***PicoCTF Challenges***

-Ishan Surana

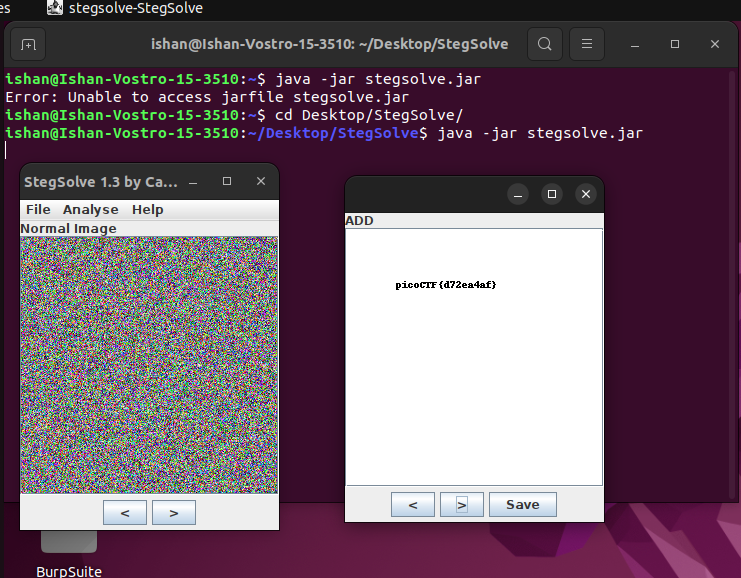
Challenge 8:- **Pixelated**

This was more of a steganography challenge. I had some idea of steganography but not much, so I obviously couldn’t get much on my own. I tried to overlap images online, because I assumed stacking meant image over image with 50% transparency would give me the hidden text, but to no avail :(.

So I searched online on how to solve this problem and found two methods. One of them used python to combine the images, but since I am not very experienced in python, I didn’t use this method (as it would be just me copying the code; better learn python some more before reading the code again).

The 2nd method involved downloading quite a handy tool called StegSolve. This allows many operations on images, one of which was Image Combining. Combining the images gives you many types of combinations (AND, XOR, etc.) by clicking the right arrows. The correct one was ADD, as we stacking meant we basically had to add the bytes information on each pixel of the 2 images which were seperated, to get the original combined image.

Take note to remove the space between “F” of picoCTF and the “{“.



Challenge 10:- **Tunnel Vision**

This was my first challenge in forensics, and it was very tough (but still fun to learn). I had almost gave up on the problem due to failing so many times, but I still finally completed it.

This involved using a hex editor, which I had no idea about so I watched videos on YouTube how to solve this problem. I watched the whole process and then tried myself, but couldn’t get it to work. For the first part I had to set the values of parts which were saying BA D0 to numerical values as accoding to the standard format of bitmap images. I wasn’t able to get it to open even after that and this was because I was inputting wrong values (out of bounds).

[PLEASE NOTE that edited values must be in a certain range for image to be displayed correctly. Hence, thereafter I used the same values as the video]

After that I found the cropped out image which didn’t have a flag. So, I saw rest of the video to get to the location having parameters about the image length and height and edited that too [Here too, use video values to not go out of bounds].

Hence, I uncovered the complete image and got the flag.

Video I referred:- https://www.youtube.com/watch?v=X4kJiQdDn7M&list=PLJ\_vkrXdcgH\_m6ADxGVLj7iy9ttnjaq7G&index=20

[This was one of the many videos I watched, but exact values are taken from here]

Hex editor:- <https://hexed.it/>

Challenge 11:- **Wireshark**

This challenge introduced me to captured packets, what they are and how to view them. I watched a video which was very insightful; he cleared the level along with teaching what all was happening.

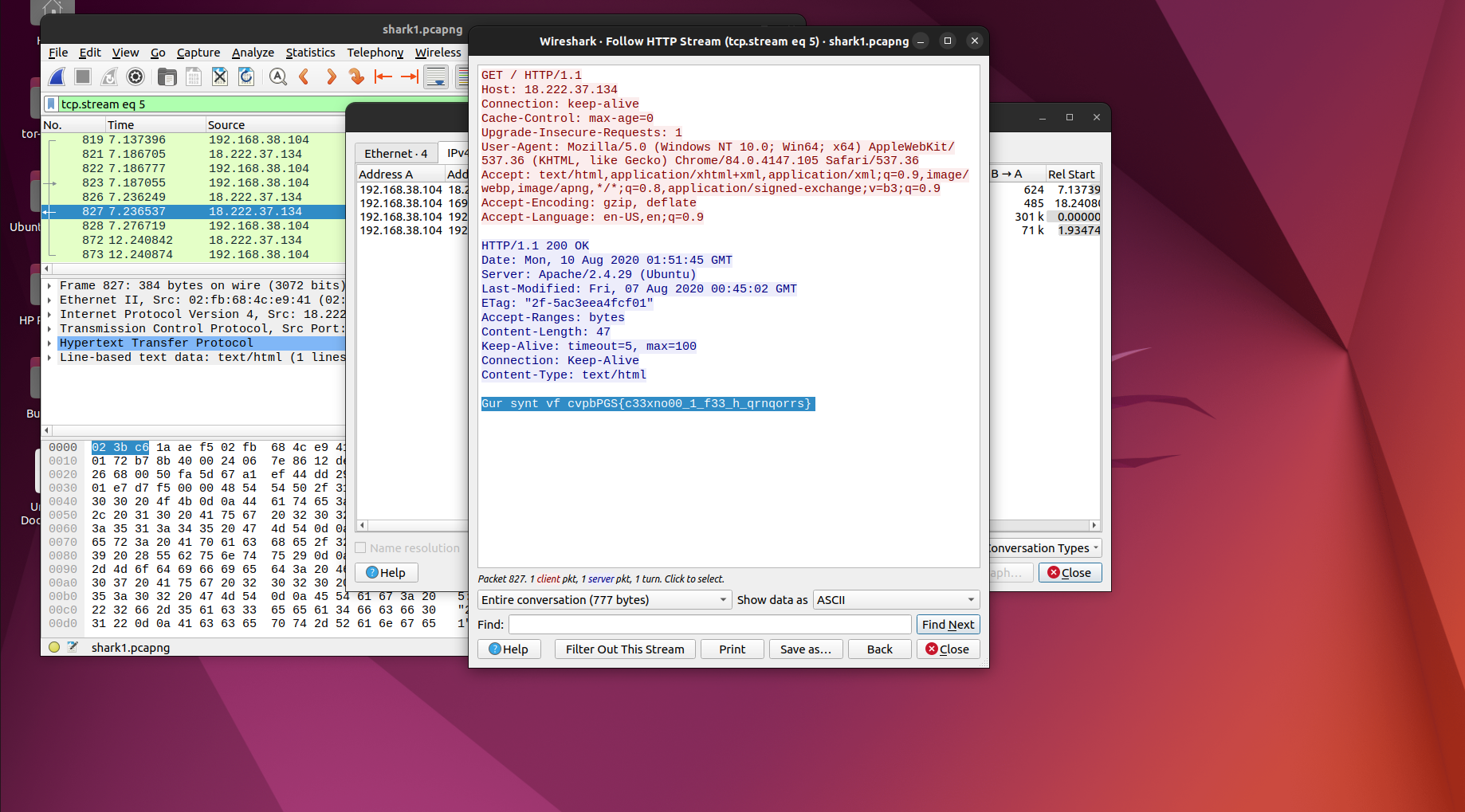
Basically, pcap files are packet captures which are essentially records of data transferes happening. We required to extract the data exhanged (flag) from the pcapng file using Wireshark, which is a pcap file viewer.

Seeing the IPV4 conversations and then reading a chosen exhange, we see the

sync requests and acknowledgements at first, and then the actual data transfers.

We can compile the furthur exhanges into and HTML, where we can see a text line in similar format to our flag. Analysing it a little with known characters like

picoCTF reveals it to be ROT13, and thus we can find out the correct flag.



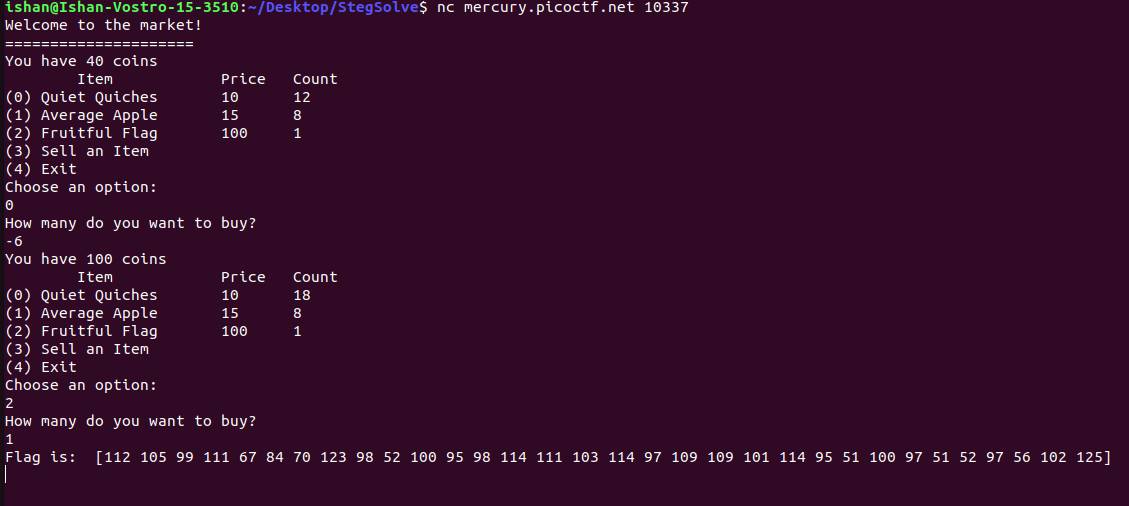
Challenge 12:- **Shop**

This challenge was the first one (in the list) that I have solved on my own (without watching any videos)(I have solved others on on my own but they are not in the list).

It involved me breaking the code, but unfortunately the source code I downloaded wasn’t running. But when I experimented around with the program on the nc link a little, I understood the basics of what the code might be. Basically, it was keeping track of my purchases and sales of the items, and displaying my possessions (number of items) and money. Giving it a condition which was more than possible (ie buying items worth more than my money possesed, or asking items more than what the seller can provide, or me selling more than I have) leads it to display error messages.

However, in checking the more than condition, the code fails to neglect the practicality by not focusing upon the less than condition. Hence, I can buy or sell a negative number of items and it will pwerform the arithmatic operations on it as usual (even though there is practically nothing such as buying/selling a negative number of items).

Therefore, to get money to buy the flag, I first bought a negative number of other items (which was equivalent to selling a number of that items which I don’t have, but when checked against 0, a negative number is lesser, hence condition fulfilled). Therefore, I got the required money and bought a fruitful flag (which was in ascii and converted to required flag afterwards).

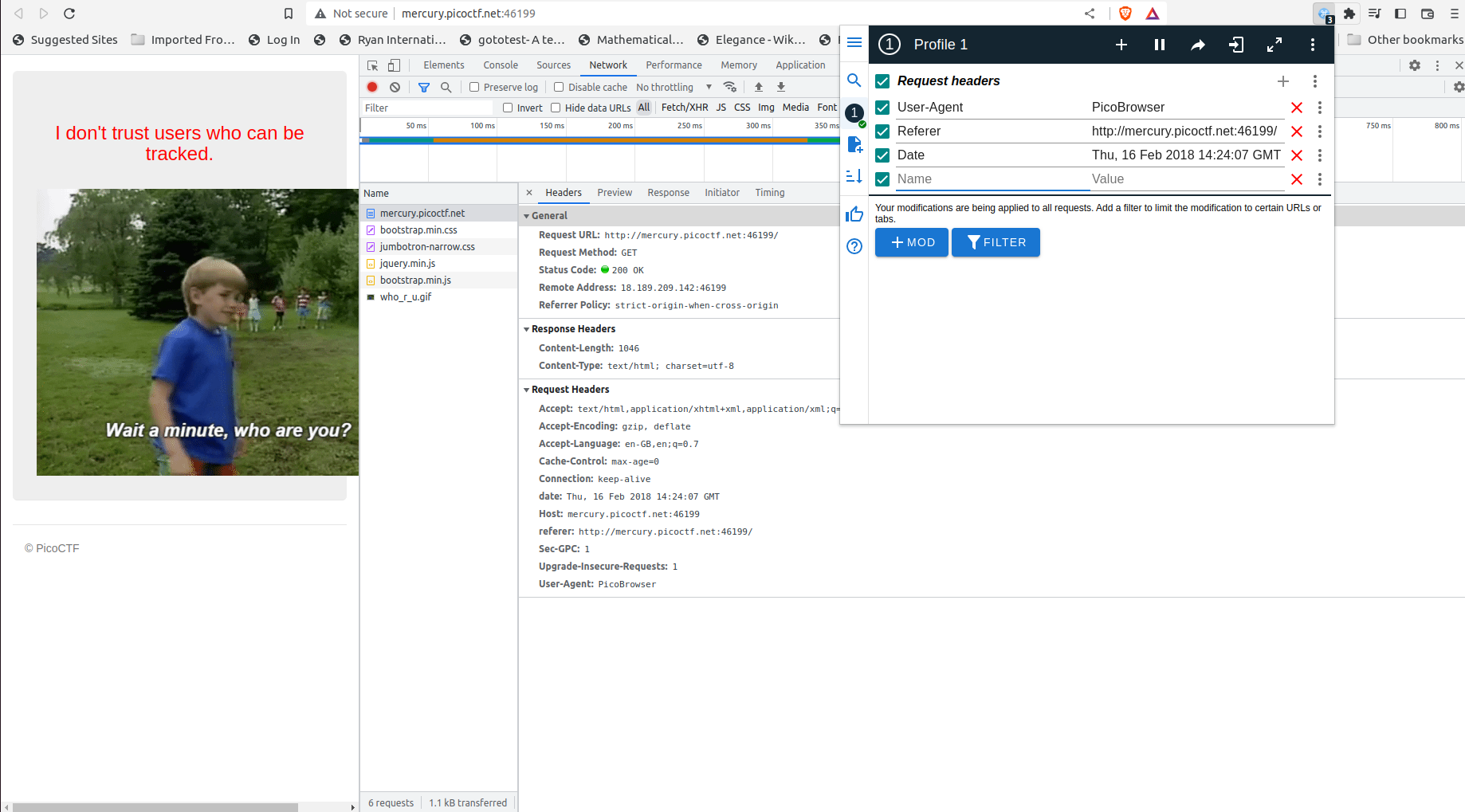


Challenge 13:- **Who are you?**

This is one of the challenges not in the list, but had to be solved in order to understand one of the challenges which was in the list. This challenge introduced me to HTTP headers, and since I knew almost nothing about them, I took the help of a video. It was an extremely detailed video with complete knowledge about what to do and what is happening.

So, HTTP headers are the information the site receives and uses to complete the connection request receives. It contains information about various things such as browser used, location, date & time (when the connection was requested), IP address, languages used, etc. These can also be viewed using inspect element but not edited.

However, there is an extension in Chrome and Chrome-based browsers (such as Brave, the one I was using) called ModHeader,which as the name suggests, modifies the Header value to what we want it to and sends the connection request.

So, I first analysed the site and traffic using BurpSuite (not needed, but the video creator was using it while teaching and I needed to learn as well for later use), then installed ModHeader. I firsed modified the browser name, which is inferred from the tag User-Agent, and after refreshing the error message changed. This required me to modify another header, and so on until all error messages were removed and I finally received the flag. A thing to note is that I didn’t know the header names, so it can be studied in the Mozilla developers site, where a list along with description of headers is given (so read the whole list anyways or you can also narrow descriptions having specific keyword).

Challenge 14:- **picobrowser**

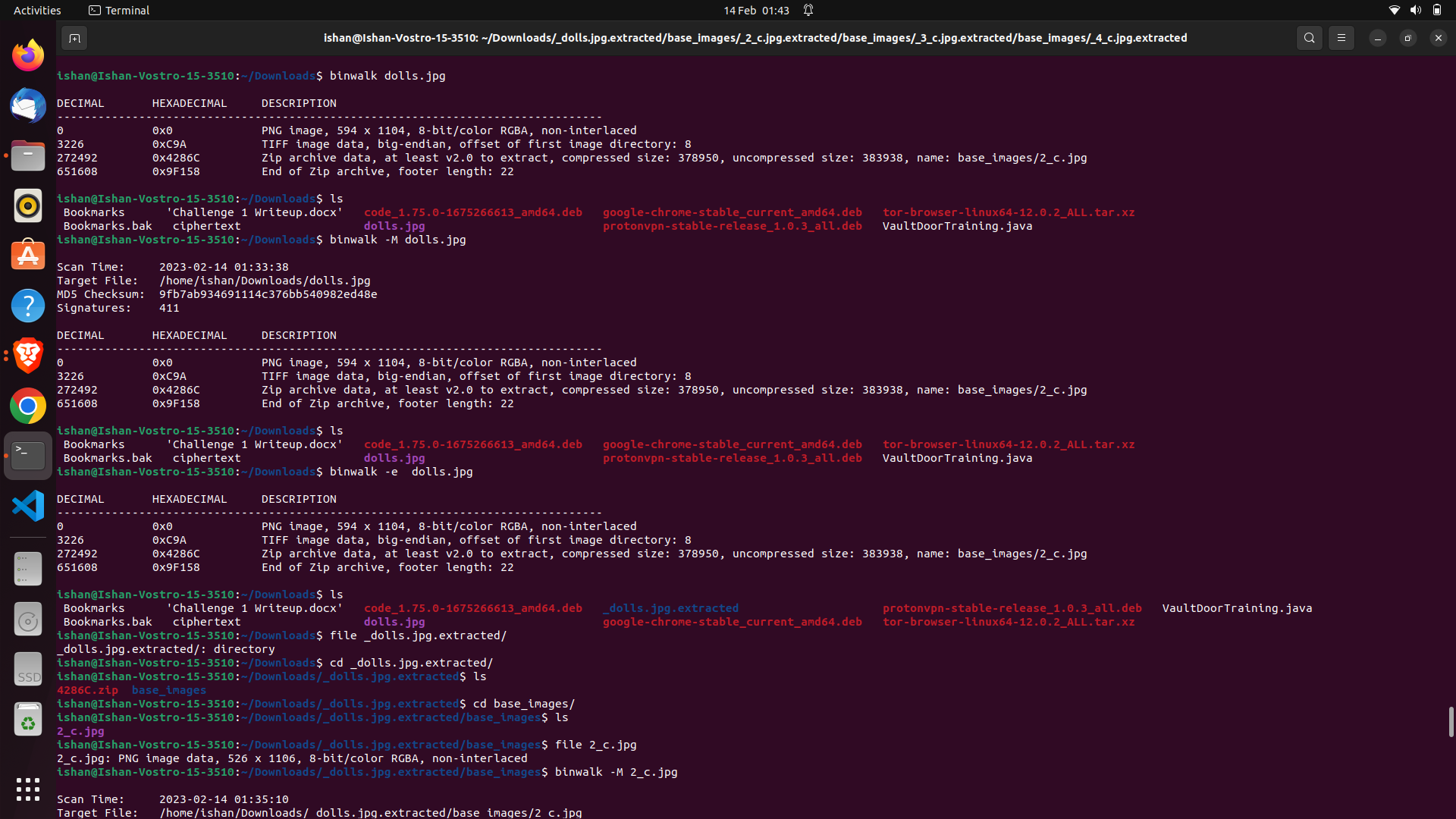
This challenge is a piece of cake once you have understood what was happening in the previous challenge. Here, only one header is causing issue (name of the browser, ie tag User-Agent).

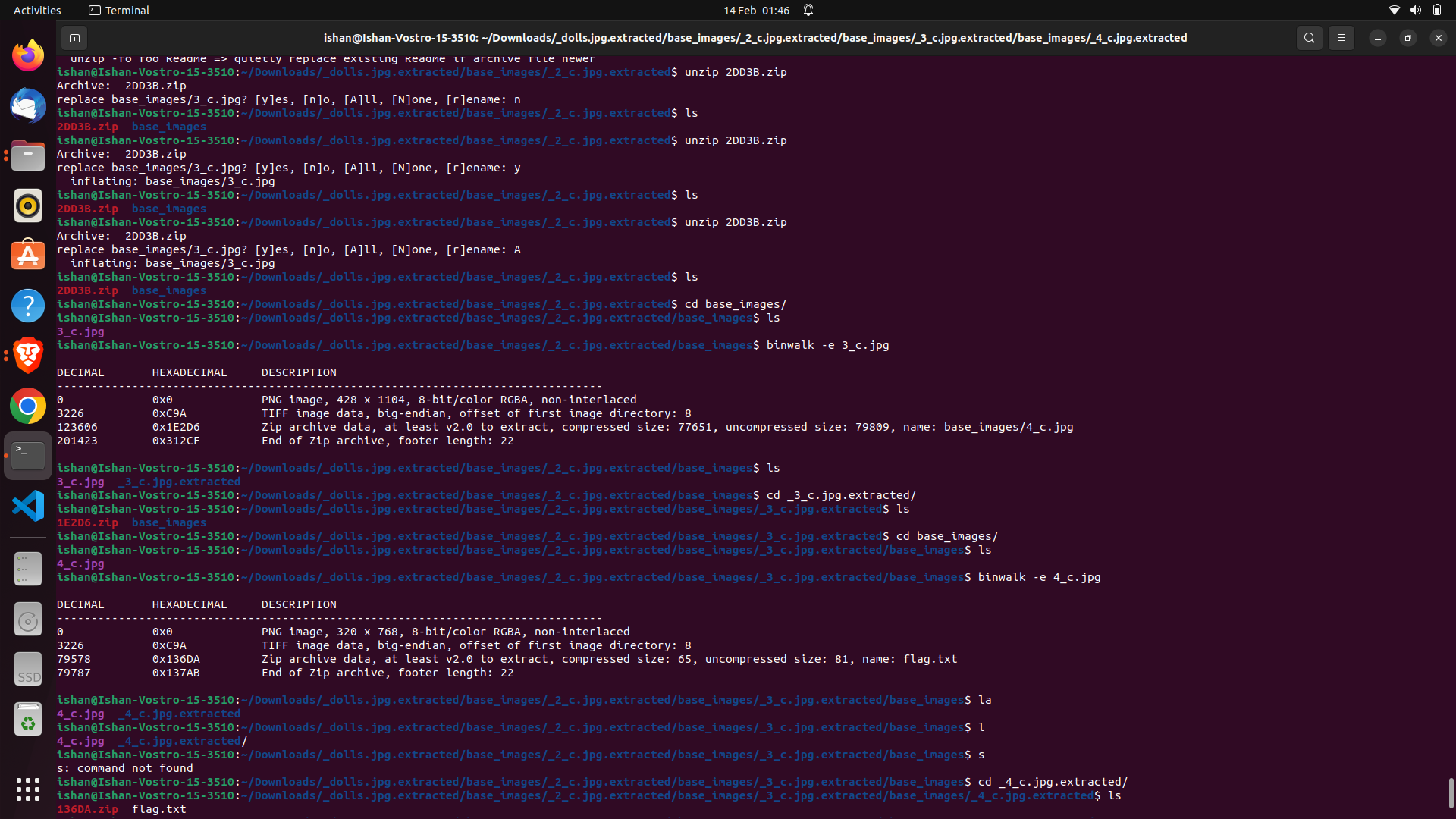
Simply use ModHeader to set the value of User-Agent to picobrowser and refresh the page. You can then get the flag.

NOTE:- Enter the correct spelling of picobrowser, case-sensitive.

Challenge 15:- **Matryoshka**

This challnege was done as a part of learning from an event happening (Aurora Tech Week, event CTF). The organisers there explained about a Matryoshka file and usage of “binwalk”. It searches for and extractsembedded files in an image. Reading the man page helped find the required tags for embedded file viewing and extraction, and after a series of extractions (like a Matryoshka doll), I got to the text file containing the flag.





Challenge 16:- **Vaultdoor 4**

Revisiting on the vaultdoor challenge series, this one was also very easy. The source code had the password itself again, but this time in 4 parts and in 4 different formats. The first part was in integer (decimal) values, 2nd part in hexadecimal and 3rd in octal format. I converted them using CyberChef, a superb tool with a whole lot of tools for conversion or encryption/decryption. Combine all parts (converted to ascii) and wrap in picoCTF{} format.

Challenge 17:- **Vaultdoor 5**

This challenge was also easy, but introduced me to URL conversion (an addition to the decimal, binary, ascii, octadecimal, hexadecimal & base64 that I previously knew about). This challenge had a hint which included yet another awesome site for data type conversion (https://encoding.tools), where you can attack multiple encryption/decryption parts by dragging them onto the workspace and drawing lines between them.

The source code again provided the encryptions happening and therefore I attached the 2 decryptors (url then base64) and got the flag.

