HACKY EASTER 2015 WRITEUP BY HARDLOCK

challenge 01 - Puzzword 2
challenge 02 - It's in the Media 3
challenge 03 - Lego Stego 4
challenge 04 - Twisted Num63rs 5
challenge 05 - Phone Fumbling 6
challenge 06 - Hack to the Future 7
challenge 07 - Vista de la Calle 8
challenge 08 - Spread the Sheet 9
challenge 09 - Fisheye 10
challenge 10 - Thumper's Den 11
challenge 11 - You've got Mail 12
challenge 12 - This is just a Test 13
challenge 13 - Leet TV 14
challenge 14 - Wise Rabbit's Return 16
challenge 15 - Photo Shooting 17
challenge 16 - Ghost Room 18
challenge 17 - Spot the Difference 19
challenge 18 - Sharks on Wire 21
challenge 19 - Cut'n'Place 22
challenge 20 - Lots of Bots 23
challenge 21 - Cony Code 25
challenge 22 - Hashes to Ashes 27
challenge 23 - Beat the Nerd Master 28
challenge 24 - SHAM Hash 30
challenge 25 - Jad & Ida 31
challenge 26 - Clumsy Cloud 34
challenge 27 - Too Many Time Pad 36

CHALLENGE 01 - PUZZWORD

this challenge shows an image of some letterpuzzle and we notice, that some stuff actually is missing. ACEHKRZ – this is an anagram and can be rearranged to HACKERZ. entering this in lowercase characters will reveal our first egg.

hacking for babies

CHALLENGE 02 - IT'S IN THE MEDIA

in this challenge we can see an egg already, but its not scannable. also there is the word "NO" visible in the QR code. looking at the page source, we can see this hint:

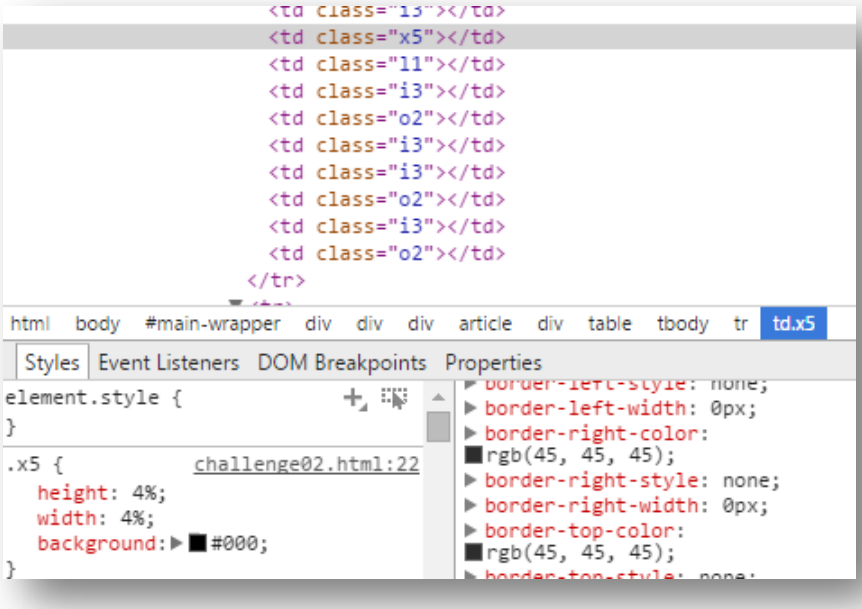
```
<script>document.writeln(String.fromCharCode(117, 115, 101, 32, 99, 104, 114, 111, 109, 101));</script>
```

using an online converter (http://jdstiles.com/java/cct.html) i ran fromCharCode() on it and revealed:

ok then.. lets check this in **chrome**. i opened the QR with "inspect element" and noticed, that its all made up with html tables (<td></td>) and uses CSS to colorize them. there are different classes (i3, o2, x5).

here i just played a little and changed the colors from the classes in the inspector. the class x5 will complete the QR code when we set it to black (#000)

and here we got our egg:



CHALLENGE 03 - LEGO STEGO

ok this challenge took me some time, but just because i keep overcomplicating the things. we are given a file with lxf extension. google will help us to find out, that this is a lego designer file.

we can download the application here: http://ldd.lego.com/en-us/download/ and we can open and edit the given file with it.

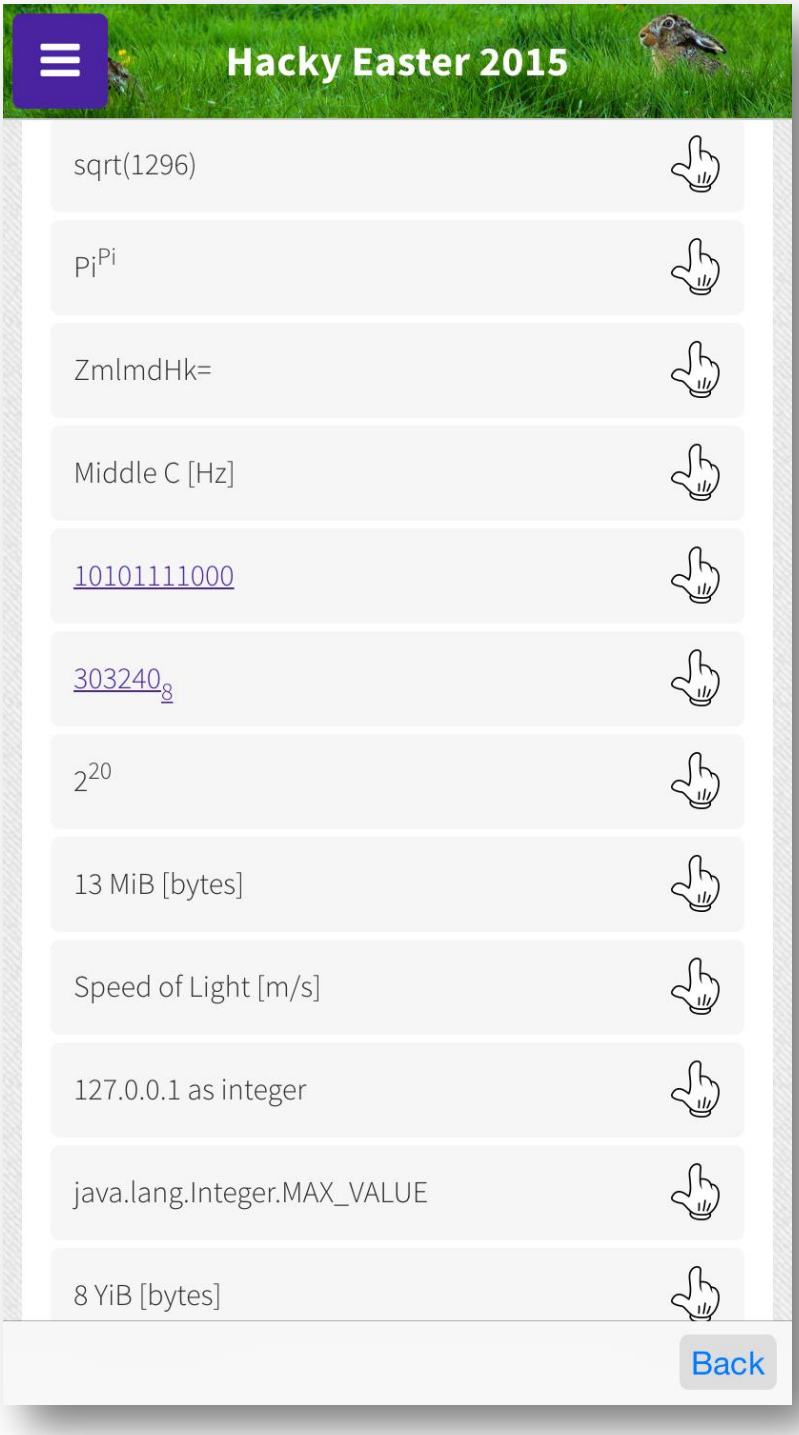
here i thought, that i have to make my own QR code in lego from the stones that are given. i pressed F7 for the builder mode, which created an animation how to build this lego image. watching a part of it, i noticed that there are some black stones hidden under the white ones!

nice... now it was easy. i just removed the upper layer of the image with the multi selector tool and voila: a scannable QR code.

CHALLENGE 04 - TWISTED NUM63RS

In this mobile challenge we are given some different numbers in various formats. The goal is, to order them **ascending**.

With the help of some online converters and google, I made up the solution which looks like this:



CHALLENGE 05 - PHONE FUMBLING

this is a mobile challenge, which wants us to use the phone sensors to make all bars appear full in the hacky easter app.

i checked the disassembled iphone binary and i have found these indicators:

```
__text:00016E00                 MOV             R1, PC ; selRef_deviceMotion
__text:00016E02                 LDR             R5, [R0] ; CMMotionManager *motionManager;
__text:00016E04                 LDR             R4, [R1] ; "deviceMotion"
__text:00016E06                 LDR.W           R0, [R8,R5]
```

some motion sensors are in the game

```
__text:00017004                 MOV             R1, R6
__text:00017006                 BLX             _objc_msgSend
__text:0001700A                 MOV             R1, #(selRef_batteryLevel - 0x17016)
__text:00017012                 ADD             R1, PC ; selRef_batteryLevel
__text:00017014                 LDR             R1, [R1] ; "batteryLevel"
```

and some battery level is checked too

I just charged my phone while doing other challenges and then i ran the app, moved the phone around but it didnt work.

i decided to check it out later and just put the iphone horizontally on my desk – then a funny thing happened:

without doing anything, the phone solved the challenge alone and the QR code appeared

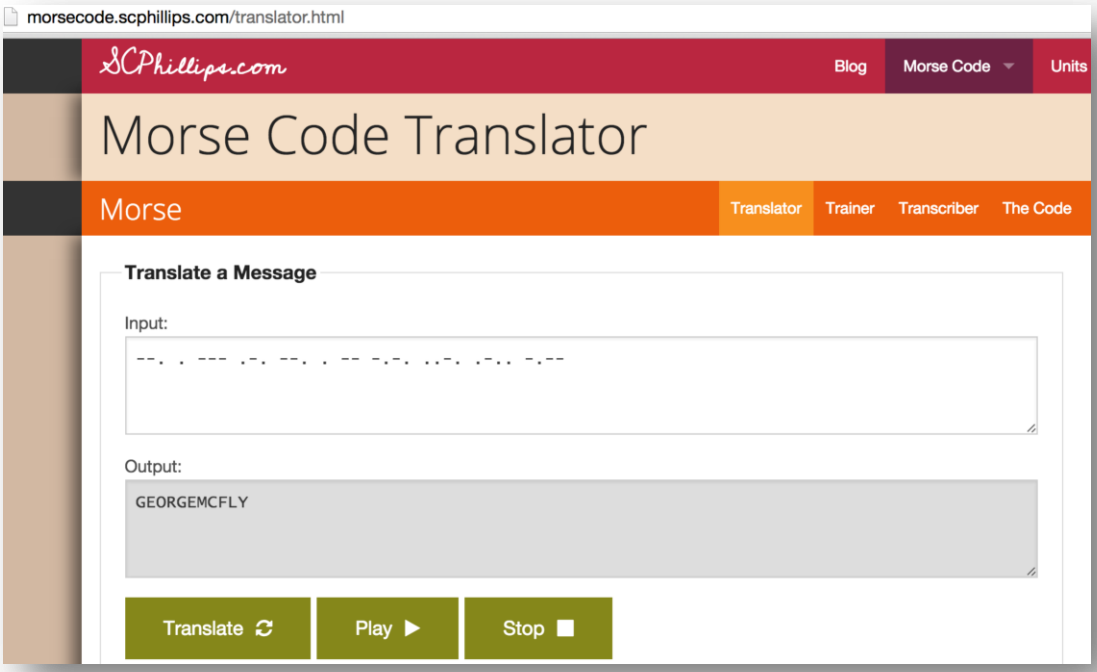
i actually dont know what this challenge really required. lol.

CHALLENGE 06 - HACK TO THE FUTURE

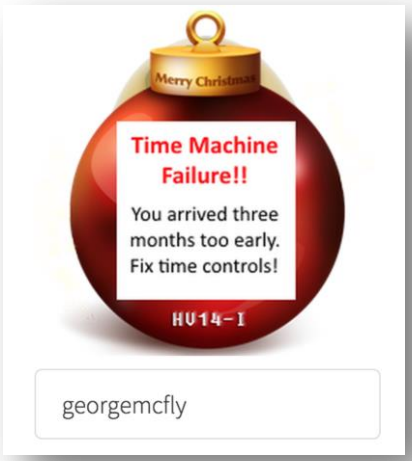
In this challenge we are given a code – which we can identify rather quickly - a morse code.

dah-dah-dit dit dah-dah-dah di-dah-dit dah-dah-dit dit dah-dah dah-di-dah-dit di-di-dah-dit di-dah-di-dit dah-di-dah-dah

to convert it, we can search and replace "dah" with a dash and "dit" and "di" with a dot. This will give us this morse code, which we can translate online:



the input box tells us to use lowercase only but when enter the key, we get a time error message



in the html source code, we can see that this is verified on the client side and therefore I just set my computers date three months to the future and when I resent the code, it gave me the correct egg.

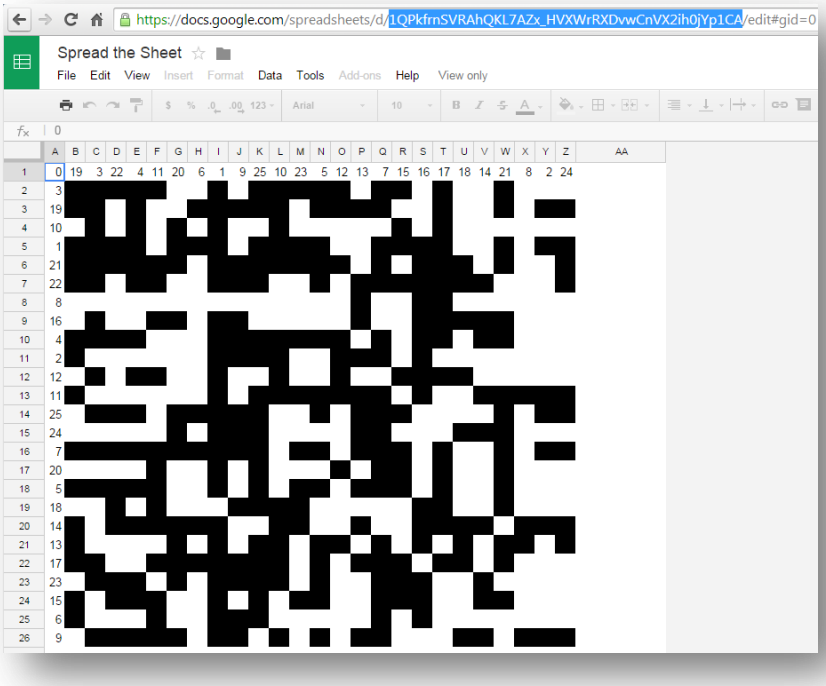
CHALLENGE 07 - VISTA DE LA CALLE

this is another mobile challenge which offers a 3D view of an aera. we have to find the qr code by traveling through these locations. since i already dumped the mobile app from my iphone and had full access to all files, i have found this qr code without even trying to solve this challenge. the qr code is in the **quito2_u.jpg** file, but its not clear enough to scan.

no problem sir. in paint.NET i cropped the qr code, then changed colors to black and white, selected automatic corrections and contrast to 100% - now it looks much better!

CHALLENGE 08 - SPREAD THE SHEET

from the screenshot of this challenge, we can see some online spreadsheet. the first thing that came in my mind was google drive. i opened an already existing spreadsheet from my google account and changed the spreadsheet id to the one from the challenge:



nice. now we just have to rearrange the rows and columns to make a proper QR code. for that i have copied everything to my local excel, because it was easier to order there.

first i ordered the rows with the normal order function in excel and then i changed to a user defined ordering, to sort the columns. in the end i changed the width of the cells to 2 and had my QR code nicely fixed.

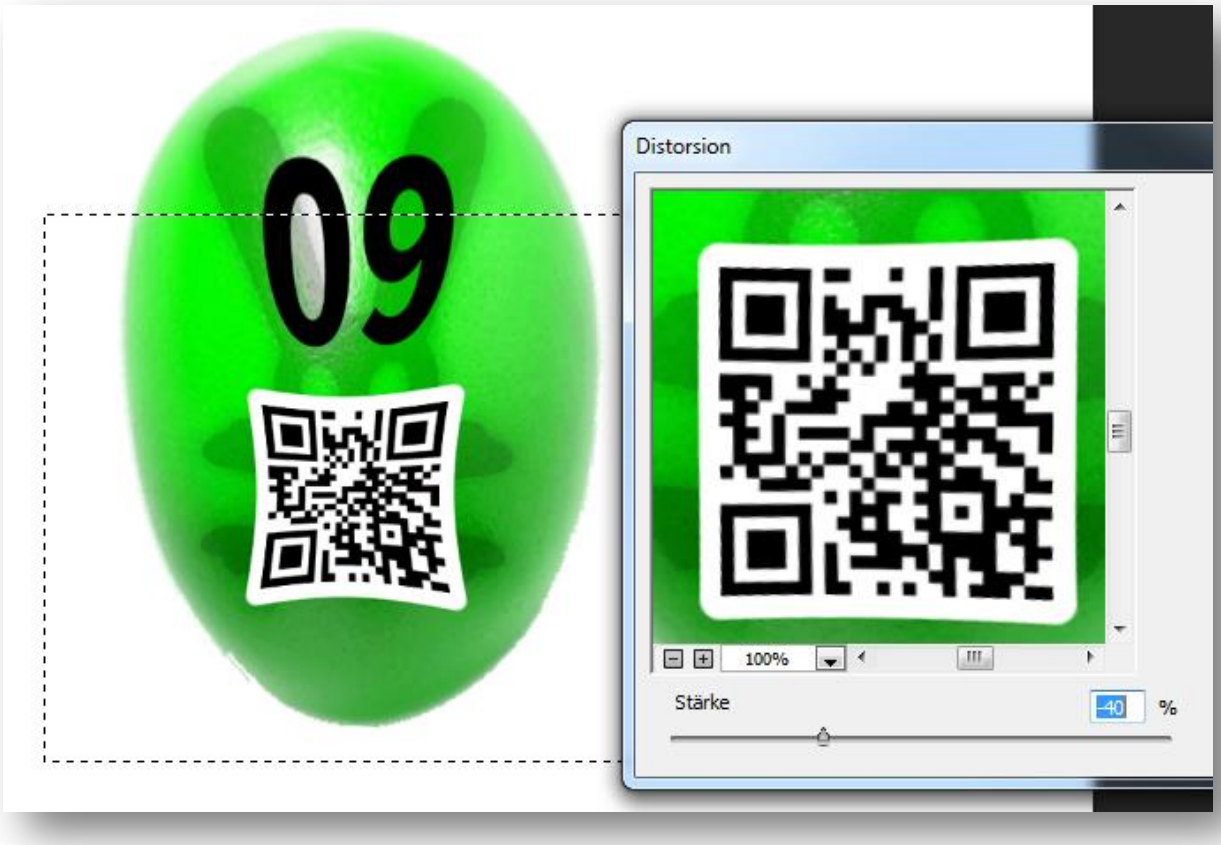
CHALLENGE 09 - FISHEYE

this challenge wants us to find a hidden egg in the mobile app.

actually its the first visible egg for everyone who launches the mobile hacky easter app. its the splash screen – at least on iOS.

since i have copied and decrypted the app from my iOS device already, it was easy to get the image with a file explorer.

from the "**Hacky Easter.app**" i just opened the LaunchImage.png (various sizes available) but the qr code was distorted – so, lets undistort it in photoshop!



i just selected an aera around the qr code, from the filter gallery i have chosen the distorsion filter and there i have set the level to minus 40.

not perfect, but this made the qr scanner happy.

CHALLENGE 10 - THUMPER'S DEN

this easter egg alike challenge was actually pretty easy, but still took me a while to find. it gives us a hint about a hidden egg which Thumper himself has bagged. the only way to find something related a different person on the site is the egg basket. we can therefore check everybodys basket by just changing the name.

now guess what we can find when checking Thumper's basket?

oh nooo you have found my bagged egg!

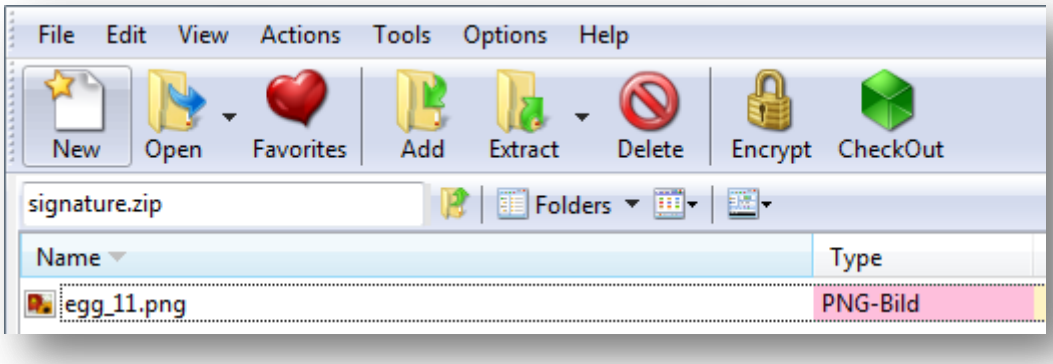
CHALLENGE 11 - YOU'VE GOT MAIL

for this challenge we get some sortof mail files. a few files are zipped up and we can assume they are inbox or mailclient related things.

i simply opened the "inbox" file in ultraedit and scrolled down until i have found an attachement encoded in base64:

```
--047d7b4501642dc6f905043957bd--
--047d7b4501642dc6fe05043957bf
Content-Type: application/zip; name="signature.zip"
Content-Disposition: attachment; filename="signature.zip"
Content-Transfer-Encoding: base64
X-Attachment-Id: f_i0o7q80j0

UEsDBBQAAAAIAJ2iMUVXUT5FQfwAAJP9AAAKAAAAZWdnXzExLnBuZ3xYdzQbXhuuKmoU1apNW7PU
VlSM+qm9WrR+paFq7y2IUbX3rhqhpShae0fsvfdKImLWSGImCPH1+/8733vO/ePec+57znvPc5/n
ed/YUwbX3rhqhpShae0fsvfdKImLWSGImCPH1+/8733vO/ePec+57znvPc5/n
ed/YVwaad2jYaW7cuHFHW+ul0Y0bN7H/XbfJ57ERq92rOxPoBYqD46
gG9sDJ6erB4dIE+O0Qe45Y6OBPzJGv50vb8/k3C61t+X3tOdjEI0w5eqr6+v/16/XqPp+UXVU0zX
```

then i just converted this base64 to binary and named it signature.zip. opening this zip revealed the egg 11.

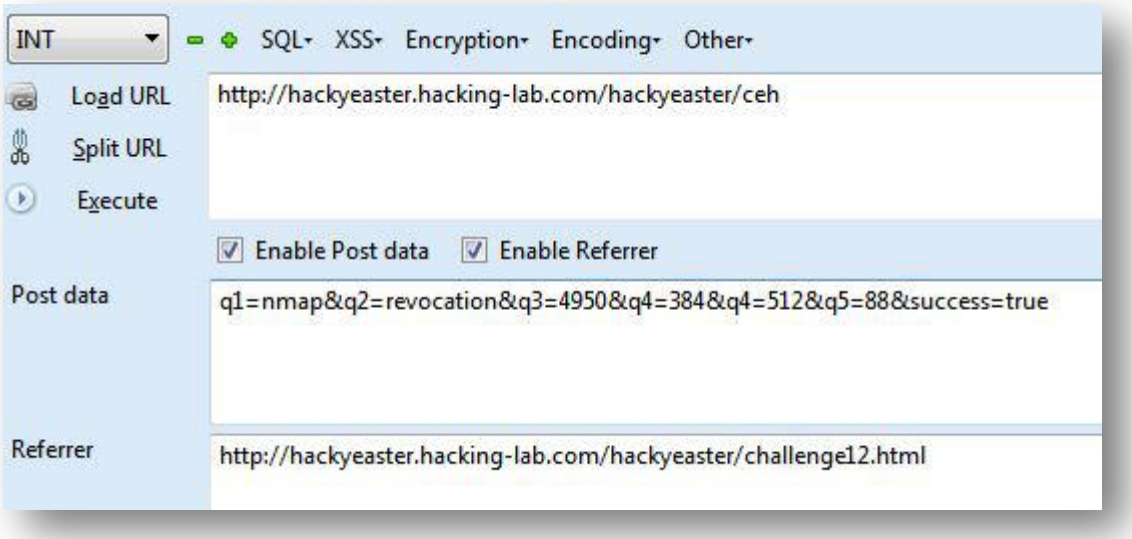


CHALLENGE 12 - THIS IS JUST A TEST

this challenge wants us to pass some online test, but the fields are not allowing the correct answers. also we have to find the correct answers to these questions.

- **nmap** is the portscanner
- **revocation** is the missing word in CRL
- we need N*(N-1)/2 (**4950**) keys for 100 people to use symmetric encryption
- sha2 allows bitsizes of **384** and **512**
- kerberos uses port **88** (UDP)
- success must be set to **true**

to send in my answers, i have used the hackbar addon in firefox:



CHALLENGE 13 - LEET TV

this challenge shows us a video with a lot of QR codes. since its a quite long video, we cannot just scan all QR codes manually and the ones i have tested, didnt work.

so lets just extract all the codes and scan them all to see if we can find something interesting. it looks like the code changes each second and we can use some command line tools to get all of them.

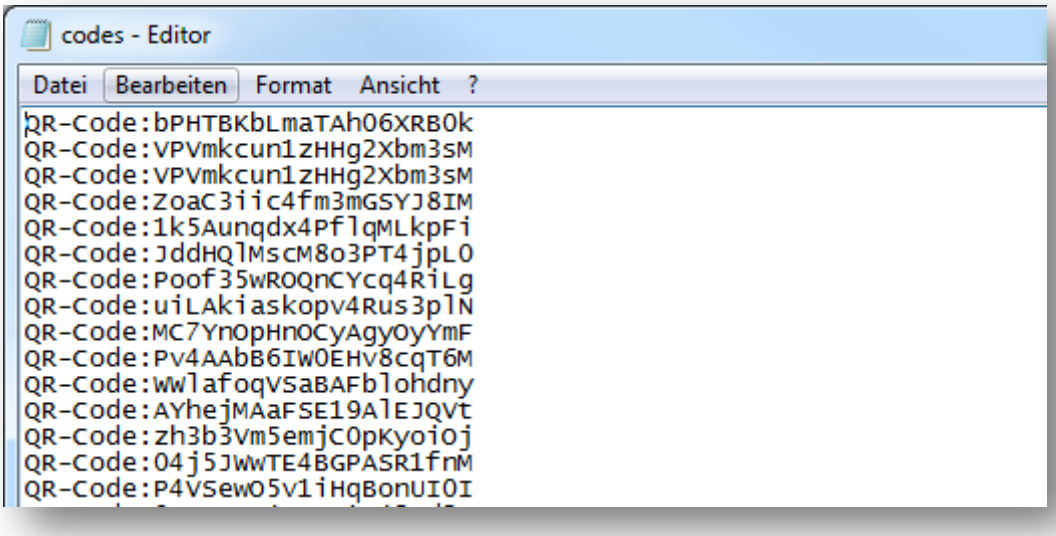
i simply used ffmpeg (https://www.ffmpeg.org/download.html) to extract an image of the video on every second with this command line:

ffmpeg -i leettv.mp4 -r 1 -f image2 image-%3d.jpeg

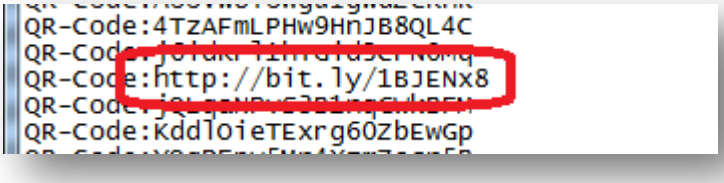
this resulted in 900 images – all with a different qr code. now we just need to scan them all – another command line utility will help us here: ZBar (http://zbar.sourceforge.net/index.html).

on windows we can use this simple commandline to scan all files at once and capture the output in a new file:

forfiles /m *.jpeg /c "cmd.exe /c zbarimg.exe @file" | find "QR" >>codes.txt



oh.. a lot of qr codes.. now what? a quick scroll through them revealed something special



this URL will redirect us to this file:

http://hackyeaster.hacking-lab.com/hackyeaster/leettv_qbEtJZKLTLB3jByIWSpE.wav

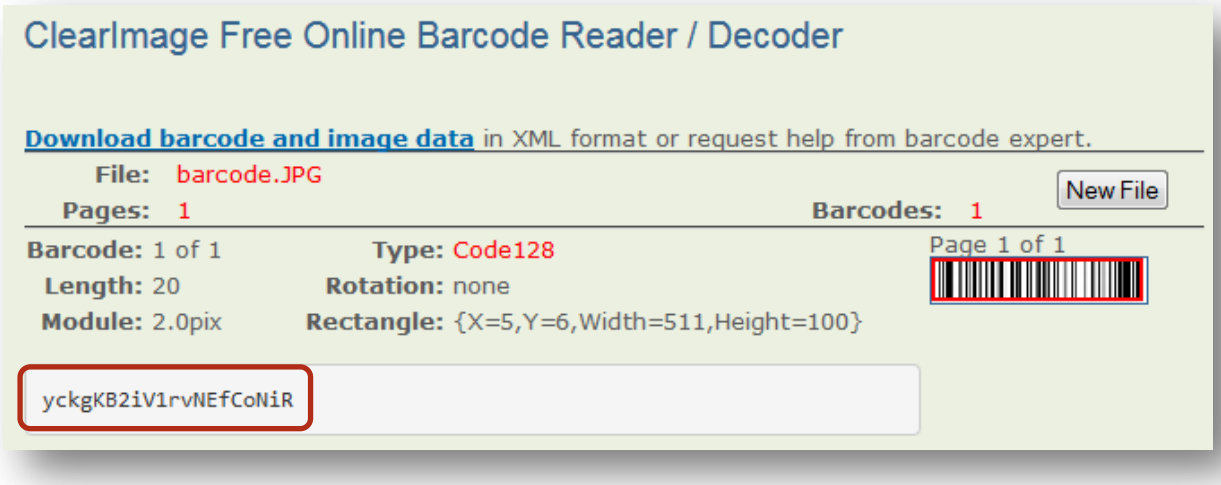
but this sample sounds like its reversed. i used some wav editor (sound forge) to reverse it back and then a voice told me "eight fourty two" –we can also find a hint about reversing it, if we open the wav in a hex editor.

now we can play our video and pause it on the position 8:42 – this is our egg for this challenge.

quite a nice challenge actually. i liked it!

CHALLENGE 14 - WISE RABBIT'S RETURN

we are not getting a QR code this time, but we get a barcode. now what? lets scan this with an online barcode reader:



and now lets just make a QR code from the result

scanning this with the hacky easter app, solved this level straight away.

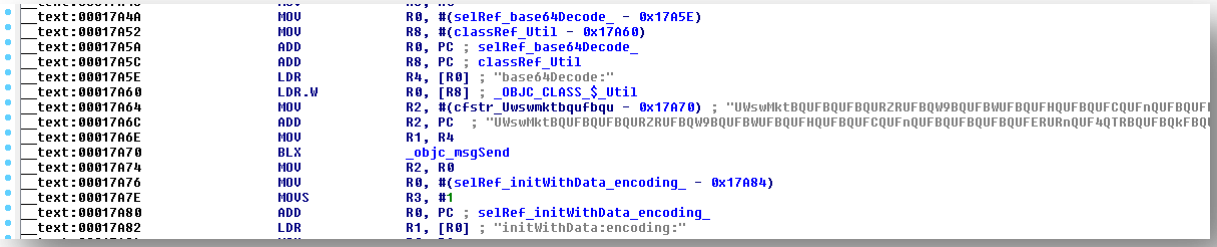
CHALLENGE 15 - PHOTO SHOOTING

in this challenge, we should take some snapshots of something and actually i have found two solutions to find the qr code.

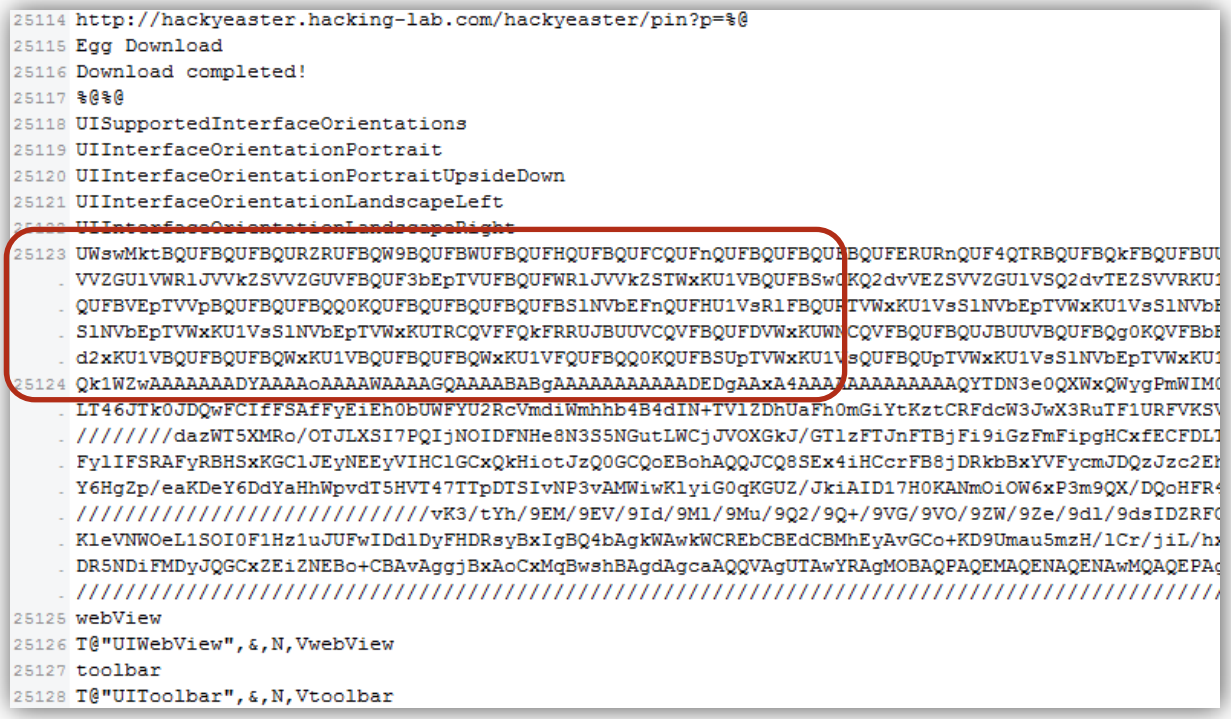
the first – probably the correct way – is to make some pictures with the hacky easter app and trying to get some different colors on the images. from time to time a half QR will appear, but mostly rick astley messes up our pictures. i opened some different colors in google images and captured it until the second half of the qr code appeared. then i used photoshop to merge them.

the other solution is ways better and faster:

since my iphone is jailbroken, i was able to disassemble the hacky easter app and to follow the code. for this challenge i simply ran **strings.exe** on the binary and it gave me a list with some suspicious big base64 encoded strings. i converted them and one string gave the rick astley foto. the other one was double encoded with base64 and resulted in the qr code for this challenge.

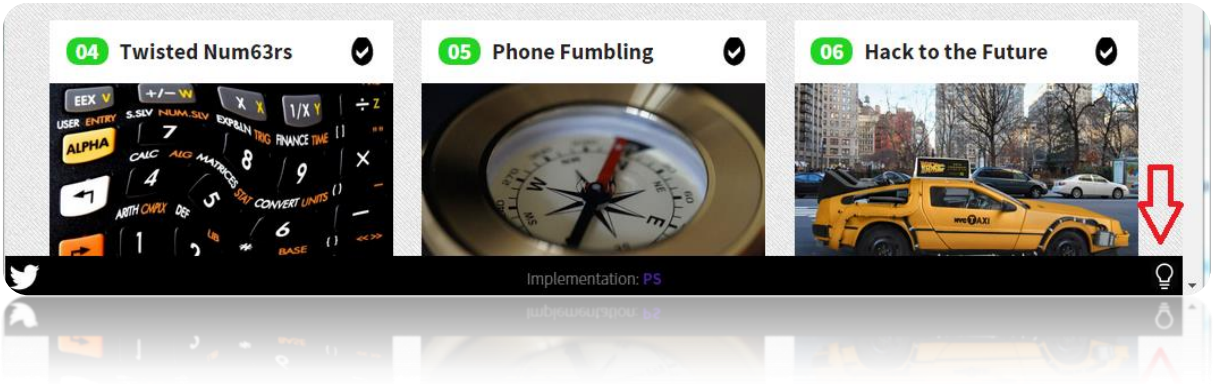


strings output:



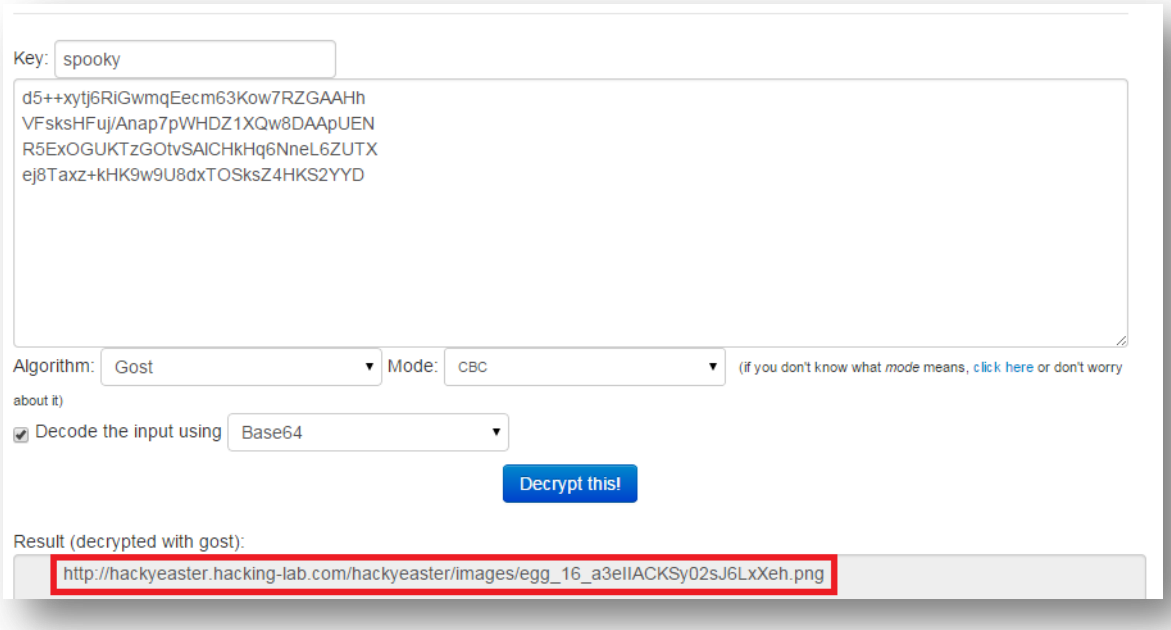
CHALLENGE 16 - GHOST ROOM

on this challenge i was stuck a while, because i simply overlooked the lamp on the challenge page.



i only found this, when i was looking through the .js files from the webpage to solve a another challenge. my eyes simply did not see this. after finding the lamp, and switching to dark mode, a crypto code appeared in the ghost room. also there was a nice hint about GOST – which is a blockcipher.

i just used an online decoder (http://www.tools4noobs.com/online_tools/decrypt/) to quickly decrypt it with the word from the image (spooky) and the egg URL appeared immediately.

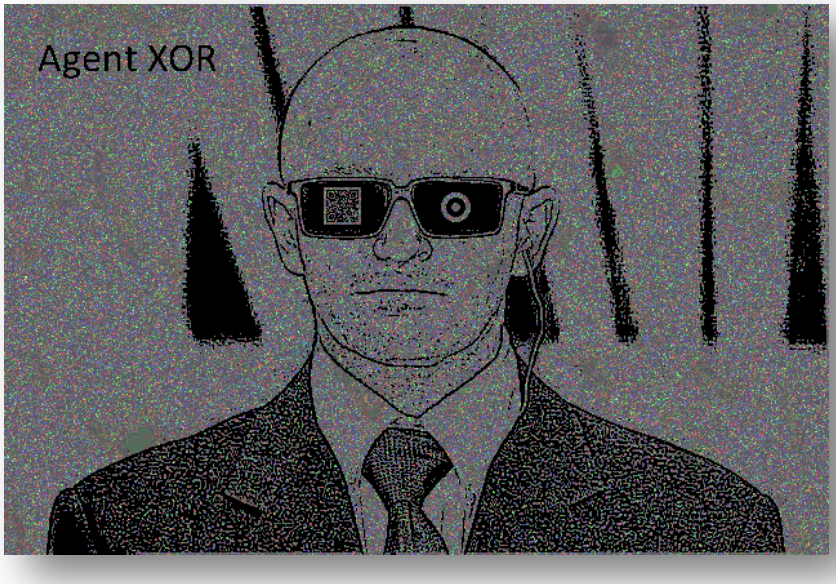


CHALLENGE 17 - SPOT THE DIFFERENCE

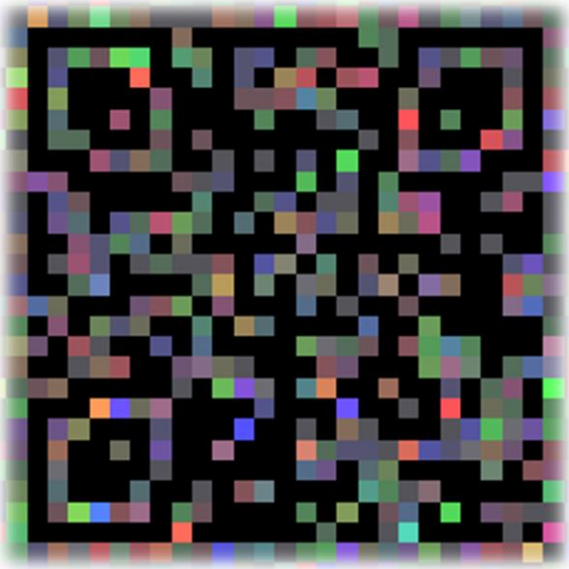
here we are given two images and we have to find the differences. i was stuck here quite some time, because i went the wrong way. first i coded a python script that collected the differences in a new file. actually the bytes only had a difference of "1" in the second file, but not everywhere.

first i thought this is something in binary, but it was not. of course i tried to layer the images in an image editor and i even did the correct thing, but was not able to see the resulting image on the first try. later i just tried it again and the same thing worked. bah.

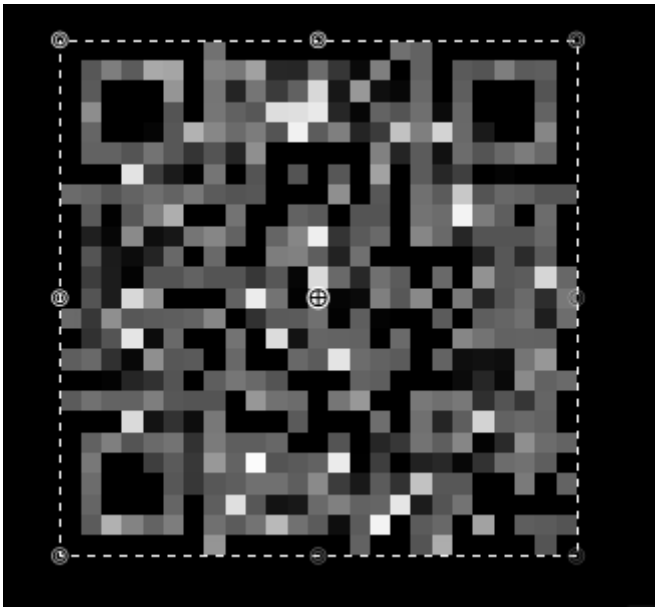
how to fix: open both files in paint.NET and use them as layers. then select layer properties and change mode to XOR. this gave "something" but first i did not recognize it. but when i looked harder, i was able to see a face with glasses in this XOR'ed image! but how can i reveal it completely? or did i maybe some mistake? just to try something, i merged the layers and selected corrections, automatic in paint.NET (ctrl+shift+L) and it revealed the image instantly!



wow. wtf is going on? hello agent xor! this is amazing! and hey there is even a qr code in his glasses. but.. its not scannable. argh.



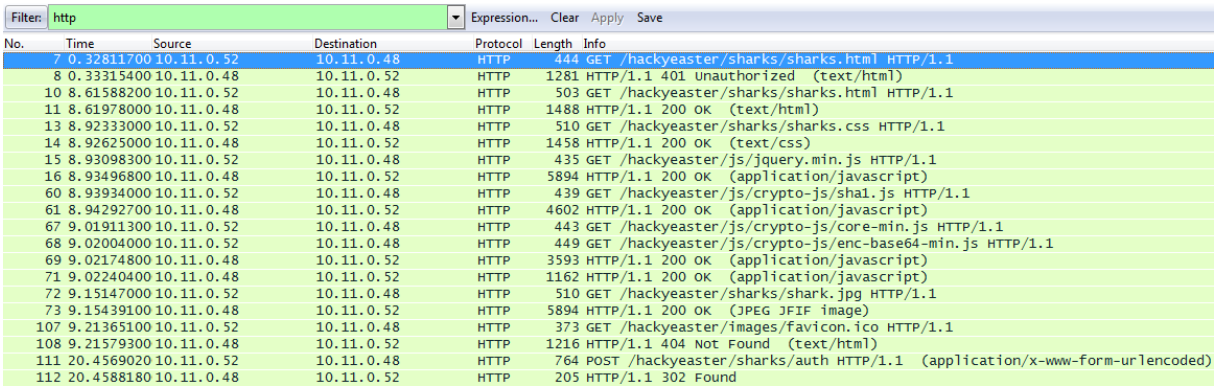
here ive lost some time again, because i tried to correct the qr code manually, but this didnt work. this is somehow modified, but how? how would agent xor modify it? xoring of course but with what? wait.. if we look closely at the qr code, we can see that it contains a round shape of modification. uhh... the other glasses eyes matches this! lets try something. i changed the colors to black and white and copied the qr code over the eye on the image in a new layer, again selecting XOR for the layering. this looked better, but not scanable yet. but then i played with brightness and contrast, set both to the max for both layers and the qr code appeared cristal clear!



this was a very nice challenge!

CHALLENGE 18 - SHARKS ON WIRE

this challenge gives us a pcap file and an address to a webpage. when we try to access it, we are asked for credentials – a typical basic authentication popup. lets see what we can find in the pcap then. first ive put a filter for http which revealed already some interesting info.



No.	Time	Source	Destination	Protocol	Length	Info
7	0.32811700	10.11.0.52	10.11.0.48	HTTP	444	GET /hackyeaster/sharks/sharks.html HTTP/1.1
8	0.33315400	10.11.0.48	10.11.0.52	HTTP	1281	HTTP/1.1 401 Unauthorized (text/html)
10	8.61588200	10.11.0.52	10.11.0.48	HTTP	503	GET /hackyeaster/sharks/sharks.html HTTP/1.1
11	8.61978000	10.11.0.48	10.11.0.52	HTTP	1488	HTTP/1.1 200 OK (text/html)
13	8.92333000	10.11.0.52	10.11.0.48	HTTP	510	GET /hackyeaster/sharks/sharks.css HTTP/1.1
14	8.92625000	10.11.0.48	10.11.0.52	HTTP	1458	HTTP/1.1 200 OK (text/css)
15	8.93098300	10.11.0.52	10.11.0.48	HTTP	435	GET /hackyeaster/js/jquery.min.js HTTP/1.1
16	8.93496800	10.11.0.48	10.11.0.52	HTTP	5894	HTTP/1.1 200 OK (application/javascript)
60	8.93934000	10.11.0.52	10.11.0.48	HTTP	439	GET /hackyeaster/js/crypto-js/sha1.js HTTP/1.1
61	8.94292700	10.11.0.48	10.11.0.52	HTTP	4602	HTTP/1.1 200 OK (application/javascript)
67	9.01911300	10.11.0.52	10.11.0.48	HTTP	443	GET /hackyeaster/js/crypto-js/core-min.js HTTP/1.1
68	9.02004000	10.11.0.52	10.11.0.48	HTTP	449	GET /hackyeaster/js/crypto-js/enc-base64-min.js HTTP/1.1
69	9.02174800	10.11.0.48	10.11.0.52	HTTP	3593	HTTP/1.1 200 OK (application/javascript)
71	9.02240400	10.11.0.48	10.11.0.52	HTTP	1162	HTTP/1.1 200 OK (application/javascript)
72	9.15147000	10.11.0.52	10.11.0.48	HTTP	510	GET /hackyeaster/sharks/shark.jpg HTTP/1.1
73	9.15439100	10.11.0.48	10.11.0.52	HTTP	5894	HTTP/1.1 200 OK (JPEG JFIF image)
107	9.21365100	10.11.0.52	10.11.0.48	HTTP	373	GET /hackyeaster/images/favicon.ico HTTP/1.1
108	9.21579300	10.11.0.48	10.11.0.52	HTTP	1216	HTTP/1.1 404 Not Found (text/html)
111	20.4569020	10.11.0.52	10.11.0.48	HTTP	764	POST /hackyeaster/sharks/auth HTTP/1.1 (application/x-www-form-urlencoded)
112	20.4588180	10.11.0.48	10.11.0.52	HTTP	205	HTTP/1.1 302 Found

by selecting the first entry and then right click, follow tcp stream, we can quickly find the basic auth info:

```
sharks/sharks.html HTTP/1.1
Host: 10.11.0.48:8080
Connection: keep-alive
Authorization: Basic c2hhcmttYW46c2hhcmtzX2hhdmVfajR3cw==
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/39.0.2171.71 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,de;q=0.6,it;q=0.4
```

this is just base64 and decoded gives us a login: sharkman:sharks_have_j4ws

ok, with this info, we can pass the first check but then another login window appears. now this is a form based login and therefore it will post to the webpage. just lets go back to the http filtered wireshark and check for a post request.

108	9.21579300	10.11.0.48	10.11.0.52	HTTP	1216	HTTP/1.1 404 Not Found (text/html)
111	20.4569020	10.11.0.52	10.11.0.48	HTTP	764	POST /hackyeaster/sharks/auth HTTP/1.1
112	20.4588180	10.11.0.48	10.11.0.52	HTTP	205	HTTP/1.1 302 Found

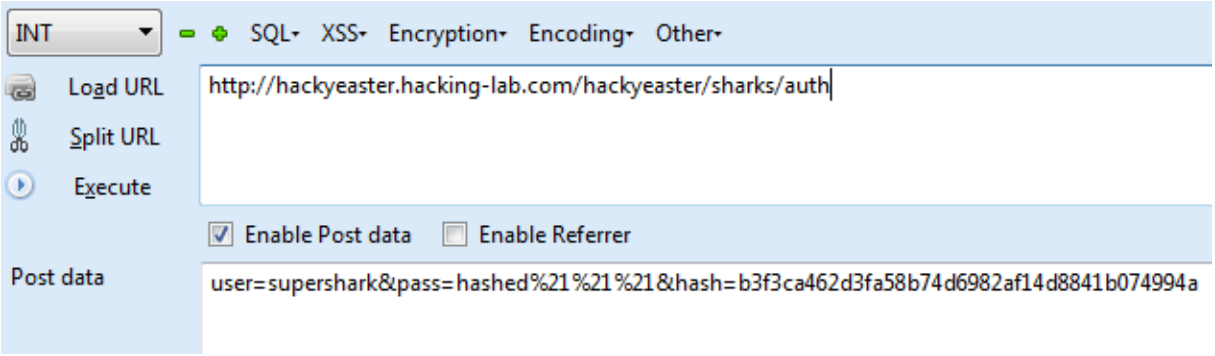
nice, there is one. lets follow this stream. in the end of this stream, we can find some other credentials:

```
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8,de;q=0.6,it;q=0.4

user=supershark&pass=hashed%21%21%
21&hash=b3f3ca462d3fa58b74d6982af14d8841b074994aHTTP/1.1 302 Found
Server: Apache-Coyote/1.1
Location: http://en.wikipedia.org/wiki/Shark
```

there is a hash, but we cannot crack it. no problem, since the hash is made up on the client (clearly visible in the page source code) we can simply resubmit it to login successfully.

```
<form action="auth" method="post" onsubmit="$('#hash').val(CryptoJS.SHA1($('#pass').val()));$('#pass').val('hashed!!!');">
  <input class="input" type="text" name="user" placeholder="User"/>
  <input class="input" type="password" name="pass" id="pass" placeholder="Password"/>
  <input class="input" type="hidden" name="hash" id="hash" />
  <input class="button" type="submit" value="Dive in"/>
</form>
```

this is another case for the hackbar addon in firefox.



just execute this and the egg for this level appears.

CHALLENGE 19 - CUT'N'PLACE

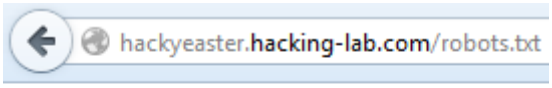
this challenge was the latest for me to solve. i knew that i was overcomplicating it and also i had the feeling, that it must contain certain words from the image. i have tried some online anagram solvers and other stuff, but nothing helped. actually it was rather easy, but it was non-digital and therefore somehow different thinking.

the main idea which helped me, was to arrange the paper strips in a way, that there no special characters appeared anymore. from there it was easy and when i saw the first word "paper", it was easy to finish. here is the correct solution:

paperstrips made by shredder. oh dear.

CHALLENGE 20 - LOTS OF BOTS

think like a bot – this message already gives a hint about the robots.txt file from the hackyeaster webpage. lets check it out:




```
User-agent: EasterBot
Disallow: /
Allow: /hackyeaster/bots/bots.

User-agent: *
Disallow: /
```

so there is some hidden path, but no file extension. the most logic thing was to try .html and other webpage formats on this path. this somewhat worked, but i always got redirected to wikipedia – even faking my user-agent to EasterBot.

then i checked with live http headers addon in firefox, but i didnt see any 302 redirection. this made me think, that there is no serverside redirection. hmm... better lets download the file with wget or curl then.

```
Resolving hackyeaster.hacking-lab.com... 212.254.178.171
Connecting to hackyeaster.hacking-lab.com|212.254.178.171|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 922 [text/html]
Saving to: `bots.html'

100%[======================================>] 922         --.-K/s   in 0s

2015-04-10 17:23:27 (94.8 MB/s) - `bots.html' saved [922/922]
```

now the source code contained some javascript fun:

```
<head>
  <title>Bots</title>
  <script type="text/javascript">
    eval(String.fromCharCode(105, 102, 32, 40, 33, 40, 110, 97, 118, 105,
  </script>
</head>
<body style="background: white; border: 20px solid white;">
  <div style="widht: 100%; height: 100%; background: url('./robotbg.jpg')
```

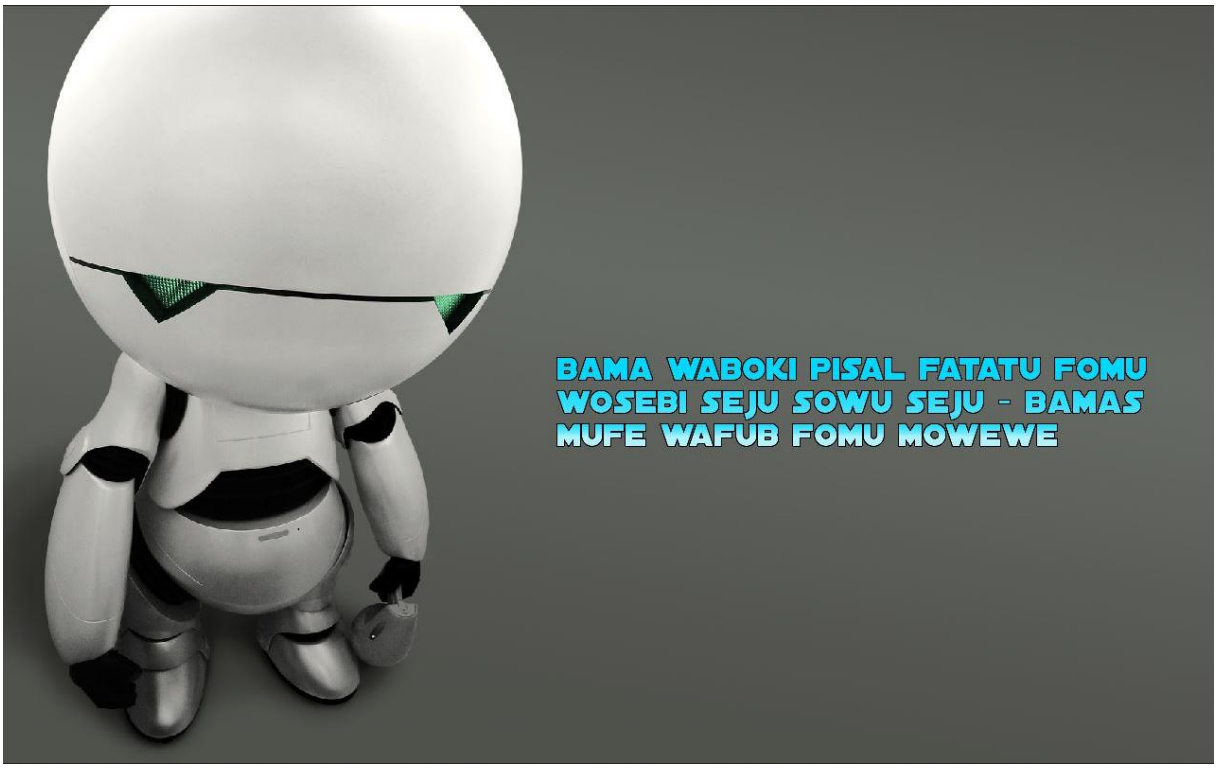
this eval executes:

if (!(navigator.userAgent === 'EasterBot')) { location.replace('http://en.wikipedia.org/wiki/C-3PO');}

ok now we know why the fake agent didnt work. the user agent faker do only change the agent in a request to the remote webpage, but not locally and javascript checks it without needing a request – its client side executed.

but wait.. there is more.

what about this **robotbg.jpg**?



what? i dont understand a word at all. but after some googling it turned out to be roila – a robot interaction language. haha very funy. with the help of a vocabulary (http://roila.org/language-guide/vocabulary/) i was able to translate it to:

you must make word of addition two and two - this be name of page

ok so we need to open a page with the name four.html – again with wget i downloaded it and checked the source code – else i would have been redirected again.

```
<head>
  <title>Bots</title>
  <meta name="description" content="Robots talk in ROILA language: eman egap eht esrever tsum">
  <meta name="keywords" content="secret, page, robots, fun, hacky easter, blrt, five, beep">
  <script type="text/javascript">
    eval(String.fromCharCode(105, 102, 32, 40, 33, 40, 110, 97, 118, 105, 103, 97, 116, 111, 114, 46, 117,
  </script>
</head>
<body style="background: white; border: 20px solid white;">
  <div style="widht: 100%; height: 100%; background: url('./robotbg2.jpg') no-repeat center center fixed;
```

a new image appeared but this time with no helping message. but wait... there is something in meta name:

Robots talk in ROILA language: eman egap eht esrever tsum

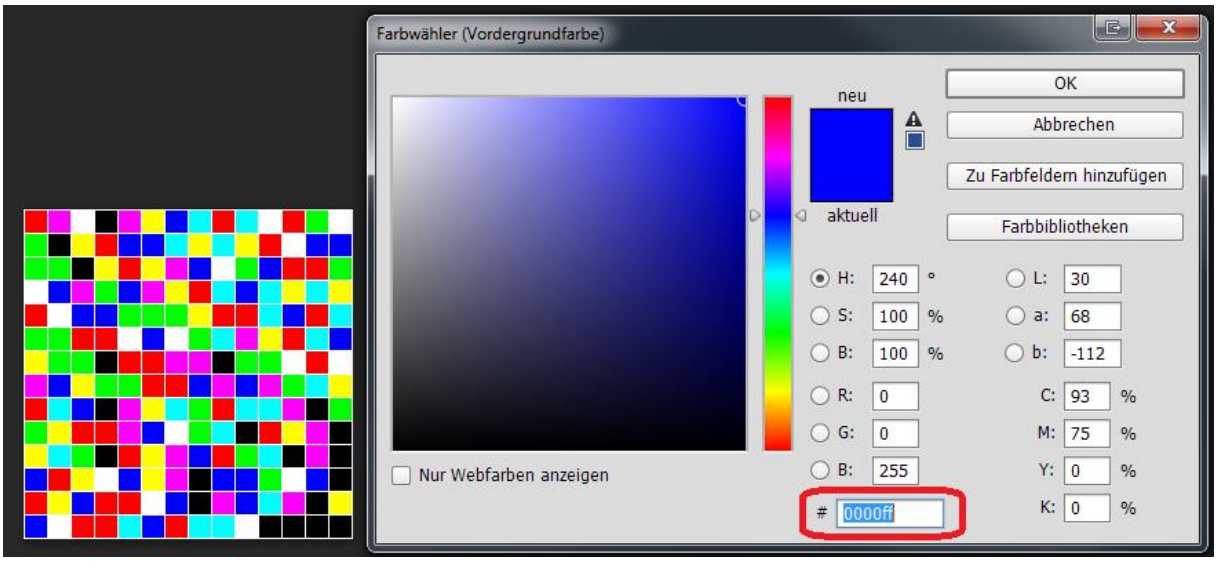
not a robot language anymore. this is just reversed: **must reverse the page name**

ok so i downloaded ruof.html (first didnt work, because i made a typo) with wget and checked the source code. this time we can find the egg for this level:

http://hackyeaster.hacking-lab.com/hackyeaster/bots/egg_20_j5fir8U6g8.png

CHALLENGE 21 - CONY CODE

this challenge took me some time to understand. not because its very difficult, but i did not get the idea in the beginning. i noticed that the colors all are pretty straight – means always using maximum range of RGB values.



blue: 00 00 ff
red: ff 00 00
yellow: ff ff 00

and so on. we are given only one hint: blue is 110 – this obviously is binary but it took some time until i realized, that its the same number, but the other way around. means FF = 0 and 00 = 1

110 = 00 00 ff
001 = ff ff 00

using this scheme we can now decode the image – from top left to the right and then each line. if we check each 17th pixel, we will hit each color once. this time my language of choice is vb.net.

```
Dim myBitmap As New Bitmap("C:\Users\administrator\Desktop\hackyeaster\conycode.bmp")

For y = 1 To myBitmap.Height - 1 Step 17
    For x = 1 To myBitmap.Width - 1 Step 17
        Dim pixelColor As Color = myBitmap.GetPixel(x, y)
        Dim bits = 0
        If pixelColor.R = 255 Then
            bits = bits Or Convert.ToInt32("100", 2)
        End If

        If pixelColor.G = 255 Then
            bits = bits Or Convert.ToInt32("010", 2)
        End If

        If pixelColor.B = 255 Then
            bits = bits Or Convert.ToInt32("001", 2)
        End If

        bits = (Not bits) And 7
        TextBox1.Text += Convert.ToString(bits, 2).PadLeft(3, "0")
    Next
Next
```

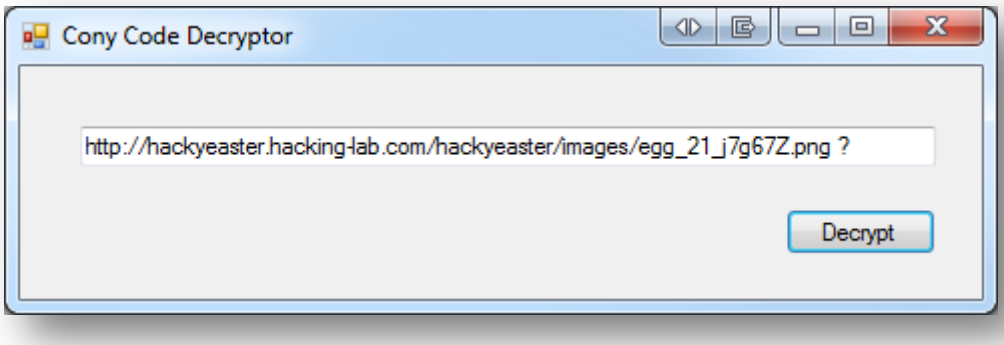
the result of this color converter gave me this binary string:

```
01101000011101000111010001110000001110100010111100101111011010000110000101100011011010110111
11001011001010110000101110011011101000110010101110010001011100110100001100001011000110110101
11011010010110111001100111001011010110110001100001011000100010111001100011011011111011011010
01011110110100001100001011000110110101101111001011001010110000101110011011101000110010101111
00100010111101101001011011010110000010110011101100101011100110010111011001010110011101100111
10101111100110010001100010101111101101010001101110110011100110110001101110101101000101110011
11000001101110011001110010000011111111111111
```

then i added some super leet converter code to my application and converted the binary to ascii.

```
Dim myBitmap As New Bitmap("conycode.bmp")
Dim binary = ""

For y = 1 To myBitmap.Height - 1 Step 17
    For x = 1 To myBitmap.Width - 1 Step 17
        Dim pixelColor As Color = myBitmap.GetPixel(x, y)
        Dim bits = 0
        If pixelColor.R = 255 Then
            bits = bits Or Convert.ToInt32("100", 2)
        End If

        If pixelColor.G = 255 Then
            bits = bits Or Convert.ToInt32("010", 2)
        End If

        If pixelColor.B = 255 Then
            bits = bits Or Convert.ToInt32("001", 2)
        End If

        bits = (Not bits) And 7
        binary += Convert.ToString(bits, 2).PadLeft(3, "0")
    Next
Next

Dim BinaryText As String = binary
Dim Characters As String = Regex.Replace(BinaryText, "[^01]", "")
Dim ByteArray((Characters.Length / 8) - 1) As Byte
For Index As Integer = 0 To ByteArray.Length - 1
    ByteArray(Index) = Convert.ToByte(Characters.Substring(Index * 8, 8), 2)
Next
TextBox1.Text = (ASCIIEncoding.ASCII.GetString(ByteArray))
```

and here is the solution:



CHALLENGE 22 - HASHES TO ASHES

in this challenge we are given some hashes and we have to crack them. we dont know the exact hash algorithm used, but we can guess it since it can only be MD5 or SHA.

i have used crypttool to convert these hashes into hex string format. then i have used cudaHashcat64 to actually crack them. here is how i did it for the different hashes:

1. this hash has 160bits and must therefore be SHA1. in hashcat i have used a custom charset, since the hint tells us that not all numbers are used. the commandline that i have used was: ***--hash-type 100 --attack-mode 3 --custom-charset1 0179 ?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1*** and the solution is ada2eeebe7809857a57f6fee4b2ffaee24eae7b1:**1199019170177790**
2. this hash has 383bits and we can assume it is SHA384. i did not understand the hint, but using some wordlists solved it for me quite quickly. the commandline was just ***--hash-type 10800*** with a wordlist (not the given one) and the file with the hash. the solution is 6bbf7528d9dd2959a7afb37898425f67555f67f677987cae7e86210a2c8a0dbdfc248ec2d7b24010f440badc2223b4b5:**hopelessly**
3. the third hash has 128bit and must be MD5. the hint tells us about the format which is not so easy to reproduce in hashcat but we have a wordlist and can modify it to make it better. for that i have used john with rules (john --wordlist=wordlist.txt --rules --stdout | unique mangled.lst) and then another rulefile in hashcat. the syntax was ***--rules-file best64.rule hash3_md5.txt mangled_wordlist.txt*** and the result is b80814c5e0f386b0637163fd8afea929:**Disc0very.5**
4. the last hash has 248bits and must be SHA256. we know that this is a long word made up from four words out of the wordfile. to prepare some good wordlist, we can use the hashcat utils (https://hashcat.net/wiki/doku.php?id=hashcat_utils) which offer a combinator.exe. with this i made a combined wordlist from the given one. then i have run a combiner attack in hashcat with the combined wordlist. syntax: ***--hash-type 1400 --attack-mode 1 hash4_sha256.txt combined_wordlist.txt combined_wordlist.txt*** and the solution is: 9791cbe0ae919a0330994a2d6ba26b8f0c3a1da15c73bce5fca39495881a6c90:**enginebulbgoatimportant**

actually all hashes were cracked really fast (just some seconds) using the correct preparations and tools. now i had all hashes cracked and entering the solutions on the webpage showed me the egg for level 22.

CHALLENGE 23 - BEAT THE NERD MASTER

this challenge asks us to connect to hackyeaster.hacking-lab.com on port 1400 and send some insults like in the classic monkey island game. we are given an example insult and we can start to collect all questions and answers.

i made up a little python script first, that connected a few times and sent always the same insult. then i saved the insults from the server in a text file. after that, i sent all collected insults and saved all corresponding answers.

now i had all possible combinations and just needed to write a code, that sends random insults and answers with the correct phrases when the server asks for it.

my python solution was not so good, but i managed to get the qr code with it.

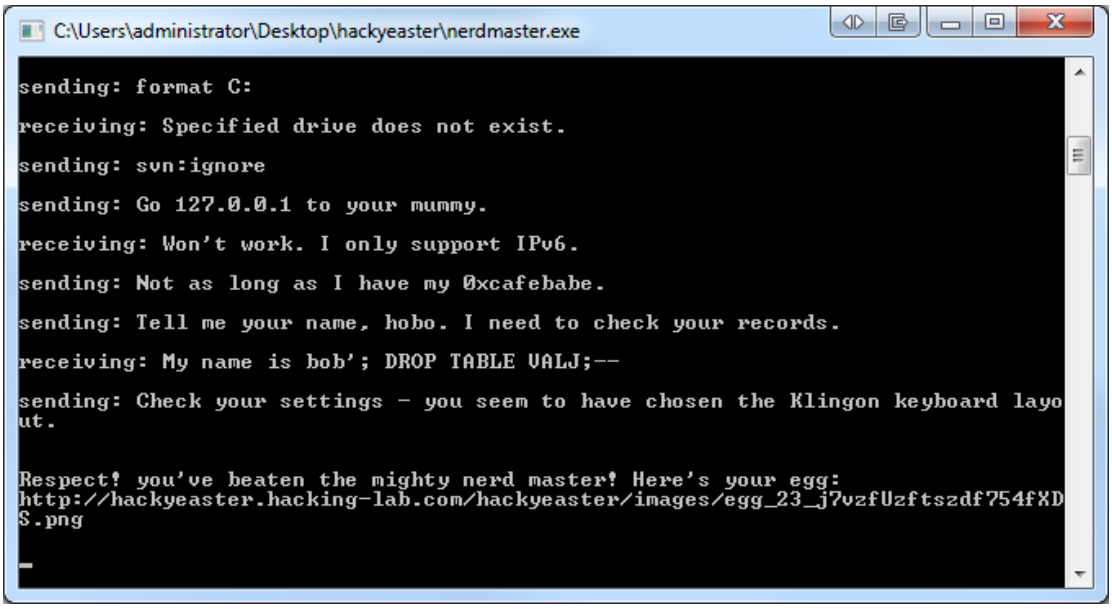
just because i was not satisfied and i wanted to try something different, i recoded the solution in C#. i have used a dictionary to lookup the combinations and after using one, my code removes it, because the server doesnt like the same insults more than once.

here is the dictionary with all insults:

```
Dictionary<string, string> insults =  new Dictionary<string, string>();

insults.Add("You'll be 0xdeadbeef soon.", "Not as long as I have my 0xcafebabe.");
insults.Add("Ping! Anybody there?", "ICMP type 3, code 13: Communication Administratively Prohibited");
insults.Add("format C:", "Specified drive does not exist.");
insults.Add("I'll check you out - any last words?", "svn:ignore");
insults.Add("I bet you don't even understand binary.", "Sure I do. Me and you, we are 10 different kind of persons.");
insults.Add("Go 127.0.0.1 to your mummy.", "Won't work. I only support IPv6.");
insults.Add("Tell me your name, hobo. I need to check your records.", "My name is bob'; DROP TABLE VALJ;--");
insults.Add("Af7ter th1s f1gh7, I w1ll pwn ur b0x3n.", "Check your settings - you seem to have chosen the Klingon keyboard layout.");
insults.Add("Pna lbh ernq guvf?", "EBG13 vf sbe ynzref.");
insults.Add("You're so slow, you must have been written in BASIC.", "At least I don't have memory leaks like you.");
insults.Add("You should leave your cave and socialize a bit.", "I'm not anti-social. I'm just not user friendly.");
insults.Add("You must be jealous when seeing my phone's display.", "Not really - Your pixels are so big, some of them have their own region code!");
insults.Add("After loosing to me, your life won't be the same anymore.", "A Life? Cool! Where can I download one of those?");
insults.Add("This fight is like a hash function - it works in one direction only.", "Too bad you picked LM hashing.");
insults.Add("I have more friends than you.", "Yeah, but only until you update your Facebook profile with a real picture of you!");
insults.Add("1f u c4n r34d th1s u r s70p1d.", "You better check your spelling. Stoopid has two 'o's.");
```

and here is the C# code that beats the nerd master every time:

```
byte[] data = new byte[1024];
string input, stringData;
TcpClient server = new TcpClient("hackyeaster.hacking-lab.com", 1400);          // connect to the server
NetworkStream ns = server.GetStream();
stringData = Encoding.ASCII.GetString(data, 0, ns.Read(data, 0, data.Length));  // get server answer
ns.Write(Encoding.ASCII.GetBytes("y\n"), 0, 2);                                 // send "y"
stringData = Encoding.ASCII.GetString(data, 0, ns.Read(data, 0, data.Length));  // get server answer

while (ns.CanRead)
{
    byte[] myReadBuffer = new byte[1024];
    StringBuilder myCompleteMessage = new StringBuilder();
    int numberOfBytesRead = 0;
    do{
        numberOfBytesRead = ns.Read(myReadBuffer, 0, myReadBuffer.Length);
        myCompleteMessage.AppendFormat("{0}", Encoding.ASCII.GetString(myReadBuffer, 0, numberOfBytesRead));
    }while (ns.DataAvailable);
    string answer = myCompleteMessage.ToString();

    if (answer.Contains("---- YOUR TURN ----")){
        List<string> keys = new List<string>(insults.Keys);                     // make directory to a list
        Random rand = new Random();                                             // get a random number
        string randomKey = keys[rand.Next(insults.Count)];                      // select a random insult

        input = randomKey + "\n";
        insults.Remove(randomKey);                                              // remove used insult
        Console.WriteLine("sending: " + input);
        ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

        stringData = Encoding.ASCII.GetString(data, 0, ns.Read(data, 0, data.Length));
        Console.WriteLine("receiving: " + stringData);
    }
    else if (answer.Contains("---- MY TURN ----")){
        string response = answer.Replace("---- MY TURN ----", "").Replace("\n","");
        input = insults[response]+"\n";                                         // get answer from the dictionary
        insults.Remove(response);                                               // remove it, to stay unique
        Console.WriteLine("sending: " + input);
        ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);
    }
    else if (answer.Contains("You loose!")){
        Console.WriteLine(answer);
        return;
    }
    else if (answer.Contains("Respect!")){
        Console.WriteLine(answer);
        return;
    }
```



CHALLENGE 24 - SHAM HASH

Crypto Chiefs Ltd developed a pretty weak hash algorithm and we should try to find a string for the given hash. the weakness is obviously: the hash length is reduced and therefore collisions can be found rather quickly. this time im doing some python, because pycrypto offers us all needed hash algorithms and itertools some good bruteforce stuff.

```
import itertools
import string

from Crypto.Hash import MD2,MD5,SHA,SHA256,SHA512

def bruteforce(charset, maxlength):
    return (''.join(candidate)
        for candidate in itertools.chain.from_iterable(itertools.product(charset, repeat=i)
        for i in range(6, maxlength + 1)))

sham="757c479895d6845b2b0530cd9a2b11"
print "we are going to crack SHAM hash now..."

found_md2,found_md5,found_sha1,found_sha256,found_sha512=0,0,0,0,0

for attempt in bruteforce(string.letters+string.digits, 10):

    if found_md2==0:
        m = MD2.new()
        m.update(attempt)
        md2=m.hexdigest()
        if md2[0:6]==sham[0:6]:
            print "found: " + attempt + ":" + md2 + " - (MD2)"
            found_md2=1
            s_md2=attempt

    if found_md5==0:
        m = MD5.new()
        m.update(attempt)
        md5=m.hexdigest()
        if md5[6:12]==sham[6:12]:
            print "found: " + attempt + ":" + md5 + " - (MD5)"
            found_md5=1
            s_md5=attempt

    if found_sha1==0:
        m = SHA.new()
        m.update(attempt)
        sha1=m.hexdigest()
        if sha1[12:18]==sham[12:18]:
            print "found: " + attempt + ":" + sha1 + " - (SHA1)"
            found_sha1=1
            s_sha=attempt

    if found_sha256==0:
        m = SHA256.new()
        m.update(attempt)
        sha256=m.hexdigest()
        if sha256[18:24]==sham[18:24]:
            print "found: " + attempt + ":" + sha256 + " - (SHA256)"
            found_sha256=1
            s_sha256=attempt

    if found_sha512==0:
        m = SHA512.new()
        m.update(attempt)
        sha512=m.hexdigest()
        if sha512[24:30]==sham[24:30]:
            print "found: " + attempt + ":" + sha512 + " - (SHA512)"
            found_sha512=1
            s_sha512=attempt

    if found_md2+found_md5+found_sha1+found_sha256+found_sha512==5:
        print "SHAM Hash cracked: "+s_md2+s_md5+s_sha+s_sha256+s_sha512
        break
```

```
hackyeaster — bash — 96×37
bash-3.2$ python sham.py
we are going to crack SHAM hash now...
found: aadH2Y:55b1af1c4493ce17490530cd2d9a6bf09824b2494d65f089164ce4c0c2b9f880 - (SHA256)
found: aaqidt:46e0109895d6278b858d2161a53d6163 - (MD5)
found: aaIKK8:2cb6175aa711729dfa407d289a2b118625c06da2a6bfb5b9bfe36639926b40e11733ae6e976fc1b225
99ca8d0d7cb90bef7fc6cf018399bdec9edd7d04dcd4d8 - (SHA512)
found: abaE0H:757c4766c3342421873600fdc28b773c - (MD2)
found: adEVMd:d5d9235d4132845b2b9119298a5a1328ebf3e85f - (SHA1)
SHAM Hash cracked: abaE0HaaqidtadEVMdaadH2YaaIKK8
bash-3.2$
```

CHALLENGE 25 - JAD & IDA

finally a reverse engineering challenge! but it also needs some java reversing. bah. luckily its not very complicated and we can - like the title suggests - use our well known tools IDA and JAD. actually i have used jd-gui (http://jd.benow.ca/) to decompile the java code.

```
dll = LizzleDLL.INSTANCE;
System.out.println("Enter the key: ");
System.out.flush();
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
String k = in.readLine();
String h = k;
for (int z = 0; z < 10; z++) {
  h = fizzle(rizzle(shizzle(bizzle(h))));
}
if ("v3O] pmWm<Y(0=21".equals(h)) {
  System.out.println("Congrats!");
  byte[] plain = Files.readAllBytes(Paths.get("s3cr3t.bin", new String[0]));
  byte[] cipher = decrypt(k.getBytes(), plain);
  Files.write(Paths.get("eggizzle_25.png", new String[0]), cipher, new OpenOption[0]);
} else {
  System.out.println("nope!");
```

from here we can see, that it loads a DLL and runs some fancy functions in a loop on our input. then it compares it to a hardcoded string and if matched, decrypts the file s3cr3t.bin. the functions Shizzle and Fizzle are in the DLL and we have to check them with IDA. Shizzle actually just looks like a string reversing function and Fizzle does some different sort of thing.

```
.text:65C01370 loc_65C01370:                           ; CODE XREF: _Fizzle+66↓j
.text:65C01370                 mov     edx, ecx
.text:65C01372                 imul    edx, ecx
.text:65C01375                 lea     ebp, [eax+edx-1Bh]
.text:65C01379                 mov     eax, ebp
.text:65C0137B                 imul    edi
.text:65C0137D                 mov     eax, ebp
.text:65C0137F                 sar     eax, 1Fh
.text:65C01382                 add     edx, ebp
.text:65C01384                 sar     edx, 6
.text:65C01387                 sub     edx, eax
.text:65C01389                 imul    edx, 5Bh
.text:65C0138C                 sub     ebp, edx
.text:65C0138E                 lea     eax, [ebp+20h]
.text:65C01391                 mov     [ebx+ecx], al   ; save resulting byte (char)
.text:65C01394                 add     ecx, 1
.text:65C01397                 cmp     ecx, 10h
.text:65C0139A                 jz      short loc_65C013B3
```

we can maybe reverse this math or we simply notice, that it does the "encryption" byte for byte and not in a block. this means, we can send single bytes into the algo and compare the result with the hardcoded string. this makes a bruteforce attack in a loop pretty easy. this time i have chosen assembly to solve this challenge because its easy to load the dll and call the functions without knowing what they really do. yay!

first i have recoded the two functions from the java code. good for me that they were not so advanced.

rizzle just converts the case of the characters – like – if lowercase, make it upper and the other way around.

bizzle increments each character of the string until it it reaches "Z" or "z" and then switches it back to "A" or "a".

in assembly these functions look like this (not optimized, just recoded from the java snippet)

```
bizzle  PROC, str_in:DWORD ;increment characters
    mov     esi, str_in
    .WHILE !byte ptr [esi]==0
        .IF byte ptr [esi]>="a" && byte ptr [esi]
            inc byte ptr [esi]
        .ELSEIF byte ptr [esi]=="z"
            mov byte ptr [esi], "a"
        .ELSEIF byte ptr [esi]>="A" && byte ptr [
            inc byte ptr [esi]
        .ELSEIF byte ptr [esi]=="Z"
            mov byte ptr [esi], "A"
        .ENDIF
        inc esi
    .ENDW
    ret
bizzle  ENDP
```

```
rizzle  PROC, str_in:DWORD ;invert case
    mov     esi, str_in
    .WHILE !byte ptr [esi]==0
        .IF byte ptr [esi]>="A" && byte ptr [esi]<="Z"
            add byte ptr [esi], 20h
        .ELSEIF byte ptr [esi]>="a" && byte ptr [esi]<="z"
            sub byte ptr [esi], 20h
        .ENDIF
        inc esi
    .ENDW
    ret
rizzle  ENDP
```

now i just made up a buffer with characters lower than "0". then i increment the character at the desired position until it matches the encrypted character at the same index.

```
findchar    PROC    index:DWORD ; try to get a char for given index
uses edi
    mov eax, 2fh
    mov ecx, 16
    lea edi, szInput
    repz stosb ; init buffer with "0" - 1
    mov edi, index
    mov al, byte ptr [szEncrypted+edi]
    .WHILE !byte ptr [szInput+edi]==al ; check if byte matched
        mov ecx, 16
        lea edi, szInput
        lea esi, szStart
        repz movsb ; init brute buffer
        mov dwCount, 0
        .WHILE !dwCount==10 ; loop 10 times
            call    bizzle, offset szInput
            call    Shizzle, offset szInput, offset szOutput
            add     esp, 2*4 ; adjust stack
            call    rizzle, offset szOutput
            call    Fizzle, offset szOutput, offset szInput
            add     esp, 2*4 ; adjust stack
            inc     dwCount
        .ENDW
        mov edi, index
        inc byte ptr [szStart+edi] ; increment character
        mov al, byte ptr [szEncrypted+edi]
    .ENDW
    mov edi, index
    movzx eax, byte ptr [szStart+edi]
    dec eax
    ret
findchar    ENDP
```

then i just needed some code to load the dll and get the function offsets. a little loop to scan all 16 characters and a little messagebox to display the result. just for easier handling, my solution copies the result in the clipboard after closing the popup message.

```
call    LoadLibrary, offset szLibrary
mov     hLib, eax
call    GetProcAddress, hLib, offset szShizzle
mov     Shizzle, eax
call    GetProcAddress, hLib, offset szFizzle
mov     Fizzle, eax
xor edi, edi
.WHILE !edi==16
    call    findchar, edi
    mov byte ptr [szResult+edi], al
    inc edi
.ENDW
call    MessageBox, NULL, offset szResult, offset szFound, MB_ICONINFORMATION
call    copy, offset szResult ; copy text to the clipboard
call    ExitProcess, NULL

copy    PROC input:DWORD
LOCAL hMem:DWORD, pMem:DWORD, sLen:DWORD
    call lstrlen, input
    mov sLen, eax
    inc eax
    call GlobalAlloc, GMEM_MOVEABLE+GMEM_DDESHARE+GMEM_ZEROINIT, eax
    mov hMem, eax
    call GlobalLock, hMem                       ; lock memory
    mov pMem, eax
    call lstrcpyn, pMem, input, sLen
    call OpenClipboard, NULL                    ; open clipboard
    call EmptyClipboard
    call SetClipboardData, CF_TEXT, pMem         ; write data to it
    call CloseClipboard                        ; close clipboard
    call GlobalUnlock, hMem                     ; unlock memory
    call GlobalFree, hMem                       ; deallocate memory
    ret
copy    ENDP
```

and here we go:

oh yeah.. this is borland tasm 5 code – oldschool but still working on modern OS!

CHALLENGE 26 - CLUMSY CLOUD

this is supposed to be a mobile challenge, but actually i was not able to make up something from the info that i have got on my mobile phone. we can see the key and we also find out the crypto parameters when looking up the numbers from "h" and "e" parameters in the passphrase_backup.txt (AES, SHA1). but we dont really know how this all works together.

in my case i did disassemble the app (iOS) in IDA to understand how it really works. to do that, i had to dump the binary on my iphone with Clutch. this is only possible on jailbroken iOS.

```
__text:000171F2                 MOV             R3, #(selRef_keyFromPin_andSalt_andIterations_ - 0x171FE)
__text:000171FA                 ADD             R3, PC ; selRef_keyFromPin_andSalt_andIterations_
__text:000171FC                 LDR             R1, [R1]
__text:000171FE                 STR             R1, [SP,#0x38+var_1C]
__text:00017200                 LDR             R1, [R3] ; "keyFromPin:andSalt:andIterations:"
__text:00017202                 MOV             R3, #(cfstr_Ovaederecumsal - 0x17210) ; "ovaederecumsale"
__text:0001720A                 STR             R6, [SP,#0x38+var_38]
__text:0001720C                 ADD             R3, PC  ; "ovaederecumsale"
__text:0001720E                 BLX             _objc_msgSend
__text:00017212                 MOV             R4, R0
__text:00017214                 MOV             R0, #(selRef_base64Decode_ - 0x17228)
__text:0001721C                 MOV             R5, #(classRef_Util - 0x1722A)
__text:00017224                 ADD             R0, PC ; selRef_base64Decode_
__text:00017226                 ADD             R5, PC ; classRef_Util
__text:00017228                 LDR             R1, [R0] ; "base64Decode:"
__text:0001722A                 LDR             R0, [R5] ; _OBJC_CLASS_$_Util
__text:0001722C                 MOV             R2, #(cfstr_8qendedkspv61i - 0x17238) ; "8QeNdEdkspV6+1I77SEEEF4aWs5dl/auahJ46MMufkg="
__text:00017234                 ADD             R2, PC  ; "8QeNdEdkspV6+1I77SEEEF4aWs5dl/auahJ46MMufkg="
__text:00017236                 BLX             _objc_msgSend
__text:0001723A                 MOV             R2, R0
__text:0001723C                 MOV             R0, #(selRef_aesDecrypt_key_ - 0x1724A)
__text:00017244                 MOV             R3, R4
__text:00017246                 ADD             R0, PC ; selRef_aesDecrypt_key_
__text:00017248                 LDR             R1, [R0] ; "aesDecrypt:key:"
__text:0001724A                 LDR             R0, [R5] ; _OBJC_CLASS_$_Util
__text:0001724C                 BLX             _objc_msgSend
```

later we can see, that it does download a egg_26.png from the hackyeaster server, but we have to decrypt this stuff to get the exact path. recoding this in objective-c is not an easy task and also translating it to python or some other language is not trivial. i dont own an android device, but i actually used an apk downloader to have a look at the android binary (http://apk-dl.com) – android stuff is based on java and we can get a better overview of whats going on there.

to decompile the apk i have used AndroChef decompiler from this website: http://www.neshkov.com/ac_decompiler.html

i was able to find the same code from the iOS binary in the Activity.java file:

```
private int a(String var1, Context var2) {
   try {
      SecretKeySpec var5 = new SecretKeySpec(a(var1, "ovaederecumsale", 10000), "AES");
      Cipher var3 = Cipher.getInstance("AES");
      var3.init(2, var5);
      String var7 = new String(var3.doFinal(Base64.decode("8QeNdEdkspV6+1I77SEEEF4aWs5dl/auahJ46MMufkg=", 0)));
      DownloadManager var6 = (DownloadManager)this.getSystemService("download");
      Request var8 = new Request(Uri.parse("http://hackyeaster.hacking-lab.com/hackyeaster/pin?p=" + var7));
      var8.setTitle("Hacky Easter");
      var8.setDescription("Egg Download");
      var8.setDestinationInExternalPublicDir(Environment.DIRECTORY_DOWNLOADS, "egg_26.png");
      this.registerReceiver(new d(this), new IntentFilter("android.intent.action.DOWNLOAD_COMPLETE"));
      var6.enqueue(var8);
      Toast.makeText(var2, "Download started", 0).show();
      return 0;
   } catch (Exception var4) {
      return 1;
   }
}
```

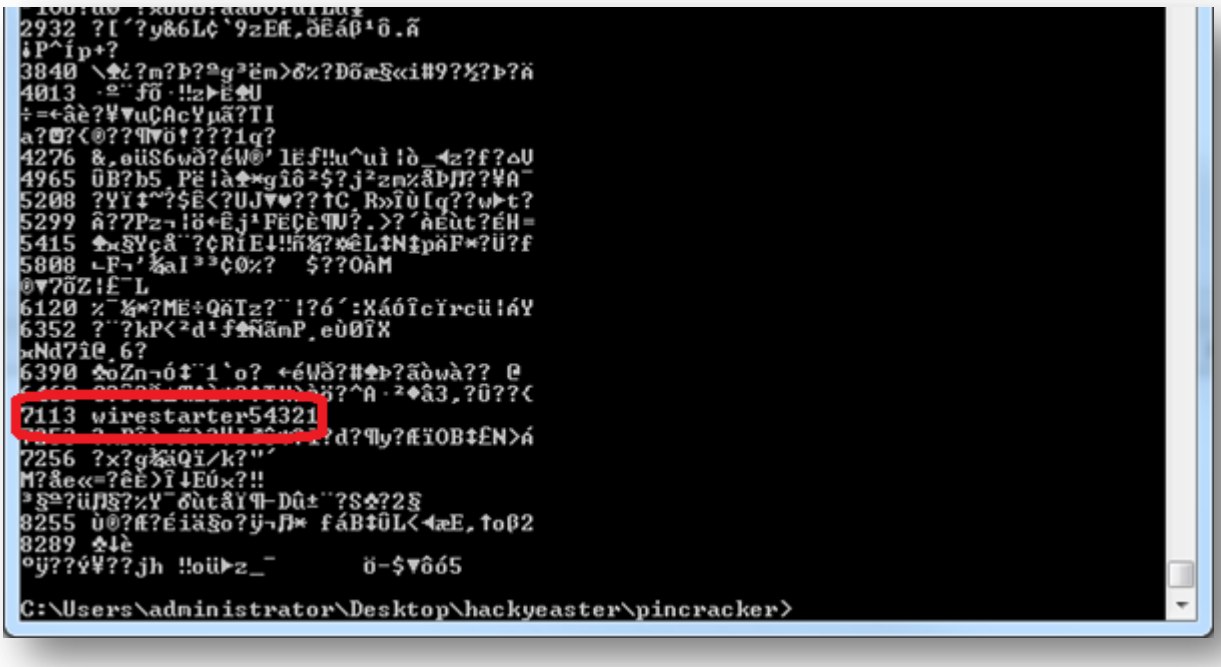
now this is much easier to understand and its also clear that we can code a little bruteforcer that just tries every PIN until we get something in cleartext.

first i thought, its easy to recompile this, but **JAVA I HATE YOU @#°§¬|¢)(/&%ç*"**

i spent more time in finding a good base64 code and compiling this, than actually cracking it.

```
public static int do_crypt(){
for(int x = 0; x < 10000; x++) {
try
    {
        SecretKeySpec var5 = new SecretKeySpec(get_sha(Integer.toString(x), "ovaederecumsale", 10000), "AES");
        Cipher var3 = Cipher.getInstance("AES");
        var3.init(2, var5);
        String var7 = new String(var3.doFinal(decode("8QeNdEdkspV6+1I77SEEEF4aWs5dl/auahJ46MMufkg=")));
        System.out.println(Integer.toString(x) + " " + var7);
    }
        catch(Exception e){
        //System.out.println(e.getMessage());
        //System.out.println(Integer.toString(x));
    }
}
    return 0;
}

public static byte[] get_sha(String var0, String var1, int var2) {
    try
    {
        MessageDigest var4 = MessageDigest.getInstance("SHA1");
        byte[] var5 = (var1 + var0).getBytes();

        for(int var3 = 0; var3 < var2; ++var3) {
         var5 = var4.digest(var5);
        }

        byte[] var6 = new byte[16];
        System.arraycopy(var5, 0, var6, 0, 15);
        return var6;
    }

    catch(Exception e){
        return null;
    }
}
```

i obviously stripped all unnecessary code and added some error handling. this way, it will only print something, when the decryption worked. i could have added some character checking code, which only prints it when it contains ascii chars, but it worked very well like this.



entering the PIN 7113 in the mobile app downloads the **egg_26.png**. rock n roll!

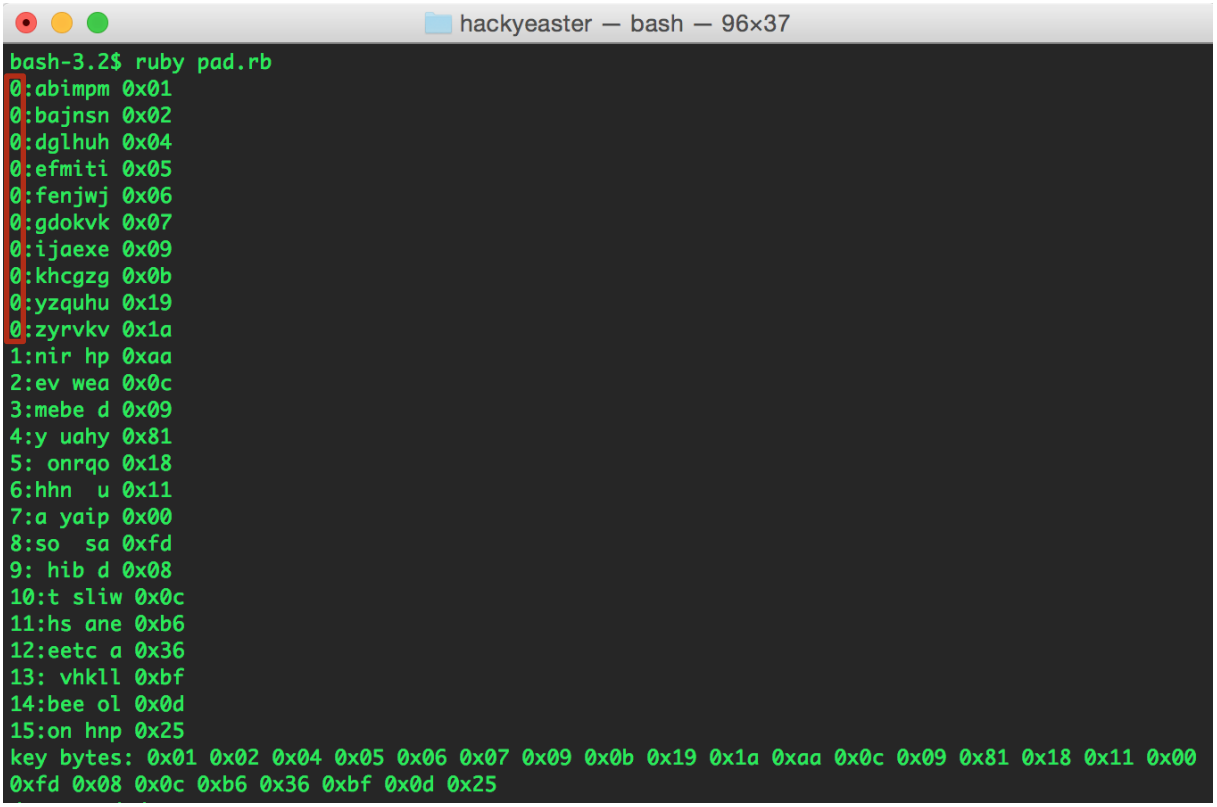
CHALLENGE 27 - TOO MANY TIME PAD

in this challenge we are given a ciphertext and the information, that a one-time-pad was not used properly. we can assume that all crypto texts are encrypted with the same key and we also know, that it does only contain lowercase letters and spaces.

so lets just make a script which xor's every byte of the same position in the crypto text with the same random byte and then check the result for lowercase letters and spaces only. i did this in ruby with the help of a simple regex. only results of all decrypted ciphertexts that contain letters and spaces only are printed out.

```
code1 = ["60c46964f83879618e2878de539f6f4a6271d716"].pack("H*").unpack("C*")
code2 = ["63c37a6ca177792092602cc553c9684b"].pack("H*").unpack("C*")
code3 = ["68d82c6bf4767f79dd617f9642d768057f63c1"].pack("H*").unpack("C*")
code4 = ["6c8a7b6ce06a3161dd6a60d755d42d4d6d67"].pack("H*").unpack("C*")
code5 = ["71c26929e96931698e2865d816d3624b687cd6"].pack("H*").unpack("C*")
code6 = ["6cda6d6df87764709c6c7bd357d361556d77"].pack("H*").unpack("C*")

(0..15).each do |x|           #do it for each byte of the shortest ciphertext
  (0..255).each do |i|        #try every byte between 0 and 255
    a=(code1[x] ^ i).chr      #xor the bytes
    b=(code2[x] ^ i).chr
    c=(code3[x] ^ i).chr
    d=(code4[x] ^ i).chr
    e=(code5[x] ^ i).chr
    f=(code6[x] ^ i).chr
    s=a+b+c+d+e+f             #join the results
    if (s =~ /^[a-z ]{6}$/)!=nil  #check result for lowercase and spaces only
     puts x.to_s+":"+s+" "+sprintf("0x%02x", i)
     keybytes+=sprintf("0x%02x ", i)
    end
  end
end
puts "key bytes: "+keybytes #print all candidates
```

and here is the result of my script (notice my leet hacker shell):



nice! we have collisions only on the first byte and because the ciphertexts are not all the same length, we are missing some bytes in the end - but lets go on.

now i added a decrypter to my script which uses these discovered bytes to decrypt the codes. for the first byte i just tried manually all candidates until the cleartext made sense- it was the 0x05 byte:

```
#decrypt with the key found
a,b,c,d,e,f="","","","","",""
key = ["05aa0c0981181100fd080cb636bf0d25"].pack("H*").unpack("C*")
(0..15).each do |x|
  a+=(code1[x] ^ key[x]).chr
  b+=(code2[x] ^ key[x]).chr
  c+=(code3[x] ^ key[x]).chr
  d+=(code4[x] ^ key[x]).chr
  e+=(code5[x] ^ key[x]).chr
  f+=(code6[x] ^ key[x]).chr
end
puts "decrypted data:"+"\n"+"---------------"+"\n"+a+"\n"+b+"\n"+c+"\n"+d+"\n"+e+"\n"+f
```

```
decrypted data:
---------------
enemy has the bo
five oh oh seven
mr bunny is the
i wear a black h
the hq is in lon
ipadyoupadweallp
bash-3.2$
```

from here, we are able to calculate the missing bytes in the end, but we actually dont even have to. the last missing characters are guessable:

ipadyoupadweallpad

HAPPY EASTER

thanks PS for this awesome event. this was super fun. i have for sure learned some new things and this event kept me busy for some days! too bad its already over.

sadly, i was only the second person that finished all challenges (damn you paper challenge) – M. was faster again – like in 2014.

anyway enjoy my solutions and see you next year!