

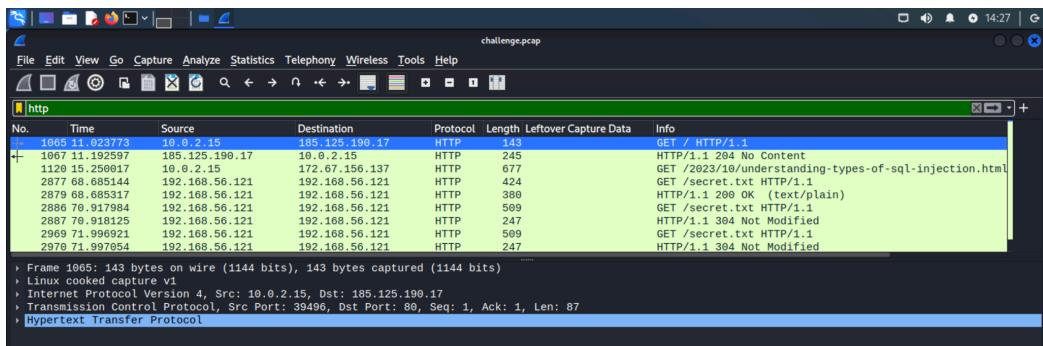
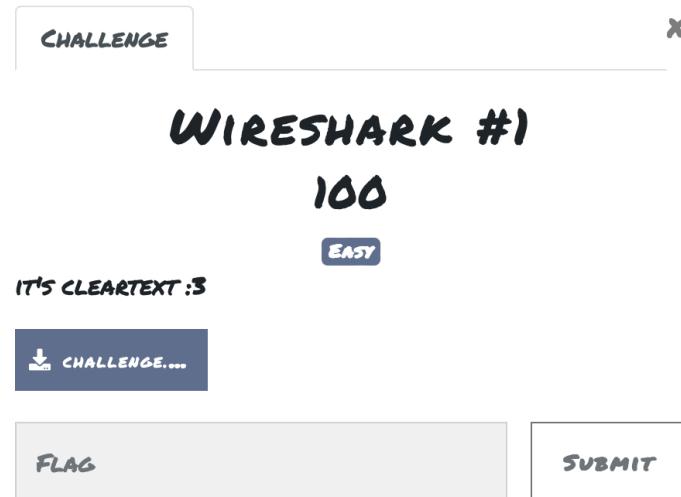


Girls in CTF
2023

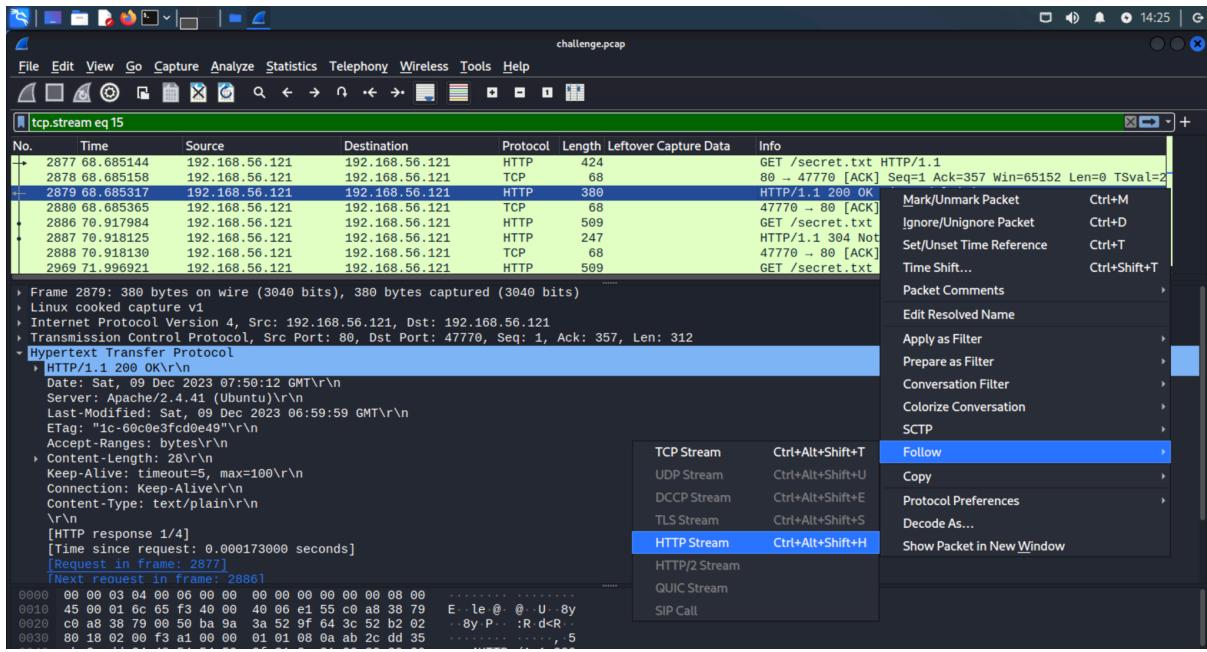
Wrote by

Dark Phoenix

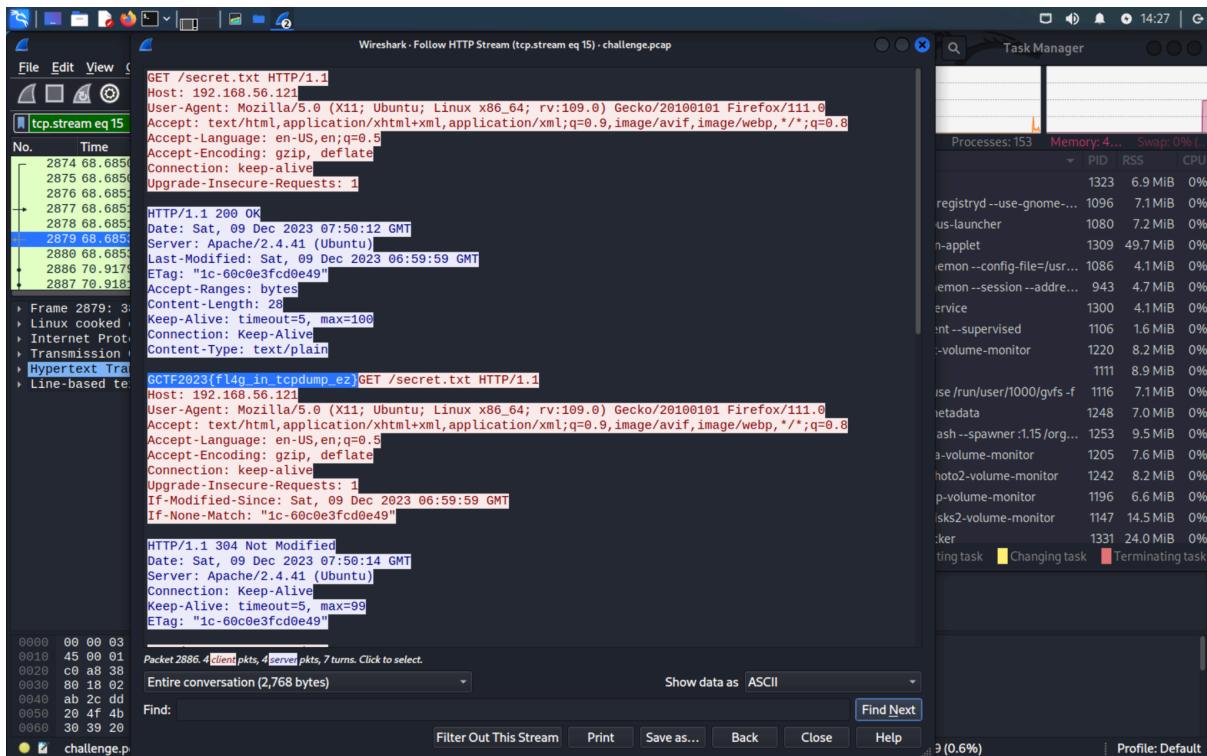
WIRESHARK #1



So firstly, I opened the file and applied a filter for http in the wireshark. Just like the hint given, it is clear text, and there is a suspicious plain txt file named “**secret.txt**” in the info.



Then I just right click on the pcap, and follow the **HTTP Stream**.



As shown above, I have highlighted the flag in the **HTTP Stream**.

Flag: GCTF2023{fl4g_in_tcpdump_ez}

I LOVE PNG!

CHALLENGE

X

I LOVE PNG!

447

EASY

FLAG IS LITERALLY IN FRONT OF YOUR EYES. NOT OBVIOUS ENOUGH?

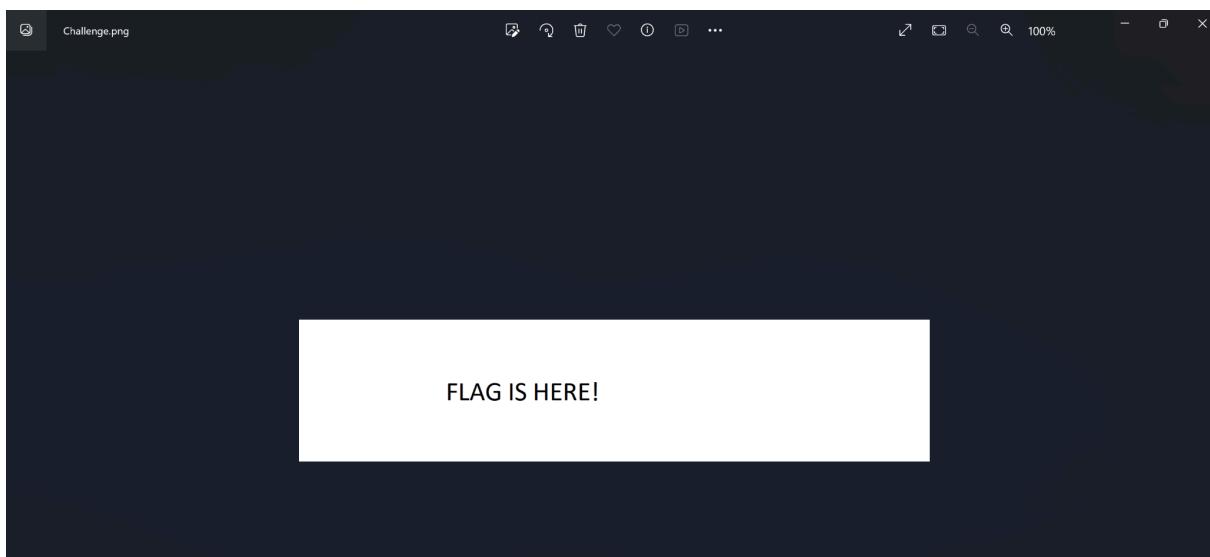
VIEW HINT

CHALLENGE....

FLAG

SUBMIT

Initially I couldn't open the file in mac os or kali linux, but then I tried to open it in windows vm and it works fine there.

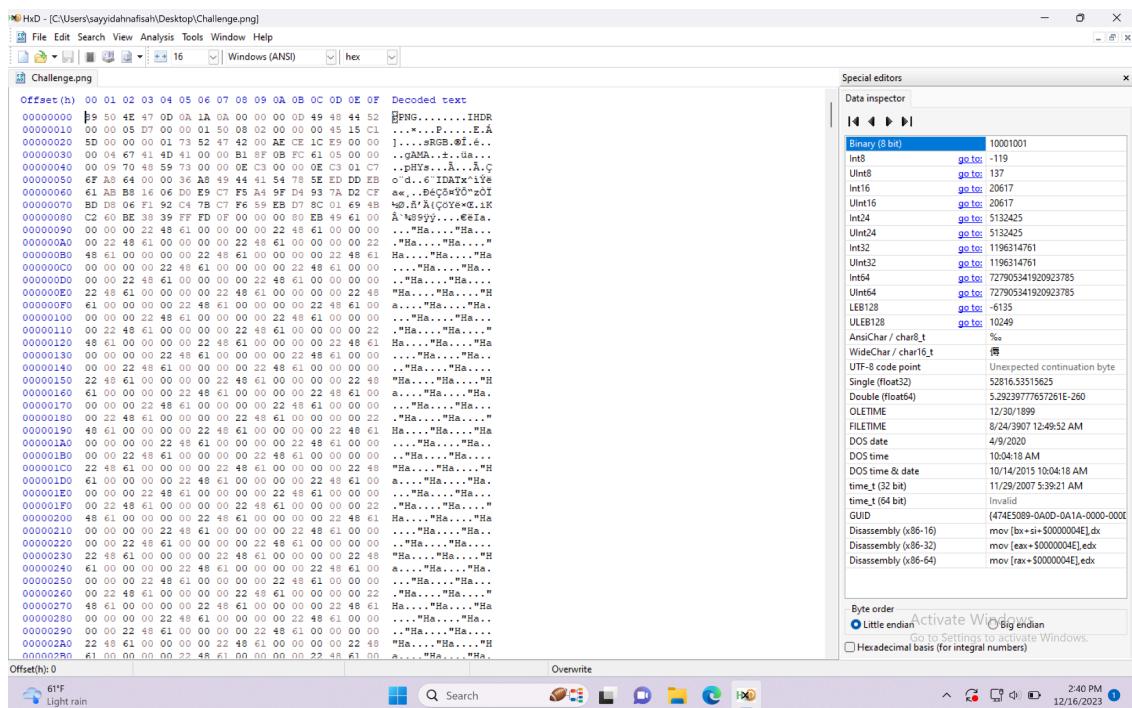


So this is how the image originally looks like. However, this was the only thing on the image and there was no flag. The image also looks like it has been cropped.

```

File Actions Edit View Help
  (parallels@kali-linux-2022-2) [~/Desktop/GCTF2023]
$ exiftool Challenge.png
ExifTool Version Number : 12.67
File Name : Challenge.png
Directory :
File Size : 14 kB
File Modification Date/Time : 2023:12:15 22:02:31+09:00
File Access Date/Time : 2023:12:16 14:37:51+09:00
File Inode Change Date/Time : 2023:12:16 14:35:02+09:00
File Permissions : -rw-r--r--
File Type : PNG
File Type Extension : image/png
MIME Type : image/png
Image Width : 1495
Image Height : 336
Bit Depth : 8
Color Type : RGB
Compression : Deflate/Inflate
Filter :
Interlace : Adaptive
SRGB Rendering : Noninterlaced
Gamma : Perceptual
Pixels Per Unit X : 2.2
Pixels Per Unit Y : 3779
Pixel Units : meters
Image Size : 1495x336
Megapixels : 0.502
  
```

Then I use **exiftool** to see the image info. As we can see here, the height is 336. This info will be used later to edit the image's height.



Then, I opened the file using **HxD**, a hex editor program to edit the hex value of the image. So the numbers here are in hex values.

```
Last login: Sat Dec 16 12:37:23 on ttys000
(base) sayyidahnaifah@Sayyidahs-MacBook-Air ~ % python
Python 3.10.9 (main, Mar 1 2023, 12:33:47) [Clang 14.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
[>>> hex(336)
'0x150'
[>>> hex(1000)
'0x3e8'
>>>
```

Then, I used python to convert the height that I retrieved from the information earlier into a hex value and the hex value for 336 is **0x150**.

I also converted 1000 to hex value because I'm going to change the image height from 336 to 1000. The new height will be **0x3E8**

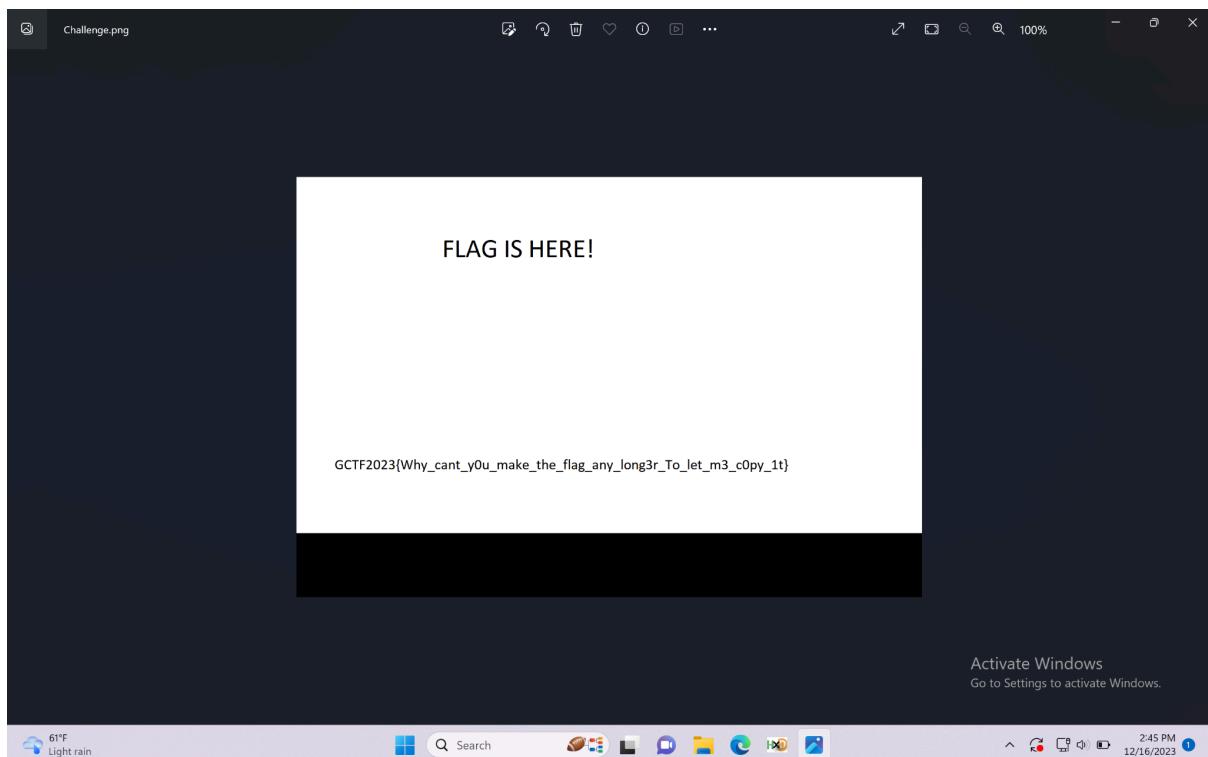
Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Decoded text
00000000	89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52	%PNG.....IHDR
00000010	00 00 05 D7 00 00 01 50 08 02 00 00 00 45 15 C1	...x...E.....E.Á
00000020	5D 00 00 00 01 73 52 47 42 00 AE CE 1C E9 00 00]....sRGB.ØÍ.é..
00000030	00 04 67 41 4D 41 00 00 B1 8F 0B FC 61 05 00 00	..gAMA..±..üa...
00000040	00 09 70 48 59 73 00 00 0E C3 00 00 0E C3 01 C7	..pHs...Ã...Ã.Ç
00000050	6F A8 64 00 00 36 A8 49 44 41 54 78 5E ED DD EB	o'd..6"IDATx^iÝé
00000060	61 AB B8 16 06 D0 E9 C7 F5 A4 9F D4 93 7A D2 CF	a<...Đéçô¤ÝÔ"zòÍ
00000070	BD D8 06 F1 92 C4 7B C7 F6 59 EB D7 8C 01 69 4B	¾Ø.ñ'À(QÖYé¢.iK
00000080	C2 60 BE 38 39 FF FD 0F 00 00 00 80 EB 49 61 00	Â~%89yy....€éIa.
00000090	00 00 00 22 48 61 00 00 00 22 48 61 00 00 00	..."Ha...."Ha...

As I have highlighted, that is the height that we are going to change. To know which value to change, I simply just look for 01 and 50 because the original height is 0x150.

```
HxD - [C:\Users\sayyidahnafisah\Desktop\Challenge.png]
File Edit Search View Analysis Tools Window Help
16 | Windows (ANSI) | hex |
Challenge.png

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text
00000000 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52 %PNG.....IHDR
00000010 00 00 05 D7 00 00 03 E8 b8 02 00 00 00 45 15 C1 ...x... ... . 
00000020 5D 00 00 00 01 73 52 47 42 00 AE CE 1C E9 00 00 ]....sRGB.  . 
00000030 00 04 67 41 4D 41 00 00 B1 8F 0B FC 61 05 00 00 ..gAMA.. .. ...
00000040 00 09 70 48 59 73 00 00 0E C3 00 00 0E C3 01 C7 ..pHYs... ... . 
00000050 6F A8 64 00 00 36 A8 49 44 41 54 78 5E ED DD EB o" .6"IDATx^i  
00000060 61 AB B8 16 06 D0 E9 C7 F5 A4 9F D4 93 7A D2 CF ak...    z  
00000070 BD D8 06 F1 92 C4 7B C7 F6 59 EB D7 8C 01 69 4B % . ' (  Y    . 
00000080 C2 60 BE 38 39 FF FD 0F 00 00 00 80 EB 49 61 00 A^%89  ...  Ia.
00000090 00 00 00 22 48 61 00 00 00 00 22 48 61 00 00 00 ..."Ha...."Ha...
000000A0 00 22 48 A1 00 00 00 00 22 48 A1 00 00 00 00 22 "Ha...."Ha...."
```

The red hex values are the new height of the image. 03 E8 is the hex value for 1000 which is 0x3E8.



After I edited and saved the new hex value, I reopened the png file and there's the flag!

Flag: GCTF{Why_cant_y0u_make_the_flag_any_long3r_To_let_m3_c0py_1t}

BROKEN

CHALLENGE

X

BROKEN

100

EASY

I GOT THIS PDF AND IT'S VERY IMPORTANT TO ME. WHY
CAN'T IT OPEN OMG!

 CHALLENGE....

FLAG

SUBMIT



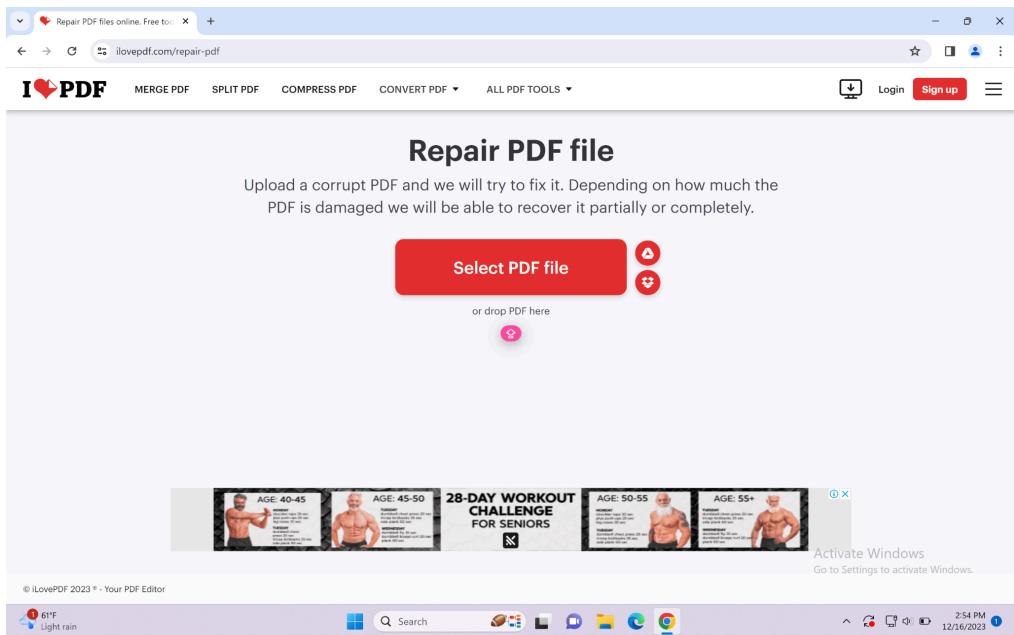
The file “Challenge.pdf” could
not be opened.

It may be damaged or use a file format
that Preview doesn’t recognise.

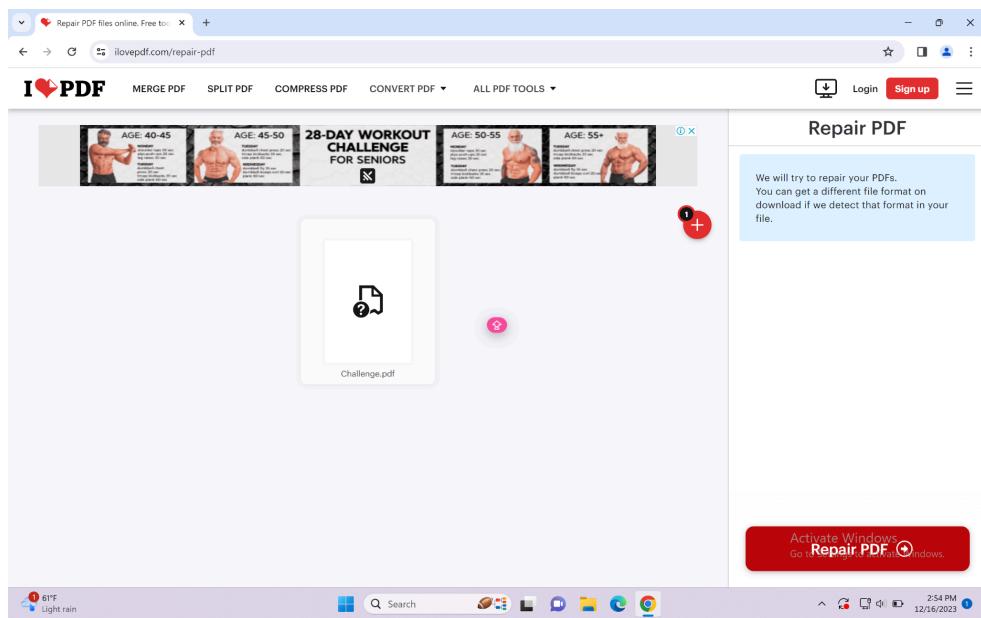
OK

So we were given a broken pdf file. Then I googled and found a website to repair pdf files. The website link is as below:

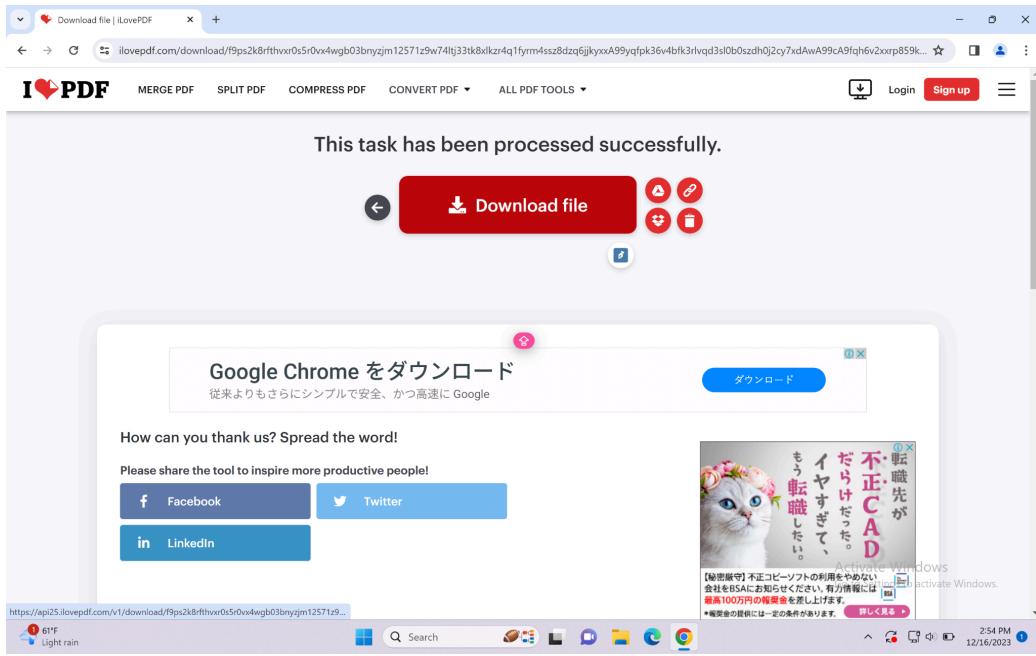
<https://www.ilovepdf.com/repair-pdf>



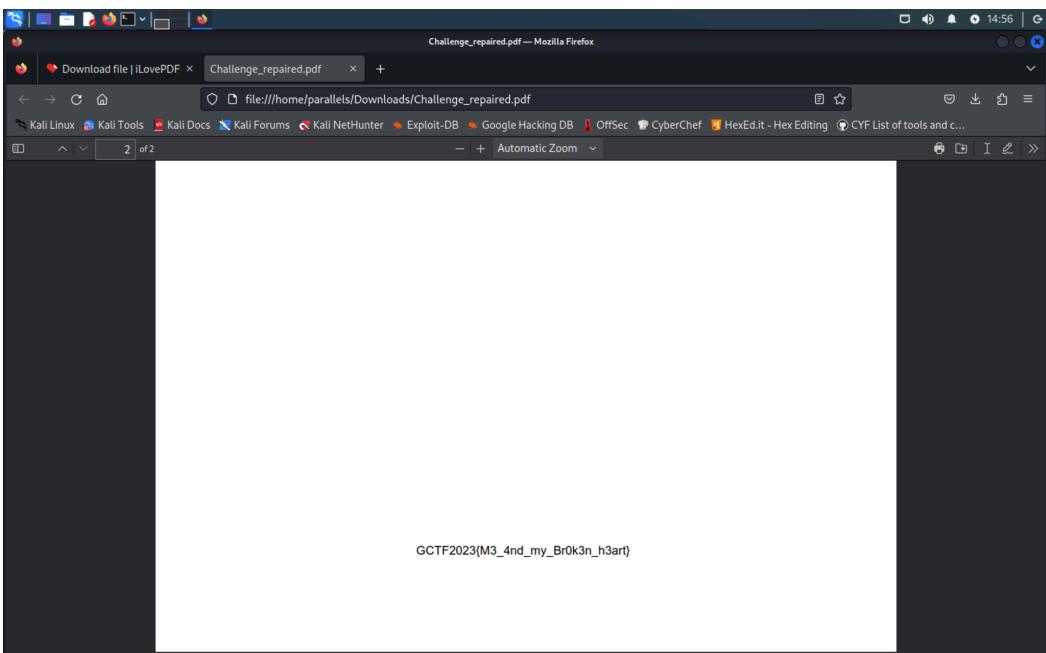
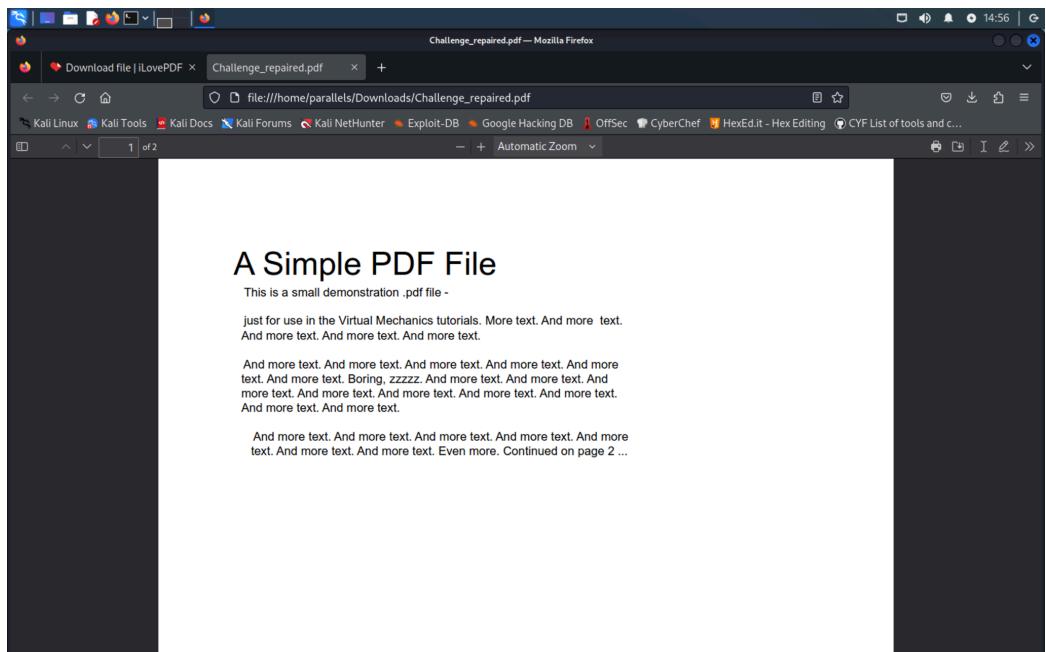
I clicked “select pdf file” to upload a broken pdf file.



I uploaded Challenge.pdf



After that, I downloaded the repaired file and found the pdf files as below.



Found the flag!

Flag: GCTF2023{M3_4nd_my_Br0k3n_h3art}

KB

CHALLENGE

X

**KB
400**

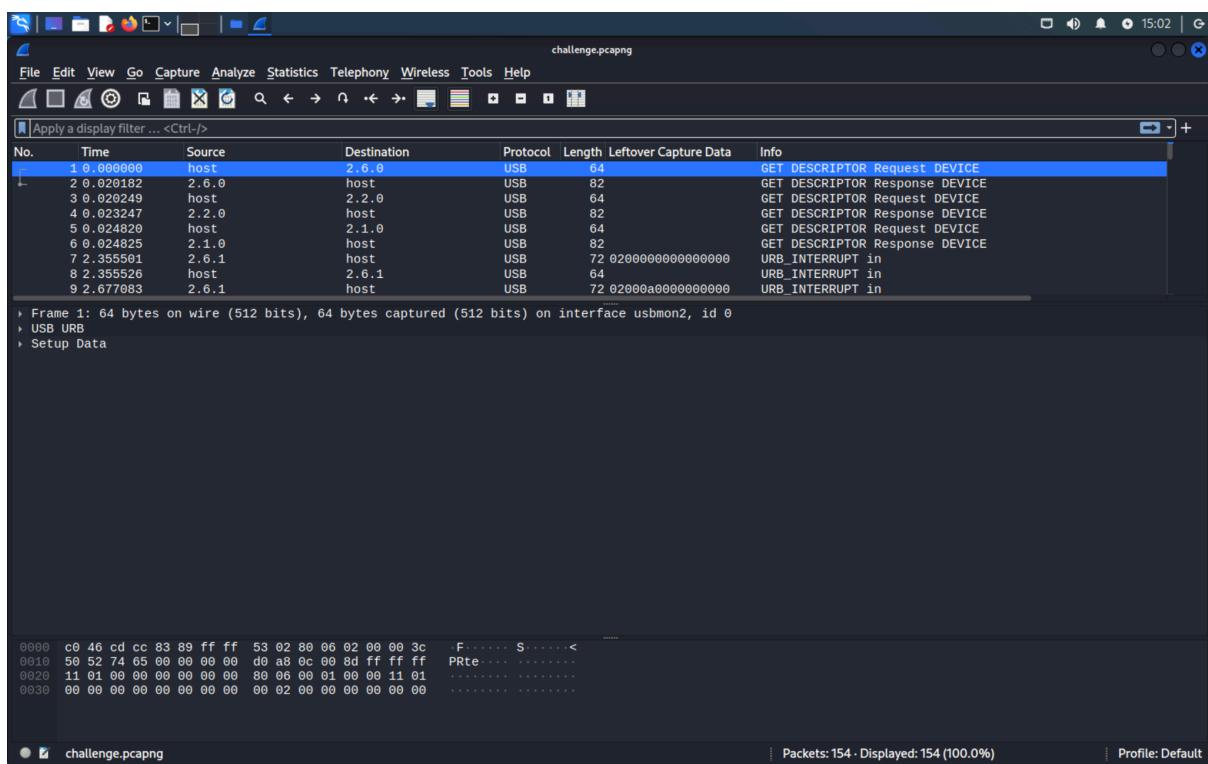
HARD

HMM, HAVEN'T SEEN THIS TYPE OF WIRESHARK FILE IN A
WHILE NOW.

CHALLENGE....

FLAG

SUBMIT



For this challenge, I opened the file using wireshark, and this is what it shows.

Google search results for "pcapng usb interrupt in":

- Medium · AllBawazeer** - [kaizen-ctf 2018 — Reverse Engineer usb keystrok from pcap file](#)
this CTF challenge contain pcapng file and no hint provided only flag needed to earn the points ... [usbnutshell/usb4.shtml#lInterrupt which came ...](#)
- stayontarget.org** - [Decoding Mixed Case USB Keystrokes from PCAP](#)
Mar 26, 2019 — During a recent assessment, I captured USB keystrokes as a part of a larger set of data from a system. ... `#print(ucasekey[int(bytesArray[2])]) #...`
- CTFTime.org / VishwaCTF 2023 / 1nj3ct0r / Writeup**
pcapng. image. It is a capture of **USB** protocol. Generally in a **usb** dump we find keyboard **interrupts** or mouse clicks as user input data, so I started ...
- GitHub** - [Techniques for analyzing USB protocols](#)
Most of the protocol we need to implement lies in these two Wireshark fields. In **Interrupt** or bulk transfers all data is this **cdapdata** , the rest is just **USB** ...
- GitHub** - [JohnDMcMaster / usbrply](#)
Library for USB Mass Storage Devices. Read and Write UCRP and UCRP+ files

Then, I just googled some keywords and I found this website (link below) and I referred to it for this challenge.

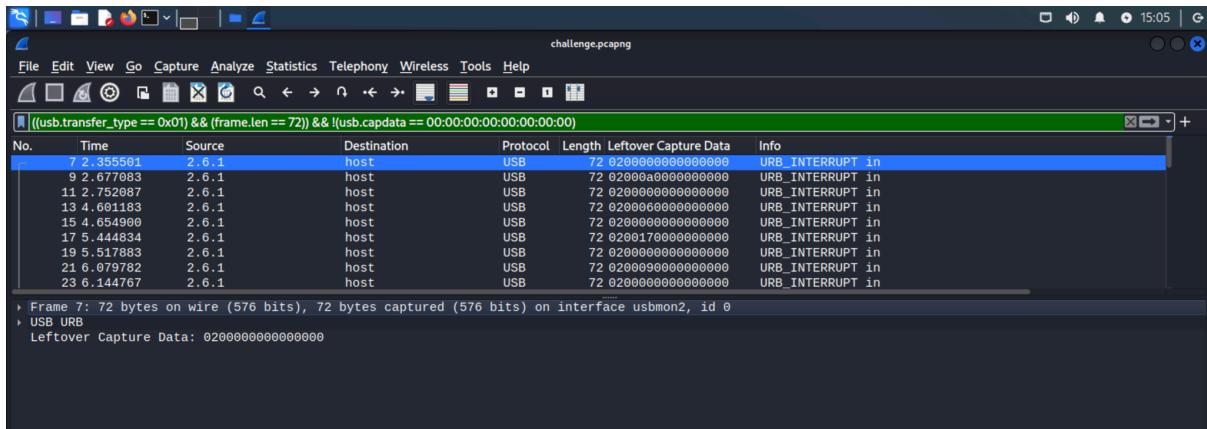
<https://abawazeeer.medium.com/kaizen-ctf-2018-reverse-engineer-usb-keystrok-from-pcap-file-2412351679f4>

No.	Time	Source	Destination	Protocol	Length	Leftover Capture Data	Info
1	0.000000	host	2.6.0	USB	64		GET_DESCRIPTOR Request DEVICE
2	0.020182	2.6.0	host	USB	82		GET_DESCRIPTOR Response DEVICE
3	0.020249	host	2.2.0	USB	64		GET_DESCRIPTOR Request DEVICE
4	0.023247	2.2.0	host	USB	82		GET_DESCRIPTOR Response DEVICE
5	0.024820	host	2.1.0	USB	64		GET_DESCRIPTOR Request DEVICE
6	0.024825	2.1.0	host	USB	82		GET_DESCRIPTOR Response DEVICE
7	2.355501	2.6.1	host	USB	72	0200000000000000	URB_INTERRUPT in
8	2.355526	host	2.6.1	USB	64		URB_INTERRUPT in
9	2.677083	2.6.1	host	USB	72	02000a0000000000	URB_INTERRUPT in
10	2.677102	host	2.6.1	USB	64		URB_INTERRUPT in
11	2.752087	2.6.1	host	USB	72	0200000000000000	URB_INTERRUPT in
12	2.752113	host	2.6.1	USB	64		URB_INTERRUPT in
13	4.601183	2.6.1	host	USB	72	0200060000000000	URB_INTERRUPT in
14	4.601206	host	2.6.1	USB	64		URB_INTERRUPT in
15	4.654900	2.6.1	host	USB	72	0200000000000000	URB_INTERRUPT in
16	4.654920	host	2.6.1	USB	64		URB_INTERRUPT in
17	5.444834	2.6.1	host	USB	72	0200170000000000	URB_INTERRUPT in
18	5.444860	host	2.6.1	USB	64		URB_INTERRUPT in
19	5.517883	2.6.1	host	USB	72	0200000000000000	URB_INTERRUPT in
20	5.517906	host	2.6.1	USB	64		URB_INTERRUPT in
21	6.070782	2.6.1	host	USB	72	0200000000000000	URB_INTERRUPT in

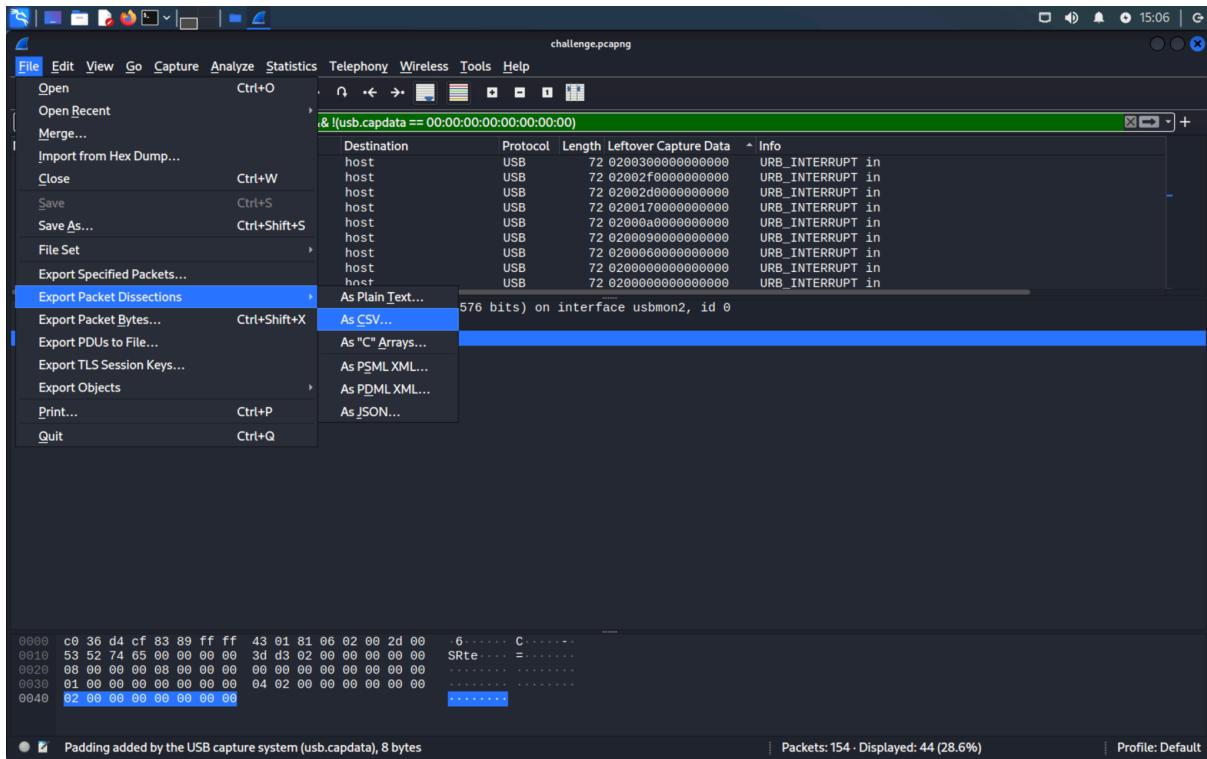
```

Frame 13: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface usbmon2, id 0
  USB URB
    [Source: 2.6.1]
    [Destination: host]
    URB id: 0xffff8983cf436c0
    URB type: URB_COMPLETE ('C')
    URB transfer type: URB_INTERRUPT (0x01)
    > Endpoint: 0x81, Direction: IN
    Device: 6
    ...
    0000 c0 36 d4 cf 83 b9 ff ff 43 01 81 06 02 00 2d 00 ..6.....C.....
    0010 55 52 74 65 00 00 00 00 ef 92 06 00 00 00 00 00 URt... .....
    0020 08 00 00 00 08 00 00 00 00 00 00 00 00 00 00 00 .....
    0030 01 00 00 00 00 00 00 00 04 02 00 00 00 00 00 00 .....
    0040 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
  
```

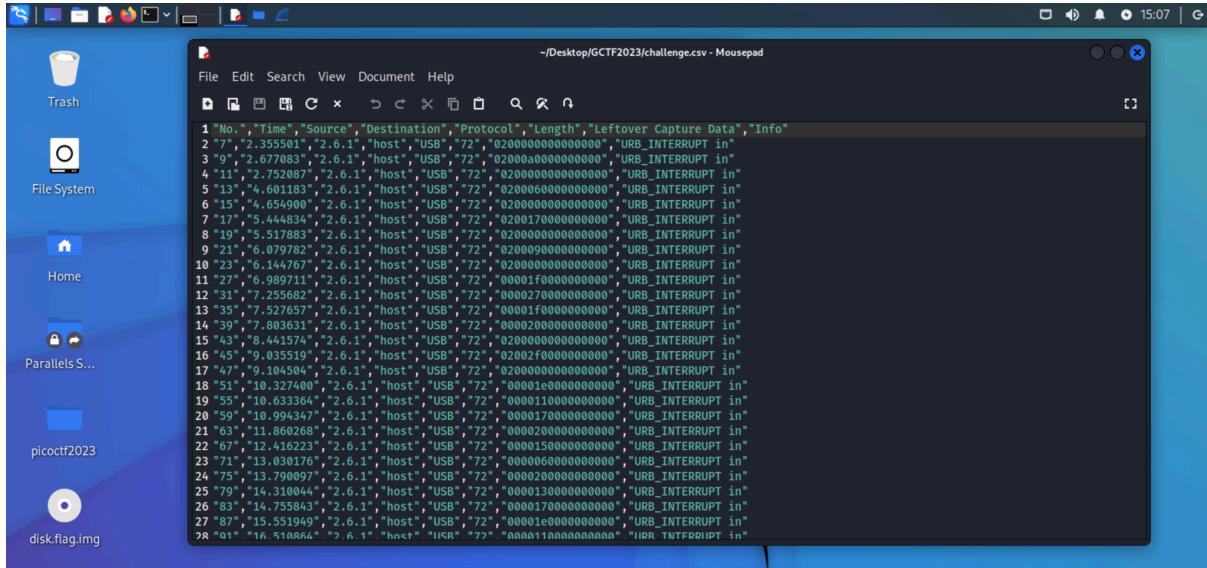
So, as we can see, there is some pattern in Leftover Capture Data, where a certain number changed on that position only, and it occurs on USB transfer type URB_INTERRUPT with frame length 72 only.



So, we filtered the data using this logic where usb transfer type is 0x01, and frame length 72:
`((usb.transfer_type == 0x01) && (frame.len == 72)) && !(usb.capdata == 00:00:00:00:00:00:00:00:00:00)`



Then, I exported the filtered data into a csv file and included the Leftover Capture Data into the column because that is the data that we need.



This is how the data looks like in the csv file.

```
(parallels@kali-linux-2022-2) [~/Desktop/GCTF2023]
$ cat challenge.csv | cut -d ',' -f 7 | cut -d ''"' -f 2 | grep -vE 'Leftover Capture Data' > hexoutput.txt

[parallels@kali-linux-2022-2) [~/Desktop/GCTF2023]
$ cat hexoutput.txt
0200000000000000
02000a0000000000
0200000000000000
0200060000000000
0200000000000000
0200170000000000
0200000000000000
0200090000000000
0200000000000000
00001f0000000000
0000270000000000
00001f0000000000
0000200000000000
0200000000000000
020002f0000000000
0200000000000000
00001e0000000000
0000110000000000
0000170000000000
0000200000000000
0000150000000000
0000060000000000
0000200000000000
0000130000000000
0000170000000000
00001e0000000000
0000110000000000
00000a0000000000
0200000000000000
020002d0000000000
0200000000000000
0000e00000000000
0000080000000000
00001c0000000000
0000150000000000
0000170000000000
0000150000000000
```

To save the Leftover Capture Data only, into an another txt file (hexoutput.txt) automatically, I use this command:

```
cat challenge.csv | cut -d ',' -f 7 | cut -d ""' -f 2 | grep -vE "Leftover Capture Data" > hexoutput.txt
```

Then, I read the `hexoutput.txt` file by using “`cat hexoutput.txt`” to make sure that everything is written there correctly.

The image shows two terminal windows side-by-side. The left window displays the Python script `keys.py`, which defines a dictionary `newmap` mapping hex values to characters. The right window displays the text file `hexoutput.txt`, which contains a list of hex values and their corresponding characters.

```
File Edit Search View Document Help ~/Desktop/GCTF2023/keys.py - Mousepad
1 newmap = {
2     0x02: "PostFail",
3     0x04: "a",
4     0x05: "b",
5     0x06: "c",
6     0x07: "d",
7     0x08: "e",
8     0x09: "f",
9     0x0A: "g",
10    0x0B: "h",
11    0x0C: "i",
12    0x0D: "j",
13    0x0E: "k",
14    0x0F: "l",
15    0x10: "m",
16    0x11: "n",
17    0x12: "o",
18    0x13: "p",
19    0x14: "q",
20    0x15: "r",
21    0x16: "s",
22    0x17: "t",
23    0x18: "u",
24    0x19: "v",
25    0x1A: "w",
26    0x1B: "x",
27    0x1C: "y",
28    0x1D: "z",
29    0x1E: "1",
30    0x1F: "2",
31    0x20: "3",
32    0x21: "4",
33    0x22: "5",
34    0x23: "6",
35    0x24: "7",
36    0x25: "8",
37    0x26: "9",
38    0x27: "0",
39    0x28: "Enter",
40    0x29: "esc",
41    0x2A: "del",
42    0x2B: "tab",
43
44
45
46
47
48
49
50
51
52
53
54
55 myKeys = open('hexoutput.txt')
56 i = 1
57
58 for line in myKeys:
59     bytesArray = bytearray.fromhex(line.strip())
60     # print("Line Number: " + str(i))
61     for byte in bytesArray:
62         if byte != 0:
63             keyVal = int(byte)
64             if keyVal in newmap:
65                 # print("Value map : " + str(keyVal) + " -> " + newmap[keyVal])
66                 print(newmap[keyVal])
67             else:
68                 print("No map found for this value: " + str(keyVal))
69             # print(format(byte, '02X'))
70             i += 1
71 }
```

```
File Edit Search View Document Help ~/Desktop/GCTF2023/keys.py - Mousepad
1 newmap = {
2     0x02: "PostFail",
3     0x04: "a",
4     0x05: "b",
5     0x06: "c",
6     0x07: "d",
7     0x08: "e",
8     0x09: "f",
9     0x0A: "g",
10    0x0B: "h",
11    0x0C: "i",
12    0x0D: "j",
13    0x0E: "k",
14    0x0F: "l",
15    0x10: "m",
16    0x11: "n",
17    0x12: "o",
18    0x13: "p",
19    0x14: "q",
20    0x15: "r",
21    0x16: "s",
22    0x17: "t",
23    0x18: "u",
24    0x19: "v",
25    0x1A: "w",
26    0x1B: "x",
27    0x1C: "y",
28    0x1D: "z",
29    0x1E: "1",
30    0x1F: "2",
31    0x20: "3",
32    0x21: "4",
33    0x22: "5",
34    0x23: "6",
35    0x24: "7",
36    0x25: "8",
37    0x26: "9",
38    0x27: "0",
39    0x28: "Enter",
40    0x29: "esc",
41    0x2A: "del",
42    0x2B: "tab",
43    0x2C: "space",
44    0x2D: " ",
45    0x2E: "[",
46    0x30: "]",
47    0x38: "/",
48    0x39: "CapsLock",
49    0x4B: "RightArrow",
50    0x4C: "LeftArrow"
51
52
53
54
55 myKeys = open('hexoutput.txt')
56 i = 1
57
58 for line in myKeys:
59     bytesArray = bytearray.fromhex(line.strip())
60     # print("Line Number: " + str(i))
61     for byte in bytesArray:
62         if byte != 0:
63             keyVal = int(byte)
64             if keyVal in newmap:
65                 # print("Value map : " + str(keyVal) + " -> " + newmap[keyVal])
66                 print(newmap[keyVal])
67             else:
68                 print("No map found for this value: " + str(keyVal))
69             # print(format(byte, '02X'))
70             i += 1
71 }
```

This is the python code that I used to convert the **hexoutput.txt** into a readable or ASCII character.

The screenshot shows a Kali Linux desktop environment. In the terminal window, the command `python keys.py` is run, displaying a long string of characters (PostFail, g, PostFail, c, PostFail, t, PostFail, f, PostFail, 1, n, t, 3, r, p, t, 1, n, g, k, PostFail, PostFail, -PostFail, k, e, y, s, t, r, 0, k, 3) which is the output of `hexoutput.txt`. The file manager shows files like `file.pcap`, `flag`, `flag.enc`, `flag-2.enc`, `FlagChecker.exe`, `hexoutput.txt`, `jail.py`, `keys.py`, `secret.zip`, `yes.txt`, and `yes.zip`. The task manager lists various system processes.

This is the output from **hexoutput.txt** after I run the python code.

After I type all the characters manually, I got the flag:

GCTF2023{1nt3c3pt1ng_keystr0k3}

WIRESHARK #2

CHALLENGE

X

WIRESHARK #2

387

MEDIUM

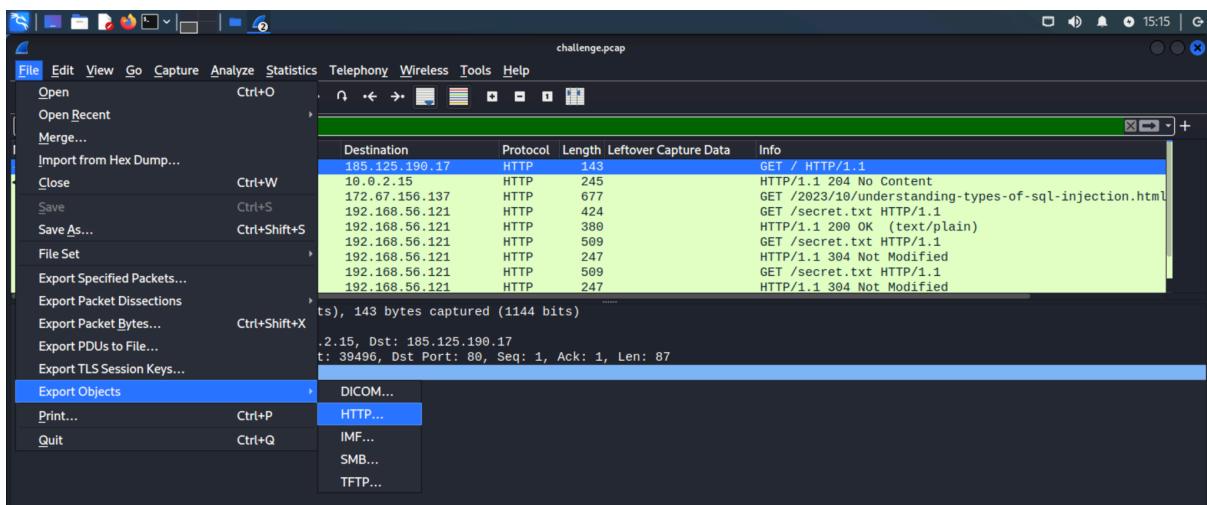
I REMEMBER THERE IS A COMPRESSED FILE. PLEASE FIND IT
AND EXPOSE THE SECRET!

SAME CHALLENGE FILE AS WIRESHARK #1

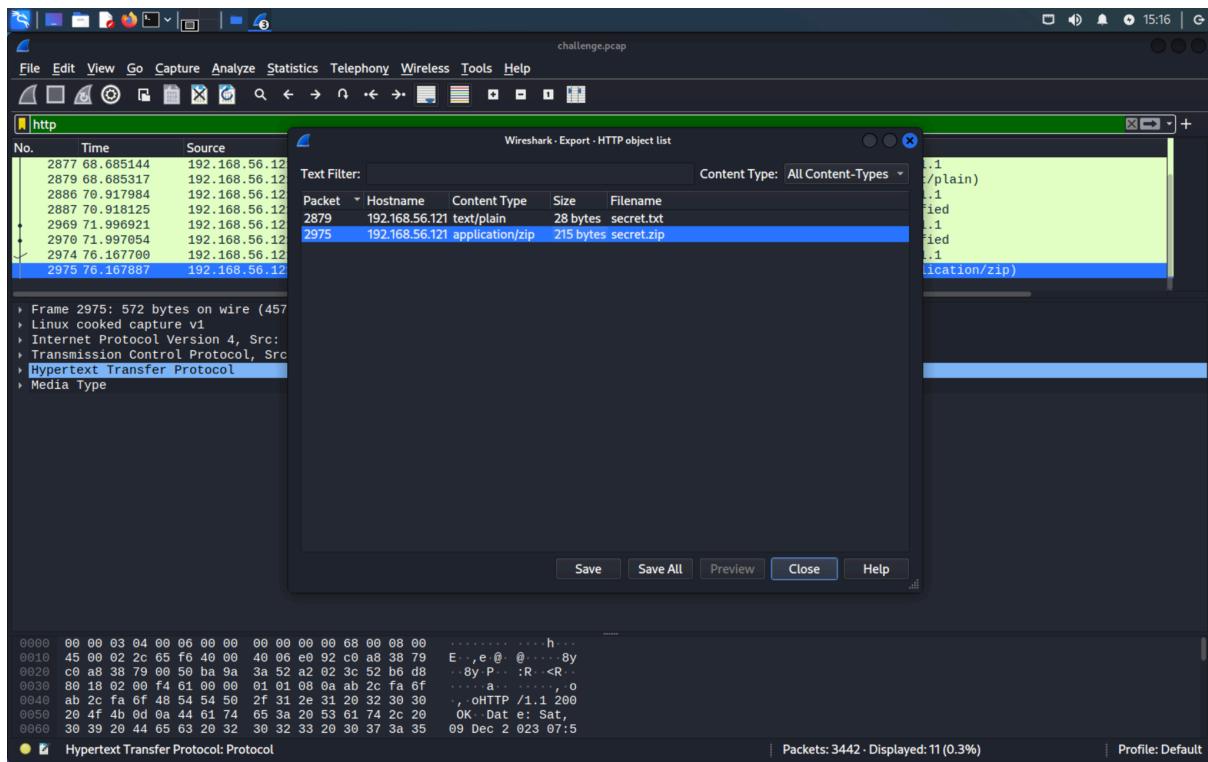
VIEW HINT

FLAG

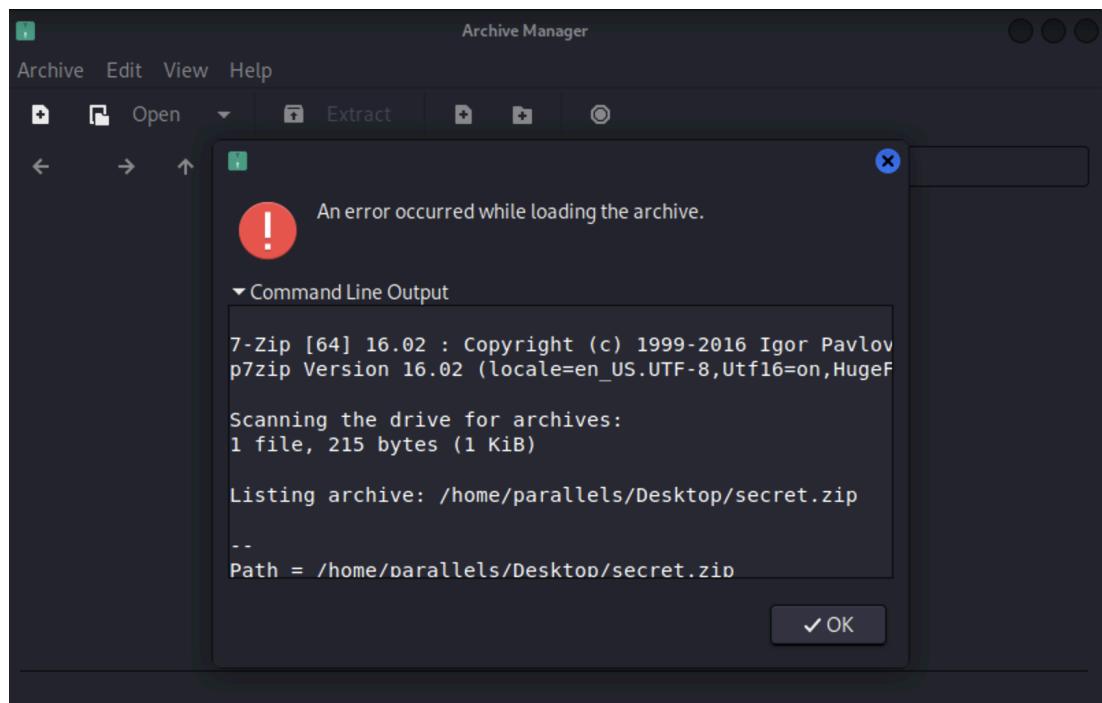
SUBMIT



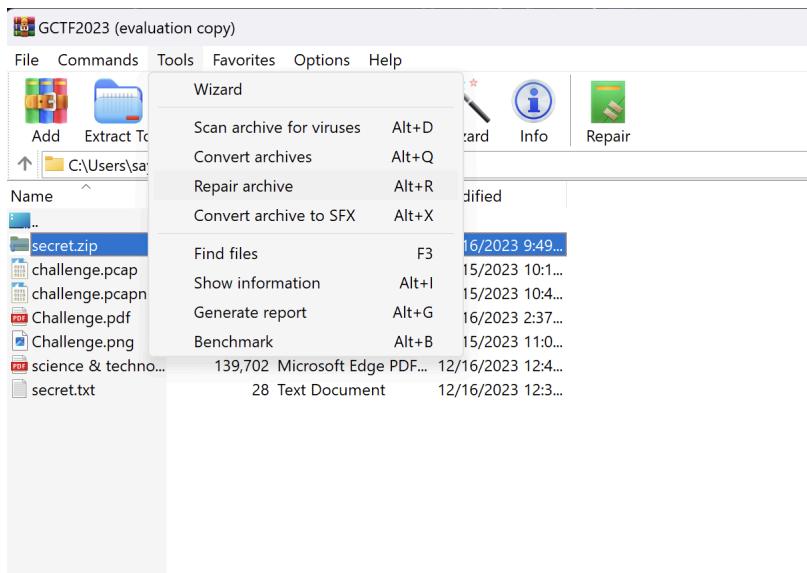
As this is a continuation from challenge [Wireshark #1](#), I am using the same pcap file, then I clicked export objects and clicked http to export the [secret.zip](#) file.



As shown above, I exported the file.



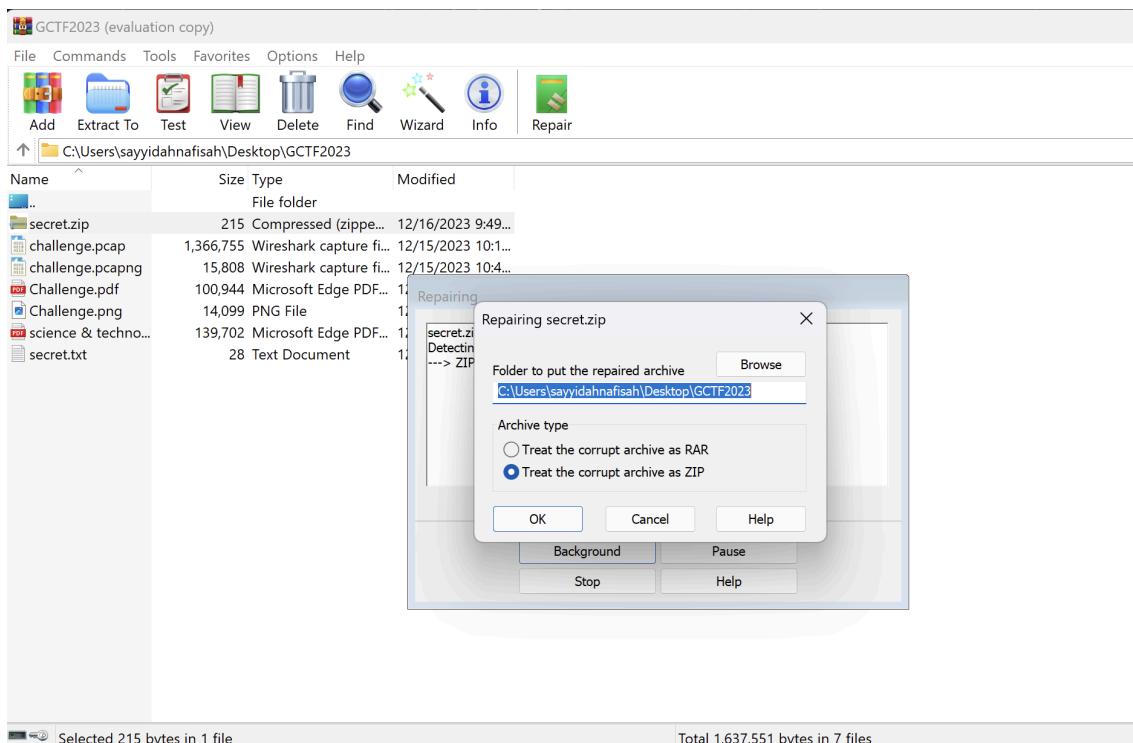
However, the zip file is corrupted and I couldn't open the file.



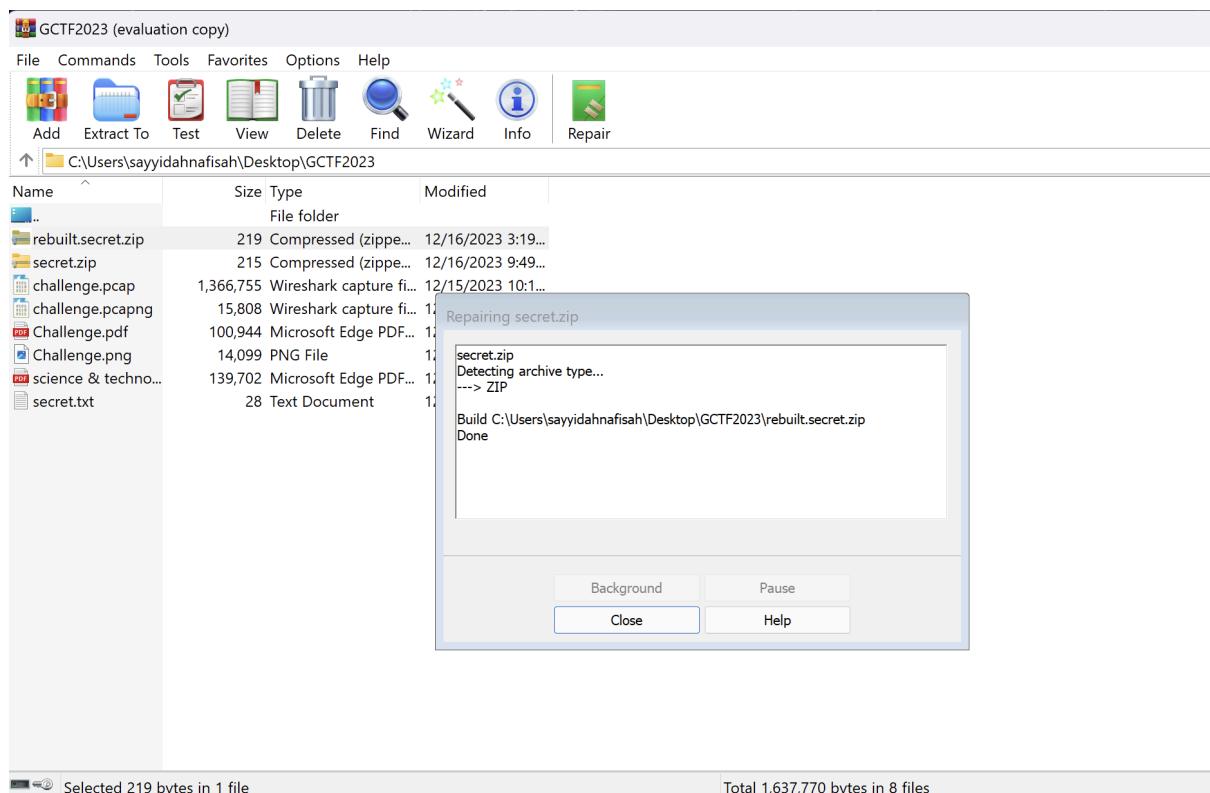
Then I used Win-rar apps to fix the zip file.

The steps to fix the file:

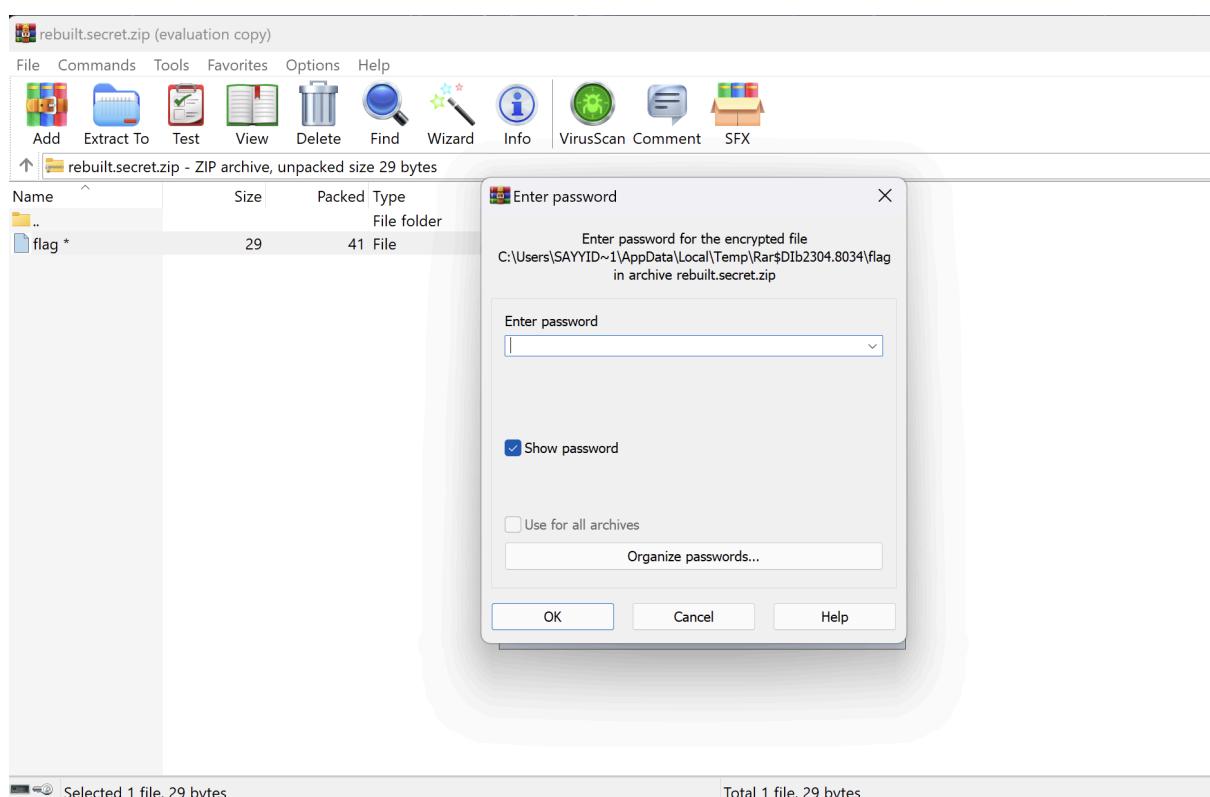
1. select **secret.zip** file
2. click Repair archive



3. Click OK



4. Click close



5. The file is successfully repaired but we are still not done yet because the file is protected by password.

The screenshot shows a terminal window titled "parallels@kali-linux-2022-2: ~/Desktop". The terminal history is as follows:

```
(parallels@kali-linux-2022-2) [~/Desktop]
$ zip2john rebuilt.secret.zip > secret.txt
ver 1.0 efh 5455 efh 7875 rebuilt.secret.zip/flag PKZIP Encr: 2b chk, TS_chk, cmplen=41, decmplen=29, crc=7E896D4A ts=01C5 cs=01c5
type=0

(parallels@kali-linux-2022-2) [~/Desktop]
$ john secret.txt
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
No password hashes left to crack (see FAQ)

(parallels@kali-linux-2022-2) [~/Desktop]
$ john secret.txt -show
rebuilt.secret.zip/flag:batman:flag:rebuilt.secret.zip::rebuilt.secret.zip

1 password hash cracked, 0 left

(parallels@kali-linux-2022-2) [~/Desktop]
$
```

To crack the password, I referred to this video:

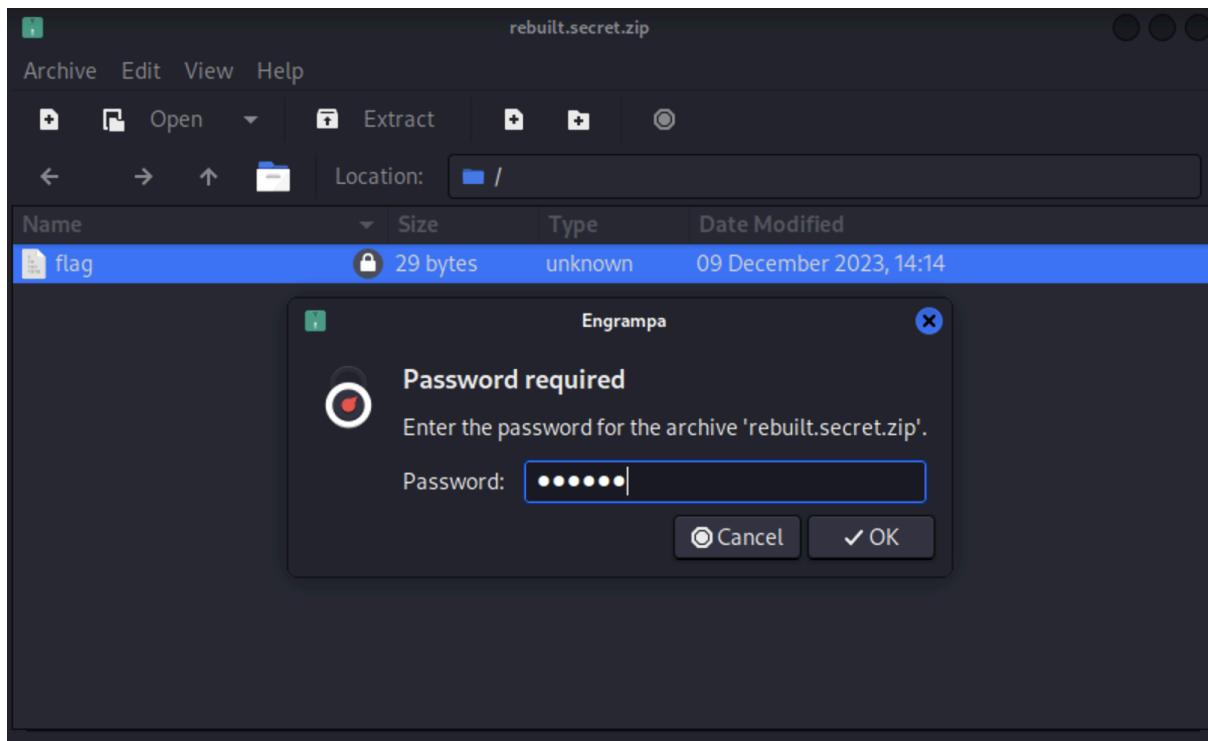
<https://www.youtube.com/watch?v=yyIoX0QT6QM>

I cracked it using **John the ripper**.

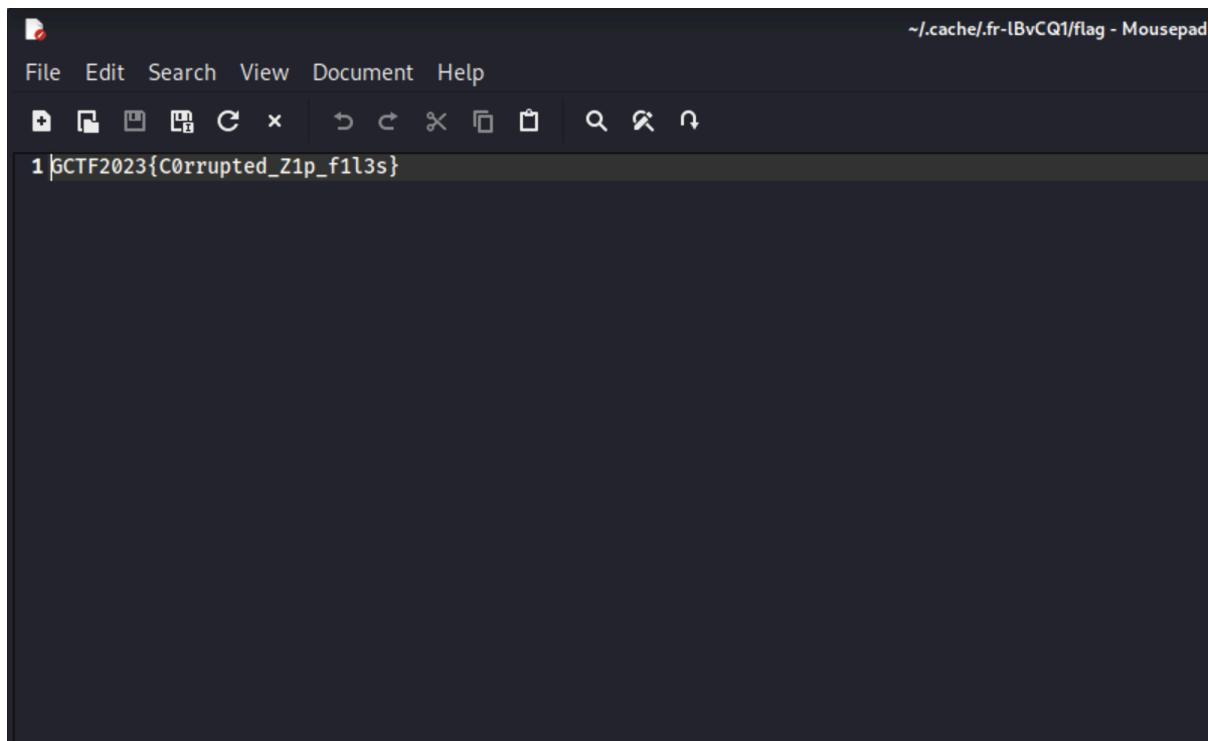
The linux command is as follows:

```
zip2john rebuild.secret.zip > secret.txt
john secret.txt
john secret.txt -show
```

Then, as shown in the picture, the password for the file is **batman**.



I entered the password “**batman**”



I successfully unlocked the file and found the flag!

Flag: GCTF2023{C0rrupted_Z1p_f1l3s}