VishwaCTF
Challenges' write-ups

Team MadrHacks

MadrHacks

13-14 March 2021

Contents

Warmup

The two warmup challenges were supposed to be really easy and intended as a warm-up for the team.

Warmup

Flag Format ✓
25

Discord-bot ✓
356

Flag Format

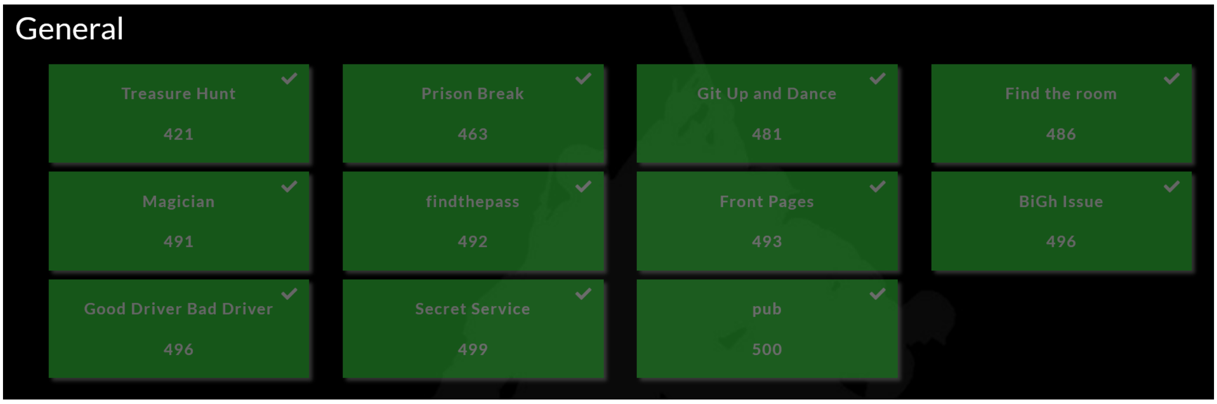
We are given the flag in the challenge description: `vishwaCTF{welcome_to_vishwaCTF}`.

Discord-bot

We knew from the description that bot commands started with `$`. After trying to get some `$help` from the bot, we tried `$flag`, which gave us the flag: `vishwaCTF{d15c0rd_5p1ll3d_th3_b34n5}`.

General

These challenges are aimed at testing your general skills, spanning a lot of different topics and requiring you to think outside of the box.



General

Treasure Hunt ✓
421

Prison Break ✓
463

Git Up and Dance ✓
481

Find the room ✓
486

Magician ✓
491

findthepass ✓
492

Front Pages ✓
493

BiGh Issue ✓
496

Good Driver Bad Driver ✓
496

Secret Service ✓
499

pub ✓
500

Treasure Hunt

The challenge was about finding three parts of a flag in three different social media accounts. We were provided with an Instagram, a Linkedin and a Twitter account.

- The first part, `w31c0m3`, was found in a comment on a post in the Instagram account.
- The second part, `_t0_`, was also found in the comment section of a post but this time in the Linkedin account.
- The third part, `v1shw4ctf`, was easily found on a tweet in the third account.

By assemblying the flag we ended up getting: `w31c0m3_t0_v1shw4ctf` which, enclosed in the usual format, was the correct flag.

`vishwaCTF{w31c0m3_t0_v1shw4ctf}`

Prison Break

We are given a link to *https://prisonbreak.vishwactf.com/*, which is a simple web decision-based game. The challenge description states that we need to make the correct choices in order to get out of prison (in the game) and obtain thus the flag.

The game is about a man named Zed, who is a thief and has been put in jail for stealing gold from a bank. In the game, we impersonate Zed and we need to get out of jail.

First of all we need to press two times `1` in order to start the game. The first choiche asks us about whether we want to accept the carceration and stay in jail, or if we would like to try and escape. Obviously, we press `2`. At the following step we decide to be kind and greet the jailer (`1`), then we tell him we have understood what he tells us about the prison rules (again `1`). After that, we decide to arrange out things and take some rest (`1`).

The following day we are waken up by an alarm bell and our cell neighbour greets us. Again, we decide to be kind and we introduce ourselves (`1` and then `1` again). At the following step we decide to go and find Ted, our cell neighbour (`2`). Being a little bit shy, we decide not to tell him why we were put in jail (`2`), but after that we tell him anyway in order not to look rude. After Ted's question, we decide to go for the wood workshop (`1`). When presented to Fred, we decide to accept the gum (`1`) as it may be useful later.

At lunch we decide once again to be kind with Ted and we ask him where he works (`1` and then `1` again).

After a couple of weeks of observing and gathering information, we decide that it's time to plan our escape (`1`). In order to fabricate the cell keys, we decide to hide our pieces of wood in our bottle (`2` and then `1`). After having built the keys, we decide to use the broom to open the cell door as this is the only way of escaping we have right now (`1`). Unfortunately, the keys get stuck and falls on the floor, so we decide to use the gums we got previously to try to pick them up (`1` and then `1` again).

At the alarm sounding (`1`) we are inspected because we didn't wake up, and the guard notices a piece of wood in our cell. We answer that it is used to propup the photos (`2`), then we decide to hide our keys properly (`1`). After that, we decide to ask Fred map of the outside (`1`) and to wait for a chance to get to know what's there between the cell and the outside (`1` and then `1` again).

We finally decide to escape from the south-east gate (`2`) and at early day (`1`) as there is more people and we have less chances of being discovered. At the end, we made it! We just press `1` to get the flag: `vishwaCTF(G@mE_0f_DeC1$ions)`

To summarize, the entire sequence to win the game and obtain the flag is the following:

```
(1 1 2 1 1 1 1 1 2 2 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 1 1)
```

Git Up and Dance

We are given a zip file which contains a git repo. We start by investigating all the history of the files.

With `git log -p workspace.a4362daf.js | grep vish` we get (among other lines) the following:

```
This is the flag vishwaCTF{d4nc3_4nd_giitupp}.
```

So we submit the flag: `vishwaCTF{d4nc3_4nd_giitupp}`.

Find the room

This challenge asks us to find the room number for the principal's office in VIIT.

Searching with Google Maps for `Vishwakarma Institute of Information Technology`, we found the building and used `street view` to look for the correct room. This lead us to a courtyard where the plaque `Principal's Office` was visible and under it was the room number `A 003`, which we used as our flag:

```
vishwaCTF{A 003}
```

Magician

This was a cron job, giving us a single character of the flag at a time, once every ~20 minutes. After collecting all the characters (this was 12 hours long!), we managed to assemble the flag.

```
vishwaCTF{cr0nj0bs_m4k3_l1f3_s1mp13}
```

findthepass

We are given a rar file. This file contains a VM (a `VirtualBox` save), so we proceed to import it into `VirtualBox`.

In the home directory we find `this_is_what_you_need/wordlist.txt`. We try all the listed passwords (with `su`). The password `password` gives us a root shell (this can be confirmed with `whoami`).

We submit the flag: `vishwaCTF{password}`.

Front Pages

What is the front page (TM) of the internet? Reddit obviously, so let's search.

Searching for `vishwaCTF` on Reddit gives us an interesting user: `u/vishwaCTF`

Looking through the post history of the account we can find a *post* that mentions a deleted flag!

Let's search for this post inside *wayback machine*… and sure enough we have a flag on the *first snapshot*!

`vishwaCTF{0$dVl_1z_kFV3g_0a3mT0graD}`

This flag is not correct though, we first need to decrypt it using the Vigenere chiper. With the key `VISHACTF` we obtain the correct flag:

`vishwaCTF{0$iNt_1s_oFT3n_0v3rL0okeD}`

BiGh Issue

The challenge description mentions the frontpage website for the CTF and a *very big issue*. Maybe we could try searching on GitHub?

We can easily find the *link to the github repo*, so let's look at the closed issues.

We can find an issue named *Huge Issue* and by carefully looking at the comments we can see that one of these was edited, let's see what was changed… and sure enough in the history we have our flag!

`vishwaCTF{bh41yy4_g1thub_0P}`

Good Driver Bad Driver

For this challege we split the labelled set in:

- 75% training set, and

- 25% of validation set.

The features were distance and speeding. We decide to use the Random Forest Classifier as our model. We train that model using the training set. For computing the accuracy, we use the validation set in order to compute the accuracy, which was `1.0`.

Finally, we do the prediction on the unlabelled set (test set) and we find the driving class for each item.

`vishwaCTF{d4t4_5c13nc3_15_n3c3554ry}`

Secret Service

The challenge provides an image called `cicada.png` and tells us to find 3 prime numbers.

The first number is provided in the description itself and it's `3301`. After inspecting the image and its properties, the other two prime numbers are found in its size: `1019x911` pixels.

Referring to the original Cicada 3301 puzzle, we multiplied the three numbers and used them as the needed string, leading to correct flag:

`vishwaCTF{www.3064348009.com}`

pub

For this challenge we were given an `apk` and told to go through a list of Marvel movies.

After installing the apk on an Android emulator, we noticed that one of the movies was called `external_package`. At this point we tried going through the apk's archive but didn't find anything useful. We then tried to see if the name of the app was of any use. We tried going to `pub.dev`, Flutter's package repository, and looked for this `external_package` and found it. On its page there was a link to a github repository and by looking at the commits, we noticed `pubspec.yaml`. Going through the file we found a long string of `pub/spec`:

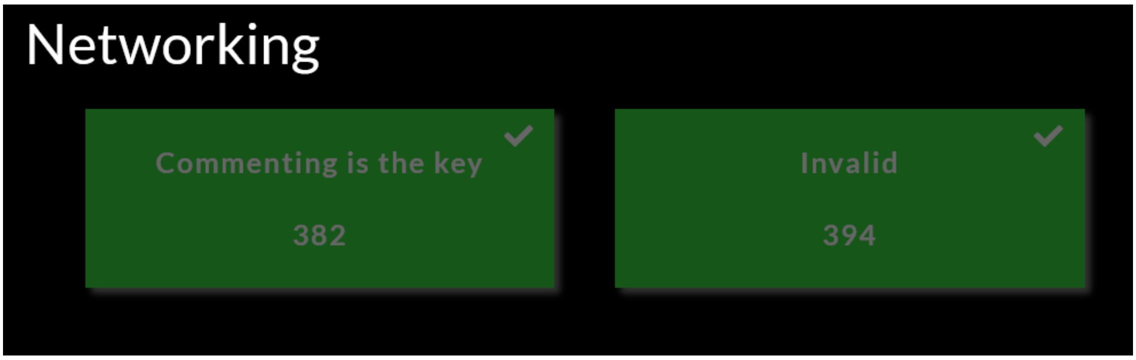
```
pubpubpubspec pubpub pubpubpub pubpubpubpub pubspecspec pubspec
    specpubspecpub spec pubpubspecpub{pubpubspec pubpubpub
    pubpubpubspecspec pubpubspecpub pubpubspec pubspecspecspecspec
    pubpubspecspecpubspec pubpubspecpub pubspecspecspecspec pubpubspec
    spec spec pubpubpubspecspec pubspecpub pubpubspecspecpubspec
    pubspecspecpub pubspecspecpubspecpub specpubspecpub specpubspec
    pubspec specspecpub pub
```

Given that the string was fairly long, we tried to convert it to morse (using `pub` as `.`, and spec as `-`) and deciphered the morsed code. The result was the actual flag:

`vishwaCTF{US3FU1_F1UTT3R_P@CKAGE}`

Networking

The two challenges that follow are about networks.



Networking

Commenting is the key ✓
382

Invalid ✓
394

Commenting is the key

We are given a simple pcapng and we opened it with Wireshark. Packet 5 and 12 are commented. The comment statesthe following:

`flag==packets_are_editable`

The flag is thus: `vishwaCTF{packets_are_editable}`.

Invalid

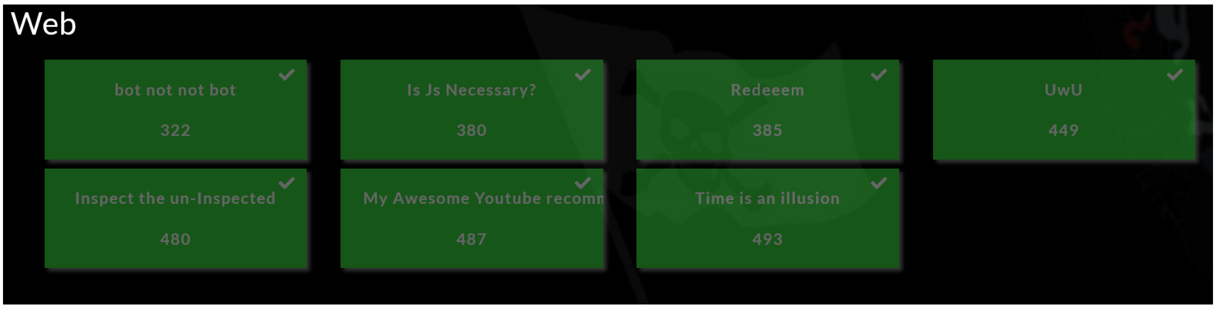
We are given a simple pcapng (the same as in *Commenting is the key*), and again we opened it with Wireshark.

Packet 32 is a `SIP 403 Wrong Password`, while the conversation starts at Packet 20. The source IP of this conversation is 212.242.33.35, so the flag is:

`vishwaCTF{212.242.33.35}`

Web

All the web challenges were pretty easy and targeted some known web vulnerabilities.



bot not not bot

The challenge has a webpage with 500 links on it. We can therefore write a simple command to download them all:

```
touch out
for i in {1..500}
do
  curl "https://bot-not-not-bot.vishwactf.com/page$i.html" >> out
done
```

In this way we can obtain all the indexes.

Most of them are like

```
<html><head><title>bot-not-not-bot1</title></head><body><p>Useless Page<br>-1</p></body></html>
```

but on some you can find

```
<html><head><title> bot-not-not-bot8</title></head><body><h1>v</h1><p>Useful Page<br>0</p></body></html>
```

On the latest example you can find the letter of the flag, `v`, and its position on the flag, `0`.

The flag is `vishwaCTF{r0b0t_15_t00_0P}`.

Is Js Necessary?

This one take us to a page from where we are immediately redirected to google. We can disable redirect to view the page content.

Example (Firefox):

```
about:config
search for javascript
set javascript.enabled to false
```

If we reload the page without javascript, we find the question "how many days did Brendan take to develop this language?". Look for the answer on google, we find the answer which is `10`. We type it, submit the answer and get the flag:

`vishwaCTF{2ava5cr1pt_can_be_Dis@bleD}`

Redeeem

Redeem propose us to buy some flags, but it also state that we're poor.

Open it on firefox, and open the *Network* section of the developer tools. We try to buy the flag, and we can see the request to `handle.php`. In the request parameter, we can see the fields `current` and `buy`. We then press *Edit and Resend*, set `current` to `10000` and `buy` to `0`. We click on the new generated request, and we can find the flag in the *response* section.

`vishwaCTF{@DDed_T0_C@rT_}`

UwU

UwU welcomes us with a cool music and video.

Since there's nothing on the *home* and *about* sections, we try to look for a hint in the description of the challenge. The description states *when php, anime and robot come together...*, and we get the hint! We try to look for the *robots.txt* file and we get the text:

```
this time.. there might be a directory called as robots lol
```

So we connect to the `/robot` directory, where there's a php file and we can see its source. The source looks for the get parameter `php_is_hard`, and compare it to `suzuki_harumiya` after replacing the occurrence `suzuki_harumiya` in its value with nothing. To bypass this simple check, we enter the get parameter `suzuki_suzuki_harumiyaharumiya`. By doing this, the `preg_replace` function will replace the occurrence which is found in the middle of the string, leaving it as `suzuki_harumiya` .

We get the flag in the response, which is `vishwaCTF{well_this_was_a_journey}`.

Inspect the un-Inspected

This challenge has no links, and tells us something about *home*, *practice* and *ask question*.

The idea is to look in the home of the ctf website for something in the source code. So we go to `https://vishwactf.com/`, right click and look for the source code. By looking for the word flag, we find the first part of the flag in the comment `//Flag part 1/3 : vishwaCTF{EvEry_`.

By looking on the `practice` section, we get redirected to `play-vishwactf-mini.ml` and we can find the `flag` link in links above, near `Users`,`Teams` and the CTF logo. By looking at the html code, we get the second part of the flag which is `C0iN_ha$`.

The last part of the flag is on the `faq` page source code, and it is `_3_s1Des}`.

We now have the full flag, which is `vishwaCTF{EvEry_C0iN_ha$_3_s1Des}`.

My Awesome Youtube recommendation

This challenge has an app make in *Flask*, and redirects us to *YouTube* querying our input.

First, we need to block the redirection. We can done this on firefox with:

```
about:config
search for accessibility.blockautorefresh
set it to true
```

Now, by submitting our query, we get redirected to `results?query=examplequery`. Since the app is made in *Flask* and the text is displayed in the response, we immediately think about *Server Side Template Injection*. One way to try this vulnerability for *Flask*, is to use the common payload `{{7*7}}` in the query field. This gives us the expected result, by substituting the payload with the result (`49`) in the response.

We can try to look for common configuration object in *Flask*, such as `config`. This gives us the configuration of the server, and we can find the flag inside.

`vishwaCTF{th3_f14g_ln_c0nflg}`

Time is an illusion

This challenge allows us to see the source code.

From the source code, we can see two things:

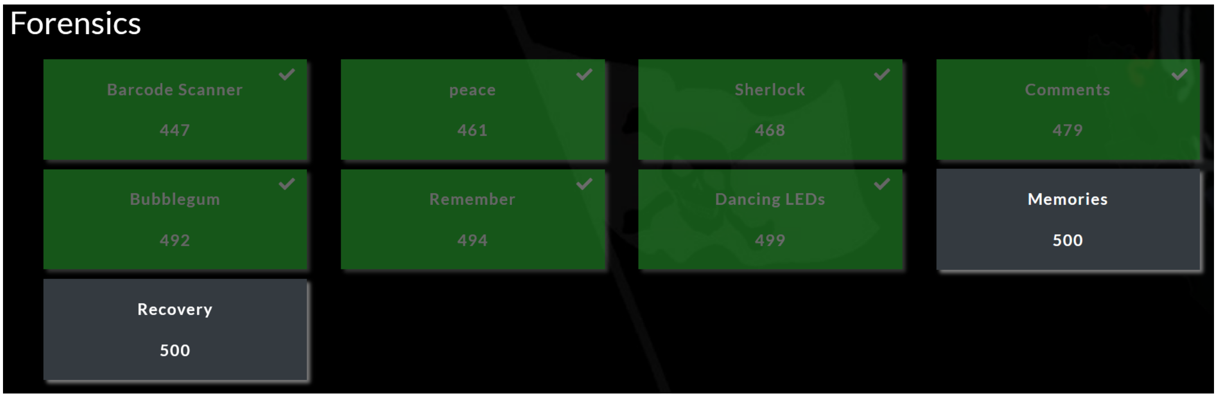
- The key must be of 5 characters, otherwise we get an error;
- Every character of the key is compared to the variable `let_check` one by one, and if the character matches the program executes a `usleep(1000000)`, so the loading time will be `1` second longer.

We can write a simple script to automate the requests and find the flag:

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

import requests
```

```
from string import ascii_letters, digits
import time
from pwn import *

url = "https://time-is-an-illusion.vishwactf.com/handle.php"

alphabet = ascii_letters + digits

p = log.progress('PASSWORD')
p2 = log.progress('ELAPSED')
pwd = "K"
while len(pwd) != 5:
    for l in alphabet:
        time.sleep(0.1)
        curr_pwd = pwd + l
        curr_pwd += '?' * (5-len(curr_pwd))
        p.status(curr_pwd)
        start = time.time()
        response = requests.get(url, params={'key':curr_pwd})
        elapsed = time.time() - start

        p2.status(str(elapsed))

        if elapsed > len(pwd) + 1:
            pwd += l
            break
```

Forensics



Forensics

Barcode Scanner ✓
447

peace ✓
461

Sherlock ✓
468

Comments ✓
479

Bubblegum ✓
492

Remember ✓
494

Dancing LEDs ✓
499

Memories
500

Recovery
500

Barcode Scanner

We are given a simple jpeg image. The challenge description states that it is unreadable and that we should find a way to read it.

We tried to open it with Gimp, invert its colors (i.e. black becomes white and viceversa), then we scanned it with Google Lens and there it is! We enclosed the flag in the usual format and this challenge is solved.

`vishwaCTF{5oo_3asY}`

peace

We are given a simple rar archive and it is password-protected. With hashcat we crack the password: `india`. We find a `wav` file. It is clearly a morse-code transmission. We use `fldigi` to decode that.

We get the following sequence:

```
76 69 73 68 77 61 63 74 66 7B 37 68 33 79 5F 34 72 45 5F 46 30 72 33 66 65 37 31 6E 67
```

By decoding the hex we get the flag:

`vishwactf{7h3y_4rE_F0r3fe71ng}`

Sherlock

We are given a JPEG image. We open it with `stegsolve`. Plotting the *Gray bits* we notice a noisy column on the right. With `Analyse -> Data Extract` we get the flag by extracting the *LSB* of the *green channel* by *columns*.

We submit the flag: `vishwaCTF{@w3s0Me_sh3Rl0cK_H0m3s}`.

Comments

We are given a `docx` file. Given a `docx` is just a `zip` with custom extension, we can extract the contents. We are left with three folders and one file. The interesting folder is `word`, as it contains all the pages data.

We use `cat word/*| grep wishwaCTF` and we find `<!--vishwaCTF{comm3nts_@r3_g00d}-->`, that gives us the flag:

`vishwaCTF{comm3nts_@r3_g00d}`

Bubblegum

We are given an audio file, `bkk.wav`, and told to simplify the lyrics of a particular section of the song.

By playing the audio file we noticed noise around the `00:18` mark. Inspecting the spectogram, the noise was added to visualize in the spectogram the phrase `0.55-1.07`. We understood that this was the section of the lyrics to simplify and looked them up. We ended up with `oh bubble gum dear im yours forever i would never let them take your bubblegum away`, which was the correct flag.

`vishwaCTF{oh bubble gum dear im yours forever i would never let them take your bubblegum away}`

Remember

We are given a two files. We execute `file` on them, and we get `MS Windows registry file, NT/2000 or above`.

So we use `regripper` to understand them. With `regripper -r file2 -p samparse` we find the information needed:

```
Username         : Shreyas Gopal [1001]
Full Name        :
User Comment     :
Account Type     :
Name             :
Password Hint    :
Last Login Date  : 2013-01-10 08:24:36Z
Pwd Reset Date   : 2013-01-10 08:24:36Z
Pwd Fail Date    : Never
Login Count      : 5
Embedded RID     : 1001
  --> Password not required
  --> Password does not expire
  --> Normal user account
```

We check that the weekday for `2013-01-10 08:24:36` was a Thursday, and thus we get the flag:

`vishwaCTF{thursday_january_10_08_24_36_2013}`

Dancing LEDs

We are given a screen recording of some blinking LEDs. We write down the led values: 1 for ON, 0 for OFF.

We get the following:

```
0110100
1101001
```

```
1110000
1011010
1001010
1001000
1111010
1111000
0110100
0110001
```

We decode the binary: `4ipZJHzx41`, but this is not the flag. The video title is `Video58`, so we apply Base58 and we get `b1!nk3r`.

The flag is `vishwaCTF{b1!nk3r}`.

Reverse Engineering

Apollo 11

This challenge provides an *iso* image.

By running the command `strings` on the *iso*, we obtain all the printable strings which are contained in the file. At this point, we only need to filter them in some way. Since we know the flag format, which starts with `vishwaCTF{`, we can use the command `grep` to filter the result of `strings` and get only what we're looking for!

By running `strings Apollo11.iso | grep vishwaCTF`, we get the following output:

`vishwaCTF{I50_1s_A_MEs5}`,

which is the flag for the challenge.

Rotations

Let's download the binary and run it:

```
> file mm
mm: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
    dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID
    [sha1]=9d1420344c3a7c70c70b68b947ef2ec8ae498eb1, for GNU/Linux
    3.2.0, not stripped

> ./mm
```

The binary waits for some input so let's see what happens:

```
> ./mm
hello
EWWWW DUMBBB
```

Okay… I don't think this will lead anywhere so let's debug the program since it's not stripped.

```
gdb mm

gdb> start
```

We can now check if we have some interesting function inside the binary:

```
gdb> info functions
```

And sure enough we find something:

```
0x00005555555551a9  flag
```

We can try to call it and see what happens:

```
gdb> jump flag
Continuing at 0x5555555551b1.
ivfujnPGS{s1Nt_1f_e0g4gRq_Ol_!3}[Inferior 1 (process 2578) exited
    normally]
```

We have something that looks like a flag, but it's not quite correct. Remembering the name of the challenge we can easily see it's rotated with the Caeser cipher, so let's try to decrypt it.

Trying a couple of rotations we can finally get our flag with a shift of 13:

`vishwaCTF{f1Ag_1s_r0t4tEd_By_!3}`

Misleading Steps

The challenge description suggests that we might have something misleading inside, and sure enough when running strings on the binary we find something that is not our flag:

`vishwaCTF{1_0ft3n_M1sl3ad_pPl}`

So let's search some more!

By inspecting the binary with objdump we can find something interesting inside the main section:

```
> objdump -d mislead -M intel
...
    126f:       c7 85 50 ff ff ff 76    mov    DWORD PTR [rbp-0xb0],0x76
    1276:       00 00 00
    1279:       c7 85 54 ff ff ff 69    mov    DWORD PTR [rbp-0xac],0x69
    1280:       00 00 00
    1283:       c7 85 58 ff ff ff 73    mov    DWORD PTR [rbp-0xa8],0x73
    128a:       00 00 00
    128d:       c7 85 5c ff ff ff 68    mov    DWORD PTR [rbp-0xa4],0x68
    1294:       00 00 00
    1297:       c7 85 60 ff ff ff 77    mov    DWORD PTR [rbp-0xa0],0x77
    129e:       00 00 00
...
```

and so on…

Taking all the values up to the end of the section we can recover the flag in hexadecimal:

```
7669736877614354467b556d4d5f77336952446f6f6f305f315f416d5f7468335f7233346c5f306e337d
```

And by converting it to ASCII we get our points!

`vishwaCTF{UmM_w3iRDooo0_1_Am_th3_r34l_0n3}`

Facile

For this challenge we have a file with a weird extension, let's inspect it:

```
 > file s1mple.gzf
s1mple.gzf: Java serialization data, version 5
```

This doesn't seem to help. Maybe `binwalk` can help us, let's see what it finds.

```
 > binwalk s1mple.gzf

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
56            0x38            Zip archive data, at least v2.0 to
                              extract, name: FOLDER_ITEM
```

Let's extract the archive!

```
 > binwalk -e s1mple.gzf
 > cd _s1mple.gzf.extracted/
 > ls
38.zip   FOLDER_ITEM
```

We have something interesting!

```
 > file FOLDER_ITEM
FOLDER_ITEM: data
```

Okay, simply trying with `strings` we can extract a lot from this file that seems to contain executable informations. This is enough to find the flag:

```
 > strings FOLDER_ITEM | grep vishwa
vishwaCTF{r3v_1t_1s5s5s}
```

Give it to get it

This challenge gives us the flag, but asks to provide the right input for the given program to produce the flag.

By executing it, we clearly see that it prints something based on the argument given. So we try to execute it with the following command:

`/a.out 444444555555555`

and we get the output:

```
Here's your flag darling...
DDDUUUU
```

Since we know that 44 is the hexadecimal value for the character *D*, and 55 is the hexadecimal value for the character *U*, the execution of the program looks clear to us: it translates every digits we give to ascii, reading it as hex, and print it back to us. Now, let's go on cyberchef and encode the desired payload to hex:

```
7669736877614354467b663134675f31735f57683372335f5468335f68336152745f4c3145737d
```

When we run `a.out` with this payload, we get the response:

```
Here's your flag darling...
vishwaCTF{f14g_1s_Wh3r3_Th3_h3aRt_L1Es
```

So, it looks like we need to add something more to it. Let's run it with the argument `7669736877614354467 b663134675f31735f57683372335f5468335f68336152745f4c3145737d00`

```
Here's your flag darling...
vishwaCTF{f14g_1s_Wh3r3_Th3_h3aRt_L1Es}
```

We got the full flag, and now just submit

```
7669736877614354467 b663134675f31735f57683372335f5468335f68336152745f4c3145737d00
```

in the website and get the points.

Suisse

We are given a binary. The description says stuff about the LUHN checksum, but that's totally useless. We can simply reverse it with ghidra or use gdb to call the function `_flag()`, which prints out the following chars:

```
111 88 107 81 113 93 52 118 56 104 102 88 85 104
```

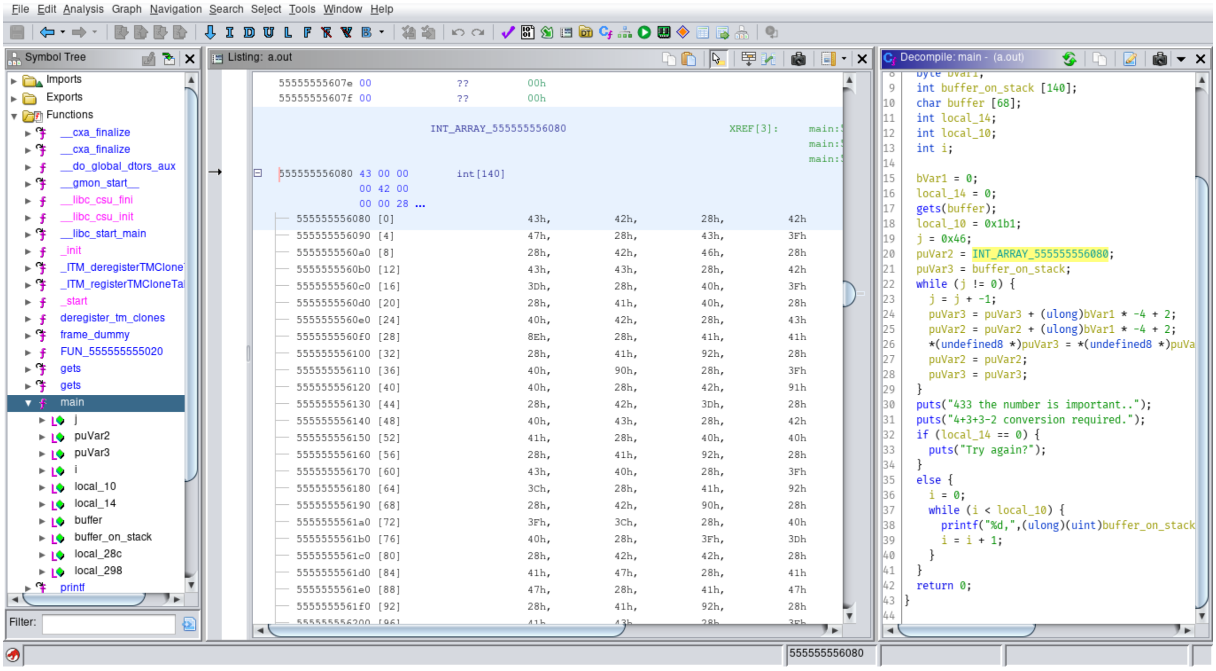
As the description was hinting, we subtract 3 from each of these and convert it to ascii:

```
lUhNnZ1s5ecURe
```

So the flag was: `vishwaCTF{lUhNnZ1s5ecURe}`.

FlowRev

We are given a binary. Reversing it with ghidra we can find that there is a weird int array.



```
55555555607e 00          ??          00h
55555555607f 00          ??          00h

                   INT_ARRAY_555555556080                  XREF[3]:     main:
                                                                        main:
                                                                        main:
555555556080 43 00 00        int[140]
             00 42 00
             00 00 28 ...
   555555556080 [0]          43h,      42h,      28h,      42h
   555555556090 [4]          47h,      28h,      43h,      3Fh
   5555555560a0 [8]          28h,      42h,      46h,      28h
   5555555560b0 [12]         43h,      43h,      28h,      42h
   5555555560c0 [16]         3Dh,      28h,      40h,      3Fh
   5555555560d0 [20]         28h,      41h,      40h,      28h
   5555555560e0 [24]         40h,      42h,      28h,      43h
   5555555560f0 [28]         8Eh,      28h,      41h,      41h
   555555556100 [32]         28h,      41h,      92h,      28h
   555555556110 [36]         40h,      90h,      28h,      3Fh
   555555556120 [40]         40h,      28h,      42h,      91h
   555555556130 [44]         28h,      42h,      3Dh,      28h
   555555556140 [48]         40h,      43h,      28h,      42h
   555555556150 [52]         41h,      28h,      40h,      40h
   555555556160 [56]         28h,      41h,      92h,      28h
   555555556170 [60]         43h,      40h,      28h,      3Fh
   555555556180 [64]         3Ch,      28h,      41h,      92h
   555555556190 [68]         28h,      42h,      90h,      28h
   5555555561a0 [72]         3Fh,      3Ch,      28h,      40h
   5555555561b0 [76]         40h,      28h,      3Fh,      3Dh
   5555555561c0 [80]         28h,      42h,      42h,      28h
   5555555561d0 [84]         41h,      47h,      28h,      41h
   5555555561e0 [88]         47h,      28h,      41h,      47h
   5555555561f0 [92]         28h,      41h,      92h,      28h
   555555556200 [96]         41h,      43h,      28h,      3Fh
555555556080
```

```
  byte bVar1;
  int buffer_on_stack [140];
  char buffer [68];
  int local_14;
  int local_10;
  int i;

  bVar1 = 0;
  local_14 = 0;
  gets(buffer);
  local_10 = 0x1b1;
  j = 0x46;
  puVar2 = INT_ARRAY_555555556080;
  puVar3 = buffer_on_stack;
  while (j != 0) {
    j = j + -1;
    puVar3 = puVar3 + (ulong)bVar1 * -4 + 2;
    puVar2 = puVar2 + (ulong)bVar1 * -4 + 2;
    *(undefined8 *)puVar3 = *(undefined8 *)puVa
    puVar2 = puVar2;
    puVar3 = puVar3;
  }
  puts("433 the number is important..");
  puts("4+3+3-2 conversion required.");
  if (local_14 == 0) {
    puts("Try again?");
  }
  else {
    i = 0;
    while (i < local_10) {
      printf("%d,",(ulong)(uint)buffer_on_stack
      i = i + 1;
    }
  }
  return 0;
}
```

There was also a very basic buffer overflow (notice the use of `gets()`), but as we didn't have a server to connect it was pretty much useless.

It wasn't really clear, but the string `4+3+3-2 conversion required` was pointing that the weird int array was encoded using octal. Decoding them from octal gave us the flag:

`vishwaCTF{U_M4naGeD_t0_m0D1fYYY_W3ll_d3ser\/3d}`

Useless App

We're given an apk. It appears that it is not possible to install it using adb/qemu, not even after signing it correctly. However, we can extract it's content using jadx or apktools.

The MainActivity is the following:

```
package com.example.demo_app;

import io.flutter.embedding.android.FlutterActivity;
import kotlin.Metadata;

@Metadata(bv = {1, 0, 3}, d1 = {"\u0000\f\n\u0002\u0018\u0002\n\u0002\
    u0018\u0002\n\u0002\b\u0002\u0018\u000002\u000020\u0001B\u00005¢\u0006\
    u0002\u0010\u0002¨\u0006\u0003"}, d2 = {"Lcom/example/demo_app/
    MainActivity;", "Lio/flutter/embedding/android/FlutterActivity;", "
    ()V", "app_debug"}, k = 1, mv = {1, 1, 15})
/* compiled from: MainActivity.kt */
public final class MainActivity extends FlutterActivity {
}
```

We can clearly find out that the apk is made using Flutter, probably in debug mode (`app_debug`). Searching online for a bit we found out that a Flutter app compiled in debug mode contains the source code in the `kernel_blob.bin` file.

That file contained an interesting function (found out via `strings kernel_blob.bin | grep '$flag'-C 20`:

```
void getthefl0g() {
    String text = "";
    String flag = "";
    int y = 0, d = 0;
    for (y = 0; y < 32 - 1; y += 2, d++) {
      String te = "0x" + text.substring(y, y + 2);
      if (d % 2 == 0) {
        flag = flag + String.fromCharCode((int.parse(te) ^ 0x32));
      } else {
        flag = flag + String.fromCharCode((int.parse(te) ^ 0x23));
      }
    }
    print("$flag");
}
```

But we are missing the text variable!

We also noticed a comment (which are left untouched in debug mode):

```
//what is triangular number series ?
```

After a lot of fiddling, we found an interesting hex string in the app's resources (in the `resources/res/values` directory).

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="status_bar_notification_info_overflow">999+</string>
    <string name="string_name">4
       b616e316e404467796c67207265c065617455646c792673616964217468617420086f6c6d657
       </string>
</resources>
```

Decoding the string from hex gave us the following sentence:

```
Kan1n@Dgylg reÀeatUdly&said!that .olmesPwas i`spired%by the beal-lifU
    figure%of Josepd Bell, a Curgeon atPthe Royal%Infirmary af Edinburgh
    . whom ConanPDoyle met if 1877 and hag worked for aÃ a clerk. Lik.
    Holmes, Bell#was noted for arawing broad coNclusions from minute
    observations.[12] However, he later wrote to Conan Doyle: "You are
    yourself Sherlock Holmes and well you know it".[13] Sir Henry
    Littlejohn, Chair of Medical Jurisprudence at the University of
```

```
        Edinburgh Medical School, is also cited as an inspiration for Holmes
        . Littlejohn, who was also Police Surgeon and Medical Officer of
        Health in Edinburgh, provided Conan Doyle with a link between
        medical investigation and the detection of crime.
```

Searching for the last part of the string, which seems to be unchanged, we can find out this was a sentence from wikipedia, but it wasn't useful.

After a lot of time and many failures, we found out that we had to eventually extract half bytes from the hex string following the triangular number series as indexes.

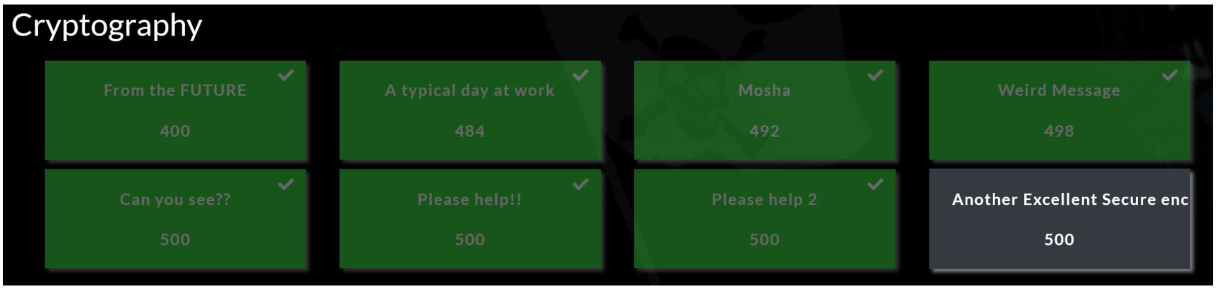
```
#!/usr/bin/env python

string_hex = "4
    b616e316e404467796c67207265c065617455646c792673616964217468617420086f6c6d6573507
    "

original_hex = original.hex()

def tn(n, start_from=0):

    if start_from == 0:
        i, t = 1, 0
    elif start_from == 1:
        i, t = 2, 1
    while i <= n:
        yield t
        t += i
        i += 1

def pick_tn(fromhere, tnstart):
    c_string = ""
    for i in list(tn(32, tnstart)):
        c_string += fromhere[i]
    return c_string

"""
void gettheflOg() {
    String text = "";
    String flag = "";
    int y = 0, d = 0;
    for (y = 0; y < 32 - 1; y += 2, d++) {
      String te = "0x" + text.substring(y, y + 2);
      if (d % 2 == 0) {
        flag = flag + String.fromCharCode((int.parse(te) ^ 0x32));
      } else {
        flag = flag + String.fromCharCode((int.parse(te) ^ 0x23));
      }
    }
    print("$flag");
"""
```

```
def get_flag(text):
    flag = b""
    assert(len(text) == 32)
    for i in range(0, 16):
        if i % 2 == 0:
            flag += bytes([ord(bytes.fromhex(text[i*2:i*2+2])) ^ 0x32])
        else:
            flag += bytes([ord(bytes.fromhex(text[i*2:i*2+2])) ^ 0x23])
        #print(flag.hex())
    return flag

text = pick_tn(string_hex, 0)
print(text)
flag = get_flag(text)

print(flag)
```

This was the flag: `vishwaCTF{y0u_d3buggg3d_!7}`.

Cryptography

The following challenges required cryptographic techniques in order to be solved.



Cryptography

From the FUTURE ✓
400

A typical day at work ✓
484

Mosha ✓
492

Weird Message ✓
498

Can you see?? ✓
500

Please help!! ✓
500

Please help 2 ✓
500

Another Excellent Secure enc
500

From the FUTURE

We were given an image, `note.png`, which featured a messagge written in an unfamiliar alphabet.

Since the challenge's description talked about `Futurama` we were able to find the series' alien alphabet and used it to decipher the message which was: `WEARENOTALONE`.

`vishwaCTF{WEARENOTALONE}`

A typical day at work

The description of the challenge says that we have to decode the following message:

yonvkahj_on_jeyonx_jeajon

It is obviously a monoalphabetic cipher. That means that we have to analyse the characters used to compose the words and try to identify the correct ones.

We notice that the last word has the first and the fourth character identical, so by using a dictionary (a list of all english words) we can search for a word made up of two identical letters (at first and fourth position) and four different letters. Using this information, we can now search for 8-characters and 6-characters words which have this property:

- the 8-character word has 4 characters belonging to the last word and the others all different
- the 6-character word has 4 characters belonging to the last word and the other all different

For each last word found in our dictionary, we can now search for all possible 8-characters and 6-characters words that satisfy those requirements.

```python
f = open("dictionary.txt", "r")
lines = f.readlines()

def substitution(phrase, line, lines):
    string = ""
    for c in phrase:
        if c == 'j':
            string = string + line[0:1]
        elif c == 'e':
            string = string + line[1:2]
        elif c == 'a':
            string = string + line[2:3]
        elif c == 'o':
            string = string + line[4:5]
        elif c == 'n':
            string = string + line[5:6]
        else:
            string = string + c

    if string[9:11] == "is" or string[9:11] == "on" or string[9:11] == "at" or string[9:11] == "if" or string[9:11] == "it" or string[9:11] == "or" or string[9:11] == "to" or string[9:11] == "an":
        print(string)
        print_different_words_8(lines,line)
        print_different_words_6(lines,line)

def word_with_different_chars_8(line):
    for i in range(8):
        if (line[i:i+1] in line[:i]+line[i+1:]):
            return False
    return True

def print_different_words_8(lines, line):
    for l in lines:
        if ((len(l[:len(l)-1]) == 8)):
            if (l[7:8] == line[0:1] and l[1:2] == line[4:5] and l[2:3] == line[5:6] and l[5:6] == line[2:3]):
                if (word_with_different_chars_8(l)):
                    print(l)

def word_with_different_chars_6(line):
    for i in range(6):
        if (line[i:i+1] in line[:i]+line[i+1:]):
            return False
    return True

def print_different_words_6(lines, line):
    for l in lines:
        if ((len(l[:len(l)-1]) == 6)):
            if (l[0:1] == line[0:1] and l[1:2] == line[1:2] and l[3:4] == line[4:5] and l[4:5] == line[5:6]):
                if (word_with_different_chars_6(l)):
                    print(l)

def print_words(lines):
    for l in lines:
        if ((len(l[:len(l)-1]) == 8)):
            if (word_with_different_chars_8(l)) :
                print(l)

for line in lines:
    phrase = "yonvkahj_on_jeyonx_jeajon"
    if (len(line[:len(line)-1]) == 6):
        if (line[0:1] == line[3:4]):
            if (line[1:2] != line [0:1] and line[1:2] != line[2:3] and line[1:2] != line[3:4] and line[1:2] != line[4:5] and line[1:2] != line[5:6]):
                if (line[2:3] != line [0:1] and line[2:3] != line[3:4] and line[2:3] != line[4:5] and line[2:3] != line [5:6]):
                    if (line[3:4] != line [4:5] and line[3:4] != line [5:6]):
                        if (line[4:5] != line [0:1] and line[4:5] != line[5:6]):
                            substitution(phrase, line[:len(line)], lines)
```

Flag: `vishwaCTF{congrats_on_second_season}`.

Mosha

We were given an image, `moshatxt.jpg`, which featured a message written in an unfamiliar alphabet.

We found an account on IG called mosha_font and here we found the strange alphabet. Using this alphabet we decipher the message which was the flag.

`vishwaCTF{Y0u4reM05hAnoW}`

Weird Message

We are given a long bitstring. This string is 50879 bit long (50880 - 1 newline). It has a lot of 0s, so we decide to plot it on an image.

We do that using `PIL`. Given the resulting image is still confused (the odd rows are reversed (?)), we decided to plot only the even rows.

```python
from PIL import Image

with open("message.txt", "r") as f:
    pixels = f.readline()[:-1]

    w, h = 613, 83

    img = Image.new("L", (w, h))
    for i, p in enumerate(pixels):
        x = i % w
        y = i // w
        c = 0 if p == "0" else 255
        if y % 2 == 0:
            img.putpixel((x, y), c)

    img.save("plot.png")
```

With that code we get the flag: `vishwaCTF{pr1m35_4r3_w31rd}`.

Can you see??

This challenge gave us a text file, `can_you_see.txt`, which contained 5 binary matrices, all 3-bits high.

It took us a while to figure out the `1`s and `0`s were used to represent words in braille. By using a braille translator, we managed to arrive to `vvho n33ds 3y3s 7o 5ee` which was the correct flag.

`vishwaCTF{vvho n33ds 3y3s 7o 5ee}`

Please help!!

This challenge provides a file with some binary strings on it. The strings are twelve binary digits each.

In the description, we can clearly read the words *distortion*,*noise*, *correct* and *decode*. This makes us think about some kind of correction code.

Given the fact that the strings are 12 binary digits, we think of the *hamming code* and try to apply that and extract the data. By applying correction code in the string, we don't get anything useful, but doing that in the reversed string and the reversing the result (Same as doing that enumerating the bits in the opposite order).

So, we have some work to do:

```
1. (check)

    1  2  3  4  5  6  7  8  9  10 11 12
    P  P  D  P  D  D  D  P  D  D  D  D
    0  0  1  1  0  1  1  0  0  1  1  1
p1  0     1     0     1     0     1    = 1
p2     0  1        1  1        1  1    = 1
p4           1  0  1  1              1 = 0
p8                       0  0  1  1  1 = 1
1011=11
10110101 -> garbage

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  1  1  0  0  1  1  0  1  1  0  0
p1     1     0     1     0     1     0 = 1
p2     1  1        1  1        1  0    = 1
p4  1              1  1  0  1          = 0
p8  1  1  1  0  0                      = 1
1011= 11
10110101 -> garbage

2. (check)
    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    0  1  0  0  1  0  0  1  1  0  0  0
p1     1     0     0     1     0     0 = 0
p2     1  0        0  0        0  0    = 1
p4  0              0  0  1  1          = 0
p8  0  1  0  0  1                      = 0
0010= 2
01000010 -> B

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    0  0  0  1  1  0  0  1  0  0  1  0
p1     0     1     0     1     0     0 = 0
p2     0  0        0  0        0  1    = 1
p4  0              0  0  1  0          = 1
p8  0  0  0  1  1                      = 0
0110= 6
01101000 -> h


3.
    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    0  1  0  1  1  1  0  0  1  0  1  1
p1     1     1     1     1     0     1 = 1
p2     1  0        1  0        0  1    = 1
p4  0              1  0  0  1          = 0
p8  0  1  0  1  1                      = 1
1011= 11
00011000 -> garbage

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  1  0  1  0  0  1  1  1  0  1  0
p1     1     1     0     1     0     0 = 1
p2     1  0        0  1        0  1    = 1
p4  1              0  1  1  1          = 0
p8  1  1  0  1  0                      = 1
1011= 11
01101001 -> i

4.
    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  1  0  1  0  1  0  0  0  1  0  1
p1     1     1     1     0     1     1 = 1
p2     1  0        1  0        1  0    = 1
p4  1              1  0  0  0          = 0
p8  1  1  0  1  0                      = 1
1011= 11
10011001 -> garbage

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  0  1  0  0  0  1  0  1  0  1  1
p1     0     0     0     0     0     1 = 1
p2     0  1        0  1        0  1    = 1
p4  1              0  1  0  1          = 1
p8  1  0  1  0  0                      = 0
0111= 7
00110101 -> 5

```

```
5.
    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    0  0  0  1  1  0  1  0  1  1  1  1
p1     0     1     0     0     1     1 = 1
p2     0  0        0  1        1  1    = 1
p4  0              0  1  0  1          = 0
p8  0  0  0  1  1                      = 0
0011= 3
00010100 -> garbage

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  1  1  1  0  1  0  1  1  0  0  0
p1     1     1     1     1     0     0 = 0
p2     1  1        1  0        0  0    = 1
p4  1              1  0  1  1          = 0
p8  1  1  1  1  0                      = 0
0010= 2
01011111 -> _

6.
    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  0  0  1  1  0  1  0  0  1  1  0
p1     0     1     0     0     1     0 = 0
p2     0  0        0  1        1  1    = 1
p4  1              0  1  0  0          = 0
p8  1  0  0  1  1                      = 1
1010= 10
10110101 -> garbage

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    0  1  1  0  0  1  0  1  1  0  0  1
p1     1     0     1     1     0     1 = 0
p2     1  1        1  0        0  0    = 1
p4  0              1  0  1  1          = 1
p8  0  1  1  0  0                      = 0
0110= 6
01101110  -> n
01110110  -> v

7.
    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  0  0  1  1  1  1  0  0  1  1  1
p1     0     1     1     0     1     1 = 0
p2     0  0        1  1        1  1    = 0
p4  1              1  1  0  0          = 1
p8  1  0  0  1  1                      = 1
```

```
1100= 12
00011101 -> garbage

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  1  1  0  0  1  1  1  1  0  0  1
p1     1     0     1     1     0     1 = 0
p2     1  1        1  1        0  0    = 0
p4  1              1  1  1  1          = 1
p8  1  1  1  0  0                      = 1
1100= 12
01101110 -> n
01110110 -> v

8.
    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  1  1  1  1  1  0  1  0  0  0  1
p1     1     1     1     1     0     1 = 1
p2     1  1        1  0        0  0    = 1
p4  1              1  0  1  0          = 1
p8  1  1  1  1  1                      = 1
1111=  ??????

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  0  0  0  1  0  1  1  1  1  1  1
p1     0     0     0     1     1     1 = 1
p2     0  0        0  1        1  1    = 1
p4  1              0  1  1  1          = 0
p8  1  0  0  0  1                      = 0
0011= 3
01100001 -> a

9.
    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  1  0  1  0  1  1  0  0  1  0  0
p1     1     1     1     0     1     0 = 0
p2     1  0        1  1        1  0    = 0
p4  1              1  1  0  0          = 1
p8  1  1  0  1  0                      = 1
1100=12
01011101 -> ]

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    0  0  1  0  0  1  1  0  1  0  1  1
p1     0     0     1     0     0     1 = 0
p2     0  1        1  1        0  1    = 0
p4  0              1  1  0  1          = 1
```

```
p8  0  0  1  0  0                      = 1
1100= 12
00110101 -> 5

10.
    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    0  1  0  1  1  0  1  0  1  1  0  1
p1     1     1     0     0     1     1 = 0
p2     1  0        0  1        1  0    = 1
p4  0              0  1  0  1          = 0
p8  0  1  0  1  1                      = 1
1010=10
01110101 -> u

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  0  1  1  0  1  0  1  1  0  1  0
p1     0     1     1     1     0     0 = 1
p2     0  1        1  0        0  1    = 1
p4  1              1  0  1  1          = 0
p8  1  0  1  1  0                      = 1
1011= 11
01011111 -> _

11.
    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    0  1  0  0  1  1  0  0  0  1  0  0
p1     1     0     1     0     1     0 = 1
p2     1  0        1  0        1  0    = 1
p4  0              1  0  0  0          = 1
p8  0  1  0  0  1                      = 0
0111=7
01000001 -> A

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    0  0  1  0  0  0  1  1  0  0  1  0
p1     0     0     0     1     0     0 = 1
p2     0  1        0  1        0  1    = 1
p4  0              0  1  1  0          = 0
p8  0  0  1  0  0                      = 1
1011= 11
01100110 -> f

12.

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    0  0  1  1  1  0  1  0  0  1  0  1
```

```
p1     0     1     0     0     1     1 = 1
p2     0  1        0  1        1  0    = 1
p4  0              0  1  0  0          = 1
p8  0  0  1  1  1                      = 1
111 = ????

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  0  1  0  0  1  0  1  1  1  0  0
p1     0     0     1     1     1     0 = 1
p2     0  1        1  0        1  0    = 1
p4  1              1  0  1  1          = 0
p8  1  0  1  0  0                      = 0
0011= 3
01010101 -> U

13.

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  1  0  0  1  1  0  1  1  1  1  1
p1     1     0     1     1     1     1 = 1
p2     1  0        1  0        1  1    = 0
p4  1              1  0  1  1          = 0
p8  1  1  0  0  1                      = 1
1001=9
11011011 -> garbage

    12 11 10 9  8  7  6  5  4  3  2  1
    D  D  D  D  P  D  D  D  P  D  P  P
    1  1  1  1  1  0  1  1  0  0  1  1
p1     1     1     0     1     0     1 = 0
p2     1  1        0  1        0  1    = 0
p4  1              0  1  1  0          = 1
p8  1  1  1  1  1                      = 1
1100= 12
01110110 -> v
01101110 -> n
```

We couldn't recover the first character, but we can clearly read the flag as `7hi5_vva5_fUn` /`thi5_vva5_fUn`.

The flag is `vishwaCTF{7hi5_vva5_fUn}`.

Please help 2

We are given some binary data organized in chunks of 8 bits (with are not plain ASCII btw, we checked). In addition, the challenge description states that this challenge is similar to the previous one, thus involving some kind of Hamming Code FEC.

Searching deep in the web, we found *this video* that explains a FEC scheme based on Hamming that works on 16-bit 4x4 squares and uses 5 bits of parity and 11 bits of data. It works as follows:

```
PPPX
PXXX
PXXX
XXXX
```

The "P" places are parity bits, while the "X" places are data bits. The places are numbered by rows. You can compute where a single bit error is by applying XOR operation in the following patterns:

```
OOOO
OOOO
XXXX
XXXX

OOOO
XXXX
OOOO
XXXX

OOXX
OOXX
OOXX
OOXX

OXOX
OXOX
OXOX
OXOX
```

Then you concatenare the four XOR bits, and this gives you the position of the incorrect bit, starting from zero.

Therefore, we try this code on the given binary data, pairing the chunks two by two. Follows a little scheme that summarizes what we did by hand (just the first chunks):

```
CHUNK 1:
0011  0
1011  0
0010  1
1011  1
-> Error on 3rd bit
-> Data bits: 0-011-010-1011

CHUNK 2:
0010  1
0100  0
```

```
1001  0
1011  1
-> Error on 9th bit
-> Data bits: 0-100-101-1011

...
```

By performing the described procedure, we were able to find a single incorrect bit for each 16-bit chunk of data, and therefore extracted the 11 bits of data in each block. Finally, using CyberChef, we decoded this data and got the flag (that needed to be enclosed in the standard format, btw)!

`vishwaCTF{5imil4r_y37_diff3r3n7!}`

References

1. https://google.com/
2. https://pequalsnp-team.github.io/cheatsheet/writing-good-writeup
3. https://ryankozak.com/how-i-do-my-ctf-writeups/