Write Up: Cool Runnings

This CTF is based on cryptography. It is my first CTF, so is fairly basic.

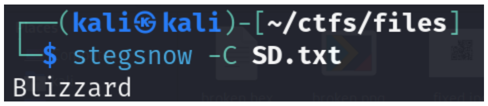
1 INTRODUCTION

The user is presented with two files – a password protected zip file and a text file. The text file contains the password required to unlock the zip folder, which contains 3 different files. Each file contains a part of the flag. This challenge should be carried out on Kali Linux.

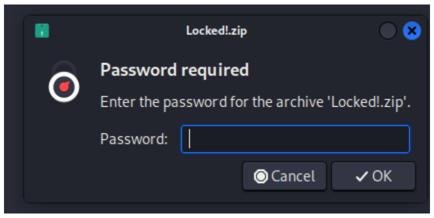
2 Solution

2.1 Text File

The text file uses whitespace to conceal the password for the zip folder. To extract the password, the user should use *stegsnow* on Kali Linux. The user will find that the password for the zip file is **"Blizzard".**



```
┌──(kali㉿kali)-[~/ctfs/files]
└─$ stegsnow -C SD.txt
Blizzard
```



Locked!.zip

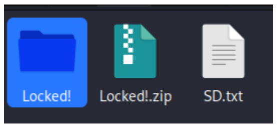
Password required

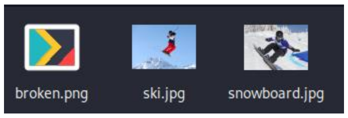
Enter the password for the archive 'Locked!.zip'.

Password:

2.2 Zip File

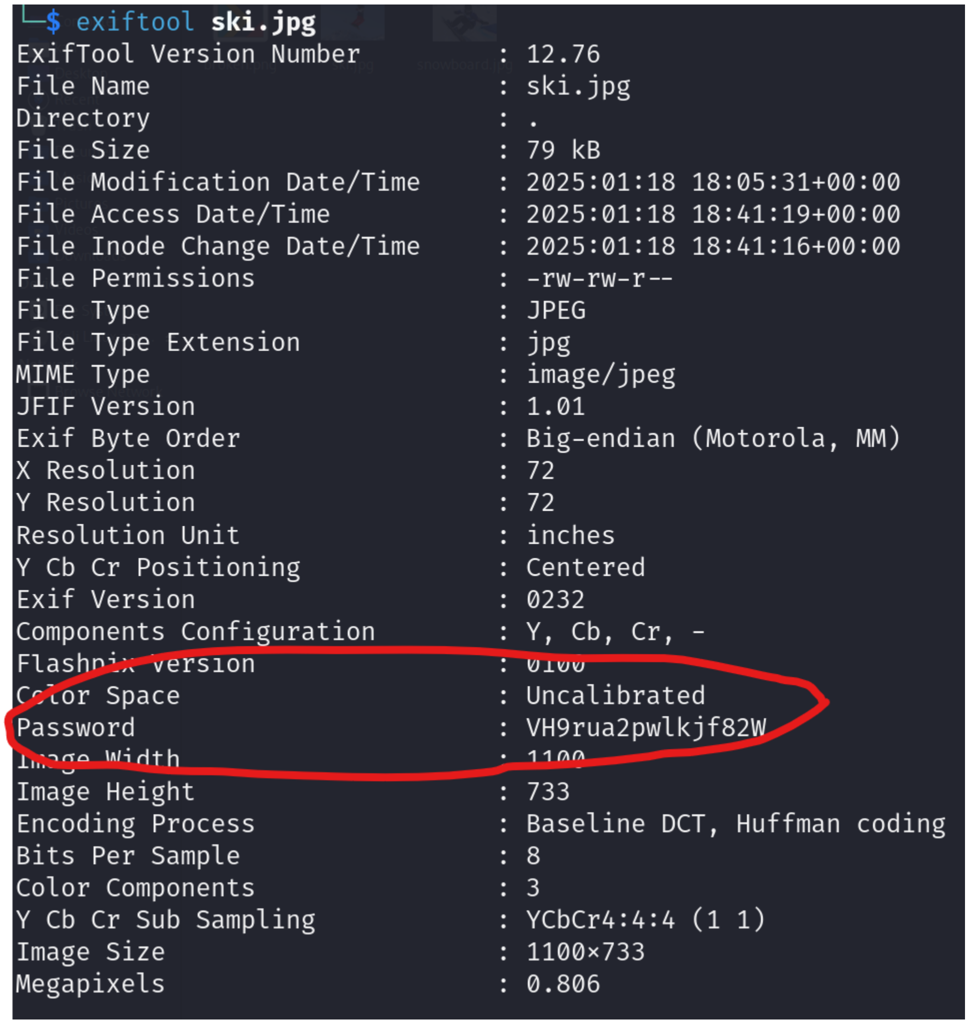
When the zip file is unlocked and extracted, the user will find three files.





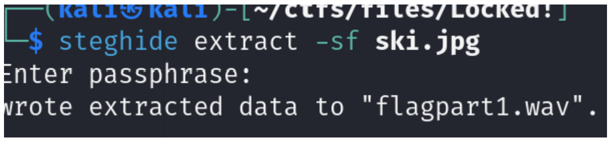
2.2.1 Ski.jpg

The Ski.jpg file contains the first part of the flag. This image hides a wav file, which needs to be extracted using *steghide*. *Steghide* requires a password to use, and this password can be found in the EXIF data of the image. I created a custom "password" tag for this, however this does not show up on any online EXIF data viewers. To find the password, the user must use *exiftool* on Kali Linux. Once *exiftool* is used against this image, the password will be: **VH9rua2pwlkjf82W**



```
└─$ exiftool ski.jpg
ExifTool Version Number         : 12.76
File Name                       : ski.jpg
Directory                       : .
File Size                       : 79 kB
File Modification Date/Time     : 2025:01:18 18:05:31+00:00
File Access Date/Time           : 2025:01:18 18:41:19+00:00
File Inode Change Date/Time     : 2025:01:18 18:41:16+00:00
File Permissions                : -rw-rw-r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Exif Byte Order                 : Big-endian (Motorola, MM)
X Resolution                    : 72
Y Resolution                    : 72
Resolution Unit                 : inches
Y Cb Cr Positioning             : Centered
Exif Version                    : 0232
Components Configuration        : Y, Cb, Cr, -
Flashpix Version                : 0100
Color Space                     : Uncalibrated
Password                        : VH9rua2pwlkjf82W
Image Width                     : 1100
Image Height                    : 733
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:4:4 (1 1)
Image Size                      : 1100x733
Megapixels                      : 0.806
```

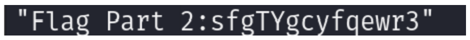
When this password is used with *steghide*, the audio file “flagpart1.wav” will be extracted. This file contains morse code which must be translated. The translation, and first part of the flag, is “QPLDOHUDGSI”.



```
┌──(kali㉿kali)-[~/ctfs/files/Locked!]
└─$ steghide extract -sf ski.jpg
Enter passphrase:
wrote extracted data to "flagpart1.wav".
```

2.2.2 Snowboard.jpg

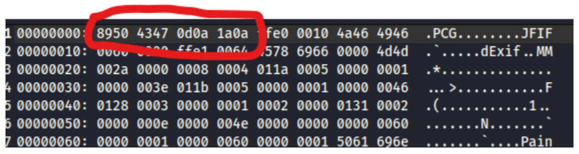
This part of the challenge requires this image to be opened in a text editor, or the *strings* utility. The flag is simply displayed as “**Flag part 2: sfgTYgcyqwer3**”



```
"Flag Part 2:sfgTYgcyfqewr3"
```

2.2.3 Broken.png

When the user attempts to open this image, it will be unsuccessful. To open the image, the file header must be changed. The current header is 8950 4347 0d0a 1a0a, which is the correct header for a png image. However, this file is actually a jpg image disguised as a png. To fix this, the file header must be changed to “ffd8” and the extension must be changed to jpg. After this is done, the image will be displayed.

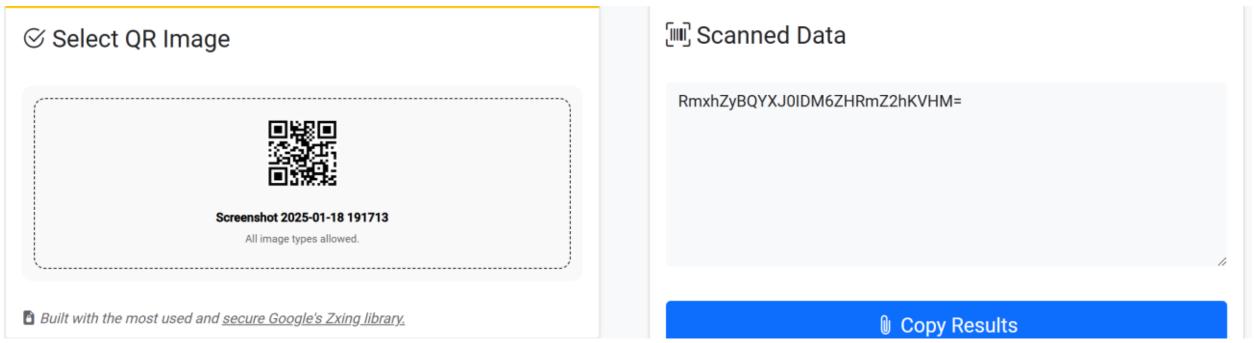


```
00000000: 8950 4347 0d0a 1a0a ffe0 0010 4a46 4946  .PCG........JFIF
00000010: 0060 [illegible] ffe1 0064 4578 6966 0000 4d4d  .`.....dExif..MM
00000020: 002a 0000 0008 0004 011a 0005 0000 0001  .*..............
00000030: 0000 003e 011b 0005 0000 0001 0000 0046  ...>...........F
00000040: 0128 0003 0000 0001 0002 0000 0131 0002  .(...........1..
00000050: 0000 000e 0000 004e 0000 0000 0000 0060  .......N.......`
00000060: 0000 0001 0000 0060 0000 0001 5061 696e  .......`....Pain
```

The image is a QR code. In its current state, the QR code will not scan – it is broken. One of the position markers (big squares in the corner) is missing.

To fix this, this must be opened in an image editor (the free image editor I used was called “*paint.net”*). One of the other two position markers should be copied and moved into place (although I suppose you could also use the shape tool to create the image). In *paint.net*, I highlighted one of the position markers, copied the selection and moved it into place. The QR code will then scan properly. You can either use your phone to scan this or an online QR code scanner (not all online scanners are good).

Once scanned, you’ll find the resulting text to be “RmxhZyBQYXJ0IDM6ZHRmZ2hKVHM=”.


Select QR Image
Screenshot 2025-01-18 191713
All image types allowed.
Built with the most used and secure Google's Zxing library.
Scanned Data
Copy Results

```
RmxhZyBQYXJ0IDM6ZHRmZ2hKVHM=
```

This is encoded in base64 (very easy I know but I felt I should at least add some form of encoding). When the base64 is decoded, it will result with “Flag Part 3:**dtfghJTs**”.

3 FLAG

After obtaining all parts to the flag, they will be put together and result in a flag of "**QPLDOHUDGSIsfgTYgcyfqewr3dfJiugf**".

Flag: **SECURI-TAY{QPLDOHUDGSIsfgTYgcyfqewr3dfJiugf}**