PicoCTF

JAVA SCRIPT KIDDIE

HURRY UP! WAIT!

The only thing in the description of this CTF is the download link to a file.

When you use the ‘file’ command to determine the type of file it is, we find that it is an ELF type.

ELF here, stands for Executable and Linkable Format of a file, which is usually the output of a compiler.

LSB here stands for Least Significant Byte, which implies that the file is Little-Endian. This is determined by the argument ‘file’ from the sixth byte of the header:

>5 byte 1 LSB (source in references)

References:

For ELF header:

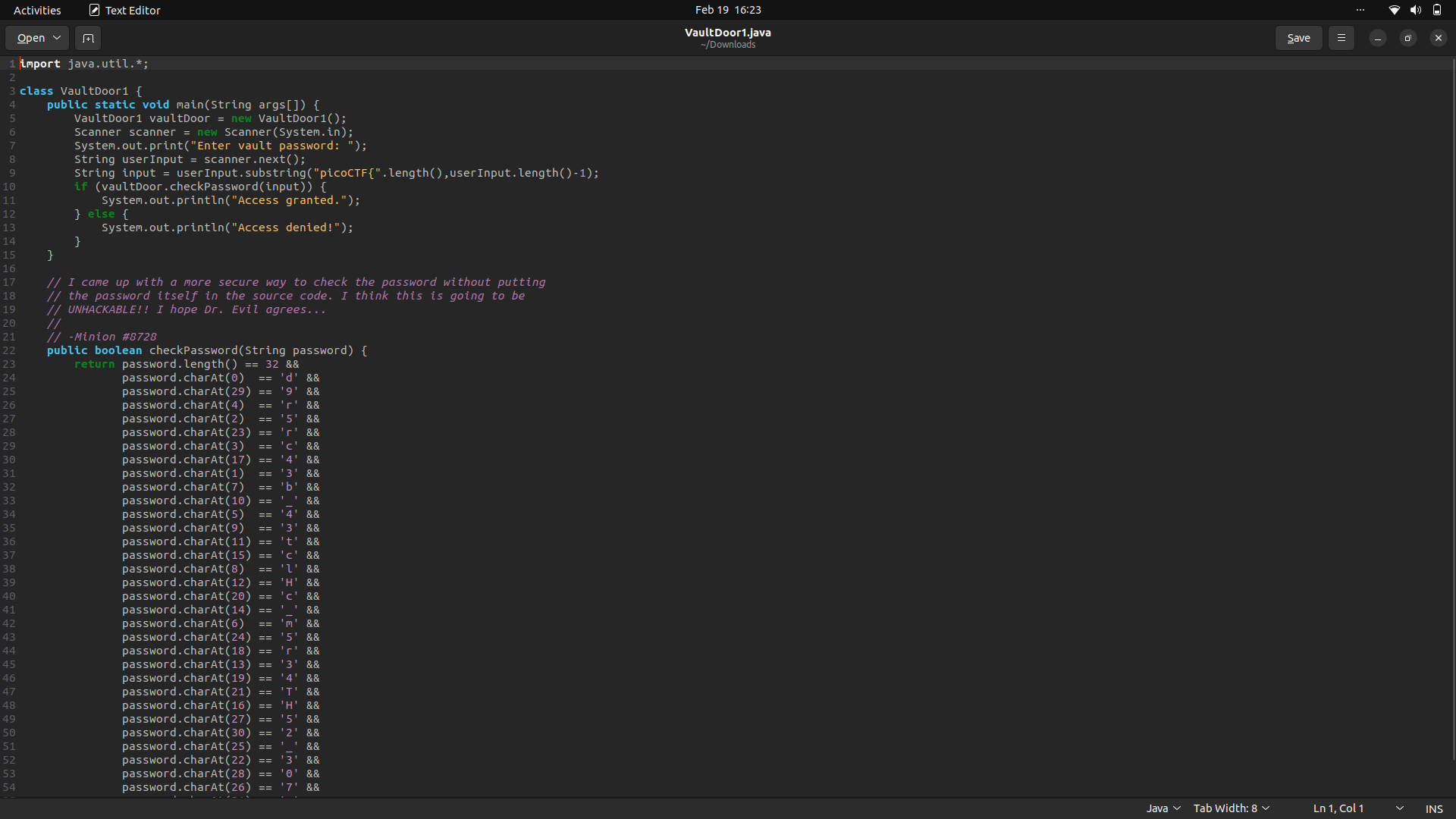
<https://github.com/file/file/blob/4264364d4a46d632ceb095e8cef56339f592931d/magic/Magdir/elf#L305>

<https://en.wikipedia.org/wiki/Executable_and_Linkable_Format#File_header>

For ELF:

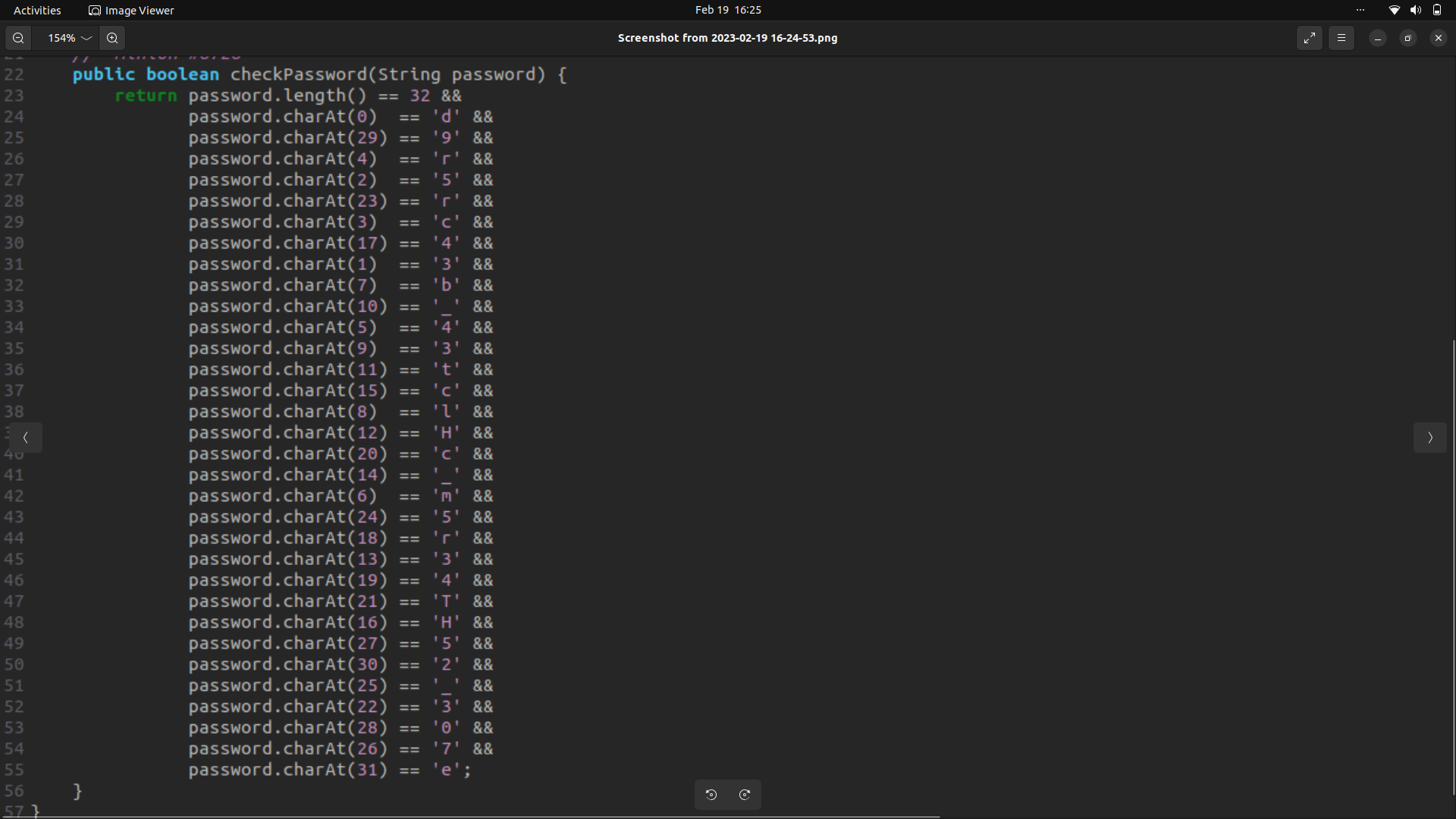
<https://linux-audit.com/elf-binaries-on-linux-understanding-and-analysis/#what-is-an-elf-file>

VAULT DOOR-1



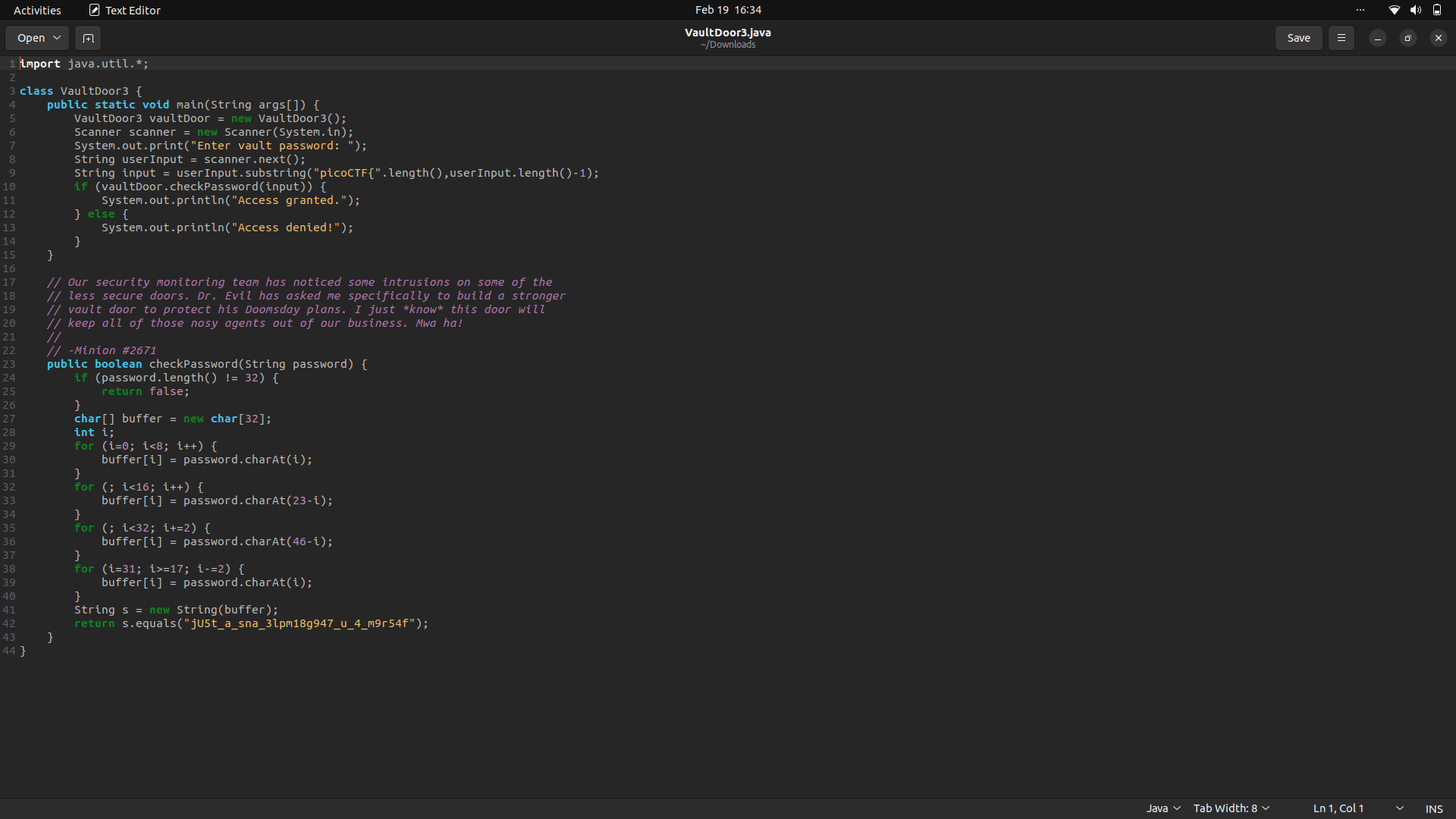
As can be seen upon downloading the source code, this program makes use of the charAt function, which returns the index position of any specified character in an array.

Thus, if we simply arrange the array from the indices given, we get the flag:



picoCTF{d35cr4mbl3\_tH3\_cH4r4cT3r5\_75092e}

VAULT-DOOR-3

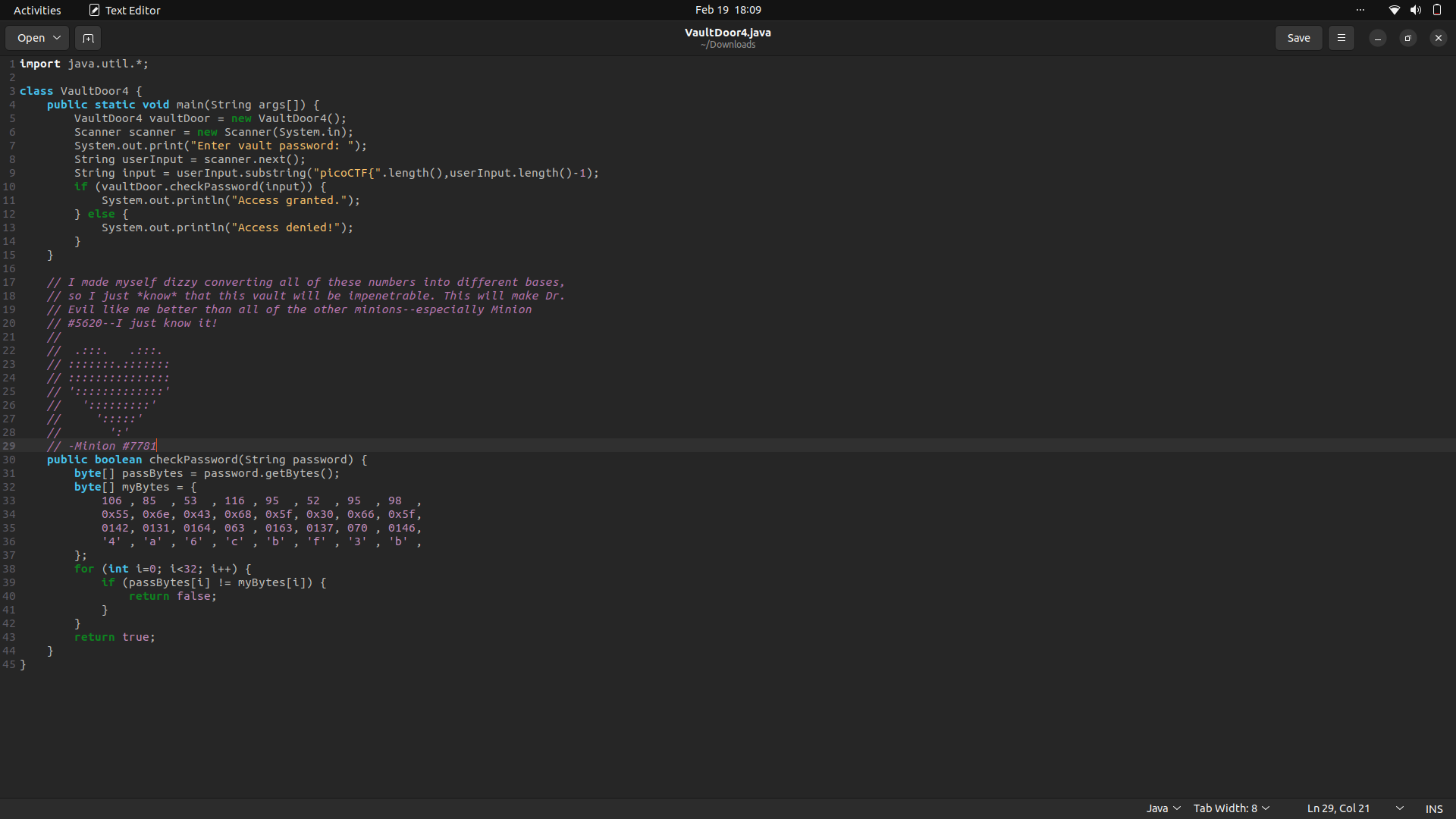


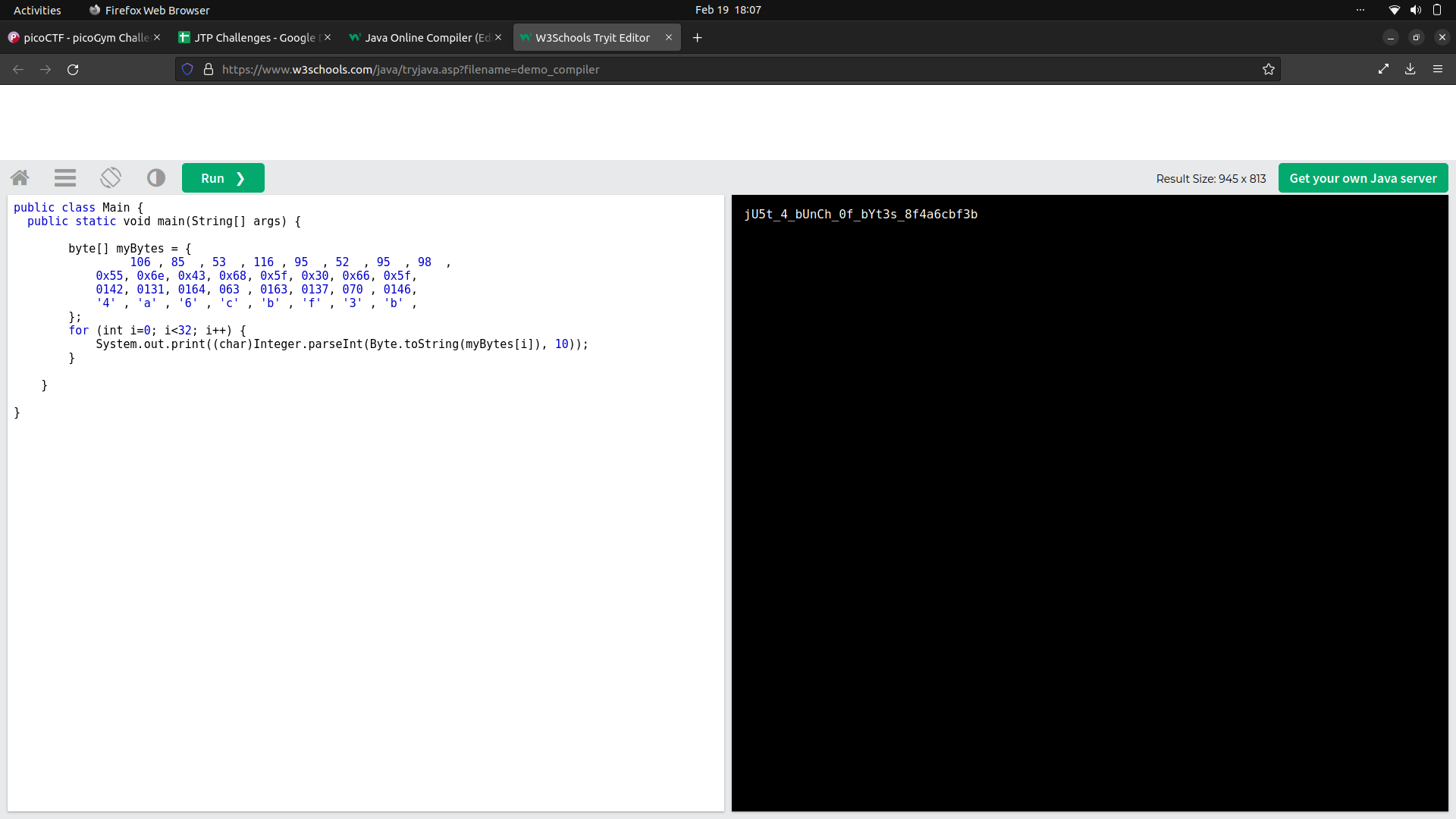
As can be observed, the program compares the value of the input string with the characters at particular indices of the buffer string. Simply grouping all the required characters from the buffer string in the right order gives us the flag:

picoCTF{jU5t\_a\_s1mpl3\_an4gr4m\_4\_u\_c79a21}

VAULT-DOOR 4

All we need to do is convert bytes to strings of base 10, and then convert it further to ASCII characters. This can be done using the function Byte.toString( ) for the first one and Integer.parseInt( ) for the second conversion.



Resources:

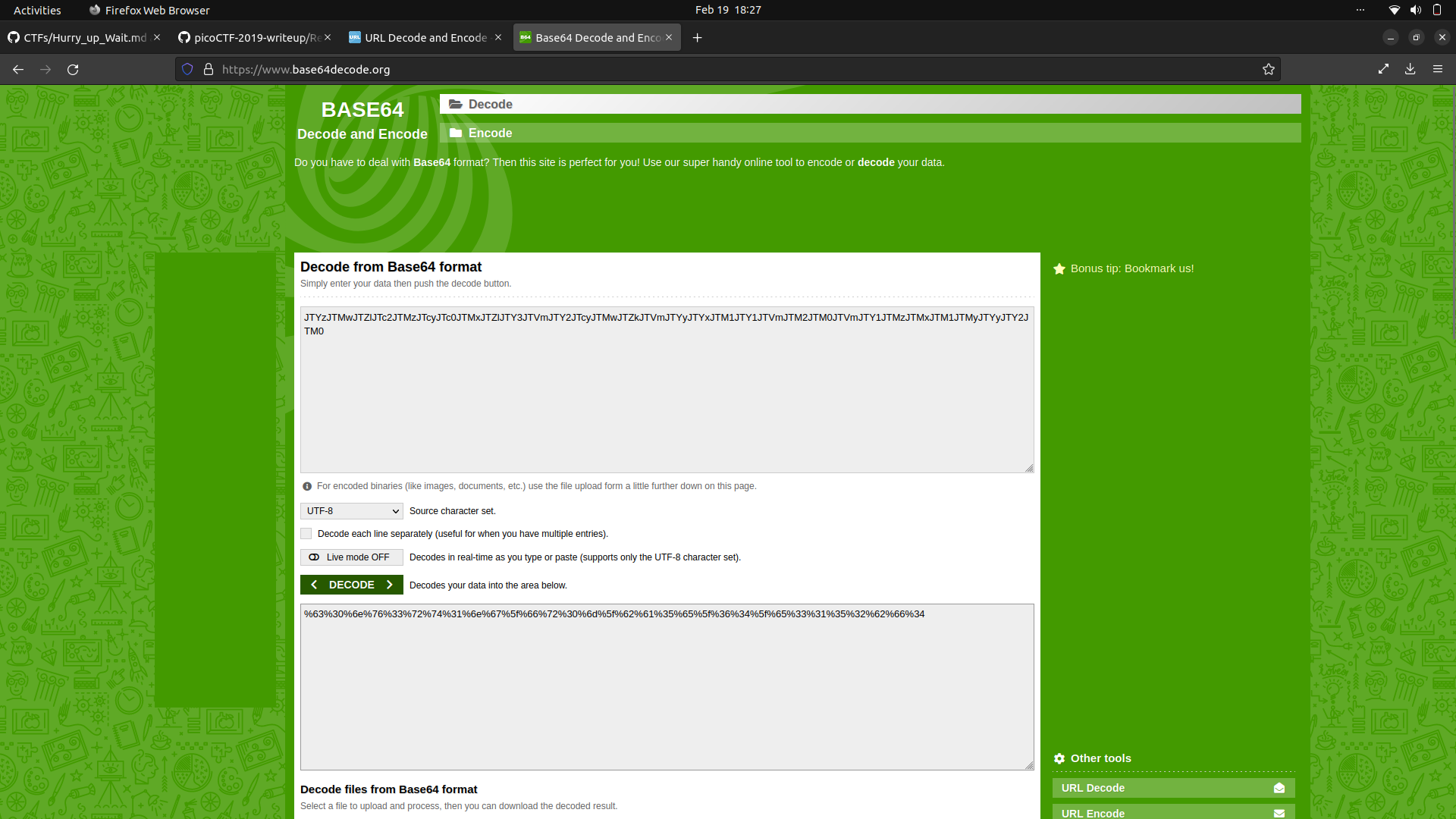
<https://www.geeksforgeeks.org/byte-tostring-method-in-java-with-examples/>

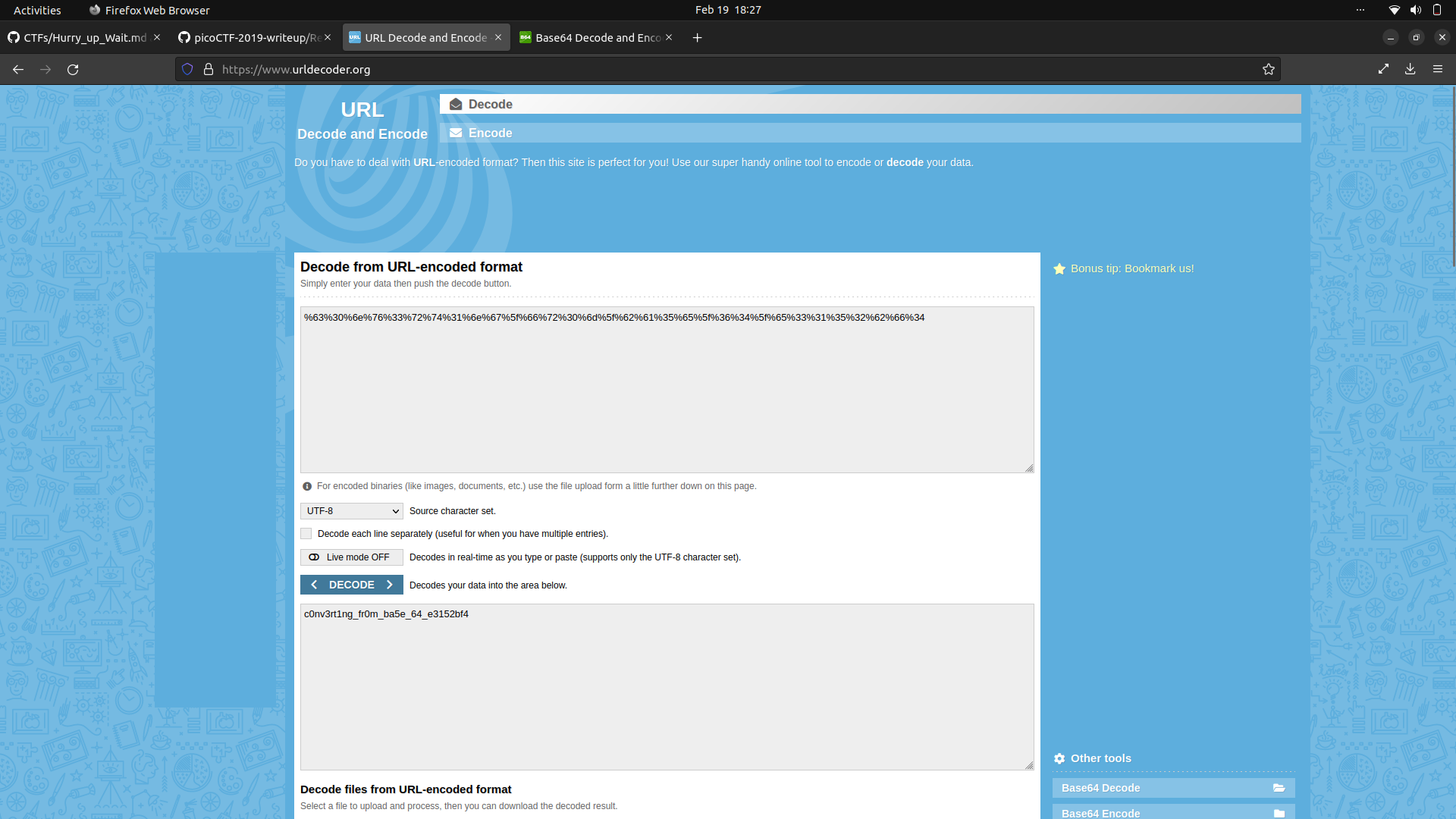
VAULT-DOOR-5

As we can observe in the source code, the string is first encoded by base64 and then through an URL encoder.

While both are forms of encoding an array of characters, base64 converts binary to a corresponding number in base 64, while URL encoding uses only the characters from ASCII which are permitted in an URL to encode it.

We can simply use and online decoder for decoding the string to get our flag.





picoCTF{c0nv3rt1ng\_fr0m\_ba5e\_64\_e3152bf4}

References:

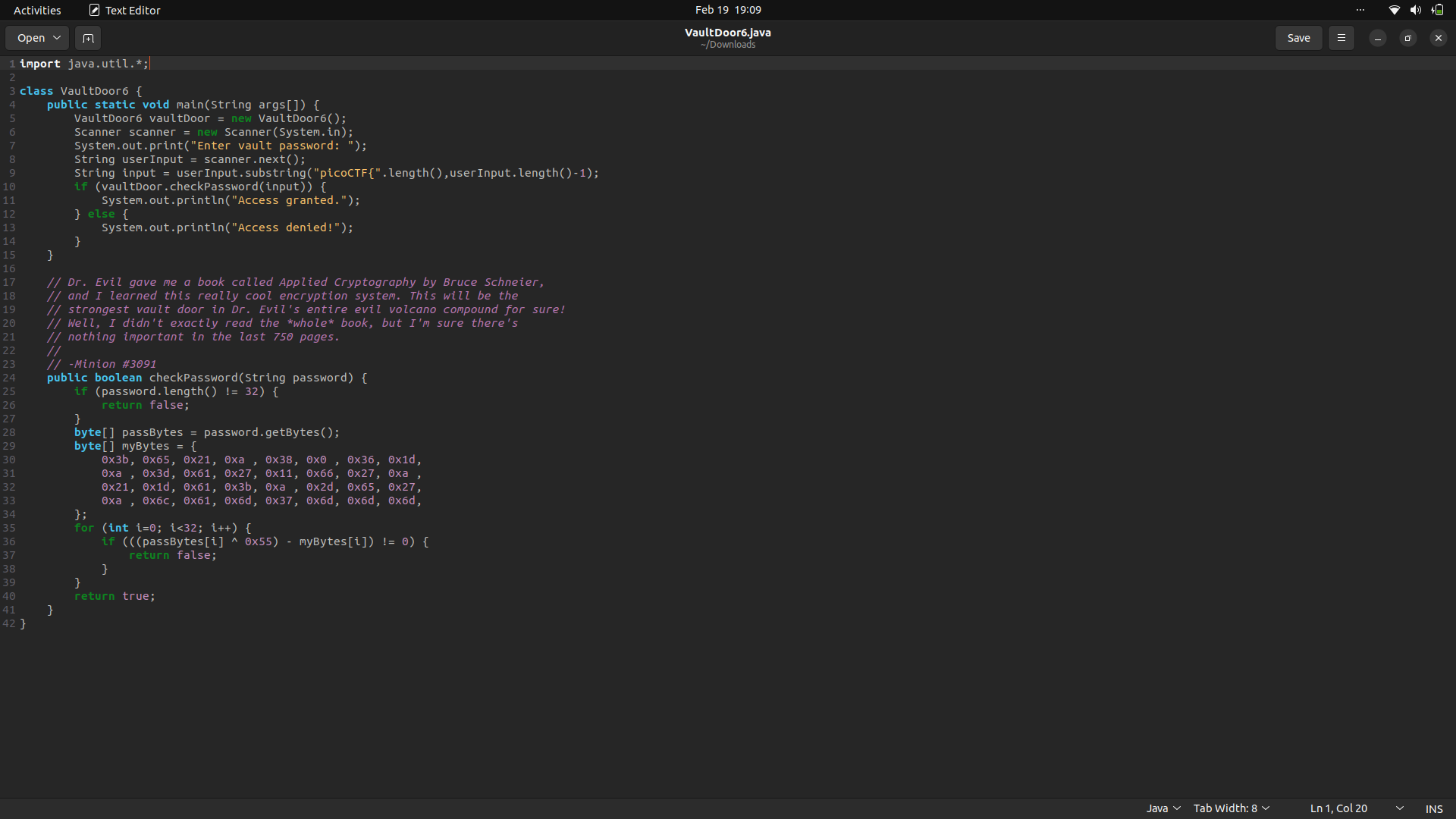
Base64:

<https://en.wikipedia.org/wiki/Base64>

URL Encoding:

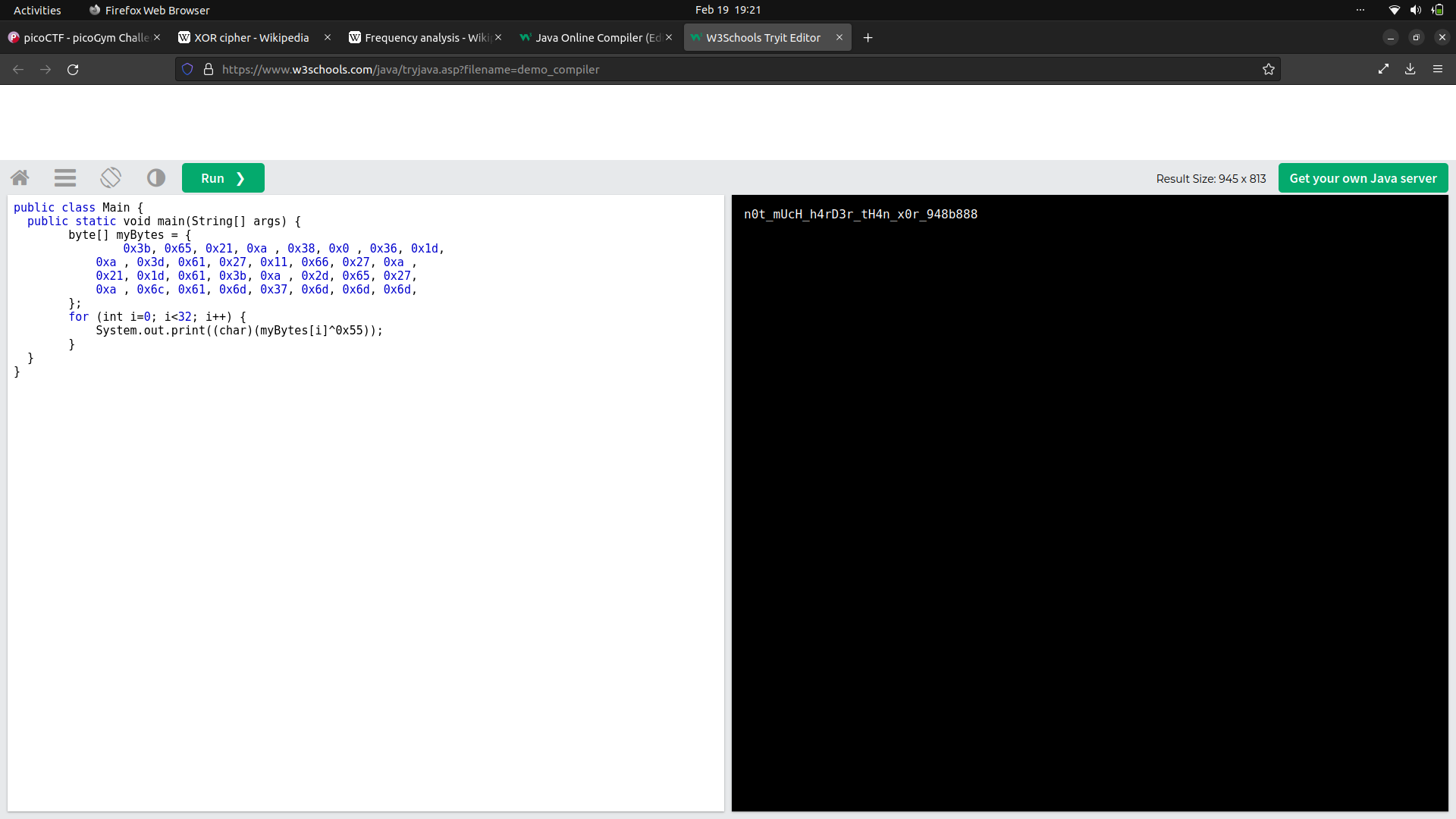
<https://en.wikipedia.org/wiki/URL_encoding>

VAULT-DOOR 6

As can be observed, we first need to convert bytes to string and for convenience, we’ll be using the same program we used in a previous CTF, altered slightly such that we will XOR the output of the string first before printing. We could alternatively also use an online tool to convert the string which has not been decoded by XOR.

We know that each byte is equal to each byte in the string XOR 0x55 as can be seen in the line from the code:

“if (((passBytes[i] ^ 0x55) - myBytes[i]) != 0)”

The flag is:

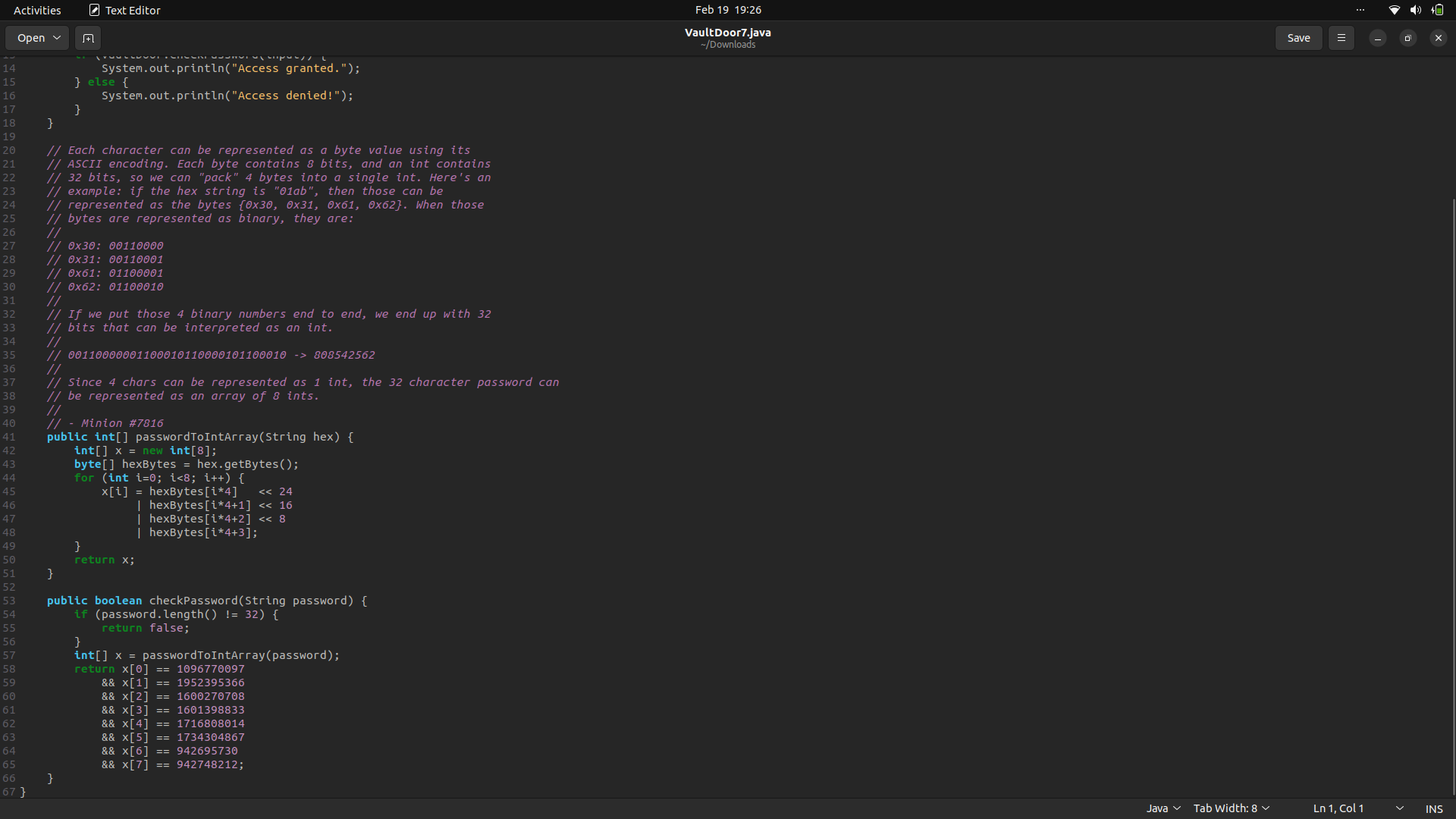
picoCTF{n0t\_mUcH\_h4rD3r\_tH4n\_x0r\_948b888}

Resources and References:

XOR: <https://en.wikipedia.org/wiki/XOR_cipher>

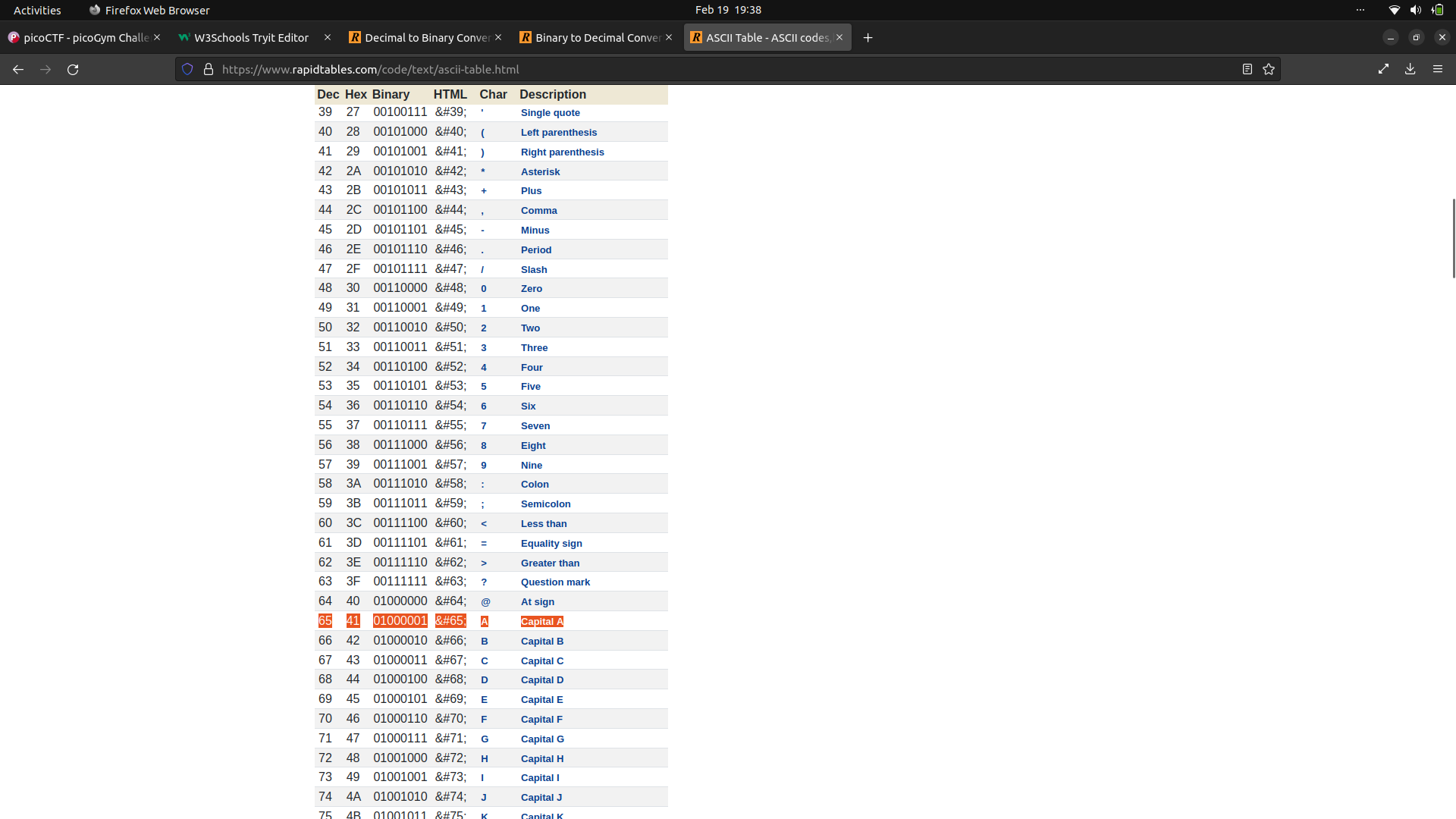
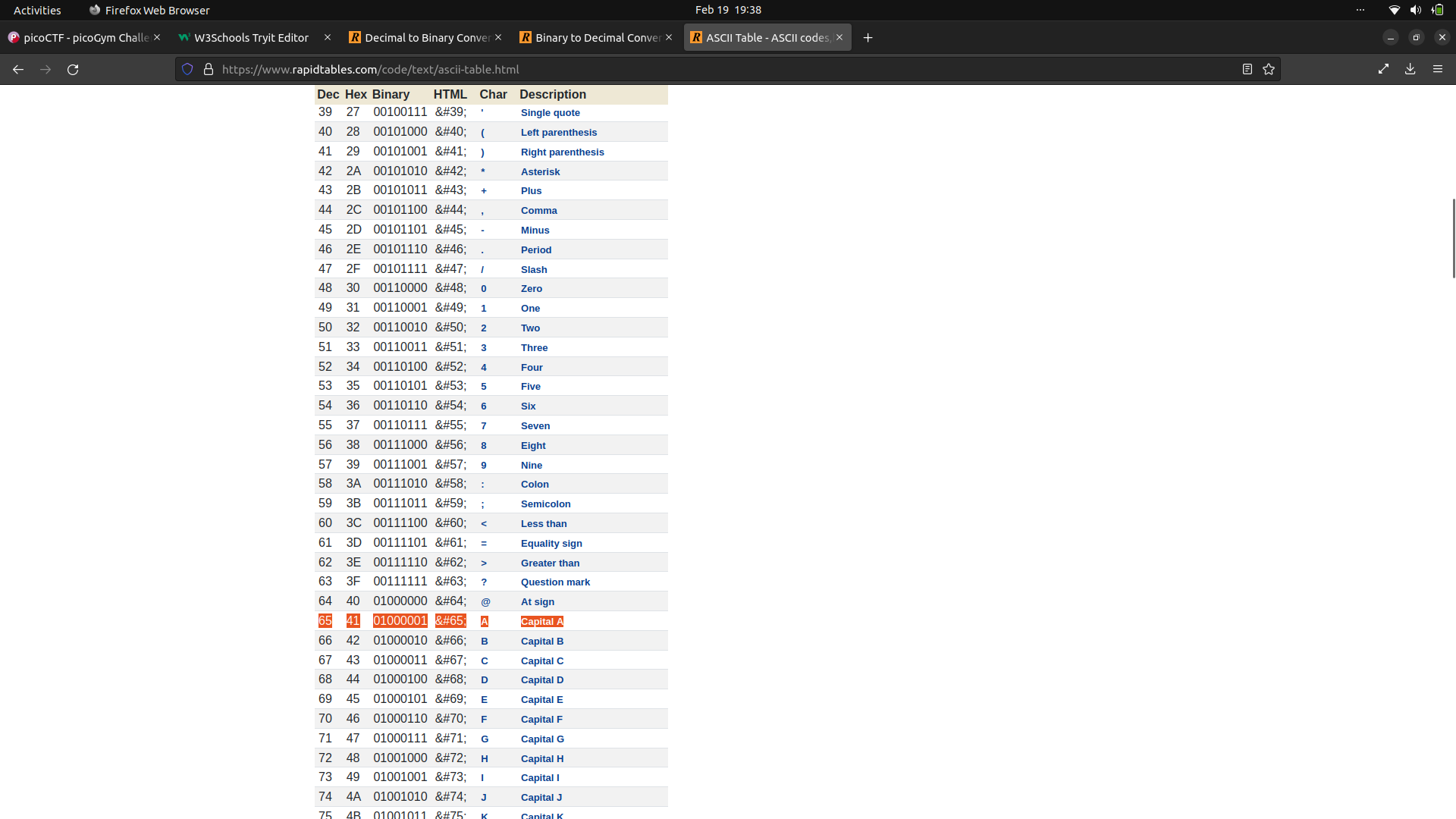
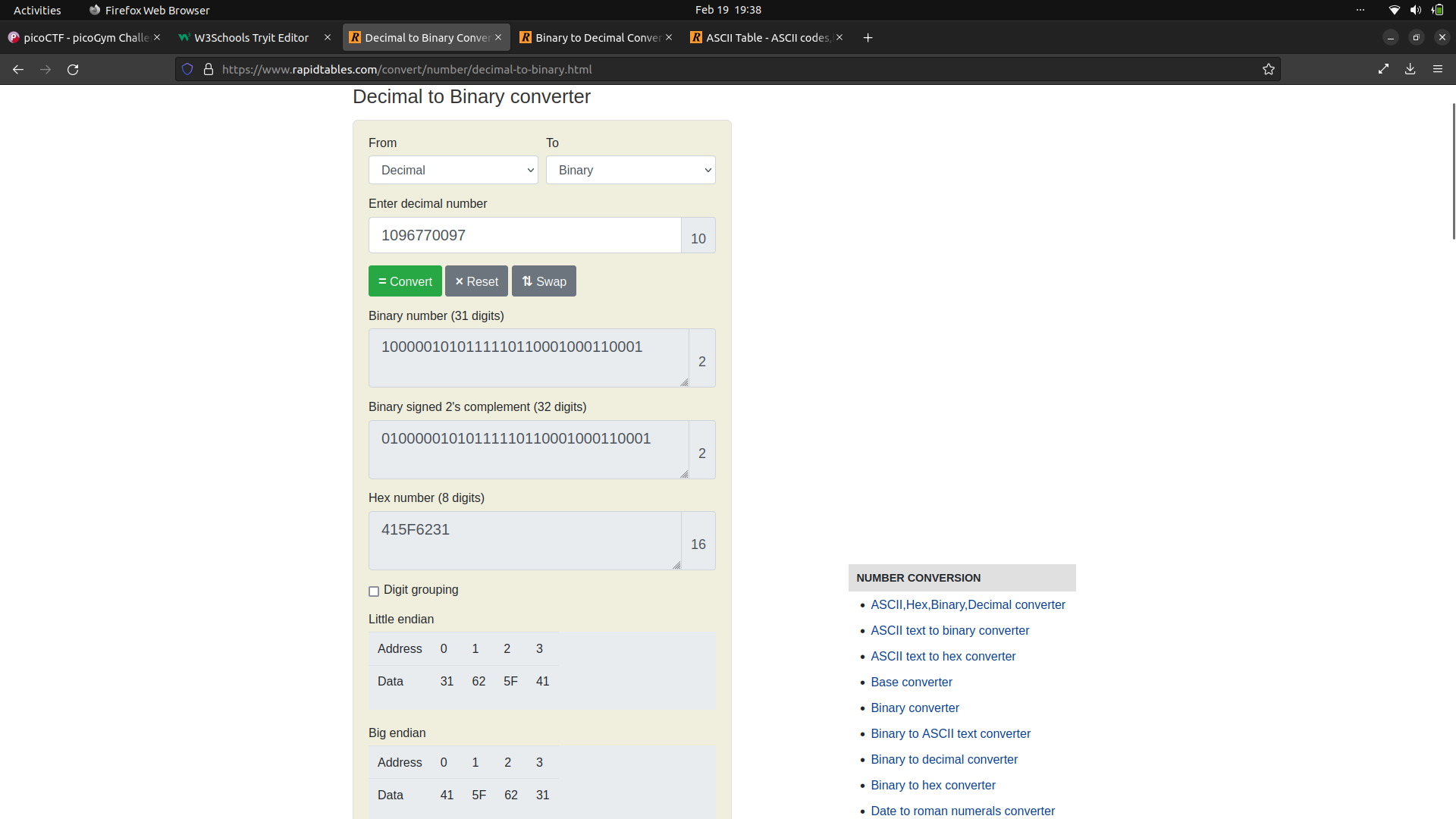
Online Java Editor and Compiler: <https://www.w3schools.com/java/tryjava.asp?filename=demo_compiler>

VAULT-DOOR 7



The comment helps us understand what needs to be done in this challenge.

We first need to convert given integers to binary which is then to be broken down into four 8-bits long sequences. Each such sequence represents a byte which represents an integer which represents an ASCII character. This string of ASCII characters is our flag.

The flag is: picoCTF{A\_b1t\_0f\_b1t\_sh1fTiNg\_dc80e28124}

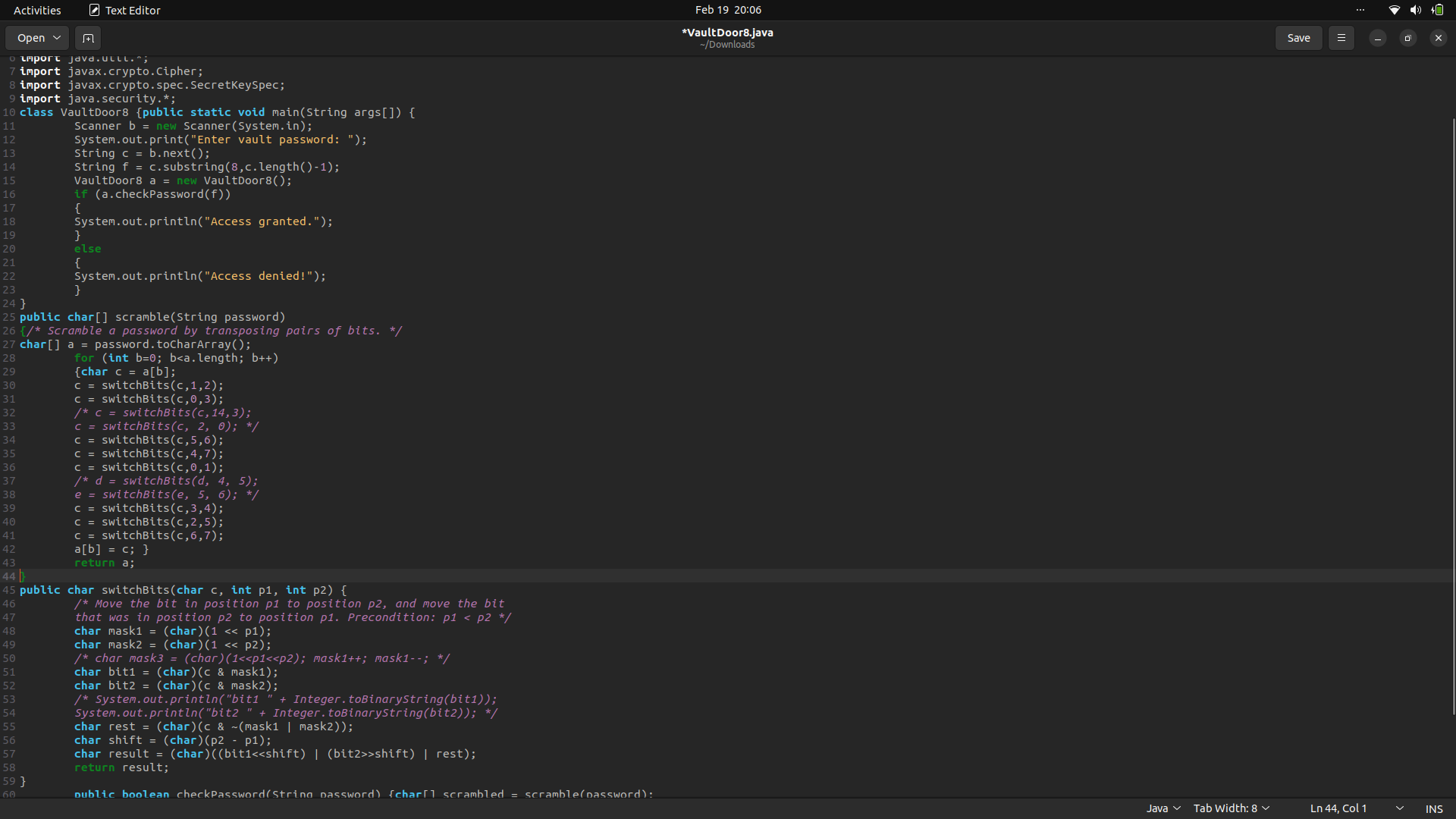
Resources:

Decimal to Binary: <https://www.rapidtables.com/convert/number/decimal-to-binary.html>

ASCII table: <https://www.rapidtables.com/code/text/ascii-table.html>

VAULT-DOOR 8

When we open the source code, we are greeted with a code without any spacing. After spending a lot of time beautifying it, we get a code like the following:



Here we see a number of bit shifts. Lets start off by representing each character as what they are: 8 bits. Before the character is put through the scramble function, its indices are like this:

|  |
| --- |
| **Index** |
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |

Pretty obvious. Lets now go through the bit switches. The first instance of the switchBits function switches bit 1 and 2. The character now look like this:

|  |  |
| --- | --- |
| **Index** | **Original Index** |
| 0 | 0 |
| 1 | 2 |
| 2 | 1 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |

We then switch bits 0 with 3, 5 with 6, and 4 with 7. Now every single bit has been switched once. The character now looks like this:

|  |  |
| --- | --- |
| **Index** | **Original Index** |
| 0 | 3 |
| 1 | 2 |
| 2 | 1 |
| 3 | 0 |
| 4 | 7 |
| 5 | 6 |
| 6 | 5 |
| 7 | 4 |

Once every bit has been switched once, we have to remember to switch the bits according to their current index and not their original index. So, when we continue to switch bits, we have to switch them based on their index. We continue and switch 0 with 1, 3 with 4, 2 with 5, and 6 with 7. The character now looks like this:

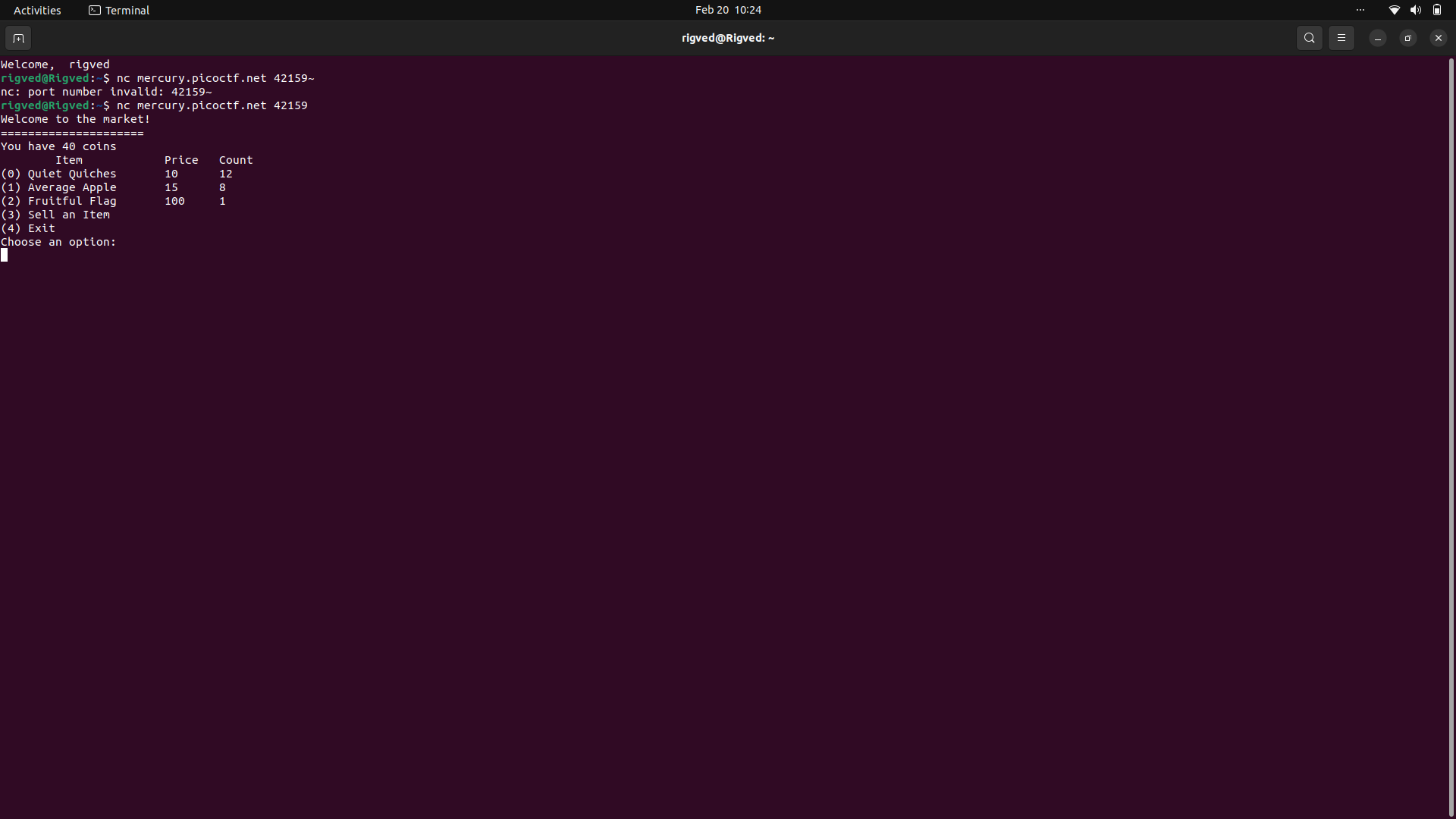
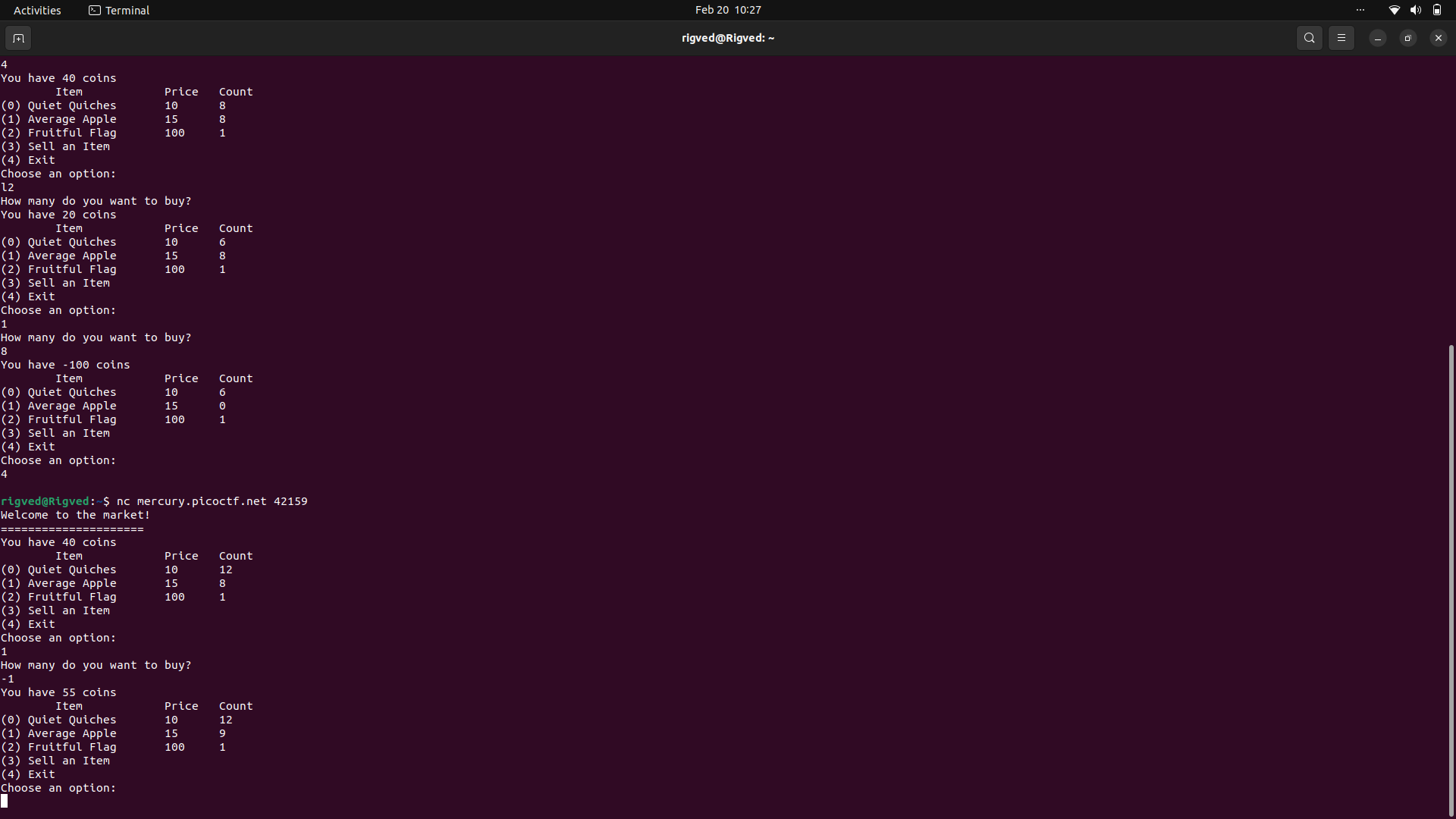
|  |  |
| --- | --- |
| **Index** | **Original Index** |
| 0 | 2 |
| 1 | 3 |
| 2 | 7 |
| 3 | 0 |
| 4 | 6 |
| 5 | 1 |
| 6 | 4 |
| 7 | 5 |

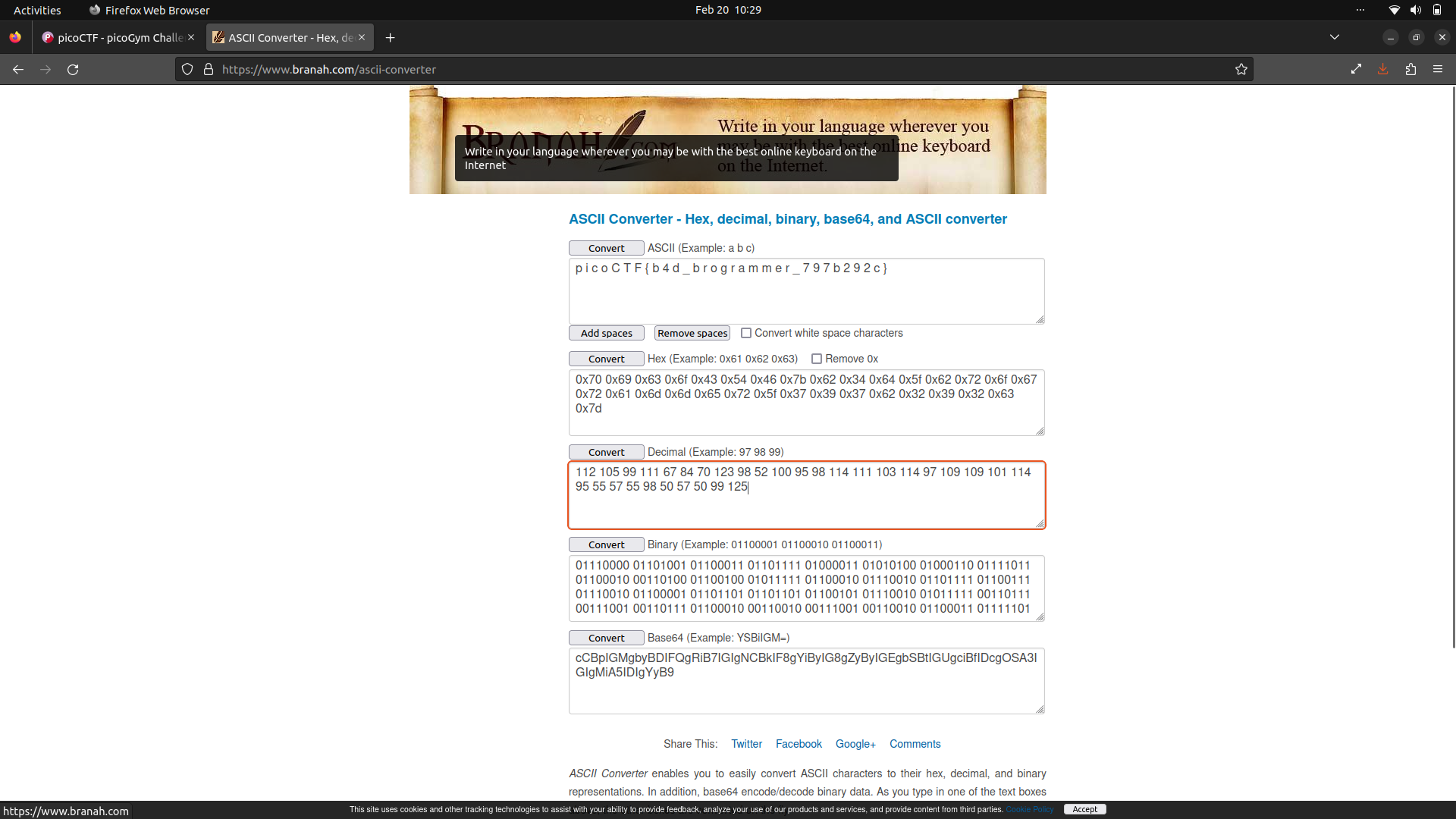
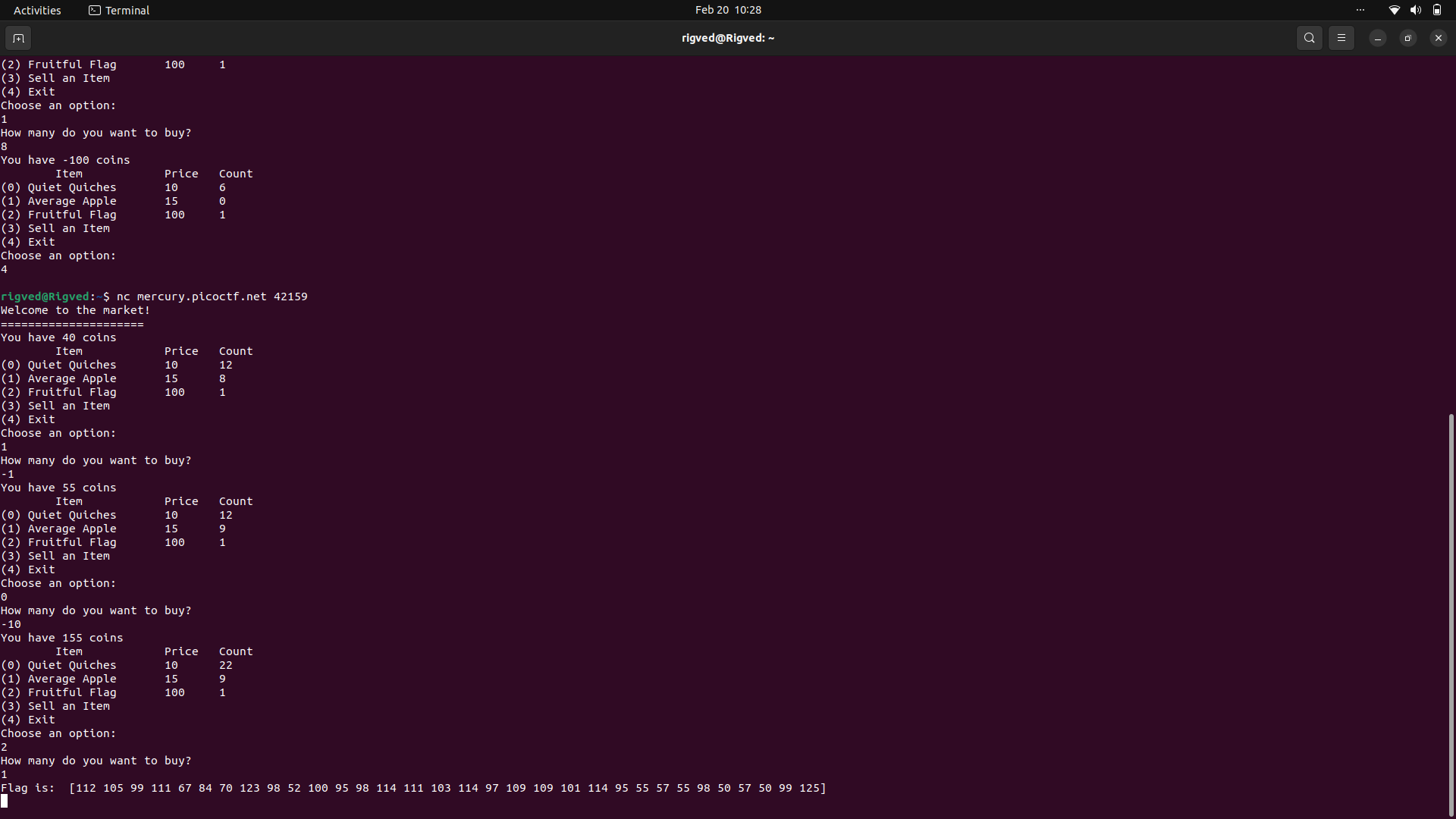
NOTE: The above explanation has been taken from the internet simply because this is too long to write and the original author has explained it nicely.

Now all we have to do is reverse this. We switch each index in the character with its original index as according to the above table to get the original character, switching 0 with 2, 1 with 3, 2 with 7, etc. and putting it into a char array which we then convert to a String and get our flag.

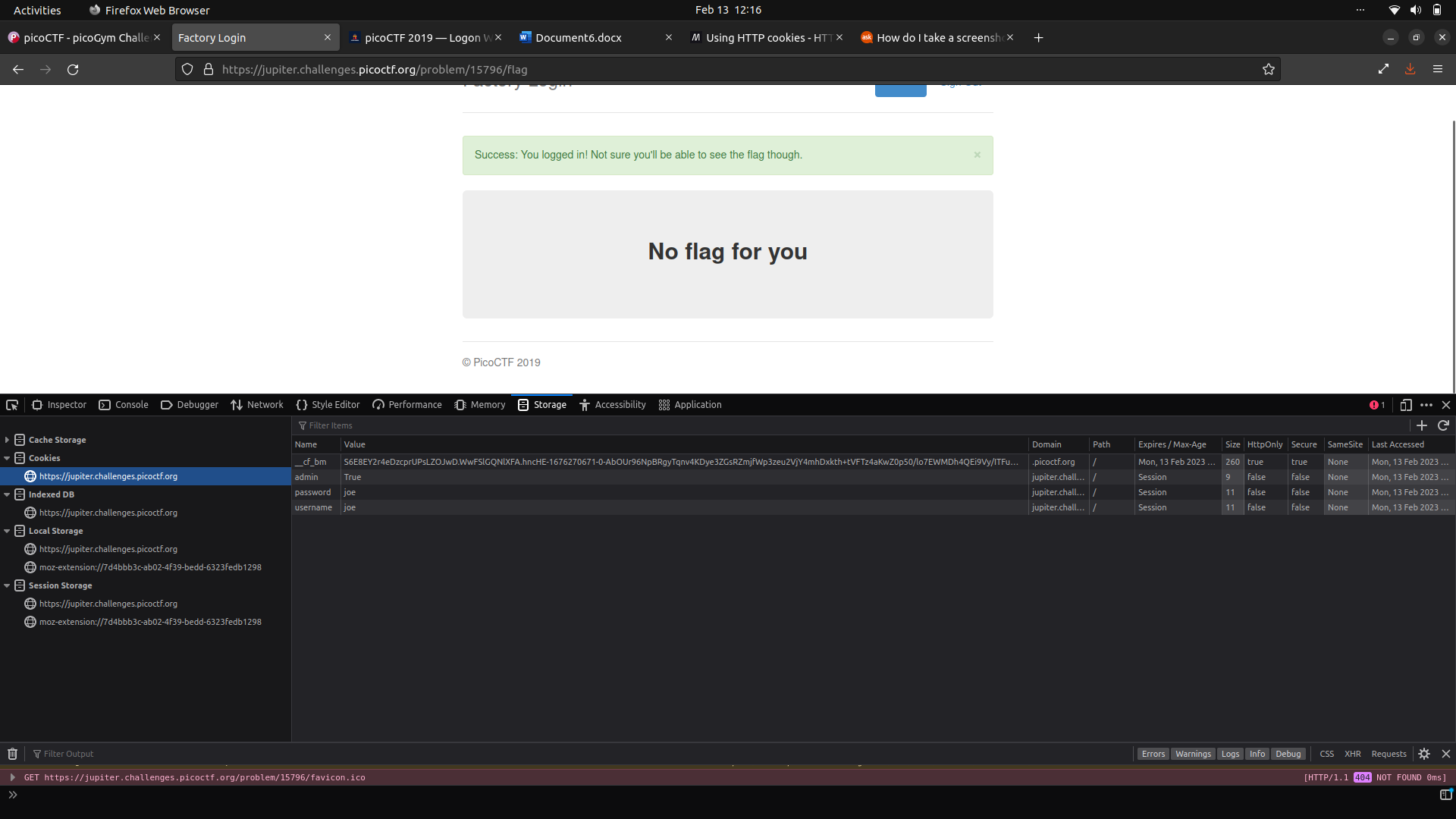
The flag is: picoCTF{

SHOP

As can be observed, our money decreases when we purchase any item from the shop. Moreover, our money is permitted to go into negative. This leads us to question whether one can also buy negative quantities.As is seen above, it is possible to buy negative materials. Thus, we increase the money available to us and buy the flag which is later translated with an ASCII converter.

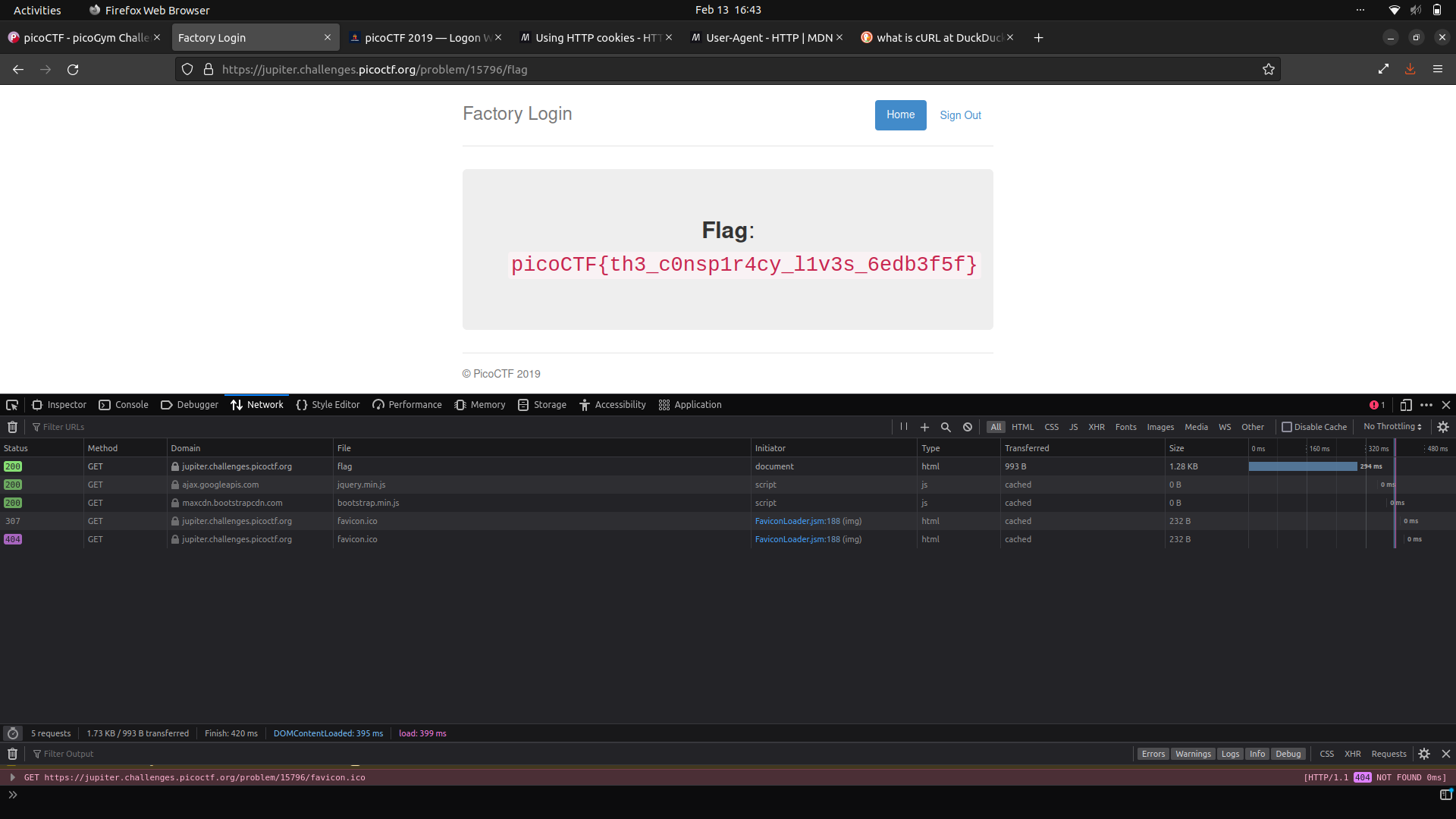


LOGON

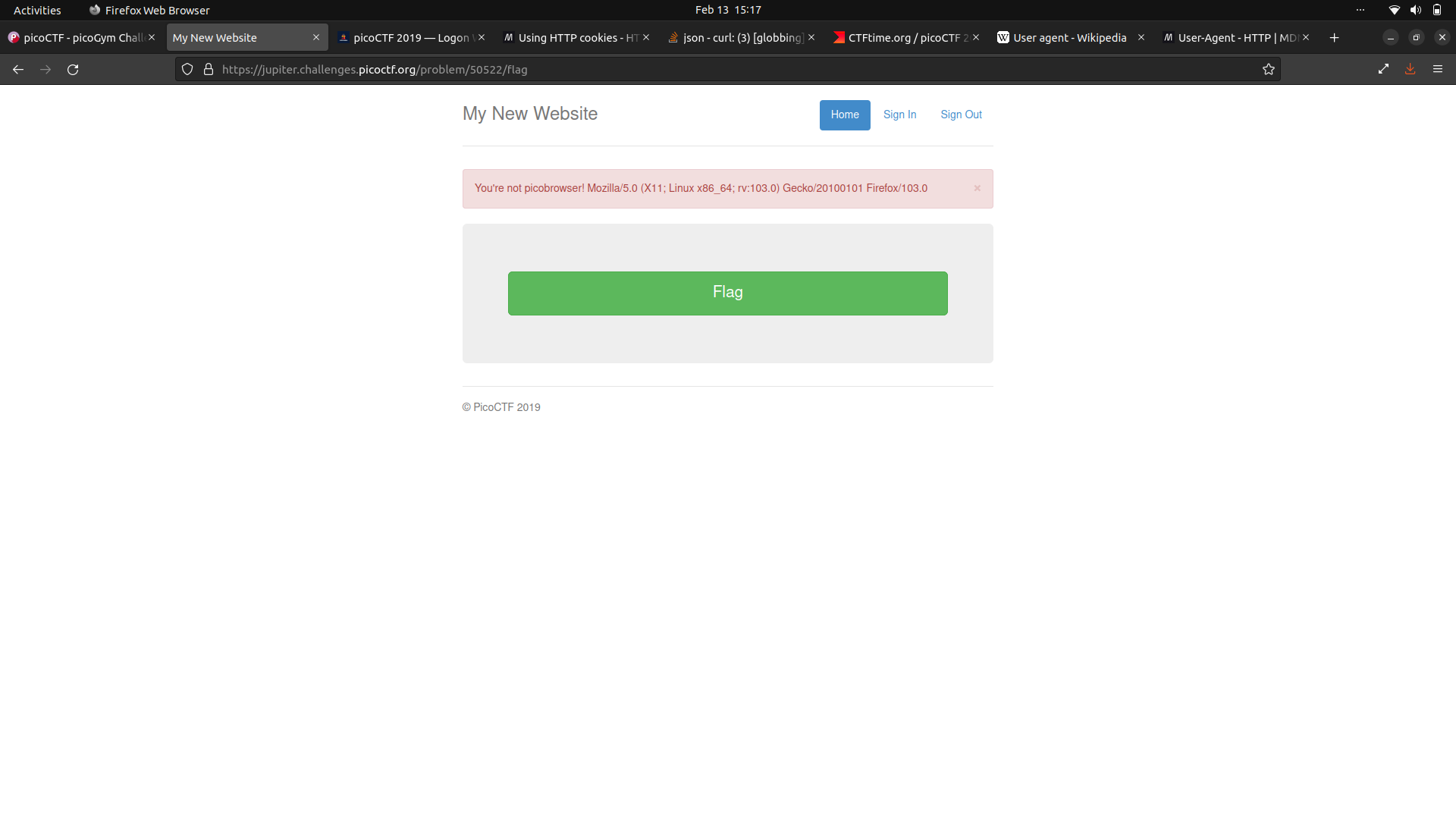


Here, we are to login to the webpage of a factory. We can access it using any login id and password, but we won’t be given the flag. This is caused by the fact that we aren’t logging in as an admin.

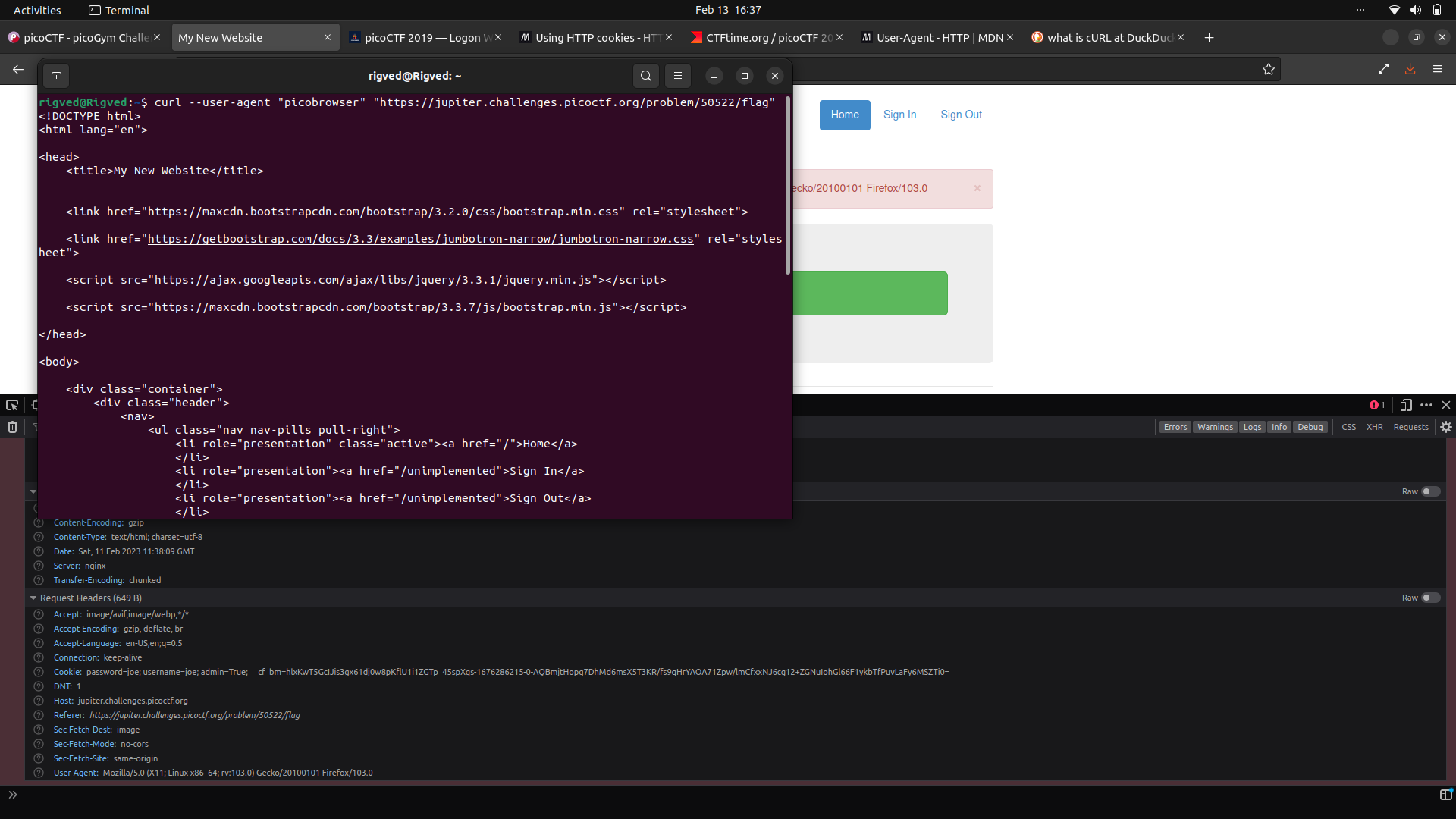
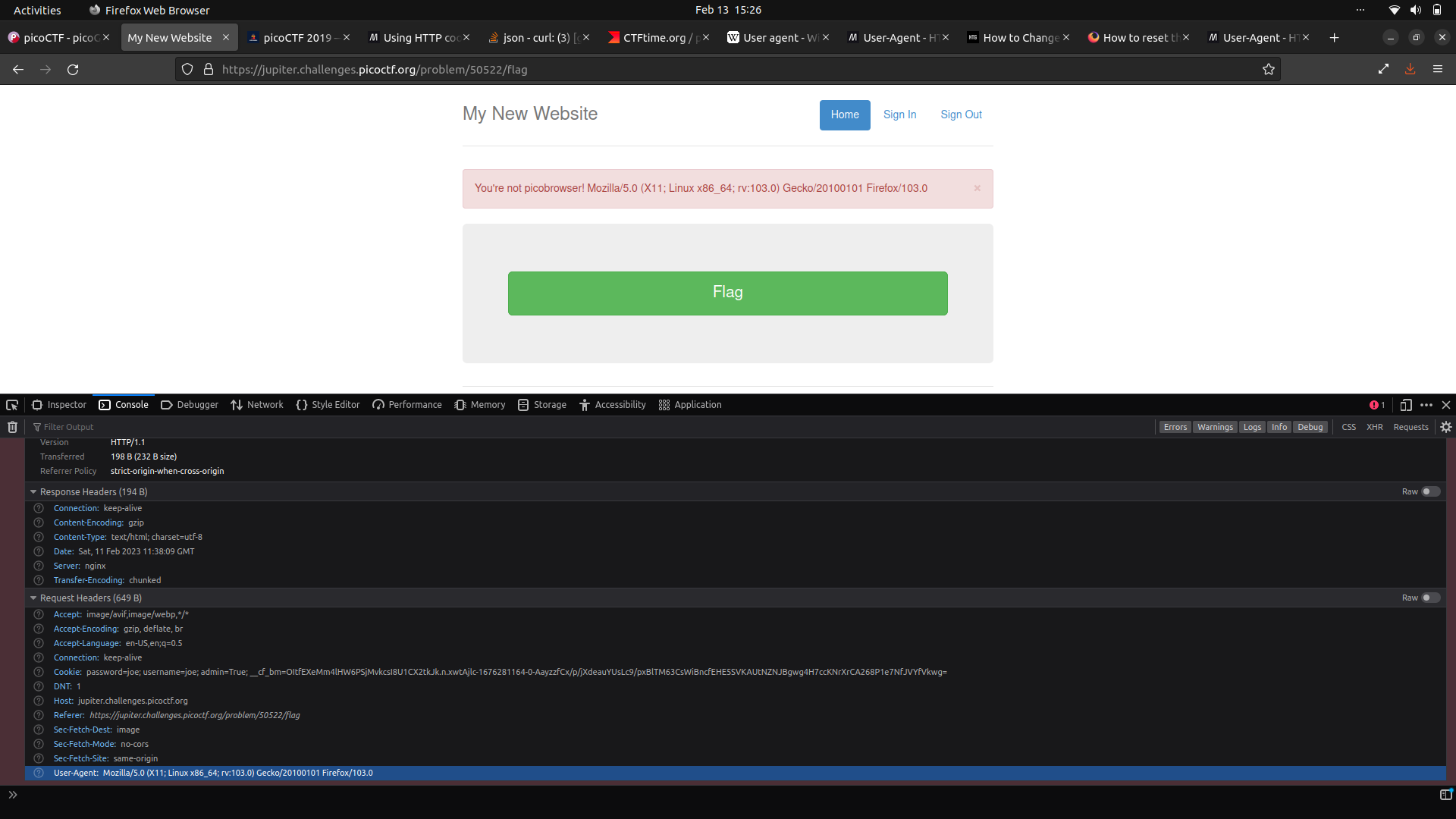
All we need to do is to change a particular value classifying us as not an admin to ‘true’ and to refresh the network connection. This gives us access to the flag. Alternatively, we can also use cURL command to get access to the flag.

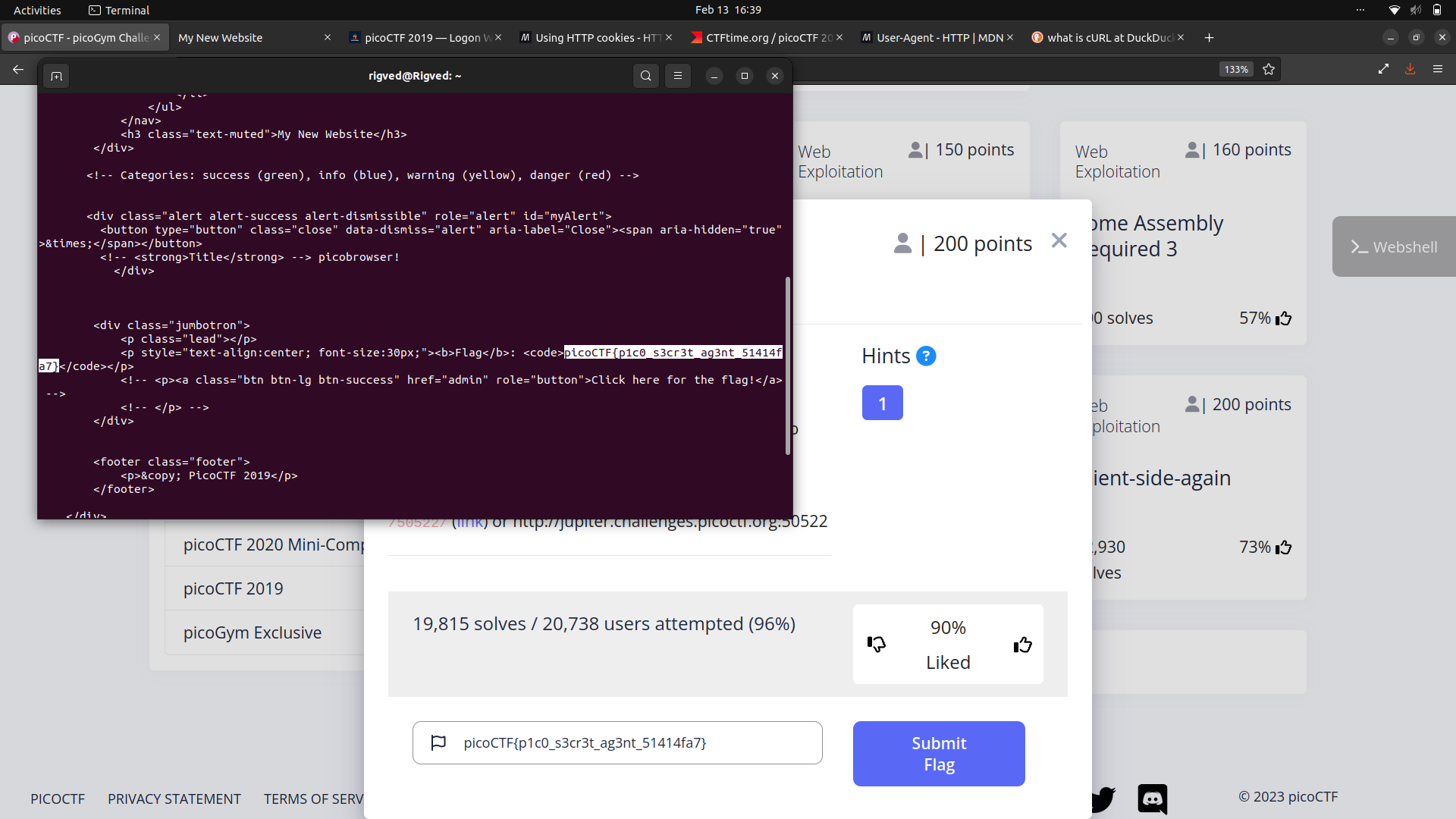


PICOBROWSER



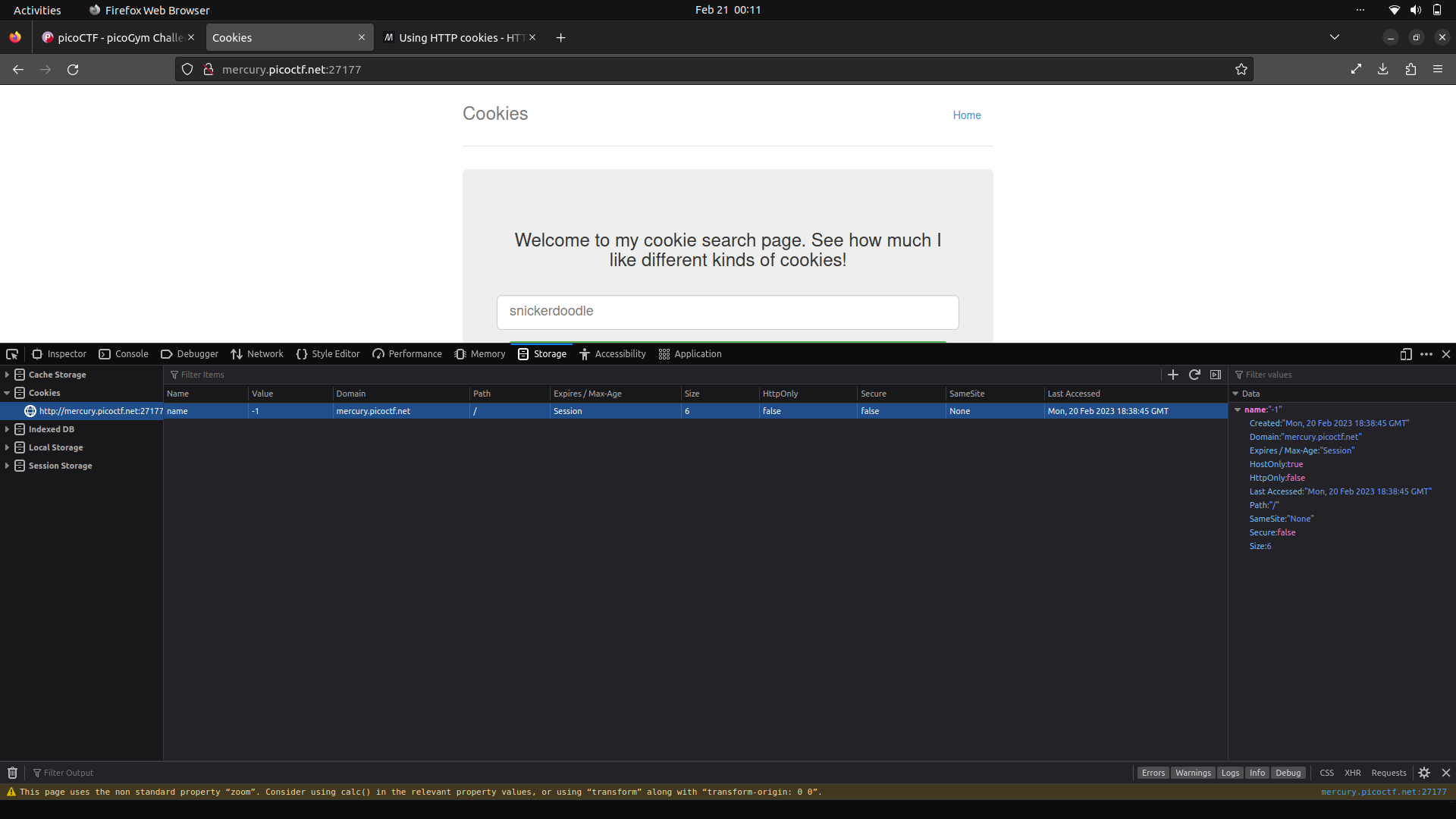
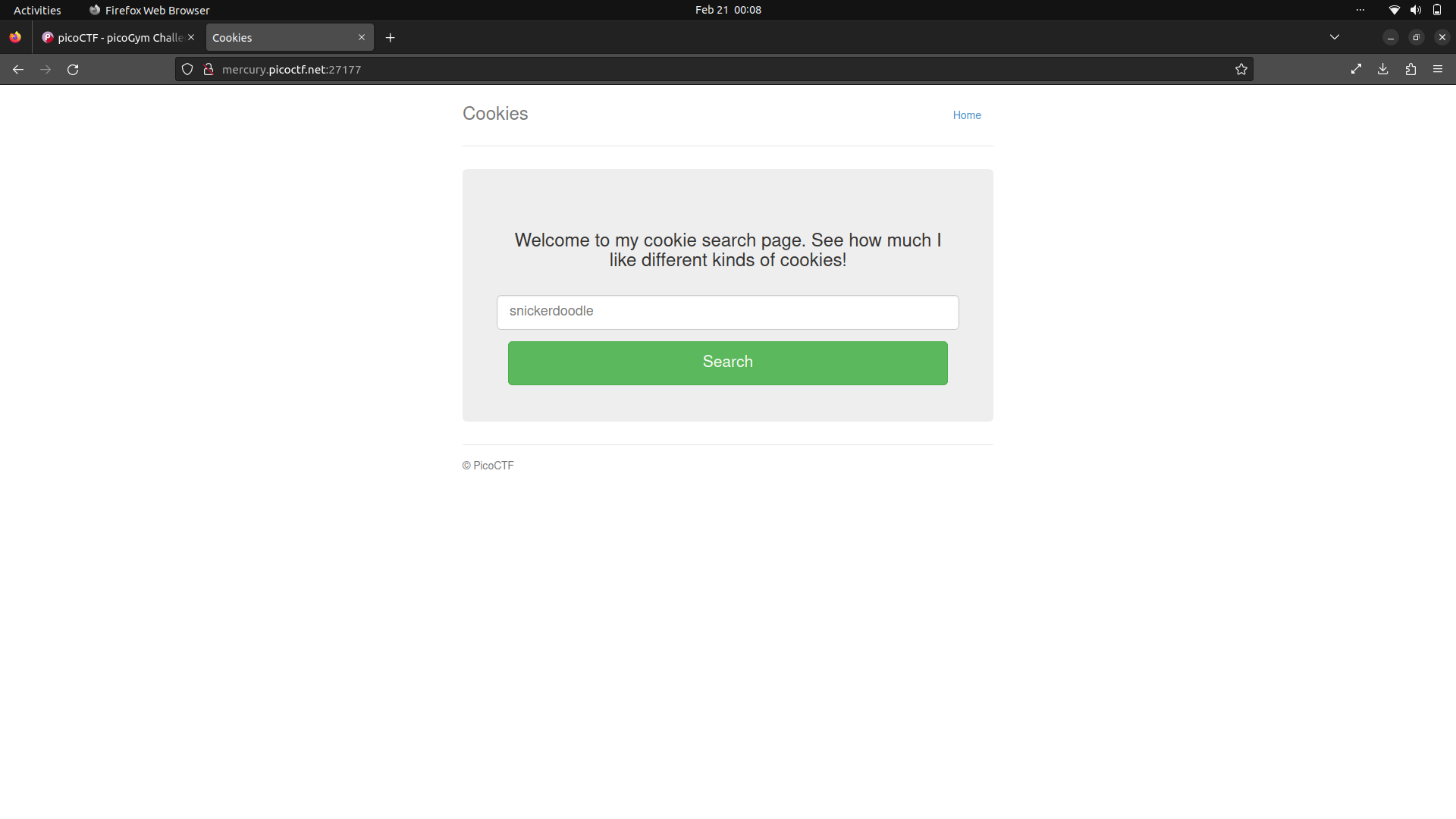
Here, as is highlighted in the red prompt, we are not accessing the webpage using picobrowser as the user-agent.



At this point, it is simply a matter of using the cURL command with a change in the value of user-agent to picobrowser. This gives us access to 

COOKIES

We can try entering a cookie name in the search box, which results in the following:



As can be observed above, the value of ‘name’ = ‘-1’

This is important as it is the only observable difference between no input and when you feed it the prompted value, which is now ‘0’.



After attempting brute force, we get the flag at value of name = ‘18’.

**Flag**: picoCTF{3v3ry1\_l0v3s\_c00k135\_064663be}

FORENSICS