HACKY EASTER 2016 WRITEUP BY HARDLOCK

INDEX

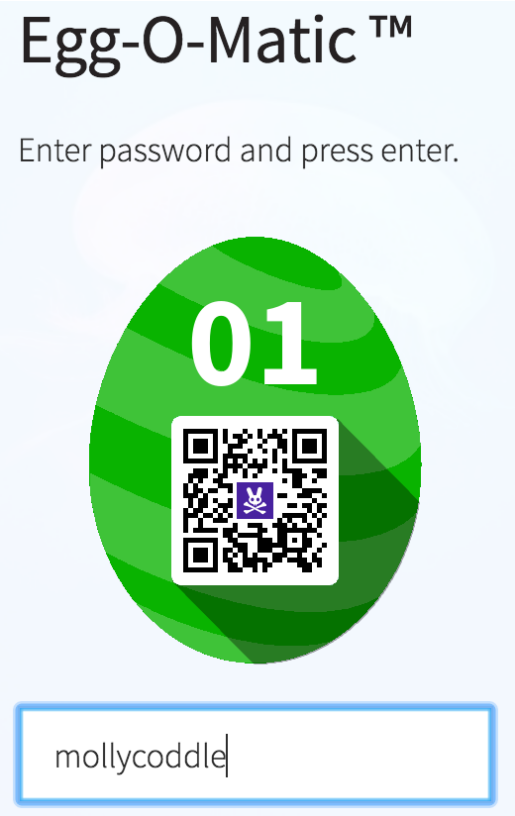
CHALLENGE 01 - EASY ONE

this challenge should be an easy starter, but actually it wasnt the first one which i solved. like always i overcomplicated it and didnt get it at first sight.

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

it looks like crypto, but its not. removing the spaces between the words, will show the solution rather quickly:

xthexyhiddenyystringyyisyymollycoddlexysoxxsimpley

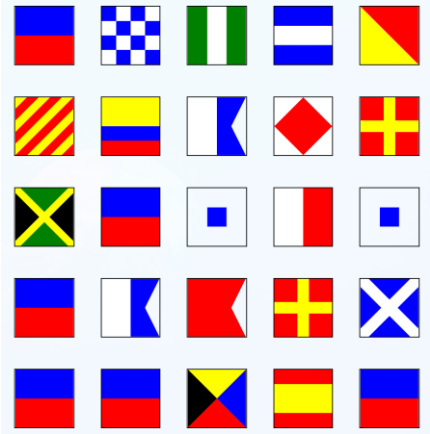


so simple - really.

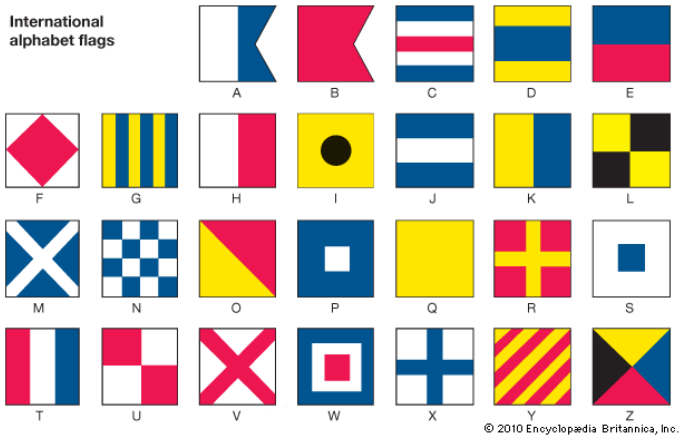
CHALLENGE 02 - JUST CRUISIN'

the second challenge gives us an image puzzle. first i didnt get the idea, but after googling some of these signs, i have found the signal flags used on ships and boats:

https://en.wikipedia.org/wiki/International_maritime_signal_flags



using a lookup table, i was able to find the solution.



i had to skip the unknown flags and the "m":

CHALLENGE 03 - BIRD'S NEST

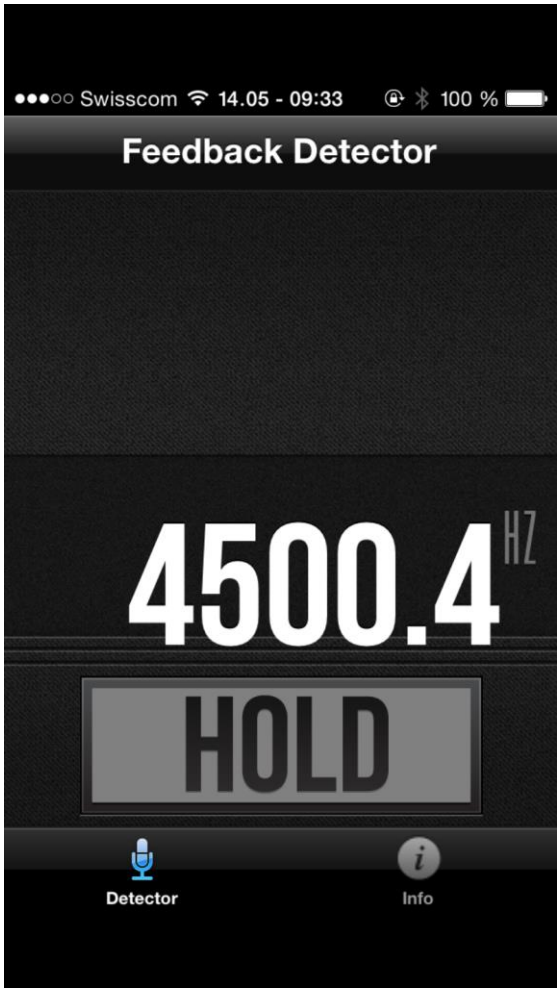
i was stuck at this challenge quite some time, because it is a mobile challenge and i was thinking that i have to do something with it. i decompiled the app and tried steganography on it, checked all html sources, but no luck.

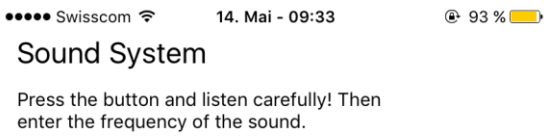
actually the # (hash sign) is the hint in this image. i first interpreted it as a number, but on the internet this is also known as hash tag. oh wait... lets check if we can find something using these words as hashtag. looking on twitter for posts containing "egg03" and "nest" will lead us to this post:

following the url from that post will give us the egg03:

the mobile app lets us play some sounds and obviously we have to find out which frequency they have and we should select the proper number from the dropdown. to solve this, i have used two iphones and the "feedback detector" app:

https://itunes.apple.com/us/app/feedback-detector/id560799394?mt=8



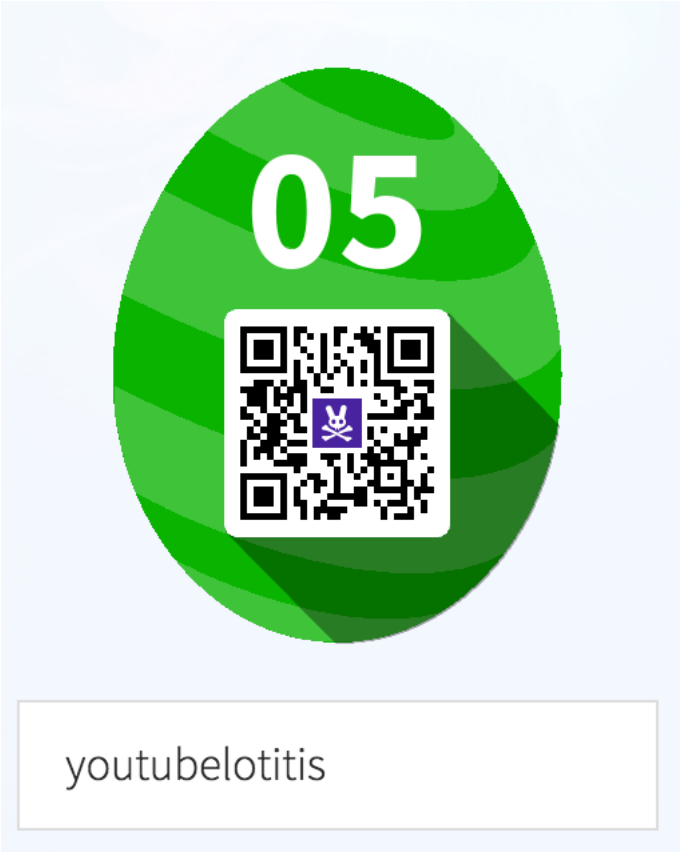


having a second phone was really helpful in this case, also to scan the QR code in the end.

CHALLENGE 05 - PLAY IT AGAIN, PAUL

this challenge offers us a youtube video. as a hint we have a phrase in a different language, which turned out to be "esperanto". we can change the languages for the video file (with the CC button) and if we set it to esperanto and watch it until the end, the passphrase will appear for us:

oh that was easy:



CHALLENGE 06 - GOING UP

this challenge looks like an elevator panel and every level has a link behind the button.

but there is one suspicious button (13)

challenge06.html?sybbe=punatrzr

looks like a simple cipher... lets try rot13 (because of the floor 13)

sybbe=punatrzr

This is your encoded or decoded text:

floor=changeme

ok, so changeme should obviously be thirteen:

floor=thirteen

This is your encoded or decoded text:

sybbe=guvegrra

http://hackyeaster.hacking-lab.com/hackyeaster/challenge06.html?sybbe=guvegrra

will lead us to the egg for challenge 06:

wise rabbit returns and tells us:

Wise Rabbit Once More

Wise Rabbit says:

The solution is in the solutions!

Go back and scroll to 123!

this is obviously a reference to the last years solutions which can be found here:

https://www.hacking-lab.com/references/hackyeaster2015/

opening the solutions PDF (http://media.hacking-lab.com/hackyeaster/HackyEaster2015_Solutions_high.pdf) and scrolling to the end will give us the password for this challenge:

```
#crack
while read password; do
        #truecrypt teaser_video.mp4  --password=$password
        echo $password;
        truecrypt -t teaser_video.mp4  --password=$password --non-interactive
        STATUS=$?
        if [ $STATUS -eq 0 ]; then
            clear;
                echo "----------------------";
                echo "Passwort gefunden : ";
            echo $password;
                break;
        fi
done <text_neu2.txt
```

password: goldfish

Hack Easter 2015 Page 123

Enter password and press enter.

goldfish

CHALLENGE 08 - JUST DRIVE

first i didnt get this challenge at all and therefore i had a look at the disassembled binary from my iphone. i dumped the HackyEaster binary with Clutch (only works on a jailbroken phone) and then loaded it in IDA.

```
if ( !((unsigned int)objc_msgSend(v3, "hasPrefix:", v6) & 0xFF) )
  v3 = (id)objc_msgSend(v6, "stringByAppendingString:", v3);
if ( (unsigned int)objc_msgSend(v3, "length") >= 9 )
  v3 = (id)objc_msgSend(v3, "substringToIndex:", 8);
v7 = (__CFString *)objc_msgSend(&OBJC_CLASS___Util, "sha1hex:", v3);
if ( !((unsigned int)objc_msgSend(v7, "hasPrefix:", CFSTR("4692bd56dd3070f74b7e")) & 0xFF) )
{
  if ( (unsigned int)objc_msgSend(v3, "length") >= 7
    && (v8 = 7,
        v9 = objc_msgSend(v3, "substringToIndex:", 7),
        v10 = objc_msgSend(&OBJC_CLASS___Util, "sha1hex:", v9),
        (unsigned int)objc_msgSend(CFSTR("1ab9a97066f747c25d9c6a6b0fda647fae98cb98"), "isEqualToString:", v10) & 0xFF)
    || (unsigned int)objc_msgSend(v3, "length") >= 6
    && (v8 = 6,
        v11 = objc_msgSend(v3, "substringToIndex:", 6),
        v12 = objc_msgSend(&OBJC_CLASS___Util, "sha1hex:", v11),
        (unsigned int)objc_msgSend(CFSTR("58192f7d1263bd420efb788cc884a84f871239cf"), "isEqualToString:", v12) & 0xFF)
    || (unsigned int)objc_msgSend(v3, "length") >= 5
    && (v8 = 5,
        v13 = objc_msgSend(v3, "substringToIndex:", 5),
        v14 = objc_msgSend(&OBJC_CLASS___Util, "sha1hex:", v13),
        (unsigned int)objc_msgSend(CFSTR("e8f235e79be9f8b13598b285a6fdaf2ac70a66ca"), "isEqualToString:", v14) & 0xFF)
    || (unsigned int)objc_msgSend(v3, "length") >= 4
    && (v8 = 4,
        v15 = objc_msgSend(v3, "substringToIndex:", 4),
        v16 = objc_msgSend(&OBJC_CLASS___Util, "sha1hex:", v15),
        (unsigned int)objc_msgSend(CFSTR("3daee0c0bada99f5bf0866728fd78299acaafa15"), "isEqualToString:", v16) & 0xFF)
    || (unsigned int)objc_msgSend(v3, "length") >= 3
    && (v8 = 3,
        v17 = objc_msgSend(v3, "substringToIndex:", 3),
        v18 = objc_msgSend(&OBJC_CLASS___Util, "sha1hex:", v17),
        (unsigned int)objc_msgSend(CFSTR("f11fa81c0b72716bba0536dd34b9b0987af69b03"), "isEqualToString:", v18) & 0xFF)
    || (unsigned int)objc_msgSend(v3, "length") >= 2
    && (v8 = 2,
        v19 = objc_msgSend(v3, "substringToIndex:", 2),
        v20 = objc_msgSend(&OBJC_CLASS___Util, "sha1hex:", v19),
        (unsigned int)objc_msgSend(CFSTR("a8643e0e26d5ead82e73aae64966ca144f152d8a"), "isEqualToString:", v20) & 0xFF) )
  {
    v7 = CFSTR("");
    goto LABEL_30;
  }
  if ( objc_msgSend(v3, "length") )
  {
    v21 = objc_msgSend(v3, "substringToIndex:", 1);
    v22 = objc_msgSend(&OBJC_CLASS___Util, "sha1hex:", v21);
    v7 = CFSTR("");
    v8 = objc_msgSend(CFSTR("4dc7c9ec434ed06502767136789763ec11d2c4b7"), "isEqualToString:", v22) != 0;
    goto LABEL_30;
  }
```

there is some hash checking going on based on the orientation of the phone. lets see if we can find what these hashes mean. for that i have used https://hashkiller.co.uk/md5-decrypter.aspx

```
We found 7 hashes! [Timer: 222 m/s] Please find them below...

4dc7c9ec434ed06502767136789763ec11d2c4b7
a8643e0e26d5ead82e73aae64966ca144f152d8a
f11fa81c0b72716bba0536dd34b9b0987af69b03
3daee0c0bada99f5bf0866728fd78299acaafa15
e8f235e79be9f8b13598b285a6fdaf2ac70a66ca
58192f7d1263bd420efb788cc884a84f871239cf
1ab9a97066f747c25d9c6a6b0fda647fae98cb98

4dc7c9ec434ed06502767136789763ec11d2c4b7 SHA1 : r
a8643e0e26d5ead82e73aae64966ca144f152d8a SHA1 : nr
f11fa81c0b72716bba0536dd34b9b0987af69b03 SHA1 : rnr
3daee0c0bada99f5bf0866728fd78299acaafa15 SHA1 : xrnr
e8f235e79be9f8b13598b285a6fdaf2ac70a66ca SHA1 : rxrnr
58192f7d1263bd420efb788cc884a84f871239cf SHA1 : xrxrnr
1ab9a97066f747c25d9c6a6b0fda647fae98cb98 SHA1 : rxrxrnr
```

interesting. one hash is not complete in the disassembly, but we obviously need exactly that one.

```
if ( !((unsigned int)objc_msgSend(v7, "hasPrefix:", CFSTR("4692bd56dd3070f74b7e")) & 0xFF) )
```

by trial and error i discovered the correct string (nrxrxrnr):

SHA1 and other hash functions online generator

nrxrxrnr [hash]

sha-1

Result for sha1: 4692bd56dd3070f74b7e81c6b2f69339b0fd6062

then i used this full hash as json.k in the html file and replaced it with the one on my iphone (in the app www folder) like this:

```
function rotFeedback(jsonString) {
    var json = JSON.parse(jsonString);
    json.k='4692bd56dd3070f74b7e81c6b2f69339b0fd6062';
    if (json) {
```

loading the app with the static hash patched in, did indeed decrypt the egg:

Just Drive

first i thought this challenge was about crypto and i tried to interpret the code somehow, but that didnt work out.

i dont know why i didnt try this in the beginning, but after i googled some of these codes, i discovered that this is a PGN code (Portable Game Notation) and this actually represents a real chess game between Magnus Carlsen (Worlds best chess player) and Teimour Radjabov.

we obviously have to interpret the final position in a binary format like shown in the example and then convert it to decimal:

00000001 = 1
00100001 = 33
01001100 = 76
10101000 = 168
00100100 = 36
01000011 = 67
10000011 = 131
00000000 = 0

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

CHALLENGE 10 - BLUEPRINT

this challenge wants us to use a plot function and recreate a hacky easter 2016 banner. checking the source code from the webpage reveals this function:

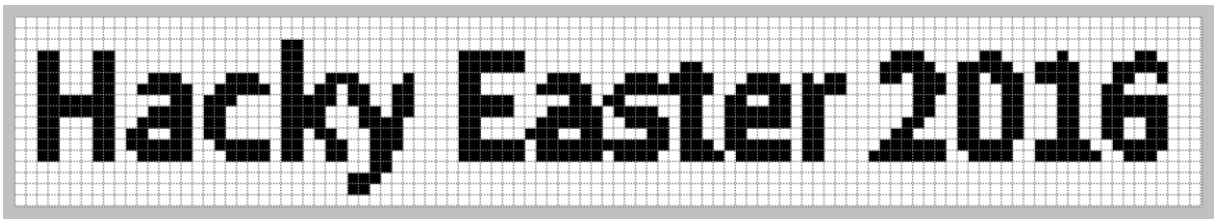
```
function plot() {
    var value = $("#value").val();
            var bigint = BigInteger(value);
            var y, yy, xx = 0;
    var context = $("#canvas")[0].getContext('2d');
            context.fillStyle = "#0000dd";
            context.fillRect(0, 0, 106 * 4, 17 * 4);
    for (x=105; x>=0; x--) {
            yy = 0;
            y = bigint;
            for (v=0; v<=16; v++) {
                    if (BigInteger.remainder(BigInteger.divide(BigInteger.divide(y, 17), BigInteger.pow(2,(17*x)+BigInteger.remainder(y, 17))), 2) > 0.5){
                    context.fillStyle = "#ffffff";
                            context.fillRect(xx, yy, 3, 3);
                    }
                    y = BigInteger.add(y, 1);
                    yy += 4;
            }
            xx += 4;
    }
    if (CryptoJS.SHA1(CryptoJS.SHA1(value)) == 'c63733cf3c66f0311b5c029f2a981f0239d990ad') {
                    alert("Match!");
            window.location.href = "challenge10_" + CryptoJS.SHA1(value) + '.html';
    }
}
```

by googling "**BigInteger.remainder(BigInteger.divide(BigInteger.divide(y, 17), BigInteger.pow(2,(17*x)+BigInteger.remainder(y, 17))), 2) > 0.5**" we can find out that this is the Tupper Self-Referencial Formula (https://en.wikipedia.org/wiki/Tupper%27s_self-referential_formula):

$$\frac{1}{2} < \left\lfloor \mathrm{mod}\left(\left\lfloor \frac{y}{17} \right\rfloor 2^{-17\lfloor x \rfloor - \mathrm{mod}(\lfloor y \rfloor, 17)}, 2\right)\right\rfloor,$$

now, since this is a well known formula, we also can find tools which produce the code for a given image, like this webpage here: http://damnsoft.org/others/tupper.html

i used the reference image and adjusted it in paint.net until it matched correctly:



using it on webpage got me this code:

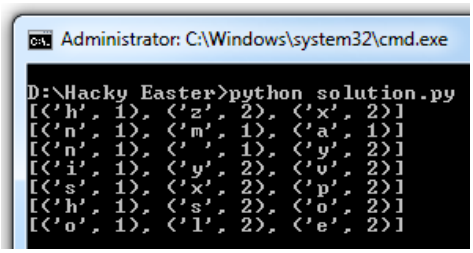
176579492015814901528872625529774615508215478614638628392406643239116428074897541681641793325671248874580950499668382723958388333354648532262931698930639856835422348683939828636055448533804591049653503261373974416464862181695983478562079067833614229059113869197437699759742373674003028861535476027091552243616865735457697656105444429506238584383051262100293283222118456901855469818763894181110080508013645884497726056403410392923221554648832542085467262023169013883606836051142881849628644501102960568482494045783425454238497295564 80

CHALLENGE 11 - TWISTED DISC

this challenge gave me some headache in the beginning. i thought i am smart and made up all possible solutions in python. but this file was very big and even with wordlists i didnt find the solution.

later i discovered, that there are a lot of similar chars in every ring, but some letters were only present once. so i stripped all chars away that were present multiple times per ring and checked the remaining chars. since i had already a python script available, i just used it to find the chars.

```
import collections

ring1 = "llepeop"
ring2 = "fflaohlskakos"
ring3 = "jqninxmpelxjqsimpel"
ring4 = "dopoefussycrlhuviddlyrvchefp"
ring5 = "muffyiolyumsschajdett cpidephjklonka"
ring6 = "wreezymklcfdtvqhvsnrzxubuwxhappysdfbkcooltq"
ring7 = "oowqfdhilfrmgpsrtvkbeanstzkcdauueiyzybmvxgpjlcxnjqw"
r1 = "".join(sorted(ring1))
r2 = "".join(sorted(ring2))
r3 = "".join(sorted(ring3))
r4 = "".join(sorted(ring4))
r5 = "".join(sorted(ring5))
r6 = "".join(sorted(ring6))
r7 = "".join(sorted(ring7))
print collections.Counter(r7).most_common()[:-3-1:-1]
print collections.Counter(r6).most_common()[:-3-1:-1]
print collections.Counter(r5).most_common()[:-3-1:-1]
print collections.Counter(r4).most_common()[:-3-1:-1]
print collections.Counter(r3).most_common()[:-3-1:-1]
print collections.Counter(r2).most_common()[:-3-1:-1]
print collections.Counter(r1).most_common()[:-3-1:-1]
```



so we have some collisions, but the chars remaining are

h, n or m or a, n or ' ', i, s, h, o

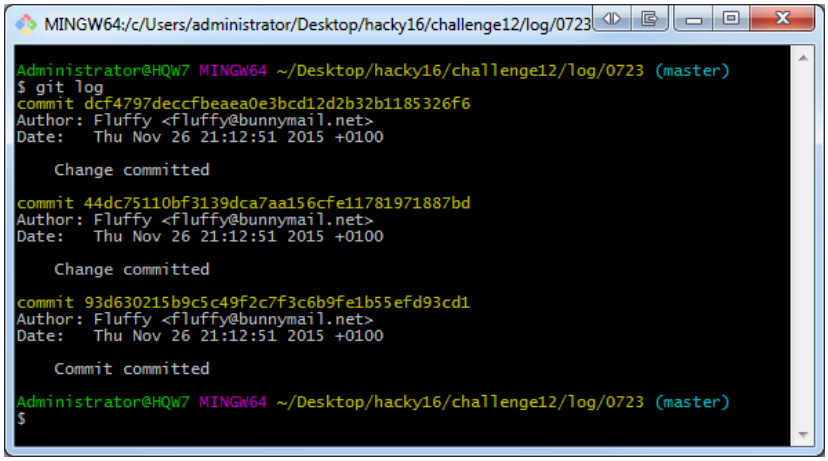
the only "word" that makes sense here is: **hanisho**

CHALLENGE 12 - VERSION OUT OF CONTROL

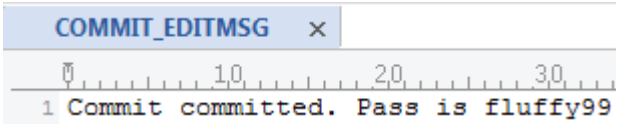
i must admit, that i dont have any clue about git and this challenge took me some time to solve. but in the end i installed a git shell and used a simple bash script to unpack all the zip files:

```
for ((i=1000; i>1; i--))
do
    unzip *.zip ; rm *.zip
    cd */
    git show | grep differ >>../log.txt
    git ls-files -d | xargs git checkout --
    mv *.zip ../
    cd ..
done
```

sadly this worked not straight until the end. whenever it got stuck, i manually checked the git commits (with git log) and unpacked (git checkout) the previous files from it to run it again in my script.



i had to do that about 4 times but finally found the last zip including the egg. this zip was password protected, but i was able to locate the password in the COMMIT_EDITMSG from the previous file:



CHALLENGE 13 - FRACTAL FUMBLING

this challenge was the most time consuming one – or lets say... most memory and patience consuming. we got a rather big image here, full of QR codes and somewhere on it, there is a code to get the egg.

i checked the file with paint.net and discovered, that the smallest QR codes are 21x21 pixels in size. the most obvious thing is, to crop all these codes and scan them. manually this is not possible obviously and therefore i have used imagemagick (http://www.imagemagick.org).

first i created 194480 small images from it with this command:

convert -crop 21x21 wallpaper.jpg tile%06d.jpg

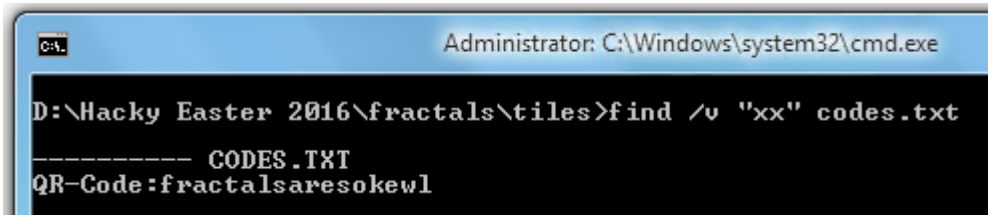
this took a lot of memory and time until these files were created, but it worked. then i tried to scan these QR codes with zbarimg (http://zbar.sourceforge.net), but it was not able to scan them because they were too small. therefore i resized them by 200% and made them black and white. this helped a lot and most of the QR's were scanable. with "forfiles" i converted the images and scanned them. the QR codes i saved in a txt file called codes.txt.

forfiles /m *.png /c "cmd.exe /c convert @file -scale 200% -colorspace Gray @file & zbarimg.exe @file" | find "QR" >>codes.txt"

this gave me thousands of results and all these had something in common:

```
4250 QR-Code:xxdnohjpaptlsfku
4251 QR-Code:xxpwgsuupddkdwyf
4252 QR-Code:xxqnztvyqbxgkopo
```

obviously we have to find a code without "xx" in the beginning. with the "find" command this is rather easy to do: **find /v "xx" codes.txt**



CHALLENGE 14 - P.A.L.M.

this challenge is obviously a client side authentication using javascript, but the first code does not work and is a honeypot.

```
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 + '.html';
    } else {
            alert('nope');
    }
}
```

the real code can be found at the end of the sites source code and is obfuscated:


```
<script>addFooter();
eval(atob("ZXZhbChmdW5jdGlvbihwLGEsYyxrLGUsZCl7ZTlmdW5jdGlvbihjKXtyZXRlcm4gYy50blN0cmluZygzNil9O2lmKCEnJy5yZXBsYWNlKC9eLyxTdHJpbmcpKXt3aGlsZShjLS0pe2RbYy50blN0cmluZyhhKV09a1tjKXx8Yy50blN0cmluZyhhKX1rPVtmdW5jdGlvbihlKXtyZXRlcm4gZFtlXXldO2U9ZnVuY3Rpb24oKXtyZXRlcm4nXFx3Kyd9O2M9MX07d2hpbGUoYy0tKXtpZihrW2NdKXtwPXAucmVwbGFjZSh1ZXcgUmVnRXhwKCdcXGInK2UoYykrJ1xcYicsJ2cnKSxrW2NdKXl9cmV0dXJuIHB9K
CdyIHEoKXsyIHU9Ny44KFwndlwnKS5iOzIgcD03LjgoXCd2XCcpLmI7MiA0PVswLDAsMCwwLDAsMCwwLDAsMCwwXTsyIDU9YzszKHU9PTlcJ2dcJyl7MyhwPjAmJnAuaD09OSl7NTlqO2YoaT0xO2k8PTk7aSsrKXsyIDY9cC5rKGktMSk7MiBhPXAuZSgwLGk
pOzMoNFs2XSE9MHx8YSVpIT0wKXs1PWN9Myg0WzZdPT0wKXs0WzZdPTF9fX19Myg1KXs3Lm0ubjlcJ29cJytlK1wneFwnK3ArXCcuclwnfXR7ZChcJ2xcJyl9fScsMzQsMzQsJ3x8dmFyfGlmfHVzZWR8b2t8ZGlnaXR8ZG9jdWllbnR8Z2V0RWxlbWVudEJ5S
WR8MTB8cGFydHx2YWx1ZXxmYWxzZXxhbGVydHxzdWJzdHJpbmd8Zm9yfGVsc2F8bGVuZ3RofHx0cnVlfGNoYXJBdHxub3BlfGxvY2F0aW9ufGhyZWZ8Y2hhbGxlbmdlMTRffHxjaGVja0VudHJpZXN8ZnVuY3Rpb258aHRtbHxlbHNlfHxwcGFzc3xwdXNlcnx
fJy5zcGxpdCgnfCcpLDAse30pKQ=="));</script>
```


i decoded the string (base64) and used jsbeautifier (http://jsbeautifier.org) to make it more readable:

```
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];
    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, we have here a loop that checks for digits and also if a digit was used already. we must find a number that matches a modulus for each round using the previous digits. this is an interesting algorithm - i made up a python script to solve it:

```
from itertools import permutations

perms = [''.join(p) for p in permutations('9876543210')]
for perm in perms:
    for i in range(1, 11):
        part = perm[:i]
        if int(part) % (i) != 0:
            break
        if i==10:
            print perm
```

```
Administrator: C:\Windows\system32\cmd.exe

D:\Hacky Easter>python perms.py
3816547290
```

this gave only one hit: 3816547290. so the login is **elsa:3816547290**

CHALLENGE 15 - BIG BAD WOLF

this challenge gives us a file which seems to be a sortof disk image. using winhex we can open it and simply extract the containing files:

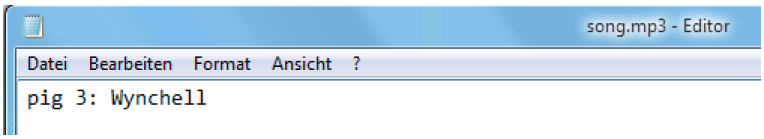
disk

\

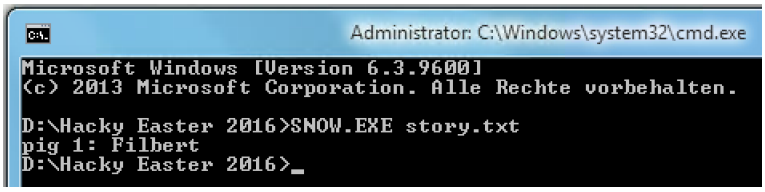
Name	Ext.	Size	Created	Modified	Inode changed	Attr.	1st sector
(Root directory)		1.0 KB		29.11.2015 07:51:4...	29.11.2015 07:51:4...	rwxr-xr-x	636
lost+found		12.0 KB		29.11.2015 07:51:2...	29.11.2015 07:51:2...	rwx----...	638
.badblocks		0 B		29.11.2015 07:51:2...	29.11.2015 07:51:2...	---------	
piglet.jpg	jpg	91.6 KB		29.11.2015 07:51:4...	29.11.2015 07:51:4...	rwxr-x---	1'026
pigs.jpg	jpg	93.6 KB		29.11.2015 07:51:4...	29.11.2015 07:51:4...	rwxr-x---	1'212
song.mp3	mp3	2.5 MB		29.11.2015 07:51:4...	29.11.2015 07:51:4...	rwxr-x---	1'402
story.pdf	pdf	71.6 KB		29.11.2015 07:51:4...	29.11.2015 07:51:4...	rwxr-x---	6'612
story.txt	txt	1.6 KB		29.11.2015 07:51:4...	29.11.2015 07:51:4...	rwxr-x---	6'758
wolf.jpg	jpg	186 KB		29.11.2015 07:51:4...	29.11.2015 07:51:4...	rwxr-x---	6'762

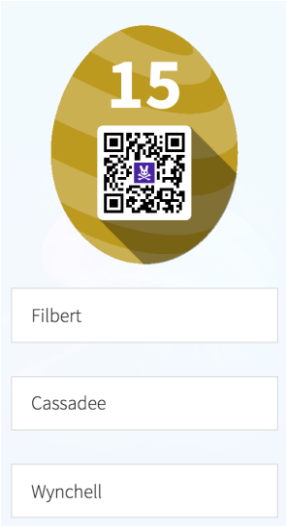
by looking through the files we can find the pig 2 rather easy:

using mp3stego (decode.exe -X song.mp3, empty password) we can find the second pig:



the first pig was a little harder to find. i concentrated too much on the pdf, but didnt find anything. there is actually the possibility to hide information in plain txt files using the whitespace steganography. this tool here helped in this case: http://www.darkside.com.au/snow/





CHALLENGE 16 - EGG COLORING

this was my first challenge, which i solved @10:01 – instantly when the event started. this was possible, because i decompiled the android application before hacky easter actually started.

```
private void colorize(int index) {
    String url = "http://hackyeaster.hacking-lab.com/hackyeaster/egg?code=" + codes[index] + "&key=" + KEY + "&hmac=" + hmacs[index];
    ImageView image = (ImageView) findViewById(C0174R.id.imageView);
    new EggLoader().execute(new Object[]{url, image, this});
```

from this code it was clear what we had to send to the server. we even had some examples in the binary to work with:

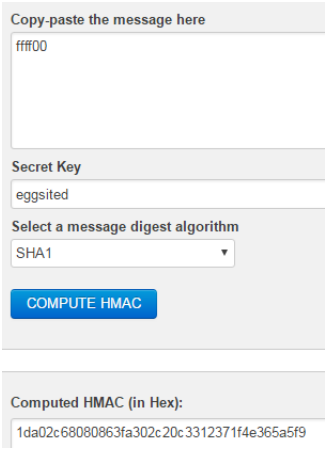
```
static {
    codes = new String[]{"ff0000", "00ff00", "0000ff", "00ffff", "ff00ff", "ffffff", "000000"};
    hmacs = new String[]{"f4e075524ba4470867e1891c1a8d1fc21df1f56a", "b23f66454417de5be448da84a846989b42f304c8",
        "f5ecdd0f12749fe75734b42bf29943d28acf4573", "2cf2a7cd695a462adcbc324df9302003a99c688a",
        "543e3853ac9318587c10c7645b6828e2a858ecf5", "c46ffadf392698e28fdeb344239130e2ade2c809",
        "7b06466eb80d88533a2d1c7b9de62d98c4e20d1d"};
}
```

so, the app wants us to send yellow, but there is no yellow option available. this could have been done with a proxy for example, but i decided to go the easy way. to create a HMAC we need the secret key, which is also visible in the source code:

```
private static final String EGG_URL = "http://hackyeaster.hacking-lab.com/hackyeaster/egg";
private static final String KEY = "eggsited";
```

now, we can find out easy, that yellow should be 0xffff00 and knowing the secret key, we can use an online HMAC generator to get our values:

http://www.freeformatter.com/hmac-generator.html



we know now, how to build our url to get the egg:
http://hackyeaster.hacking-lab.com/hackyeaster/egg?code=ffff00&key=eggsited&hmac=1da02c68080863fa302c20c3312371f4e365a5f9

the result was in base64, which can be converted online or in cryptool to get the final PNG file (visible in the file header)

CHALLENGE 17 - BUNNY HOP

so this challenge gives us a source code, including the hint, that it is probably a known language. googling for "right 180" and "programming language" leads us directly to "logo". there is obviously something called turtle graphics, which can be used to paint images using the logo language.

https://en.wikipedia.org/wiki/Turtle_graphics

so the goal was obviously to recreate the functions: backhop, hop and egg. the main problem for me was to understand, that the egg (which is a square here) must be painted from the middle of it and the cursor must return to the center of the egg afterwards. anyway here is my working code:

```
clearscreen

to backhop :size
    penup
    back :size
    pendown
end

to hop :size
    penup
    forward :size
    pendown
end

to egg
    penup
    back 5 right 90 back 5 left 90
    pendown
    REPEAT 4 [FORWARD 10 RIGHT 90]
    penup
    PENUP
    RIGHT 45
    FORWARD 5
    SETFLOODCOLOR [0 0 0]
    FILL
    BACK 5
    LEFT 45
    forward 5 right 90 forward 5 left 90
    PENDOWN
end
```

i ran his in the software FMSLogo (http://fmslogo.sourceforge.net) and got a nice QR code:

CHALLENGE 18 - BUG HUNTER

in this challenge we got some buggy code in C#. sadly this didnt run in visual studio 2010 and i had to install the newer version to get it working. there were various errors, but the comments helped to fix them:

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

here we should use decimal instead of hex.

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

this will not run 1000 iterations, only 999. we can fix this by using <= instead of < only.

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

this is not a proper condition. we need to use { and } to fix it. else it will only run the first instruction on the condition.

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

basic mathematics. we need to use () to seperate the operations properly.

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

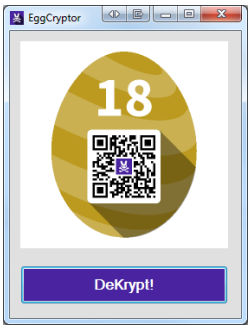
this will not multiply b by 3, but by 4. we can use b = b + b + b; instead.

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

this is not swapping the values. we can use c = c + d; d = c - d; c = c - d; to swap without temp variable.

```
// Take last four digits (modulo 10'000),
a %= 10000;
b %= 10000;
c %= 10000;
d %= 100000;
```

obviously a typo. we just need to fix d %= 10000;



CHALLENGE 19 - ASSEMBLE THIS!

finally a hard challenge and even in assembly. nice! we are given an assembly output generated by gcc. using an x64 linux distro, we can compile this code into a binary using gcc like this:

gcc -c file.S -o file.o && gcc file.o -o file

i opened this binary in IDA and had a look at the pseudo code. i suck at high-languages and the assembly was not very clean and x64, which im not really familiar with. i probably could reduce the operations and code a script to solve it, but i decided to go the lazy way: bruteforce.

from the code it seems like there are 4 checks and there must be some collisions therefore. also there is a check for the length of the input, which must be bigger than 16 chars. i have used nasm to compile it in assembly (my first x64 assembly program) and injected my own bruteforcer.

i discovered an interesting instruction for the x64 processor: https://en.wikipedia.org/wiki/RdRand

RDRAND (previously known as Bull Mountain) is an instruction for returning random numbers from an Intel on-chip hardware random number generator.

pretty helpful. i used this instruction to get a random number and then filled the 18 bytes buffer with random bytes. first i tried to generate ascii values, but that never gave a result.

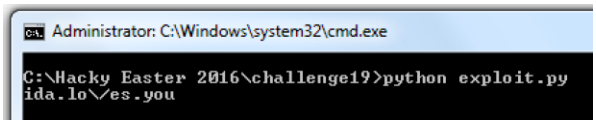
i ran the exe in x64dbg (http://x64dbg.com) and went out for lunch. when i came back my breakpoint after the checks was reached:

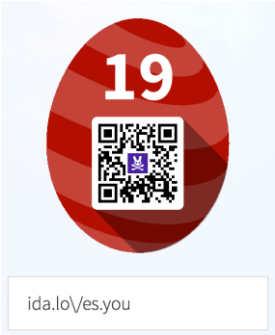
using the data window in x64dbg i was able to dump the key:

87 06 48 16 94 F0 42 0E 4E 1E F6 05 29 DD 30 29 87 17

ok, now i had only to send the result to the server, which didnt work manually because of the non ascii bytes. a small python script helped in this case:

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = "hackyeaster.hacking-lab.com"
port = 1234
s.connect((host,port))
s.send("\x87\x06\x48\x16\x94\xF0\x42\x0E\x4E\x1E\xF6\x05\x29\xDD\x30\x29\x87\x17")
s.send("\x0a")
data = s.recv(1024)
print data
s.close ()
```





CHALLENGE 20 - HUMPTY'S DUMP

this challenge is based on mysql and offers us a database dump. to play with the full database, i have setup a local mysql on my pc and imported the dump with:

create database humpty;
mysql -u root -p humpty < humpty_fyle.sql – of course for all files. then i was able to use the database with queries:

```
mysql> use humpty;
Database changed
mysql> select * from uzr;
+-------+--------+-----------------------------------------+-------+
| id    | neighm | puzz                                    | sawlt |
+-------+--------+-----------------------------------------+-------+
| 1342  | stuart | de2278f5bcafcbb097ecc1fb54e5ab8a9e912c55 | efgh  |
| 1875  | beaver | 943f9ecbbd91306a561d0e3c15e18ee700007083 | abcd  |
| 3443  | chuck  | 0cf32f8f418659f23f8968d4f63ea5c98b39f833 | zyxw  |
| 5420  | yogo   | 1742ae4507fc480958e2437104e677e70aa5e857 | jklm  |
| 8944  | flint  | 915d253cb5ba6f0a220bca83e2d6d3258af15e68 | nmlk  |
+-------+--------+-----------------------------------------+-------+
```

first i had to crack a hash to get a password. for this i used python and some wordlists:

```python
import hashlib

with open(r"D:\Dictionaries\inside3.txt") as f:
    for line in f:
        line=line.rstrip('\n').upper()
    if hashlib.sha1("efgh."+line+".efgh").hexdigest()=="de2278f5bcafcbb097ecc1fb54e5ab8a9e912c55":
        print line, hashlib.sha1("efgh."+line+".efgh").hexdigest()
    if hashlib.sha1("abcd."+line+".abcd").hexdigest()=="943f9ecbbd91306a561d0e3c15e18ee700007083":
        print line, hashlib.sha1("abcd."+line+".abcd").hexdigest()
    if hashlib.sha1("zyxw."+line+".zyxw").hexdigest()=="0cf32f8f418659f23f8968d4f63ea5c98b39f833":
        print line, hashlib.sha1("zyxw."+line+".zyxw").hexdigest()
    if hashlib.sha1("jklm."+line+".jklm").hexdigest()=="1742ae4507fc480958e2437104e677e70aa5e857":
        print line, hashlib.sha1("jklm."+line+".jklm").hexdigest()
    if hashlib.sha1("nmlk."+line+".nmlk").hexdigest()=="915d253cb5ba6f0a220bca83e2d6d3258af15e68":
        print line, hashlib.sha1("nmlk."+line+".nmlk").hexdigest()
```

this gave me only one hit. the password was "snakeoil" for user chuck. with this pass i was able to create the next hash which then is used to decrypt the password in the database:

```python
import hashlib

count = 1
p_tmp="kee.snakeoil.kee"
while (count <= 10000):
    p_tmp=hashlib.sha1(p_tmp).hexdigest()
    count = count + 1
print p_tmp
```

using this new hash, i decrypted the key from the database with this query:

```
mysql> SELECT AES_DECRYPT(UNHEX(kee), '1b7b21c2204fcfdb18006c8b4b2bf6437e7850a3'
) from kee where id=2332;
+-----------------------------------------------------------------------+
| AES_DECRYPT(UNHEX(kee), '1b7b21c2204fcfdb18006c8b4b2bf6437e7850a3') |
+-----------------------------------------------------------------------+
| jpP8HeoEC5OCCBqdf9N3                                                  |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

and finally to decrypt the blob i ran this command:

```
mysql> SELECT AES_DECRYPT(blahb,'jpP8HeoEC5OCCBqdf9N3') FROM fyle WHERE id=3492
INTO DUMPFILE 'D:\\Hacky Easter\\solution.png';
Query OK, 1 row affected (0.00 sec)
```

CHALLENGE 21 - CRYPTO COUNCIL

this challenge was too easy for being a hard one. the images on the left were hints to the inventors of the ciphers and therefore very easy to locate (with google image search).

we have here:

- Caesar
- Polybios
- Blaise de Vigenère
- William Playfair

now we also know the ciphers we have to use. these are all solveable with cryptocrack or online tools: https://sites.google.com/site/cryptocrackprogram/

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

Polybios: 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

Vigenere (Key: PARIS): 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

Playfair: the playfair cipher was the first practical digraph substitution cipher the scheme was invented by charles w heatstone but was named after lord playfair who promoted the use of the cipher password is bletchley

CHALLENGE 22 - DUMPSTER DIVING

in this challenge we have to crack some hashes. looking them up online didnt give any results and as a hint we get a part of the code, that has been used to create the hashes. first i didnt notice it, but its actually just a modified SHA-1 algorithm, where the constants has been replaced:

```
h0 = 0x10325476
h1 = 0x98BADCFE
h2 = 0xEFCDAB89
h3 = 0x67452301
h4 = 0x0F1E2D3C
```

original constants:

```
h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0
```

to crack these hashes, we have to implement a modified SHA-1 algorithm and then run it on some dictionaries. i used python to make a script which does exactly that:

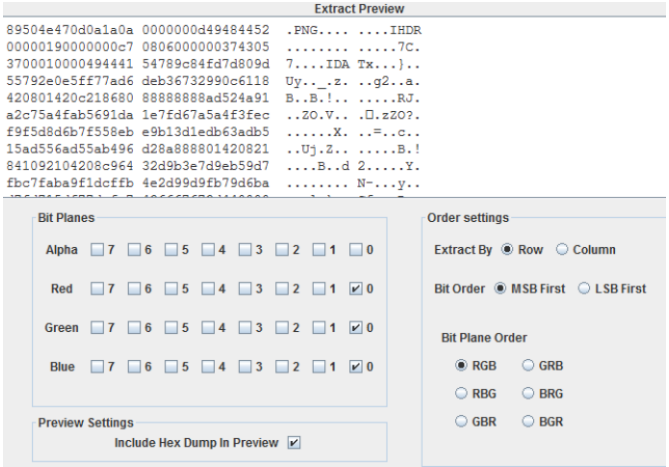
```
def sha1(data):
    bytes = ""

    h0 = 0x10325476
    h1 = 0x98BADCFE
    h2 = 0xEFCDAB89
    h3 = 0x67452301
    h4 = 0x0F1E2D3C

with open("D:\\Dictionaries\\inside1.txt") as f:
    for line in f:
        if sha1(line.rstrip('\n'))=="0d6bb0df8918168798ce6b770014aeb81ac6ce76":
            print line, sha1(line.rstrip('\n'))
```

this took some time and a few good wordlists, but it worked:

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

CHALLENGE 23 - HEIZOHACK

in this challenge we should find a password in the image. this sounds like steganography. for such cases i usually have a quick look with stegsolve (https://www.wechall.net/de/forum-t527/Stegsolve_1_3.html). with a little trial and error, i discovered a new PNG file in the given image:



saving this as binary resulted indeed in a new image:

so the formula tells us to do something with the r bit (LSB) and we can see that there is something in stegsolve, but its not extractable with it. to get the binary data i have used VB.NET and an online binary converter:

```
Dim bmp As New Bitmap("pngimage.png")
For j = 0 To 50
    For i = 0 To bmp.Width - 1
        Dim pixel As Color = bmp.GetPixel(i, j)
        Dim r = pixel.R And 1
        Dim g = pixel.G And 1
        Dim b = pixel.B And 1
        Dim a = pixel.A And 1
        If (r Xor b Xor g Xor a) = 1 Then
            TextBox1.Text += r.ToString
        End If
    Next
Next
```

lostinthewoooods

To Binary

01101100011011110111001101110100011010010110111001110100011010000110010101110111011011111011011110110111101111011011110100001100011

To Text

CHALLENGE 24 - CRUNCH.LY

oh no, a java challenge. every year im losing hair because of these java challenges. ohwell... it was still fun and i only ran my code the whole night, because i made a mistake and checked always the same key. then i failed at string comparing, because java works different and i used "==" which does not compare strings. bleh.

anyway.. the idea of the challenge is to find a collision of this code:

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
md.update(url.getBytes("UTF-8"));
byte[] hash = md.digest();
byte[] part = new byte[4];
System.arraycopy(hash, 0, part, 0, 4);
String b32 = new String(new Base32().encode(part), "UTF-8");
b32 = b32.replaceAll("=", "");
return "http://crunch.ly/" + b32;
```

this took quite some time and i had only luck with a string of 6 chars length (upper, lower, digits).

```
String myurl = "http://evileaster.com/"+curr;
if (calculateShortUrl(myurl).equals("http://crunch.ly/IU66SMI")) {
    System.out.println("found: "+myurl);
```

i have found http://evileaster.com/E0r4aS as a working collision.

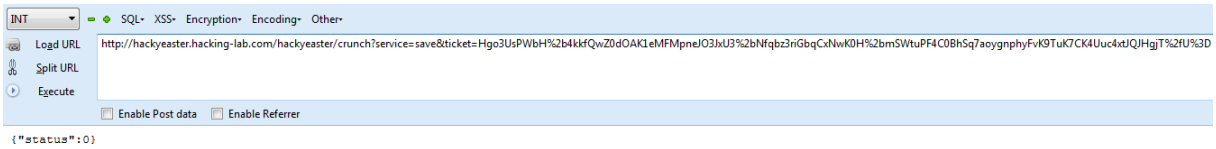
then we had to bruteforce the KEY which obviously was only 5 bytes long (very bad mistake in the code). for that i have used my fake url: http://t.c which gave http://crunch.ly/OG7IPLA in the shortener. here only upper and lower characters worked pretty well:

```
found: tKguF
ticket: Hgo3UsPWbH+4kkfQwZ0dOAK1eMFMpneJO3JxU3+Nfqbz3riGbqCxNwK0H+mSWtuPF4C0BhSq
7aoygnphyFvK9TuK7CK4Uuc4xtJQJHgjT/U=
```

now that we had the ticket and our fake url, we just needed to store it somehow. from the script.js file in the page source code we knew the service url:

```
var serviceUrl = "http://hackyeaster.hacking-lab.com/hackyeaster/crunch?";
```

we can now add the ticket to the service url to store our fake URL - but we need to URLencode it – i used hackbar for that:




```
{"status":0}
```

when we enter now IU66SMI in the URL shortener service, we will be redirected to our egg: