[HV21.01] X-wORd Puzzle (Author: charon)

The crossword clues weren't too hard to solve:

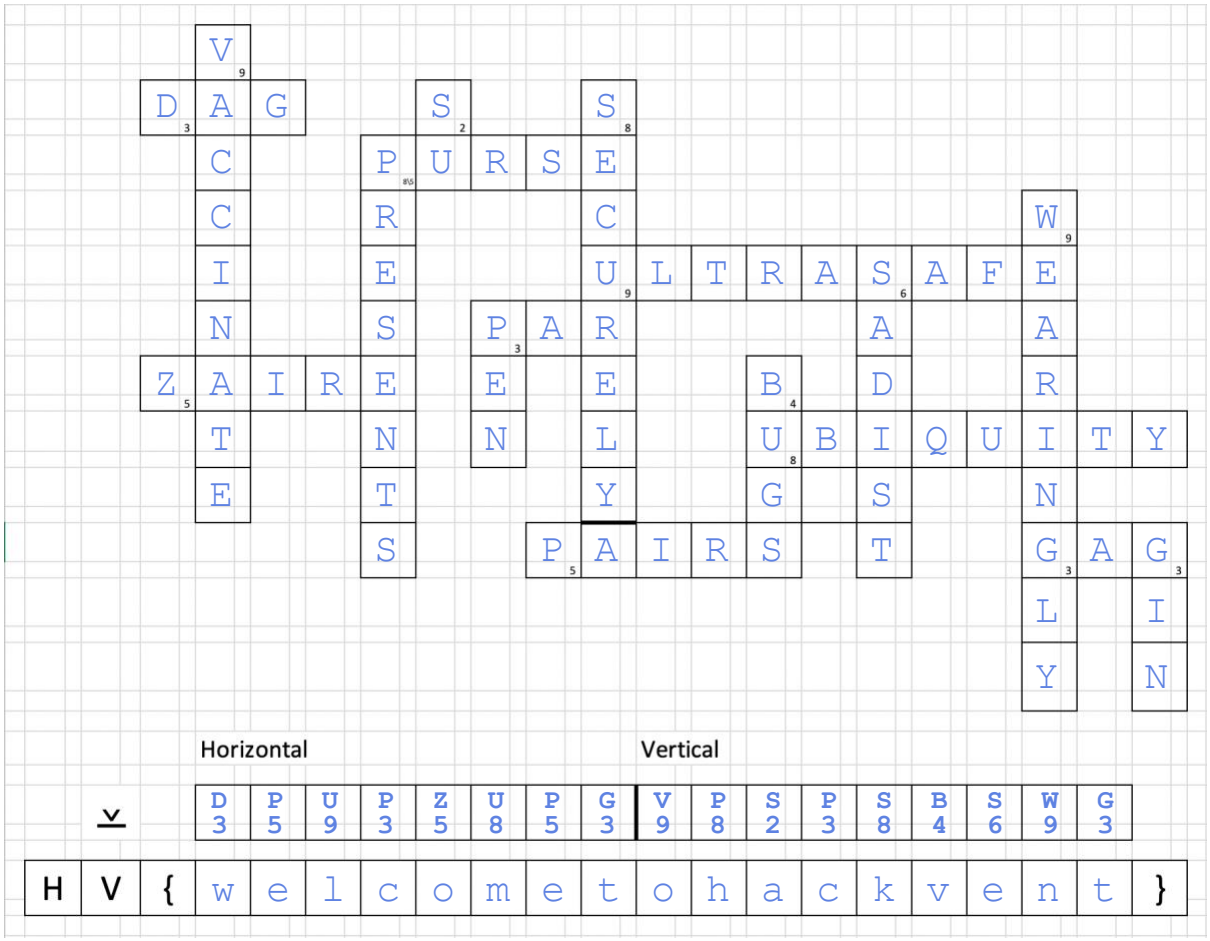
Horizontal

1 A diagram of arrows not allowing cycles
2 A handbag for carrying around money
3 Very, very secure
4 Golf: number of strokes required
5 Congo between 1971 and 1997
6 State of appearing everywhere
7 Tuples in everyday language
8 Makes you laugh or silences you

Vertical

1 Plea by many doctors right now
2 Put in parcels
3 Lets you change user
4 ...-test
5 How you should transmit your data
6 Need to squash them - fix your code!
7 Attributed to a marquis - no pain, no gain.
8 Doing something in a way that causes fatigue is doing it...
9 A drink you may need after finishing this puzzle.

The ⩒ symbol and the challenge name made it abundantly clear that some sort of XOR operation was necessary to obtain the flag. But XORing the first letter of each answer (either its ASCII value or its position in the alphabet) with the clue number and/or the number of letters in each clue didn't help at all. I finally realised that we were supposed to XOR the first letters with *ASCII values* corresponding to the number of letters in each answer. For example the answer to the first clue is "DAG", and if you XOR the ASCII values of "D" and "3" (being the number of letters in this answer), you get "w". This is the first letter of the flag.

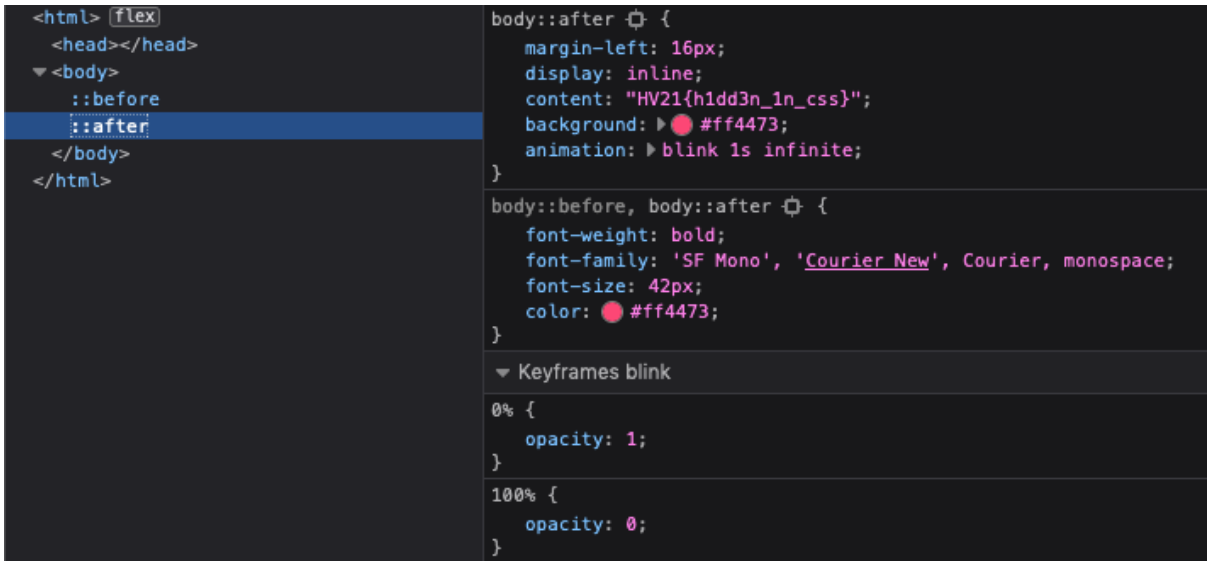


Flag: HV{welcometohackvent}

[HV21.02] No source, No luck! (Author: explo1t)

When you start up and launch the web instance, you get redirected to a YouTube video (that one with Rick Astley in it; I'm sure you've seen it before!). It should be easy enough to find out what's going on by viewing the page source before the redirect happens, but it's completely empty. What's going on?? Before you have a chance to find out, the page has already redirected to YouTube and Rick is singing that song again :-(

Fortunately, it's easy to stop this annoying behaviour. In the `about:config` page, change `accessibility.blockautorefresh` to `false`. That stops the redirect and gives you as much time as you like to look at the source code. Using the browser's Developer Tools, it's easy to see what's going on. The document body is indeed empty, but has `::before` and `::after` rules that add various bits of content, including a flag:



For those that are interested, I found an online video that describes this challenge in more detail: https://www.youtube.com/watch?v=dQw4w9WgXcQ

Flag: `HV21{h1dd3n_1n_css}`

[HV21.03] Too Much GIItTer! (Author: HaCk0)

The capitalised letters in the title of this challenge suggest that there's a git repository available somewhere. Just add `.git` to the end of the URL, and you're there:

```
Index of /.git/

../
hooks/                     02-Dec-2021 17:22                   -
info/                      02-Dec-2021 17:22                   -
logs/                      02-Dec-2021 17:22                   -
objects/                   02-Dec-2021 17:22                   -
refs/                      02-Dec-2021 17:22                   -
COMMIT_EDITMSG             02-Dec-2021 15:49                  22
HEAD                       02-Dec-2021 15:49                  23
config                     02-Dec-2021 15:49                  92
description                02-Dec-2021 15:49                  73
index                      02-Dec-2021 15:49                5142
```

The file `logs/HEAD` lists the commits to the project. I'm not an expert on these things but it looks like something with ID `b009ea9155d990aa9185e1157aaf583a636e93fd` is where the flag was added. In `objects/b0/09ea9155d990aa9185e1157aaf583a636e93fd`, we find the following record (PHP's `zlib_decode()` function can be used to unpack these objects if necessary:

```
commit 1104tree 39b03c5e6da0e645b22cd97a38a1c0c2d79ac216
parent 9189c31b7f1c3f2e40133851ba3b6c39ffb704bd
author Mathias Scherer <scherer.mat@gmail.com> 1638372513 +0100
  :
  :
Adds flag to flag.html
```

This record refers to another item at `objects/39/b03c5e6da0e645b22cd97a38a1c0c2d79ac216`, which includes the following binary data. Notice the reference to a file called `flag.html`:

```
  :
  :
00000070: b569 d50f 924a 3130 3036 3434 2066 6c61  .i...J100644 fla
00000080: 672e 6874 6d6c 002e a9de 8ad7 2761 6140  g.html......'aa@
00000090: 08a3 cd0a b442 620a 9462 bd34 3030 3030  .....Bb..b.40000
  :
  :
```

This filename is followed by another ID: `2ea9de8ad72761614008a3cd0ab442620a9462bd`. And it turns out that the file at `2e/a9de8ad72761614008a3cd0ab442620a9462bd` is the page we want, containing the flag in plain view.

```
<h3><span><span><strong> </strong>Flag</span></span></h3>
<span style="font-size: 12pt !important;">Here is the flag:
HV{n3V3r_Sh0w_Y0uR_.git}</span>
```

Flag: `HV{n3V3r_Sh0w_Y0uR_.git}`

[HV21.05] X-Mas Jumper (Author: monkey) ← *That's me! I hope you liked it* 😁

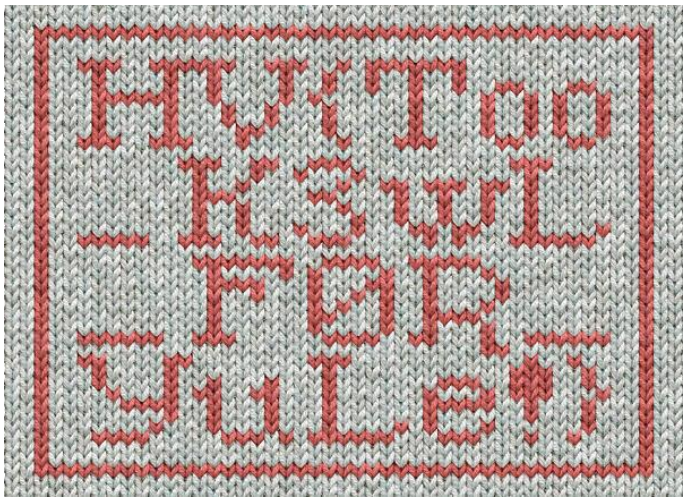
It looks like the elves forgot to resize the image properly when they scaled up the design to fit Santa's XXL dimensions. If the number of stitches per row is changed from 48 to 37 per row, the answer can clearly be seen:

```
XXX..XXXXXX..XXX..X.XXXXXXX.....X....X..X.
...X..X..X..X..X...........X....X..X....X..X..X.
.X..X..XX...XX..XXXXXX..X....X.X......X....X..X.
X..X.X....X...X..X...X.....X....X..X.X..X.X....X
...X..X...X.....X....X..X.X..X.X....X....XX....X
.....X....X..X.X..XXXX..XXX...X......X...XXX....
XX...XX.........................................
....XXX..XX..XXX...........XXX...........X...X.
.X...X..........X...........X..X......X..XX.X
XX.XX.X............XXX.....XX....X..X..X..X....
.........X..X......X...X..X..X.............X.
.X.......X..X..X..X..X............X..X...X...X.
..XX.XX...X....XXXXXX..XXX..XX..XXX.....X.X...XX
XXXX.........................................
..XXXXXXX..XXXX..XXXXX.................X....X.X
....X..X...X................XXXX...X...XX..X...
.X..............X..X...X..X.X..X....X.........
.......X......X.X..X..X...X..............X...
...XX...X..XXXX.................X......X....X..
X...X........XXXXXX..XXX......XXXX..XX...X..XXXX
XX.........................XX.......XX....XX.
.......XXX............X..X...X....X.........X..
..........XXX..X..X....X..X...X...X.......XXX..X
XX..X...X...X.XX..XX...X......X...X.XXX...X...XX
XX..X...X...X......XXXXX..X...X.......X..X...X..
.X......X......X...X......X...X...X....X.X..
.X......X..XXXX.....XXX.X.XXXXXX...XXX...X..X...
```

➔

```
XXX..XXXXXX..XXX..X.XXXXXXX.....
.X....X..X....X..X..X..X..X.....
.X....X..X....X..X..X..X..X..XX...XX.
.XXXXXX..X....X.X......X....X..X.X..X
.X....X...X..X...X.....X....X..X.X..X
.X....X...X..X...X.....X....X..X.X..X
.X....X....XX....X.....X....X..X.X..X
XXX..XXX...X......X...XXX....XX...XX.
.....................................
.......XXX..XX..XXX...........XXX....
........X...X..X...X...........X.....
........X..X......X..XX.XXX.XX.X.....
........XXX.....XX....X..X..X..X.....
........X..X......X...X..X..X..X.....
........X..X.......X..X..X..X..X.....
........X..X...X...X...XX.XX...X....X
XXXXX..XXX..XX..XXX.....X.X...XXXXXX.
.....................................
........XXXXXXX..XXXX..XXXXX.........
.........X....X.X....X..X...X........
.........XXXX...X...XX..X....X.......
.........X..X...X..X.X..X....X.......
.........X......X.X..X..X...X........
.........X......XX...X..XXXX.........
.........X......X....X..X...X........
XXXXXX..XXX......XXXX..XX...X..XXXXXX
.............................XX.......
XX....XX........XXX...........X..X..
.X....X..........X...........XXX..X.
.X....X..X...X...X.......XXX..XXX..X.
..X...X.XX..XX...X......X...X.XXX...X
...XXXX..X...X...X......XXXXX..X...X.
......X..X...X...X......X......X...X.
.....X...X...X...X....X.X...X......X.
.XXXX.....XXX.X.XXXXXX...XXX...X..X..
```

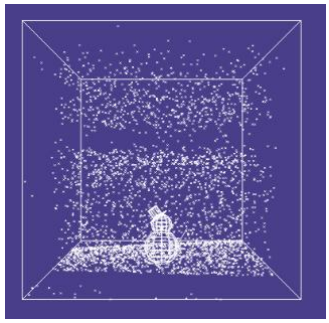
It's what all the best dressed elves are wearing this season!



Flag: `HV{Too_K3WL_F0R_YuLe!}`

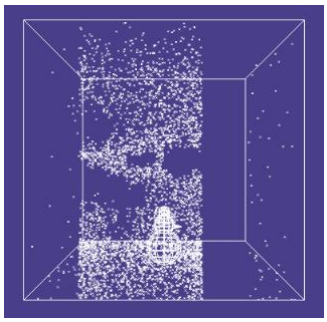
[HV21.06] Snow Cube (Author: Dr Nick)

This was a very creative challenge. The falling snowflakes spell out the characters of the flag, but only when viewed from the side. In the default front view, they look more or less random, and it's impossible to discern anything meaningful. But a quick inspection of the source code shows that this can be fixed by typing in `alpha = Math.PI / 2` at the JavaScript console.



Front view: Snow

Once that's done, it's easy enough to jot down each character of the flag as it scrolls past. With a bit of modification to the source code, it's also possible to display the entire flag in one image. This is what I ended up with:



Side view: The letter 'A'

```
<!DOCTYPE html>
<html lang="en">
  <body style="background: #fff">
        <canvas width="910" height="100" id="scan"></canvas>
    <script>
      const canvas = document.getElementById("scan");
      const context = canvas.getContext("2d");
      let b = 800;

      snowCube = [[800,800,800],[800,800,-800],...,[0,520,-133]];
      snowFlakes = [[670,-287,781,2.01],...,[-626,-20682,618,3.24]];
      let snowman = snowCube.length;

      for (let i = 0; i < snowFlakes.length; i++) {
        snowCube[snowCube.length] = [snowFlakes[i][0],snowFlakes[i][1],snowFlakes[i][2]];
      }

      context.fillStyle = "rgba(0,0,0,0.25)";
      let px = 0;

      while (px < canvas.width) {
        for (let i = snowman; i < snowCube.length; i++) {
          if (snowCube[i][1] < b*2) {
              oy = snowCube[i][1];
            snowCube[i][1] += snowFlakes[i - snowman][3] * 10;
            ny = snowCube[i][1];
            if (ny>=0 && oy<0 && px<canvas.width) {
                sy = 53 - snowCube[i][2] / 20;
                context.fillRect(px-1, sy, 2, 2);
            }
          }
          px+=0.00006;
        }
      }
    </script>
  </body>
</html>
```

And here's the result:

Flag: `HV21{M3SSAGE_OUT_OF_FLAKES}`

[HV21.08] Flag Service (Authors: nichtseb and logical overflow)

The challenge consists of a web server that appears to fall just short of providing you with a flag:

```
<!DOCTYPE html>
    <html>
    <head>
          :
          :
    </head>
    <body>
        <div id="flex-wrapper">
        <div id="container">
            <h1>Thanks for using the Flag service.<br/> Your Flag is:</h1>
            <h2>
```

On closer inspection, it turns out that this page is provided with a `Content-Length` header that claims the document consists of 1878 bytes. This appears to be correct; the `<h2>` tag ends exactly at the 1878th byte. So where is the flag?

It turns out that the `Content-Length` header is incorrect. If you access the web page by some other method that doesn't respect HTTP headers, then you get the whole document including the flag. I used `netcat` to do this as follows:

```
$ host=0027dd80-97db-41bc-b7e8-0f74543f465f.rdocker.vuln.land
$ echo -ne 'GET / HTTP/1.1\r\nHost: $host\r\n\r\n' | nc $host 80
HTTP/1.1 200 OK
Connection: close
Content-Type: text/html
Content-Length: 1878


<!DOCTYPE html>
    <html>
    <head>
          :
          :
    </head>
    <body>
        <div id="flex-wrapper">
        <div id="container">
            <h1>Thanks for using the Flag service.<br/> Your Flag is:</h1>
            <h2>HV21{4lw4y5_c0un7_y0ur53lf_d0n7_7ru57_7h3_53rv3r}</h2>
            </div>
        </div>
        </div>
    </body>
</html>
```

Flag: `HV21{4lw4y5_c0un7_y0ur53lf_d0n7_7ru57_7h3_53rv3r}`

[HV21.09] Santa's Shuffle (Author: 2d3)

Santa's code is in a right mess:

```
#include/*502_-_zU3X)}tM1#Hq$4D"35*/<stdio.h>//W6juf:tvs.]DrIoMM(axv0@|k?+jkES5r
#define/*&jhm|0zs(*/B/*zDq|:OHcU~Dv|;7,FE)9s(Ue!5gM*/break//v9BF(TT1Gq"19#?kJ2*H
#define/*JH8gDjl*/C(x)/*c9UOy:3*/case/*@MgHEK+94c9*/x/*bb]V+F#*/://u$T._.$ms'cjF
#define/*XSGrEWMy94I!VMe_n*/E(x)/*UUG9F{)zJB*/else/*CJsY*9D|SfgQ-XL*/x//s{2GfRjU
#define/*jDdwh4pU,*/F(x)/*@48h|llEw&qpgsJl7ifhb)*/if/*ux7-7_$}9*P*/(x)//s0qQes26
... (76 more lines of this garbage) ...
```

But wait a minute. If we treat this as C source code and delete the comments, the first line just reduces to:

```
#include <stdio.h>
```

It even compiles without any problems to produce a program that asks for a key, and then quits. Presumably it will output something more interesting if we provide the right key. To do this, we're going to have to disentangle the code somehow. We can use the C compiler to expand all the macros defined at the top of the file:

```
$ cc -E santa_shuffle.c >santa_shuffle.expanded.c
```

That results in a rather large source file that starts with the entire content of `stdio.h`, but ends with the de-obfuscated source code of this program. After using the C formatter at https://codebeautify.org/c-formatter-beautifier to make it more legible, it became clear that the program tests each input character against the following list of `char` values:

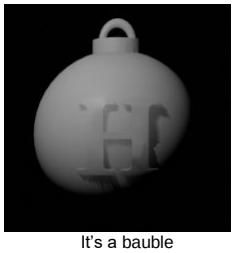
```
(1<<6)+2, (4<<4)+6, 0x34, 58*2, 104, 16*5+21, 0x57, 4*25+5, 0x4E, 36, 82, 105,
103, 104, 7*16+4, (2<<5)-1
```

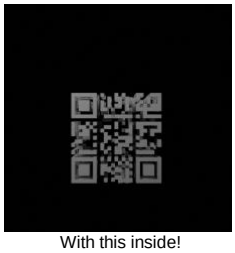
When converted to ASCII characters, this corresponds to the string `"BF4theWiN$Right?"`. So when the program asks for the key, enter this string and you'll get the flag. (So apparently that long `char*` variable at the start of the code is a BrainF*ck program. I never realised.)

Flag: `HV21{-HidDeN-bRaiNF-Ck-dEcoDer-}`

[HV21.15] Christmas Bauble (Author: Dr Nick)

This one was easy. STL files are used to store 3D objects. Blender (https://www.blender.org/) can open them no problem. This one contains a bauble. If you delete all the outside parts (enter edit mode, select a vertex, select connected vertices, press X to delete), then you get something that looks very much like a QR code. If you switch to an orthographic (i.e., non-perspective) camera and change the colours to get a matt black object on a white background, then you can scan the QR code directly from the screen. Rotate by 90° about the vertical axis to get another QR code, and by 90° about the horizontal axis to get a third. These give you three pieces of text that join to form the flag


It's a bauble


With this inside!

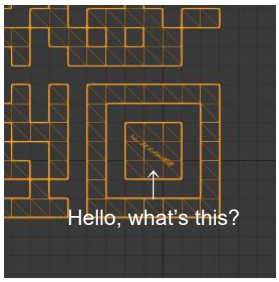

th3_fl4g_with_the_2nd


P4rt_c0mb1ned_w17th_th4t}

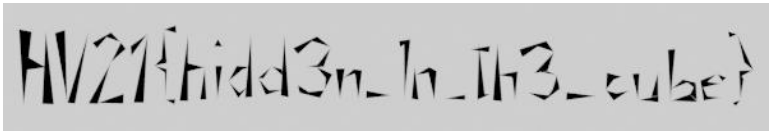

HV21{1st_P4rt_0f_

Flag: HV21{1st_P4rt_0f_th3_fl4g_with_the_2nd_P4rt_c0mb1ned_w17h_th4t}

[HV21-Hidden] Where did you find that??



If you look at this QR cube in wireframe mode, you can see something else lurking within. If you delete everything else and rotate this thing into view, it reveals another flag. Very sneaky!



Flag: HV21{hidd3n_ln_th3_cube}

[HV21.18] Lost Password (Author: monkey) ← *Me again!*

PDF files with embedded fonts leak information about which characters they use:

```
00001b10: 6f46 6f6e 742d 302d 3020 2f46 6f6e 7444  oFont-0-0 /FontD
00001b20: 6573 6372 6970 746f 720a 3135 2030 2052  escriptor.15 0 R
00001b30: 202f 456e 636f 6469 6e67 202f 4d61 6352   /Encoding /MacR
00001b40: 6f6d 616e 456e 636f 6469 6e67 202f 4669  omanEncoding /Fi
00001b50: 7273 7443 6861 7220 3335 202f 4c61 7374  rstChar 35 /Last
00001b60: 4368 6172 2031 3235 202f 5769 6474 6873  Char 125 /Widths
00001b70: 205b 2037 3230 0a30 2030 2030 2030 2030   [ 720.0 0 0 0 0
00001b80: 2030 2030 2030 2030 2030 2030 2037 3230   0 0 0 0 0 0 720
00001b90: 2030 2030 2030 2030 2030 2030 2037 3230   0 0 0 0 0 0 720
00001ba0: 2030 2030 2030 2030 2030 2030 2030 2030   0 0 0 0 0 0 0 0
00001bb0: 2030 2030 2030 2030 2030 2030 2037 3230   0 0 0 0 0 0 720
00001bc0: 0a30 2030 2037 3230 2030 2030 2030 2030  .0 0 720 0 0 0 0
00001bd0: 2030 2030 2030 2030 2037 3230 2037 3230   0 0 0 0 720 720
00001be0: 2030 2030 2030 2037 3230 2030 2030 2030   0 0 0 720 0 0 0
00001bf0: 2030 2030 2030 2030 2030 2030 2030 2030   0 0 0 0 0 0 0 0
00001c00: 2030 2030 2030 2030 2030 0a37 3230 2030   0 0 0 0 0.720 0
00001c10: 2030 2037 3230 2030 2030 2030 2030 2030   0 720 0 0 0 0 0
00001c20: 2030 2030 2030 2030 2030 2030 2030 2037   0 0 0 0 0 0 0 7
00001c30: 3230 2037 3230 2030 2030 2037 3230 2030  20 720 0 0 720 0
00001c40: 2037 3230 205d 203e 3e0a 656e 646f 626a   720 ] >>.endobj
```

In the table of character widths, unused code points are assigned a width of zero. Since the first character in this list has an ASCII value of 35 (based on `"/FirstChar 35"`), we can easily get a list of all 14 of the characters used in this file:

```
widths = [ 720,0,0,0,0,0,0,0,0,0,0,0,720,0,0,0,0,0,0,720,0,0,0,0,0,0,0,0,0,
           0,0,0,0,0,720,0,0,720,0,0,0,0,0,0,0,0,720,720,0,0,0,720,0,0,0,0,
           0,0,0,0,0,0,0,0,0,0,0,0,720,0,0,720,0,0,0,0,0,0,0,0,0,0,0,0,720,
           720,0,0,720,0,720 ]
print(''.join([chr(x+35) for (x,y) in enumerate(widths) if y>0]))

#/6EHQRVgjwx{}
```

We're told that the flag is of the form `HV{..........}` (14 characters in length, if anyone's counting), so we know where the characters 'H', 'V', '{' and '}' should go. The ten additional characters must therefore be some permutation of the string `#/6EQRgjwx`. A simple Python script can generate all the possible passwords of this form:

```
from itertools import permutations
for p in permutations('#/6EQRgjwx'):
    print('HV{' + ''.join(p) + '}')
```

Using the output of this script as a password list, the John the Ripper password cracker (https://www.openwall.com/john/) takes only a few seconds to find the correct one.

```
pdf2john.py password.pdf >pdfcrack
john --wordlist=pdf_password_list.txt pdfcrack
```

Flag: `HV{E6wRx#jQ/g}`