12 SEPTEMBER 2023

CYBER SECURITY

Systems_CTF{Flag}

POINTS - 225

Arjun Sekar 2nd Year BTech EE
Username : CARBON_VIBES
Roll_No : 22110034

Steganography:

1. Strange Image :

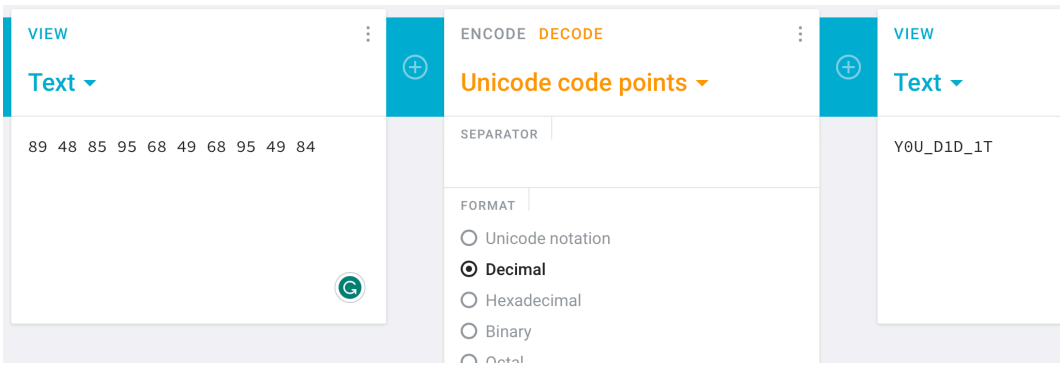
Strange Image

In this problem we are given a png file which needs to processed to extract the flag which is embedded inside it. According to me one of the best tools to start with these type of problems is the exiftool, On running the command exiftool file.png we get :

```
Red Matrix Column               : 0.51512 0.2412 -0.00105
Green Matrix Column             : 0.29198 0.69225 0.04189
Blue Matrix Column              : 0.1571 0.06657 0.78407
Red Tone Reproduction Curve     : (Binary data 2060 bytes, use -b option to extract)
Video Card Gamma                : (Binary data 48 bytes, use -b option to extract)
Native Display Info             : (Binary data 62 bytes, use -b option to extract)
Make And Model                  : (Binary data 40 bytes, use -b option to extract)
Blue Tone Reproduction Curve    : (Binary data 2060 bytes, use -b option to extract)
Green Tone Reproduction Curve   : (Binary data 2060 bytes, use -b option to extract)
Exif Byte Order                 : Big-endian (Motorola, MM)
X Resolution                    : 144
Y Resolution                    : 144
Resolution Unit                 : inches
User Comment                    : Screenshot
Exif Image Width                : 1326
Exif Image Height               : 1114
Pixels Per Unit X               : 5669
Pixels Per Unit Y               : 5669
Pixel Units                     : meters
XMP Toolkit                     : XMP Core 6.0.0
Apple Data Offsets              : (Binary data 28 bytes, use -b option to extract)
Comment                         : this may be flag: 89488595684968954984
```

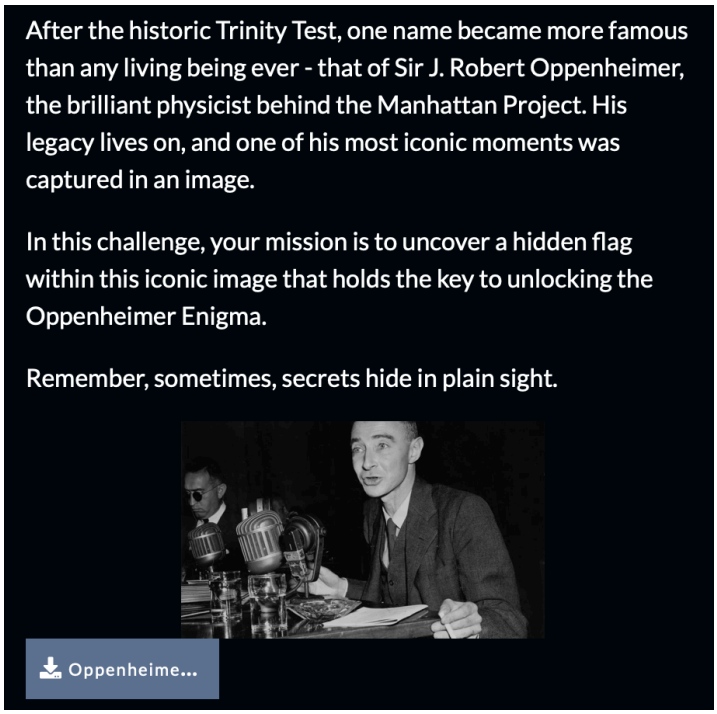
OUTPUT obtained by using exiftool

From the above figure we can observe a clue stating that the flag is 89488595684968954984, however it is evident that this flag obtained is not in the final form which is supposed to be submitted , one can easily guess that it is encoded, and it also doesn't look like hexadecimal or octal encoding, therefore we can probably guess that this is decimal encoding. On using Decimal(ASCII) to Text converter we obtain :



DECIMAL(ASCII) TO TEXT

2. BOOM BOOM :

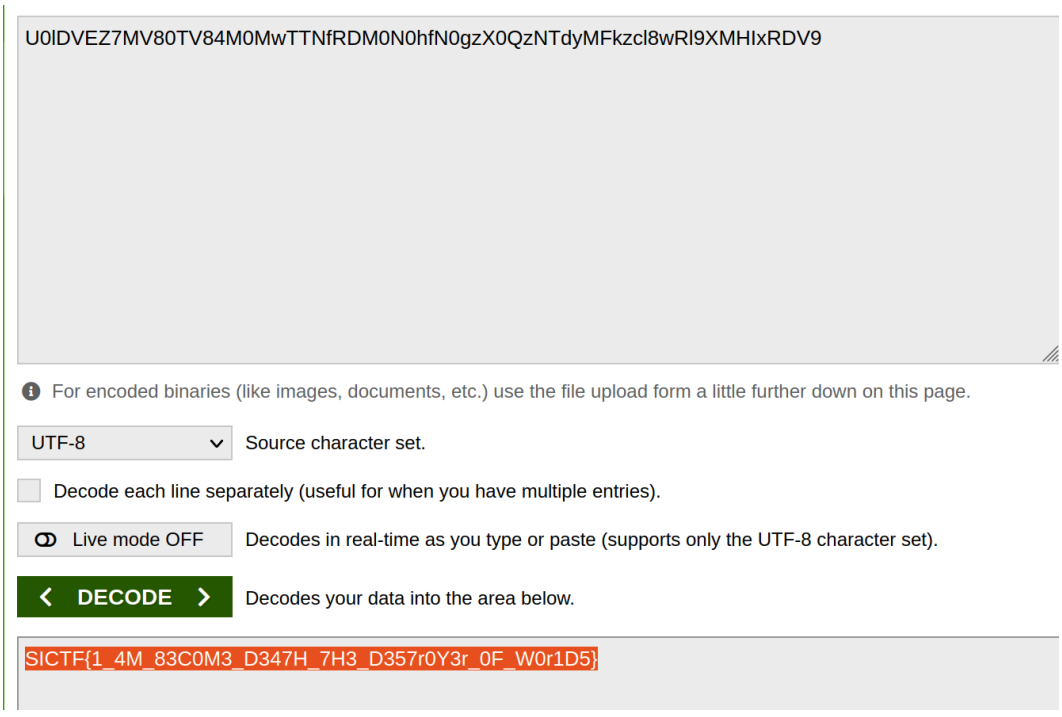


Caption

This problem is quite similar to the previous one but, on running commands like exiftool and binwalk, we don't find any suspicious embedded files or any clues like in the previous question, but there is a possibility that the flag might be embedded directly into the image using tools like steghide. On exploring on that idea by running the command steghide extract -sf Oppenheimer.jpg we get:

```
parallels@ubuntu-linux-22-04-desktop:~/step$ steghide extract -sf Oppenheimer.jpg
Enter passphrase:
the file "flag" does already exist. overwrite ? (y/n) y
wrote extracted data to "flag".
parallels@ubuntu-linux-22-04-desktop:~/step$
parallels@ubuntu-linux-22-04-desktop:~/step$ cat flag
U0lDVEZ7MV80TV84M0MwTTNfRDM0N0hfN0gzX0QzNTdyMFkzcl8wRl9XMHIxRDV9
```

With the help of steghide command we were able to obtain the encoded flag, which on keen observation seems to be a base64 or base32 encoding. On trying out base64 decoding we get the final decoded flag as shown below :

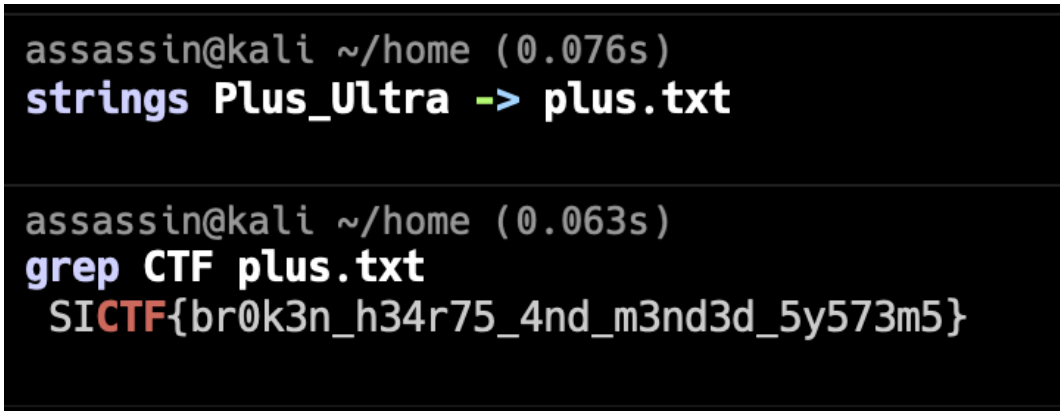


Base64 decoding

ELF :

1. Break My Limits :

In this problem we are provided with a file with a "Plus_Ultra" which has no extension, This can be considered as a classic entry_level CTF problem, which helps people to get started with the contest. So in order to find the hidden flag we can just cat the contents of the Plus_Ultra File, on doing that we can observe that this file contains a lot of unwanted binary data which is not important to us. So in order to obtain the FLAG which is a text data, we can use strings command to extract the text content and then we can use grep common to search for the hidden flag.



Caption

MISC :

1. DOES THIS EVER END ?

In this problem we are given a random zip file which on extracting contains another zip file this pattern continues so on and so forth. This is a typical nested zip file CTF problem, where the flag is hidden in a text file in the innermost directory. It is almost impossible for manually extracting each and every zip file separately as this is an extremely tedious way to solve this question. So the best way to do this is by writing a python code using the OS library to extract the files until the innermost text file is obtained. It is to be noted that recursion doesn't seems to be a nice idea in this question as it exceeds the maximum recursion depth so writing a loop would be the ideal method to proceed.

```python
import zipfile
import os

def extract_nested_zip(zip_file_path, extract_to):
    stack = [(zip_file_path, extract_to)]

    while stack:
        current_zip, current_extract_to = stack.pop()

        with zipfile.ZipFile(current_zip, 'r') as zip_ref:
            zip_ref.extractall(current_extract_to)

            for entry in zip_ref.namelist():
                entry_path = os.path.join(current_extract_to, entry)<0x0d>]
                if entry.endswith('.zip'):
                    stack.append((entry_path, current_extract_to))
                else:
                    return entry_path

    return None

oz = '/home/parallels/Documents/abc.zip'
od = '/home/parallels/Documents'

final_file_path = extract_nested_zip(oz, od)

if final_file_path:
    print(f"Final file extracted: {final_file_path}")
else:
    print("No final file found.")

```

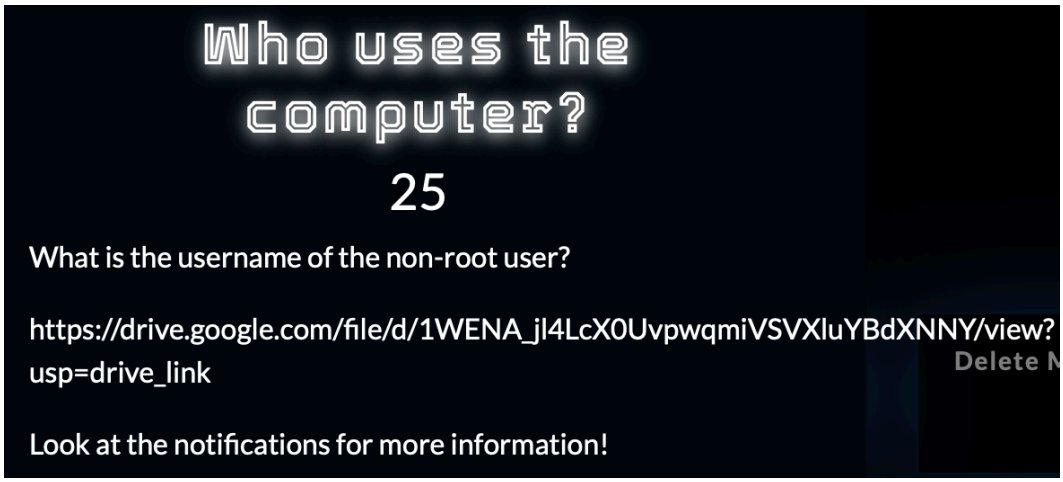
Python Code

With the help of the above python code we could extract all the zip files to the Documents folder and we can obtain the hidden flag.txt file as well. The command cat flag.txt is to be used to find the final flag.

Extracted Zip Files

LINUX LONG CHALLENGE :

1. Who Uses The Computer



Caption

In this category of questions under linux long challenge we are given a link to download the image (.img) file of a linux distro with some data in it, it worth mentioning that the linux distro provided is pre-used one containing the data of the user. I wasn't able to burn the .img file to a bootable disk and live boot it as it was showing me that the .img file is corrupted, therefore I used 7-zip to extract the contents of the .img file and moved it to my local device. Therefore I wasn't able to directly find the users using a simple linux command. But one with basic understanding of linux subsystem knows that the user details along with their password and other details are stored in /etc/shadow.

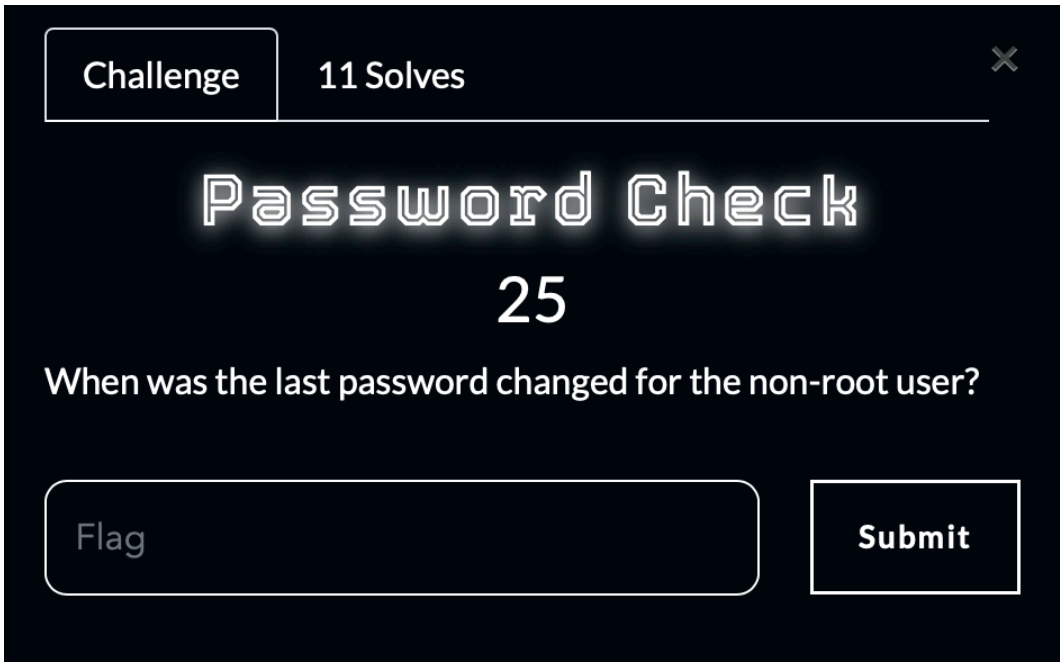
```
parallels@ubuntu-linux-22-04-desktop:~/Downloads$ cd arjun/
parallels@ubuntu-linux-22-04-desktop:~/Downloads/arjun$ ls
'[SYS]'       boot   home         media   proc   sbin   sys   var
'[UNKNOWN]'   dev    lib          mnt     root   srv    tmp
 bin          etc    lost+found   opt     run    swap   usr
parallels@ubuntu-linux-22-04-desktop:~/Downloads/arjun$
```

```
parallels@ubuntu-linux-22-04-desktop:~/Downloads/arjun/etc$ cat shadow
root:$6$uvX6.DiJCeNJv7cA$VW8RXpb4maY.jcznupkOgo7JqEwdqzRIN2PQj3T36ObWhTn3xfpzyEoyiFnCHyxldKJp
smtzg5oS4GIatlbw3/:19609:0:::::
bin:!::0:::::
daemon:!::0:::::
adm:!::0:::::
lp:!::0:::::
sync:!::0:::::
shutdown:!::0:::::
halt:!::0:::::
mail:!::0:::::
news:!::0:::::
uucp:!::0:::::
operator:!::0:::::
man:!::0:::::
postmaster:!::0:::::
cron:!::0:::::
ftp:!::0:::::
sshd:!::0:::::
at:!::0:::::
squid:!::0:::::
xfs:!::0:::::
games:!::0:::::
cyrus:!::0:::::
vpopmail:!::0:::::
ntp:!::0:::::
smmsp:!::0:::::
guest:!::0:::::
nobody:!::0:::::
chrony:!:19609:0:99999:7:::
emmettbrown:$6$DA49lDfc5Ls6//UP$vEK//h5a5.y4wKiFdR/4GU.vbV13eoVVXitbkGttWrAIQ4bSmmDg4T.BkPtNp
B3SY.viNKKnqzfFafrzhqFX4.:19609:0:99999:7:::
```

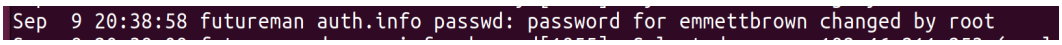
Caption

From the above output we can observe that the other user is emmetbrown. We can observe the password hash followed by the number of days since password change. It is to be noted that the number 19609 represents the number of days after Jan 1 1970 (unix time) since the password has been changed.

2. Password Check

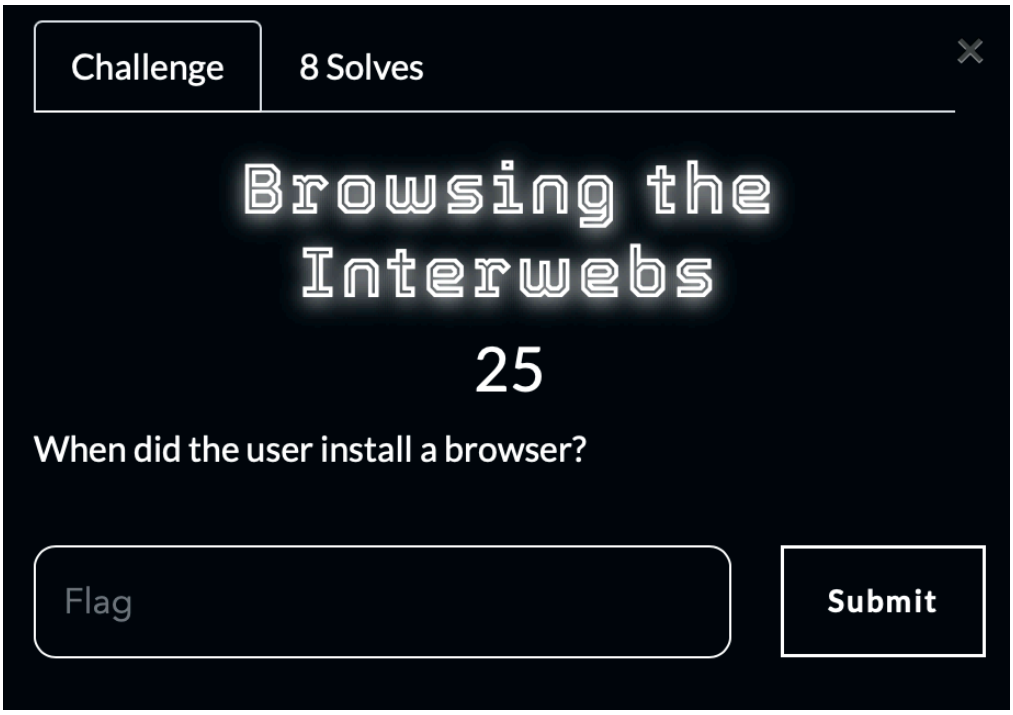


In this question we are tempted to believe that the flag is epoch time of (1 Jan 1970) + 19609. But this is incorrect as the data stored in the shadow file is approximate and not accurate because it doesn't take hrs and seconds into consideration. So in order to find out the exact time the password was changed we check the log, after a lot of searching I found out that log data is stored in /var/log/messages. When I used the command cat messages I got :



Which should be converted to epoch time using online converter to obtain the final flag.

3. Browsing the Interwebs

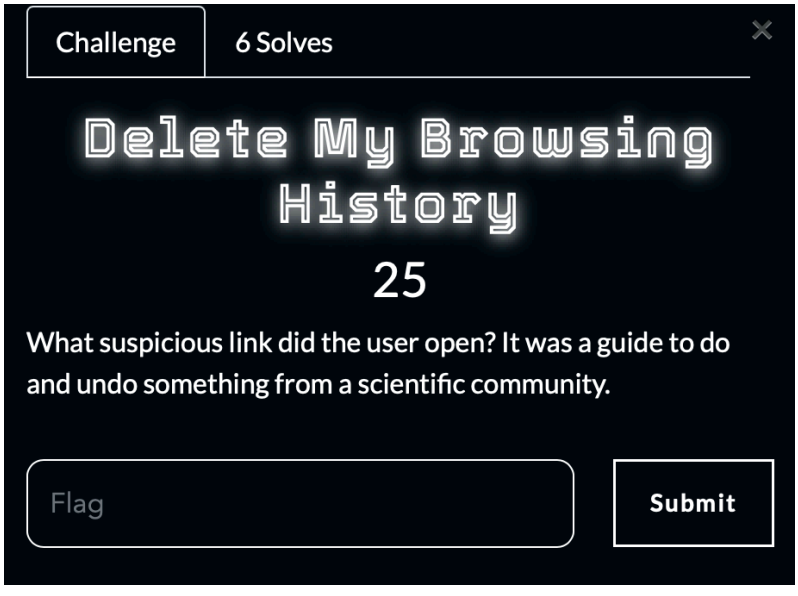


This question is similar to the previous question where we checked the log for the timestamp. Coping the content of messages to messages.txt and the using grep to search for browser/firefox has proved to be a successful method to obtain the timestamp.

```
parallels@ubuntu-linux-22-04-desktop:~/Downloads/arjun/var/log$ grep firefox messages.txt
Sep  9 21:19:51 futureman authpriv.info : emmettbrown ran command apk add firefox as root fro
m /home/emmettbrown
```

After obtaining the timestamp we can use a online converter to convert Sep 9 21:19:51 to epoch time which yields us the final flag.

4. Delete My Browsing History



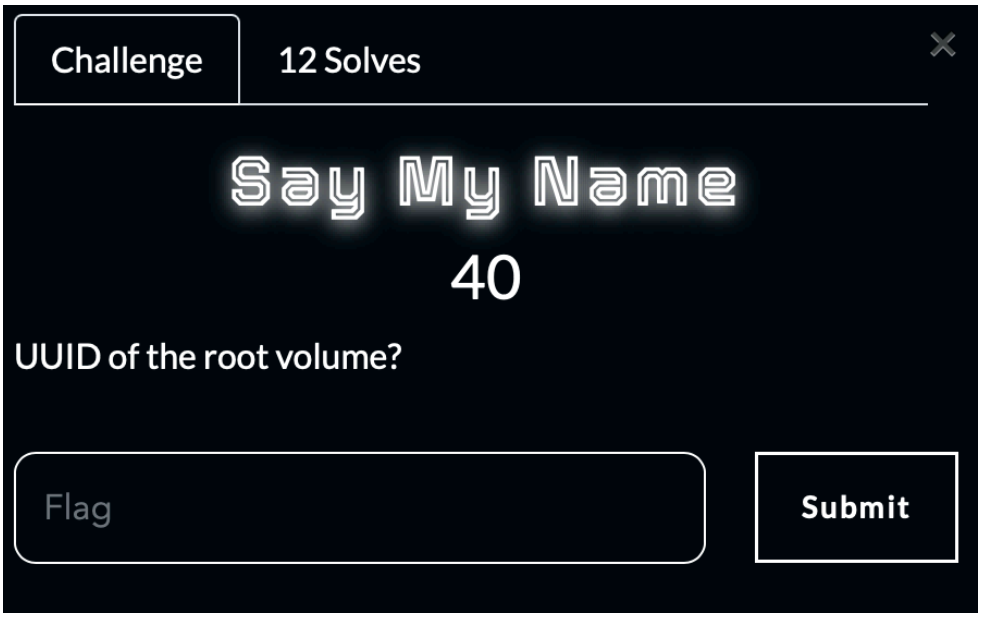
Browsing History is usually stored in users/.mozilla this Mozilla folder is hidden and it is .mozilla as the default browser in linux is Mozilla Firefox. So with that knowledge we can access the hidden ./mozilla folder and searching all folders inside ./mozilla we can find that the the browsing history is stored in places.sqlite which is a SQL database file. With the linux command sqlite3 and the knowledge of basic SQLITE commands we can access the database file.

```
parallels@ubuntu-linux-22-04-desktop:~/Downloads/arjun/home/emmettbrown/.mozilla/firefox/slao
wkrn.default-release$ sqlite3 places.sqlite
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.
sqlite> .tables
moz_anno_attributes                 moz_keywords
moz_annos                           moz_meta
moz_bookmarks                       moz_origins
moz_bookmarks_deleted               moz_places
moz_historyvisits                   moz_places_metadata
moz_inputhistory                    moz_places_metadata_search_queries
moz_items_annos                     moz_previews_tombstones
sqlite> SELECT * FROM moz_places;
```

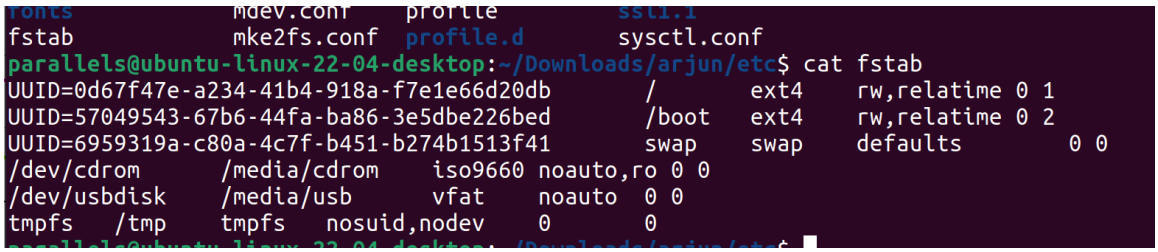
```
13|http://www.home.cern/totally-legit-time-travel-guide||nrec.emoh.www.|1|1|1|2000|1694294506
472786|cppMk377mRiv|0|125508316499178||||5|0||1
```

From the above table we can easily guess that the only URL which is related to scientific community. We can also observe the timestamp in epoch microsecond that is 1694294506472786, which can be converted to seconds to obtain the final flag.

5. SAY MY NAME



In this problem we are asked to find out the UUID (Unique Universal Identifier) of the root volume. Usually storage details are stored in swap or etc folder. On close inspection and trial and errors we can find that the UUID of the root storage including those of swap and boot are stored in the stab file.



From the above image we can find the UUID of the root storage '/', which in this case is the final flag.