



Mission Name

The truth

Historical Background

In their relentless pursuit of the truth, Ethan and Claire receive a critical piece of evidence from Dr. Pinche. The missing footage from Jailnor's holding room at Skytech could hold the key to understanding his fate. However, the images are in disarray, scrambled beyond immediate recognition. Dr. Pinche, unable to decipher the data, entrusts Ethan with this crucial task.

Technical High-Level Overview

Ethan is faced with a complex challenge: to analyze and reorder the scrambled footage provided by Dr. Pinche. This task requires a keen eye for detail and a deep understanding of image analysis techniques. The secret hidden within these images is vital for unraveling the mystery surrounding Jailnor's circumstances and could potentially expose hidden truths about Skytech.

Short Mission Description

Ethan, the information from Dr. Pinche is now in your hands. Your mission is to sift through the scrambled footage, reassemble the images correctly, and extract the concealed truth about what happened to Jailnor. Your skills in data analysis and decryption are crucial to success. Good luck on your quest for the truth.

Mission Description

Dr. Pinche has managed to obtain the elusive footage from Jailnor's holding room within the confines of Skytech, a piece of the puzzle that has remained missing until now. Unfortunately, the footage is not in a viewable state; the images are jumbled and need to be meticulously reorganized. Unable to solve this puzzle himself, Dr. Pinche turns to Ethan, hoping his expertise can unlock the secrets held within. It's a race against time to piece together the footage and shed light on the events that transpired in Jailnor's holding room.

Location

RECON CAR – AIR



Questions

Algorithm use to hide text in part1?

- XTEA

Encoding using in text hidden part3?

- 7 bit ASCII

Stego technic used in part4?

- LPS

Hints

1. Try to find in google stego tools using the clues hidden in each part
2. Hexdump is your best friend
3. Sorry it's the last challenge

Categories

- Steganography
- Forensic
- Encoding
- Puzzle

Write Up

This challenge is a Puzzle composed of 4 parts of the flag, each one gives a part of the flag that is also a password to open the 7zip file protected next part

Part 1: Matroschka Steganography

Files: foo.png, m.png, Password.txt.

Approach: The password is encoded in Malbolge, resembling Base85. Use the Matroschka tool with the decoded password and foo (from the image name) as HMAC to extract the first part of the flag: flag{Puzzle}.



Figure 1



Figure 2

password.txt - Notepad

File Edit Format View Help

```
'&%$#"~6;:92165.Rsrqponmlk#(~}$$#"yx>_^\]\[ZYXWVUTSRQPONMibgf_dcba`Y}@?>=<;:9876543210/KJIHGFE'CBA:^>~6;43Wxwvutsrqponmlkjihgfedcba`_^\]\[ZYXWVUTSRQPONMLKJIHGFE'DCBA@?>=<;:9876543210/.-,+*)('&%$#"!~}\{zyxwvutsrqponmlkjihgfedcba`_^\]\[ZYXWVUTSRQPONMLKJIHGFE'DCBA@?>=<;:98765QPOONMLKJIHGFE>bB;:^!=<;:3Wxwvutsrqponmlkjihgfedcba`_^\]\[ZYXWVUTSRQPONMLKJIHGFE'DCBA@?>=<;:RQPUTSRQPINGk.-,+*)('&%$#"!~}\{zyxwvutsrqponmlkjihgfedcba`_^\]\[ZYXWVUTSRQPONMLKJIHGFE'DY^]\[ZYXWPo6543210/.-,+*)('&%A#?87<;:9870Tutsrqponmlkjihgfedcba`_^\]\[ZYXWVUTSRQPONMLKJIHGFE'DCBA@UZYXWPUTSRQJnNMLKDCgG@E>b%$#?87<;:981U543,10)M-,+*)i!Ef$#"!~}\v<]\rwputsrqpi/Pfkjiba`_^\$bD`YX]Vz=<;:9876543210/.-,+*)('&B$@?>=<5:921Uvutsrqponmlkjihgfedcba`_^\]\[ZYXWVUTSRQPONMLKJIHGFE'DCBA]?UZSXWVUTSRKoINMFj-,+*)(D=<A@?>=<;4Xyxwvutsrqponmlkjihgfedcba`_^\]\[ZYXWVUTSRQPONMLKJIHGFE'DCBA]\UZYXWVOTSLp3210/.-,+*)('&%$#"87<;:987654-Q+0)('K1kjihgfedcba`_^\]\[ZYXWVUqponPfkjihfa`&GFEDCBA@?>=<;:9876543210/.-,+*)('&%$#"!~}\{zyxwvutsrqponmlkjihgfedcba`_^\]\rwputsrqpingf,jiKgf_%FEDCBA@?>=<;:9876543210/.-,+*)('&%$#"!~}\{zyxwvutsrqponmlkjihgfedcba`_^\]\[ZYXWVUTSRQPOnMLKJIHGFE'DCBA]\>TYXWVUNSRKonHGFKDhHGFE'DCBA;:87<54Xyxwvutsrqponmlkjihgfedcba`_^\]\[ZYXWVUTSRQPONMLKJIHGFE'DCBA@?>=<;:9876ROPONMLKJc+*@d0
```

Figure 3

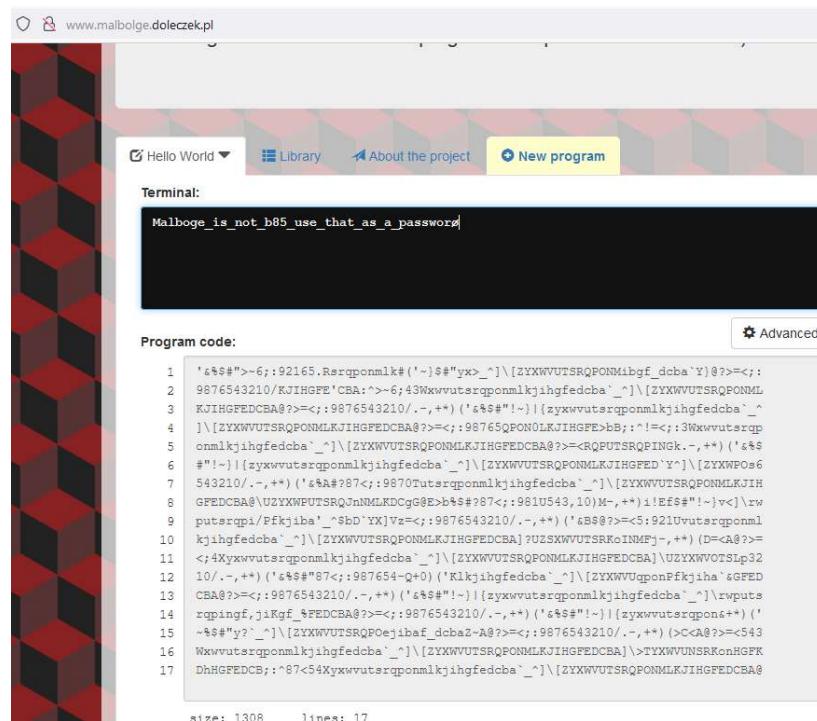


Figure 4

```
# python matroschka.py -open -m foo -k "Malboge_is_not_b85_use_that_as_a_password" .../foo.PNG
The hidden message is:
flag{Puzzle

HMAC hex is:
4dd46b9ce0c7a917c5213b76d75caf79225f7bb5a53afc58050b0e5b705ebf51

HMAC validation:
True
```

Figure 5

Part 2: Hexadecimal Analysis

Files: dhsdshdhk.PNG, Password.txt.

Approach: Analyze Password.txt with xxd and hexdump to identify and extract bytes different from 0000. Filtering out 0000 and interpreting dead and beef as binary (0 and 1), convert the binary sequence to a string, revealing the password. Use this password to decrypt dhsdshdhk.PNG with an appropriate steganography tool, guided by the name's hint.

```
L cat password.txt
```

Figure 6

```
000c4dc0: 0000 0000 0000 0000 0000 0000 0000 0000 .....  
000c4dd0: 0000 0000 0000 0000 0000 0000 0000 0000 .....  
000c4de0: 0000 0000 0000 0000 0000 0000 0000 0000 .....  
000c4df0: 0000 0000 0000 0000 0000 0000 0000 0000 .....  
000c4e00: 0000 0000 0000 0000 0000 0000 0000 0000 .....  
000c4e10: 0000 0000 0000 0000 0000 0000 0000 0000 .....  
000c4e20: 0000 0000 0000 0000 0000 0000 0000 0000 .....  
000c4e30: 0000 0000 0000 0000 0000 0000 0000 0000 .....  
000c4e40: 0000 0000 0000 0000 0000 0000 0000 0000 .....  
000c4e50: 0000 0000 0000 0000 0000 0000 0000 0000 .....  
000c4e60: 0000 0000 0000 0000 0000 0000 0000 0000 .....  
000c4e70: 0000 0000 0000 0000 0000 0000 0000 0000 .....  
000c4e80: 0000 0000 0000 0000 0000 0000 0000 0000 .....  
000c4e90: 0000 0000 0000 0000 0000 0000 0000 0000 .....  
000c4ea0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
```

Figure 7

```
*  
00ae9a0 adde 0000 0000 0000 0000 0000 0000 0000 0000  
00ae9b0 0000 0000 0000 0000 0000 0000 0000 0000 0000  
*  
00b2100 0000 0000 0000 efbe 0000 0000 0000 0000 0000  
00b2110 0000 0000 0000 0000 0000 0000 0000 0000 0000  
*  
00b5870 efbe 0000 0000 0000 0000 0000 0000 0000 0000  
00b5880 0000 0000 0000 0000 0000 0000 0000 0000 0000  
*  
00b7320 0000 0000 efbe 0000 0000 0000 0000 0000 0000  
00b7330 0000 0000 0000 0000 0000 0000 0000 0000 0000  
*  
00b96d0 adde 0000 0000 0000 0000 0000 0000 0000 0000  
00b96e0 0000 0000 0000 0000 0000 0000 0000 0000 0000  
*  
00bc3e0 adde 0000 0000 0000 0000 0000 0000 0000 0000  
00bc3f0 0000 0000 0000 0000 0000 0000 0000 0000 0000  
*  
00beda0 0000 efbe 0000 0000 0000 0000 0000 0000 0000  
00bedb0 0000 0000 0000 0000 0000 0000 0000 0000 0000  
*  
00c4eb0
```

Figure 8

```
└─# xxd password.txt | grep -v "0000 0000 0000 0000 0000 0000 0000 0000"
000087a0: 0000 dead 0000 0000 0000 0000 0000 0000
0000b410: 0000 0000 0000 beef 0000 0000 0000 0000
0000d040: 0000 0000 0000 beef 0000 0000 0000 0000
0000f2d0: 0000 0000 0000 0000 0000 0000 0000 beef
00012ff0: 0000 0000 dead 0000 0000 0000 0000 0000
00016080: 0000 0000 dead 0000 0000 0000 0000 0000
00017c90: 0000 0000 0000 0000 0000 0000 0000 beef
000196c0: 0000 0000 0000 beef 0000 0000 0000 0000
0001b3c0: 0000 0000 0000 0000 dead 0000 0000 0000
0001d560: 0000 0000 0000 0000 beef 0000 0000 0000
00020540: 0000 0000 0000 0000 0000 0000 beef 0000
00022050: 0000 0000 0000 0000 0000 0000 0000 dead
000243f0: 0000 0000 0000 0000 0000 0000 0000 beef
00026290: dead 0000 0000 0000 0000 0000 0000 0000
00027e20: 0000 0000 0000 0000 dead 0000 0000 0000
0002b630: dead 0000 0000 0000 0000 0000 0000 0000
0002e220: 0000 0000 0000 0000 0000 0000 0000 dead
0002fa20: 0000 0000 0000 dead 0000 0000 0000 0000
00032e50: 0000 beef 0000 0000 0000 0000 0000 0000
00036400: 0000 0000 0000 0000 0000 0000 beef 0000
00039c90: 0000 0000 0000 0000 dead 0000 0000 0000
0003b7a0: 0000 0000 0000 0000 beef 0000 0000 0000
0003e470: dead 0000 0000 0000 0000 0000 0000 0000
000421f0: 0000 0000 0000 dead 0000 0000 0000 0000
000444e90: 0000 0000 0000 dead 0000 0000 0000 0000
```

Figure 9

By filtering out everything except for the values `dead` and `beef`, and then substituting `dead` with `0` and `beef` with `1`, we are left with the binary sequence `011100110110100000110100011110000101111100110010001100000011010000111001`. Converting this binary sequence into ASCII text gives us the desired string.

Binary To String Converter

Enter the binary text to decode, and then click "Convert":

```
0111001101101000001101000111100001011111001100100011000000110100
00111001
```

Convert!

The decoded string:

```
sh4x_2049
```

Figure 10

dhsdshdhk stego

X | 

 Todo  Imágenes  Shopping  Videos  Maps  Más Herramientas

Aproximadamente 138 resultados (0,45 segundos)

<https://github.com/dhsdshdhk/stegpy> ▾ Traducir esta página

dhsdshdhk/stegpy: Simple steganography program ... - GitHub

Simple **steganography** program based on the LSB method. - **dhsdshdhk/stegpy** ... for encoding information in image and audio files through **steganography**.

Visítate esta página el 07/07/21

Figure 11

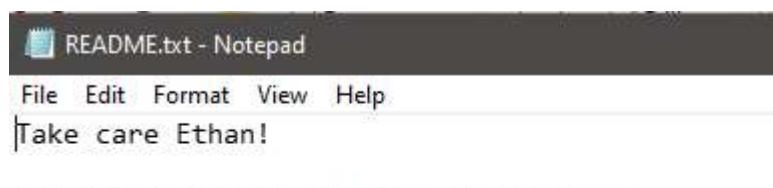
```
[#] stegpy dhsdshdhk.PNG -p
Enter password (will not be echoed):
has_b3en_r3solv3d
```

Figure 12

Part 3: Zip Bomb Analysis

Files: Readme, b.zip.

Approach: Carefully inspect b.zip, noting it's a zip bomb. Use 7z l to list contents and identify a file of differing size. Extracting this file and using binwalk can unveil hidden text, which, when copied and analyzed, reveals a message utilizing whitespace for hidden communication.



README.txt - Notepad
File Edit Format View Help
Take care Ethan!

Some kind of evil is in the file b.zip
DONT EXTRACT OR OPEN!!

Figure 13

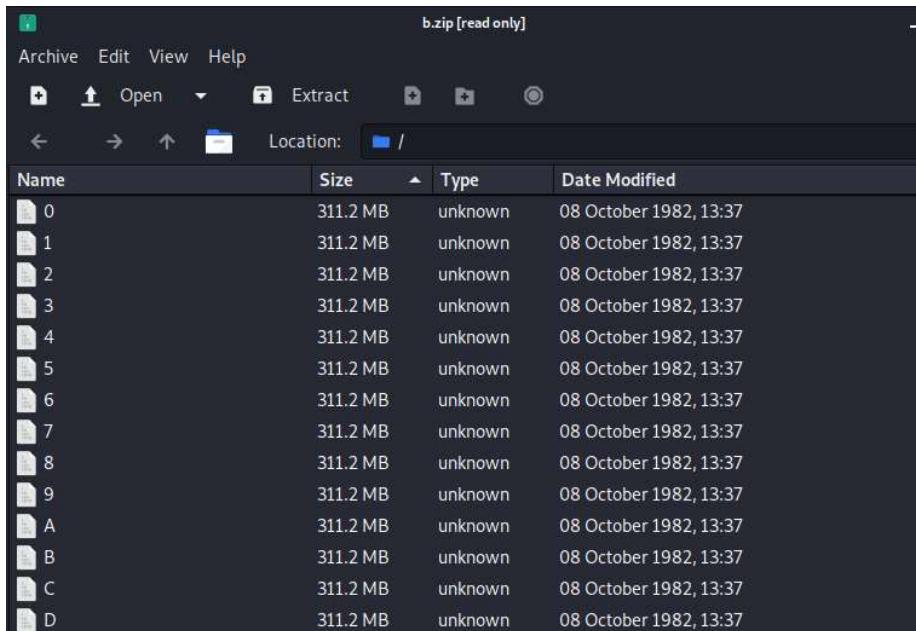


Figure 14

```
7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,4 C

Scanning the drive for archives:
1 file, 4651698 bytes (4543 KiB)

Listing archive: b.zip

--
Path = b.zip
Type = zip
Physical Size = 4651698

      Date      Time    Attr         Size   Compressed  Name
      --      --:--  ----       ----   ----:----  --
1982-10-08 13:37:00  ....       188276        309  0101
1982-10-08 13:37:00  ....     311235134      2200608  0
1982-10-08 13:37:00  ....     311235103      2200572  1
1982-10-08 13:37:00  ....     311235072      2200536  2
1982-10-08 13:37:00  ....     311235041      2200500  3
1982-10-08 13:37:00  ....     311235010      2200464  4
1982-10-08 13:37:00  ....     311234979      2200428  5
1982-10-08 13:37:00  ....     311234948      2200392  6
1982-10-08 13:37:00  ....     311234917      2200356  7
1982-10-08 13:37:00  ....     311234886      2200320  8
1982-10-08 13:37:00  ....     311234855      2200284  9
```

Figure 15

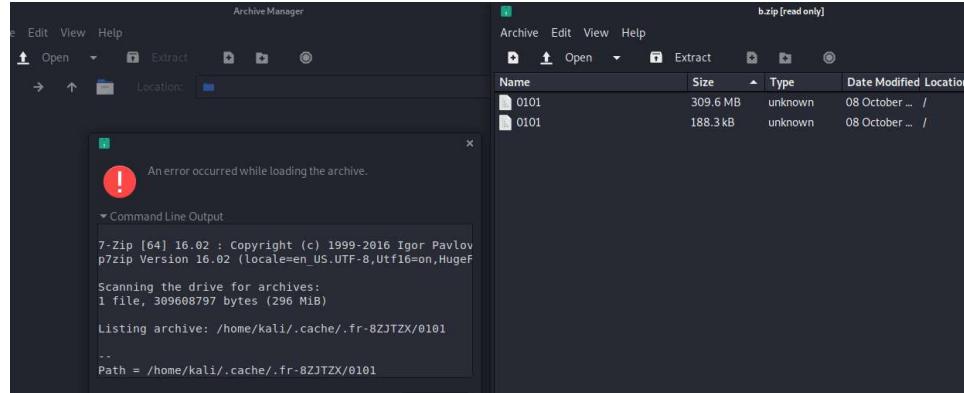


Figure 16

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	Zip archive data, at least v2.0 to extract, compressed size: 490, uncompressed size: 188470, name: 0101
524	0x20C	Zip archive data, at least v2.0 to extract, compressed size: 2200608, uncompressed size: 311235134, name: 04042389
4042389	0x3DAE95	Zip compressed data - version 75.1 (34865172 bytes)
4651857	0x46FB51	End of Zip archive, footer length: 22

Figure 17

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	Zip archive data, at least v2.0 to extract, compressed size: 490, uncompressed size: 188470, name: 0101
524	0x20C	Zip archive data, at least v2.0 to extract, compressed size: 2200608, uncompressed size: 311235134, name: 0
4042389	0x3DAE95	rzip compressed data - version 75.1 (34865172 bytes)
4651857	0x46FB51	End of Zip archive, footer length: 22

Figure 18

Message! ... this should only be read it for someone can understand. sylvarc0n created ways of communication that dont rely just in what you see ...

Figure 19

[Traducir](#) | Desactivar para: inglés

More Advanced Unicode Stego:

This one is more complex than the one below, and uses better looking Homoglyphs. Mostly it uses whitespace for encoding. Got some ideas and suggestion from [Mick Douglas](#) on this one. It uses 7 bit ASCII to save space.

Cover Text To Use:	<input type="text" value="Congratz_You_have"/>	0 characters, can encode 0 bits.
Input (output if decoding):	<input type="text" value="Message! ...this should only be read it for someone can understand. sylvarcOn created ways of communication that dont rely just in what you see..."/>	0 Bits to encode
Stegotext (input if decoding):	<input type="text" value="145"/>	

Encode | **Decode** | **Reset**

Figure 20

Part 4: Linked-Pixel Steganography

Files: linked_all.PNG, flag.txt.

Approach: The file name hints at using Linked-Pixel-Steganography. Analyze the image with the tool, considering the file name for potential coordinates or keys. This step reveals the final part of the flag.

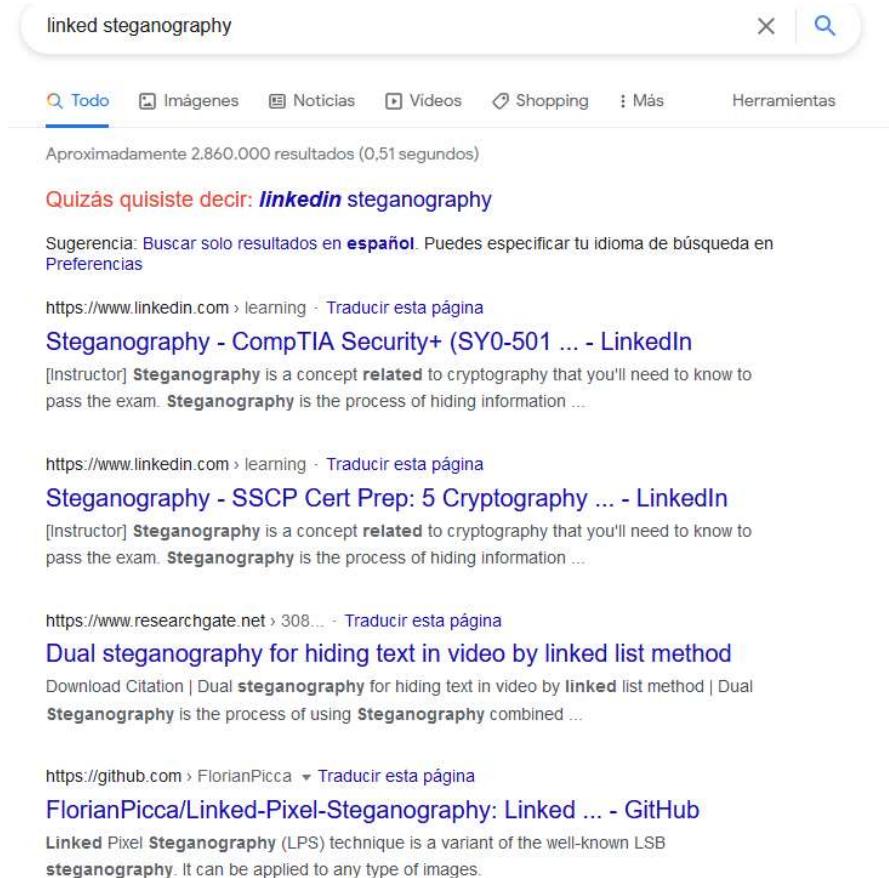


Figure 21

Using Linked-Pixel Steganography, the tool requires an image along with specific X and Y coordinates for uncovering hidden data. The clue for determining these coordinates is ingeniously embedded within the naming itself, by combining the words "linked" and "all" to guide us towards the solution.

```
In [3]: ord('l')+ord('i')+ord('n')+ord('k')+ord('e')+ord('d')
...
In [3]: 631

In [4]: ord('a')+ord('l')+ord('l')
In [4]: 313
```

Figure 22

```
[root@kali /tmp/Linked-Pixel-Steganography]
# python3 unhide.py -f all_x_linked.PNG -o bb.txt -y 313 -x 631
[root@kali /tmp/Linked-Pixel-Steganography]
# cat bb.txt
pass3d_sylvarc0n}
```

Figure 23

Assemble the flag by concatenating the segments uncovered from each part, resulting in the complete flag formatted as `part1_part2_part3_part4`.

Flag Information

flag{Puzzle_has_been_r3solv3d_Congratz_You_have_pass3d_sylvarc0n}