Homework 3

~Purva Naresh Rumde

Problem ID	Captured Flag	Steps
P1	MS-DOS: flag{ 7187D272409E78A914658319A 79D888B} ELF 32: flag{F90F2C6BC5F485027C1F9 AE4CDFF30FA} JPEG: flag{CB0E653885F739E0865A0 626648B4B6E} PDF: flag{90AB7D41D0905B780D29 BECD007E1C1D} ZIP: flag{DECFE44CB3DD1785841D 504797A88D03}	In this problem, you used a hex editor to insert 4 to 16 bytes into a simple text file to convert it into various requested file types. For the EXE file type, the resulting file was identified as an MS DOS executable, representing both PE32 and PE32+ formats. You then used the "file" command to determine the file extension and ran a hexdump equivalent command in PowerShell to display the hexadecimal values of each file. Finally, you calculated the MD5 hash of each file to verify its integrity and uniqueness.
P2	fsuctf{c0rrupted_z1p_h1des_st3ged_jp3g}	In this problem, the provided archived folder was found to be corrupt after downloading and unzipping. To resolve this issue, you used a hex editor to add specific bytes to the folder, allowing it to be downloaded and accessed successfully. After extracting the contents, you discovered an image file containing a hidden flag. Utilizing an online steganography tool called aperisolve, you extracted the hidden flag from the image, which was stored in a text file within the folder.
P3	ictf{z1ps_r3vealed_with_m4gic_byt3s}	In this problem, you were tasked with decoding a provided string using CyberChef. Initially decoding from base64 resulted in garbage symbols. To obtain the correct output, you chose the zip option and dragged the decoded text. Downloading and unzipping the provided zip file revealed the flag, successfully completing the task.
P4	picoCTF{15_y0ur_que57_qu1x071c_0r_h3r01c_ea7deb4c}	In this problem, you utilized the GitHub repo "sigBits" to extract hidden data from an image.

		Initially, you ran the program in the webshell and loaded the image. Using the provided command, you extracted data from the image, creating an output file. To locate the flag within the large file, you used "grep" to filter for relevant content. Finally, using the "vi" command, you successfully located and extracted the flag.

Detailed Explanation:

Q1)

In this problem I have used hexed where I have inserted 4 bytes to 16 bytes for converting the simple text file to the requested file type. I have provided the detailed screenshots for every file type.

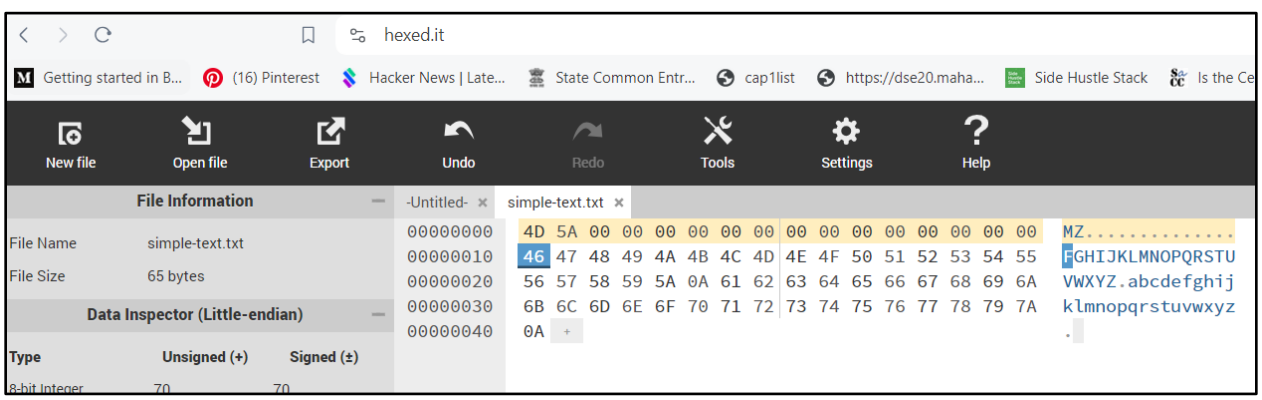
For EXE file the file type is MS DOS executable. It is said that this is represents both PE32 and PE32+. I got the list of files and found out bytes for every file type.

After inserting the bytes. I ran the command “file” on every file to know its extension.

Later I have ran the hexdump equivalent command in PowerShell for every file which gives the hex values.

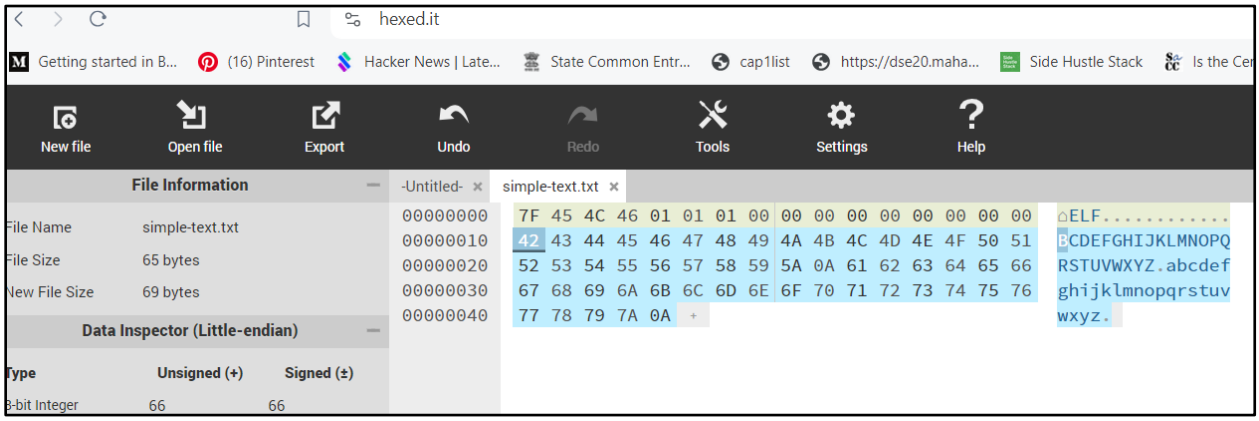
Then I ran one more command to know the MD5 hash of every file.

<u>MS-DOS: flag{ 7187D272409E78A914658319A79D888B}</u>



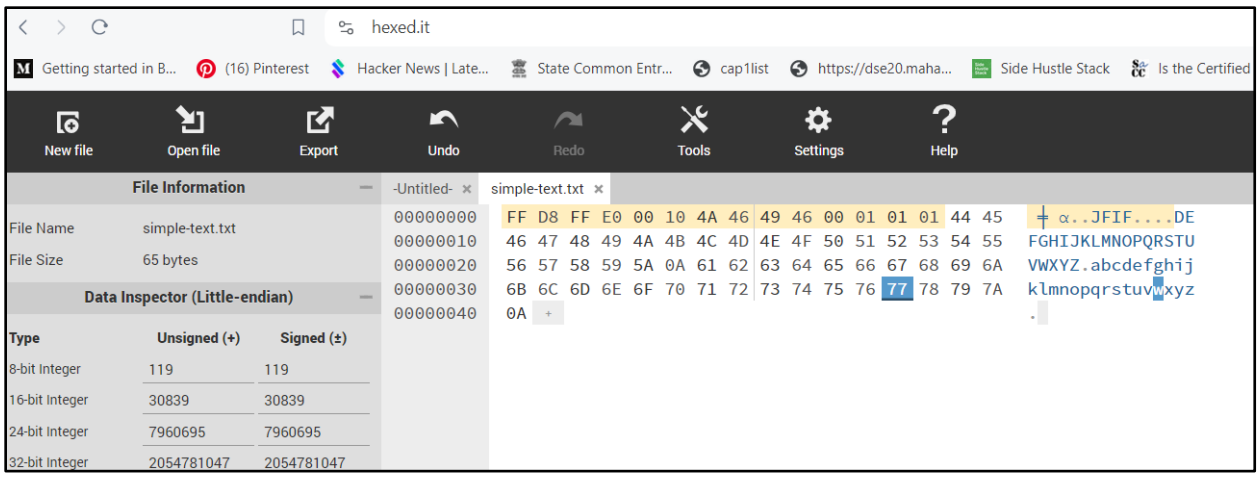
```
PS C:\studies\PCO> Get-Content -Path "EXE.txt" -Encoding Byte -TotalCount 16 | Format-Hex

           Path:

           00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   4D 5A 00 00 00 00 00 00 00 00 00 00 00 00 00 00  MZ..............

PS C:\studies\PCO> Get-FileHash -Algorithm MD5 -Path EXE.txt

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
MD5             7187D272409E78A914658319A79D888B                                       C:\studies\PCO\EXE.txt
```

ELF 32 bit: flag{F90F2C6BC5F485027C1F9AE4CDFF30FA}



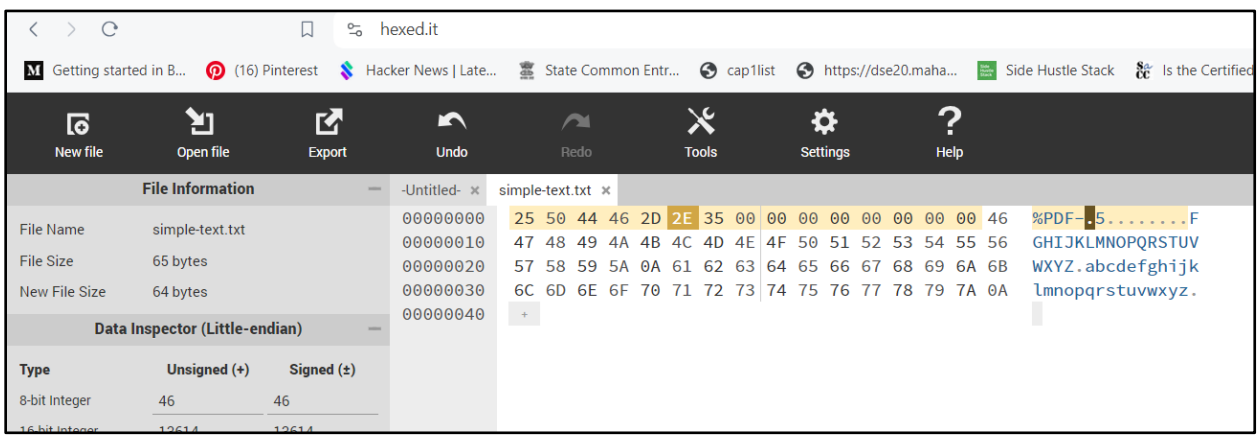

```
PS C:\studies\PCO> Get-Content -Path "ELF.txt" -Encoding Byte -TotalCount 16 | Format-Hex

           Path:

           00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00  ELF............

PS C:\studies\PCO> Get-FileHash -Algorithm MD5 -Path ELF.txt

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
MD5             F90F2C6BC5F485027C1F9AE4CDFF30FA                                       C:\studies\PCO\ELF.txt

PS C:\studies\PCO> Get-Content -Path "ZIP.txt" -Encoding Byte -TotalCount 16 | Format-Hex
```

JPEG: flag{CB0E653885F739E0865A0626648B4B6E}




```
PS C:\studies\PCO> Get-Content -Path "JPEG.txt" -Encoding Byte -TotalCount 16 | Format-Hex

           Path:

           00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   FF D8 FF E0 00 10 4A 46 49 46 00 01 01 01 44 45  .Ø.à..JFIF....DE


PS C:\studies\PCO> Get-FileHash -Algorithm MD5 -Path JPEG.txt

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
MD5             CB0E653885F739E0865A0626648B4B6E                                       C:\studies\PCO\JPEG.txt
```

PDF: flag{90AB7D41D0905B780D29BECD007E1C1D}

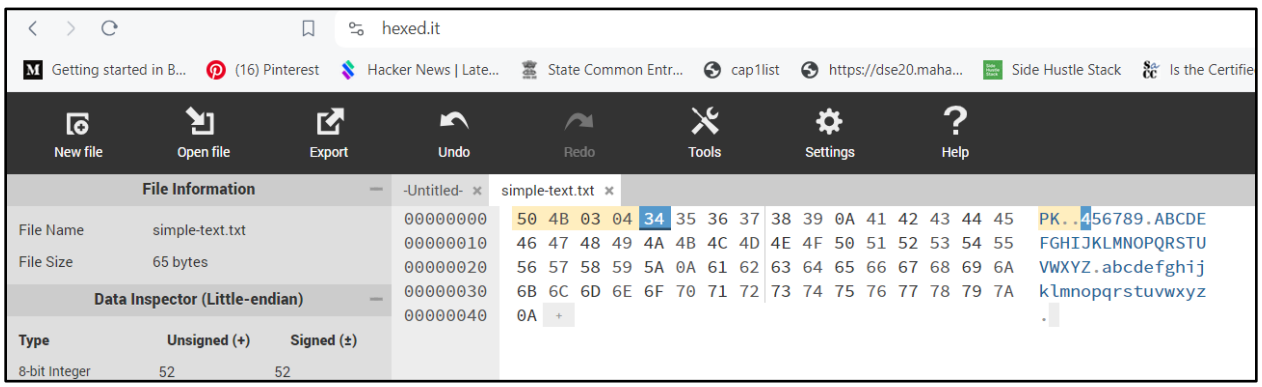


```
PS C:\studies\PCO> Get-Content -Path "PDF.txt" -Encoding Byte -TotalCount 16 | Format-Hex

           Path:

           00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   25 50 44 46 2D 2E 35 00 00 00 00 00 00 00 00 46  %PDF-.5.......F

PS C:\studies\PCO> Get-FileHash -Algorithm MD5 -Path PDF.txt

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
MD5             90AB7D41D0905B780D29BECD007E1C1D                                       C:\studies\PCO\PDF.txt
```

ZIP: flag{DECFE44CB3DD1785841D504797A88D03}/ flag{ 4369A0904E44037BF6CBD15226F28AB4}

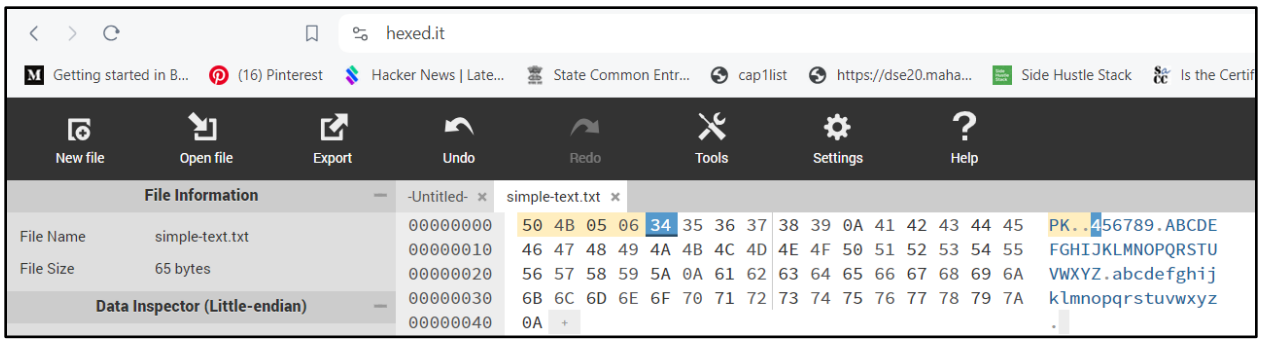
For ZIP file I have tried to different bytes since the first one is reading the file type as data and not zip whereas the other byte is reading it as empty archive. But I am aware that it should not be giving empty archive. I will attaching both. ZIP.txt is the first one and ZIP2.txt is done using empty archive.

ZIP.txt: flag{ 4369A0904E44037BF6CBD15226F28AB4}




```
PS C:\studies\PCO> Get-Content -Path "ZIP.txt" -Encoding Byte -TotalCount 16 | Format-Hex

           Path:

           00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   50 4B 03 04 34 35 36 37 38 39 0A 41 42 43 44 45  PK..456789.ABCDE

PS C:\studies\PCO> Get-FileHash -Algorithm MD5 -Path ZIP.txt

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
MD5             4369A0904E44037BF6CBD15226F28AB4                                       C:\studies\PCO\ZIP.txt
```

ZIP1.txt: flag{ DECFE44CB3DD1785841D504797A88D03}




```
PS C:\studies\PCO> Get-Content -Path "ZIP1.txt" -Encoding Byte -TotalCount 16 | Format-Hex

           Path:

           00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   50 4B 05 06 34 35 36 37 38 39 0A 41 42 43 44 45  PK..456789.ABCDE


PS C:\studies\PCO> Get-FileHash -Algorithm MD5 -Path ZIP1.txt

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
MD5             DECFE44CB3DD1785841D504797A88D03                                       C:\studies\PCO\ZIP1.txt
```

To know the file type:

```
purva@Purva UCRT64 ~
$ file "C:\studies\PCO\EXE.txt"
C:\studies\PCO\EXE.txt: MS-DOS executable
```

```
purva@Purva UCRT64 ~
$ file "C:\studies\PCO\JPEG.txt"
C:\studies\PCO\JPEG.txt: JPEG image data, JFIF standard 1.01, resolution (DPI), density 17477x17991,
 segment length 16, thumbnail 72x73

purva@Purva UCRT64 ~
$ file "C:\studies\PCO\ELF.txt"
C:\studies\PCO\ELF.txt: ELF 32-bit LSB *unknown arch 0x4948* (SYSV)

purva@Purva UCRT64 ~
$ file "C:\studies\PCO\PDF.txt"
C:\studies\PCO\PDF.txt: PDF document, version ..

purva@Purva UCRT64 ~
$ file "C:\studies\PCO\ZIP.txt"
C:\studies\PCO\ZIP.txt: data
```

```
purva@Purva UCRT64 ~
$ file "C:\studies\PCO\ZIP1.txt"
C:\studies\PCO\ZIP1.txt: Zip archive data
```

Q2)

fsuctf{c0rrupted_z1p_h1des_st3ged_jp3g}

In this problem we are supposed to download the provided archived folder. But after downloading and unzipping it, the file in that folder didn't open and gave a file corrupt error.

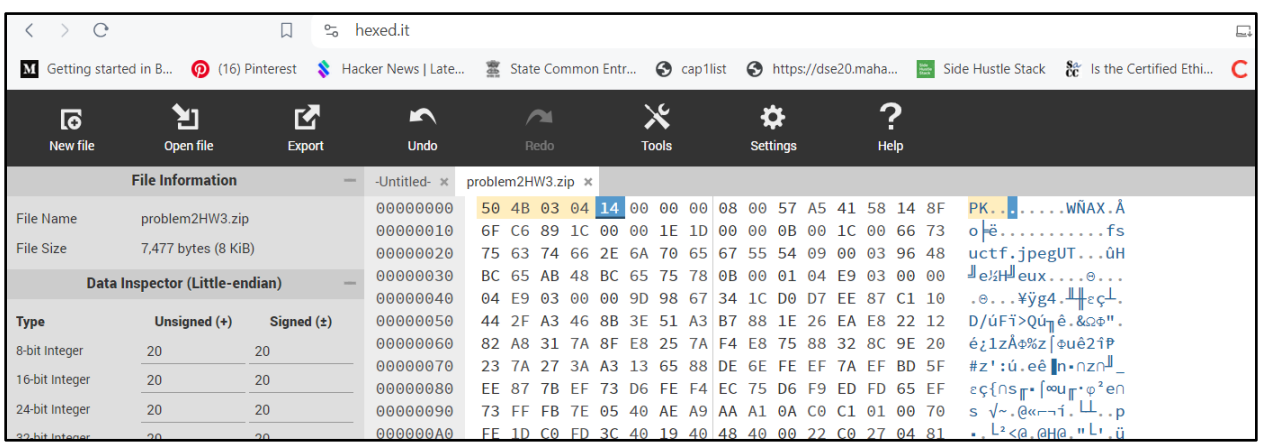
So for this problem I tried using the first problem solution that is adding bytes to the unzipped folder by opening it in hexed.

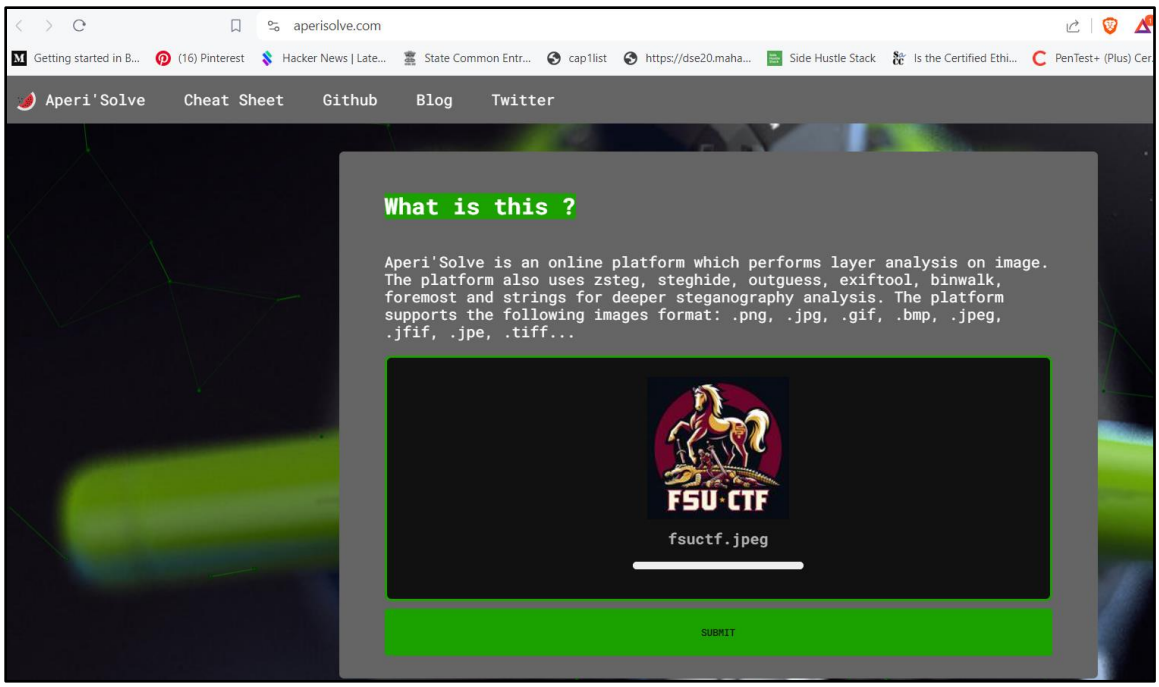
I added the "`50 4B 03 04`" and then downloaded the folder. This worked for me and I could open the image provided in that folder.

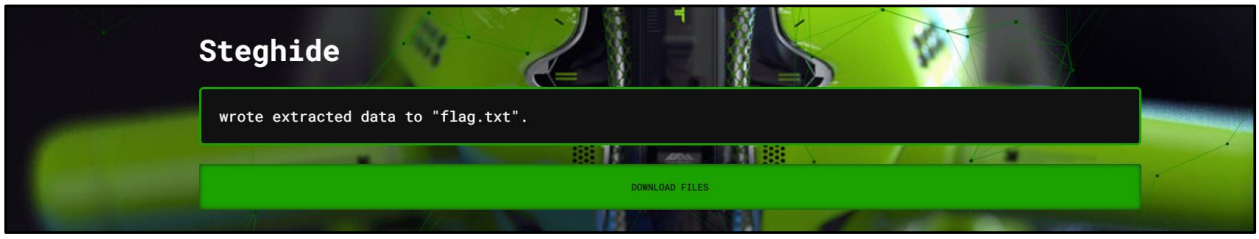
Next I had to find the flag hidden in that image. For that I used an online tool aperisolve.

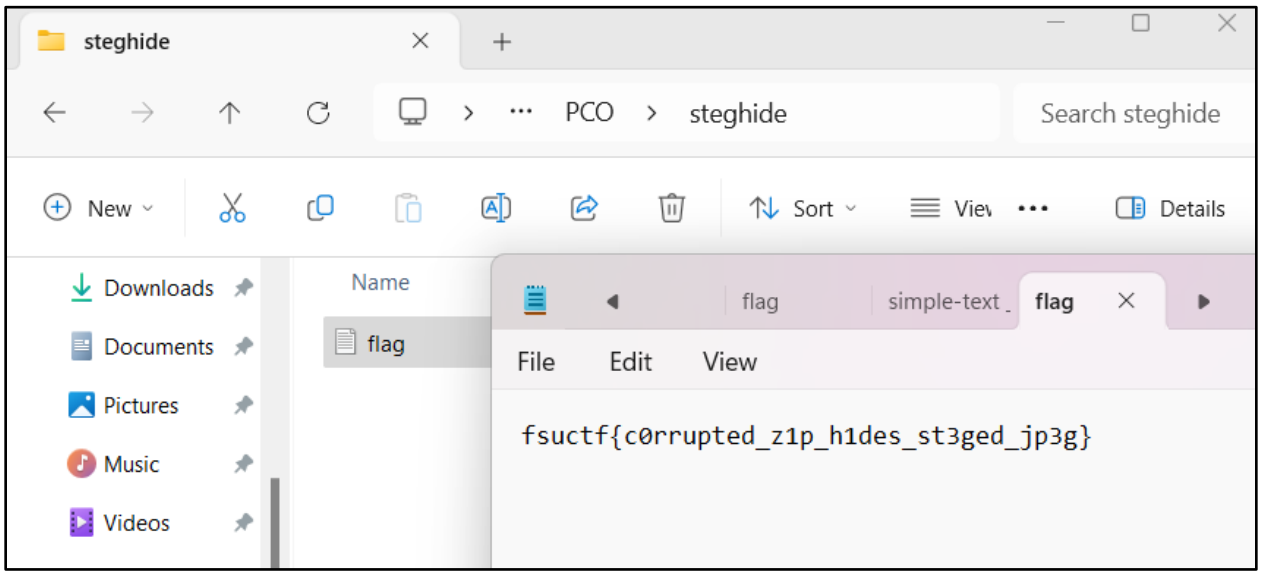
I opened the image in it and downloaded from the steghide section a text file that provided me with the flag.

50 4B 03 04 50 4B 05 06 (empty archive) 50 4B 07 08 (spanned archive)	PK_{ETX}_{EOT} PK_{ENQ}_{ACK} PK_{BEL}_{BS}	0	zip aar apk docx epub ipa jar kmz maff msix odp ods odt pk3 pk4 pptx usdz vsdx	zip file format and formats based on it, such as EPUB, JAR, ODF, OOXML




aperisolve.com
Aperi'Solve
Cheat Sheet
Github
Blog
Twitter
What is this ?
Aperi'Solve is an online platform which performs layer analysis on image. The platform also uses zsteg, steghide, outguess, exiftool, binwalk, foremost and strings for deeper steganography analysis. The platform supports the following images format: .png, .jpg, .gif, .bmp, .jpeg, .jfif, .jpe, .tiff...
FSU·CTF
fsuctf.jpeg
SUBMIT


Steghide
wrote extracted data to "flag.txt".
DOWNLOAD FILES


steghide
PCO
steghide
Search steghide
New
Sort
View
Details
Downloads
Documents
Pictures
Music
Videos
Name
flag
flag
simple-text
flag
File
Edit
View
fsuctf{c0rrupted_z1p_h1des_st3ged_jp3g}

Q3)

ictf{z1ps_r3vealed_with_m4gic_byt3s}

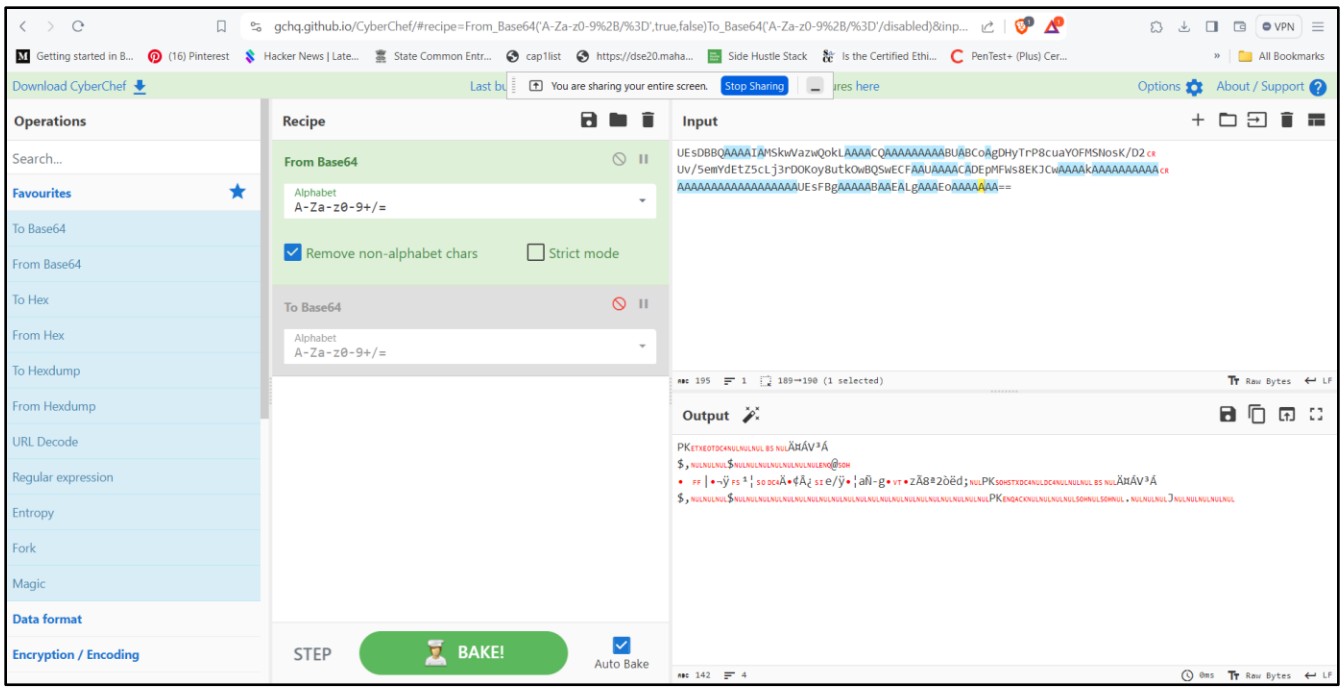
In this problem we are supposed to decode the provided string. I utilized a site called CyberChef.

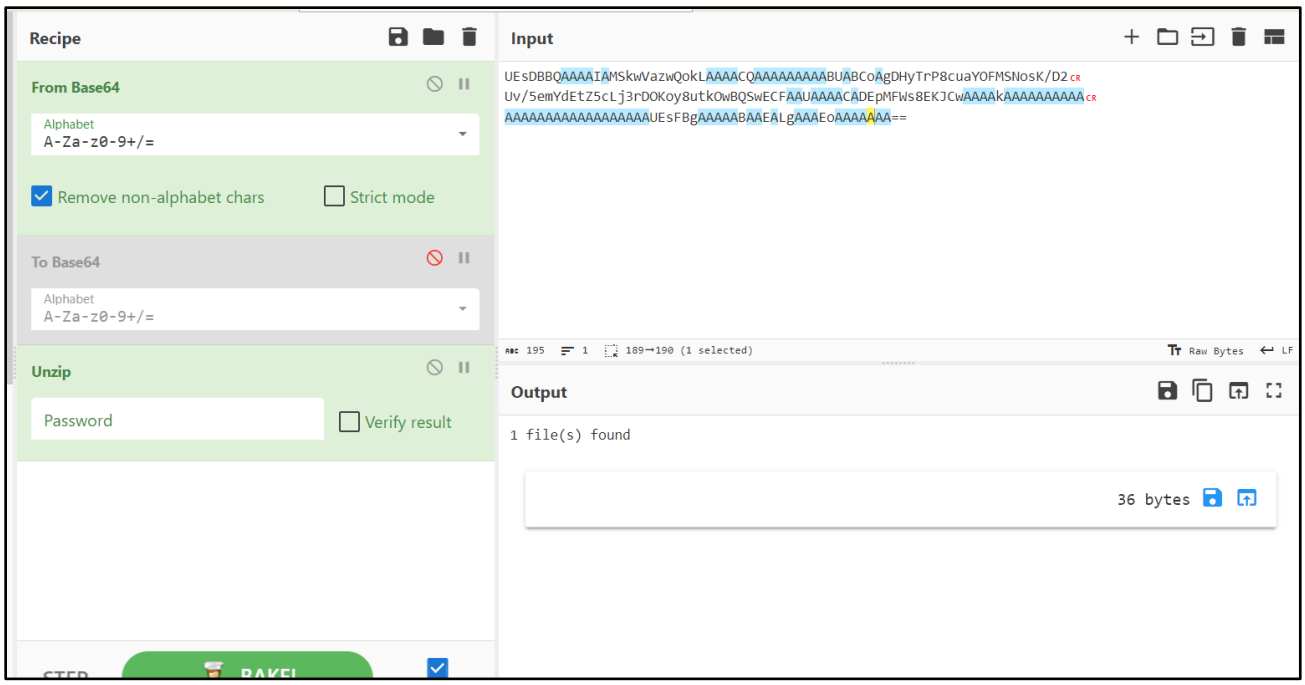
I pasted the string in it and chose the option “from base64”, which gave me the ouput in a garbage symbol format.

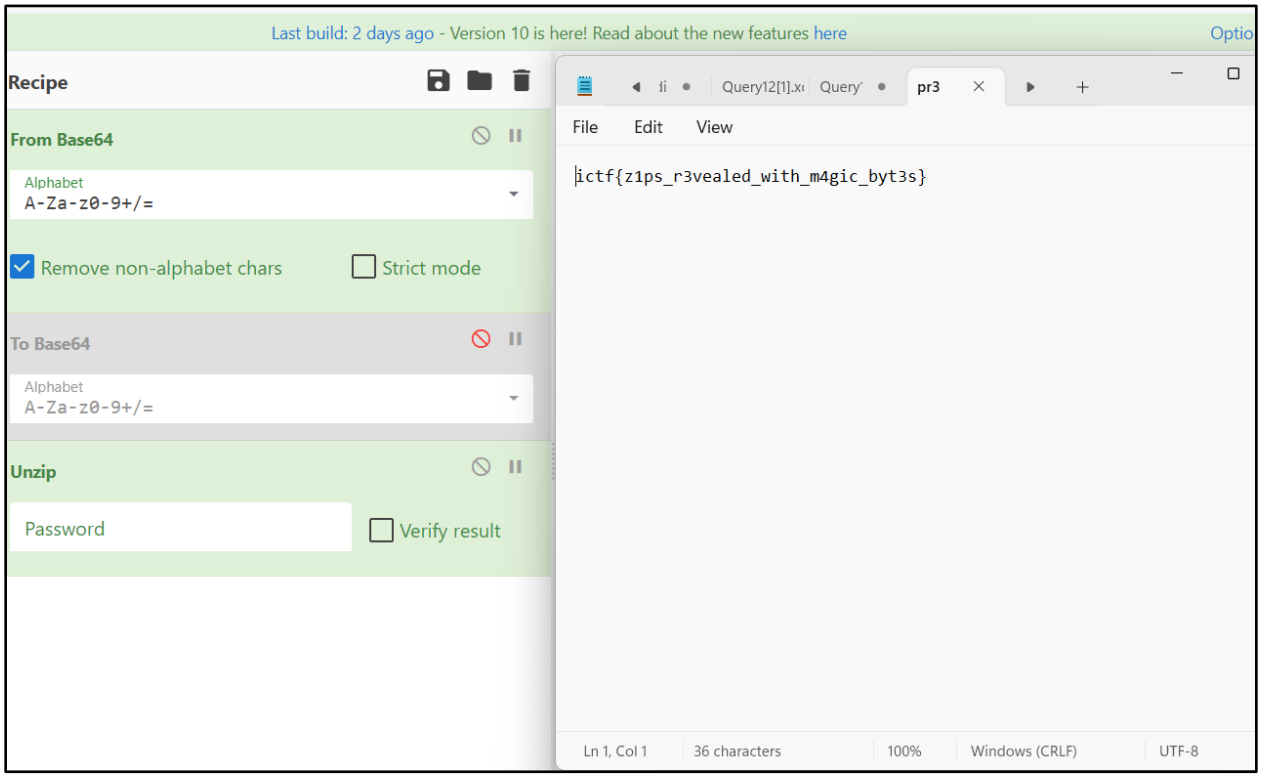
To get the right output that is the flag, I then chose the zip option and dragged it.

Now I just clicked on download the provided zip file and unzipped it on my system.

And that worked and I could see the flag.




Recipe
From Base64
Alphabet
A-Za-z0-9+/=
Remove non-alphabet chars
Strict mode
To Base64
Alphabet
A-Za-z0-9+/=
Unzip
Password
Verify result
Input
UEsDBBQAAAAIAMSkwVazwQokLAAAACQAAAAAAAAABUABCoAgDHyTrP8cuaYOFMSNosK/D2
Uv/5emYdEtZ5cLj3rDOKoy8utkOwBQSwECFAAUAAAACADEpMFWs8EKJCwAAAAkAAAAAAAAAA
AAAAAAAAAAAAAAAAAUEsFBgAAAAABAAEALgAAAEoAAAAAAA==
195 1 189→190 (1 selected)
Raw Bytes LF
Output
1 file(s) found
36 bytes


Last build: 2 days ago - Version 10 is here! Read about the new features here
Recipe
From Base64
Alphabet
A-Za-z0-9+/=
Remove non-alphabet chars
Strict mode
To Base64
Alphabet
A-Za-z0-9+/=
Unzip
Password
Verify result
Query12[1].x
Query
pr3
File Edit View
ictf{z1ps_r3vealed_with_m4gic_byt3s}
Ln 1, Col 1
36 characters
100%
Windows (CRLF)
UTF-8

Q4)

picoCTF{15_y0ur_que57_qu1x071c_0r_h3r01c_ea7deb4c}

In this problem I used the GitHub repo called sigBits that helps in getting the hidden output from an image.

First, I ran the program in the webshell. And loaded the image in webshell.

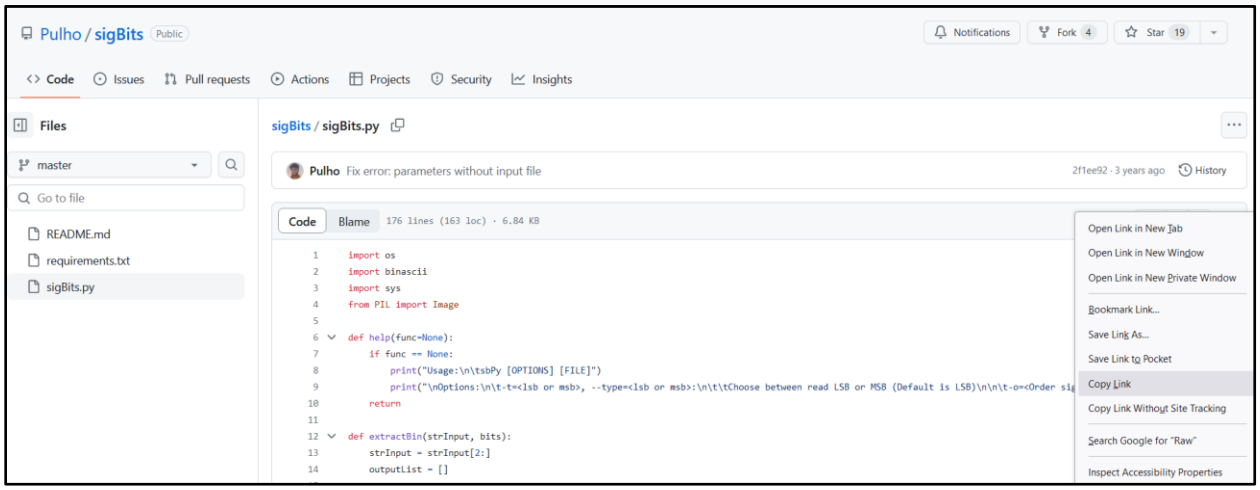
Then ran the provided command "**`python sigBits.py --type=Msb Ninja-and-Prince-Genji-Ukiyoe-Utagawa-Kunisada.flag.png`**"

After which I did ls and checked if I have got the outputSB.txt file created and it did.

Now I opened this file with cat but since the file was too big it was not easy to find the flag.

For which I used the cat outputs.txt | grep pico

And then with the help of vi command I found the flag.



https://webshell.picoctf.org

Getting Started | Getting started in Bug ... | (16) Pinterest | Hacker News | Latest ... | State Common Entran... | cap1list | https://dse20.mahacet... | Side Hustle Stack | Is the Certified Ethical ... | PenTest+ (Plus) Certifi...

```
Enter your picoCTF username: bingchillingg
Enter your picoCTF password (characters will be hidden):

==================================================================

Welcome to the picoCTF webshell!

The webshell is intended only for solving picoCTF challenges. Any
other usage is a violation of our terms and conditions.

Sessions are monitored and logged to prevent abuse. Please do not
enter any sensitive information into the webshell.

Files stored outside of your home directory will not persist between
webshell sessions.

Network connectivity and resources are limited. Some limits can be
checked by typing usage.

Idle sessions will automatically log out after 15 minutes.

For more information and a beginner's guide, type less ~/README.txt.

==================================================================

bingchillingg-picoctf@webshell:~$ wget https://github.com/Pulho/sigBits/raw/master/sigBits.py
--2024-02-08 16:08:01--  https://github.com/Pulho/sigBits/raw/master/sigBits.py
Resolving github.com (github.com)... 140.82.114.3
Connecting to github.com (github.com)|140.82.114.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/Pulho/sigBits/master/sigBits.py [following]
--2024-02-08 16:08:01--  https://raw.githubusercontent.com/Pulho/sigBits/master/sigBits.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.110.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7008 (6.8K) [text/plain]
Saving to: 'sigBits.py'

sigBits.py                                          100%[===================================================================================================>]  6.84K  --.-KB/s    in 0.001s

2024-02-08 16:08:01 (5.48 MB/s) - 'sigBits.py' saved [7008/7008]
```

icoCTF Learn Practice Compete

Open Link in New Tab
Open Link in New Window
Open Link in New Private Window
Bookmark Link...
Save Link As...
Save Link to Pocket
Copy Link
Copy Link Without Site Tracking
Search Google for "here"
Inspect Accessibility Properties
Inspect (Q)

MSB

Tags: picoCTF 2023 For

AUTHOR: LT 'SYREAL' JONES

Description

This image passes LSB st ... help but think there must be som ... esent in this image...

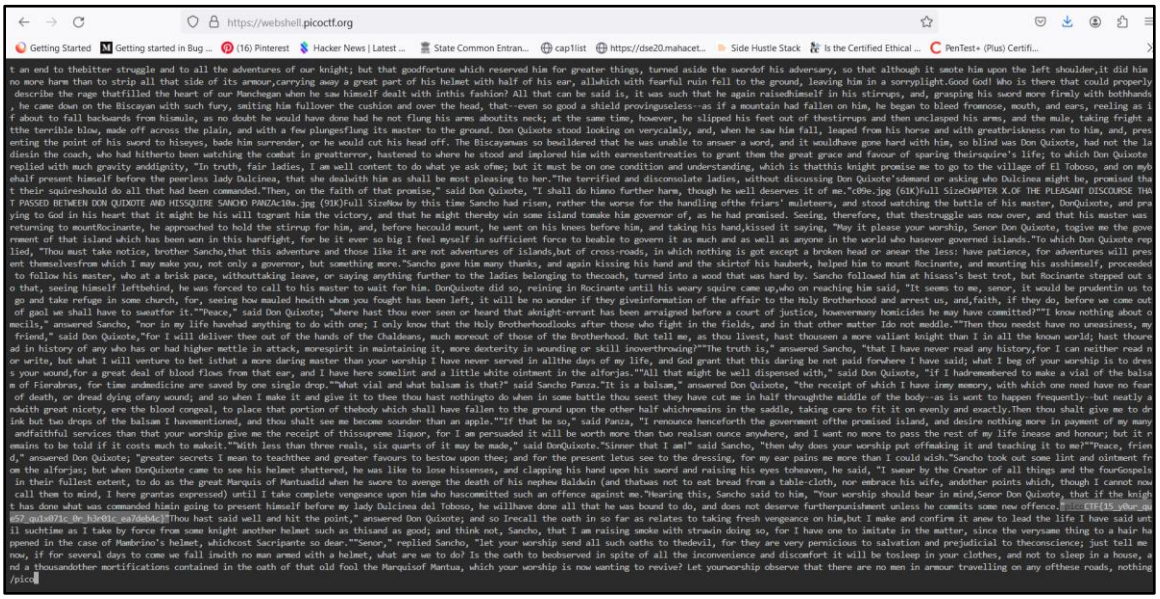
Download the image here

2,682 users solved

```
bingchillingg-picoctf@webshell:~$ wget https://artifacts.picoctf.net/c/302/Ninja-and-Prince-Genji-Ukiyoe-Utagawa-Kunisada.flag.png
--2024-02-08 16:09:34--  https://artifacts.picoctf.net/c/302/Ninja-and-Prince-Genji-Ukiyoe-Utagawa-Kunisada.flag.png
Resolving artifacts.picoctf.net (artifacts.picoctf.net)... 3.160.22.16, 3.160.22.43, 3.160.22.128, ...
Connecting to artifacts.picoctf.net (artifacts.picoctf.net)|3.160.22.16|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3354312 (3.2M) [application/octet-stream]
Saving to: 'Ninja-and-Prince-Genji-Ukiyoe-Utagawa-Kunisada.flag.png'

Ninja-and-Prince-Genji-Ukiyoe-Utagawa-Kunisada.flag.png    100%[===================================================================================

2024-02-08 16:09:36 (1.82 MB/s) - 'Ninja-and-Prince-Genji-Ukiyoe-Utagawa-Kunisada.flag.png' saved [3354312/3354312]

bingchillingg-picoctf@webshell:~$
```

```
bingchillingg-picoctf@webshell:~$ python sigBits.py --type=Msb Ninja-and-Prince-Genji-Ukiyoe-Utagawa-Kunisada.flag.png
Done, check the output file!
bingchillingg-picoctf@webshell:~$ ls
Ninja-and-Prince-Genji-Ukiyoe-Utagawa-Kunisada.flag.png    Ninja-and-Prince-Genji-Ukiyoe-Utagawa-Kunisada.flag.png.2  outputSB.txt  sigBits.py.1
Ninja-and-Prince-Genji-Ukiyoe-Utagawa-Kunisada.flag.png.1  README.txt                                                 sigBits.py    sigBits.py.2
bingchillingg-picoctf@webshell:~$ ls -l
total 10884
-rw-rw-r-- 1 bingchillingg-picoctf bingchillingg-picoctf 3354312 Aug  4  2023 Ninja-and-Prince-Genji-Ukiyoe-Utagawa-Kunisada.flag.png
-rw-rw-r-- 1 bingchillingg-picoctf bingchillingg-picoctf 3354312 Aug  4  2023 Ninja-and-Prince-Genji-Ukiyoe-Utagawa-Kunisada.flag.png.1
-rw-rw-r-- 1 bingchillingg-picoctf bingchillingg-picoctf 3354312 Aug  4  2023 Ninja-and-Prince-Genji-Ukiyoe-Utagawa-Kunisada.flag.png.2
-rw-r--r-- 1 root                  root                    4443 Feb  8 16:26 README.txt
-rw-rw-r-- 1 bingchillingg-picoctf bingchillingg-picoctf  529769 Feb  8 16:33 outputSB.txt
-rw-rw-r-- 1 bingchillingg-picoctf bingchillingg-picoctf    7008 Feb  8 16:08 sigBits.py
-rw-rw-r-- 1 bingchillingg-picoctf bingchillingg-picoctf    7008 Feb  8 16:14 sigBits.py.1
-rw-rw-r-- 1 bingchillingg-picoctf bingchillingg-picoctf    7008 Feb  8 16:27 sigBits.py.2
bingchillingg-picoctf@webshell:~$ cat outputSB.txt
```

```
mmmmmmmmi6Im$I $&HIIHI i$I Imm}Hmm4i&MMII$@&@ Mi&m$M$I&m$i6I6i$mmmImmmmi&mM}mm$mmi$m&mMI$I  ?mI$I$I$I$i$I$I$I I$I&I6II$A$I$HI$MMi$Im$mMMi$I$
I6$[d$I$I$m$$I6Im$I$mmOmI$mi&I$I$mm&Ii4I$I$I$I $I$MI$Mi&mmMIom&MMmmmioomi6omMMm&I&Mi6I$mi&mMiMmmm&@$$$I$i$i4I$IImmmmOMMmmMI6mmi&omi$HoI$I$I$
I$mmIiH&I6m$M6mI'm'}M$H6I%_H$A&A$HA$II&KI$m&mooi&immmI&i$I,mI$m&IMm&mmmmmmimi6mm}MimI$m$i&om$mMI6Mmi6M$mM/I6&MMi$/m$I?MI&}mm6I/IMOImmi6Mi$Im
IM$MmmmmmmOmmoOm$i&mi?mmmmMMi$M6Mm$mmIMmMm&I6H@$I$I$MI$i$Iommmm_ooi6mom&mmmmM$mI6mI$mI?OI_$Mm$m$mIII'M&i6m&i&o_M'i4omm$MIMI&i/mM4Im?OK%M?imm
bingchillingg-picoctf@webshell:~$ cat outputSB.txt | grep pico
```



MSB

| 200 points

Tags: picoCTF 2023 | Forensics | steganography

AUTHOR: LT 'SYREAL' JONES

Description

This image passes LSB statistical analysis, but we can't help but think there must be something to the visual artifacts present in this image...

Download the image here

Hints

1

2,682 users solved

83% Liked

picoCTF{15_y0ur_que57_qu1x071c_0r_h3r01c_ea7deb4c}

Submit Flag