

stegCTF writeup on Tryhackme

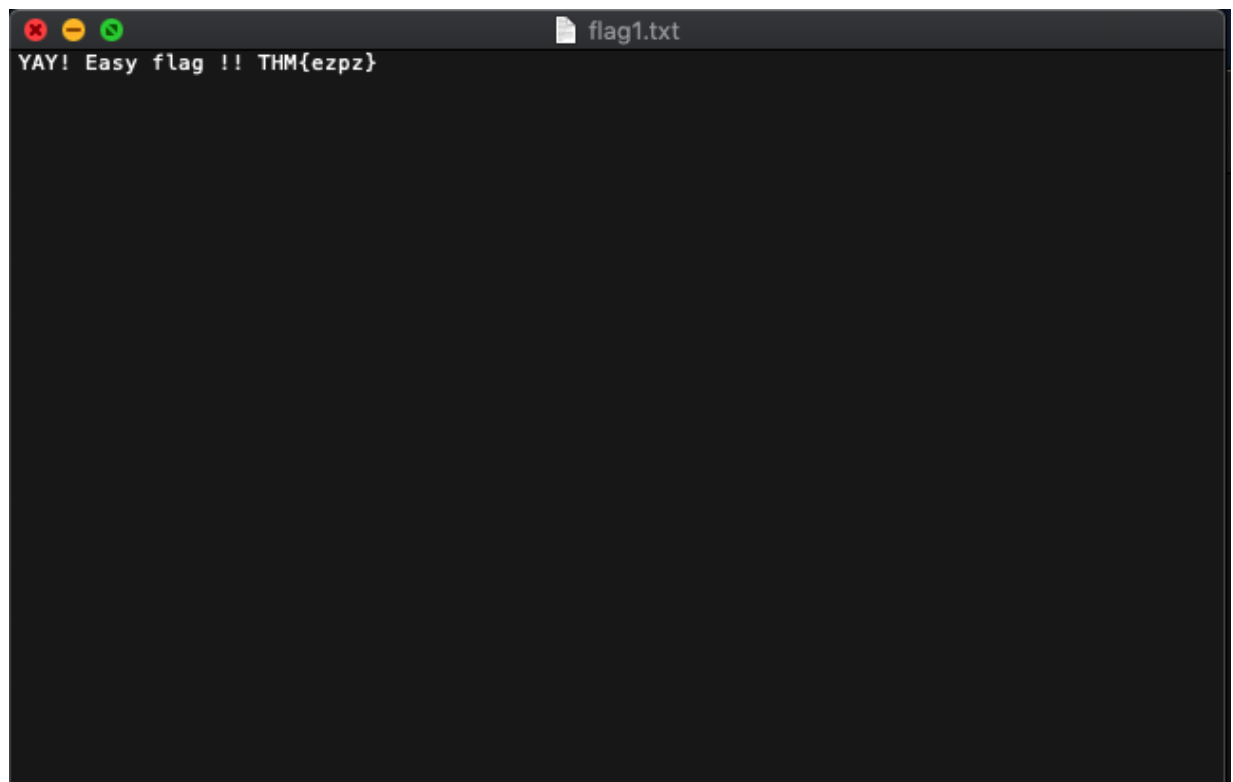
The following writeup is for the room stegCTF on tryhackme.

TASK 1: Download the zip file and unzip it.

```
> unzip 'stegtime!.zip'  
Archive:  stegtime!.zip  
creating: stegtime!/  
stegtime!/
```

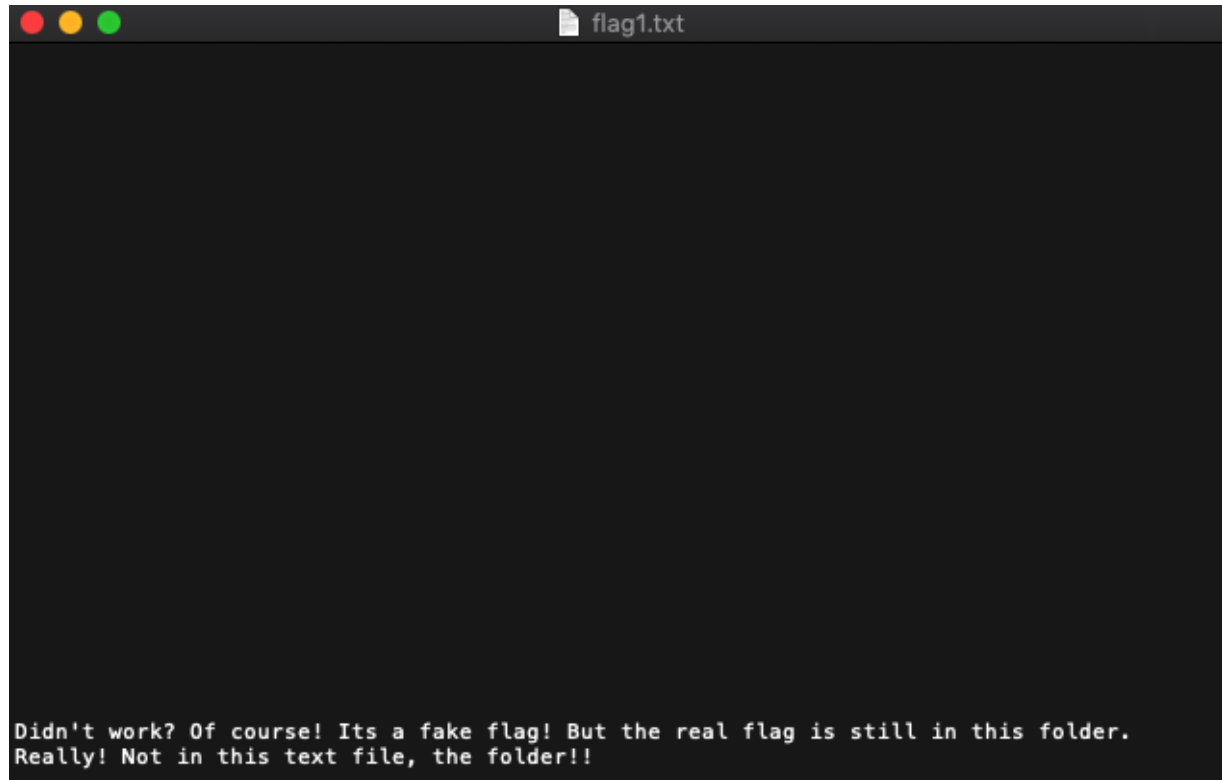
Task 2 : Finding the flags!

Flag1 :- Lets investigate the extracted file . It is imperative that to find flag1 we have to check flag1.txt. Open flag1.txt.

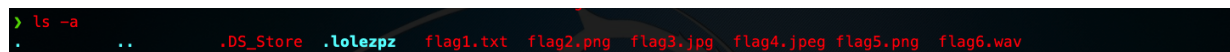


Well that was easy! Well...the flag does not work, no biggie! We scroll down on the text file and

find this.



Now it says the flag is in same folder. Lets check the hidden directories.



										Go To Offset										Find (Text search)															
Save	Copy	Cut	Paste	Undo	Redo																														
000	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	65	73	74	65	67	0A	69	6C	1lovesteg.1lovesteg.1lovesteg.1l		
020	6F	76	65	73	74	65	67	0B	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	0vesteg.1lovesteg.1lovesteg.1lo		
040	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	esteg.1lovesteg.1lovesteg.1loves		
060	74	65	67	0B	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	teg.1lovesteg.1lovesteg.1loveste		
080	67	0B	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0A	g.1lovesteg.1lovesteg.1lovesteg.		
0A0	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	0lovesteg.1lovesteg.1lovesteg.1l		
0C0	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	esteg.1lovesteg.1lovesteg.1loves		
0E0	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	teg.1lovesteg.1lovesteg.1loveste		
100	74	65	67	0B	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	65	73	74	65	g.1lovesteg.1lovesteg.1lovesteg.		
120	67	0B	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	65	73	74	65	67	0B	q.1lovesteg.1lovesteg.1lovesteg.		
140	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	ilovesteg.1lovesteg.1lovesteg.1l	
160	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	0vesteg.1lovesteg.1lovesteg.1lo		
180	65	73	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	65	73	74	65	67	0A	69	6C	6F	76	65	73	esteg.1lovesteg.1lovesteg.1loves		
1A0	74	65	67	0A	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	65	73	74	65	teg.1lovesteg.1lovesteg.1loveste		
1C0	67	0A	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	65	73	74	65	67	0B	69	6C	6F	76	65	73	74	65	67	0A	g.1lovesteg.1lovesteg.1lovesteg.		
1E0	6F	76	65	73	74	65	67	0A	69	6C	6F																								

Flag2: We are given a flag2.png. Lets check out the image's metadata.(since its an image of a meta knight from kirby's adventure!) We use [exiftool](#) for this purpose.

```
Exiftool Version Number      : 11.85
File Name                    : flag2.png
Directory                    : .
File Size                    : 864 kB
File Modification Date/Time   : 2020:06:28 04:02:33+05:30
File Access Date/Time        : 2020:06:28 09:37:19+05:30
File Inode Change Date/Time   : 2020:06:28 09:37:18+05:30
File Permissions              : rw-r--r--
File Type                    : PNG
File Type Extension          : png
MIME Type                    : image/png
Image Width                  : 1822
Image Height                 : 1078
Bit Depth                   : 8
Color Type                   : RGB with Alpha
Compression                  : Deflate/Inflate
Filter                      : Adaptive
Interlace                   : Noninterlaced
Gamma                       : 2.2
Pixels Per Unit X            : 13780
Pixels Per Unit Y            : 13780
Pixel Units                  : meters
XMP Toolkit                  : XMP Core 5.4.0
Orientation                  : Horizontal (normal)
Artist                      : %w|LE9b06cD'6DE0_?bPN
Image Size                   : 1822x1078
Megapixels                   : 2.0
```

We notice that there in the artist section we have some characters under the artist section. Rot47 on them to get the flag!

Flag3: We have the flag3.jpg given to us. On extracting the data with steghide, without password, we get a new zip file!

```
> steghide extract -sf flag3.jpg
Enter passphrase:
wrote extracted data to "flag3.zip".
>
```

On trying to open we find that the file is password protected. Lets run strings on the zip file to obtain data.

```
password is [REDACTED] txtUT
#2!%te
u4)V
```

Unzipping the zip file with the password, we see a text file filled with '+' and ','. This is an esolang called reversefuck. On interpreting we get something similar to binary. This is a yet another esolang called binaryfuck. Interpreting we get the flag. To interpret the esolangs use this interpreter for [reversefuck](#) and [binaryfuck](#).

Flag4: For flag4 we are given an image of a keyboard as flag4.jpeg. Steghide isn't able to recover any file from the image. Even brute forcing with [stegcracker](#) returns nothing. We then use another steganography tool for jpegs called [jsteg](#).

```
> go run main.go reveal '[REDACTED]' stegtime!/flag4.jpeg'
36 47 411 49 34 37 57 310 32 35 44 55 39 311 47 34 35 39 43 410 39 36
```

These numbers are actually [keyboard coordinate cipher](#). Decoding and adapting it according to the flag format, we have our flag!

Flag5: We are given a flag5.png which seems to be corrupted. A quick check in a hex editor shows us that the image has wrong header. On fixing the header, we are still unable to open the image. Lets check the image using [pngcheck](#).

```
> pngcheck flag5.png
flag5.png first chunk must be IHDR
ERROR: flag5.png
```

Lets change the ADHD chunk to IHDR. We are still unable to open the file. On checking with pngcheck again,

```
> pngcheck flag5.png
flag5.png illegal (unless recently approved) unknown, public chunk DRAT
ERROR: flag5.png
```

```
> stegolsb steglsb -r -i '/Users/[REDACTED]/stegtime!/flag5.png' -o output.txt
Files read          in 0.11s
310 bytes recovered  in 0.00s
Output file written  in 0.00s
```

A quick look at [libpng](#) suggests that the chunk DRAT is actually IDAT. On changing the chunk we are finally able to open the image. Steghide can't be

used here as its a png image file. So we search for alternatives. We find [stegolsb](#). Now we extract data from the image.

```
> stegolsb steglsb -r -i '/Users/[REDACTED]/stegtime!/flag5.png' -o output.txt
Files read          in 0.11s
310 bytes recovered  in 0.00s
Output file written  in 0.00s
```

Now we have a text file which contains what looks like chess moves.

On observing closely we see that the image has a hidden pastebin link. On following the link we are presented with a vigenere cipher and a base58 encoded text. Decoding the base58 text we get hint to the key size. Finding an online [auto-solver](#) for vigenere takes no time. Setting the key size accordingly we obtain the cracked cipher and a hint to what to do next.

Auto Solve results		
Score	Key	Text
38838	[REDACTED]	[REDACTED] might help i am adding this so that auto solvers can get help so thank me later anyways this challenge is very bad i know sorry i am running out of ideas to write bye bye

Following on to the site, we find a chess steganography decoder! On decoding without blunders, we get the flag!

See [my blog post](#) for more information. Also check out [Jonas Enge's CLI implementation](#).

Steg »	Steg without blunders »	Unsteg <	Unsteg without blunders <
<div>[REDACTED]</div>		<div>1. a4 Nc6 2. f3 b6 3. Nc3 Bb7 4. d3 e6 5. Nb1 Qf6 6. h4 Bb4+ 7. Kf2 g6 8. e3 Nge7 9. Be2 Kd8 10. Kf1 Ng8 11. Rh3 Ke8 12. Rh1 Nge7 13. Rh3 Na5 14. f4 h5 15. c4 Rh7 16. g3 e5 17. Ra2 Rg7 18. e4 Kd8 19. Qc2 Ba6 20. Rh1 Rc8 21. Qd1 Bd6 22. Nd2 Bc5 23. Nb1 Rg8 24. Nf3 d6 25. Nh2 Rh8 26. Bf3 { Black resigns. } 1-0</div>	

Flag6: For flag we have a wav file. Audio for a change! In the stegolsb tool we just downloaded we find a special subtool called wavsteg. But extracting data demands the number of bytes to be extracted. On hearing the audio file we have our answer! Extracting with given no. bytes we get a file.

```
> stegolsb wavsteg -r -i flag6.wav -o output3.txt -b [REDACTED]
Files read          in 0.01s
Recovered 1342 bytes in 0.00s
Written output file  in 0.00s
```

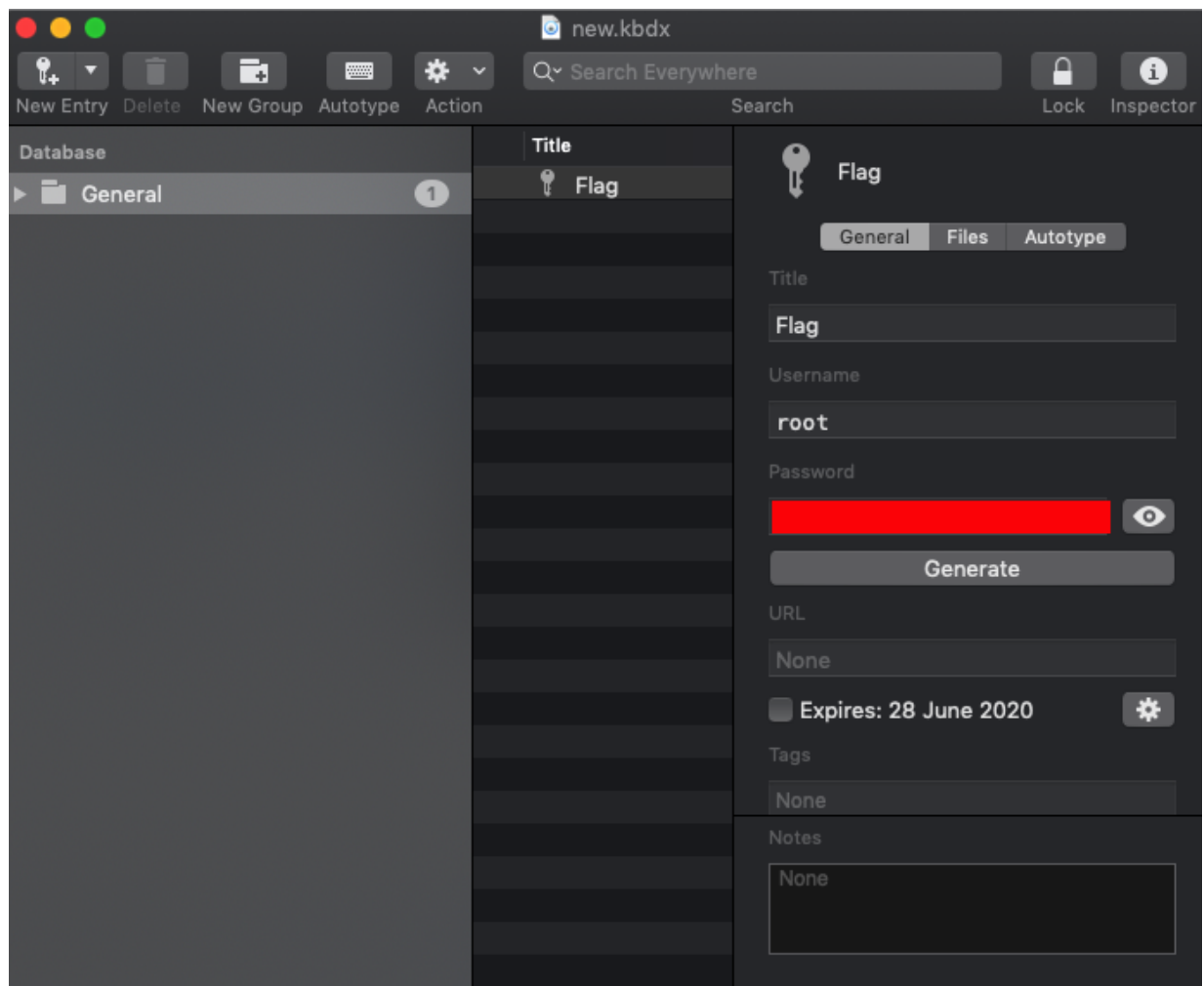

Now on checking the filetype with the file command, we see,

```
> file output.txt
output.txt: KeePass password database 2.x KDBX
```

On googling we find that its file extension is .kdbx. Now on further researching we find many [applications](#) to open a keepass file. On trying to open with any of the application we are prompted for a password. Now since we do not know the password, we have to bruteforce! Converting to its hash and bruteforcing with johntheripper we have the password!

```
> ./keepass2john
Usage: ./keepass2john [-k <keyfile>] <.kdbx database(s)>
> ./keepass2john '/Users/[redacted]/stegtime!/output.kdbx' > hash
> ./john hash --wordlist=rockyou.txt --format=KeePass
Using default input encoding: UTF-8
Loaded 1 password hash (KeePass [SHA256 AES 32/64 OpenSSL])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
0g 0:00:00:51 0.02% (ETA: 2020-07-01 02:11) 0g/s 82.08p/s 82.08c/s 82.08C/s 258963..ravens
[redacted]
(output)
1g 0:00:01:14 DONE (2020-06-28 11:15) 0.01340g/s 82.06p/s 82.06c/s 82.06C/s camden..[redacted]
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Use the password to open the file and find the flag!



And there we have it! We have solved all the 6 challenges!