Hackvent 2019 Writeup by Vyrixx

Contents:

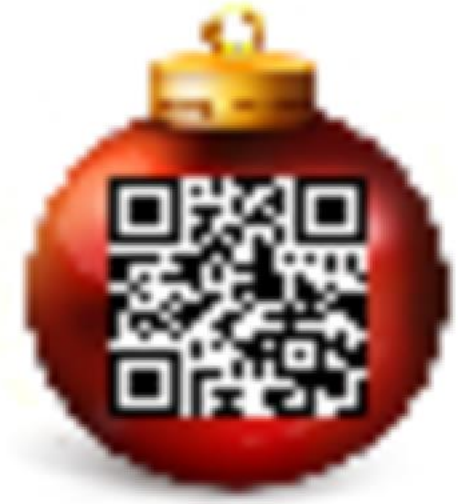
HV19.01 censored

Image with blurred QR code

Hint: 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?

Extract thumbnail with

```
exiftool -b -ThumbnailImage f182d5f0-1d10-4f0f-a0c1-7cba0981b6da.jpg
> my_thumbnail.jpg
```



Scan with QR Code App or upload image to decoding website
FLAG: HV19{just-4-PREview!}

HV19.02 Triangulations

Zip with file Triangulation.stl inside
Open with SolidWorks, remove Ball 3d models -> MaxiCode

Take screenshot of maxicode and decode it with zxing.org
Flag: HV19{Cr4ck_Th3_B411!}

HV19.03 Hodor, Hodor, Hodor

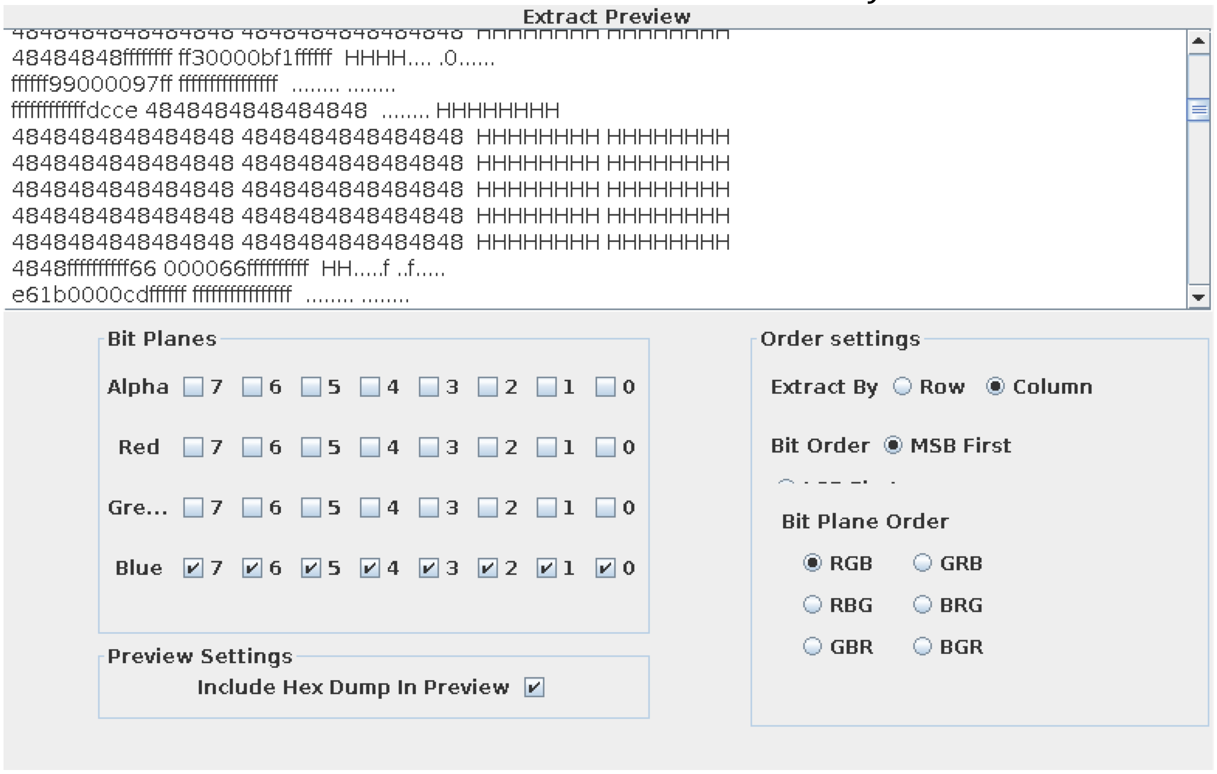
```
Install npm & Hodor language
https://github.com/hummingbirdtech/hodor
save description as challenge.hd
execute file: hodor challenge.hd
output: SFYxOXtoMDFkLXRoMy1kMDByLTQyMDQtbGQ0WX0=
decode Base64
FLAG: HV19{h01d-th3-d00r-4204-ld4Y}
```

HV19.04 password policy circumvention

```
Download zip and extract .ahk file
Install autohotkey on windows
Slowly! enter password „merry christmas geeks“ into editor
Flag: HV19{R3memb3r, rem3mber - the 24th 0f December}
```

HV19.05 Santa Parcel Tracking

```
Download Image, open with Stegsolve
View the contents oft he blue channel and order by column
```



Extract Preview

```
48484848ffffffff ff30000bf1ffffff  HHHH.... .0......
ffffff99000097ff ffffffffffffffff  ........ ........
ffffffffffffdcce 4848484848484848  ........ HHHHHHHH
4848484848484848 4848484848484848  HHHHHHHH HHHHHHHH
4848484848484848 4848484848484848  HHHHHHHH HHHHHHHH
4848484848484848 4848484848484848  HHHHHHHH HHHHHHHH
4848484848484848 4848484848484848  HHHHHHHH HHHHHHHH
4848484848484848 4848484848484848  HHHHHHHH HHHHHHHH
4848ffffffffff66 000066ffffffffff  HH.....f ..f.....
e61b0000cdffffff ffffffffffffffff  ........ ........
```

Bit Planes

Alpha ☐ 7 ☐ 6 ☐ 5 ☐ 4 ☐ 3 ☐ 2 ☐ 1 ☐ 0

Red ☐ 7 ☐ 6 ☐ 5 ☐ 4 ☐ 3 ☐ 2 ☐ 1 ☐ 0

Gre... ☐ 7 ☐ 6 ☐ 5 ☐ 4 ☐ 3 ☐ 2 ☐ 1 ☐ 0

Blue ☑ 7 ☑ 6 ☑ 5 ☑ 4 ☑ 3 ☑ 2 ☑ 1 ☑ 0

Preview Settings

Include Hex Dump In Preview ☑

Order settings

Extract By Row [✓ Column]

Bit Order [✓ MSB First]

Bit Plane Order

[✓ RGB] GRB

RBG BRG

GBR BGR

```
Every vertical line has the hex-value of a letter
Flag: HV19{D1fficult_to_g3t_a_SPT_R3ader}
Explanation: The ascii values oft he flag are hidden in the blue
channel of the barcode stripes
```

HV19.06 Bacon and eggs

Bacon Cipher
Decode the text: regular font is „a“, italic font is „b“
Decoded text:
SANTALIKESHISBACONBUTALSOTHISBACONTHEPASSWORDISHVXBACONCIPHERISSIMPL
EBUTCOOLXREPLACEXWITHBRACKETSANDUSEUPPERCASEFORALLCHARACTERS

FLAG: HV19{BACONCIPHERISSIMPLEBUTCOOL}

HV19.07 Santa Rider

Download video and watch it.
After a while multiple LEDs are blinking at the same time
lights represent multiples of 2
code is 72 86 49 57 123 49 109 95 97 108 115 48 95 119 48 114 107 49
110 103 95 48 110 95 97 95 114 53 109 48 116 51 95 99 48 110 116 114
48 108 125
translate to ascii
FLAG: HV19{1m_als0_w0rk1ng_0n_a_r3m0t3_c0ntr0l}

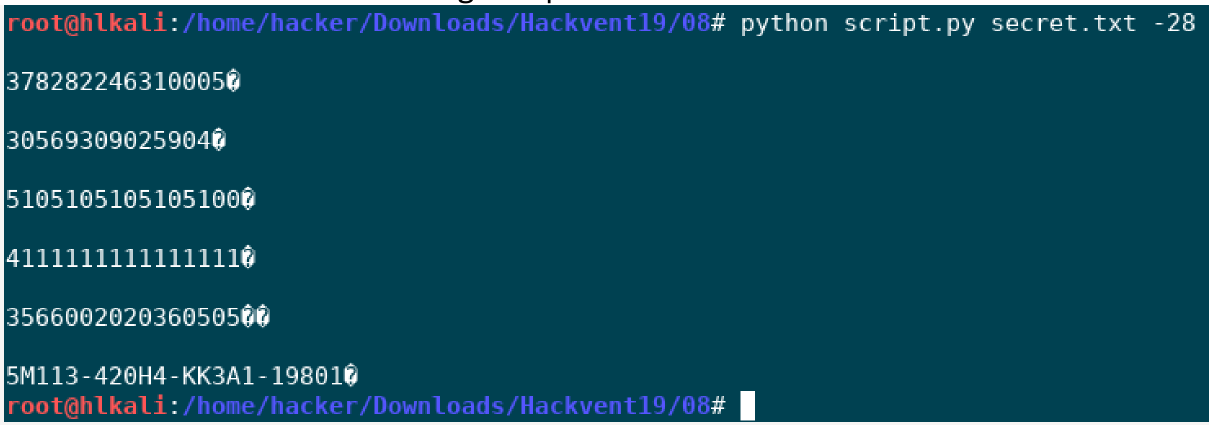
HV19.08 SmileNcryptor 4.0

SQL Dump with credit card values and a flag that are „encrypted“
-> Encoded not encrypted
all cc numbers have to consist of letters between 1-9
offset of characters to ascii 1-9 values increases from start to end.
Write a python script that reads the values and adds a rotation cipher with a variable offset (increases 1 for every letter)

```
import sys
inFile = sys.argv[1]
offset = int(sys.argv[2])
newline = ""
with open(sys.argv[1], 'r') as file:
    for line in file:
     newline = ""
        for letter in line:
          if (ord(letter)+offset) > 256:
               modletter = chr(ort(letter)+offset -256)
          elif (ord(letter)+offset) < 0:
               modletter = chr(ord(letter)+offset + 256)
          else:
               modletter = chr(ord(letter)+offset)
```

```
            offset = offset - 1
            newline = newline +modletter
        print newline
        offset = int(sys.argv[2])
```

This leads to the following output:



```
root@hlkali:/home/hacker/Downloads/Hackvent19/08# python script.py secret.txt -28

3782822463100050

305693090259040

5105105105105100

4111111111111110

3566002020360505

5M113-420H4-KK3A1-19801
root@hlkali:/home/hacker/Downloads/Hackvent19/08#
```

FLAG: HV19{5M113-420H4-KK3A1-19801}

HV19.09 Santas Quick Response 3.0

Image is a qr code but is not readable
qr code is 33x33 cells
Image in description leads to rule30
-> generate rule30 image on http://kidojo.com/cellauto/generate.cgi
rule:30
Image Width: 44
Image height 44
Pixel size: 1
Image format: jpg

Create files that contain „0“ for all white cells and „1“ for all black cells with a python script:

```
from PIL import Image
import sys
inFile = sys.argv[1]
outFile = sys.argv[2]
numtiles = sys.argv[3]
im = Image.open(inFile) # Can be many different formats.
pix = im.load()
length =  im.size  # Get the width and hight of the image for
iterating over
offset = int(length[0])/int(numtiles)
indexx = 0
indexy = 0
out = open(outFile,"w+")
while indexy < length[1]:
      while indexx < length[0]:
            value = pix[indexx, indexy]
            #print value
            if(value > 250):
                  out.write("0")
            else:
                  out.write("1")
            indexx = indexx + offset
            #print indexx
      #print indexy
      indexx = 0
      indexy = indexy + offset
      out.write("\n")
```

XOR the 2 textfiles with another script:

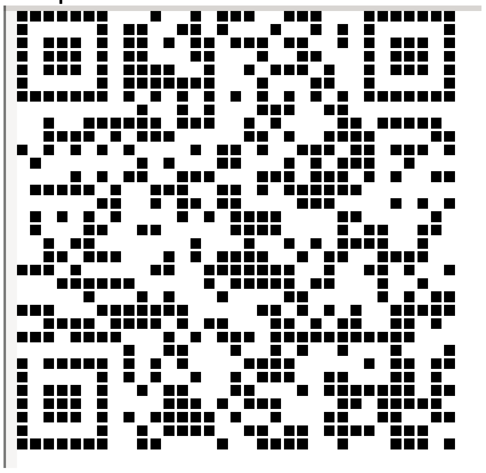
```
import sys
lines1 = open(sys.argv[1], 'r').readlines()
lines2 = open(sys.argv[2], 'r').readlines()
out = open(sys.argv[3], 'w+')
for i in range (0, len(lines1)):
      line1 = lines1[i]
      line2 = lines2[i]
      for y in range(0, len(line1)-1):
            value1 = line1[y]
            value2 = line2[y+5]
            if(value1 == value2):
                  out.write("0")
            else:
                  out.write("1")
      out.write("\n")
```

the second file is larger than the original qrcode (44x44) because elsewise the generation rules get confused. Rule30 file has to be centered, hint: centering is hard so I tried around with the horizontal offset   -> value2 = line2[y+5]

create image out of txt file with a 3rd python script:

```
import sys
from PIL import Image
img = Image.new('RGB', (220, 220), color='white')
pixels = img.load()
lines = open(sys.argv[1], 'r').readlines()
for x in range (0, len(lines)):
      line = lines[x]
      for y in range(0, len(line)-1):
            value = line[y]
            if(value == "1"):
                  for z in range(0,4):
                    for a in range(0,4):
                        pixels[(y*5)+a,(x*5)+z] = (0,0,0)
        else:
                for z in range(0,4):
                    for a in range(0,4):
                        pixels[(y*5)+a,(x*5)+z] = (255,255,255)
img.save('output.png')
```

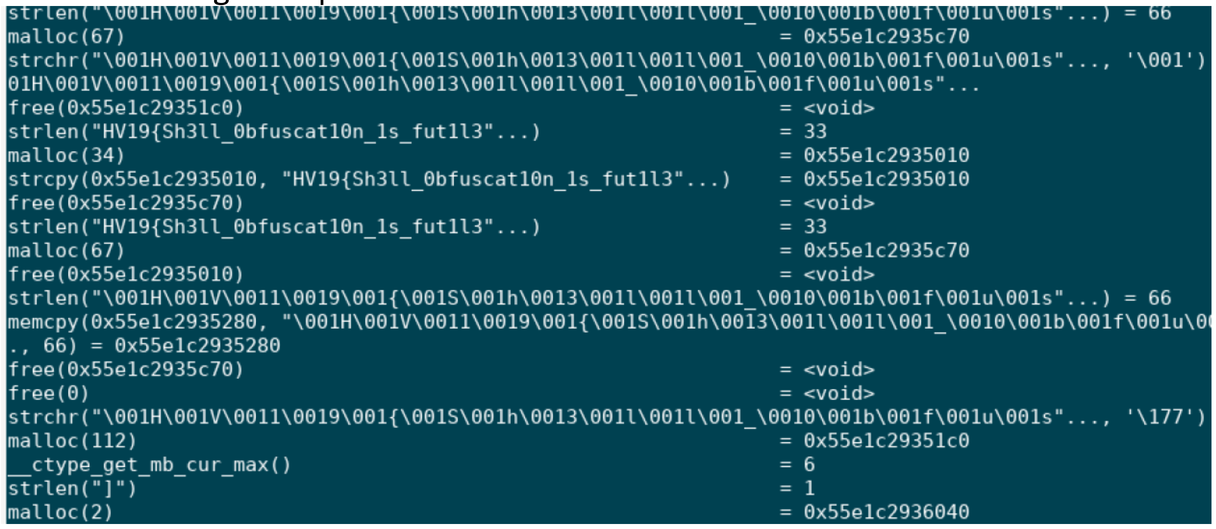
output:



FLAG: HV19{Cha0tic_yet-0rdered}

HV19.10 Guess what

Download zip and unzip
analyze binary with ltrace:
ltrace ./guess3
Scroll through output



```
strlen("\001H\001V\0011\0019\001{\001S\001h\0013\001l\001l\001_\0010\001b\001f\001u\001s"...) = 66
malloc(67)                                                   = 0x55e1c2935c70
strchr("\001H\001V\0011\0019\001{\001S\001h\0013\001l\001l\001_\0010\001b\001f\001u\001s"..., '\001')
01H\001V\0011\0019\001{\001S\001h\0013\001l\001l\001_\0010\001b\001f\001u\001s"...
free(0x55e1c29351c0)                                         = <void>
strlen("HV19{Sh3ll_0bfuscat10n_1s_fut1l3"...)                = 33
malloc(34)                                                   = 0x55e1c2935010
strcpy(0x55e1c2935010, "HV19{Sh3ll_0bfuscat10n_1s_fut1l3"...)   = 0x55e1c2935010
free(0x55e1c2935c70)                                         = <void>
strlen("HV19{Sh3ll_0bfuscat10n_1s_fut1l3"...)                = 33
malloc(67)                                                   = 0x55e1c2935c70
free(0x55e1c2935010)                                         = <void>
strlen("\001H\001V\0011\0019\001{\001S\001h\0013\001l\001l\001_\0010\001b\001f\001u\001s"...) = 66
memcpy(0x55e1c2935280, "\001H\001V\0011\0019\001{\001S\001h\0013\001l\001l\001_\0010\001b\001f\001u\0
., 66) = 0x55e1c2935280
free(0x55e1c2935c70)                                         = <void>
free(0)                                                      = <void>
strchr("\001H\001V\0011\0019\001{\001S\001h\0013\001l\001l\001_\0010\001b\001f\001u\001s"..., '\177')
malloc(112)                                                  = 0x55e1c29351c0
__ctype_get_mb_cur_max()                                     = 6
strlen("]")                                                  = 1
malloc(2)                                                    = 0x55e1c2936040
```

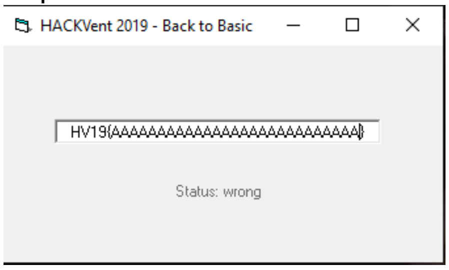
FLAG: HV19{Sh3ll_0bfuscat10n_1s_fut1l3}

HV19.11 Frolicsome Santa Jokes API

```
FSJA Api
Tokens are Json Web Tokens

My Token:
yJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjp7InVzZXJuYW1lIjoidGFzdGV1c2VyIiwicGxh
dGludW0iOmZhbHNlfSwiZXhwIjoxNTc2MDk5MzU1Ljk5ODAwMDAwMH0.mF5FvxAb02H9
qqEDAXe2HbZXQfg39VTdKUVMdqfGxr8

Decoded:
{
  "alg": "HS256"
}
{
  "user": {
    "username": "tasteuser",
    "platinum": false
  },
  "exp": 1576099355.998
}

Modify Token
{
  "user": {
    "username": "santa",
    "platinum": true
  },
  "exp": 1576099355.998
}

And run against api:

Curl -X GET "http://whale.hacking-
lab.com:10101/fsja/random?token=eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjp7InV
zZXJuYW1lIjoic2FudGEiLCJwbGF0aW51bSI6dHJ1ZX0sImV4cCI6MTU3NjA5OTM1NS4
5OTh9.MCvQJltiaf3h0Vdu16O73zDYclbhyxPvFttrZpnrNaU"
{"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}","
FLAG: HV19{th3_cha1n_1s_0nly_as_str0ng_as_th3_w3ak3st_l1nk}
```

HV19.12 back to basic

Back2Basic.exe a visual studio .exe binary
Analyze binary with IDA Pro and Immunity Debugger.
Program checks input in 3 steps – Prefix “HV19{“ Length (33) and a loop where all input characters of the flag starting from the 6th position are xored with a byte array where the value increases by one for each position. It contains not printable characters, so it is not correctly recognized by the disassembler.
Input:

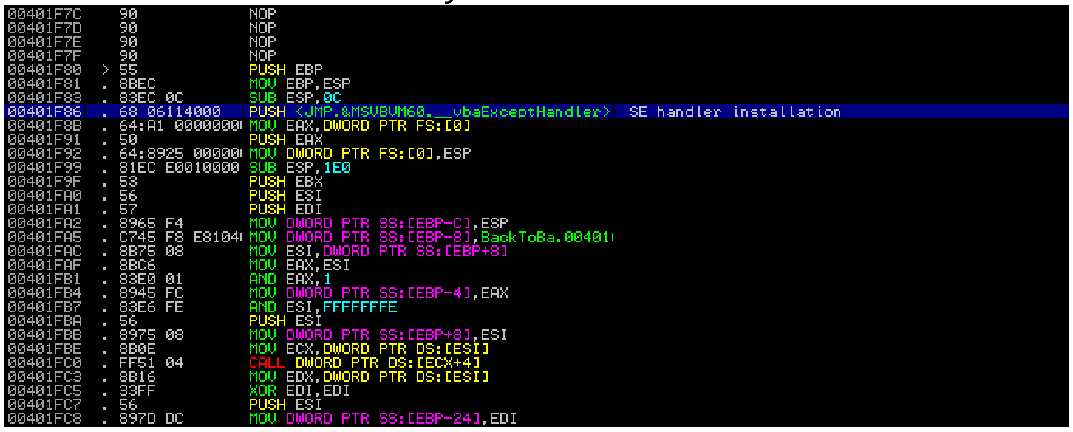


HACKVent 2019 - Back to Basic

HV19{AAAAAAAAAAAAAAAAAAAAAAAAAAA}

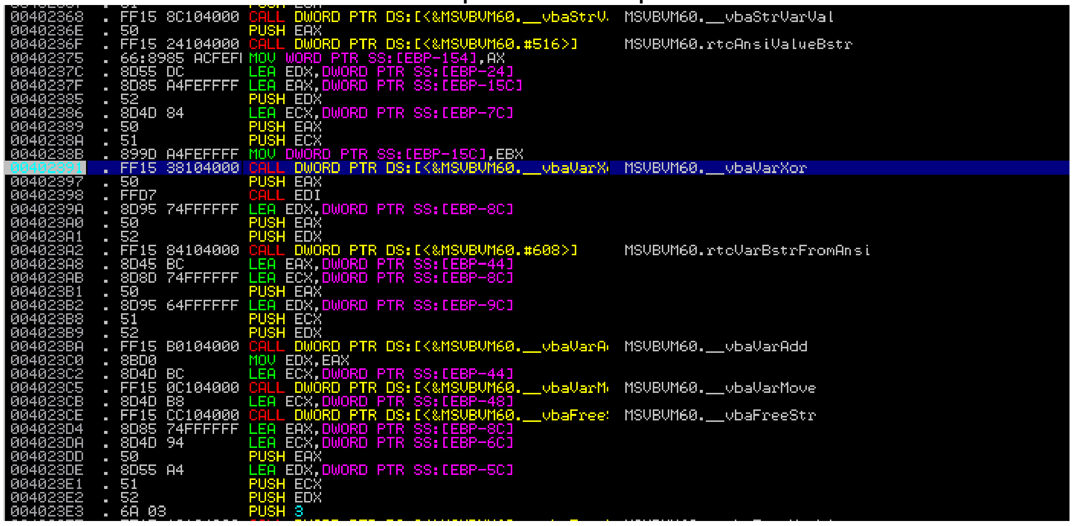
Status: wrong

The relevant functionality starts at 00401F86

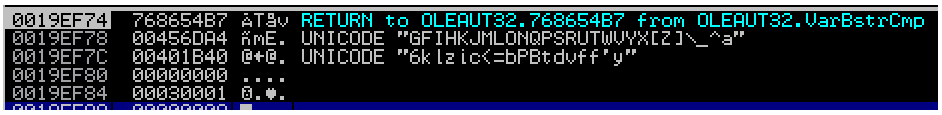


```
00401F7C    90              NOP
00401F7D    90              NOP
00401F7E    90              NOP
00401F7F    90              NOP
00401F80  > 55              PUSH EBP
00401F81  . 8BEC            MOV EBP,ESP
00401F83  . 83EC 0C         SUB ESP,0C
00401F86  . 68 06114000     PUSH <JMP.&MSVBVM60.__vbaExceptHandler>  SE handler installation
00401F8B  . 64:A1 0000000(  MOV EAX,DWORD PTR FS:[0]
00401F91  . 50              PUSH EAX
00401F92  . 64:8925 00000(  MOV DWORD PTR FS:[0],ESP
00401F99  . 81EC E0010000   SUB ESP,1E0
00401F9F  . 53              PUSH EBX
00401FA0  . 56              PUSH ESI
00401FA1  . 57              PUSH EDI
00401FA2  . 8965 F4         MOV DWORD PTR SS:[EBP-C],ESP
00401FA5  . C745 F8 E8104(  MOV DWORD PTR SS:[EBP-8],BackToBa.00401(
00401FAC  . 8B75 08         MOV ESI,DWORD PTR SS:[EBP+8]
00401FAF  . 8BC6            MOV EAX,ESI
00401FB1  . 83E0 01         AND EAX,1
00401FB4  . 8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
00401FB7  . 83E6 FE         AND ESI,FFFFFFFE
00401FBA  . 56              PUSH ESI
00401FBB  . 8975 08         MOV DWORD PTR SS:[EBP+8],ESI
00401FBE  . 8B0E            MOV ECX,DWORD PTR DS:[ESI]
00401FC0  . FF51 04         CALL DWORD PTR DS:[ECX+4]
00401FC3  . 8B16            MOV EDX,DWORD PTR DS:[ESI]
00401FC5  . 33FF            XOR EDI,EDI
00401FC7  . 56              PUSH ESI
00401FC8  . 897D DC         MOV DWORD PTR SS:[EBP-24],EDI
```

Xor instruction for the input in loop:



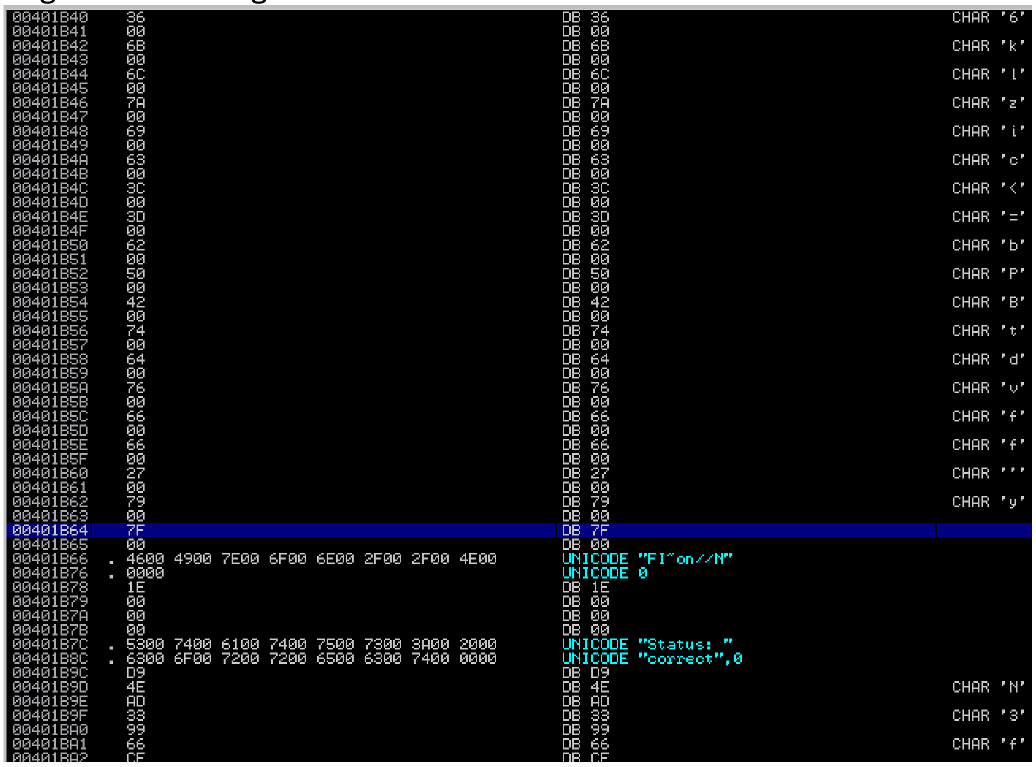
```
00402368  . FF15 8C104000   CALL DWORD PTR DS:[<&MSVBVM60.__vbaStrV(  MSVBVM60.__vbaStrVarVal
0040236E  . 50              PUSH EAX
0040236F  . FF15 24104000   CALL DWORD PTR DS:[<&MSVBVM60.#516>]      MSVBVM60.rtcAnsiValueBstr
00402375  . 66:8985 ACFEF(  MOV WORD PTR SS:[EBP-154],AX
0040237C  . 8D55 DC         LEA EDX,DWORD PTR SS:[EBP-24]
0040237F  . 8D85 A4FEFFFF   LEA EAX,DWORD PTR SS:[EBP-15C]
00402385  . 52              PUSH EDX
00402386  . 8D4D 84         LEA ECX,DWORD PTR SS:[EBP-7C]
00402389  . 50              PUSH EAX
0040238A  . 51              PUSH ECX
0040238B  . 899D A4FEFFFF   MOV DWORD PTR SS:[EBP-15C],EBX
00402391  . FF15 38104000   CALL DWORD PTR DS:[<&MSVBVM60.__vbaVarX(  MSVBVM60.__vbaVarXor
00402397  . 50              PUSH EAX
00402398  . FFD7            CALL EDI
0040239A  . 8D95 74FFFFFF   LEA EDX,DWORD PTR SS:[EBP-8C]
004023A0  . 50              PUSH EAX
004023A1  . 52              PUSH EDX
004023A2  . FF15 84104000   CALL DWORD PTR DS:[<&MSVBVM60.#608>]      MSVBVM60.rtcVarBstrFromAnsi
004023A8  . 8D45 BC         LEA EAX,DWORD PTR SS:[EBP-44]
004023AB  . 8D8D 74FFFFFF   LEA ECX,DWORD PTR SS:[EBP-8C]
004023B1  . 50              PUSH EAX
004023B2  . 8D95 64FFFFFF   LEA EDX,DWORD PTR SS:[EBP-9C]
004023B8  . 51              PUSH ECX
004023B9  . 52              PUSH EDX
004023BA  . FF15 B0104000   CALL DWORD PTR DS:[<&MSVBVM60.__vbaVarA(  MSVBVM60.__vbaVarAdd
004023C0  . 8BD0            MOV EDX,EAX
004023C2  . 8D4D BC         LEA ECX,DWORD PTR SS:[EBP-44]
004023C5  . FF15 0C104000   CALL DWORD PTR DS:[<&MSVBVM60.__vbaVarM(  MSVBVM60.__vbaVarMove
004023CB  . 8D4D B8         LEA ECX,DWORD PTR SS:[EBP-48]
004023CE  . FF15 CC104000   CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree(  MSVBVM60.__vbaFreeStr
004023D4  . 8D85 74FFFFFF   LEA EAX,DWORD PTR SS:[EBP-8C]
004023DA  . 8D4D 94         LEA ECX,DWORD PTR SS:[EBP-6C]
004023DD  . 50              PUSH EAX
004023DE  . 8D55 A4         LEA EDX,DWORD PTR SS:[EBP-5C]
004023E1  . 51              PUSH ECX
004023E2  . 52              PUSH EDX
004023E3  . 6A 03           PUSH 3
```



```
0019EF74   768654B7  ÀT♗v  RETURN to OLEAUT32.768654B7 from OLEAUT32.VarBstrCmp
0019EF78   00456DA4  ñmE.  UNICODE "GFIHKJMLONQPSRUTWVYX[Z]\_^a"
0019EF7C   00401B40  @+@.  UNICODE "6klzic<=bPBtdvff'y"
0019EF80   00000000  ....
0019EF84   00030001  ☺.♥.
```

This graphic shows the stack with the input(HV19{AAAAAAAAAAAAAAAAAAAAAAAAAAAA}) after it is xored and the

first part of bytearray that the xored input is xored against to check if the flag is correct.
Copy the values of the comparison array from the data section and write a python script that xores it with the correct offsets to regain the flag.



```
00401B40    36                                DB 36                    CHAR '6'
00401B41    00                                DB 00
00401B42    6B                                DB 6B                    CHAR 'k'
00401B43    00                                DB 00
00401B44    6C                                DB 6C                    CHAR 'l'
00401B45    00                                DB 00
00401B46    7A                                DB 7A                    CHAR 'z'
00401B47    00                                DB 00
00401B48    69                                DB 69                    CHAR 'i'
00401B49    00                                DB 00
00401B4A    63                                DB 63                    CHAR 'c'
00401B4B    00                                DB 00
00401B4C    3C                                DB 3C                    CHAR '<'
00401B4D    00                                DB 00
00401B4E    3D                                DB 3D                    CHAR '='
00401B4F    00                                DB 00
00401B50    62                                DB 62                    CHAR 'b'
00401B51    00                                DB 00
00401B52    50                                DB 50                    CHAR 'P'
00401B53    00                                DB 00
00401B54    42                                DB 42                    CHAR 'B'
00401B55    00                                DB 00
00401B56    74                                DB 74                    CHAR 't'
00401B57    00                                DB 00
00401B58    64                                DB 64                    CHAR 'd'
00401B59    00                                DB 00
00401B5A    76                                DB 76                    CHAR 'v'
00401B5B    00                                DB 00
00401B5C    66                                DB 66                    CHAR 'f'
00401B5D    00                                DB 00
00401B5E    66                                DB 66                    CHAR 'f'
00401B5F    00                                DB 00
00401B60    27                                DB 27                    CHAR '''
00401B61    00                                DB 00
00401B62    79                                DB 79                    CHAR 'y'
00401B63    00                                DB 00
00401B64    7F                                DB 7F
00401B65    00                                DB 00
00401B66  . 4600 4900 7E00 6F00 6E00 2F00 2F00 4E00   UNICODE "FI~on//N"
00401B76  . 0000                              UNICODE 0
00401B78    1E                                DB 1E
00401B79    00                                DB 00
00401B7A    00                                DB 00
00401B7B    00                                DB 00
00401B7C  . 5300 7400 6100 7400 7500 7300 3A00 2000   UNICODE "Status: "
00401B8C  . 6300 6F00 7200 7200 6500 6300 7400 0000   UNICODE "correct",0
00401B9C    D9                                DB D9
00401B9D    4E                                DB 4E                    CHAR 'N'
00401B9E    AD                                DB AD
00401B9F    33                                DB 33                    CHAR '3'
00401BA0    99                                DB 99
00401BA1    66                                DB 66                    CHAR 'f'
00401BA2    CE                                DB CE
```

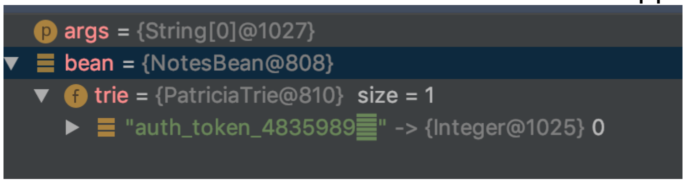
Script:

```
#Hex value of the not printable character is 0x7f (DEL)
encrypted = "6klzic<=bPBtdvff'y FI~on//N"
flag = "HV19{"
for i in range(len(encrypted)):
    c = chr(ord(encrypted[i]) ^ (6 + i))
    flag += c
flag += '}'
print(flag)
```

FLAG: HV19{0ldsch00l_Revers1ng_Sess10n}

HV19.13 TrieMe

Examine the Notebean java class and debug it in a test Java Project.
The Login uses a PatriciaTrie to store the data.
Input is added via the setTrie Method where a unescapeJava method is called so escaped characters .
Find an input that destroys the lookup and the `auth_token_4835989` key.
`auth_token_4835989`\0 is a input that does that and replaces the key so that the isAdmin condition is flipped.



```
args = {String[0]@1027}
▼ bean = {NotesBean@808}
  ▼ trie = {PatriciaTrie@810}  size = 1
    ▶ "auth_token_4835989▒" -> {Integer@1025} 0
```

Log in with the modified token
FLAG: HV19{get_th3_chocolateZ}

HV19.14 Achtung das Flag

Perl implementation of Achtung die Kurve
just patch out the collision detection, reduced speed and print the flag to the console
Modified game:

```
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_/;$PMMtQJOcHm8eFQfdsdNAS20=sub{$zvYPxUpXMSsw=($zvYPxUpXMSsw*16807)&0xFFFFFFFF;};
($a1Ivn0ECw49I5I0oE0='07&3-"11*/(')=~y$!-=$`-~$;($Sk61A7pO='K&:P3&44')=~y$!-=$`-~$;m/Mm/g;
($sk6i47pO='K&:R&-&"4&')=~y$!-=$`-
~$;;;;$d28Vt03MEbdY0=sub{print(pack('n',$fff[$S9cXJIGB0BWce++]
^($PMMtQJOcHm8eFQfdsdNAS20->()&0xDEAD)));};'42';($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;
$_8NZQooI5K4b=0;$Sk6lA7p0=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=$vMnyQdAkfgIIik->#FOO
createText($PMMtQJOcHm8eFQfdsdNAS20->()%600+20,$PMMtQJOcHm8eFQfdsdNAS20->()%440+20,#Perl!!
"-text"=>$d28Vt03MEbdY0->(),"-$y"=>$z);})->();$HACK;$i=$vMnyQdAkfgIIik->repeat(50,sub{$_=(
$_8NZQooI5K4b+=0.1*$Sk6lA7p0);;$p[0]+=3.0*cos;$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;$vgOjwRk4wIo7_->bind("<$Sk61A7pO-n>"=>sub{
$Sk6lA7p0=1;});$vgOjwRk4wIo7_->bind("<$Sk61A7pO-m>"=>sub{$Sk6lA7p0=-1;});$vgOjwRk4wIo7_#%"
->bind("<$sk6i47pO-n>"=>sub{$Sk6lA7p0=0 if$Sk6lA7p0>0;});$vgOjwRk4wIo7_->bind("<$sk6i47pO"
."-m>"=>sub{$Sk6lA7p0=0 if $Sk6lA7p0<0;});$::{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__
```

FLAG: 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

Mqtt Broker version 1.4.11
-> CVE-2017-7650
The fix addresses the problem by restricting access for clients with a '#', '+', or '/' in their username or client id. '/' has been included in the list of characters disallowed because it also has a special meaning in a topic and may represent an additional risk.
Copy contents from the website and host them on an own server, so the js files for requests can be modified. (writing a python script did not work for unknown reasons)

Query system topics from the broker:
```
var topic = '$SYS/#';
```

Message:
Topic: $SYS/broker/version message: mosquitto version 1.4.11 (We elves are super-smart and know about CVE-2017-7650 and the POC. So we made a genious fix you never will be able to pass. Hohoho)

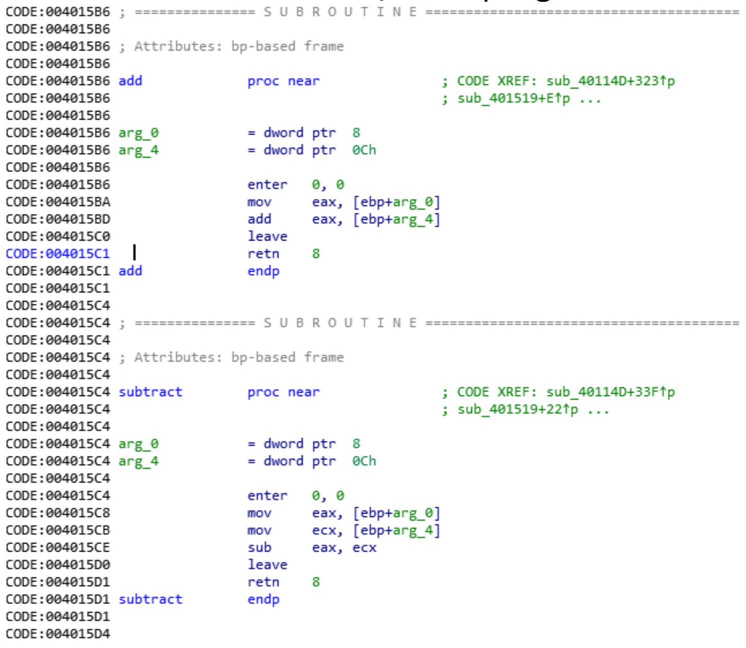
Elves did close the CVE, but not completely, / is still allowed in the id.

Modify the clientid:
```
var clientid = '0735724891935373/#';
```

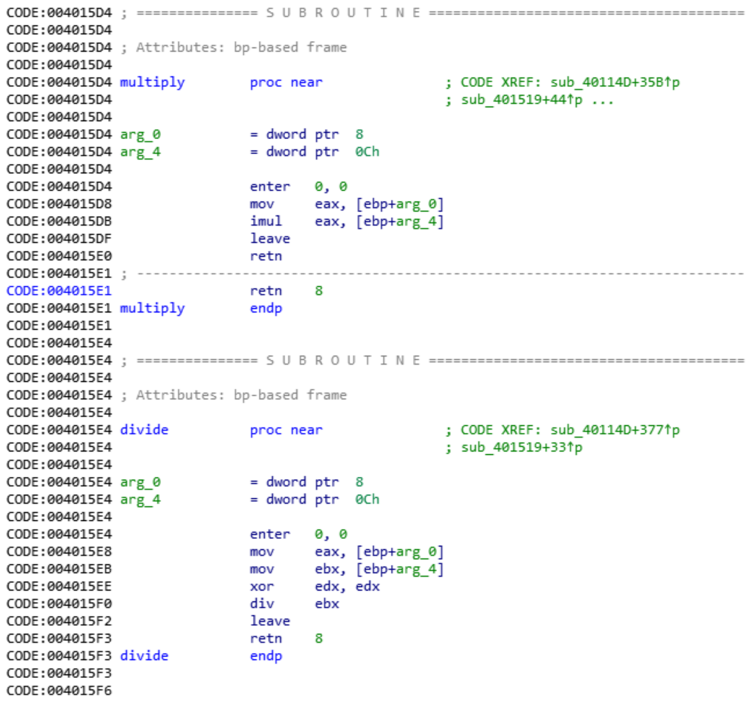
Message:
Topic:
HV19/gifts/0735724891935373/HV19{N0_1nput_v4l1d4t10n_3qu4ls_d1s4st3r} message: Congrats, you got it. The elves should not overrate their smartness!!!
FLAG: HV19{N0_1nput_v4l1d4t10n_3qu4ls_d1s4st3r}

HV19.16 B0rked Calculator

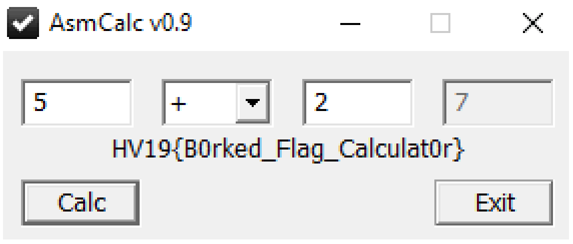
Windows executable binary that contains a calculator.
Open with IDA Pro and insert the right commands to complete the calculation functions, the program then outputs the flag.



```
CODE:004015B6 ; =============== S U B R O U T I N E =======================================
CODE:004015B6
CODE:004015B6 ; Attributes: bp-based frame
CODE:004015B6
CODE:004015B6 add             proc near               ; CODE XREF: sub_40114D+323↑p
CODE:004015B6                                         ; sub_401519+E↑p ...
CODE:004015B6
CODE:004015B6 arg_0           = dword ptr  8
CODE:004015B6 arg_4           = dword ptr  0Ch
CODE:004015B6
CODE:004015B6                 enter   0, 0
CODE:004015BA                 mov     eax, [ebp+arg_0]
CODE:004015BD                 add     eax, [ebp+arg_4]
CODE:004015C0                 leave
CODE:004015C1                 retn    8
CODE:004015C1 add             endp
CODE:004015C1
CODE:004015C4
CODE:004015C4 ; =============== S U B R O U T I N E =======================================
CODE:004015C4
CODE:004015C4 ; Attributes: bp-based frame
CODE:004015C4
CODE:004015C4 subtract        proc near               ; CODE XREF: sub_40114D+33F↑p
CODE:004015C4                                         ; sub_401519+22↑p ...
CODE:004015C4
CODE:004015C4 arg_0           = dword ptr  8
CODE:004015C4 arg_4           = dword ptr  0Ch
CODE:004015C4
CODE:004015C4                 enter   0, 0
CODE:004015C8                 mov     eax, [ebp+arg_0]
CODE:004015CB                 mov     ecx, [ebp+arg_4]
CODE:004015CE                 sub     eax, ecx
CODE:004015D0                 leave
CODE:004015D1                 retn    8
CODE:004015D1 subtract        endp
CODE:004015D1
CODE:004015D4
```



```
CODE:004015D4 ; =============== S U B R O U T I N E =======================================
CODE:004015D4
CODE:004015D4 ; Attributes: bp-based frame
CODE:004015D4
CODE:004015D4 multiply        proc near               ; CODE XREF: sub_40114D+35B↑p
CODE:004015D4                                         ; sub_401519+44↑p ...
CODE:004015D4
CODE:004015D4 arg_0           = dword ptr  8
CODE:004015D4 arg_4           = dword ptr  0Ch
CODE:004015D4
CODE:004015D4                 enter   0, 0
CODE:004015D8                 mov     eax, [ebp+arg_0]
CODE:004015DB                 imul    eax, [ebp+arg_4]
CODE:004015DF                 leave
CODE:004015E0                 retn
CODE:004015E1 ; ---------------------------------------------------------------------------
CODE:004015E1                 retn    8
CODE:004015E1 multiply        endp
CODE:004015E1
CODE:004015E4
CODE:004015E4 ; =============== S U B R O U T I N E =======================================
CODE:004015E4
CODE:004015E4 ; Attributes: bp-based frame
CODE:004015E4
CODE:004015E4 divide          proc near               ; CODE XREF: sub_40114D+377↑p
CODE:004015E4                                         ; sub_401519+33↑p
CODE:004015E4
CODE:004015E4 arg_0           = dword ptr  8
CODE:004015E4 arg_4           = dword ptr  0Ch
CODE:004015E4
CODE:004015E4                 enter   0, 0
CODE:004015E8                 mov     eax, [ebp+arg_0]
CODE:004015EB                 mov     ebx, [ebp+arg_4]
CODE:004015EE                 xor     edx, edx
CODE:004015F0                 div     ebx
CODE:004015F2                 leave
CODE:004015F3                 retn    8
CODE:004015F3 divide          endp
CODE:004015F3
CODE:004015F6
```



HV19{B0rked_Flag_Calculat0r}

FLAG: HV19{B0rked_Flag_Calculat0r}

HV19.17 Unicode Portal

Examine the sourcecode file:

```
/**
 * 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;
}
```

LOWER(username) = BINARY LOWER('".$usr."')"); <- Suspicious
Find a value that has a different binary representation for „santa"
than the username in the db

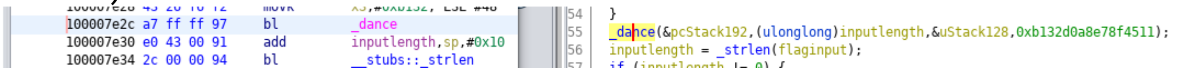
```
/**
 * Registers a new user.
 */
function registerUser($conn, $username, $password) {
  $usr = $conn->real_escape_string($username);
  $pwd = password_hash($password, PASSWORD_DEFAULT);
  $conn-
>query("INSERT INTO users (username, password) VALUES (UPPER('".$usr."'),'".$p
wd."') ON DUPLICATE KEY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

if the user already exists in the db, the password is updated.
When a user that is already in the db is registered, the password
gets updated

The collation of the SQL server treats for example a and á in the
same way when it is accent insensitive
-> santá is a valid username to update the password and access the
admin page
FLAG: HV19{h4v1ng_fun_w1th_un1c0d3}

HV19.18 Dance with me

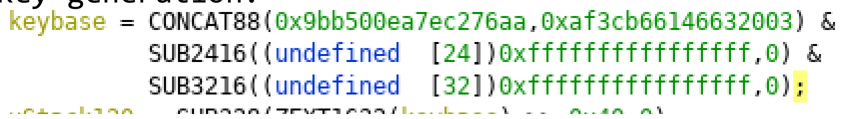
AARCH64 binary -> decompile with Ghidra
After research it turned out to be a Salsa20 cipher
Call of dance function with iv at the and (attention: reverse byte
order!)



```
100007e2c a7 ff ff 97     bl       _dance
100007e30 e0 43 00 91     add      inputlength,sp,#0x10
100007e34 2c 00 00 94     bl       __stubs::_strlen
```

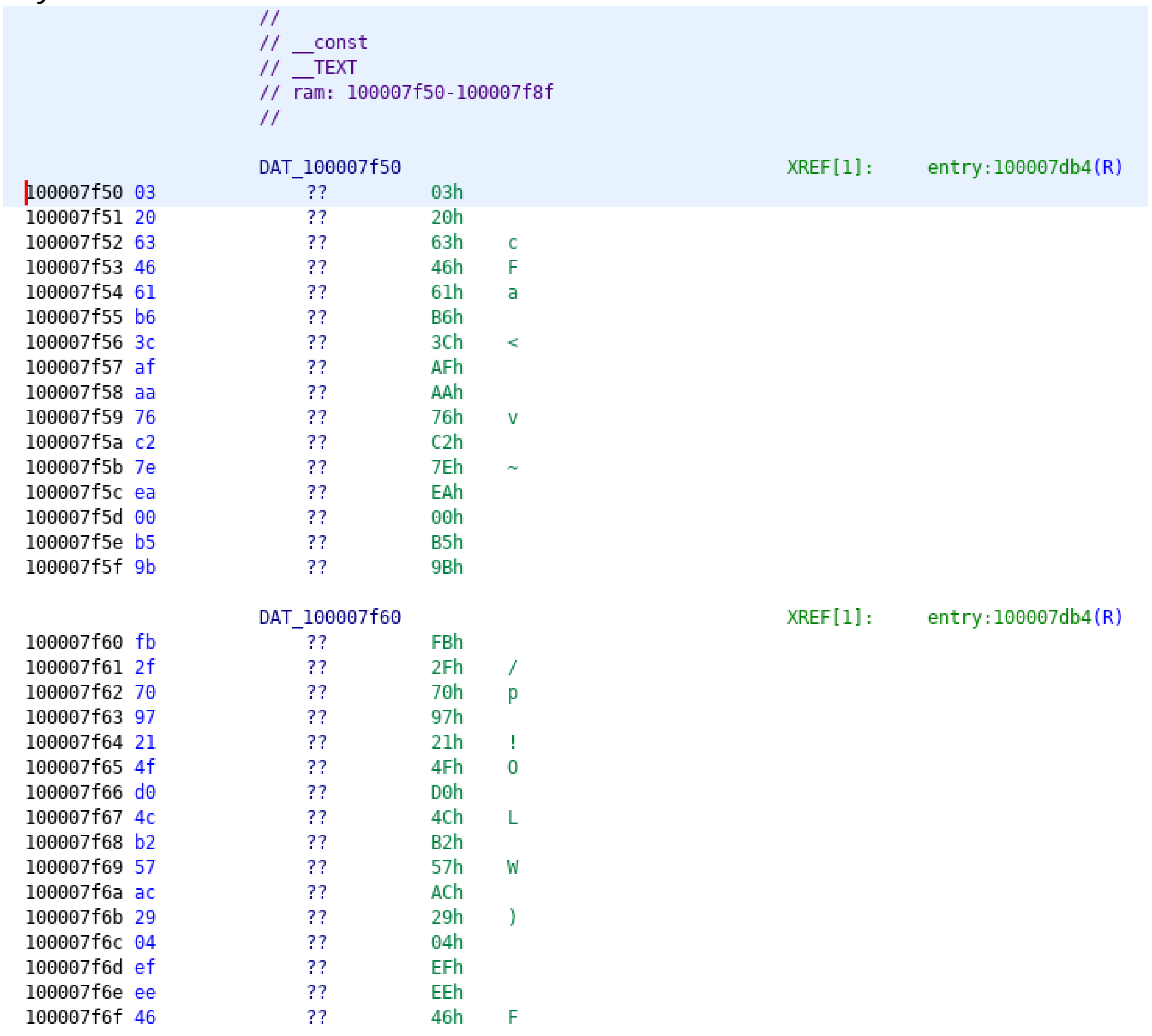
```
}
_dance(&pcStack192,(ulonglong)inputlength,&uStack128,0xb132d0a8e78f4511);
inputlength = _strlen(flaginput);
```

Key generation:



```
 keybase = CONCAT88(0x9bb500ea7ec276aa,0xaf3cb66146632003) &
           SUB2416((undefined  [24])0xffffffffffffffff,0) &
           SUB3216((undefined  [32])0xffffffffffffffff,0);
```

Key in data section:



```
                        //
                        // __const
                        // __TEXT
                        // ram: 100007f50-100007f8f
                        //

                        DAT_100007f50                          XREF[1]:     entry:100007db4(R)
 100007f50 03              ??         03h
 100007f51 20              ??         20h
 100007f52 63              ??         63h    c
 100007f53 46              ??         46h    F
 100007f54 61              ??         61h    a
 100007f55 b6              ??         B6h
 100007f56 3c              ??         3Ch    <
 100007f57 af              ??         AFh
 100007f58 aa              ??         AAh
 100007f59 76              ??         76h    v
 100007f5a c2              ??         C2h
 100007f5b 7e              ??         7Eh    ~
 100007f5c ea              ??         EAh
 100007f5d 00              ??         00h
 100007f5e b5              ??         B5h
 100007f5f 9b              ??         9Bh

                        DAT_100007f60                          XREF[1]:     entry:100007db4(R)
 100007f60 fb              ??         FBh
 100007f61 2f              ??         2Fh    /
 100007f62 70              ??         70h    p
 100007f63 97              ??         97h
 100007f64 21              ??         21h    !
 100007f65 4f              ??         4Fh    O
 100007f66 d0              ??         D0h
 100007f67 4c              ??         4Ch    L
 100007f68 b2              ??         B2h
 100007f69 57              ??         57h    W
 100007f6a ac              ??         ACh
 100007f6b 29              ??         29h    )
 100007f6c 04              ??         04h
 100007f6d ef              ??         EFh
 100007f6e ee              ??         EEh
 100007f6f 46              ??         46h    F
```

write a python script to decrypt the input:

```
from Crypto.Cipher import Salsa20

secret =
b'\x03\x20\x63\x46\x61\xb6\x3c\xaf\xaa\x76\xc2\x7e\xea\x00\xb5\x9b\x
fb\x2f\x70\x97\x21\x4f\xd0\x4c\xb2\x57\xac\x29\x04\xef\xee\x46'
msg_nonce = b'\x11\x45\x8f\xe7\xa8\xd0\x32\xb1'
ciphertext =
b'\x09\x6C\xD4\x46\xEB\xC8\xE0\x4D\x2F\xDE\x29\x9B\xE4\x4F\x32\x28\x
63\xF7\xA3\x7C\x18\x76\x35\x54\xEE\xE4\xC9\x9C\x3F\xAD\x15'
cipher = Salsa20.new(key=secret, nonce=msg_nonce)
plaintext = cipher.decrypt(ciphertext)
print(plaintext)
```

FLAG: HV19{Danc1ng_Salsa_in_ass3mbly}

HV19.19 🎅

```
Copy the emojis from the description into a file and install
Emojicode.
Run the script from description
outputs linux elf binary

Run binary -> program asks for input (emojis)
Create a list of possible passwords with emoji characters
(emojis.json) and write a python script that bruteforces the correct
password.

#!/usr/bin/python
# coding=utf-8
from pwn import *
import json
with open ('emojis.json') as json_file:
    emojis = json.load(json_file)
    for emoji in emojis:
        p = process('./ch19')
        print(p.recvuntil('🔒 ➡️ 🎅⁉️ ➡️ 🎄🚩 '))
        p.sendline(emoji.encode('utf-8'))
        output = p.readall()
        if not "👎" in output:
            print("Done: " + emoji)
            print(output)
            break

Solution: 🔑
Since the correct input is luckily only one emoji, it runs through
very fast
FLAG: HV19{*<|:-)____\o/____;-D}
```

HV19.20 I want to play a game

```
Binary (apparently a PlayStation4 crack).
The file contains a hash value f86d4f9d2c049547bd61f942151ffb55,
googling this hash leads to a PS4UPDATE.PUP file that must be
downloaded.
Analyzing the file with a disassembler shows that the file is seeked
with an offset of 0x1337 in 2 nested loops and these read bytes are
xored with a byte array from the data section of the crack program.
Write a python script that does this to get the flag:

data =
list(b'\xce\x55\x95\x4e\x38\xc5\x89\xa5\x1b\x6f\x5e\x25\xd2\x1d\x2a\
x2b\x5e\x7b\x39\x14\x8e\xd0\xf0\xf8\xf8\xa5')
```

```
offset = 0x1337
with open("PS4UPDATE.PUP", "rb") as f:
     while offset != 0x1714908:
          f.seek(offset)
          key = f.read(len(data))
          newdata = ""
          count = 0
          for c in data:
               newdata += chr(ord(c) ^ ord(key[count]))
               count += 1
          data = newdata
          offset += 0x1337
print(data)
```

FLAG: HV19{C0nsole_H0mebr3w_FTW}

HV19.21 Happy Christmas 256

In the description is the public key, ciphertext and some information about the encryption.

A SHA256 hash of a password is used to generate the private key. The correct password is “santacomesatxmas”, the only password in the rockyou.txt passwordlist that produces a valid ECC private key with the x & y parameters of the public key and P-256 curve.
This private key is used in the pbkdf2 hash function to create the key for the AES decryption of the ciphertext.

Write a python script that does these things:

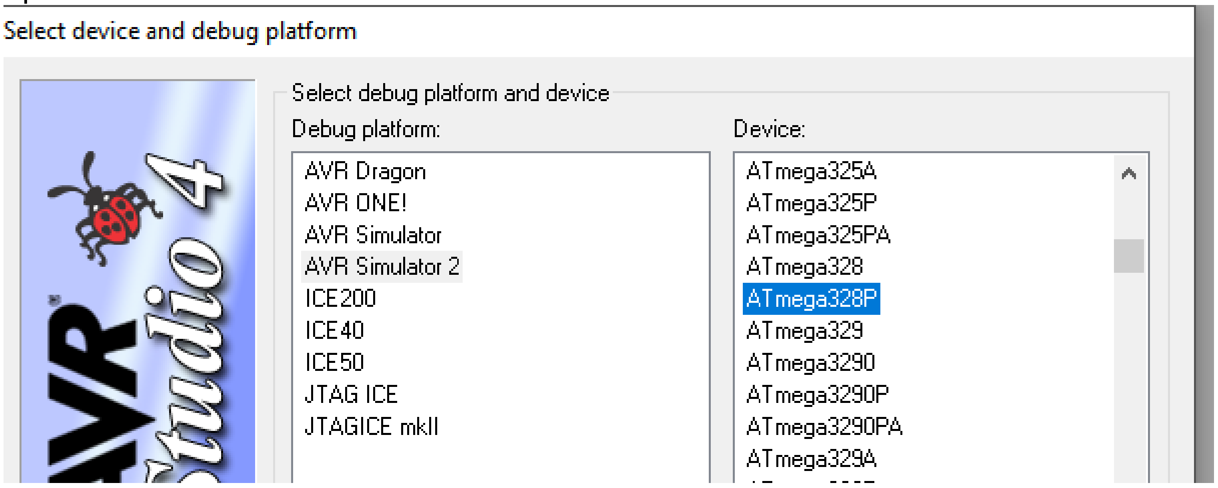
```
import hashlib, pyaes, pbkdf2, binascii, os, base64
from Crypto.PublicKey import ECC
from Crypto.Cipher import AES
x =
0xc58966d17da18c7f019c881e187c608fcb5010ef36fba4a199e7b382a088072f
y =
0xd91b949eaf992c464d3e0d09c45b173b121d53097a9d47c25220c0b4beb943c
salt = 'TwoHundredFiftySix'
enc = 'Hy97Xwv97vpwGn21finVvZj5pK/BvBjscf6vffm1po0='
passwords = []
#corrpassword = 'santacomesatxmas'
unpad = lambda s: s[:-ord(s[len(s) - 1:])]
with open('/usr/share/wordlists/rockyou.txt') as fp:
     line = fp.readline()
     while line:
          if(len(line)==17):
               passwords.append(line.strip())
          line = fp.readline()
```

```
print(len(passwords))
print(passwords[0])
for pw in passwords:
	try:
		d = hashlib.sha256(pw.encode('utf-8')).hexdigest()
		ecckey = ECC.construct(curve='NIST P-256',d=int(d,16),
point_x=x, point_y=y)
		print(pw)
		print(d)
		corrpassword = pw
		break
	except:
		continue
if corrpassword != '':
	aeskey = hashlib.pbkdf2_hmac('sha256', corrpassword, salt, 256
* 256 * 256)
	print(base64.b64encode(aeskey))
	#aeskey = '6x4EQsplZuXWh3QNJGyupts7KFH3dBQNFTyEjVlRVwU='
	benc = base64.b64decode(enc)
	decipher = AES.new(base64.b64decode(aeskey), AES.MODE_ECB)
	print(decipher.decrypt(benc).decode('utf-8'))
```

This outputs the flag.
FLAG: HV19{sry_n0_crypt0mat_th1s_year}

HV19.22 The command … is lost

Arduino Firmware File
Download .data file and rename to hex
Open in AVR Studio 4

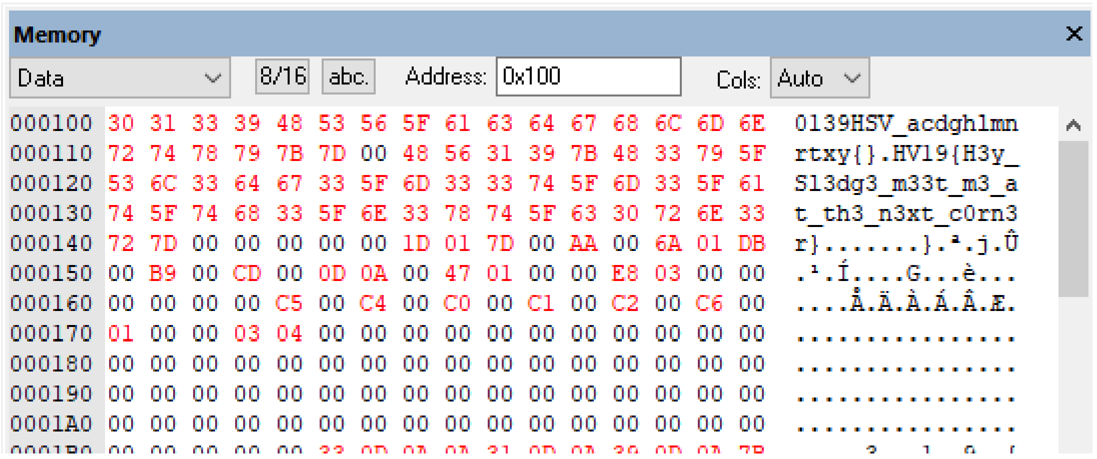

Select device and debug platform

Select debug platform and device

Debug platform: AVR Simulator 2

Device: ATmega328P

Execute, Pause and search Data Section



```
000100 30 31 33 39 48 53 56 5F 61 63 64 67 68 6C 6D 6E  0139HSV_acdghlmn
000110 72 74 78 79 7B 7D 00 48 56 31 39 7B 48 33 79 5F  rtxy{}.HV19{H3y_
000120 53 6C 33 64 67 33 5F 6D 33 33 74 5F 6D 33 5F 61  Sl3dg3_m33t_m3_a
000130 74 5F 74 68 33 5F 6E 33 78 74 5F 63 30 72 6E 33  t_th3_n3xt_c0rn3
000140 72 7D 00 00 00 00 00 1D 01 7D 00 AA 00 6A 01 DB  r}.......}.ª.j.Û
000150 00 B9 00 CD 00 0D 0A 00 47 01 00 00 E8 03 00 00  .¹.Í....G...è...
000160 00 00 00 00 C5 00 C4 00 C0 00 C1 00 C2 00 C6 00  ....Å.Ä.À.Á.Â.Æ.
000170 01 00 00 03 04 00 00 00 00 00 00 00 00 00 00 00  ................
000180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

FLAG: `HV19{H3y_Sl3dg3_m33t_m3_at_th3_n3xt_c0rn3r}`

HV19.23 Internet Data Archive

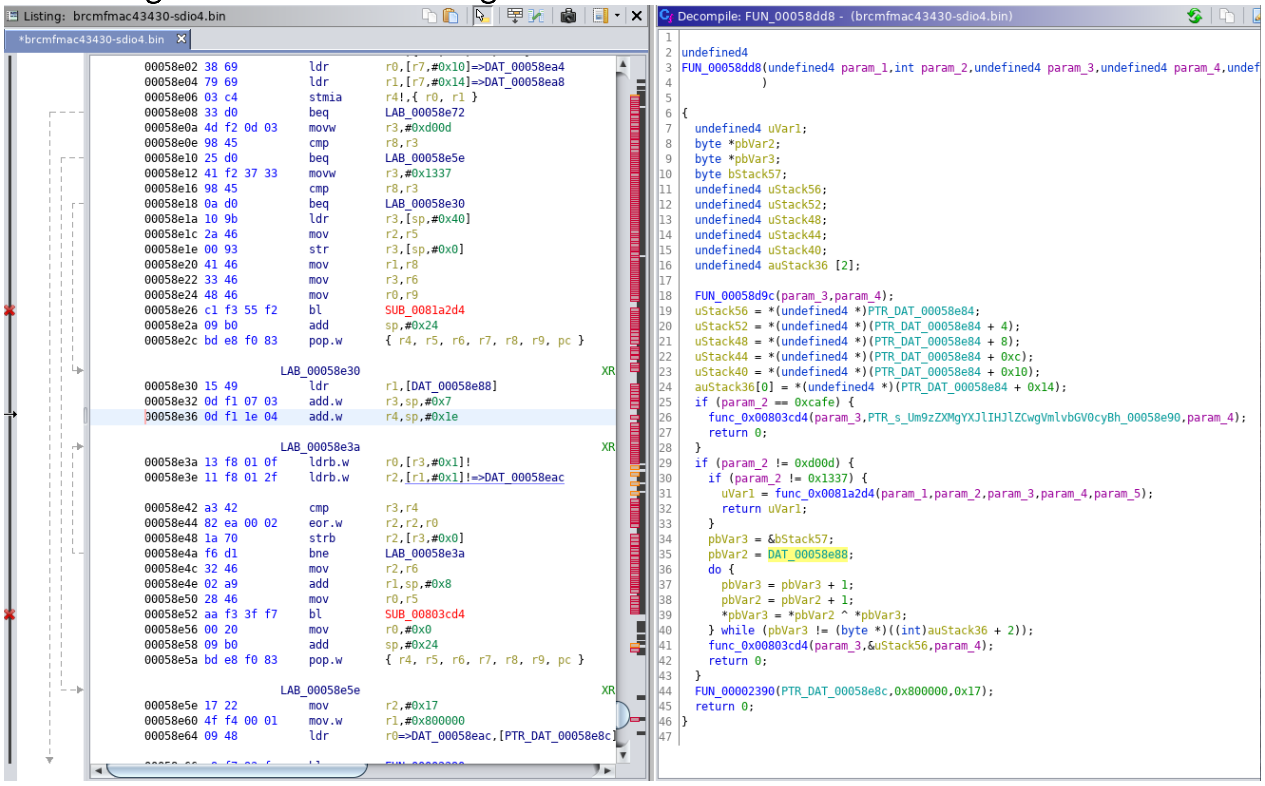
Reconnaissance shows that the server has a /tmp/ directory with a phpversion.info and Santa-data.zip file inside. The password generation algorithm for the zip file is the same as for IDA Pro.
https://devco.re/blog/2019/06/21/operation-crack-hacking-IDA-Pro-installer-PRNG-from-an-unusual-way-en/
Follow the tutorial to generate possible 12 character long passwords and use them as input for John. After a while, the correct password is found.
Script:

```
<?php
$ca = str_split("abcdefghijkmpqrstuvwxyzABCDEFGHJKLMPQRSTUVWXYZ23456789");
$key = 0;
$pw = "";
$fh = fopen('php://output', 'w');
for ($s =0; $s < 2147483647; $s++){
        srand($s);
        for($j=0;$j<12;++$j){
          $key = rand(0, 53);
          $pw .= $ca[$key];
        }
        $pw .= "\n";
        fwrite($fh, $pw);
        $pw = "";
}
?>
```

FLAG: HV19{Cr4ckin_Passw0rdz_like_IDA_Pr0}

HV19.24 ham radio

Inspecting the binary with strings shows a Base64 string:
Um9zZXMgYXJlIHJlZCwgVmlvbGV0cyBhcmUgYmx1ZSwgRHJTY2hvdHRreSBsb3ZlcyBob29raW5nIGlvY3Rscywgd2h5IHNob3VsZG4ndCB5b3U
which says: Roses are red, Violets are blue, DrSchottky loves hooking ioctls, why shouldn't you
Searching for the b64 string in Ghidra leads to this function:



Listing: brcmfmac43430-sdio4.bin

*brcmfmac43430-sdio4.bin

```
00058e02 38 69          ldr       r0,[r7,#0x10]=>DAT_00058ea4
00058e04 79 69          ldr       r1,[r7,#0x14]=>DAT_00058ea8
00058e06 03 c4          stmia     r4!,{ r0, r1 }
00058e08 33 d0          beq       LAB_00058e72
00058e0a 4d f2 0d 03    movw      r3,#0xd00d
00058e0e 98 45          cmp       r8,r3
00058e10 25 d0          beq       LAB_00058e5e
00058e12 41 f2 37 33    movw      r3,#0x1337
00058e16 98 45          cmp       r8,r3
00058e18 0a d0          beq       LAB_00058e30
00058e1a 10 9b          ldr       r3,[sp,#0x40]
00058e1c 2a 46          mov       r2,r5
00058e1e 00 93          str       r3,[sp,#0x0]
00058e20 41 46          mov       r1,r8
00058e22 33 46          mov       r3,r6
00058e24 48 46          mov       r0,r9
00058e26 c1 f3 55 f2    bl        SUB_0081a2d4
00058e2a 09 b0          add       sp,#0x24
00058e2c bd e8 f0 83    pop.w     { r4, r5, r6, r7, r8, r9, pc }

                    LAB_00058e30                                XR
00058e30 15 49          ldr       r1,[DAT_00058e88]
00058e32 0d f1 07 03    add.w     r3,sp,#0x7
00058e36 0d f1 1e 04    add.w     r4,sp,#0x1e

                    LAB_00058e3a                                XR
00058e3a 13 f8 01 0f    ldrb.w    r0,[r3,#0x1]!
00058e3e 11 f8 01 2f    ldrb.w    r2,[r1,#0x1]!=>DAT_00058eac

00058e42 a3 42          cmp       r3,r4
00058e44 82 ea 00 02    eor.w     r2,r2,r0
00058e48 1a 70          strb      r2,[r3,#0x0]
00058e4a f6 d1          bne       LAB_00058e3a
00058e4c 32 46          mov       r2,r6
00058e4e 02 a9          add       r1,sp,#0x8
00058e50 28 46          mov       r0,r5
00058e52 aa f3 3f f7    bl        SUB_00803cd4
00058e56 00 20          mov       r0,#0x0
00058e58 09 b0          add       sp,#0x24
00058e5a bd e8 f0 83    pop.w     { r4, r5, r6, r7, r8, r9, pc }

                    LAB_00058e5e                                XR
00058e5e 17 22          mov       r2,#0x17
00058e60 4f f4 00 01    mov.w     r1,#0x800000
00058e64 09 48          ldr       r0=>DAT_00058eac,[PTR_DAT_00058e8c]
```

Decompile: FUN_00058dd8 - (brcmfmac43430-sdio4.bin)

```

undefined4
FUN_00058dd8(undefined4 param_1,int param_2,undefined4 param_3,undefined4 param_4,undef
            )

{
  undefined4 uVar1;
  byte *pbVar2;
  byte *pbVar3;
  byte bStack57;
  undefined4 uStack56;
  undefined4 uStack52;
  undefined4 uStack48;
  undefined4 uStack44;
  undefined4 uStack40;
  undefined4 auStack36 [2];

  FUN_00058d9c(param_3,param_4);
  uStack56 = *(undefined4 *)PTR_DAT_00058e84;
  uStack52 = *(undefined4 *)(PTR_DAT_00058e84 + 4);
  uStack48 = *(undefined4 *)(PTR_DAT_00058e84 + 8);
  uStack44 = *(undefined4 *)(PTR_DAT_00058e84 + 0xc);
  uStack40 = *(undefined4 *)(PTR_DAT_00058e84 + 0x10);
  auStack36[0] = *(undefined4 *)(PTR_DAT_00058e84 + 0x14);
  if (param_2 == 0xcafe) {
    func_0x00803cd4(param_3,PTR_s_Um9zZXMgYXJlIHJlZCwgVmlvbGV0cyBh_00058e90,param_4);
    return 0;
  }
  if (param_2 != 0xd00d) {
    if (param_2 != 0x1337) {
      uVar1 = func_0x0081a2d4(param_1,param_2,param_3,param_4,param_5);
      return uVar1;
    }
    pbVar3 = &bStack57;
    pbVar2 = DAT_00058e88;
    do {
      pbVar3 = pbVar3 + 1;
      pbVar2 = pbVar2 + 1;
      *pbVar3 = *pbVar2 ^ *pbVar3;
    } while (pbVar3 != (byte *)((int)auStack36 + 2));
    func_0x00803cd4(param_3,&uStack56,param_4);
    return 0;
  }
  FUN_00002390(PTR_DAT_00058e8c,0x800000,0x17);
  return 0;
}

```

Here 2 byte arrays are xored: One is from the Data section at 0x00058e94 and the other is from the ROM Memory that is not included in the binary. The ROM can be found here:
https://github.com/seemoo-lab/bcm_misc/blob/master/bcm43430a1/rom.bin
Xor the ROM and the byte array with python to receive the flag:

```
with open("rom.bin",'rb') as binfile:
      key = binfile.read(23)
with open("brcmfmac43430-sdio.bin",'rb') as binfile:
      binfile.seek(0x58e94)
      encrypted = binfile.read(23)
flag = ""
for(k, e) in zip(key, encrypted):
      flag += chr(k ^ e)
print(flag)
```

FLAG: HV19{Y0uw3n7FullM4Cm4n}

HV19 Writeup

Write a writeup and submit the PDF file 😉

HV19.H1 Hidden One

Hidden text in the description of challenge 6.
At the end: spaces and tabs

Born: January 22
Died: April 9
Mother: Lady Anne

Father: Sir Nicholas

Secrets: unknown

Save to file and execute stegsnow

```
root@hlkali:/home/hacker/Downloads/Hackvent19/06# stegsnow -C secret.txt
FLAG: HV19{1stHiddenFound}
```

HV19.H2 Hidden Two

Filename from Day 7 Video download:
3DULK2N7DcpXFg8qGo9Z9qEQqvaEDpUCBB1v.mp4
Base58 encoded
FLAG: HV19{Dont_confuse_0_and_O}

HV19.H3 Hidden Three

nmap -sT whale.hacking-lab.com
reveals that port 17 for the „quote of the day“ service is open
write short pythonscript

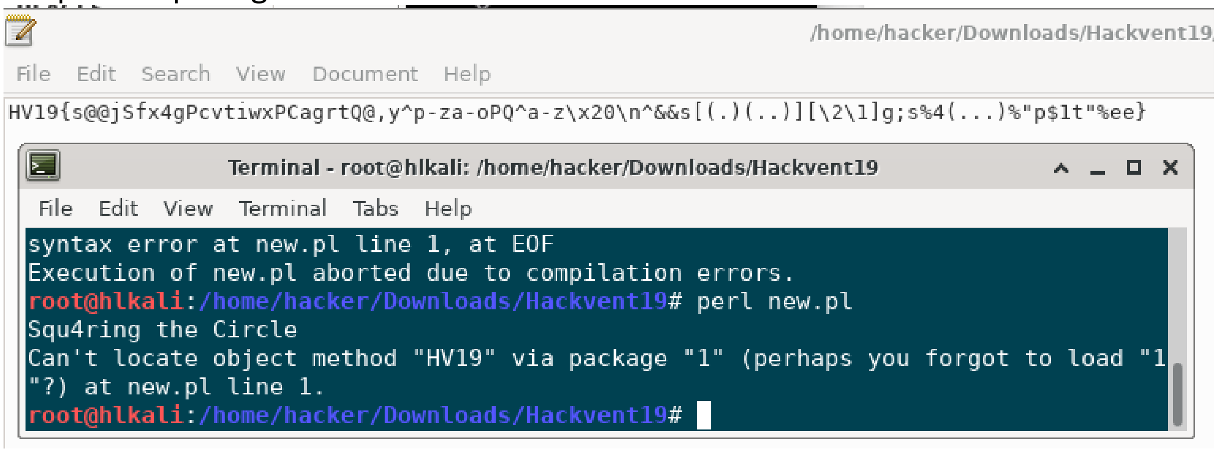
```
#!/usr/bin/env python

import socket


TCP_IP = 'whale.hacking-lab.com'
TCP_PORT = 17
BUFFER_SIZE = 2048
MESSAGE = "1"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE)
data = s.recv(BUFFER_SIZE)
s.close()
```

```
print "received data:", data

root@hlkali:/home/hacker/Downloads/Hackvent19# ./qotd.py
received data: 0
```

Letter changes every hour.
FLAG: HV19{an0ther_DAILY_fl4g}

HV19.H4 Hidden Four

Execute Flag from day 14 in perl
Output: Sq4ring the Circle



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

```
syntax error at new.pl line 1, at EOF
Execution of new.pl aborted due to compilation errors.
root@hlkali:/home/hacker/Downloads/Hackvent19# perl new.pl
Squ4ring the Circle
Can't locate object method "HV19" via package "1" (perhaps you forgot to load "1
"?) at new.pl line 1.
root@hlkali:/home/hacker/Downloads/Hackvent19#
```

FLAG: HV19{Sq4ring the Circle}