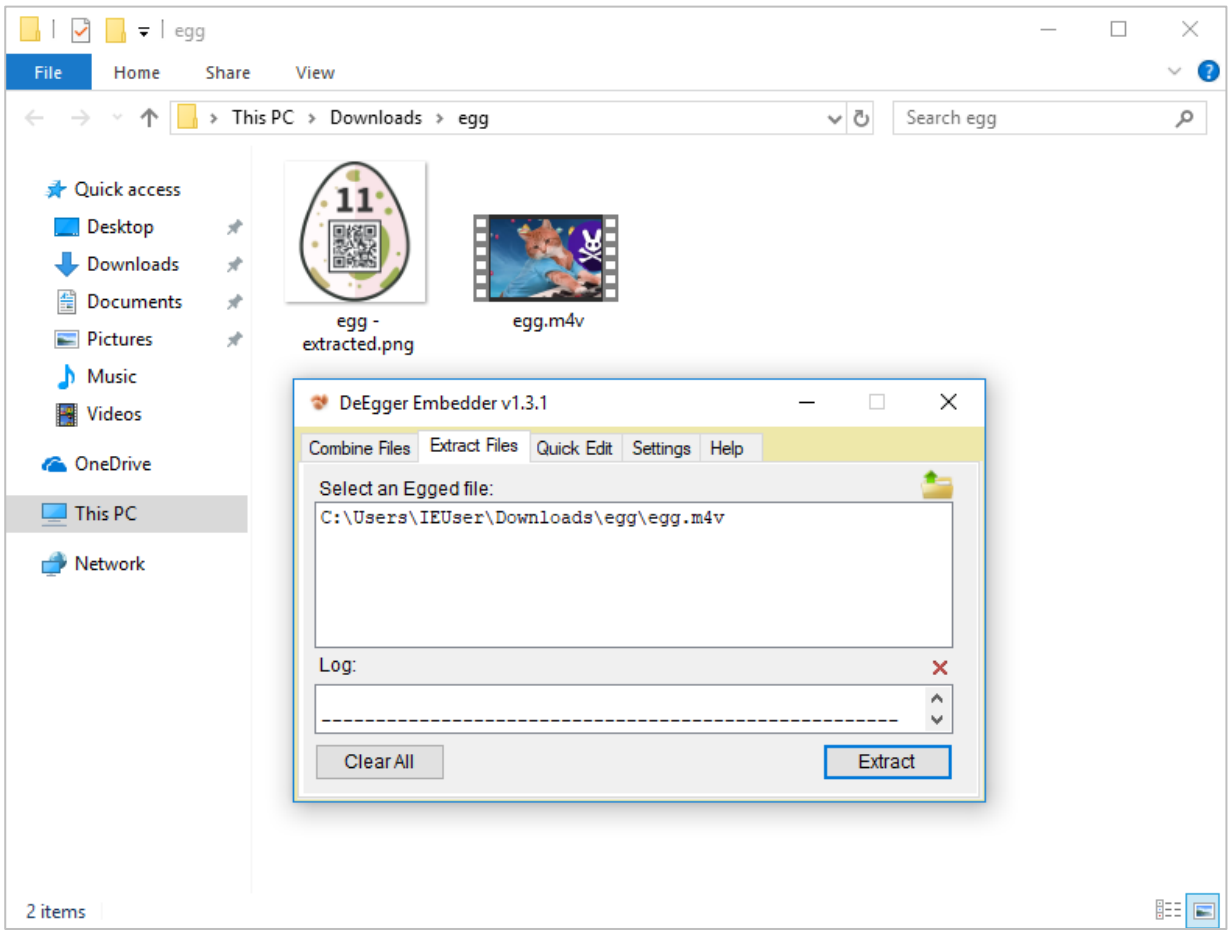
```
$ cat egg1 egg2 egg3 egg4 egg5 egg6 > egg.m4v
```

It worked and I got a playable movie file.

At this point I got completely lost. I manually inspected the movie frame by frame and discovered some suspicious black horizontal bars at the end of the movie which looked promising at first look. I fell into a rabbit hole.

It took me several days to realize I have to scrap this idea and make a step back. I started to look for a video steganography tools which might do the job. After countless attempts, I added deegg keyword from the challenge title to my google search query. Heureka! I finally found the tool I needed - DeEgger Embedder.

I used this tool to extract the egg hidden in the movie.



I must admit that this challenge was a disappointment to me. It had much higher potential.

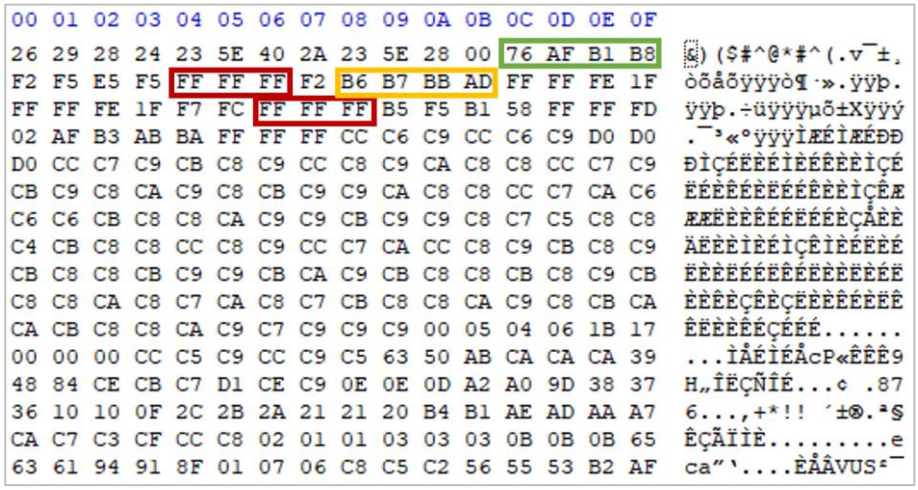
Solution by Meliver

Run fcrackzip with rockyou.txt wordlist --> **thumper**

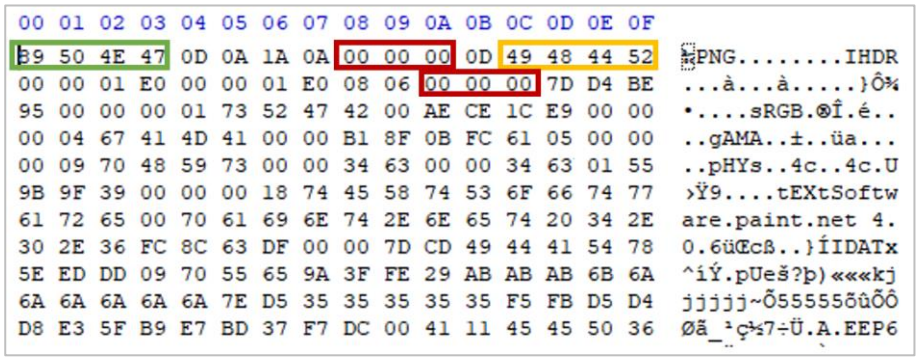
Get all the eggs and have a look at them
Looks like it is a split up mp4… oh no, cat video? ^.^

```
cat egg* > egg_complete.mp4
```

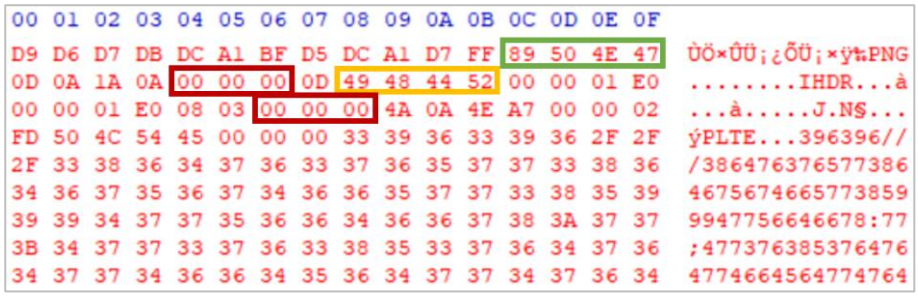
Enjoy the cat for a moment, then back tu focuz!
Have a look at the raw data. There is some strange data after the mp4 file officially has ended (moov):



It looks very much like a png but somehow wrong:



After some investigation on the steps from each character to the next, you get the hunch that it is inversed. Just XOR with FF and get the PNG.



Solution by daubsi

When we try to unzip the archive basket.zip we're asked for a password. As we're not given any hint about the password, we need to try to crack it. In order to crack a ZIP archive with john or hashcat, we need to extract the pw hash. This is done using "zip2john"

```
zip2john /tmp/basket.zip > /tmp/basket.hash
```

We use jtr for this and crack the password using:

```
daubsi@bigigloo:/tmp/JohnTheRipper/run$ ./john /tmp/basket/basket.hash
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
thumper          (basket.zip)
1g 0:00:00:01 DONE 2/3 (2018-04-16 21:18) 0.8771g/s 17712p/s 17712c/s 17712C/s
123456..ferrises
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

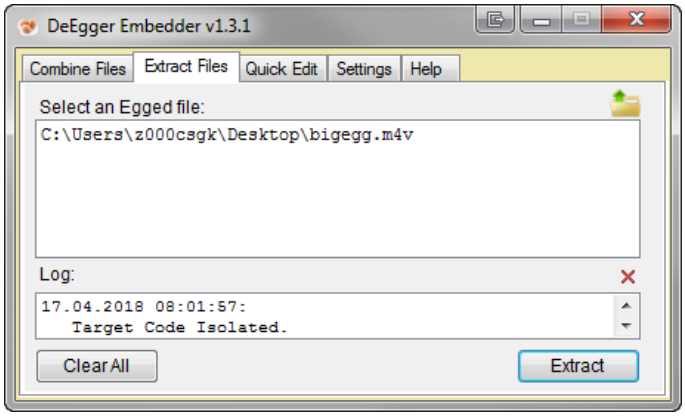
Oh, nice! The password is thumper. When we unpack the archive with the eggs we notice that all files but the last one are of equal size only the last one is smaller. This is usually an indication of having a split archive. When we look at the header

```
daubsi@bigigloo:/tmp/basket$ hexdump -C bigegg.m4v | head -n 10
00000000  00 00 00 1c 66 74 79 70  4d 34 56 20 00 00 00 01  |....ftypM4V ....|
00000010  4d 34 56 20 6d 70 34 32  69 73 6f 6d 00 72 db 1c  |M4V mp42isom.r..|
00000020  6d 64 61 74 00 00 0e 1b  65 88 80 40 07 6c 98 a0  |mdat....e..@.l..|
00000030  00 22 4b 27 27 27 27 27  27 27 27 27 27 27 27 27  |."K''''''''''''''|
00000040  27 27 27 27 27 27 27 27  27 27 27 27 27 27 27 27  |''''''''''''''''|
*
00000e40  27 27 80 00 00 15 d3 41  9a 02 05 8a df 8c aa aa  |''.....A........|
00000e50  aa aa aa aa aa aa c4 e2  f1 3e 27 c4 f8 9f 13 e2  |.........>'.....|
00000e60  7c 4f 89 f1 ba f6 27 58  9f 1b 8a eb 13 e2 7c 4f  ||O....'X......|O|
00000e70  89 f1 3b c4 f8 9f 13 e2  7c 4e b1 3e 37 be 27 c4  |..;.....|N.>7.'.|
```

We notice, that this seems to be an MP4. We therefore join all the file into a big one:

```
cat egg1 egg2 egg3 egg4 egg5 egg6 > bigegg.m4v
```

When we watch the video we see a cat playing a keyboard, but no indication of how to proceed. binwalk, stegoveritas and all the other tools of the trade show no indication of hidden data. When we google for "deegger" we can find a tool called "Deegger Embedder" from Z.A. Software which automatically extracts the egg for us. The software can be downloaded from http://download.cnet.com/DeEgger-Embedder/3000-2144_4-75710065.html



Alternatively, we can inspect the “atom” structure of the MP4 with the tool Atomic Parsley (http://atomicparsley.sourceforge.net/)

```
D:\>AtomicParsley.exe r:\tmp\basket\bigegg.m4v -T
[…]
```

The last one looks weird… We extract the contents starting at byte 7537658 to a new file called “parsley”. Then we use xorbrute.py to look for the string “PNG” – and we are lucky! It is found with xor byte 0xff.

```
daubsi@bigigloo:/tmp/basket$ python2 ./xorBrute.py  -f parsley -s PNG -x 1
..
..
'PNG' occurred 1 times when XOR'd with 0xff
```

Finally we decode file with this little python script:

```
def xor(data, key):
    return bytearray(((data[i] ^ key) for i in range(0,
len(data))))

fname = "parsley"
fh = open(fname, "rb")
b = bytearray(fh.read())
fh.close()
xorData = xor(b, 0xff)
fname = "egg11.png"
fh = open(fname, "wb")
fh.write(xorData)
fh.close()
```

Egg 12 – Patience

Level: medium
Solutions: 219
Author: PS

Challenge

Solution by blaknyte0

I created an Android Virtual Machine, installed the HackyEaster App and let it run for a few days.
(Turn on „always active“ in developer options.).

Solution by HaRdLoCk

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:

patience.py

```
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'
```

Solution by LlinksRechts

Since I am definitely not waiting 1000000 seconds = ~11 days (even though that is actually a viable possibility), I decompiled the Android app. In the challenge, a request is sent to ps://count every three seconds containing the current count as well as the has returned for the last request, starting with 100000 and genesis respectively. The method in the Java code handling this request looks like this:

```
private String handleCount(String var1, String var2) {
    if(var2 != null && var1 != null) {
        try {
            Integer var4 = Integer.valueOf(Integer.parseInt(var2));
            String var5 = sha1(var1 + var2);
            if(var4.intValue() >= 0) {
               String var6 = "{ \"h\":\"" + var5 + "\", \"c\":\"" + (-1 + var4.intValue()) + "\" }";
               return var6;
            }
         } catch (Exception var7) {
            ;
        }
    }

    return null;
}
```

To sum up, it concatenates the hash and the count, calculates the sha1 value, and returns this as a new hash. This can be emulated in python:

```
count=100000
hash='genesis'
from hashlib import sha1
while 1:
    hash = sha1((hash+str(count)).encode("UTF-8")).hexdigest()
    count -= 1
    if count == 0:
        break

print(hash)
 > dd6f1596ab39b463ebecc2158e3a0a2ceed76ec8
```

When this is input into https://hackyeaster.hacking-l ab.com/hackyeaster/images/eggs/HASH.png, the egg is revealed.

Egg 13 – Sagittarius...

Level: medium
Solutions: 84
Author: inik

Challenge

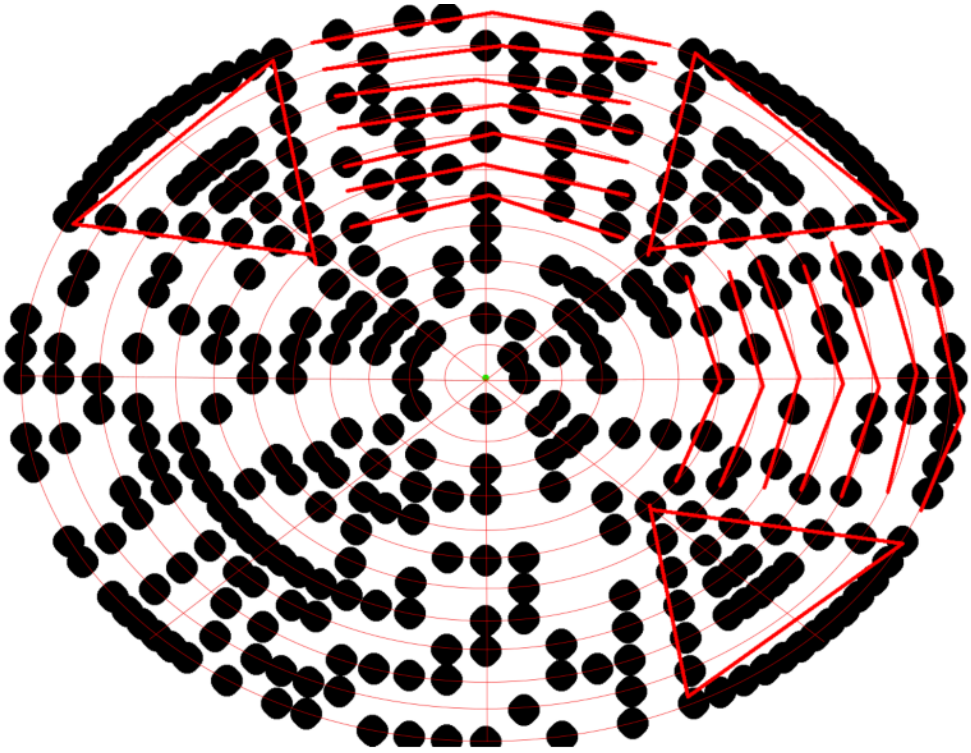
... is playing with his pila again.

Can you find the Easter egg QR code he has hidden from you?

pila.kmz

Solution by LlinksRechts

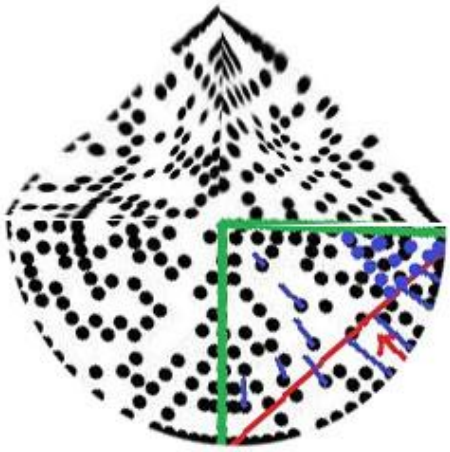
When examining the pattern closely, one can notice that it is a QR code distorted to an elliptical shape.



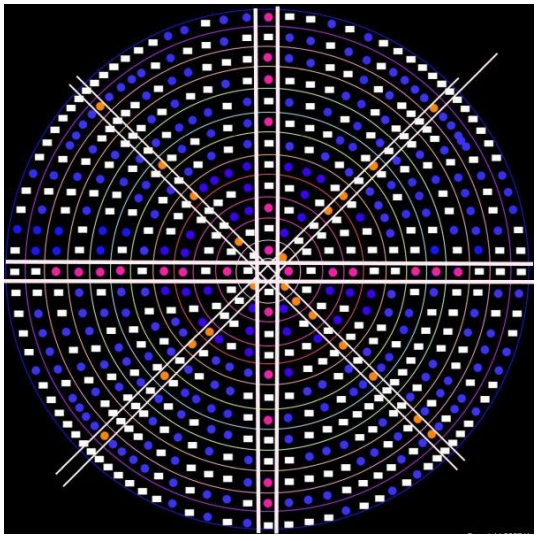
I drew the QR code in Gnumeric (probably not the most efficient method to do it my hand, but it worked) to get the original code:

Solution by opasieben

Opening the kmz file with Google Maps, some custom Waypoint were visible. These looked like a circular QR code. I made a very simple PoC with photoshop.



I tried to figure out a fitting algorithm to do this, but finally went with the manual way. Creating elipses, the missing points and some help lines to transfer the circle into a 25x25 excel matrix.



Solution by Darkice

We can open the KMZ file with Google Earth or Marble and we will see some coordinates as stars on the map.

The coordinates are arranged in a circle, but a closer look already shows the characteristics of a QR code. So, we only need to map the coordinates of the circle to those of a square to get a real QR code. This can be done using some trigonometric functions.

$$x_{square} = a(\varphi) * x_{circle}$$

$$y_{square} = a(\varphi) * y_{circle}$$

Where

$$\varphi = \left|\tan^{-1}\frac{y_{circle}}{x_{circle}}\right|$$

and

$$a(\varphi) = \begin{cases} \frac{1}{\cos\varphi}, & \varphi \leq \frac{\pi}{4} \\ \frac{1}{\sin\varphi}, & \varphi > \frac{\pi}{4} \end{cases}$$

Since only values between -1 and 1 can be used for the calculation, the coordinates must be normalized beforehand. This can easily be done by subtracting an offset from both the x and y coordinates, because the distance between the outer coordinates is 2.

We can use a python script to calculate the coordinates for the square and save the result as an image.

```
import re
from PIL import Image
from math import atan,fabs,pi,cos,sin

coords = re.findall('[0-9-.]+[,][0-9-.]+', open('pila.kml').read())
x0 = 120.0
y0 = -45.5
img = Image.new('RGB', (460 ,460), color = 'white')
pixels = img.load()
for i in coords:
    c = i.split(',')
    xCirc = float(c[0]) - x0
    yCirc = float(c[1]) - y0
    if xCirc == 0 or yCirc == 0:
        xSquare = xCirc
        ySquare = yCirc
    else:
        phi = fabs(atan(yCirc/xCirc))
        if phi <= pi/4:
            a = 1/cos(phi)
        else:
            a = 1/sin(phi)
        xSquare = xCirc * a
        ySquare = yCirc * a
    xImg = int((xSquare+1) * 200 + 20)
    yImg = int((ySquare+1) * 200 + 20)
    for x in range(18):
        for y in range(18):
            pixels[xImg+x,yImg+y] = (0,0,0)
img.save('qr13.png')
```

Egg 14 – Same same...

Level: medium
Solutions: 195
Author: Lukasz_D

Challenge

...but different!

Upload the right files and make the server return an Easter egg!

```
http://whale.hacking-lab.com:4444
```

upload.php.txt

Solution by horst3000

Needs: Two QR Codes which are “Hackvent” and “Hacky Easter”. However the equality function of the hashes of these images should return true.

First try

Magic Hashes -> Need to begin with “0e”

Create QR Code. Modify ending of image until hash is reached.

```
Same same?
Well done. You brute-forced the PHP == collision. Nevertheless, to get the flag
you need to come up with the === collision. Keep trying.
Hint: The uploaded QR code does not have to be in an image file. You can also
put it into a PDF...
```

Second try

pdf int -> shatterd (pdf with the same hash but different pictures in it)

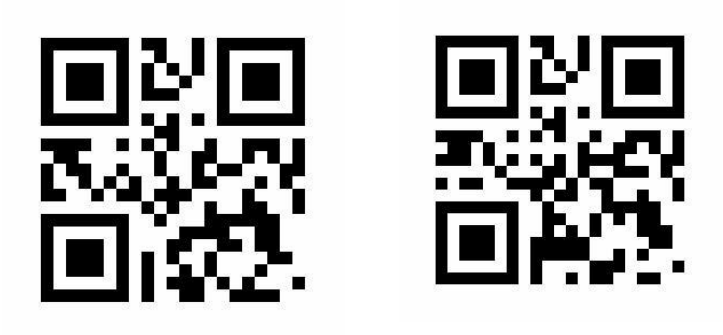
use this service: https://alf.nu/SHA1

receive qr.

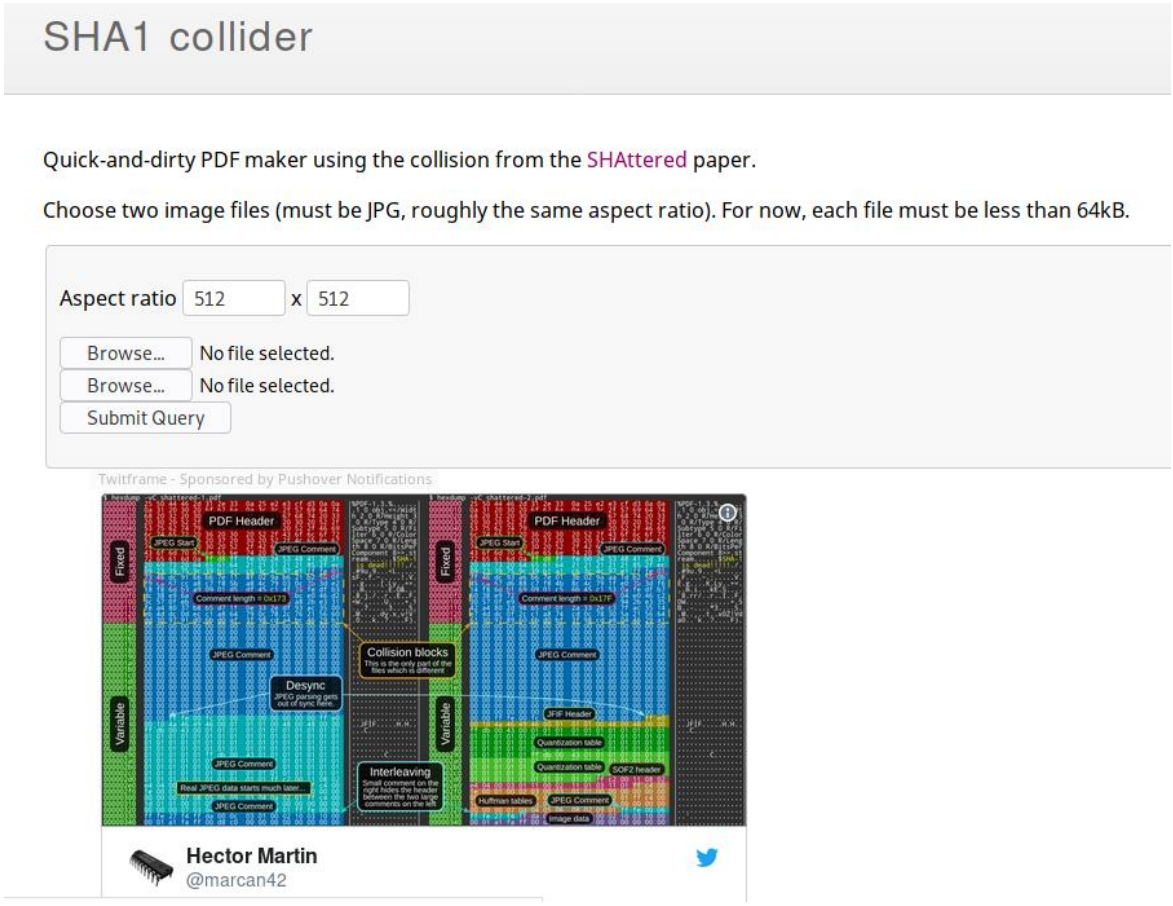
Solution by Eydis

In this challenge I had to make two files, with different QR codes, but same SHA1 hashes. I found a website (https://alf.nu/SHA1), that generates PDF files with SHA-1 collision and uploaded two .jpg images with following QR-codes:

Hackvent **Hackyeaster**



The website generated two PDF files with the same SHA-1 hash:



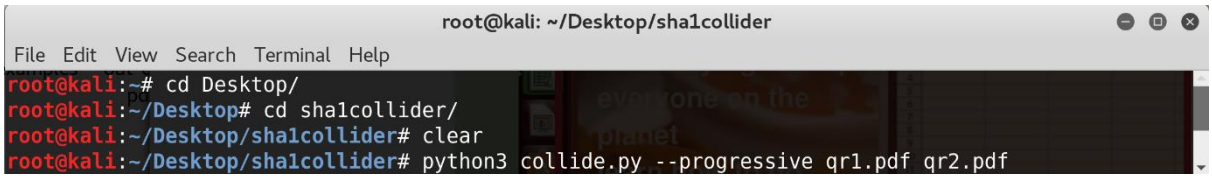
I uploaded those PDF files to a challenge website and received the egg.

Solution by mezuru

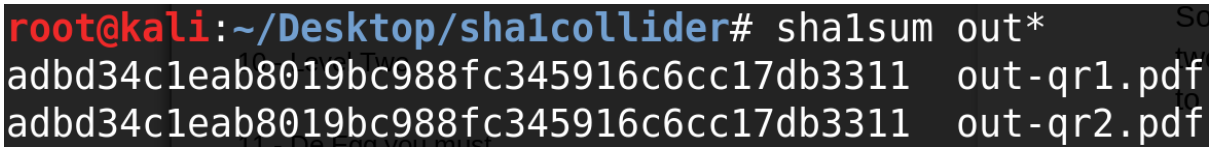
This was an interesting challenge. We are expected to upload two files using the provisioned URL. Upon examining the PHP, it turns out that the page is looking for two QR codes, one that says “Hackvent” and the other “Hacky Easter” but the catch here is that the two QR codes must have a matching SHA1 value.

My starting point was here: https://shattered.io/ where they demonstrate the weakness in SHA1 and how you can have two PDF files with the same SHA1 hash.

So I used the sha1collider script (source: https://github.com/nneonneo/sha1collider) to create two new files with the same hashes and upload them in the webpage to get the egg. In order to do this I ran the following command on the following QR codes (in pdf format):

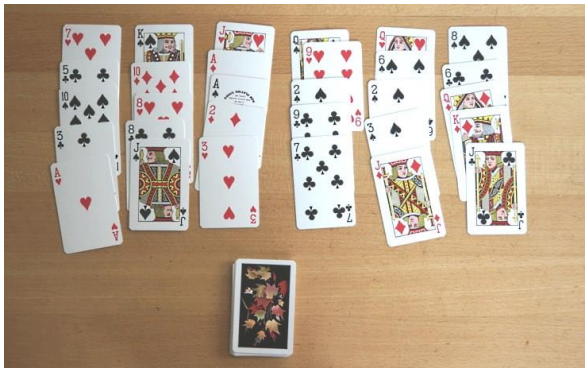


When running a sha1sum on the output file I get the following, same sha1 hashes:



Uploading the two files give you the egg.

Egg 15 – Manila greetings



Level: medium
Solutions: 213
Author: brp64

Challenge

Randy Waterhouse receives a package from his friend Enoch Root containing a deck of cards and a letter:

Dear Randy,

even though our stay in Manila was not very pleasant, I fondly think of our discussions there:

```
GTIFL RVLEJ TAVEY ULDJO KCCOK P
```

Wishing you happy Easter

Enoch

Decrypt the message and enter the password in the Egg-o-Matic below. **Uppercase only!**

deck

Solution by Floxy

The keywords “Deck of cards” and “cipher” leads me to well-known Solitaire-Cipher, so I started coding a little C#-Tool because I found a library on following site
https://www.schneier.com/academic/solitaire/

With the parsing part of the deck following website helped me:
http://jnicholl.org/Cryptanalysis/Ciphers/Solitaire.php

After executing my little script I got:

```
static void Main(string[] args)
{
    string msg = "GTIFL RVLEJ TAVEY ULDJO KCCOK P";
    List<int> algo = GetCipherAlgo();
    Solitaire crypt = new Solitaire(algo.ToArray());
    Console.WriteLine(crypt.GetDeck());

    for (int i = 0; i < 1; i++)
    {
        msg = crypt.Decrypt(msg);
        Console.WriteLine(msg);
    }
}
```

```
THEPASSWORDISCRYPTONOMICON
```

Entering “CRYPTONOMICON “ in Egg-o-Matic leads to egg:

Solution by Darkice

For this challenge we were given a ciphertext and a card deck. One cipher using cards as encryption keys is the solitaire cipher. To decrypt the message, we can use an online tool.

https://ermarian.net/services/encryption/solitaire

Since the tool uses a different notation for the cards, we had to convert it beforehand.

```
deck = open('deck.txt').readlines()
d = ''
for i in deck:
    if 'jr' in i:
        d += 'A'
    elif 'jb' in i:
        d += 'B'
    else:
        d += i[1:-1].upper().replace('10','T') + i[0]
    d = d + ' '
print d
```

Key:

```
8d 3s 7d 3d 2c 5s Ad 6c 7s 6d A Kd Qh Js Jc 7h 3h 9h 9s 8s 9c As 4h 8c
3c Kh Ah 6s 6h Ts Ks Ac Td Qd Qc B Qs 4s 9d 2s 5c Jh Th 4c Tc 5d 8h 2h
2d Jd 7c Kc 5h 4d
```

After the decryption we got the following message:

THE PASSWORD IS CRYPTONOMICON

Solution by sym

As the image and the text file name indicate, it has something to do with playing cards. After a quick search, I found the Solitaire cipher which was created by the famous Bruce Schneider.

Then I found a Python implementation by Jesux:
https://gist.github.com/jesux/0a2d243b3fdcc8827adf

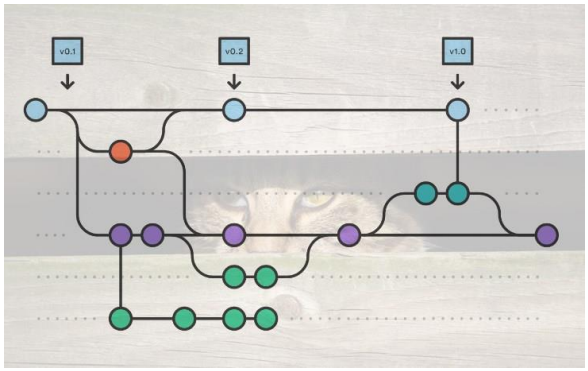
Now I only needed to convert the provided playing cards to numbers as described in the script and run it:

```
# card numbering:
#  1, 2,...,13 are A,2,...,K of clubs
# 14,15,...,26 are A,2,...,K of diamonds
# 27,28,...,39 are A,2,...,K of hearts
# 40,41,...,52 are A,2,...,K of spades
# 53 & 54 are the A & B jokers
```

```
PS C:\Temp\15> python solitaire-inverse.py -d "GTIFL RVLEJ TAVEY ULDJO KCCOK P" "21, 42, 20, 16, 2, 44, 14, 6, 46, 19, 5
3, 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, 3
7, 36, 4, 10, 18, 34, 28, 15, 24, 7, 13, 31, 17"
GTIFLRVLEJTAVEYULDJOKCCOKP
[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]
THEPASSWORDISCRYPTONOMICON
```

The password is: CRYPTONOMICON

Egg 16 – git cloak --hard



Level: medium
Solutions: 168
Author: PS

Challenge

This one requires your best Git-Fu! Find the hidden egg in the repository.

repo.zip

Solution by 0x90v1

This challenge is interesting. I just found out at least two possible solutions how to solve this challenge.

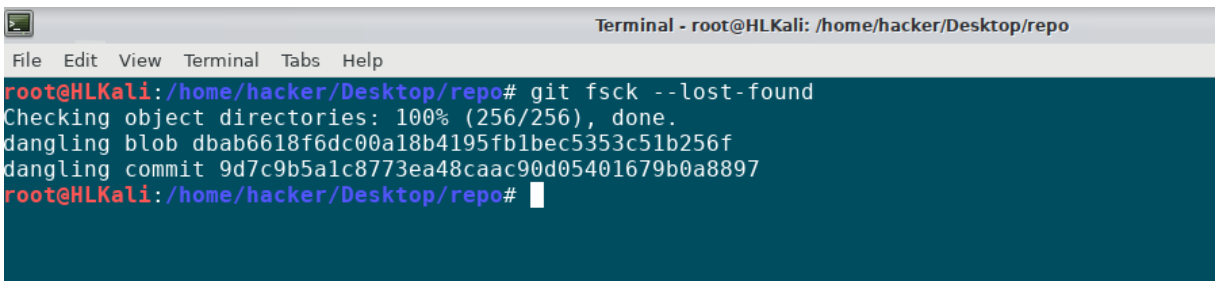
The first one I did was just checking the repository and search for PNG with notepad++. On this way, I quickly found something interesting under the following path:

```
.git\objects\db\ab6618f6dc00a18b4195fb1bec5353c51b256f
```

That looks like it could be a PNG image. Checked it with the HEX editor and removed everything in front of the PNG tag. QR code revealed in that way after I opened it with an image viewer.

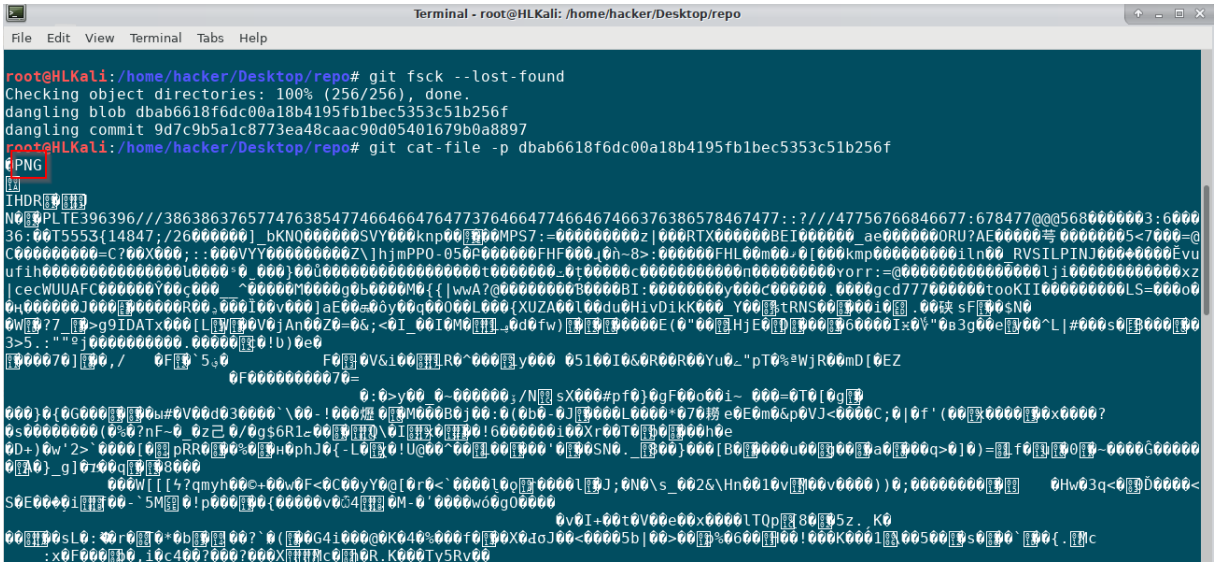
I was thinking after words, that this cannot be the only one solution so I google it for some special git commands and found out how to restoring not yet versioned changes. With the following command, I found a blob and a commit:

```
git fsck --lost-found
```



Now I wanted to know what is inside this blob. With the following command, I was able to see what was inside this blob:

```
git cat-file -p dbab6618f6dc00a18b4195fb1bec5353c51b256f
```

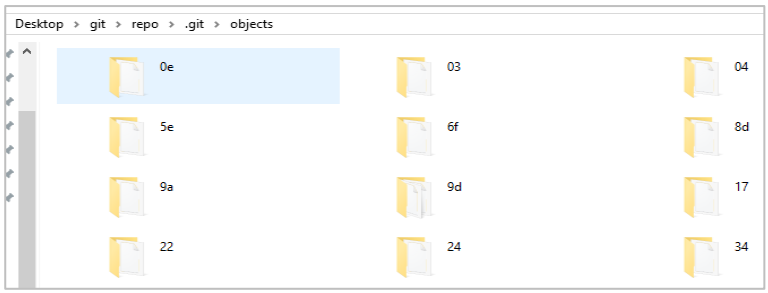


It turns out that it was a PNG so I just had to pipe the output directly into a PNG file and got the final egg for this challenge.

Solution by markie

In Continuous Integration version control - “cloak” means to exclude specific folders/files from a repo. So a file exists in this git repo, but cannot be seen in the commits.

All the images (png and jpg) in this repo are saves a blob files in git (binary large objects). All blogs and tree information is store in .git/objects. This folder contains some 26 images and trees. All these objects are sha1 of the objects in the repo.



All that is needed now is some git-fu to work out which SHA1 does not appear in the commits, and you should have the SHA1 of the egg.

Open a git bash window:

```
$ git reflog                    <- shows the commits and branches
$ git checkout HEAD@{x}         <- use this to jump around the commits
$ git log --stat                <- to see commit data  NB; q to quit#
$ git status -v                 <- show head and branch info
$ git ls-files -v --stage -s    <- show the SHA1 of the files in that commit
```

So the only sha that does not appear in the repo as a png, jpg, tree or commit is:
dbab6618f6dc00a18b4195fb1bec5353c51b256f.

Decompressing this with zlib (see below python), shows the file as bytes output. In this we see it is a .png file, so could be the missing egg!

```
In [22]: decomp[0:200]
Out[22]: b'blob 15486\x00\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x01\xe0\x00\x00\x01\xe0\x08\x03\x00
4664764773764664774664674663763865784674 77:::?///47756766846677:678477@@@568\xff\xfa\xfb\xf9\xe4\xe83:6\x
```

Decompress the sha1, convert the bytes to hex, convert hex to png and save the file:

```
import zlib
import binascii
from PIL import Image
from PIL import ImageDraw
import io

# load the git blob
f ="repo/.git/objects/db/ab6618f6dc00a18b4195fb1bec5353c51b256f"
compressed = open(f,'rb').read()

decomp = zlib.decompress(compressed)          #decompress the git blob
png = decomp[11:]                             #remove first 11 bits from blob
png = binascii.unhexlify(png)                 #convert bytes to hex

#convert to png & save
stream = io.BytesIO(ong)
img = Image.open(stream)
draw = ImageDraw.Draw(img)
img.save("egg16.png")
```

Solution by jcel

The challenge consisted in a zip file containing a git repository. It contained some images, none of which contained the desired QR code.

However, since git stores all versions of all files in the .git/objects directory, the following shell commands can be used to extract all of them (only the ones that start with "blob" are relevant here):

```
for i in `find . -type f` ; do
  echo $i
  h=`unpigz -c $i | hexdump -C | head -1 | fgrep blob`
  if [ -n "$h" ]; then
    b=`echo $i | tr -d '[/.]'`
    unpigz -c $i >../../files/$b
  fi
done
```

Removing the "blob [09-a-f]*" prefix from the files resulted in viewable JPG and PNG files. Of these, it can be easily seen that the file

```
.git/objects/db/ab6618f6dc00a18b4195fb1bec5353c51b256f
```

contains the correct egg.

Egg 17 – Space Invaders

Level: medium
Solutions: 333
Author: PS

Challenge

Alien space invaders left a secret message. Luckily, you know that they used codemoji.org for the encryption.

Decrypt the message, and save the planet!!

invaders_msg.txt

Solution by Buge

The site codemoji.org didn't seen to have an easy way to enter text to decode.
I managed to get it to decrypt by going to:
https://codemoji.org/#/encrypt

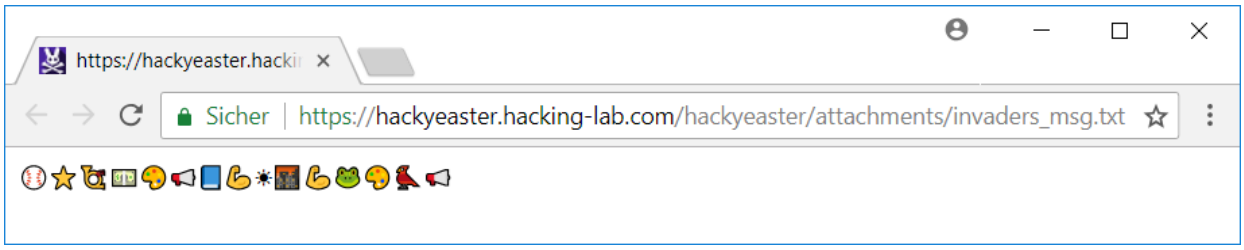
Then entering some arbitrary text in the message box, then clicking on the space invader emoji (👾 which can also be determined by googling space invader emoji). Then clicking share this message. Then I copied the link and visited it. Then I clicked on decipher it. Then I used chrome's inspect element on the message on the left, and changed the text attribute on the div from the existing emoji to the emoji from the challenge
⚾⭐📯💵🎨📢📄💪☀🏭💪🐸🎨🐦📢

Then I clicked on the space invader emoji and it gave me the message
invad3rsmustd13
I entered that into the box and got the egg.

Very strangely if I don't select the space invader for the initial useless encryption step, it doesn't work. But that should have no effect, because I'm deleting that ciphertext. My only conclusion is that codemoji is bad and is sending the key or something similar to it through a side channel.

Solution by scryh

The challenge provides a text-file invaders_msg.txt containing unicode-encoded smileys:

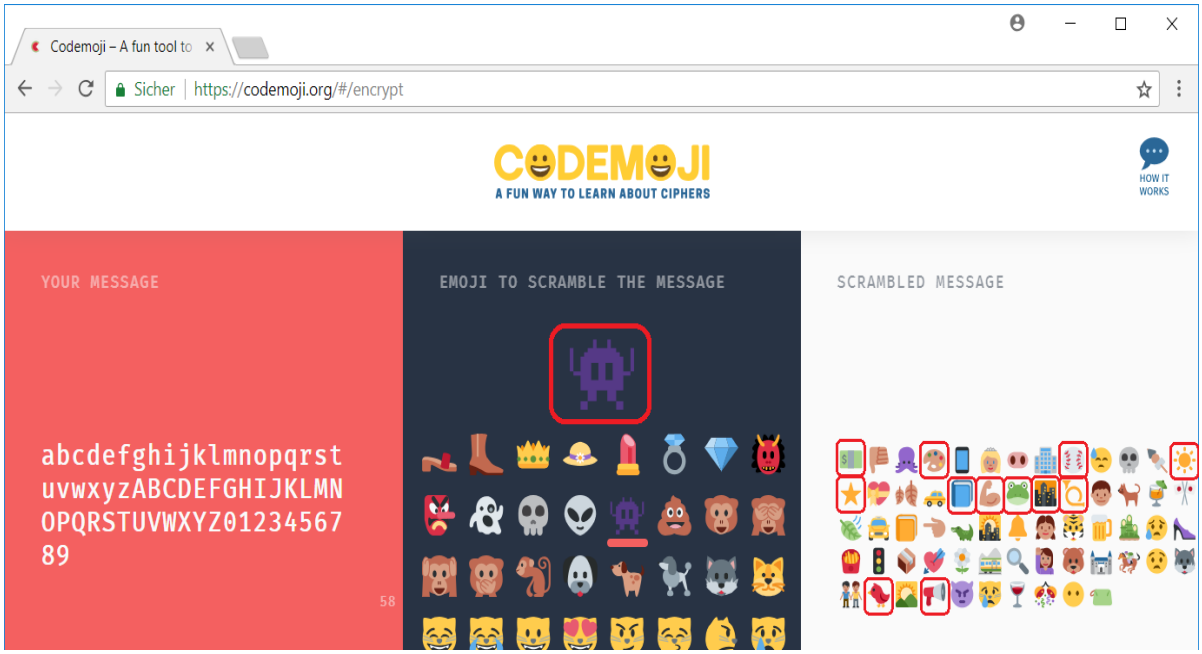


Also, there is a hint that the message encoded in the text-file has been created using codemoji.org.

On the website a message can be entered, which is encrypted by selecting one of a few hundred smileys. The ciphertext is a series of smileys just like the provided invaders_msg.txt.

I was a little bit lucky solving this challenge, because before actually starting to understand the encryption-mechanism I decided to test a few smileys with the text abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 looking for smileys of the provided ciphertext.

As there are a few pixel-invaders on the image of the challenge, I also tried the following smiley and noticed the smileys from the encrypted message on the right side:

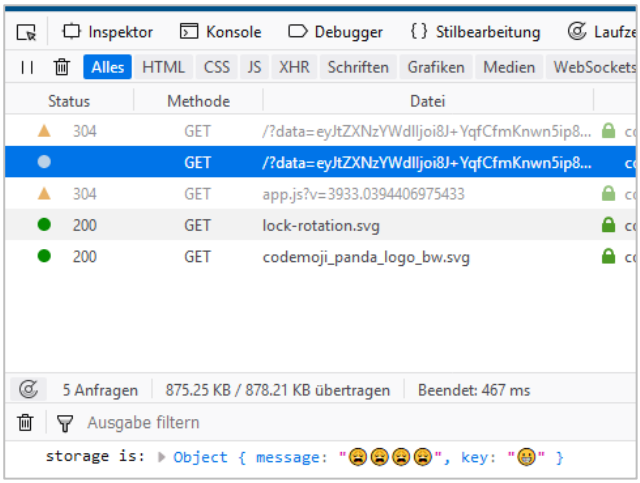


Since the mapping from characters to smileys is one-to-one, the only thing left doing was to see which smiley equals which character:

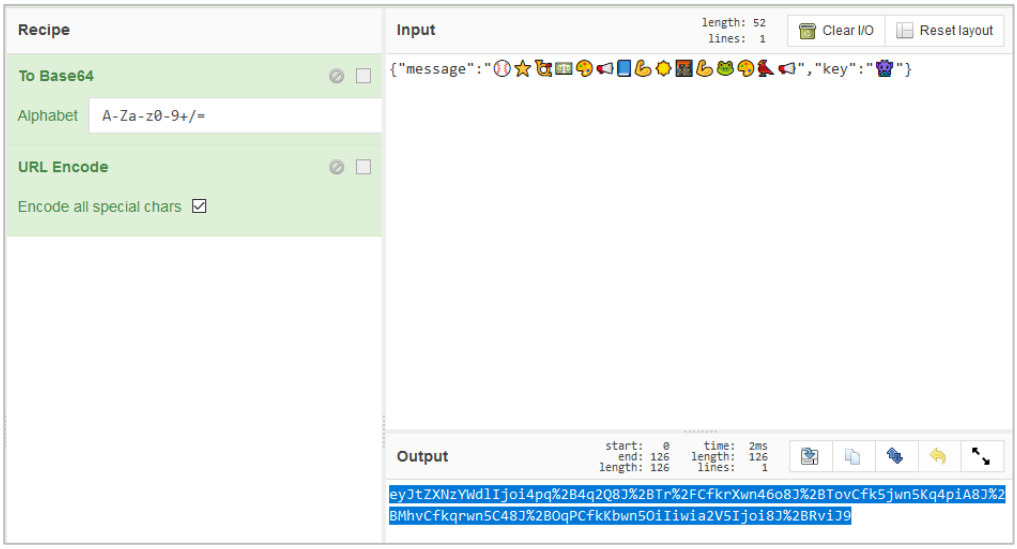
The password is invad3rsmustd13.

Solution by TheVamp

We know that the encryption was done with https://codemoji.org/. I analyzed the website and saw, that you can generate your own landing page, without knowing the key:

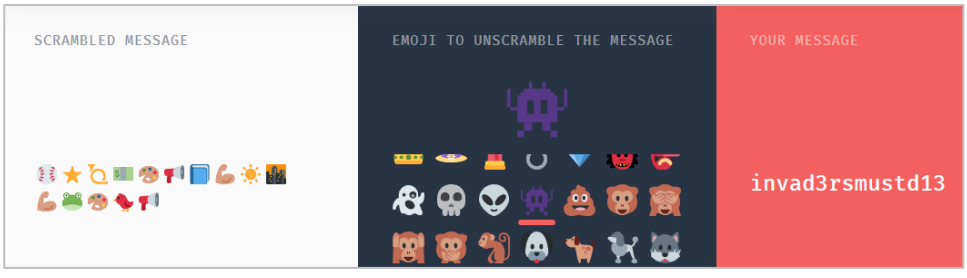


As you see it is a base64-json message with the key and the message. I used https://gchq.github.io/CyberChef/ to generate my own base64-json message. I used the space invader icon, just for fun. I mean it is at least the hint of the message:



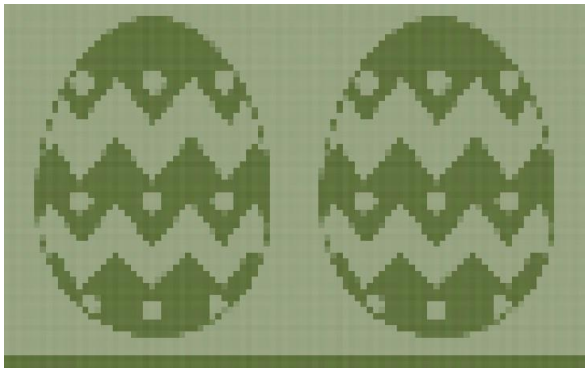
https://codemoji.org/?data=eyJtZXNzYWdlIjoi4pq%2B4q2Q8J%2BTr%2FCfkrXwn46o8J%2BTovCfk5jwn5Kq4piA8J%2BMhvCfkqrwn5C48J%2BOqPCfkKbwn5OiIiwia2V5Ijoi8J%2BRviJ9#/landing

Notice, that I added a “#/landing” after the base64-json data, so that I got to the landing page. Otherwise the script will break :)



And we got the next egg.

Egg 18 – Egg Factory



Level: medium
Solutions: 154
Author: Kiwi.wolf

Challenge

Make the egg factory write a secret word!

Then enter it into the Egg-o-Matic below, **uppercase and underscores only.**

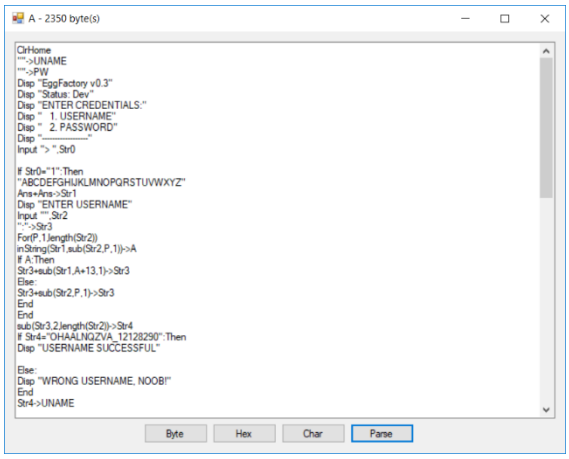
A.8xp

Solution by scryh

The provided file `A.8xp` is a program for the TI-83+ Graphing Calculator:

```
root@kali:~/Documents/he18/egg18# file A.8xp
A.8xp: TI-83+ Graphing Calculator (program)
```

I used a TI-83+ program (.8xp) Interpreter to disassemble the file:



The program seems to ask for a username and a password. But the interesting part is at the end of the output:

```
ClrDraw:AxesOff:expr(Str5)*0.01->A:Line(-
1.7067137809187278*A,1.1201413427561837*A,-1.6042402826855124*A,0.76
67844522968198*A):Line(-4.54,2.17,-4.08,2.57):Line(-
[…]
```

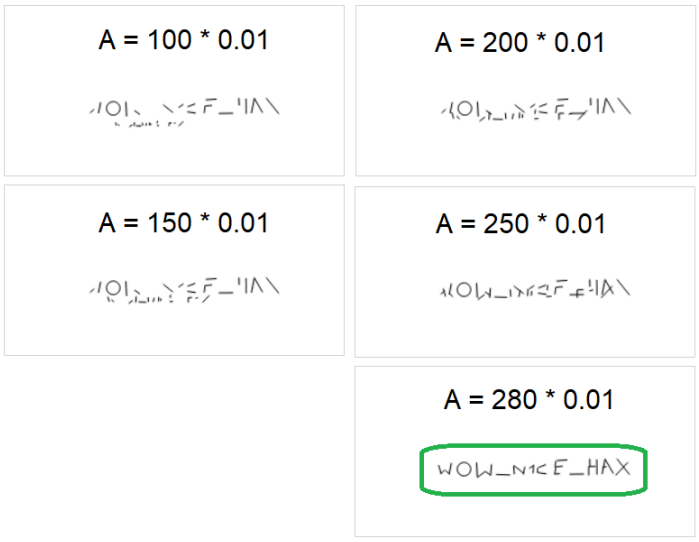
Obviously some lines and a circle are drawn here. Some of the coordinates are multiplied with the variable `A`, which has been initialized with `Str5*0.01` (`expr(Str5)*0.01->A`). `Str5` seems to be the entered password:

```
...
Disp "ENTER PASSWORD"
Input "",Str5
...
```

I decided to adapt the program for javascript in order to draw the lines and try different values for the value `A`:

```
<html>
<body>
<canvas id="myCanvas" width="300" height="150" style="border:1px solid
#d3d3d3;"></canvas>
<script>
function toPixel(x, min) {
  var r = x;
  if (min) r = -r;
  r = (r + 14)*8;
  return r;
}
var A = 280 * 0.01;
var arr = [
[-1.7067137809187278*A,1.1201413427561837*A,-1.6042402826855124*A, […] ];
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.beginPath();
for (var i = 0; i < arr.length; i++) {
  console.log(toPixel(arr[i][0]));
  ctx.moveTo(toPixel(arr[i][0]),toPixel(arr[i][1], true));
  ctx.lineTo(toPixel(arr[i][2]),toPixel(arr[i][3], true));
}
ctx.stroke();
ctx.beginPath();
ctx.arc(toPixel(-1.96), toPixel(2.67, true), 6, 0, 2 * Math.PI);
ctx.stroke();
</script>
</body>
</html>
```

It turned out to be quite easy, because the value for `A` can be adjusted gradually until a clear text is visible:



The password is `WOW_N1CE_HAX`.

Solution by Lukasz_D

The provided file turns out to be a program for a TI calculator. Decompiling it can be easily performed using an online service at https://www.cemetech.net/sc/. The last line of the program will draw several lines, parameters of some of the lines are derived from user input.

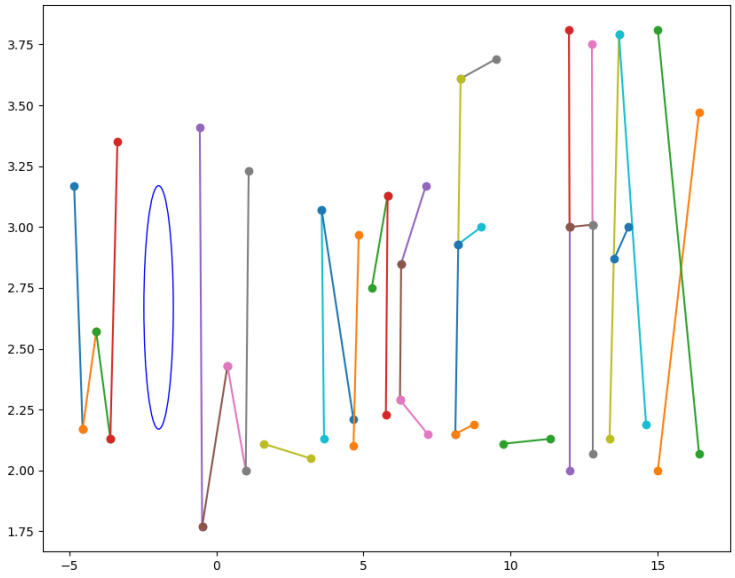
This seems like if we knew the correct input, lines will create a password needed for the egg. I assumed that at least some of the lines will end in places where other lines begin, so let's calculate for which parameter the most points that are positioned according to user input will overlap with the fixed points. The following script will automate the calculations:

```
1.  lines = "ClrDraw:AxesOff:expr(Str5)*0.01-
    >A:Line(~1.7067137809187278*A,1.1201413427561837*A,~1.6042402826855124*A,0.76678445
    22968198*A):[CUT]:Line(15,3.81,16.4,2.07)".replace('~', '-')
2.
3.  fixedpoints_re = re.findall('Line\(([-]?\d+\.?\d*),([-]?\d+\.?\d*),([-
    ]?\d+\.?\d*),([-]?\d+\.?\d*)', lines)
4.  fixedpoints = []
5.  for f in fixedpoints_re:
6.      fixedpoints.append([float(f[0]), float(f[1])])
7.      fixedpoints.append([float(f[2]), float(f[3])])
8.
9.  scalablepoints_re = re.findall('Line\(([-]?\d+\.?\d*)\*A,([-]?\d+\.?\d*)\*A,([-
    ]?\d+\.?\d*)\*A,([-]?\d+\.?\d*)\*A', lines)
10. scalablepoints = []
11. for f in scalablepoints_re:
12.     scalablepoints.append([float(f[0]), float(f[1])])
13.     scalablepoints.append([float(f[2]), float(f[3])])
14.
15. # check whether the scalable points can be placed in the same place (with accuracy
    ACC) where fixed points are, if yes, then calculate the scaling factor A
16. ACC = 0.0000001
17. for sp in scalablepoints:
18.     for fp in fixedpoints:
19.         if(abs(sp[1]/sp[0] - fp[1]/fp[0])<ACC):
20.             print "same spot: A=" + str(fp[1]/sp[1])
```

After running the script, it became obvious what should the value of parameter A:

```
same spot: A=2.83
same spot: A=2.83
same spot: A=2.83
same spot: A=2.83
same spot: A=2.83
same spot: A=2.83
same spot: A=2.83
same spot: A=2.83
same spot: A=2.83
same spot: A=2.83
same spot: A=2.83
same spot: A=2.83
same spot: A=2.83
same spot: A=2.83
same spot: A=3.87671232877
same spot: A=2.83
same spot: A=2.83
same spot: A=2.83
same spot: A=2.83
```

Drawing all the lines (and one circle, the position of which was not dependent on user input) with A=2.83 returned the following image:



Entering the password: WOW_NICE_HAX resulted in the egg.

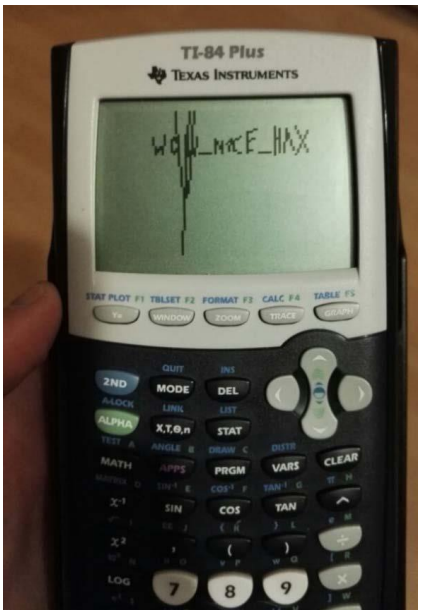
Solution by MaZeWindu

The attachment was a .8xp file, which is a program for the Texas-Instruments 83+ Graphing Calculator. The code was written with SourceCoder 3 (https://www.cemetech.net/sc/) and can be viewed with it. The site has an TI-emulator too, but the program didn't run on it properly. I own an TI84+ and used it for this challenge. At the start, there are three possibilities:

```
1 Enter Username, 2 Enter Password and 3 Seeeeecret.
```

Option 3 prints a graph, but for the calculation the correct password is needed (the username can be left out). I looked up the commands I didn't know on http://tibasicdev.wikidot.com/ and made a little python script that simulates the calculation of the correct password:

```python
#!/bin/python

import math

for m in range(1,9999):
    M = m
    N = 8956
    C = 1
    E = 0

    while(N>0 or M>0):
        N = 0.5 * math.floor(N)
        M = 0.5 * math.floor(M)
        nfrac = math.modf(N)[0]
        mfrac = math.modf(M)[0]
        x = int(nfrac != mfrac)
        E = E + (C * x)
        C = C*2
    if E == 9191:
        print(m)

print('done')
```



This way I get the correct code: 283. Now I have to adjust the window in which the graph is shown, and I get t he flag:

```
WOW_NICE_HAX
```

Egg 19 – Virtual Hen

Level: hard
Solutions: 45
Author: jcel

Challenge

Virtual hen lays virtual eggs. But only with the correct password is it an Easter Egg!

create_egg

Solution by 0x90v1

First of all i just analysed the ELF executable with IDA for quite a while. Was checking all string, functions and was doing some reverse engeneering stuff to figure out, if the password is somehow stored in the file itself. But pretty quick it was obvious that it wasn't.

So next, I was looking if I could figure out, which algo was used for the decryption part. I was able to reverse this part and figured out that it is the TEA encryption. The algo was used in the d function part.

So part was solved. Now I wanted to write a little brute force program. I figured out that the encrypted data was right after enter the password string part:

```
000009F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000A00  01 00 02 00 45 6E 74 65 72 20 70 61 73 73 77 6F  ....Enter passwo
00000A10  72 64 3A 20 00 77 00 65 67 67 00 00 00 00 00 00  rd: .w.egg......
00000A20  50 CB B5 D5 64 83 4F FE E7 91 FC 42 3F B9 11 8E  PËµÕdfOþç‘üB?¹.Ž
00000A30  6C 0A 5C 98 12 C8 F4 B9 0A CD B1 EB EA 80 27 57  l.\˜.Èô¹.Í±ëê€'W
00000A40  50 92 C5 B5 AF 6C FE A2 56 5B 42 8E 7B 22 A1 F5  P'Åµ¯lþ¢V[BŽ{"¡õ
00000A50  B8 C6 90 77 03 E3 0D 65 66 55 8C 94 52 54 2F 8E  ¸Æ.w.ã.efUŒ”RT/Ž
00000A60  00 31 28 4A B2 49 12 57 12 2E 70 B3 1B 0A 23 3A  .1(J²I.W..p³..#:
00000A70  11 3D 6C D0 E3 89 B3 AA 37 ED F6 7C E7 D2 07 0A  .=lÐã‰³ª7íö|çÒ..
```

Also there are some other stuff which I could figure out during the reversing stuff:

We can set 6 bits per byte and 8 bytes. So it seems to be a high possibility that ther are no high ascii characters used for the password. So it has to be 5 bit per byte. Also it looks like that only Uppercase letters are used.

So I have to use the first block (8 bytes) from the encrypted data only, because its ECB mode and with that I should be able to get the right password already.

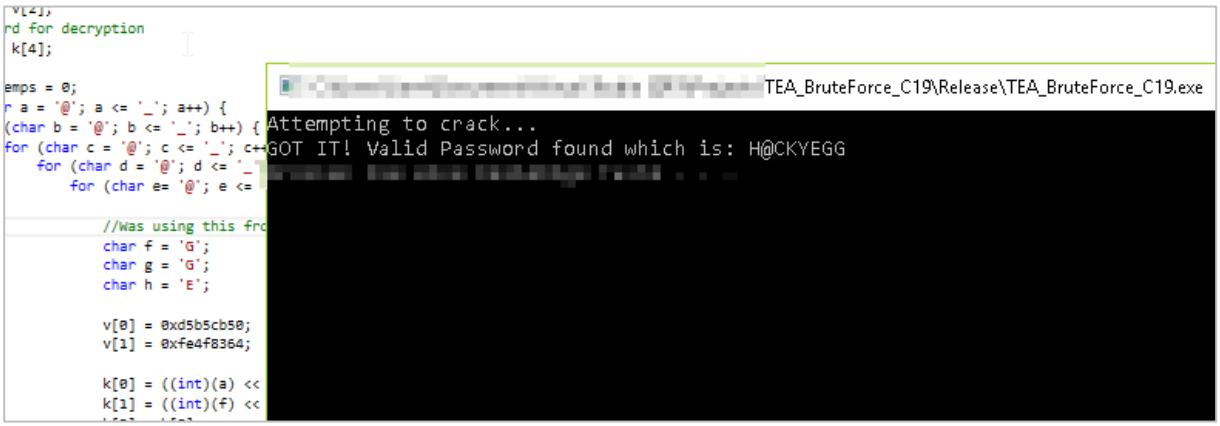
So I can crack the first block I'm able to decrypt also the rest. I also was guessing, that the decrypted data would be a PNG picture and with that I knew, for what I have to look for.

So I only have to keep in mind, that I have to check all buffers from the "wrong" direction, means if I have to look for first buffer it's not 50CBB5D5 rather d5b5cb50. If I would knew that a little bit earlier it would have saved a lot of time: P

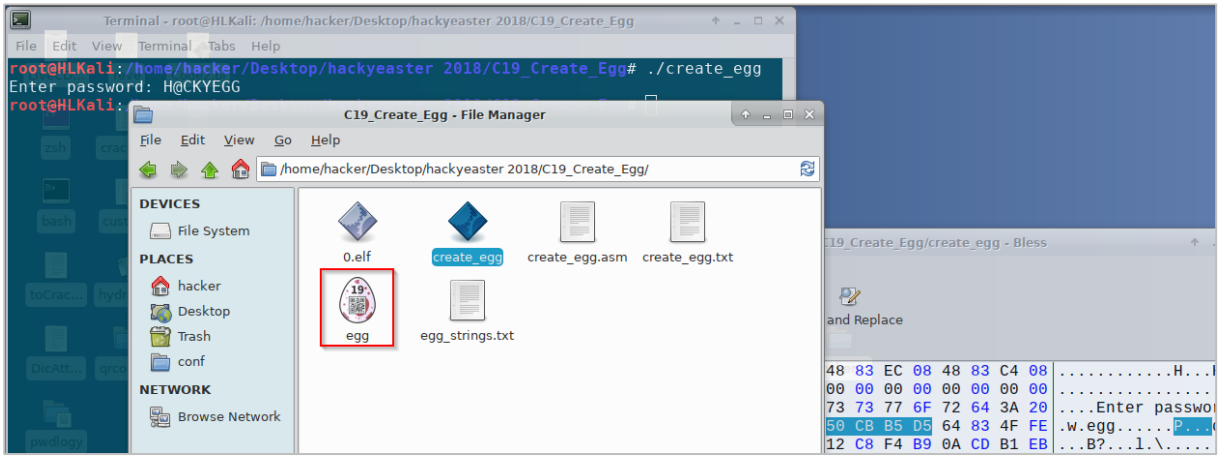
So but the final bruteforce program was looking as follow:

```
//TEA default encryption routine
//https://de.wikipedia.org/wiki/Tiny_Encryption_Algorithm
void decrypt(uint32_t* v, uint32_t* k) {
    uint32_t v0 = v[0], v1 = v[1], sum = 0xC6EF3720, i;  /* set up */
    uint32_t delta = 0x9e3779b9;                     /* a key schedule constant */
    uint32_t k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3];   /* cache key */
    for (i = 0; i<32; i++) {                         /* basic cycle start */
        v1 -= ((v0 << 4) + k2) ^ (v0 + sum) ^ ((v0 >> 5) + k3);
        v0 -= ((v1 << 4) + k0) ^ (v1 + sum) ^ ((v1 >> 5) + k1);
        sum -= delta;
    }                                                /* end cycle */
    v[0] = v0; v[1] = v1;
}

//Trying to bruteforce with the TEA decryption routine. All symbols in the ASCI Table between 0x40 and 0x95 are used for the password
//It would work also with just a single data buffer but was using two to make it more reliable.
void Crack()
{
    //encrypted data
    uint32_t v[2];
    //password for decryption
    uint32_t k[4];

    int iAttemps = 0;
    for (char a = '@'; a <= '_'; a++) {
        for (char b = '@'; b <= '_'; b++) {
            for (char c = '@'; c <= '_'; c++) {
                for (char d = '@'; d <= '_'; d++) {
                    for (char e= '@'; e <= '_'; e++) {

                        //Was using this from a hint in the challenge deccription to speed up the process a little bit :)
                        char f = 'G';
                        char g = 'G';
                        char h = 'E';

                        v[0] = 0xd5b5cb50;
                        v[1] = 0xfe4f8364;

                        k[0] = ((int)(a) << 24) | ((int)(b) << 16) | ((int)(c) << 8) | (int)(d);
                        k[1] = ((int)(f) << 24) | ((int)(g) << 16) | ((int)(h) << 8) | (int)(e);
                        k[2] = k[0];
                        k[3] = k[1];

                        decrypt(v, k);

                        if (v[0] == 0x474E5089 && v[1] == 0x0a1a0a0d) {
                            printf("GOT IT! Valid Password found which is: %c%c%c%c%c%c%c%c\n", d,c,b,a,e,h,g,f);
                            return;
                        }
                        iAttemps++;
                    }
                }
            }
        }
    }
}

int main()
{
    std::cerr << "Attempting to crack...\n";
    Crack();
    system("pause");
    return 0;
}
```

With the EGG hint from the challenge description, I was able to bruteforce the password in a matter of seconds:



And also if I entered the right password on the original file, it was spitting out the lovely PNG egg ;-)



Solution by Darkice

After some reverse engineering of the given binary we know that the Tiny Encryption Algorithm (TEA) was used to encrypt the egg. The algorithm uses a 128-bit key and should therefore be difficult to crack, but in this case a 64-bit key was duplicated to generate final key. Furthermore, the key space has been limited to characters from 0x40 to 0x5f, which is equivalent to a 40-bit key. Brute-forcing a 40-bit key only takes several hours if we use multiple CPU cores and can be done using a C program. Since the eggs for other challenges were PNG files we can assume the same for this one and use the header to check if we have found the right key.

```
#include <stdio.h>
int main(int arc, char** argv) {
  int a1,a2,a3,a4,b1,b2,b3,b4;
  a1 = 0x40;
  if (argv[1] != 0) {
    int val = atoi(argv[1]);
    if (val >= 0 && val < 32) {
      a1 += val;
      printf("a1 starts at %02x\n", a1);
    }
  }
  for (; a1<0x60; a1++) {
    for (a2=0x40; a2<0x60; a2++) {
      for (a3=0x40; a3<0x60; a3++) {
        for (a4=0x40; a4<0x60; a4++) {
          int keyA = (a1<<24) | (a2<<16) | (a3<<8) | (a4);
          printf("keyA: 0x%08x \n", keyA);
          for (b1=0x40; b1<0x60; b1++) {
            for (b2=0x40; b2<0x60; b2++) {
              for (b3=0x40; b3<0x60; b3++) {
                for (b4=0x40; b4<0x60; b4++) {
                  long keyB = (b1<<24) | (b2<<16) | (b3<<8) | (b4);
                  unsigned int rcx = 0xd5b5cb50;
                  unsigned int rdx = 0xfe4f8364;
                  int rsi = 0xc6ef3720;
                  int c = 0;
                  for (c=0;c<32;c++) {
                    rdx = (rdx-(((rcx<<0x4)+keyA)^((rcx>>0x5)+keyB)^(rcx+rsi)))&0xffffffff;
                    rcx = (rcx-(((rdx<<0x4)+keyA)^((rdx>>0x5)+keyB)^(rdx+rsi)))&0xffffffff;
                    rsi = (rsi + 0x61c88647) & 0xffffffff;
                  }
                  if (rcx == 0x474e5089 && rdx == 0x0a1a0a0d) {
                    printf("Key: 0x%08x 0x%08x \n", keyA, keyB);
                    printf("%s\n", &keyA);
                    return;
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Password: H@CKYEGG

Solution by SOKala

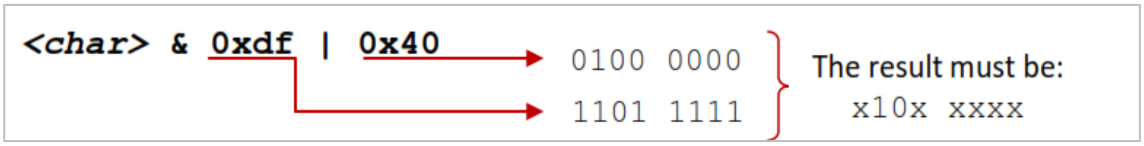
By running the create_egg program, I found that it asks about a password and generates a binary file called egg based on the entered password. By disassembling the file and analyzing it, I found that it allocates 15,624 bytes (0x3d08) of a binary data stored inside .rodata segment and keeps it for a later decryption (egg generation). Then the program asks for a password and checks if it is 8 characters long or not.

```
push    r12
push    rbp
mov     edi, 3D08h      ; size
push    rbx
sub     rsp, 30h
mov     rax, fs:28h
mov     [rsp+48h+var_20], rax
xor     eax, eax
call    _malloc
mov     edx, 3D08h      ; n
mov     esi, offset c   ; src
mov     rdi, rax        ; dest
mov     r12, rax
call    _memcpy
mov     esi, offset aEnterPassword ; "Enter password: "
mov     edi, 1
xor     eax, eax
call    ___printf_chk
mov     rdx, cs:stdin@@GLIBC_2_2_5 ; stream
lea     rsi, [rsp+48h+n] ; n
mov     rdi, rsp        ; lineptr
mov     [rsp+48h+var_48], 0
mov     [rsp+48h+n], 0
call    _getline
mov     rcx, [rsp+48h+n]
cmp     rcx, 8
jbe     loc_40080A
mov     [rsp+48h+n], 8
```

Allocating 15,624 bytes and fill it with an encrypted data from .rodata segment

Reading the password and check if its length id 8 characters long

After checking the password length, the program iterates on the first 8 characters and makes some logic operations on each character as the following:




```
loc_400730:                             ; CODE XREF: main+AE↓j
                mov     rcx, [rsp+48h+var_48]
                add     rcx, rdx
                add     rdx, 1
                movzx   eax, byte ptr [rcx]
                and     eax, 0FFFFFFDFh
                or      eax, 40h
                mov     [rcx], al
                mov     rcx, [rsp+48h+n]
                cmp     rcx, rdx
                ja      short loc_400730
```

<char>
AND
0xdf
OR
0x40

Based on the previous logical operation and with the help of ASCII character encoding map the result will be converting each character to its upper case or modifying non alphabets to binary value starting with 010xxxxx as the middle column shown in the below table:

Decimal	Binary	Symbol	Decimal	Binary	Symbol	Decimal	Binary	Symbol
032	00100000	(space)	064	01000000	@	096	01100000	`
033	00100001	!	065	01000001	A	097	01100001	a
034	00100010	"	066	01000010	B	098	01100010	b
035	00100011	#	067	01000011	C	099	01100011	c
036	00100100	$	068	01000100	D	100	01100100	d
037	00100101	%	069	01000101	E	101	01100101	e
038	00100110	&	070	01000110	F	102	01100110	f
039	00100111	'	071	01000111	G	103	01100111	g

So, each character on the 1st 8 characters will be transformed. By analyzing the remaining code, I found that it will copy the 1st modified 8 characters and append them again to a password to make it 16 characters long.

The last part of the program is the decryption of the encrypted data saved before. By analyzing the decryption function d, I found that it is using 2 constants 0x0c6ef3720 and 0x61c88647. Which means it is a decryption function of the TEA (Tiny Encryption Algorithm).

Now, we have an encrypted egg and we need to know the password that will decrypt it to a PNG image file. We need to get the key (16 bytes) that will decrypt the 1st 8 bytes of the encrypted egg to the PNG image file header (1st 8 bytes). We have only 8 characters from the pool-> @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_

I wrote a script to iterate all available 5+3 words with substituting A with @. It tries to decrypt the 1st 8 encrypted bytes and checks the result if it is a PNG file header.

```
#!/usr/bin/env python

import tea

with open('EncryptedHeader','rb') as fh:
    cipher text = fh.read()
fh.close()
with open('PNGHeader','rb') as fh:
    plain_text = fh.read()
fh.close()

fh = open('3words.txt','r')
twords = fh.readlines()
fh.close()

with open('5words.txt','r') as fh:
    for line in fh:
        for tchar in twords:
            key1 =
tchar.strip().upper().replace('A','@')+line.strip().upper().replace('A','@')
            key2 =
line.strip().upper().replace('A','@')+tchar.strip().upper().replace('A','@')
            key1 += key1
            key2 += key2
            tea1 = tea.TinyEncryptionAlgorithm()
            tea2 = tea.TinyEncryptionAlgorithm()
            if tea1.decrypt(cipher_text, key1) == plain_text:
                print 'The password is: {}'.format(key1)
                break
            if tea2.decrypt(cipher_text, key2) == plain_text:
                print 'The password is: {}'.format(key2)
                break
fh.close()
```

The password is: H@CKYEGG

Egg 20 – Artist: No Name Yet

Level: hard
Solutions: 44
Author: opasieben

Challenge

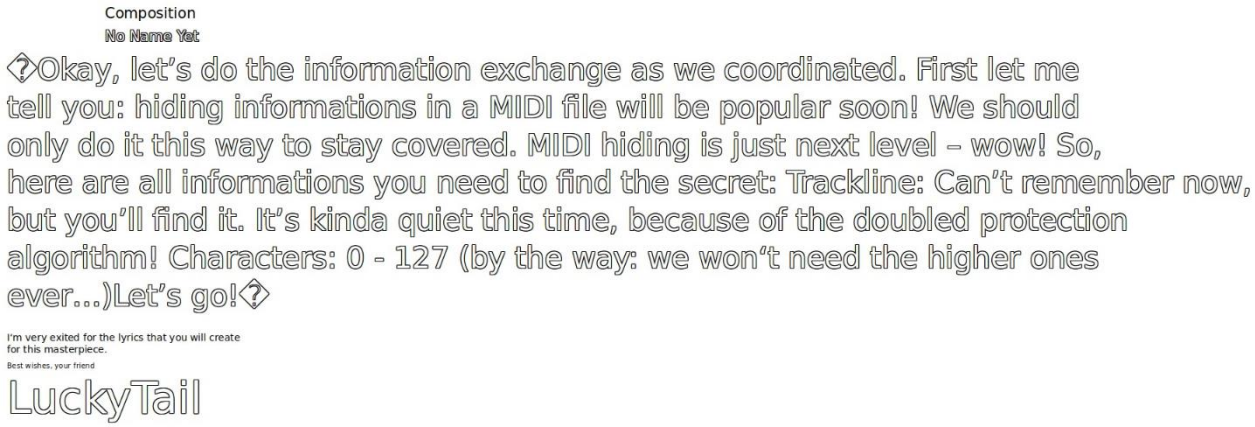
Great musical compositions are being published lately, by an unknown producer. Nobody was able yet to unveil the genius behind. It is said that he or she is placing secret messgages in his compositions.

You got hold of the original files of the latest masterpiece. Uncover the hidden message, and enter it into the Egg-o-Matic below, **case and digits only.**

artist.zip

Solution by inik

I don't know midi, installed rosegarden and found nothing useful. I also looked at the pdf, there are too many meaningless b's and #'s. So this is Stego either. After poking around I found, that the PDF has hidden text. With OpenOffice Draw I could visualize it:



Now looking into the MIDI events with rosegarden. Found out, that the velocity could be ascii values.
For this I exported all tracks into a Csound score file (because it's the easiest to read programmatically) and output the text trackwise (variable last is to eliminate doubled events, most likely an error in the midi file importer of rosegarden):

```
package egg20;
import java.io.IOException;
import java.util.List;
import util.FileUtil;
public class ExtractTextFromCSD {
  public static void main(String[] args) throws IOException {
    List<String> lines = FileUtil.readFileLines("data/20/nonameyet.csd");
    String res = "";
    String section = "";
    String last = "";
    for (String line : lines) {
      if (line.startsWith("   i")) {
        String[] cols = line.split("\t");
        if (!last.equals(cols[4])) {
          if (!section.equals(cols[0])) {
            section = cols[0];
            System.out.println("RES Val: " + res);
            res = "";
            System.out.println("RES Section:" + section);
          }
          res += (char) Integer.parseInt(cols[3]);
          last = cols[4];
        }
      }
    }
    System.out.println("RES: " + res);
  }
}
```

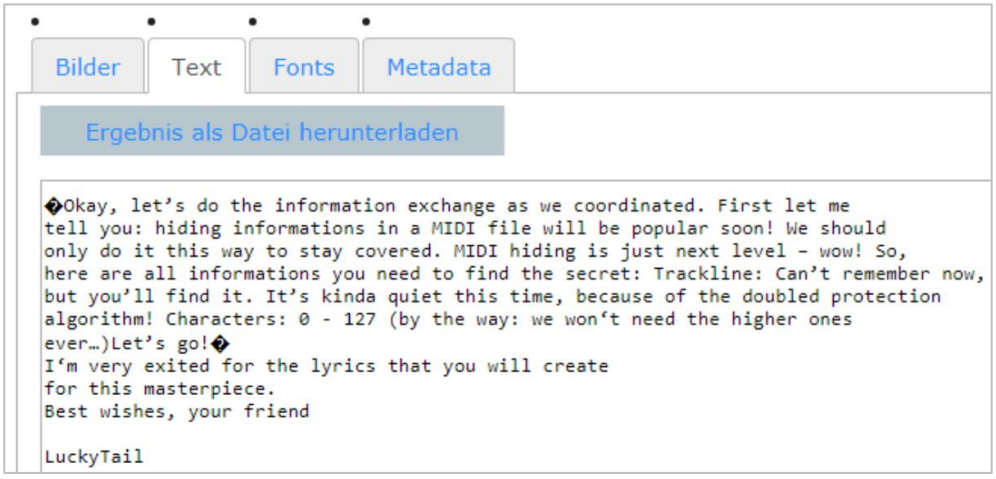
Results in

```
RES Section:   i1
RES Val: QQNNQRNSNSMEGMNNPNNKNQORPQOMKLJNLOMKI...
RES Section:   i2
RES Val: `d_UWUW_S_SWSZZVZV^QWWQWW`adN[Y]cZ]cZZ_Za`]^V[NZ
RES Section:   i3
RES Val: -.-. --- -- .--. --- ... . -.. -.. -.-- -.. .--- ... .--. ----- ----- -
. -.-
RES Section:   i4
RES Val: ((-2:><==>C@C;;;;;;>;<;<@><<><6<=:>@>@>@9;@<<Eaa__]]\...
RES Section:   i5
RES Val: QRRRPTQPOO65O65OO65O;4O;4ON?<N?<N@;KN4949NN@=N@=NN@=N=9>=9>=..
RES Section:   i6
RES Val: u>YZfttq}~}
RES Section:   i7
RES Val: .8@<DBDB;$$74>S@DN;BF6F>9<>FBKDB33/7KQ\QNSF9@>>29<I89B;7DD8...
RES Section:   i8
RES Val: falfeecd```aa`bad`fcecig^``ccabb`fh`a__`_a`cbij`cabdfdddehgdeeeefde...
RES Section:   i10
RES: 2Q(Y_;!BDPD@M:<Oj'DIRFVVbBF\9UF+Zd*ZZZDQH@QL=QTZJc]hZrmYbZ`i...
```

That's all garbage, except for track i3, which is morsecode. Decoding it online with https://gc.de/gc/morse/ I got the password COMPOSEDBYDJSP00NY.

Solution by HaRdLoCk

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



ok - so we know it's 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 what's 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.

```
1  import re
2
3  x=""
4
5  with open("C:\\Users\\administrator\\Desktop\\hacky2018\\3.txt") as f:
6      for line in f:
7          s = re.compile("vol:\s\d{1,3}")
8          m = s.search(line)
9          if m:
10             x += chr(int(m.group(0).replace("vol: ", "")))
11 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!

Solution by LlinksRechts

First, I extracted all text from the PDF using **pdftotext**. This gave me the following hidden hint:

```
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!
```

Since the character set is apparently 0-127, it became obvious pretty fast that the data is hidden in the velocity values of the MIDI events. Therefore, I converted the MIDI file to text using a python tool from https://github.com/vishnubob/python-midi and extracted a list of velocities using

```
cat nonameyet.dump|grep Off|grep -o '\d*]\)'|tr -d '])' > offVelocities
```

Then, I used python to convert them to characters:

```
with open('onVelocities', 'r') as f:
    c=f.read()

d=[int(x) for x in c.split('\n')[:-1]]
for i in d:
    print(chr(i), end="")
```

The resulting text contained some morse code,

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

which when decoded yields **COMPOSEDBYDJSP00NY**. This is the password for the Egg-o-Matic.

Egg 21 – Hot Dog

Level: hard
Solutions: 61
Author: Kiwi.wolf

Challenge

or: how to solve this darn crypto challenge to get your sleep back.

Enter the flag found, into the Egg-o-Matic below, **without brackets.**

hotdog.zip

Solution by Kiwi.wolf

Stage 1: Extract the ciphertext

When using the file command you'll see that it's actually a tiff. Tiff images can be layered.
Since layered tiff normally can't be shown by gimp, you can use the ImageMagick Library.

```
$ Convert flag.jpg'[1]' layer.tiff
```

To extract the ciphertext from the qr code you can use zbarimg and base64 -d

Stage 2: Extract the public key

This stage is easier. You can use binwalk to extract the RSA public key.

```
$ exiftool flag.jpg
```

You'll see a tag with *Don't forget to delete this*

Stage 3: Crack the RSA

Now you'll need some basic crypto knowledge and pay close attention to the challenge name and the image. A hotdog usually has a bun. You already found both sides of it with the ciphertext and the public key…but there's one key ingredient missing. You're right, it's a "Wiener" sausage. Wiener is also the name of a typical RSA attack based on the Wiener's Theorem. An attack can find p + q efficiently if d <1/3 N 1/4. You can either use the Rsactftool or write your own script using continued fractions and the Wiener's Theorem.

There are several great tools for working with rsa and the pem tool.

The last step is to use the openssl library to decode the file:

```
echo
"Arf3ThIY8VQg2GUd249wzDYi7CXqTST+9g4Q7bbT2eF+mD2KB+6oi3rVSY/eZ6/onNBNYPo2BPqIVEb
L35G62pIHvabGcrYosGCpYhi
z6EYnamnNPrHdzmEOs8lCRw1c2Pe8kl41FH0ud7tBn6qD/stnZfGkcbeIrjaSiIYSveHS" | base64
-d |  openssl rsautl -decrypt inkey
/home/hacker/rsatool/privkey.pem
```

Great job haxxor, here's your flag: {b3w4r3_0f_c0n71nu3d_fr4c710n5}

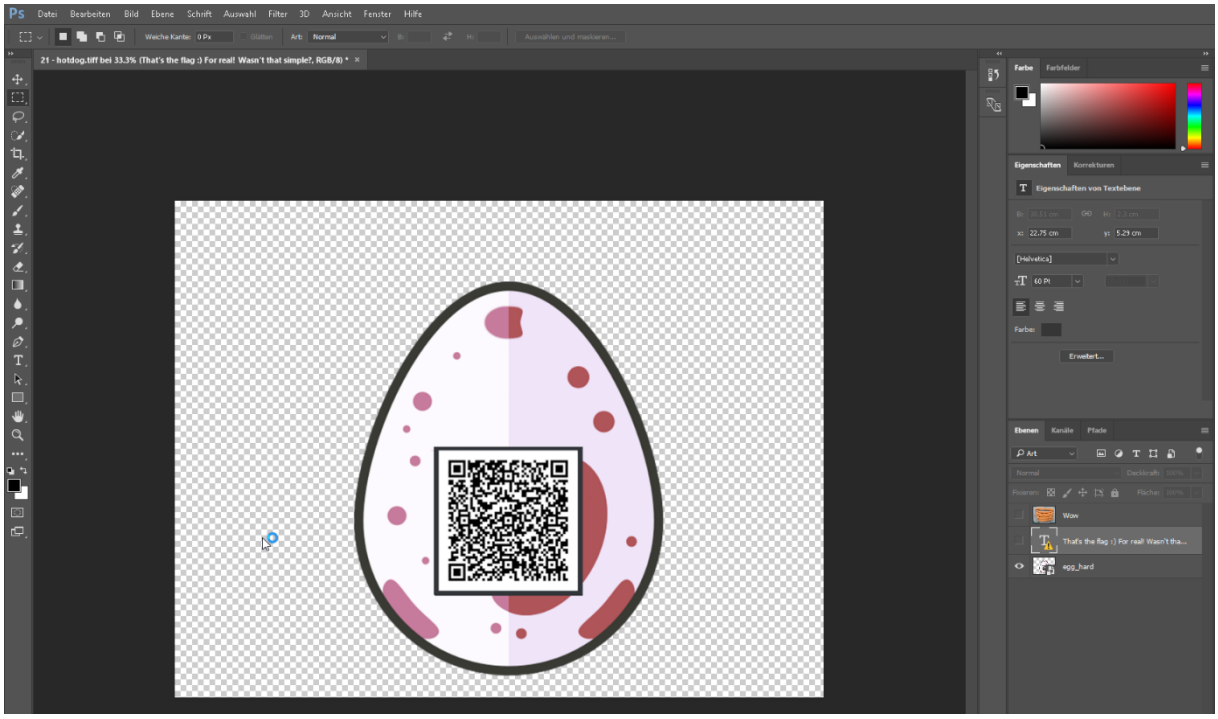
Solution by 0x90v1

First step for me it was, that I had a few on the picture with a hex viewer. There I saw that there is a Public Key inside and I saw as well some hint about Photoshop.

So the extracted public-key was:

```
-----BEGIN PUBLIC KEY-----
MIIBIDANBgkqhkiG9w0BAQEFAAOCAQ0AMIIBCAKBgQTMleqB9nvRKhTnR4/2BDDU
g5hkjbRQygvrZWDATbC9rXxCAqaegim2XUlD8yVxYkyzJZxmAYba7qLTe3bctocM
L7GXdMf3kQiVLPigN2auEiPFreWZvZ/b4FzcvOhh+SprypAkYn9SapTyGzLdpYdD
TyoWFRT7QgEhIsDGcncsXQKBgQCVbdUZa5uQ7O9bgu2WPvUwwvuI+ZK5gOZCF299
1QRa/rdDHKyYiUxxZXjemxGICxvoC698wVvmVqzG/sCT+iLArIh4OmSHgyd1yjcA
CWmsffHYLvsl3tnN9Jiu5qzN6aGthHjK/424NK0RkfjUdmnQydYN/MqfS7c+AkfJ
QWV/9w==
-----END PUBLIC KEY-----
```

After that, I opened the picture in Photoshop and saw the different layers. One of them seems to be the Egg, so quickly QR scan showed me it was not ;-)

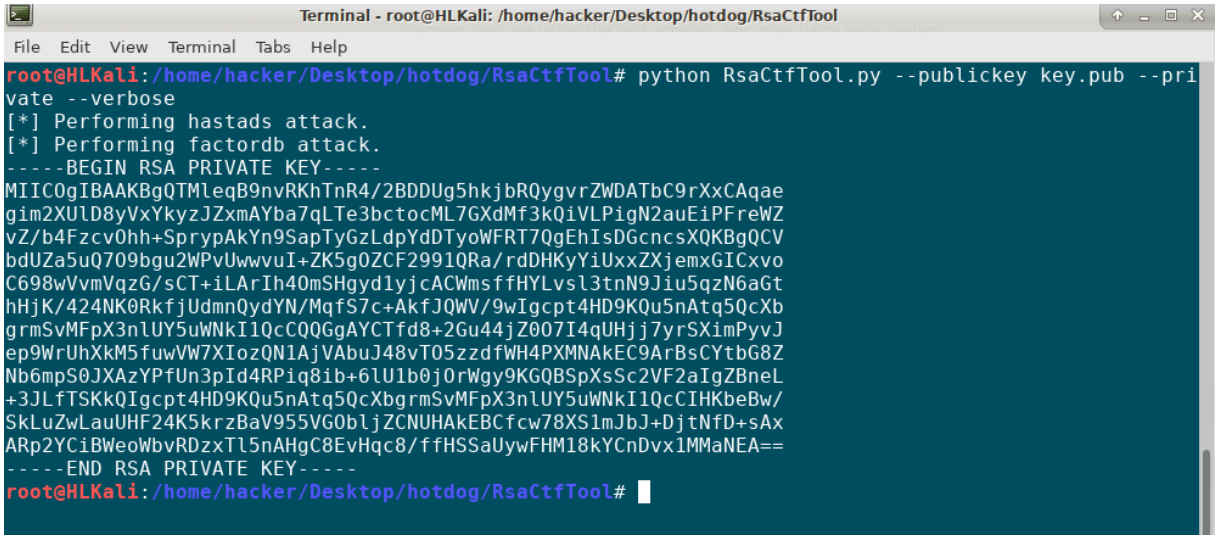


It seems to be an encrypted message and the type of if remembered me about like RSA.

So I went one step ahead and used google one more time to figure out if it's possible to get the private key out of the public key. Then I found the following python script:

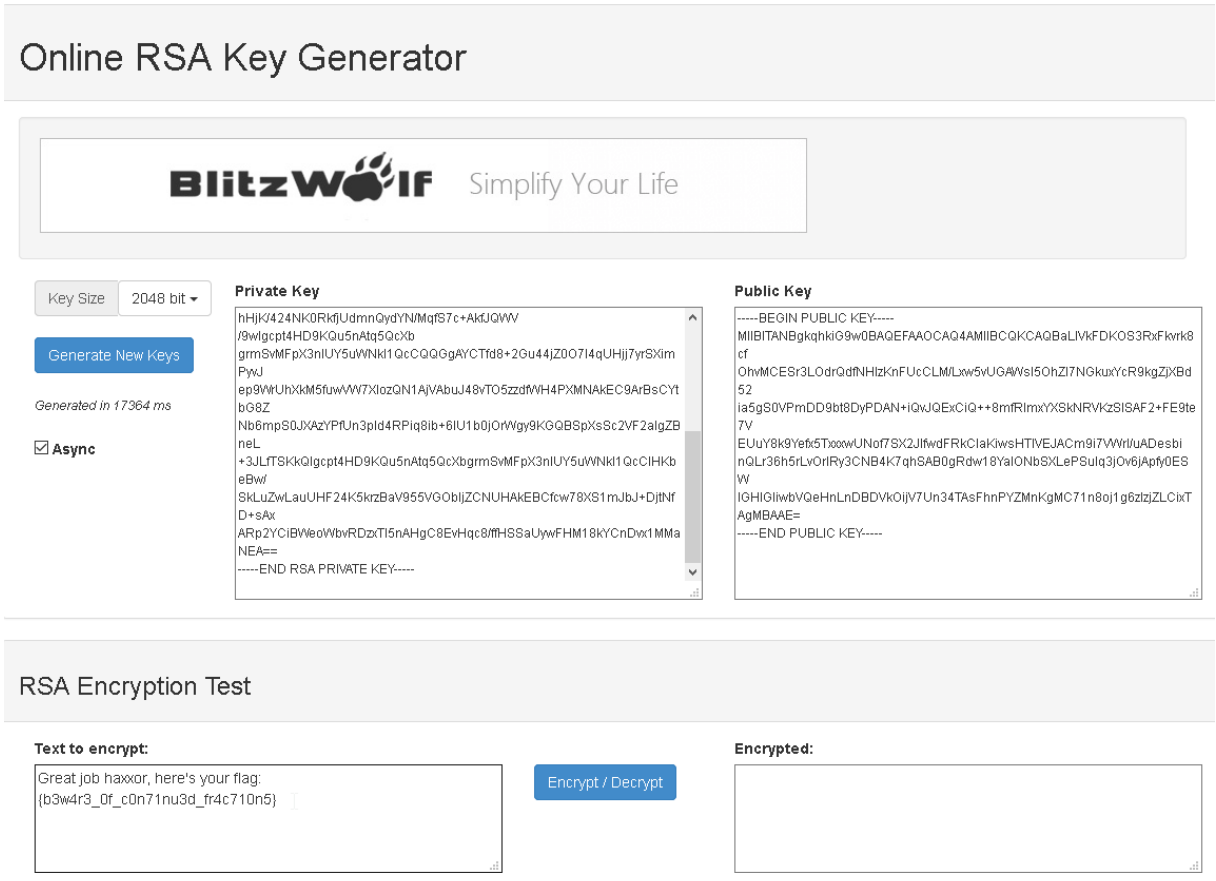
https://github.com/Ganapati/RsaCtfTool

So I gave it a shot and it really was working to get me a private key out of the public key I have found in the picture.



So now the only thing I had todo was, decrypting the message. I found a Webpage, which did the stuff for me, called http://travistidwell.com/jsencrypt/demo/

I was able to insert the private key and it seems that this was all that we needed. Of course I had to enter the encrypted message which I got from the QR code in the Photoshop.



So I finally got the password ☺

b3w4r3_0f_c0n71nu3d_fr4c710n5

Solution by SOKala

By using binwalk utility to analyze the hotdog.jpg file contents:

```
# binwalk hotdog.jpg
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             TIFF image data, little-endian offset of first image
26036         0x65B4          Copyright string: "Copyright (c) 1998 Hewlett-Packard
14752442      0xE11ABA        StuffIt Deluxe Segment (data): fVghXfgWigXjh\he^cc[_
15511511      0xECAFD7        StuffIt Deluxe Segment (data): fTecTecTecTdbShbVhdX^_
26795840      0x198DF40       mcrypt 2.2 encrypted data, algorithm: DES, mode: CBC,
27180761      0x19EBED9       TROC filesystem, 875443257 file entries
28956475      0x1B9D73B       LANCOM OEM file
35072724      0x2172AD4       PNG image, 480 x 480, 8-bit/color RGBA, non-interlace
35073815      0x2172F17       Zlib compressed data, default compression
```

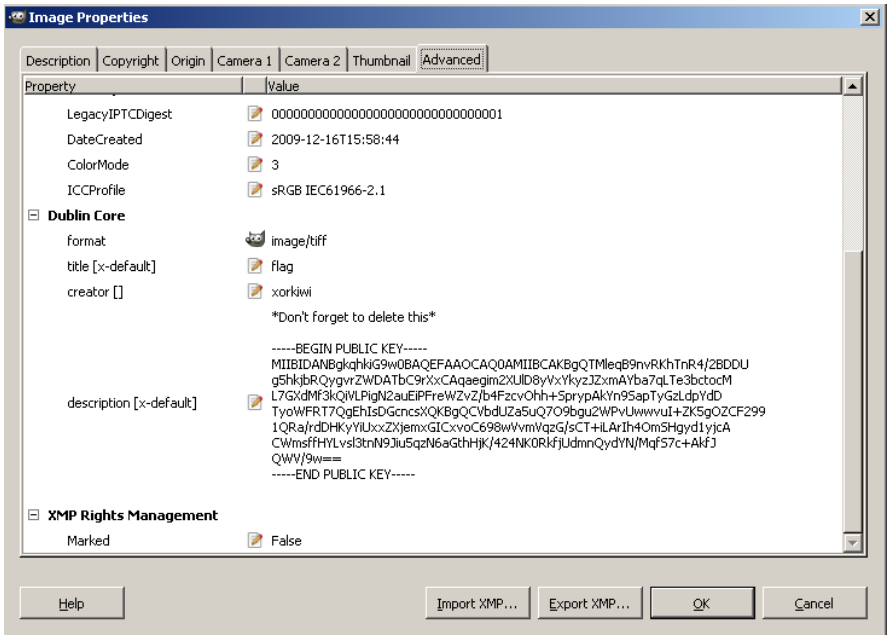
Looks like we have a PNG image at offset 0x2172AD4. By extracting the PNG image

```
# binwalk -D "png image:png" hotdog.jpg
```

We get an egg. By trying to get the QR data..

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

Looks like a base64 encrypted data. By opening the hotdog.jpg file using GIMP and getting the Image properties, I found a public key!!



I saved its contents as key.pub. Then I saved the base64 decoded contents to a Cipher.enc file..

```
# echo
"Arf3ThIY8VQg2GUd249wzDYi7CXqTST+9g4Q7bbT2eF+mD2KB+6oi3rVSY/eZ6/onNB
NYPo2BPqIVEbL35G62pIHvabGcrYosGCpYhiz6EYnamnNPrHdzmEOs8lCRw1c2Pe8kl4
1FH0ud7tBn6qD/stnZfGkcbeIrjaSiIYSveHS" | base64 -d > ../Cipher.enc
```

The public key has a weak RSA encryption, then tryig to decrypt it using RsaCtfTool.py..

```
#./RsaCtfTool.py --publickey key.pub --uncipher Cipher.enc
Great job haxxor, here's your flag: {b3w4r3_0f_c0n71nu3d_fr4c710n5}
```

Bingo!! I decrypted it..

The password is: b3w4r3_0f_c0n71nu3d_fr4c710n5

Egg 22 – Block Jane

Level: hard
Solutions: 56
Author: 3553x

Challenge

You intercepted an encryped message by Jane. Can you decrypt it?

You know that AES was used, and that the following service is receiving such encrypted messages:

```
whale.hacking-lab.com 5555
```

Find the password and enter it in the Egg-o-Matic below!

secret.enc

Solution by Floxy

AES decryption with a given message and a service which returns “error” and “ok” leads me to Padding-Oracle Attack. The most available scripts that are available only support “HTTP”-Attacks so I decided to use https://github.com/mpgn/Padding-oracle-attack and adjust it to use sockets.

Somebody already coded a part https://gist.github.com/mpgn/fce3c3f2aaa2eeb8fac5 so I adjusted only a few things a started the script with following command.

```
python exploit_custom.py -c
E343F42604CA58A731ADBF10B376EE33AA944926CDF954400D86EE4F6E35774EC510FE5767BABA99
A3ED28FA26DC99B6C1DADD087E4CEE27E45507005276C10FD9C15F27D3481A92F34DD46477F7BE3
C -l 16 --host whale.hacking-lab.com --port 5555 -v --error "error"
```

After a long time of running it reveals the secret message:

```
[+] Test [Byte 094/256 - Block 1 ]: AD2097417BA82C9748CE9520CC149F40[+] RESPONSE  error
[+] Test [Byte 095/256 - Block 1 ]: AC2097417BA82C9748CE9520CC149F40[+] RESPONSE  error
[+] Test [Byte 096/256 - Block 1 ]: 932097417BA82C9748CE9520CC149F40[+] RESPONSE  ok
[+] Test [Byte 097/256 - Block 1 ]: 922097417BA82C9748CE9520CC149F40
[+] Block M_Byte : 617373776f72642069733a206f726163
[+] Block C_{i-1}: E343F42604CA58A731ADBF10B376EE33
[+] Block Padding: 10101010101010101010101010101010

[+] Found 16 bytes : 617373776f72642069733a206f726163


[+] Decrypted value (HEX): 617373776F72642069733A206F7261636C33696E3364656C7068690A0A53656520796F7520736F6F6E210A0A4A616
E650A0A0E0E0E0E0E0E0E0E0E0E0E0E0E0E0E
[+] Decrypted value (ASCII): assword is: oracl3in3delphi

See you soon!

Jane
```

Solution by inik

This one is a padding oracle, no doubt. So I took my code from HL 7156 and modified it:

```
package egg22;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.InputStreamReader;
import java.net.Socket;
import util.StringUtil;
public class PaddingOracle {
 private static String[] t = {
  "e343f42604ca58a731adbf10b376ee33",
  "aa944926cdf954400d86ee4f6e35774e",
  "c510fe5767baba99a3ed28fa26dc99b6",
  "c1dadd087e4cee27e45507005276c10f",
  "d9c15f27d3481a92f34dd46477f7be3c"
 };
 public static void main(String[] args) throws Exception {
   if (testForPaddingError(StringUtil.hexStringToByteArray(
      "e343f42604ca58a731adbf10b376ee33aa944926cdf954400d86ee4f6e35774ec510fe5767baba99a
      3e d28fa26dc99b6c1dadd087e4cee27e45507005276c10fd9c15f27d3481a92f34dd46477f7be3c "))
     ) {
      System.out.println("TEST1 *NOT* OK");
      System.out.println("TEST1 OK");
     }
     if (testForPaddingError(

       StringUtil.hexStringToByteArray("e343f42604ca58a731adbf10b376ee33aa944926cdf954400
        d86ee4f6e35774e "))) {
        System.out.println("TEST2 *NOT* OK");
       } else {
        System.out.println("TEST2 OK");
       }
       for (int i = 1; i < (5 - 1); i++) {
        PaddingOracleAttacker oo = new
        PaddingOracleAttacker(StringUtil.hexStringToByteArray(t[i]),
         StringUtil.hexStringToByteArray(t[i + 1]));
        while (!oo.isDone()) {
         if (testForPaddingError(oo.getTestTicket())) {
          oo.setFailure();
         } else {
          oo.setSuccess();
          System.out.println("\nRES: " + oo.getResult() + " / " +
           StringUtil.byteToHexString(oo.getResultAsByteArray()));
         }
        }
        System.out.println("RES: " + oo.getResult());
        System.out.println("RES: " +
         StringUtil.byteToHexString(oo.getResultAsByteArray()));
       }
      }
      private static boolean testForPaddingError(byte[] send) throws Exception {
       Socket echoSocket = new Socket("whale.hacking-lab.com", 5555);
       DataOutputStream out = new
       DataOutputStream(echoSocket.getOutputStream());
       BufferedReader in = new BufferedReader(new InputStreamReader(echoSocket.getInputStream()));
       for (byte b: send) {
        out.writeByte(b);
       }
       // out.writeByte(0xa);
       String res = in .readLine();
       String res = in .readLine();
       echoSocket.close();
       if (res.contains("ok")) {
        System.out.print("+");
        return false;
       }
       System.out.print("-");
       return true;
      }
     }

```

This result in the following text (1 block not decodable):

```
password is: oracoracl3in3delphi
```

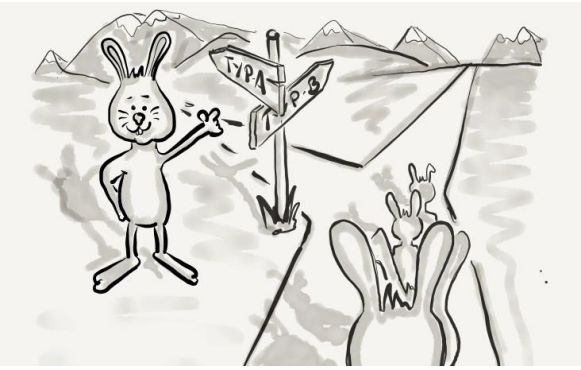
Solution by TheVamp

This webservice has a classical Oracle Padding Problem. For solving I used the paddingoracle framework (https://github.com/mwielgoszewski/python-paddingoracle):

```
from pwn import *
import socket
from Crypto.Cipher import AES
from paddingoracle import BadPaddingException, PaddingOracle

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

# extracted from pcap
with open("secret.enc") as f:
    encdata = f.read()

log.info("%d blocks of AES", len(encdata) // AES.block_size)
# split the blocks
bs = AES.block_size
iv = encdata[:bs]
blocks = [encdata[bs:2 * bs],
          encdata[2 * bs:3 * bs],
          encdata[3 * bs:4*bs],
          encdata[4 * bs:5*bs]]

encdata_s = ""
for b in blocks:
    for c in b:
        encdata_s += chr(ord(c))

class PadBuster(PaddingOracle):
    def __init__(self, **kwargs):
        super(PadBuster, self).__init__(**kwargs)

    def oracle(self, data, **kwargs):
        log.info(hexdump(data))
        #rem = remote(HOST, PORT)
        rem = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        rem.connect((HOST, PORT))
        rem.sendall("".join(chr(x) for x in data))
        line = rem.recv(1024)
        #self.history.append(line)
        log.debug(line)
        #print line
        if "error" in line:
            raise BadPaddingException()

        #else:
        #    raise Exception("NOPE")
        rem.close()
        del rem


padbuster = PadBuster()
data = padbuster.decrypt(encdata_s, block_size=AES.block_size, iv=iv)
log.info(data)
```

```
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\x0e\x0e
```

Egg 23 – Rapbid Learning



Level: hard
Solutions: 91
Author: opasieben

Challenge

The SERTC (*super exciting rabbit travel company*) wants to use the latest technology.

They currently employ an experienced guide, which is classifying the visitors into *Goodtails* and *Luckyspoons*. For the trained eye of the guide, this is easy. But how do you get a machine to do it (which would be cheaper)?

Go and help them implement a learning algorithm! Their platform is available here:

```
http://whale.hacking-lab.com:2222
```

Since quality is the main concern of SERTC, they expect an accuracy of at least 99%.

Solution by daubsi

This was a very nice challenge, especially because I had just finished attending Cousera's free Machine Learning lecture by Andrew Ng… 😊
In this challenge we have to train a classifier, apply it to a test data set, and send back our results. If our results are accurate to at least 99%, we are given a cookie which lets us access the reward page which probably gives us the flag/egg.

Using the scikit-learn ML libraries of Python the whole challenge can be solved in a couple of dozen LoC :-D

```
from sklearn.datasets import load_svmlight_file
from sklearn import svm
import requests
import re
import base64
import os

train_url = "http://whale.hacking-lab.com:2222/train"
assign_url = "http://whale.hacking-lab.com:2222/gate"
submit_url = "http://whale.hacking-lab.com:2222/predict"
reward_url = "http://whale.hacking-lab.com:2222/reward"
traindata_file = "traindata"

colors = { 'red':0, 'black':1, 'brown': 2, 'grey': 3, 'purple': 4, 'green':5,
'blue': 6, 'white': 7}
train = False

if train:
    print("Training")
    td = open(traindata_file,"w")
    for idx in range(1,1000):
        j = requests.get(train_url).json()

        td.write("{0} 1:{1} 2:{2} 3:{3} 4:{4} 5:{5} 6:{6} 7:{7}\n"
                 .format(1 if j['g00d'] is True else 0, j['t41l'], j['w31ght'],
j['ag3'], j['l3ngth'],
                         0 if j['g3nd3r'] == 'male' else 1,
colors[j['c0l0r']],j['sp00n']))
    td.close()

print("Loading train data")

X_train, y_train = load_svmlight_file(traindata_file)

print("Fitting model")
clf = svm.SVC(gamma=0.001, C=100.)
clf.fit(X_train, y_train)

print("Loading challenges")
fd = requests.get(assign_url)
cookie = fd.cookies['session_id']
j = fd.json()

# j['data'][0] = ['Harold', 'male', 4, 'grey', 4, 41, 8, 9]
# j['attributes'] = ['bjRtMw==', 'ZzNuZDNy', 'NGcz', 'YzBsMHI=', 'dzMxZ2h0',
'bDNuZ3Ro', 'c3AwMG4=', 'dDQxbA==']
# name, gender, age, color, weight, length, spoon, tail

print("Predicting")
answers = []
for i in range(0,len(j['data'])):
    d = j['data'][i]
    #sample = "1:{0} 2:{1} 3:{2} 4:{3} 5:{4} 6:{5} 7:{6}\n".format(d[7], d[4],
d[2], d[5], 0 if d[1]=='male' else 1, colors[d[3]], d[6])
    sample = [[d[7], d[4], d[2], d[5], 0 if d[1] == 'male' else 1, colors[d[3]],
d[6]]]
    prediction = clf.predict(sample)
    answers += [prediction[0]]

# Send to submit
print("Post back")
mycookies= dict(session_id=cookie)

# Burp
os.environ['http_proxy']='localhost:8080'

r = requests.post(url=submit_url,json=answers, cookies=mycookies)
print(r.text)

r = requests.get(url=reward_url, cookies=mycookies)
print("Regexing egg")
pattern = re.compile("base64,(.*)\">")
m = pattern.search(r.text)

png = open("egg23.png","wb")
png.write(base64.b64decode(m[1]))
png.close()
print("Egg saved to egg23.png")
```

Solution by Lukasz_D

For this classification challenge I used a machine learning method called logistic regression. First, the classifier has to learn how to assign visitors in one of the two available categories. For the learning process it needs to get many examples of correctly assigned visitors in order to figure out what algorithm decides on the assignments. Then, the classifier will apply the learned algorithm to predict classification of test visitors.

The entire process of getting training data, learning and predicting is implemented in the following script:

```
1.  from sklearn.linear_model import LogisticRegression
2.  import pandas as pd
3.  import json
4.  import requests
5.  import base64
6.
7.  TRAIN_SIZE = 2000
8.
9.  def prepare_data(data):
10.     data.drop(columns=['n4m3'], inplace=True) # "name" does not seem to play any ro
    le in classification
11.     data = pd.get_dummies(data, columns =['g3nd3r', 'c0l0r']) # categorical variabl
    es need to be processed differently
12.     columns = list(data.columns)
13.     columns.sort() # ensure the order of columns is always the same
14.     return data[columns]
15.
16. # Train the classifier
17. data_dict = {}
18. for i in range(TRAIN_SIZE):
19.     resp = requests.get('http://whale.hacking-lab.com:2222/train').text
20.     for k,v in json.loads(resp).iteritems():
21.         if k in data_dict:
22.             data_dict[k].append(v)
23.         else:
24.             data_dict[k] = [v]
25.
26. dataframe = pd.DataFrame(data_dict)
27. data = prepare_data(dataframe)
28.
29. x = data.iloc[:, data.columns != 'g00d'] # the data used for classification cannot
    contain the objective variable
30. y = data.iloc[:, data.columns == 'g00d'] # objective variable
31.
32. logisticRegr = LogisticRegression()
33. logisticRegr.fit(x, y) # learn the classification based on training data
34.
35.
36. # Predict the classification
37. r = requests.get('http://whale.hacking-lab.com:2222/gate')
38. cookie = r.headers['Set-
    Cookie'].split(';')[0] # get the cookie, requests.Session() does not work because t
    he cookie has "secure" attribute and requests are sent via HTTP
39. challenge = json.loads(r.text)
40. columns = [base64.b64decode(x) for x in challenge['attributes']]
41.
42. dataframe = pd.DataFrame(challenge['data'], columns=columns)
43. data = prepare_data(dataframe)
44.
45. prediction = logisticRegr.predict(data)
46. answer = [int(prediction[x]) for x in prediction]
47.
48. print requests.post('http://whale.hacking-
    lab.com:2222/predict', json=answer, headers={'Cookie': cookie).text
49. print requests.get('http://whale.hacking-
    lab.com:2222/reward', headers={'Cookie': cookie).text
```

After executing the script, the following output was obtained:

```
SCORE: MTAwLjAl - lolnice! - I'll tell my guys to set up your reward for this
shift at /reward, don't forget to bring your cookie!
```

and the base-64 encoded egg:

```
<h2>Reward</h2>
<hr>
<div>
<img
src="[
CUT]
```

Note that the script does not always produce the correct classification. Sometimes, the test data seem to be derived from a different distribution than the training data. Nevertheless, invoking the script at most a few times should give the egg.

Solution by mezuru

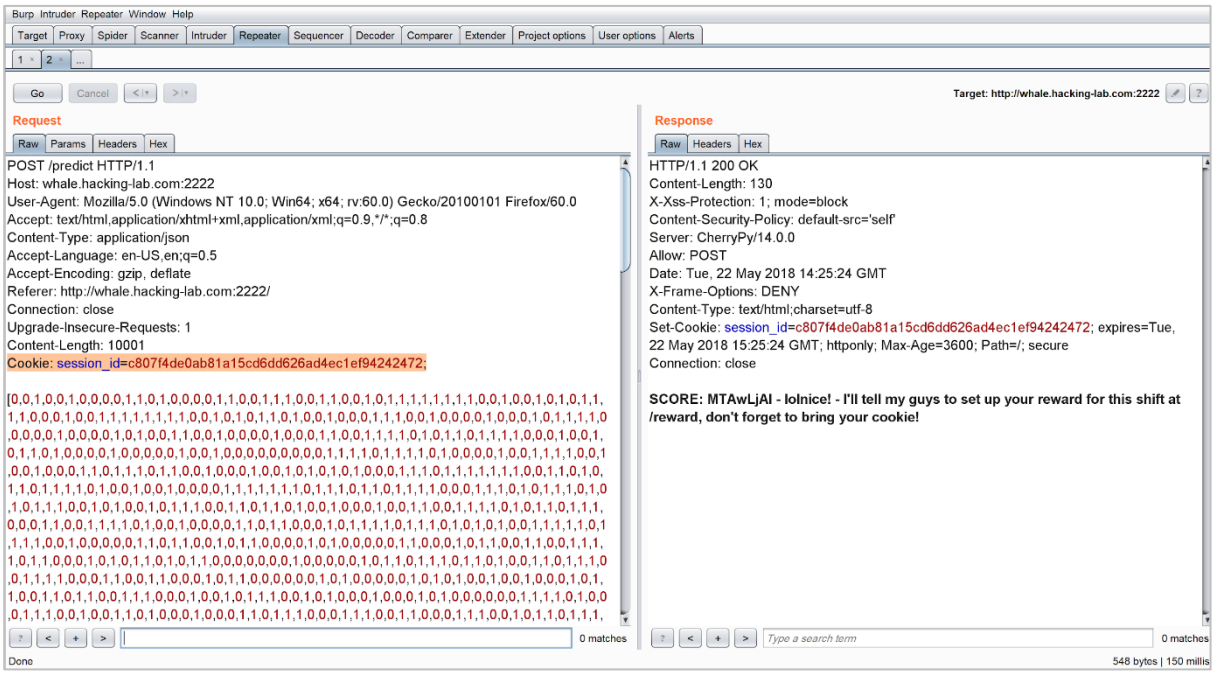
I liked this challenge :) So the idea is to sort rabbits to goodtails or luckyspoons. So first thing I did was figure out what criteria is used to classify the rabbits. I generated a bunch of requests towards this page http://whale.hacking-lab.com:2222/train and analyzed the data. It turns out the header "g00d" identifies whether the rabbit is a goodtail or not. Upon further investigation, any rabbit with weight or 2 or spoon of 13 is not considered a goodtail.

Next we will need to access the assignment page which has a list of objects in JSON format. So the idea is to go through the whole list and sort out goodtails and luckyspoon using 1 and 0 respectively. The output should be in this format [1,0,0,1,..]. Hence I wrote this script to sort them quickly:

new 2 | new 1 | json1.py

```
1  import urllib, json
2  url="http://whale.hacking-lab.com:2222/gate"
3  response=urllib.urlopen(url)
4  data=json.loads(response.read())
5  x='['
6  i=0
7  while i <= 4999:
8      if data["data"][i][4]==2 or data["data"][i][7]==13:
9          x=x+'0,'
10     else:
11         x=x+'1,'
12     i=i+1
13 x=x+']'
14 print x
```

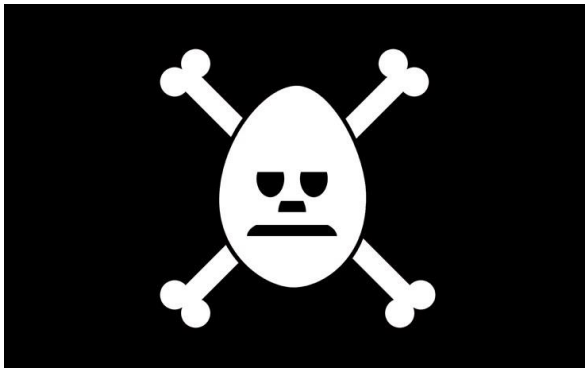
Using the output of the script and burp proxy I constructed a request towards http://whale.hacking-lab.com:2222/predict in the following way:



I changed the http request from GET to POST, added content type: application/json, added cookie of the session which I grabbed through packet sniffing, and of course used the results of the above script in the request.

Next to get the reward, it's important to use the same cookie so I submitted a simple request toward /reward page with the same cookie and got the egg as a result.

Egg 24 – ELF



Level: hard
Solutions: 100
Author: inik

Challenge

The Egg Liberation Front (ELF) already marked this binary. Go ahead and free the egg!

Once the right PIN is provided, the binary spills out the Easter egg QR code.

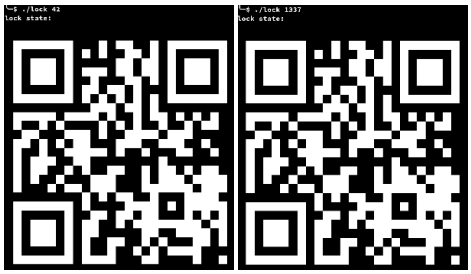
lock

Solution by darkstar

If we call the program without parameters it informs us that it would like to be started with a pin.

```
$ ./lock
./lock <pin to unlock>
```

The same QR-Code (—— locked ——) is displayed for different pins.



We can therefore assume that the code we are looking for is only displayed with the correct PIN. Now we could try to find the right pin by reversing, but a simple bruteforce attack might be enough.

```
import pwn
import hashlib

def checkPin(pin):
    x = pwn.process(['./lock', str(pin)])
    return hashlib.md5(x.readall()).hexdigest()

i = 0
while i < 2**32:
    if checkPin(i) != 'e9db32f73c358af3b5e03a88a465dc3e':
        print i
        exit()
    i+=1
```

Ok, that might have taken longer than the reversing would have taken, but in the meantime other challenges could be solved.

Solution

Pin: 1098505442

Solution by Floxy

After loading the file in gdb and stepping through the code, I first land on the “checkpin” function, but this leads me nowhere. So I decided to step through the code from the beginning.

My test-PIN was “1111” in HEX “457”.

I stepped through the code and found an interesting “cmp eax, ebx” statement, where EAX was my entered pin.

```
root@ubuntu: ~/HackingLab
[----------------------------------registers-----------------------------------]
EAX: 0x457
EBX: 0x4179dce2
ECX: 0x19
EDX: 0xf7faa870 --> 0x0
ESI: 0x5655703f --> 0x3f8b2fe
EDI: 0xf7fa9000 --> 0x1b1db0
EBP: 0xffffd2e8 --> 0xffffd2f4 --> 0xffffd543 ("1111")
ESP: 0xffffd2e8 --> 0xffffd2f4 --> 0xffffd543 ("1111")
EIP: 0x56555775 (<qr_next_line+65>:     cmp    eax,ebx)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   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
   0x56555781 <qr_next_line+77>:        xor    ebx,ebx
[-------------------------------------stack-------------------------------------]
0000| 0xffffd2e8 --> 0xffffd2f4 --> 0xffffd543 ("1111")
0004| 0xffffd2ec ("vVUVgVUVC\325\377\377")
0008| 0xffffd2f0 ("gVUVC\325\377\377")
0012| 0xffffd2f4 --> 0xffffd543 ("1111")
0016| 0xffffd2f8 --> 0x0
0020| 0xffffd2fc --> 0xf7e0f637 (<__libc_start_main+247>:       add    esp,0x10)
0024| 0xffffd300 --> 0x2
0028| 0xffffd304 --> 0xffffd394 --> 0xffffd529 ("/home/flo/HackingLab/lock")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x56555775 in qr_next_line ()
gdb-peda$
```

So I tried to convert EBX (0x4179dce2) to decimal “1098505442” and entered this as PIN. Therefore the ELF-binary revealed the egg.

Solution by Meliver

Disassemble

```
.text:0000075B                mov     esi, offset qr1
.text:00000760                xor     ebx, ebx
.text:00000762                xor     ecx, ecx
.text:00000764
.text:00000764 loc_764:                               ; CODE XREF: .text:0000076B↓j
.text:00000764                add     ebx, [esi+ecx*4]
.text:00000767                inc     ecx
.text:00000768                cmp     ecx, 19h
.text:0000076B                jnz     short loc_764
.text:0000076D                mov     eax, offset qr1
.text:00000772                mov     eax, [eax-4]
.text:00000775                cmp     eax, ebx
.text:00000777                jnz     short loc_793
.text:00000779                mov     esi, offset qr1
.text:0000077E                add     esi, 64h ; 'd'
.text:00000781                xor     ebx, ebx
.text:00000783                xor     ecx, ecx
.text:00000785
.text:00000785 loc_785:                               ; CODE XREF: .text:00000791↓j
.text:00000785                mov     ebx, [esi+ecx*4]
.text:00000788                sub     ebx, eax
.text:0000078A                mov     [esi+ecx*4], ebx
.text:0000078D                inc     ecx
.text:0000078E                cmp     ecx, 19h
.text:00000791                jnz     short loc_785
.text:00000793
.text:00000793 loc_793:                               ; CODE XREF: .text:00000777↑j
.text:00000793                mov     eax, [ebp+4]
.text:00000796                add     eax, 3Eh ; '>'
.text:00000799                push    eax
.text:0000079A                retn
.text:0000079A ; ---------------------------------------------------------------------------
```

Esi+ecx*4 => ecx is 0 in the first round

```
Ecx ++
While Ecx != 19h (25)
Ecx++
Esi -> offset qr1
Esi + 64h
=> 25*qr1
qr1 =
qr1 + 4 => 203B = pin
Esi + 64 = qr1 + 64 = 1453 + 100 = 1553
```

Example for the calculation for the first block:

```
0203F FE
02040 B2
02041 F8
02042: 3
```

=> read from bottom up: 3F8B2FE, continue doing that and sum up ->

```
./lock 1098505442
```

FE B2 F8 03	03F8B2FE	66630398	1
82 9A 0A 02	020A9A82	34249346	2
BA 36 E8 02	02E836BA	48772794	3
BA B6 EB 02	02EBB6BA	49002170	4
BA 1A E9 02	02E91ABA	48831162	5
82 68 0B 02	020B6882	34302082	6
FE AA FA 03	03FAAAFE	66759422	7
00 DA 01 00	0001DA00	121344	8
54 4B EF 03	03EF4B54	66014036	9
2C A9 50 02	0250A92C	38840620	10
E0 9E 8E 01	018E9EE0	26124000	11
C8 39 56 03	035639C8	55982536	12
64 A4 4F 02	024FA464	38773860	13
00 01 E6 02	02E60100	48627968	14
AC 65 CA 02	02CA65AC	46818732	15
DE C5 91 02	0291C5DE	43107806	16
EE C7 2D 02	022DC7EE	36554734	17
3C B2 02 00	0002B23C	176700	18
BE 1E FA 03	03FA1EBE	66723518	19
2A B6 09 02	0209B62A	34190890	20

Egg 25 – Hidden Egg #1

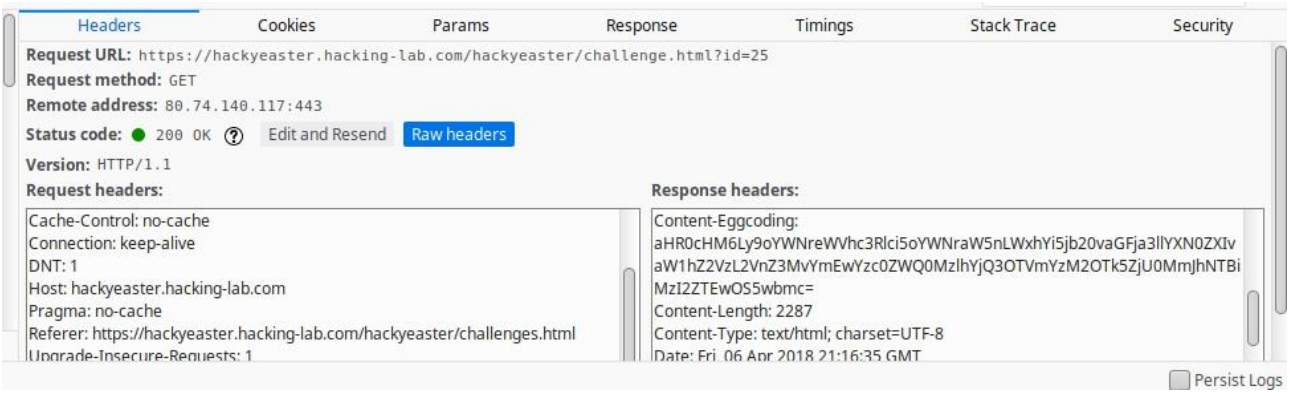
Level: hidden
Solutions: 237
Author: PS

Challenge

Heads up! You gonna find this hidden egg!

Solution by inik

First I thought it has to do with the html header, which was wrong. Then I had a look at the http header (using web developer) and found a base64 string:



Decoded I got the URL

```
https://hackyeaster.hackinglab.com/hackyeaster/images/eggs/ba0c74ed439ab4795fc36
999f542ba50b326e109.png
```

which was the egg.

Solution by khae

Looking at HTTP requests and replies, there is little gem:

```
Content-Eggcoding:
aHR0cHM6Ly9oYWNreWVhc3Rlci5oYWNraW5nLWxhYi5jb20vaGFja3llYXN0ZXIvaW1hZ2VzL2VnZ3MvYmEwYzc0ZWQ0MzlhYjQ3OTVmYzM2OTk5ZjU0MmJhNTBiMzI2ZTEwOS5wbmc=
```

Base64 decoded, we'll get the image of the egg:

```
https://hackyeaster.hacking-
lab.com/hackyeaster/images/eggs/ba0c74ed439ab4795fc36999f542ba50b326e109.png
```

Solution by pjslf

The *heads* word written in italics was obviously a hint so I took a look at the response headers. I found one particularly interesting: `Content-Eggcoding`.

It contained Base64-encoded URL of the egg.

```
$ wget https://hackyeaster.hacking-lab.com/hackyeaster/challenge.html?id=25
-O /dev/null -q -d 2>&1
 | grep Content-Eggcoding
 | cut -d' ' -f2
 | base64 -d
https://hackyeaster.hacking-
lab.com/hackyeaster/images/eggs/ba0c74ed439ab4795fc36999f542ba50b326e109.png

$ wget -O egg.png -q https://hackyeaster.hacking-
lab.com/hackyeaster/images/eggs/ba0c74ed439ab4795fc36999f542ba50b326e109.png
```

Egg 26 – Hidden Egg #2

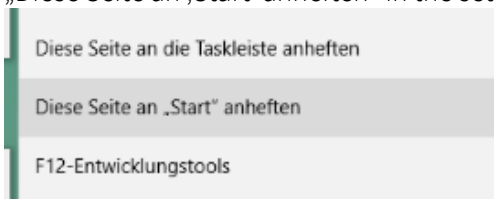
Level: hidden
Solutions: 111
Author: PS

Challenge

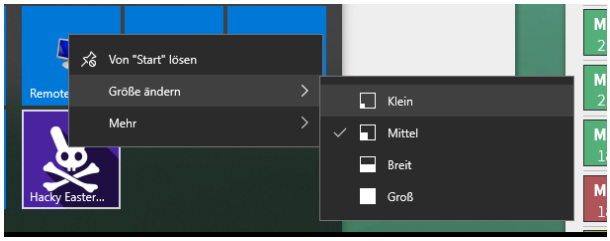
This egg is hidden in a very subtile manner. Perhaps you need to browse on the edge.

Solution by enigma69

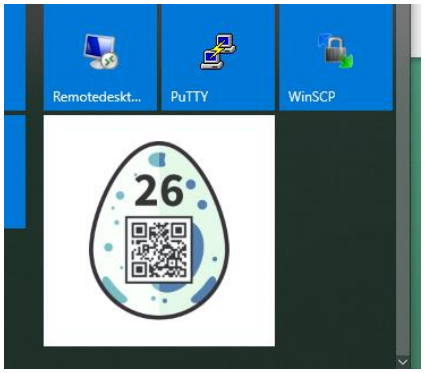
In order to solve this challenge, first I opened the Hacky-Easter webpage. After the page was loaded, I clicked „Diese Seite an ‚Start' anheften" in the settings menu:



The HackyEaster page was now available as tile in the Windows 10 start menu. Here I did a rightclick on the HackyEaster tile and changed the icon size from Middle to Large:



After then I got the easter egg and the challenge was solved:



Solution by SOKala

From the “This egg is hidden in a very subtile manner. Perhaps you need to browse on the edge.” statement, Looks like the hidden egg is located somewhere and the tool is Microsoft Edge.

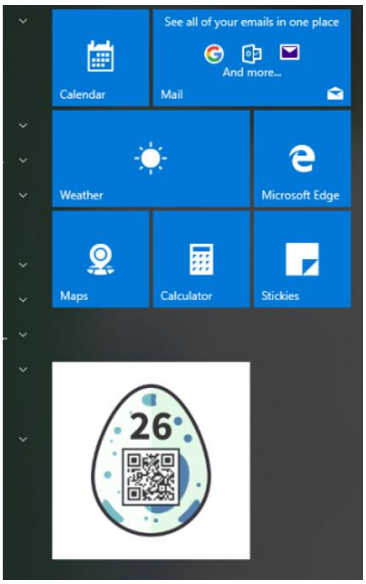
By searching tiles with Microsoft Edge, I found that all the tiles information are stored in browserconfig.xml file. By visiting **https://hackyeaster.hacking-lab.com/browserconfig.xml** URL, I got:

```
<browserconfig>
  <msapplication>
    <tile>
      <square70x70logo src="https://hackyeaster.hacking-
lab.com/hackyeaster/images/tiles/mstile70x70.png"/>
      <square150x150logo
src="https://hackyeaster.hackinglab.com/hackyeaster/images/tiles/mstile-
270x270.png"/>
      <square310x310logo src="https://hackyeaster.hacking-
lab.com/hackyeaster/images/tiles/mstile310x310.png"/>
      <wide310x150logo
src="https://hackyeaster.hackinglab.com/hackyeaster/images/tiles/mstile310x150.p
ng"/>
      <TileColor>#4923a0</TileColor>
    </tile>
  </msapplication>
</browserconfig>
```

https://hackyeaster.hacking-lab.com/hackyeaster/images/tiles/mstile-310x310.png
Bingo!!

Solution by beewasp

This egg is hidden in a very subtile manner. Perhaps you need to browse on the edge.
Opened URL in Edge.
Pinned to start menu (tile).
Changed tile size and egg was there!
VERY nice challenge ☺



Egg 27 – Hidden Egg #3

Level: hidden
Solutions: 290
Author: PS

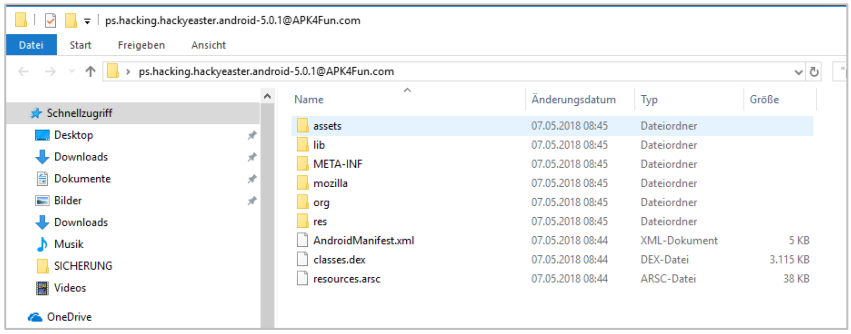
Challenge

Got *appetite*? What about an egg for launch?

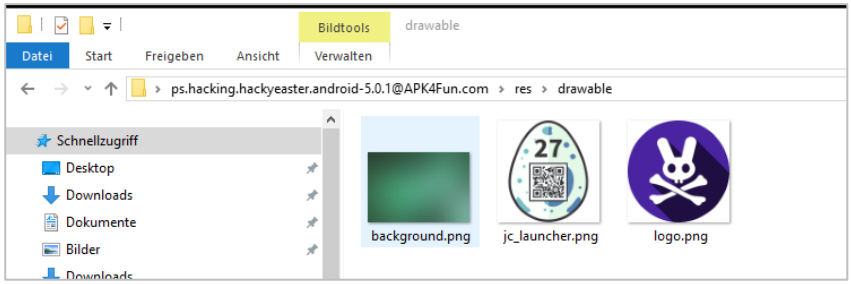
Solution by enigma69

In the challenge description there was a hint, that the easter egg could be found in an app (Got **app**etite). Of course that should be the Hacky Easter app, hopefully for Android. In order to examine this I downloaded the appropriate apk-file from https://www.apk4fun.com/apk/247768/ .

After downloading the file I unpacked it (because an apk-file is a packed archive like a zip-file). After the unpacking process I opened the new directory:

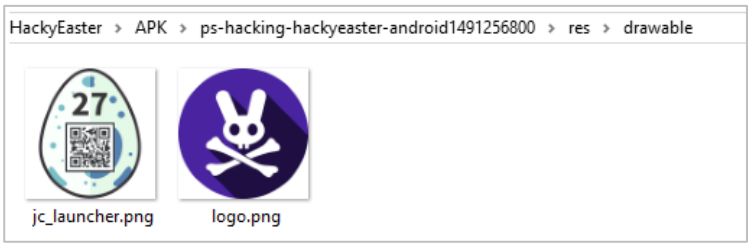


Ok, now it was time to search for the easter egg in the directory structure! After a short research I found the png-file in the directory /res/drawable/jc_launcher.png :



Solution by sym

The challenge 27 description has the word “app” written in italic. So the flag must be in the APK.
Extracting it and looking through it, reveals the flag in the resource directory: /res/drawable.



Solution by darkstar

After unpacking the apk file and listing all PNGs a hidden egg was found.

```
find . -iname \*. png -print 0 | xargs -I {} -0 cp -v {} ../pictures/
```