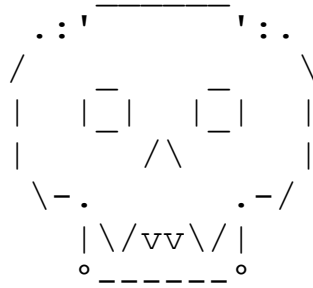


Hackyeaster Write Up



Made by TheVamp

Table of Contents

Challenge 01 – Easy One	4
Task	4
Solution	4
Challenge 02 – Just Cruisin’	5
Task	5
Solution	5
Challenge 03 – Bird’s Nest.....	6
Task	6
Task 2	6
Solution 2	6
Solution	7
Challenge 04 – Sound Check	8
Task	8
Solution	8
Challenge 05 – Play it again, Paul	10
Task	10
Solution	10
Challenge 06 – Going Up	11
Task	11
Solution	11
Challenge 07 – Wise Rabbit Once More	12
Task	12
Solution	12
Challenge 08 – Just Drive.....	13
Task	13
Solution	13
Challenge 09 – Brain Game	17
Task	17
Solution	17
Challenge 10 – Blueprint	19
Task	19
Solution	20
Challenge 11 – Twisted Disc	21
Task	21
Solution	22

Challenge 12 – Version Out Of Control.....	24
Task	24
Solution	24
Challenge 13 – Fractal Fumbling	26
Task	26
Solution	27
Challenge 14 – P.A.L.M.....	29
Task	29
Solution	29
Challenge 15 – Big Bad Wolf.....	32
Task	32
Solution	32
Challenge 16 – Egg Coloring	34
Task	34
Solution	34
Challenge 17 – Bunny Hop	36
Task	36
Solution	36
Challenge 18 – Bug Hunter	38
Task	38
Solution	38
Challenge 19 – Assemble This	41
Task	41
Solution	41
Challenge 20 – Humpt’s Dump.....	44
Task	44
Solution	44
Challenge 21 – Crypto Council.....	46
Task	46
Solutions.....	47
Challenge 22 – Dumpster Diving	48
Task	48
Solution	49
Challenge 23 - Heizohack	50
Task	50
Solution	50

Challenge 24 – crunch.ly	54
Task	54
Solution	54
Web-Submission Bomb	58

Challenge 01 – Easy One

Task

As always, the first challenge is very easy. Even babies can solve this one!

Find the code and enter it in the Egg-O-Matic™ below! One word, all lowercase.

xt	hex	yhi	dde	nyy	str
in	gyy	isy	ymo	lly	cod
dl	exy	sox	xsi	mpl	ey ☸

Solution

This is an easy one. Remove all yy, xy and then all x. The text is then “the hidden string is mollycoddle so simple”. So “mollycoddle” is the answer.

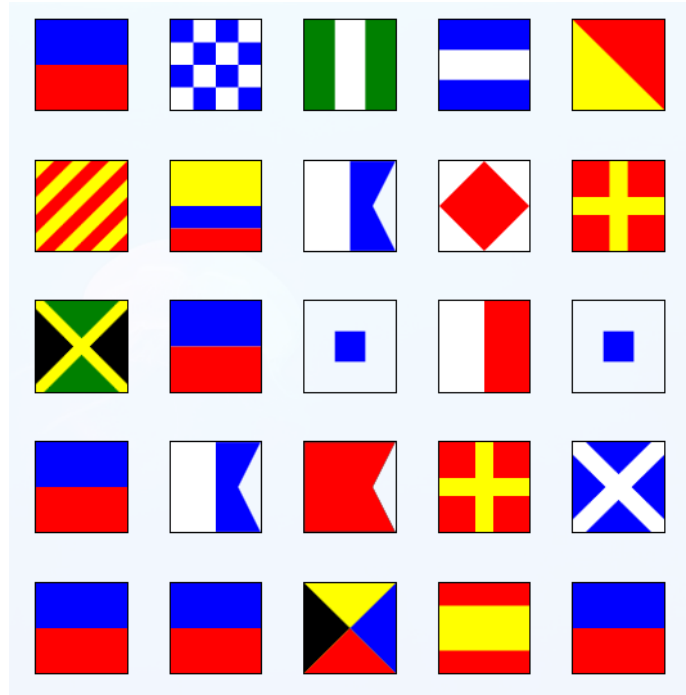


Challenge 02 – Just Cruisin’

Task

Need a holiday? Book a cruise on our new flag ship!

Seek out the promotion code below (lowercase only, no spaces) and get a free welcome package!



Solution

After a lot of google searching, I found the following site: http://www.usps.org/f_stuff/sigflgs.html

The Flags are international Signal Flags, so let’s translate the flags:

EN?JO
Y?AFR
?ESHS
EABRM
EEZ1E

Some flags are not documented on the site, so I filled the gap with a question mark. If you only take the letters you get “ENJOYAFRESHSEABRMEEZ1E”. It is not the solution, but sounds like “enjoy a fresh sea breeze”! The solution is “enjoyafreshseabreeze”. Maybe the flag signal guy drunk to much rum :D



Challenge 03 – Bird's Nest

Task

This is a mobile challenge. Check it out in the Hacky Easter app!

Wait, what? No way! I need a smartphone? I have no money for that ☹

Task 2

Reverse the APK and get the Task description!

Solution 2

OK, first of all, we need the APK file. We have the link to the google store:

<http://play.google.com/store/apps/details?id=ps.hacking.hackyeaster.android>

Now we only need an online apk-downloader. I found the following on the internet:

- <https://apps.evozi.com/apk-downloader/>
- <http://apkleecher.com/>

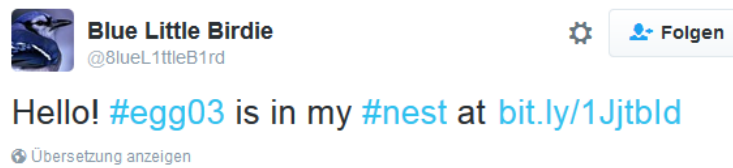
In my case, I used apkleecher, because evozi was not available.

If you know a little bit about APK reverse engineering, you know that APK files are normal ZIP files. You can unpack it out of the box. Within the assets folder I found a web-root, which contains all Challenge files. For challenge 3 the following website is displayed:



Solution

The little bird looks a little bit like the twitter bird. And the hashtags! A search on twitter after the hashtags #nest #egg03 showed up the following tweet:



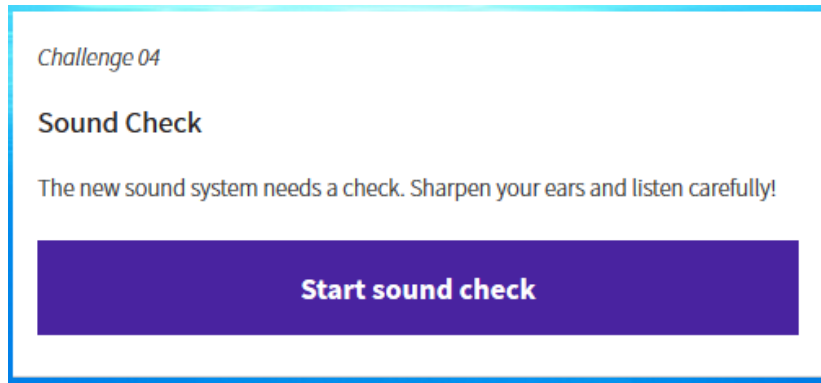
And if we follow the link, we get the egg #03



Challenge 04 – Sound Check

Task

This task was directly extracted from the APK file. Look for more information on the [Challenge 03](#) write up.

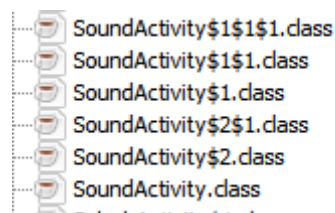


Solution

If you have the app, you should hear a sound, if you press on “Start sound check”. The easiest way, to find the frequency is another app, which analyses the frequency for you. I used in this case the app...
... *let’s look up in the app store*

OK, you don’t need such an app. You only need reversing for this task. In Challenge 03 I explained how you get the APK. For analyzing the sources of the apk file you need the awesome [Bytecode Viewer](#) an open source APK and Java Decompiling Tool!!!

But let us dive into the source code analysis. You find the code for the challenge in some SoundActivity.class:



After a little bit of analyzing I found out, that the goal is in SoundActivity\$2.class:

```

28     this.val$progressBar.post(new SoundActivity.2.1(this));
29     if (SoundActivity.access$300(this.this$0) < 5) {
30         break label206;
31     }
32     this.val$buttonSound.setVisibility(8);
33     this.val$buttonCheck.setVisibility(8);
34     this.val$spinner.setVisibility(8);
35     this.val$progressBar.setVisibility(8);
36     this.val$image.setVisibility(0);
37     arrayOfByte = Base64.decode(this.val$activity.getString(2131034134), 0);
38     for (int i = 0; i < arrayOfByte.length; i++) {
39         arrayOfByte[i] = ((byte) (i ^ arrayOfByte[i]));
40     }
41     SoundActivity.access$302(this.this$0, 0);
42 }
43 Bitmap localBitmap = BitmapFactory.decodeByteArray(arrayOfByte, 0, arrayOfByte.length);
44 this.val$image.setImageBitmap(localBitmap);
45 return;
46 label206:
47 this.val$buttonSound.setEnabled(true);
48 this.val$buttonCheck.setEnabled(false);
49 }
50 }

```

The important things happened from line 37 to 40. Load the base64 string from the resources.arsc, and do some XOR magic. Let's open the resources.arsc:

```

8 SoundActivity.NU0i4i4iVFMRAkPHA0ICQoC
XaZXL0cr9b3BxewM8LAV3eHe5e3xzvX5H7ir
ohLajpLE2aWq54K7kevitupjTTlNCKkQvANK
YWVobHKFm6WdhAsein8BOBYIEhlnY00obi97

```

This is our base64 String. But you need to know, that after `i4i4` the base64 string starts ;) Now I wrote a little python script, which will produce the egg for challenge 04:

```

import base64

base64data = '''[base64 data]'''

nearimage = base64.b64decode(base64data)
imgfile = ""

for i,c in enumerate(nearimage):
    imgfile = imgfile + chr((i ^ ord(c))%256)

print imgfile

```

Now I start the python script with "python SoundActivity.py > egg04.png". And that was it ☺



Challenge 05 – Play it again, Paul

Task

Do you know Paul? If not, it's about time to get to know him! Check out his video below!

Ĉu vi scias Paŭlo? Se ne, ĝi estas pri tempo ekkoni lin! Kontrolu lian video sube!

[Here is an embedded video, you can watch the original here:

<https://www.youtube.com/watch?v=Qgwr407AChs>]

Solution

OK, a video? Let's look at the source code:

```
<video id="video" class="video-js vjs-default-skin"
  controls preload="metadata" width="480" height="400"
  poster="paul/poster.png"
  data-setup=''>
  <source src="http://media.hacking-lab.com/hackyeaster/he2016/video/video.mp4" type="video/mp4">
  <source src="http://media.hacking-lab.com/hackyeaster/he2016/video/video.webm" type="video/webm">
  <source src="http://media.hacking-lab.com/hackyeaster/he2016/video/video.ogv" type="video/ogg">
  <track label="English" kind="captions" srclang="en" src="paul/video-en.vtt" default>
  <track label="Deutsch" kind="captions" srclang="de" src="paul/video-de.vtt">
  <track label="Français" kind="captions" srclang="fr" src="paul/video-fr.vtt">
  <track label="Esperanto" kind="captions" srclang="eo" src="paul/video-eo.vtt">
</video>
```

The second sentence in the task is Esperanto. A look in that translation-file reveals maybe something:

```
1
00:00:56.600 --> 00:00:58.700
Filmo de Viviane Michel

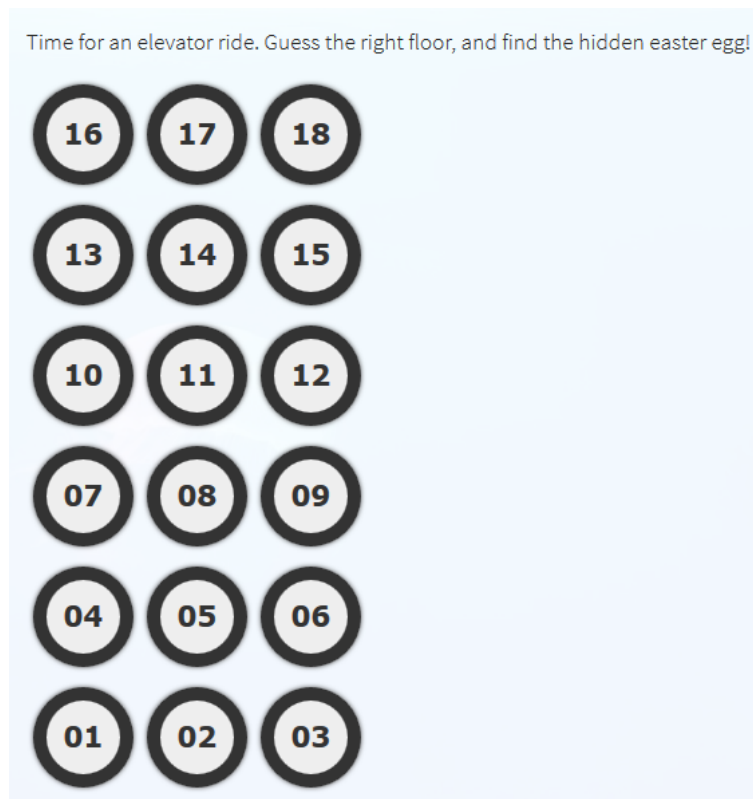
1
00:01:00.000 --> 00:01:03.000
passphrase is "youtubelotitis"
```

Cool, there is the code.



Challenge 06 – Going Up

Task



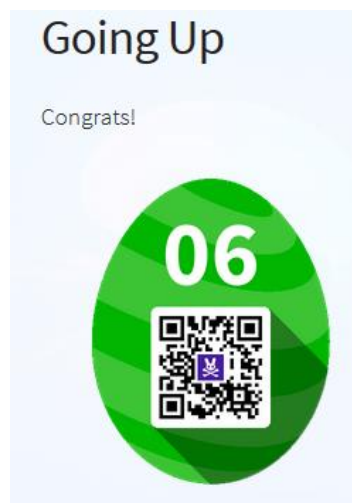
[This are the elevator button. And behind every number is a link to the floor]

Solution

If you read the source code, you notice that the link from the thirteenth floor is a little bit odd:

```
="width: 25%; padding-bottom: 10px;";
<div class="round-button-circle"><a href="?sybbe=punatrzr" cla
<div class="round-button-circle"><a href="?floor=fourteen" cla
<div class="round-button-circle"><a href="?floor=fifteen" clas
```

"sybbe=punatrzr" is in ROT13 "floor=changeme". So I submitted the ROT13 of "floor=thirteen" ("sybbe=guvegrra"):



Challenge 07 – Wise Rabbit Once More

Task

Wise Rabbit says:

The solution is in the solutions!

Go back and scroll to 123!

Solution

If you know wise rabbit from last year, he likes to play hide and seek. On the home page of Hackyeaster (HE), you find the links to the solutions of the last years. The solution from 2015 is exactly 123 pages big. A look on the last page reveals:

password: goldfish

Damn you wise rabbit. This was hard to find! :D

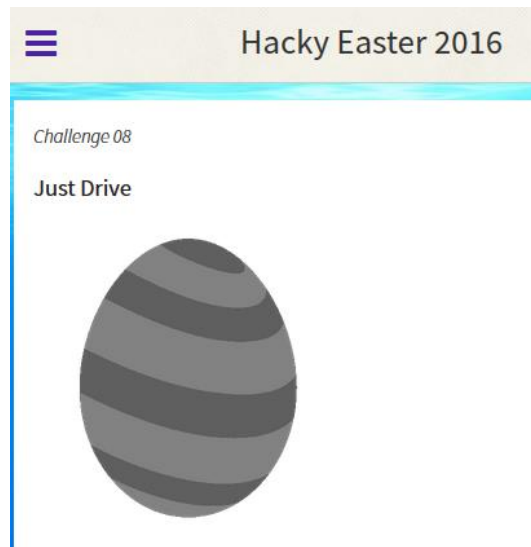
But now I have the next egg:



Challenge 08 – Just Drive

Task

This task was directly extracted from the APK file. Look for more information on the [Challenge 03](#) write up.



Solution

Hmm... Just Drive? OK, start the decompiler!

First I recommend to read [Challenge 03](#) and [Challenge 04](#). There is described how I get the APK and which tool I use for the reverse-engineering process.

Before I begin to explain the reversing process, I must mention that the website, where I take the screenshot for the task, automatically redirected to `ps://rot?h=!` So I search for this link in the source code. After a small look up I found the URL in the Activity.class:

```
private static final String URL_HOME = "ps://home";  
private static final String URL_MYSTATUS = "ps://mystatus?";  
private static final String URL_ROT = "ps://rot?";  
private static final String URL_SCANNER = "ps://scan";  
private static final String URL_SHARE = "ps://share";
```

So our task is internally called ROT. Let's look up, if we found similar functions with the name ROT in it.

```
private String handleRot(String paramString)  
{  
    char c;  
    int i;  
    Object localObject;  
    switch (((WindowManager) getSystemService("window")).getDefaultDisplay().getRotation())  
    {  
        case 2:  

```

"handleRot" seems to be our function. From line 86 to 157. But you see already, that you should rotate your smartphone like a staring wheel. But I don't have a smartphone, so no fun for me!

From line 119 to line 149 we have a big block which do the rotating checks:

```

119     if ((paramString.length() >= 7) && ("1ab9a97066f747c25d9c6a6b0fda647fae98cb98".equals(sha1(paramString.substring(0, 7)))))
120     {
121         i = 7;
122     }
123     else if ((paramString.length() >= 6) && ("58192f7d1263bd420efb788cc884a84f871239cf".equals(sha1(paramString.substring(0, 6)))))
124     {
125         i = 6;
126     }
127     else if ((paramString.length() >= 5) && ("e8f235e79be9f8b13598b285a6fdaf2ac70a66ca".equals(sha1(paramString.substring(0, 5)))))
128     {
129         i = 5;
130     }
131     else if ((paramString.length() >= 4) && ("3daee0c0bada99f5bf0866728fd78299acaafa15".equals(sha1(paramString.substring(0, 4)))))
132     {
133         i = 4;
134     }
135     else if ((paramString.length() >= 3) && ("f11fa81c0b72716bba0536dd34b9b0987af69b03".equals(sha1(paramString.substring(0, 3)))))
136     {
137         i = 3;
138     }
139     else if ((paramString.length() >= 2) && ("a8643e0e26d5ead82e73aae64966ca144f152d8a".equals(sha1(paramString.substring(0, 2)))))
140     {
141         i = 2;
142     }
143     else
144     {
145         int j = paramString.length();
146         i = 0;
147         if (j >= 1)
148         {
149             boolean bool = "4dc7c9ec434ed06502767136789763ec11d2c4b7".equals(sha1(paramString.substring(0, 1)));

```

You see some SHA1 values. Which represents the string of the rotation. The last SHA1 have only one character, because of the substring-function. So you can do it by hand, bruteforce it or just google it:

4dc7c9ec434ed06502767136789763ec11d2c4b7 = "r"

The second hash have only 2 characters. substring(0, 2) so it should be also easy to get this one:

a8643e0e26d5ead82e73aae64966ca144f152d8a = "nr"

Ok, the rotation goes further on. I analyzed the script a little bit more and found out, that the SHA1 values are have only the characters "r", "n" and "x" (see line 113 to 117). So you only have 3 rotations. With this knowledge I should be able to write a fast bruteforce script in python:

```

import hashlib

shalvalues = ['4dc7c9ec434ed06502767136789763ec11d2c4b7',
              'a8643e0e26d5ead82e73aae64966ca144f152d8a',
              'f11fa81c0b72716bba0536dd34b9b0987af69b03',
              '3daee0c0bada99f5bf0866728fd78299acaafa15',
              'e8f235e79be9f8b13598b285a6fdaf2ac70a66ca',
              '58192f7d1263bd420efb788cc884a84f871239cf',
              '1ab9a97066f747c25d9c6a6b0fda647fae98cb98',
              '4692bd56dd3070f74b7e']

bs = "rnx"
ps = ""

for i in xrange(0, len(shalvalues)):
    for c in bs:
        if hashlib.sha1(c+ps).hexdigest()[0:18] == shalvalues[i][0:18]:
            ps = c + ps
            print ps + ":" + hashlib.sha1(ps).hexdigest()
            break;

```

And the results are the following:

```

r:4dc7c9ec434ed06502767136789763ec11d2c4b7
nr:a8643e0e26d5ead82e73aae64966ca144f152d8a
rnr:f11fa81c0b72716bba0536dd34b9b0987af69b03
xrnr:3daee0c0bada99f5bf0866728fd78299acaafa15
rxnr:e8f235e79be9f8b13598b285a6fdaf2ac70a66ca
rxrnr:58192f7d1263bd420efb788cc884a84f871239cf
rxrxnr:1ab9a97066f747c25d9c6a6b0fda647fae98cb98
nrxxrnr:4692bd56dd3070f74b7e81c6b2f69339b0fd6062

```

So the solution is rnrxrnr. But why I have a small part of a SHA1 hash in my source code? Because it is in the reversed code as well:

```

103     localObject = "";
104     String str = sha1(paramString);
105     if (str.startsWith("4692bd56dd3070f74b7e")) {
106         localObject = str;
107     }

```

OK but where is our egg? Maybe in the html-file, where the challenge is displayed. Yes, in the assets folder within the web root you found challenge08.html. There is a scrambled egg. I figured already in [Challenge 11](#) out, how I can unscramble the scrambled eggs☺. Within the web-source-code is another important fact:

```
+ CryptoJS.enc.Latin1.stringify(CryptoJS.AES.decrypt(scrambledEggCipher, json.k));
```

It sends k as key, to unscramble the key. But what is k? A look back in the decompiled code said:

```

    localObject = "";
    String str = sha1(paramString);
    if (str.startsWith("4692bd56dd3070f74b7e")) {
        localObject = str;
    }
    break;
}
for (;;)
{
    return "{ \"s\": \" + i + \", \"h\": \"\" + paramString + "\", \"k\": \"\" + (String) localObject + "\" }";
}

```

So you see, that k is the localObject, which is the SHA1 of our solution. So our key is "4692bd56dd3070f74b7e81c6b2f69339b0fd6062".

Here is the python code, to unscramble the egg:

```
from Crypto.Cipher import AES
from Crypto.Hash import MD5
import base64

sEgg'''[base64-code of scrambled Egg]'''
goal = '''iVBORw0KGgoAAAANS'''#PNG-Header

secret = "4692bd56dd3070f74b7e81c6b2f69339b0fd6062"
encoded = sEgg
encrypted = base64.b64decode(encoded)
salt = encrypted[8:16]
data = encrypted[16:]

try:
    def openssl_kdf(req):
        prev = ''
        while req>0:
            prev = MD5.new(prev+secret+salt).digest()
            req -= 16
            yield prev
    mat = ''.join([ x for x in openssl_kdf(32+16) ])
    key = mat[0:32]
    iv = mat[32:48]

    dec = AES.new(key, AES.MODE_CBC, iv)
    clear = dec.decrypt(data)
    if clear[:5] == goal[:5]:
        print base64.b64decode(clear)
except:
    nothing = ""
```

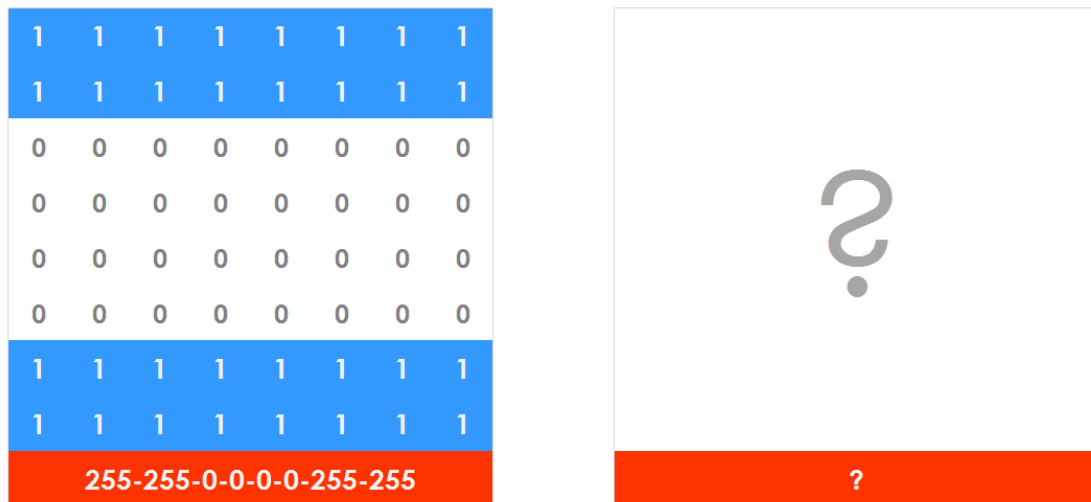
First put the base64 code into the script and the run “justdrive.py > justdrive.png”:



Challenge 09 – Brain Game

Task

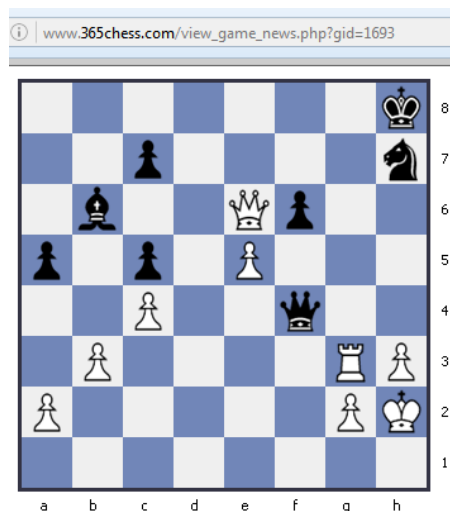
What about a little brain game?



1. e4 e5 2. Nf3 Nc6 3. Bb5 Nf6 4. d3 Bc5 5. O-O d6 6. Nbd2 O-O 7. Bxc6 bxc6 8. h3 h6 9. Re1 Re8 10. Nf1 a5 11. Ng3 Rb8 12. b3 Bb4 13. Bd2 Ra8 14. c3 Bc5 15. d4 Bb6 16. dxe5 dxe5 17. c4 Nh7 18. Qe2 Nf8 19. Be3 c5 20. Rad1 Qf6 21. Nh5 Qe7 22. Nh2 Kh7 23. Qf3 f6 24. Ng4 Bxg4 25. Qxg4 Red8 26. Qf5+ Kh8 27. f4 Rxd1 28. Rxd1 exf4 29. Bxf4 Qe6 30. Rd3 Re8 31. Nxg7 Kxg7 32. Qh5 Nh7 33. Bxh6+ Kh8 34. Qg6 Qg8 35. Bg7+ Qxg7 36. Qxe8+ Qf8 37. Qe6 Qh6 38. e5 Qc1+ 39. Kh2 Qf4+ 40. Rg3 1-0

Solution

Hmm... the image looks like a chess game. If you google the hint at the bottom, you get to <http://www.365chess.com/news> where many games are documented. A further google research reveals it was a game at the “77th Tata Steel 2015 Round 9” between Carlsen, M and Radjabov, T.



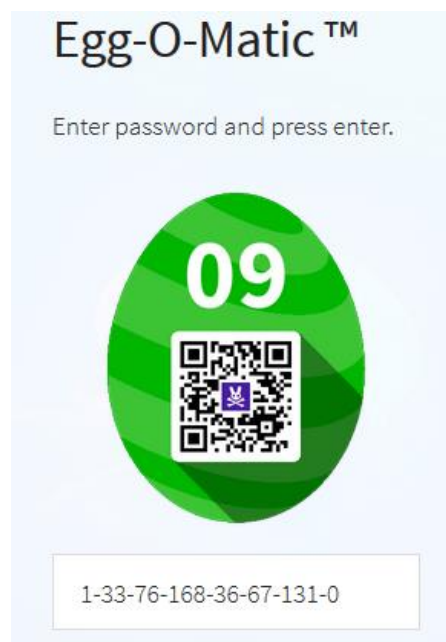
So our binary code should look like this:

```
00000001
00100001
01001100
10101000
00100100
01000011
10000011
00000000
```

Now we must translate every row from top to bottom and from binary to int:

1-33-76-168-36-67-131-0

And this is our final solution :)



Challenge 10 – Blueprint

Task

Time for some math! Find the number which produces the plot on the bottom!

Try these two samples: [sample 1](#), [sample 2](#).

```
607016736537520995594815505941151025266085572371665528200038731931478673846
727319684910259261305939019800769557502357959237333126657517770919783070679
372916294368356299408372540498306857438421105135681283249346869449792967320
249238095120706670142921097242873599571883821374208364354014530087192947246
803225868116625497030891935667180689024971315282228337059319954976085961871
005910163343474177499795224476259823484556802260391642987108868423291931973
521858141803706752591127589236027411683906358835596869641180286211179357749
072326208778420088
```

Plot!

Your plot:



Target plot:



Image from the Challenge with sample 2

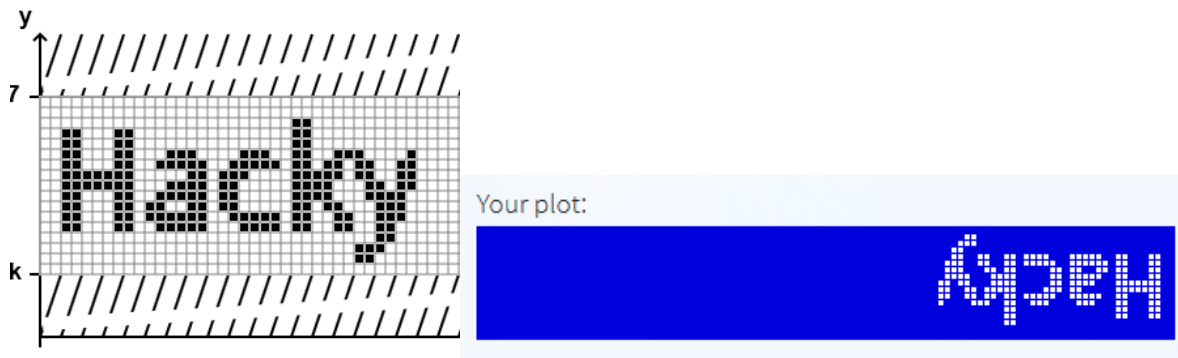
Solution

First of all, this was my last challenge I solved. I found it really hard, if you don't know what this is. My first attempts was some terrible Math and some bruteforce things. So totally stupid stuff. After a while I looked up the equation from the JavaScript in the source code in google. The formula looked like this:

$$((y/17) / \text{pow}(2, (17*x) + (y\%17))) \% 2 > 0.5$$

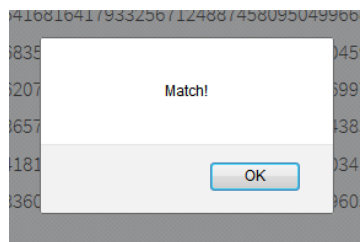
Google said it is the [Tupper's self-referential formula](http://tuppers-formula.tk/)! I must only find a website, where I can paint the target plot. And indeed, there was this website: <http://tuppers-formula.tk/>

So I began to paint, and almost finished the word "Hacky". For some reason, I wanted to see if it worked on the HE-Site:



Damn you! So I painted It again, and this time right. The plot number is now:

176579492015814901528872625529774615508215478614638628392406643239116428074897541
681641793325671248874580950499668382723958388333354648532262931698930639856835422
348683939828636055448533804591049653503261373974416464862181695983478562079067833
614229059113869197437699759742373674003028861535476027091552243616865735457697656
105444429506238584383051262100293283222118456901855469818763894181110080508013645
884497726056403410392923221554648832542085467262023169013883606836051142881849628
64450110296056848249404578342545423849729556480

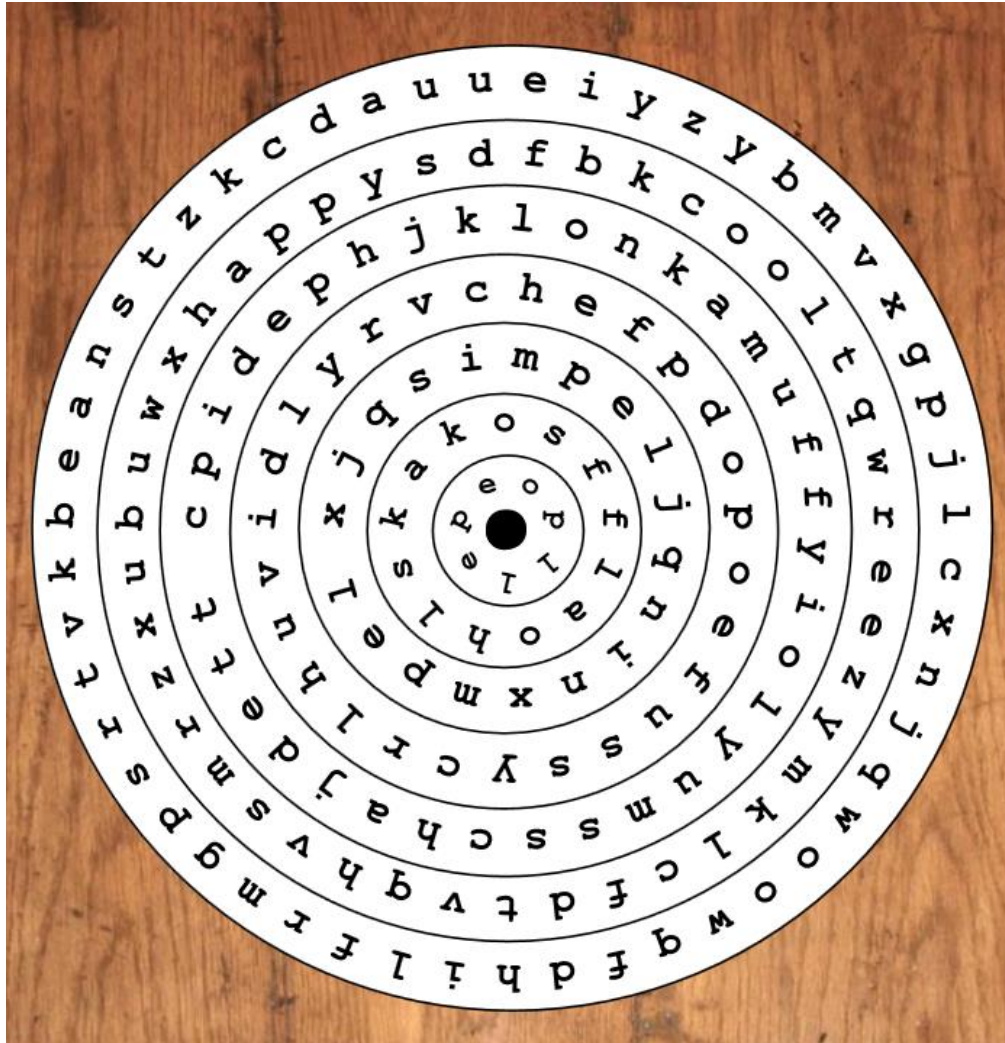


Challenge 11 – Twisted Disc

Task

You found a secret disc which conceals a secret password. Can you crack it?

Hint: Each ring of the disc holds one letter. The first letter sits on the outermost ring.



Solution

The first relevant point on this challenge is, to understand how the Egg-O-Matic works. Why? If we understand how it worked, we can bruteforce everything offline.

```
<script type="text/javascript" src="js/crypto-js/aes.js"></script>
<script type="text/javascript" src="js/crypto-js/core-min.js"></script>
<script type="text/javascript" src="js/crypto-js/enc-base64-min.js"></script>
<script>setChallId(11);</script>
<script>
    scrambledEggCipher = 'U2FsdGVkX19812lAuS9M/suVrF1KwaspK+a7/BT3lFkPqeUMteG
</script>
```

If you analyze the source code, you see that the scrambledEgg (see Image above) is encrypted with AES from crypto-js. After hours of research I found the equivalent decryption in python on [stackoverflow](https://stackoverflow.com). So now can the challenge begin!

First of all, I wrote down all rings:

```
Ring0: "ueiyyzybmvxgpgjlcxnjqwoowqfdhilfrmgpsrtvkbeanstzkcd"
Ring1: "dfbkcooltqwreezymklcfdvtqhvsmrxxubuwxhappys"
Ring2: "lonkamuffyiolyumsschajdettcpidephjk"
Ring3: "chefpdopoeufussycrlhuvidlyrv"
Ring4: "simpeljqninxmpelxjq"
Ring5: "kosfflaohlksa"
Ring6: "eopllep"
```

Because we try to bruteforce the 7 character long password, we should first remove all double characters, to improve the bruteforce speed. So here is the optimized python script:

```
import base64
from Crypto.Cipher import AES
from Crypto.Hash import MD5

sEgg='''[scrambledEgg]'''
goal = '''iVBORw0KGgoAAAANS'''#PNG Header Base64

ba = "abcdefghijklmnopqrstuvwxyz"
bb = "abcdefghijklmnopqrstuvwxyz"
bc = "abcdefghijklmnopqrstuvwxyz"
bd = "abcdefghijklmnopqrstuvwxyz"
be = "abcdefghijklmnopqrstuvwxyz"
bf = "abcdefghijklmnopqrstuvwxyz"
bg = "abcdefghijklmnopqrstuvwxyz"

for a in ba:
    for b in bb:
        for c in bc:
            for d in bd:
                for e in be:
                    for f in bf:
                        for g in bg:
                            secret = a+b+c+d+e+f+g
                            encoded = sEgg
                            encrypted = base64.b64decode(encoded)
                            salt = encrypted[8:16]
                            data = encrypted[16:]
                            try:
                                def openssl_kdf(req):
                                    prev = ''
                                    while req>0:
```

```

prev =
MD5.new(prev+secret+salt).digest()
    req -= 16
    yield prev
mat = ''.join([ x for x in

openssl_kdf(32+16) ])

key = mat[0:32]
iv  = mat[32:48]

dec = AES.new(key, AES.MODE_CBC, iv)
clear = dec.decrypt(data)
if clear[:5] == goal[:5]:
    print secret
except:
    nothing = ""

```

The final answer is “hanisho” and here is the egg:



Challenge 12 – Version Out Of Control

Task

Version control is a powerful tool. Thinking she was oh so smart, Fluffy used it to hide an easter egg. Can you pull out the egg from her file?

Hint: If you get stuck, go one step back.

[Download her file](#)

Solution

Yeah, it is a little git chall. First you must unzip the folder, change into the directory, and do a git stash. This should be done like a thousand times, so I wrote a little bash script for that:

```
#!/bin/bash

for i in {1..999}
do
    unzip ./*.zip
    cd *
    git stash
done
```

OK, the script stopped at 722, because instead of a Zip file was an image in the Zip file!



A further investigation revealed, in zip 723 was the last valid unpacking, before the image within the zip file appears. "Git whatschanged" shows that there are three commits. So let's test the first commit "git checkout 93d630215b9c5c49f2c7f3c6b9fe1b55efd93cd1". After unpacking the zip, we got the same zips as before. So let's try it from here again.



Another stop, damn. This time at the directory 0397! Within the folder was only this image (see above). Checking out the branch in git: "git show-branch" ..., ok wrong branch: "git checkout blaster". And unpacking again

```
Caution: filename not matched: ../0000.zip
Saved working directory and index state WIP on master: 37f69df Commit committed.
Pass is fluffy99
HEAD is now at 37f69df Commit committed. Pass is fluffy99
Archive: ../0045.zip
  creating: 0045/
  creating: 0045/.git/
  creating: 0045/.git/branches/
[../0045.zip] 0045/.git/COMMIT_EDITMSG password: 
```

Now my script stopped and ask for a password. Thx god it stand a few lines above. After this nothing happened. Still unpacking without errors. On directory 0001 we got the egg :)



Challenge 13 – Fractal Fumbling

Task

Do you need a new wallpaper? What about a fancy fractal?

Find the password hidden in the wallpaper image, and enter it in the Egg-O-Matic below.

[Download wallpaper](#)



The image is 9261 x 9261 pixel big! If you zoom in it looks like this:



Solution

OK for this QR-Fractal we need the following steps:

1. Get rid of that bunny, because it causes background noise and maybe the QR-reader can't read some of the QR-Codes
2. Read every QR-Code and save them in a list
3. Try every result from the generated list as password to unscramble the egg
 - You know the unscrambled egg has a PNG header ;)
 - Unscrambling eggs started at [Challenge 11](#)

Sounds easy, so here is my python source code:

```
from PIL import Image
from qrttools import QR
import PIL
import base64
from Crypto.Cipher import AES
from Crypto.Hash import MD5

img = Image.open("./wallpaper.jpg")
width, height = img.size
print "get rid of the bunny"
im = img.load()
for x in xrange(0,width):
    for y in xrange(0,height):
        if im[x,y][0] > 25 and im[x,y][1] > 25 and im[x,y][2] > 25:
            im[x,y] = (255,255,255)
print "create dictionary"
content = []
for x in xrange(0,width,21):
    for y in xrange(0,height,21):
        box = (x,y,x+21,y+21)#get 1 QR-Code
        oimg = img.crop(box)#cut QR-Code out
        oimg = oimg.resize((84,84), PIL.Image.ANTIALIAS)#resize image
        oimg.save("temp.jpg")#save it temporaly
        qr = QR(filename="temp.jpg")#read QR-Code
        if qr.decode():
            content.append(str(qr.data))#decoded QR-Code into dict

print "dict-attack"
goal = '''iVBORw0KGgoAAAANS'''#PNG-Header
sEgg = '''[scrambledEgg-base64]'''
for key in content:
    secret = key
    encoded = sEgg
    encrypted = base64.b64decode(encoded)
    salt = encrypted[8:16]
    data = encrypted[16:]
    try:
        def openssl_kdf(req):
            prev = ''
            while req>0:
                prev = MD5.new(prev+secret+salt).digest()
                req -= 16
            yield prev
        mat = ''.join([ x for x in openssl_kdf(32+16) ])
        key = mat[0:32]
        iv = mat[32:48]

        dec = AES.new(key, AES.MODE_CBC, iv)
```

```
clear = dec.decrypt(data)
if clear[:5] == goal[:5]:
    print secret
except:
    nothing = ""
```

And the solution is “fractalsaresokewl”:

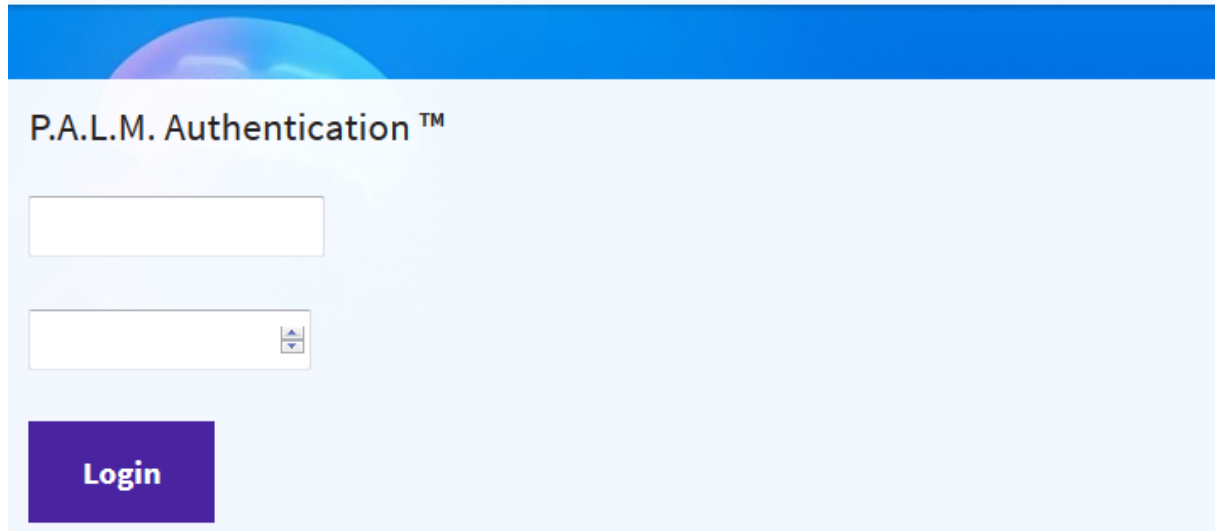


Challenge 14 – P.A.L.M.

Task

Folks at *HOB0 Authentication Systems* implemented a new authentication system named P.A.L.M.™

Prove that you can break it and find a pair of username and passcode to log on.



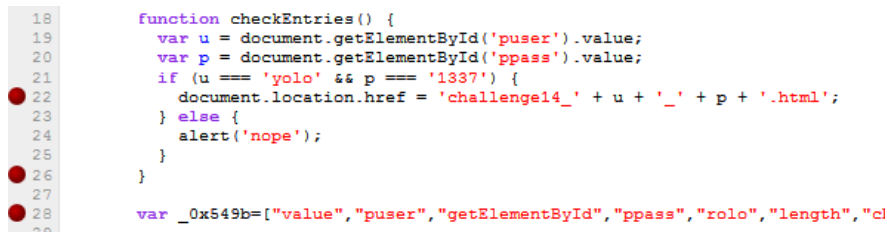
Solution

The first step is a little source code review:

```
function checkEntries() {
    var u = document.getElementById('puser').value;
    var p = document.getElementById('ppass').value;
    if (u === 'yolo' && p === '1337') {
        document.location.href = 'challenge14_' + u + '_' + p +
    } else {
        alert('nope');
    }
}

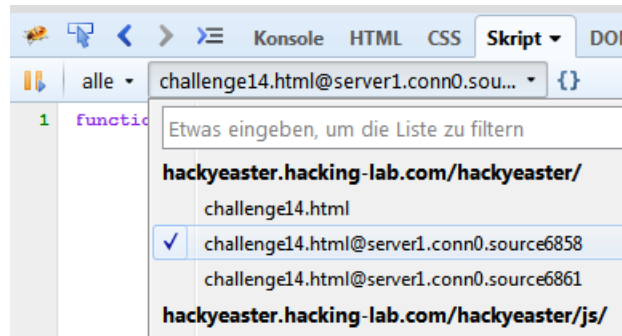
var _0x549b=["value","puser","getElementById","ppass","rolo","l
```

So the function said User is “yolo” and password is “1337”, but below that is var _0x549b which is a large line with the same checkEntries function. So it’s overwrite the first one. Let’s set some breakpoints in Firebug:



```
18 function checkEntries() {
19     var u = document.getElementById('puser').value;
20     var p = document.getElementById('ppass').value;
21     if (u === 'yolo' && p === '1337') {
22         document.location.href = 'challenge14_' + u + '_' + p + '.html';
23     } else {
24         alert('nope');
25     }
26 }
27
28 var _0x549b=["value","puser","getElementById","ppass","rolo","length","cl
29
```

After I pressed the Login-Button the script doesn’t stopped as it should, maybe there is another point where the script is executed:



There are 2 hidden scripts. After setting some breakpoints and tried to login, the script broke at the breakpoint from [challenge14.html@server1.conn0.source6858](#), but not at the second one. So only the first script should be the right one. I extracted the script and beautified it with the ScriptDeobfuscator from KahuSecurity. Here is the “beautiful” script:

```
function checkEntries() {
    var u=document.getElementById('puser').value;
    var p=document.getElementById('ppass').value;
    var used=[0,0,0,0,0,0,0,0,0,0,0];
    var ok=false;

    if(u==='elsa') {

        if(p>0&&p.length==10) {
            ok=true;for(i=1;i<=10;i++) {
                var digit=p.charAt(i-1);
                var part=p.substring(0,i);
                if(used[digit]!=0||part%i!=0) {
                    ok=false
                }

                if(used[digit]==0) {
                    used[digit]=1
                }
            }
        }

        if(ok) {
            document.location.href='challenge14_'+u+'_'+p+'.html'
        }
        else {
            alert('nope')
        }
    }
}
```

So the username is “elsa”. Now we only must find the right number. The number has a length of 10 and every digit may only be used once. I wrote a little python script, which generates some possible numbers:

```
for x in xrange(123456788,1000000000):
    clist = list((str(x).zfill(10)))
    plist = set(clist)
    if len(plist) > 9:
        print ''.join(clist)
```

Executing “python palmgen.py > palm_codes.dic” and I get a little dictionary.

In the next phase, I re-implemented the Authentication Script from JavaScript to python and bruteforced it with my dictionary:

```
print "loading all palmcodes..."
with open("./palm_codes.dic ") as f:
    content = f.read().splitlines()
print "try all palmcodes... valid palmcodes will be printed"
for l in content:
    c = list(l)
    part = ''
    cool = 1
    for i in xrange(1,11):
        part += str(c[i-1])
        if int(part)%i != 0:
            cool = 0
            break;
    if cool == 1:
        print ''.join(c)
```

After a while I got the first Code: 3816547290 and with that we have our next egg ☺



Challenge 15 – Big Bad Wolf

Task

Three little pigs have hidden in their house. You're the big, bad wolf, and your stomach is growling. Huff and puff and blow the pigs' house in! Get that juicy bacon!

Hints:

- the pigs have hidden in three different media types (image, sound, text)
- no password cracking is necessary

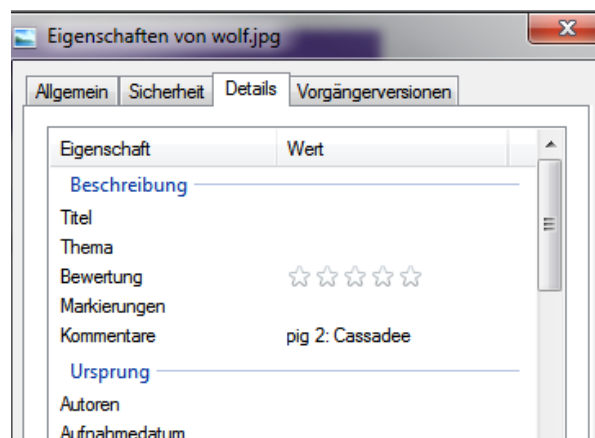
[Download the pigs' file](#)

Solution

I downloaded the pigs' file, which is a "disk.img" file. I can extract the data easily with 7zip:

Name	Änderungsdatum	Typ	Größe
lost+found	29.11.2015 07:51	Dateiordner	
disk.img	21.03.2016 18:37	IMG-Datei	16.034 KB
piglet.jpg	29.11.2015 07:51	JPEG-Bild	92 KB
pigs.jpg	29.11.2015 07:51	JPEG-Bild	94 KB
song.mp3	29.11.2015 07:51	MPEG layer 3	2.593 KB
story.pdf	29.11.2015 07:51	Adobe Acrobat D...	72 KB
story.txt	29.11.2015 07:51	TXT-Datei	2 KB
wolf.jpg	29.11.2015 07:51	JPEG-Bild	187 KB

We know, that the pigs are hidden within an image, a sound and a text. First the image, I looked a little bit in the properties and found immediately the second pig:



Now we look up at the text files. I found something suspicious in the story.txt:

1 THE THREE LITTLE PIGS → → → → → → → → → → CRUE

2 → → → → → → → → → → → → → → → → CRUE

3 Once upon a time, there were three little pigs who went off to build their houses. CRUE

4 → → → → → → → → → → → → → → → → CRUE

5 The first little pig built his house of straw, which was not very strong. CRUE

6 → → → → → → → → → → → → → → → → CRUE

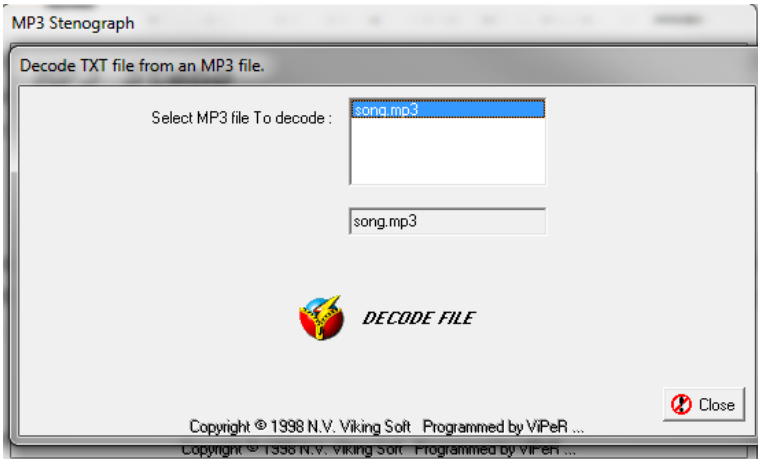
7 One day, the big bad wolf came and said, "Little pig, little pig, let me come in." CRUE

8 → → → → CRUE

What is all that noise with whitespaces and tab's? I tried to decipher it manually, but it doesn't worked. After a little research on google, I came across this [presentation](#). SNOW seems to be the right candidate:

```
2>SNOW.EXE ..\story.txt
pig 1: Filbert
```

Now we need only the mp3 file. Listening and some basic analysis didn't helped. Again after a little google research on Stegano tools, I found a good list on [Wikipedia](#). Maybe MP3Stego is the tool I searched for. I found also a GUI for that tool 😊:



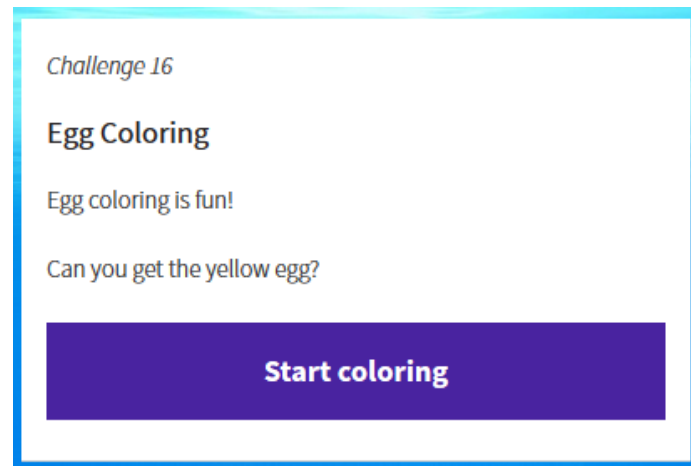
For the passphrase I used nothing. An empty password. The output was a text file with the content "pig 3: Wynchell". We found the three pigs and got the next egg on this journey \o/



Challenge 16 – Egg Coloring

Task

This task was directly extracted from the APK file. Look for more information on the [Challenge 03](#) write up.

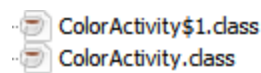


Solution

Hmm... How could I get it? Only with reverse engineering!!! :D

First I recommend to read [Challenge 03](#) and [Challenge 04](#). There is described how I get the APK and which tool I use for the reverse-engineering process.

First we have this time two classes to analyze:



The ColorActivity\$1.class is not interesting. The only thing what it does is, to pass the values from a spinner to the main-class. The Main-Class is more interesting:

```
16 public class ColorActivity
17     extends Activity
18 {
19     private static final String EGG_URL = "http://hackyeaster.hacking-lab.com/hackyeaster/egg";
20     private static final String KEY = "eggsited";
21     private static String[] codes = { "ff0000", "00ff00", "0000ff", "00ffff", "ff00ff", "ffffff", "000000" };
22     private static String[] hmacs = { "f4e075524ba4470867e1891c1a8d1fc21dflf56a", "b23f66454417de5be448da84a846989b42f304c8",
    "f5ecdd0f12749fe75734b42bf29943d28acf4573", "2cf2a7cd695a462adcbc324df9302003a99c688a", "543e3853ac9318587c10c7645b6828e2a858ecf5",
    "c46ffadf392698e28fdeb344239130e2ade2c809", "7b06466eb80d88533a2d1c7b9de62d98c4e20d1d" };
23
24     private void colorize(int paramInt)
25     {
26         String str = "http://hackyeaster.hacking-lab.com/hackyeaster/egg?code=" + codes[paramInt] + "&key=" + "eggsited" + "&hmac=" + hmacs[paramInt];
27         ImageView localImageView = (ImageView) findViewById(2131296277);
28         new EggLoader().execute(new Object[] { str, localImageView, this });
29     }
30 }
```

As you see, we have an EGG_URL, a key, some codes and some SHA1-HMAC's. The codes are the RGB colors in hex. At the following site you can generate SHA1-HMAC's:

<http://www.freeformatter.com/hmac-generator.html>. Here is the result one for the red color:

Copy-paste the message here

ff0000

Secret Key

eggsited

Select a message digest algorithm

SHA1

COMPUTE HMAC

Computed HMAC (in Hex):

f4e075524ba4470867e1891c1a8d1fc21df1f56a

It is the same code like in the source code. We need only the yellow egg. So it should be ffff00 and the SHA1-HMAC from this hex-color. At line 26 is the URL we should use, to get the egg. So here is the final URL: <http://hackyeaster.hacking-lab.com/hackyeaster/egg?code=ffff00&key=eggsited&hmac=1da02c68080863fa302c20c3312371f4e365a5f9>

If we call the URL, we got the base64 encoded image. If you want to see the image, copy the whole base64 string and type in your browser: "data:image/png;base64,[base64data]". Replace "[base64data]" with your copied base64 string. And done:



Challenge 17 – Bunny Hop

Task

Wannabe programming guru Hazel B. Easterwood created a new programming language called "Bunny Hop". You suspect Hazel to have cheated, because the language looks very familiar to you.

Download the following code and complete it! It will yield the QR code for egg 17.

[Download the code](#)

Solution

After staring some minutes on the code I came up with an idea:

First there is a function:

```
window
to lineofeggs :cnt
  repeat cnt [egg hop 10 ]
  backhop 10
end
```

First the code creates a window. Second there is a function called "lineofeggs". What I think it does is, to print cnt times 10 pixel. "egg hop 10" means write a black pixel and jump to the next position. At the end of the function is a "backhop", which means the position of the bunny (pointer) goes back. Within the code are more commands. "egg" for writing a pixel. "left" and "right" have every time multiple of 90, so I think it is the rotation of the bunny (pointer). If the bunny starts from the upper left corner, looking to the right side and the command right 90 comes up, after this command the bunny should look down. With this assumptions I wrote an Interpreter:

```
from PIL import Image
import PIL

print "load bunny data"
with open('egg17.bunny') as f:
    content = f.read().splitlines()

#lineofeggs repeat x times: write pixel (egg) + go+1 in direction where you
look, after all go-1
#right rotate by x degrees to the right
#left rotate by x degrees to the left
#hop increase index by x in direction where you look

img = Image.new("RGB", (25,25), "white")#create image
pix = img.load()
#N,E,S,W = 0,1,2,3 (North, East, South, West)
look = 1 #first you looks east
x = 0
y = 0
```

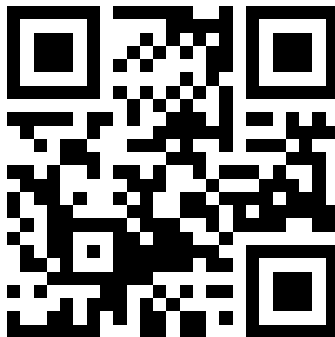
```

#function for go in the right direction
def go(x,y):
    if look == 0:
        y -= 1
    elif look == 1:
        x += 1
    elif look == 2:
        y += 1
    elif look == 3:
        x -= 1
    return x,y
print "printing Image..."
for cmd in content:
    cmddata = cmd.split(" ")
    if cmddata[0] == "lineofeggs":
        for i in xrange(0,int(cmddata[1])):
            pix[x,y] = (0,0,0)#egg command
            x,y = go(x,y)#hop 10
            look = (look + 2) % 4#look backwards
            x,y = go(x,y)#jump jump 1 (backjump)
            look = (look + 2) % 4#and look forward again
    elif cmddata[0] == "hop":
        for i in xrange(0,(int(cmddata[1])/10)):
            x,y = go(x,y)#jump x times
    elif cmddata[0] == "right":
        rotate = int(cmddata[1])/90
        look = (look + rotate) % 4#rotate right
    elif cmddata[0] == "left":
        rotate = int(cmddata[1])/90
        look = (look - rotate) % 4 #rotate left
    elif cmddata[0] == "egg":
        pix[x,y] = (0,0,0)#print black pixel
    else:
        print "unknown command: " + cmd#in case of unknown cmd

img = img.resize((100,100))#resize Image
img.save("egg17.png")#save image

```

And here are the result:



The next QR-Code, which is egg 17.

Challenge 18 – Bug Hunter

Task



Lacking of time, you were not able not complete your DeggCryptor program. In an act of desperation, you instructed Sammy, the junior programmer, to implement the missing key generation function.

As always, Sammy miserably failed. Can you fix his code? It's the *KeyGen* class. Pay attention to the comments!

Source Code

Solution

This time, we have a little C# project:

 DeggCryptor	12.05.2016 23:55	Dateiordner	
 DeggCryptor.sln	09.01.2016 15:41	SharpDevelop Proj...	1 KB

Let's fix the Source Code of the Keygen Class:

```
// Init the four seed values. 1111 and multiples of it.
int h1 = 0x1111;
int h2 = 0x2222;
int h3 = 0x3333;
int h4 = 0x4444;
```

Sammy! It said multiple of 1111!

```
// Init the four seed values. 1111 and multiples of it.
int h1 = 1111;
int h2 = 2222;
int h3 = 3333;
int h4 = 4444;
```

```
// 1'000 iterations.
for (int i = 1; i < 1000; i++)
{
```

Damn, Sammy 1000 iterations, not 999!

```
// 1'000 iterations.
for (int i = 1; i <= 1000; i++)
{
```

```
// If c is greater than d, double c and d.
if (c > d)
    c *= 2;
    d *= 2;
```

Sammy, I think you missed something here!

```

// If c is greater than d, double c and d.
if (c > d)
{
    c *= 2;
    d *= 2;
}

```

```

// Calculate new values.
// a: Take sum of a and b, and c and d. Then, multiply the two values.
a = a + b * c + d;

```

Sammy, I think you hate math, do you?

```

// Calculate new values.
// a: Take sum of a and b, and c and d. Then, multiply the two values.
a = (a + b) * (c + d);

```

```

// b: multiply with 3. Using two additions instead of multiplying -> performance boooooost!
b = b + b;
b = b + b;

```

Yeah Sammy, boost to the hell of math.

```

// b: multiply with 3. Using two additions instead of multiplying -> performance boooooost!
b *= 3;

```

```

// c, d: Swap c and d
c = d;
d = c;

```

Sammy, you don't have a clue from programming, do you?

```

// c, d: Swap c and d
int temp = c;
c = d;
d = temp;

```

```

// Take last four digits (modulo 10'000),
a %= 10000;
b %= 10000;
c %= 10000;
d %= 10000;

```

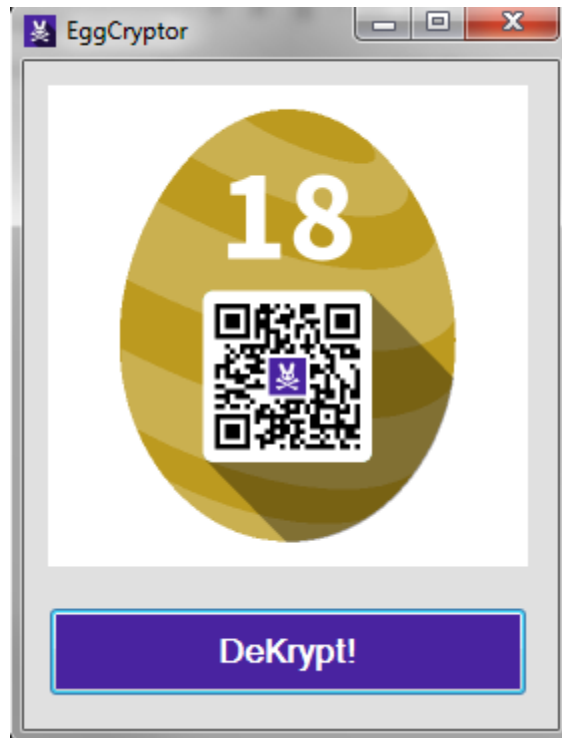
Sammy, how much coffee did you had today?

```

// Take last four digits (modulo 10'000),
a %= 10000;
b %= 10000;
c %= 10000;
d %= 10000;

```

That should be all mistakes. Sammy! F-grade!



Challenge 19 – Assemble This

Task

In this challenge you must crack a server-side program. Lucky for you, you got the assembly file of the program. First, reverse-engineer the program and find a valid code! Then, submit the code to the server.

The server is located at:

hackyeaster.hacking-lab.com:1234

Important: Do not launch brute-force attacks on the server - you'll not be lucky with it!

Download assembly file

Solution

We got a little assembly source code, which was generated right from a compiled c script. For better analyzing the assembly, I compiled it with gcc:

```
# mv assembly.txt assembly.s
# gcc -c assembly.s -o assembly.o
# gcc -o assembly assembly.o -lc
```

First I renamed the txt to .s so that I can compile the assembly-source-code with gcc. I found a [stackoverflow article](#) about this topic. After the compilation, I only need to link the assembly and I am able to debug the application.

For an easier analysis I load the assembly in IDA and looked at the decompiled source code:

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     int inputvar; // er903
4     _BYTE *bytevat; // r1003
5     signed int edii; // edi03
6     signed int calc4; // esi03
7     signed int calc3; // ecx03
8     signed int ebxx; // ebx03
9     signed int inputvar_1; // er903
10    int calc_add_char; // er809
11    int calc1; // er9011
12    int calc2; // edx011
13    int result; // eax014
14    FILE *secretfile; // rbx018
15    int getcontent; // eax021
16    char input; // [sp+0h] [bp-28h]01
17    _BYTE bytes[7]; // [sp+1h] [bp-27h]03
18
19    if ( !fgets(&input, 20, stdin) || strlen(&input) <= 16 )
20        goto LABEL_25;
21    inputvar = input;
22    bytevat = bytes;
23    edii = 0;
24    calc4 = 0;
25    calc3 = 0;
26    ebxx = 0;
27    inputvar_1 = input;
28    while ( 1 )
29    {
30        if ( edii == 3 * (edii / 3) )
31            calc3 += inputvar;
32        calc_add_char = calc4 + inputvar;
33        if ( (edii & 3) == 2 )
34            calc4 = calc_add_char;
35        ++edii;
36        calc1 = (unsigned __int8)((unsigned int)(inputvar_1 >> 0x1F) >> 24) + inputvar_1 // calc1 %0x100
37            - ((unsigned int)(inputvar_1 >> 0x1F) >> 24);
38        calc2 = (unsigned __int8)(ebxx + ((unsigned int)(ebxx >> 31) >> 24)) - ((unsigned int)(ebxx >> 31) >> 24); // calc2%0x100
39        calc3 = (unsigned __int8)((unsigned int)(calc3 >> 31) >> 24) + calc3 - ((unsigned int)(calc3 >> 31) >> 24); // calc3%0x100
40        calc4 = (unsigned __int8)((unsigned int)(calc4 >> 31) >> 24) + calc4 - ((unsigned int)(calc4 >> 31) >> 24); // calc4%0x100
41        if ( edii == 16 )
42            break;
43        inputvar = *bytevat; // INPUT char +1
44        ebxx = calc2 + inputvar;
45        inputvar_1 = inputvar + calc1;
46        if ( !(edii & 1) )
47            ebxx = calc2;
48        ++bytevat;
49    }
50    if ( calc1 == 0x85 && calc2 == 0x43 && calc3 == 0x4F && calc4 == 0xB0 )
51    {
52        usleep(5000000u);
53        secretfile = fopen("secret.txt", "r");
54        if ( secretfile )
55        {
56            while ( 1 )
57            {
58                getcontent = _IO_getc(secretfile);
```

I rewrote the source to python and simplify the whole code:

```
calc1 = 0
calc2tmp = 0
calc2=0
calc3 = 0
calc4 = 0
i=0

mystring = "123456789987654321"#test-input
inputvar = ord(mystring[0])
inputvar_1 = ord(mystring[0])

while 1:
    if(i == 3*(i/3)):
        calc3 += inputvar
    if (i&3==2):
        calc4 += inputvar
    i=i+1
    calc1 = inputvar_1%0x100
    calc2 = calc2tmp%0x100
    calc3 = calc3%0x100
    calc4 = calc4%0x100

    if i==16:
        break
    inputvar = ord(mystring[i])
    calc2tmp = calc2 + inputvar
    inputvar_1 = inputvar + calc1
    if (not(i&1)):
        calc2tmp = calc2
```

Seems a little bit complicated. I rewrote the code, so that I can solve it with z3:

```
from z3 import *

#input vector
s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15 = BitVecs('s0, s1,
s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13, s14, s15',32)

solver = Solver()
#only ASCII values from " " to "\""
solver.add(And(s0 >=32,s1 >=32,s2 >=32,s3 >=32,s4 >=32,s5 >=32,s6 >=32,s7
>=32,s8 >=32,s9 >=32,s10 >=32,s11 >=32,s12 >=32,s13 >=32,s14 >=32,s15
>=32))
solver.add(And(s0 < 126,s1 < 126,s2 < 126,s3 < 126,s4 < 126,s5 < 126,s6 <
126,s7 < 126,s8 < 126,s9 < 126,s10 < 126,s11 < 126,s12 < 126,s13 < 126,s14
< 126,s15 < 126))

#solver for calc1 to calc4
solver.add(0x85 == (s0 + s1 + s2 + s3 + s4 + s5 + s6 + s7 + s8 + s9 + s10 +
s11 + s12 + s13 + s14 + s15)%0x100)
solver.add(0x43 == (s1 + s3 + s5 + s7 + s9 + s11 + s13 + s15 )%0x100)
solver.add(0x4f == (s0 + s3 + s6 + s9 + s12 + s15 )%0x100)
solver.add(0xb0 == (s2 + s6 + s10 + s14 )%0x100)

if solver.check() == sat:
    m = solver.model()
    print m
else:
    print "nope"
```

After executing the script I got the following answer:

```
[s12, = 101,  
s11, = 122,  
s2, = 124,  
s1, = 125,  
s3, = 101,  
s9, = 96,  
s14, = 108,  
s0, = 67,  
s7, = 66,  
s13, = 103,  
s10, = 78,  
s8, = 124,  
s6, = 122,  
s15 = 104,  
s4, = 110,  
s5, = 118]
```

or in other words "ida.lo\es.you". This is the password for the Egg-O-Matic:



Challenge 20 – Humpty’s Dump

Task

Humpty's Dump

You got hold of a dump of Humpty Dumpty's secret egg database.

Search and extract the egg hidden in the dump!

Hints

- The 'puzz' is not more than 8 chars, letters only.
- The decryption of the file can be done using AES_DECRYPT().

[Download the dump](#)

Solution

In this challenge we got a MySQL Dump. For better work I imported the dump in the following order to a MySQL database:

1. humpty_routines.sql
2. humpty_uzr.sql
3. humpty_kee.sql
4. humpty_fyle.sql

After this I analyzed a little bit deeper the routines:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetPuzzMishMash` (  
  IN  p_puzz  VARCHAR(40) ,  
  IN  p_sawlt  VARCHAR(4) ,  
  OUT p_mishmash  VARCHAR(40)  
)  
BEGIN  
  DECLARE p_tmp VARCHAR(50) ;  
  SET p_tmp := CONCAT(p_sawlt, '.', p_puzz, '.', p_sawlt) ;  
  SELECT SHA1(p_tmp) INTO p_mishmash ;  
END
```

That’s the function for the hash in the uzr-table. SHA1(salt.pwd.salt)! The puzz is also letters only and 8 chars are bruteforcable. So I tried to bruteforce the hashes with cudaHashcat.

Content of 2016hackyeaster.txt:

```
943f9ecbbd91306a561d0e3c15e18ee700007083:abcd  
915d253cb5ba6f0a220bca83e2d6d3258af15e68:nmlk  
1742ae4507fc480958e2437104e677e70aa5e857:jkln  
0cf32f8f418659f23f8968d4f63ea5c98b39f833:zyxw  
de2278f5bcafcbb097ecc1fb54e5ab8a9e912c55:efgh
```

Content of mask.txt

```
.?1.  
.?1?1.
```

```
.?1?1?1.  
.?1?1?1?1.  
.?1?1?1?1?1.  
.?1?1?1?1?1?1.  
.?1?1?1?1?1?1?1.  
.?1?1?1?1?1?1?1?1.  
.?1?1?1?1?1?1?1?1?1.
```

And with the following command I am able to crack the hash:

```
cudaHashcat64.exe -a 3 -m 4900 2016hackyeaster.txt -1 ?1?u mask.txt
```

With that I test all passwords from a-zA-Z with a length from 1 to 8 and they are beginning with a dot in the beginning and in the end. After a while I got the following result:

```
0cf32f8f418659f23f8968d4f63ea5c98b39f833:zyxw:.snakeoil.
```

So our puzz is “snakeoil”. With that we can recover the key with the DeekryptKee function. With the following MySQL-Query you get the Kee:

```
call  
DeekryptKee('snakeoil','1ABF4B7CD25C61FDF0E74EC2BFB43BD1C2D8ECD803AFA3AA376  
F4C0000052813', @KEE); select @KEE;
```

I got for the key the value “jpP8HeoEC5OCCBqdf9N3”. Now we can decrypt the file from the database and save it. I used the following MySQL Query for that:

```
select aes_decrypt((select blahb from fyle where keeid=2332),  
'jpP8HeoEC5OCCBqdf9N3') into dumpfile 'egg.png'
```

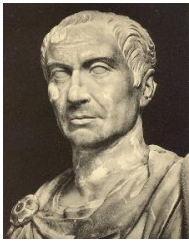
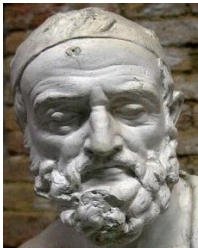


In my MySQL Database folder I found the image:



Challenge 21 – Crypto Council

Task

Clever crypto brains had a little get-together. Find out who they are and break the riddles they created! Each plain text contains a password - once you've got them all, enter them in the Egg-O-Matic below. **Lowercase** only.

	<p>DV D UXOH PHQ ZRUUB PRUH DERXW ZKDW WKHB FDQW VHH WKDQ DERXW ZKDW WKHB FDQ SDVVZRUG LV FDUWKDJR</p>
	<p>4423154215 2443 3334 52244433154343 4334 1442151114214531 3334 11131345431542 4334 4415424224123115 1143 442315 13343343132415331315 44231144 145215313143 2433 442315 2315114244 3421 1551154254 321133 3515313435343333154315 2443 442315 3511434352344214</p>
	<p>EHIKT YFC FTEU QK PLTPWBY MQYTNVZW LAJ JGGN ZVLD A EWTAE WIELP QF IHV DAALROW DF JIAC T GWMGCRQF WIJ NSIHVZ BTAE IJGAEOWS FFZ ZXM KW ZPVV I UAAJAARAC MVJCRBADN ZV HPRZA TAAZAW SE MQYTNVZW HTLLATD XZWTK RVV WESZWL UELWG AUZAPNLA LJREMTJS RVV YERV VDRRB SI TYM SVE FN SVE JMNTNKMWC HV MFIEIMV IHV LAELFUSIIT AWGVZKW PNU ZWBAZVWS TYMJT FFZ LWIIBQ NERZK UIMM QTAIA ACTF PAH CRZWTR YM SRCFUHPNZMV IHV NJTNTP WCVFG DDUZA SSHVUSG DV OJXGEIF IO KPW SIVB GU WFZEH AJ I BJNZWJ HETZWIAIG ZT EEBWGEU BZT SVZNXCW WX IHV LMZE FN FTVVZK PS YQK HETZWIAIG S EOJQLXOE PW WECL MCTZT LWE UMSIHJ WX IHV LMZE RVV WIJ AGC HV IDHO JMJKEU IK P SVKJTTRZQ IO YMFGY ZQA</p>
	<p>WEN XQWVIBQZ KGQEAL TWB WEH GKQCW QLTBAKBTU LKIQTME DWCOAKWZAKNB BKMETP WEW NOTHPA HWB GBXHCEWG IA OTTQPWD SEATWCWNBA NZW HTO BHQWG HIWAL MCMG LQWVIBQL TOF QLFVCHWG WEW XNW FM WET NGQEAL QWBDSCMF KO CPWCKWMAX</p>

Solutions

The first picture is Caesar. So a quick Caesar bruteforce attack reveals it is ROT23. It gives us the following text:

AS A RULE MEN WORRY MORE ABOUT WHAT THEY CANT SEE THAN ABOUT WHAT
THEY CAN PASSWORD IS **CARTHAGO**

The second picture shows Polybios. I did a manual analysis of the Polybios-Crypto and got the following results:

	1	2	3	4	5
1	A	B	C	D	E
2	F	_	_	I	_
3	L	M	N	O	P
4	_	R	S	T	U
5	V	W	X	Y	_

there is no witness so dreadful no accuser so terrible as the
conscience that dwells in the heart of every man **peloponnese** is the
password

The third picture shows Vigenère. Because I am lazy to do it manually, I bruteforced it online:

<http://www.mygeocachingprofile.com/codebreaker.vigenerecipher.aspx>

After looking in the messages, I found the right key at #38, which is “parisparis”. The deciphered text ist:

phrase you need is **alchemy** vigenere was born into a noble family in
the village of saint pourcain his father jean arranged for him to
have a classical education in paris blaise de vigenere studied greek
and hebrew under adrianus turnebus and jean dorat at the age of age
seventeen he entered the diplomatic service and remained there for
thirty years five years into his career he accompanied the french
envoy louis adhemar de grignan to the diet of worms as a junior
secretary he entered the service of the duke of nevers as his
secretary a position he held until the deaths of the duke and his
son he also served as a secretary to henry iii

The last picture shows Playfair. This was little bit challenging, but after I read the history on
<http://www.crosswordman.com/cgi-bin/playfair> I tried the key “wheatstone”. And indeed it worked!

THE PLAYFAIR CIPHER WAS THE FIRST PRACTICAL DIGRAPH SUBSTITUTION
CIPHER THE SCHEME WAS INVENTED BY CHARLES WHEATSTONE BUT WAS NAMED
AFTER LORD PLAYFAIR WHO PROMOTED THE USE OF THE CIPHER PASSWORD IS
BLETCHLEY

With all bold words from the deciphered texts, you are able to get the egg:



Challenge 22 – Dumpster Diving

Task

Dumpster Diving

You've sniffed some password hashes of a web site:

hash 1: fad202a6e094dd8f1d63da8bdf85b3ba099971d3

hash 2: f71e1b0b9b3a57d864c2e9f7bd6dd90f66b5a7d6

hash 3: 84c6bcb681b79b690b53f9f3a8ba24e1e970d348

hash 4: 0d6bb0df8918168798ce6b770014aeb81ac6ce76

However, none of your tools succeeded in cracking the hashes. As a last resort, you inspected the dumpster of the software development company which created the web site. And indeed you found something: a paper with a part of the hash calculation code.

Can you crack the hashes now?

```
h0 = 0x10325476
h1 = 0x98BADCFE
h2 = 0xEFCDA8B9
h3 = 0x67452301
h4 = 0x0F1E2D3C

bytelen = len(m)
bitlen = 8 * bytelen
m += b'\x80'
m += b'\x00' * ((56 - (bytelen + 1) % 64) % 64)
m += struct.pack(b'>Q', bitlen)

for i in range(0, len(m), 64):
    w = [0] * 80
    for j in range(16):
        w[j] = struct.unpack(b'>I', m[i + 4*j:i + 4*j + 4])[0]
    for j in range(16, 80):
        w[j] = rotate_left(w[j-3] ^ w[j-8] ^ w[j-14] ^ w[j-16], 1)

    a = h0
    b = h1
    c = h2
    d = h3
    e = h4

    for i in range(80):
        if 0 <= i <= 19:
            f = d ^ (b & (c ^ d))
            k = 0x5A827999
        elif 20 <= i <= 39:
            f = b ^ c ^ d
            k = 0x6ED9EBA1
        elif 40 <= i <= 59:
```

Solution

I researched some parts of the algorithm on google. It reveals it is SHA1, but h0-h1 are the magic values are different. So I grapped the source code of oclHashcat and changed the "include/constants.h" at line 62:

```
#define SHA1M_A 0x10325476u
#define SHA1M_B 0x98badcfu
#define SHA1M_C 0xefcdab89u
#define SHA1M_D 0x67452301u
#define SHA1M_E 0x0f1e2d3cu
```

After this I compiled the source code and bruteforced with the following command:

```
oclHashcat64.exe -m 100 hashes_2016.txt -r rules\T0X1C.rule
E:\dict\eNtrOpY_ALL_sort_uniq.dic
```

You find [eNtrOpY ALL sort uniq.dic](#) in an interesting Article on the internet ☺

```
f71e1b0b9b3a57d864c2e9f7bd6dd90f66b5a7d6:Denver1
fad202a6e094dd8f1d63da8bdf85b3ba099971d3:zombie
84c6bcb681b79b690b53f9f3a8ba24e1e970d348:Placebo
0d6bb0df8918168798ce6b770014aeb81ac6ce76:SHADOWLAND

Session.Name...: oclHashcat
Status.....: Cracked
Rules.Type....: File (rules\T0X1C.rule)
Input.Mode....: File (E:\d\crackstation.txt\eNtrOpY_ALL_sort_uniq.dic)
Hash.Target...: File (hashes_2016.txt)
Hash.Type....: SHA1
Time.Started...: Fri Apr 15 11:37:06 2016 (6 mins, 31 secs)
Speed.Dev.#1...: 298.0 MH/s (11.99ms)
Recovered.....: 4/4 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.....: 119383502558/341808487020 (34.93%)
Rejected.....: 118494/119383502558 (0.00%)
Restore.Point...: 29216797/83653570 (34.93%)
HWMon.GPU.#1...: 2% Util, 66c Temp, 58% Fan

Started: Fri Apr 15 11:37:06 2016
Stopped: Fri Apr 15 11:43:39 2016
```

After 6 Minutes everything was cracked ;)



Challenge 23 - Heizohack

Task

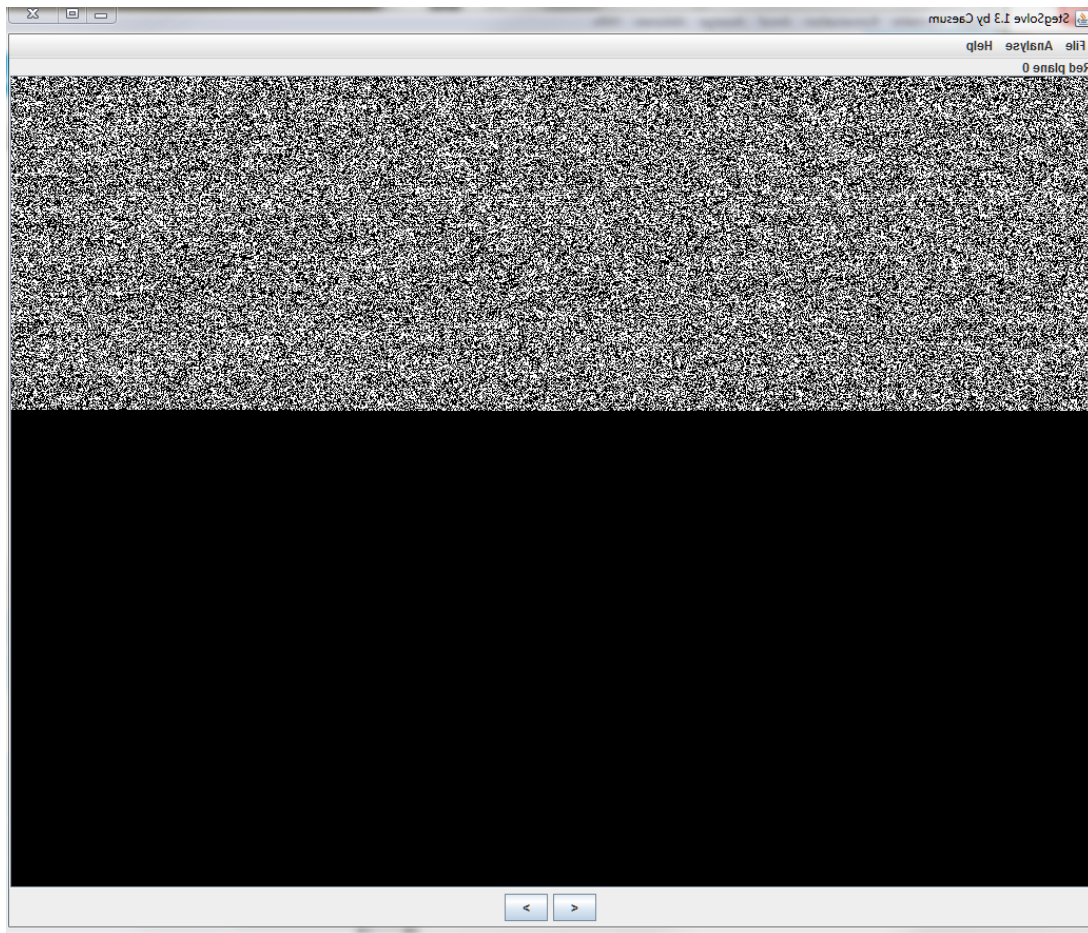
Can you crack the Heizohack?

The password for the Egg-O-Matic is hidden in the image.

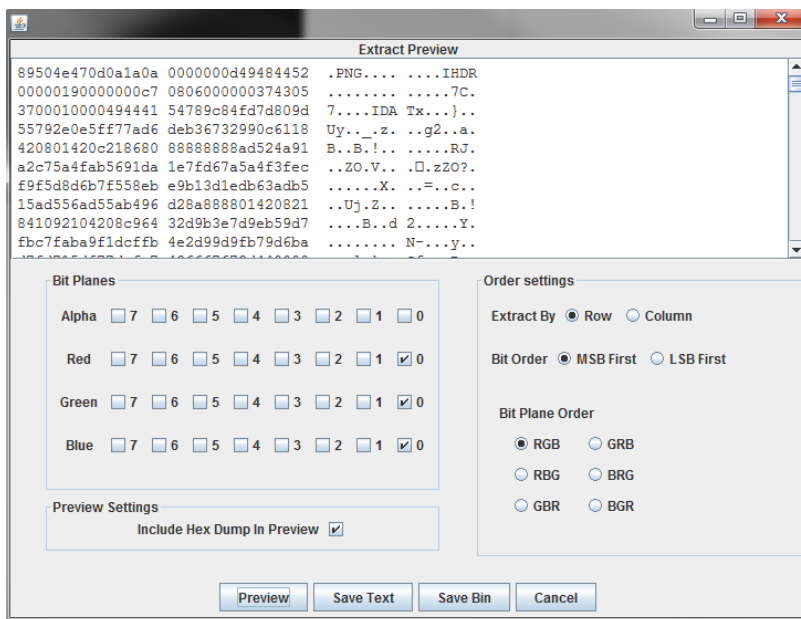


Solution

Another steganography, another time for StegSolve! The most interesting parts are the Red 0, Blue 0 and Green 0 Channels. They mostly look like the same. Here is the Red 0 channel for example:



So I tried the Data Extract function from StegSolve:



After I saved the binary, I got the following Image:



In Stegsolve I noticed there are some differences at the top 3 pixel lines:



Another hint is within the image. Bit:r, MAC: $r \oplus g \oplus b \oplus a == 1$. So I wrote a little script, which shows me every pixel with the condition " $r \oplus g \oplus b \oplus a == 1$ ":

```
from PIL import Image

img = Image.open("r0g0b0_trim.png")
pix = img.load()
x,y = img.size

for bbb in xrange(0,3):
    for aaa in xrange(0,x):
        r,g,b,a = pix[aaa,bbb]
        vxor = r^g^b^a
        if (r^g^b^a)&1 == 1:
            print pix[aaa,bbb]
```

```

root@kali:~/Desktop
(22, 22, 23, 254)
(23, 23, 22, 255)
(23, 22, 22, 254)
(22, 22, 22, 255)
(23, 23, 23, 254)
(23, 22, 23, 255)
(22, 23, 22, 254)
(22, 22, 23, 254)
(22, 23, 23, 255)
(23, 22, 23, 255)
(23, 22, 23, 255)
(23, 22, 23, 255)

```

If you look at the red channel you noticed something familiar. If you replace 22 with a 0 and 23 with an 1 you get a binary ASCII code! So I modified my script:

```

from PIL import Image

img = Image.open("r0g0b0_trim.png")
pix = img.load()
x,y = img.size

solution=""

for bbb in xrange(0,y):
    for aaa in xrange(0,x):
        r,g,b,a = pix[aaa,bbb ]
        if (r^g^b^a)&1 == 1:
            #print pix[aaa,bbb ]
            if r == 22:
                solution += "0"
            if r == 23:
                solution += "1"

decode = ""
for i in xrange(0,len(solution)/8):
    decode += chr(int(solution[(i*8):(i*8+8)],2))
print decode

```

```
lostinthewoods
```

And we got the next egg ☺



Challenge 24 – crunch.ly

Task

crunch.ly

Do you know *crunch.ly*, the fancy new URL shortener? It was used to create a short URL for Hacky Easter. In order to lure people onto the web site of your alternative hacking competition "Evil Easter", you decide to attack this service.

What you know

- Short URL for Hacky Easter: **<http://crunch.ly/IU66SMI>**
- Web site of crunch.ly (not a real domain!): [OPEN WEB SITE](#)
- Algorithms used on the web site: [DOWNLOAD](#)

Your mission

1. find a URL **starting with <http://evileaster.com>**, which produces the same short URL
2. make the web site store your URL, instead of the original URL
3. open the short URL on the web site
4. do not bomb or DoS the server - you'll have no luck with it; cracking must happen offline

Solution

We got both functions, how Short URL got calculated and how Tickets are created. So at the first part we need to recover the key. The full Key is 128 bit, so the normal key $(128-1)/3=42\text{bit!}$ One Character have 8bit, so we search for a key which is 5 chars long. This should be an easy task. First I tampered the whole process of creating and saving a short URL. I got the following data with that:

```
url = "http://asdf.de"
shorturl = "TNUJJLQ"
encrypted_ticket =
"fUbbAKUKBsUgSFwl3C5ItfNjJPFOYfOVucpifACVzWB4PC+SXpSt/rwoMDEu7p8da4aJ9Jr001
wSqJ/FzCKHig=="
```

With that knowledge I am able to bruteforce the key. I re-implemented the cryptTicket function and wrapped a bruteforce attack around that function:

```
import hashlib
import base64
from Crypto import Random
from Crypto.Cipher import AES

def pad(s):
    return s + b"\0" * (AES.block_size - len(s) % AES.block_size)
```

```

def encrypt(message, key, key_size=128):
    message = pad(message)
    iv = "hackyeasterisfun"
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return cipher.encrypt(message)

def decrypt(ciphertext, key):
    iv = "hackyeasterisfun"
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = cipher.decrypt(ciphertext)
    return plaintext.rstrip(b"\0")

iv = "hackyeasterisfun"
bs = "abcdefghijklmnopqrstuvwxyz012345679ABCDEFGHIJKLMOPQRSTUVWXYZ"
url = "http://asdf.de"
shorturl = "TNUJJLQ"
encrypted_ticket =
"fUbbAKUKBsUgSFwl3C5ItfNjJPFOYfOVucpifACVzWB4PC+SXpSt/rwoMDEu7p8da4aJ9Jr001
wSqJ/FzCKHig=="

encdata = encrypt("test", KEY_FULLL)
#print base64.b64encode(encdata)
print decrypt(encdata, KEY_FULLL)

for a in bs:
    for b in bs:
        for c in bs:
            for d in bs:
                for e in bs:
                    KEY = a + b + c+d+e
                    #print KEY
                    KEY_FULLL = "x"+KEY+KEY+KEY

                    plain = base64.b64encode(url)
                    plain += "@" + base64.b64encode(shorturl)

                    decoded = base64.b64decode(encrypted_ticket)
                    txt = decrypt(decoded, KEY_FULLL)
                    #print txt
                    if txt[:5] == plain[:5]:
                        print KEY

```

Now we know that the key is "tKguF". The next step is, that we need an URL that starts with "http://evileaster.com" and the short URL is IU66SMI. After analyzing the calculateShortUrl function, I noticed that IU66SMI is base64 and is the beginning of the sha256(url). For better understanding:

IU66SMI== is in hex **453de931**

Sha256("http://hackyeaster.hacking-lab.com") is
453de9316a0d3ae749261bb891930b4561d4a99ee9eb462548bf0f868b079957

With that knowledge I was able to write another bruteforce script, to get the right "http://evileaster.com" URL shortened. My approach was to append a numeric parameter like "http://evileaster.com/?1234".


```

import hashlib
import base64

shorturl = "IU66SMI="
hexcode = base64.b32decode(shorturl).encode('hex')
pre = "http://evileaster.com/?"

for i in xrange(0,999999999999999):
    after = str(i)
    goal = hashlib.sha256(pre+after).hexdigest()[:8]
    if hexcode == goal:
        print pre + after

```

So I started the script and went to bed. At the next morning I got the following URLs

```

http://evileaster.com/?10691651141
http://evileaster.com/?12905523265
http://evileaster.com/?15399599367
http://evileaster.com/?17899621795

```

The right injection is within the saving process. With the following script I created my evileaster ticket:

```

import hashlib
import base64
from Crypto import Random
from Crypto.Cipher import AES

def pad(s):
    return s + b"\0" * (AES.block_size - len(s) % AES.block_size)

def encrypt(message, key, key_size=128):
    message = pad(message)
    iv = "hackyeasterisfun"
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return cipher.encrypt(message)

def decrypt(ciphertext, key):
    iv = "hackyeasterisfun"
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = cipher.decrypt(ciphertext)
    return plaintext.rstrip(b"\0")

iv = "hackyeasterisfun"
KEY = "tKguF"
KEY_FULL = "x"+KEY+KEY+KEY
url = "http://evileaster.com/?10691651141"
#generate shorturl from url
shorturl_test = hashlib.sha256(url).hexdigest()[:8]
shorturl = "http://crunch.ly/" +
base64.b32encode(shorturl_test.decode('hex'))
shorturl = shorturl.replace("=", "")

#build ticketformat --> the last chars are padding for %16==0
#are there missing
plain = base64.b64encode(url)
plain += "@" + base64.b64encode(shorturl)
+"\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f"

```

```

encdata64 = base64.b64encode(encrypt(plain, KEY_FULL))
encdata64 = encdata64[:128]
encdata64 = encdata64.replace("+", "%2B")#manual html encoding
encdata64 = encdata64.replace("/", "%2F")#manual html encoding

print encdata64 #print ticket

```

After getting the evileaster ticket I called the saving URL with the ticket:

<http://hackyeaster.hacking-lab.com/hackyeaster/crunch?service=save&ticket=Hgo3UsPWbH%2B4kkfQwZ0dOEWzKmGC5YiDB%2BLWRUtYh7c4Whym7tZRF6AgkXVSdFgRrG59VNbpwGyuMI8pSI3MKg1BCqakj3kECSI2gmBRey4fMOaSVSlcl2WfztwfBCK1>

I get a status code 0, which means the evil short URL is saved. Now I can go to the short URL >:D :

<http://hackyeaster.hacking-lab.com/hackyeaster/crunch?service=go&shorturl=http%3A%2F%2Fcrunch.ly%2FIU66SMI>

```

{
  "status": ?0,
  "url":
"http%3A%2F%2Fhackyeaster%2Ehacking%2Dlab%2Ecom%2Fhackyeaster%2Fimages%2Fegg24%5FbHlrQh1VR141TPmapETM%2Epng"
}

```

And the final URL to the last egg is http://hackyeaster.hacking-lab.com/hackyeaster/images/egg24_bHlrQh1VR141TPmapETM.png



Web-Submission Bomb

After I saw, that there is possibility to submit eggs on the website, my first idea was: "Submit all egg's at 13:37". For that mission I wrote a little python script. The first step was to read all QR-Codes:

```
from qrttools import QR

for i in xrange(1,25):

    qr = QR(filename="egg/egg"+format(i, "02d")+".png")
    if qr.decode():
        print str(i)+ " : " + qr.data
```

The Output was every QR-Code. After this I created a script for the time bomb and submitting all eggs at 13:37:

```
import requests
import time

sEggs = ["1fQArOSgpdSCBr8zXove",#01
        "P2kVqrkD2ykiNcwWAAKv",#02
        "FUYEMZoOCLis7tPvs5pq",#03
        "dLYFunszMowTRHovQx2y",#04
        "X8pfJTkxJeRX0bMHZsTP",#05
        "mGMzyuWnqVnC014Irrq5S",#06
        "3OgwkgZmzaFUvPBwnKr8",#07
        "B6gZYdONrcSnRlDlvQ03",#08
        "vgOObZR6VjxuwRkBVm2F",#09
        "QOXm6kvOYt3ATkc9rDnk",#10
        "wjlKxyRNWcLnyhdXhHCV",#11
        "kR4ZCgRneYR27YAYr8eE",#12
        "Jvl8olPUI9yfuJDWIJFi",#13
        "Qa5miycTGMkfXUeliOeJ",#14
        "JMV1xX8LZGM0VspECD1b",#15
        "7dUDQDhMQkLYsQTMJq62",#16
        "nYa3ktAoTAQc6yxMAGoM",#17
        "83OHadUPAeWRfd6YBv6t",#18
        "WCeZB8yUTdgjayQol2KS",#19
        "PRKuX3CklkoZWwfOHbpK",#20
        "cH3zySmRf29wCQpE7FSK",#21
        "Jbpfr31iCjbpThSfHk6i",#22
        "mwBDDBrer7qemsD7RDSf",#23
        "avHJJ56JeUvZ8fr7wkGB"#24
    ]

ticket = "11280ae602cd6588086549e07edb3849e4ac1b502945791f8d2b4e8ab69504e8"
name = "TheVamp"
url = "http://hackyeaster.hacking-
lab.com/hackyeaster/json?service=solution"
cookies = dict(JSESSIONID="6C1600800B538C57CBB2E4E2EA959550")
#time bomb
while 1:
    t = time.localtime(time.time())
    if t.tm_hour >= 13 and t.tm_min>=37 and t.tm_sec>=01:
        print "start" + str(t)
        break
```

```
print "lets go"
i=24
while 1:
    if i >0:
        print "code "+str(i)+" : "+ sEggs[i-1]
        postdata = {"code":sEggs[i-1], "name":name, "ticket":ticket }
        r = requests.post(url, data=postdata)
        print r.text
        i = i-1
        time.sleep(0.65)
    else:
        break
```

Happy Easter was a lot of fun. Thx for this journey ☺ see you next year!