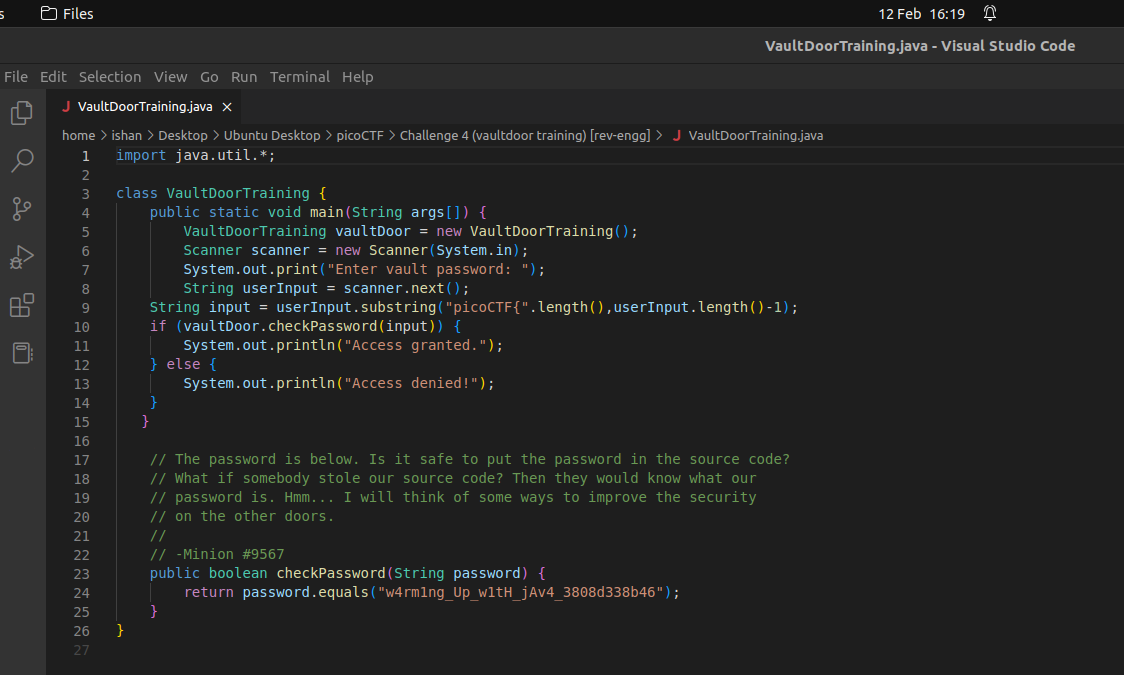
***PicoCTF Challenges***

-Ishan Surana

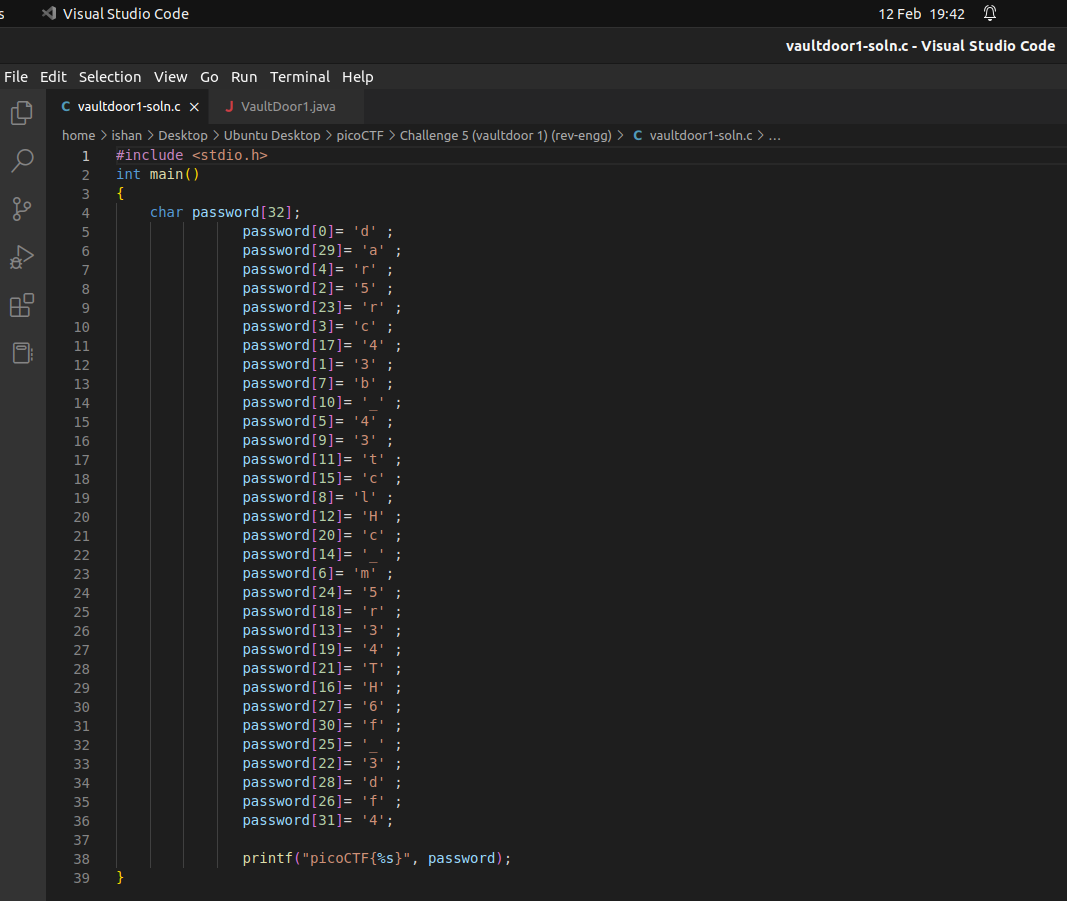
Challenge 4:- **Vaultdoor Training**

This was my 1st challenge in Reverse engineering domain. The Java code I downloaded was easy to understand as the syntax is very similar to C. The password is stated in source code where it is checking if the password matches the correct password.



Challenge 5:- **Vaultdoor 1**

This is a modification of previous level where instead of password bein stored directly, the elements are stored individually in the indices. Therefore, I made a C code by copying the password part of the source code and modifying it a bit to make the code syntax as per C (such as replacing the “ &&” with “;”, replacing normal brackets with square ones, etc.). Then I printed the string and got the required password.

My code:-

Challenge 6:- **Vaultdoor 3**

*(where’s no. 2 bruh -\_-)*

This is a challenge with a little more security than previous ones, wherein the letters of the key have been rearranged according to some rules:-

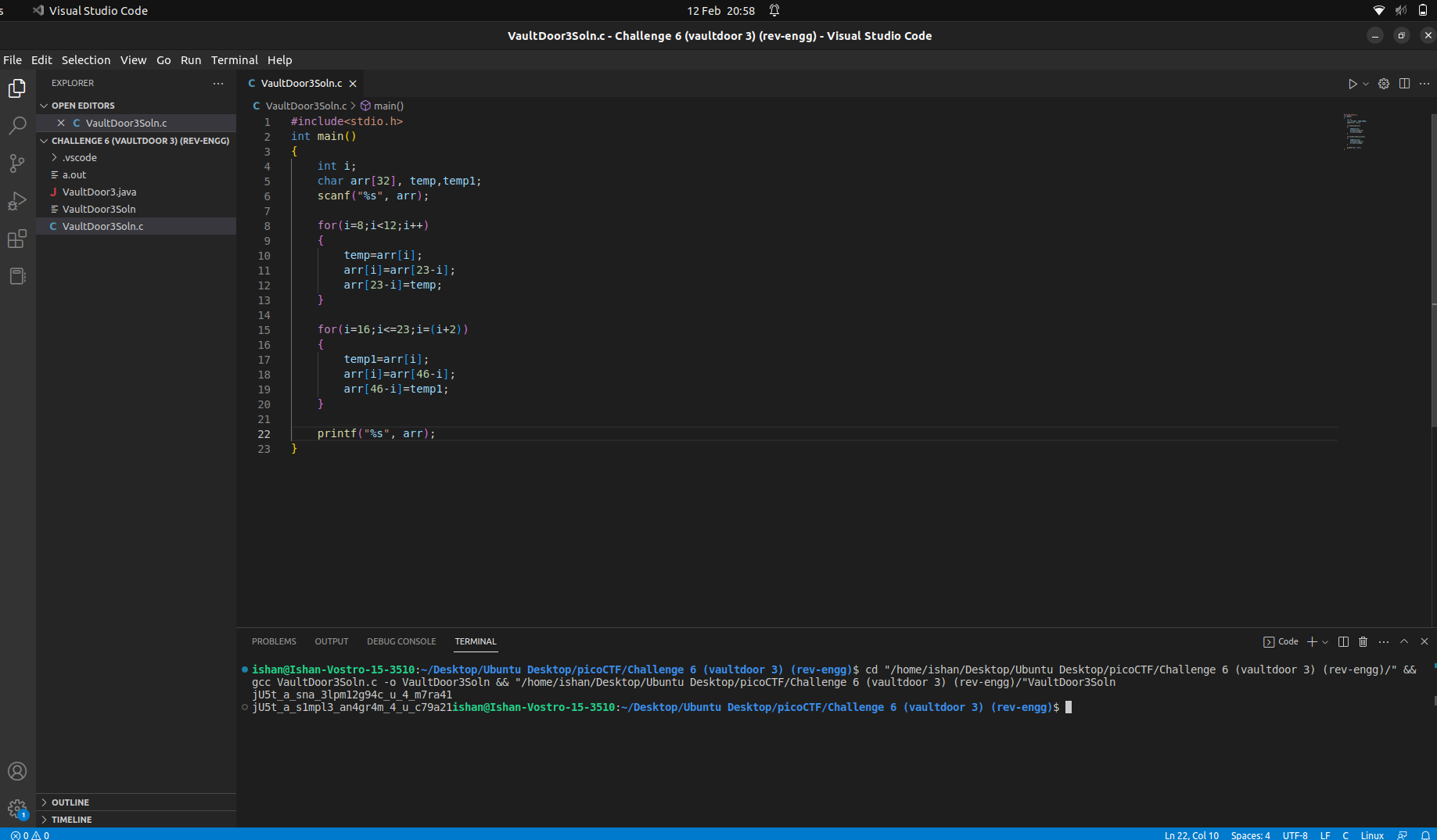
* Indices 0-7 => unchanged
* 8-15 => reversed
* 16-30 all even indices (inclusive) => revesed
* 17-31 all odd indices (exclusive) => same

Therefore I made a C code which has an array and acceots the jumbled key. Since reverse of reverse = original, I copied the same for loop conditions as the original question code.

**Note** that if we are using fgets and puts, the length of array has to be 33 (1 more than no. of characters in entered string) as the 33rd element would be the null element. Taking only 32 size omits a character at the end.

We can take size 32 if we use scanf. Also, puts string will automatically print a newline character while printf string won’t do so.

My code:-

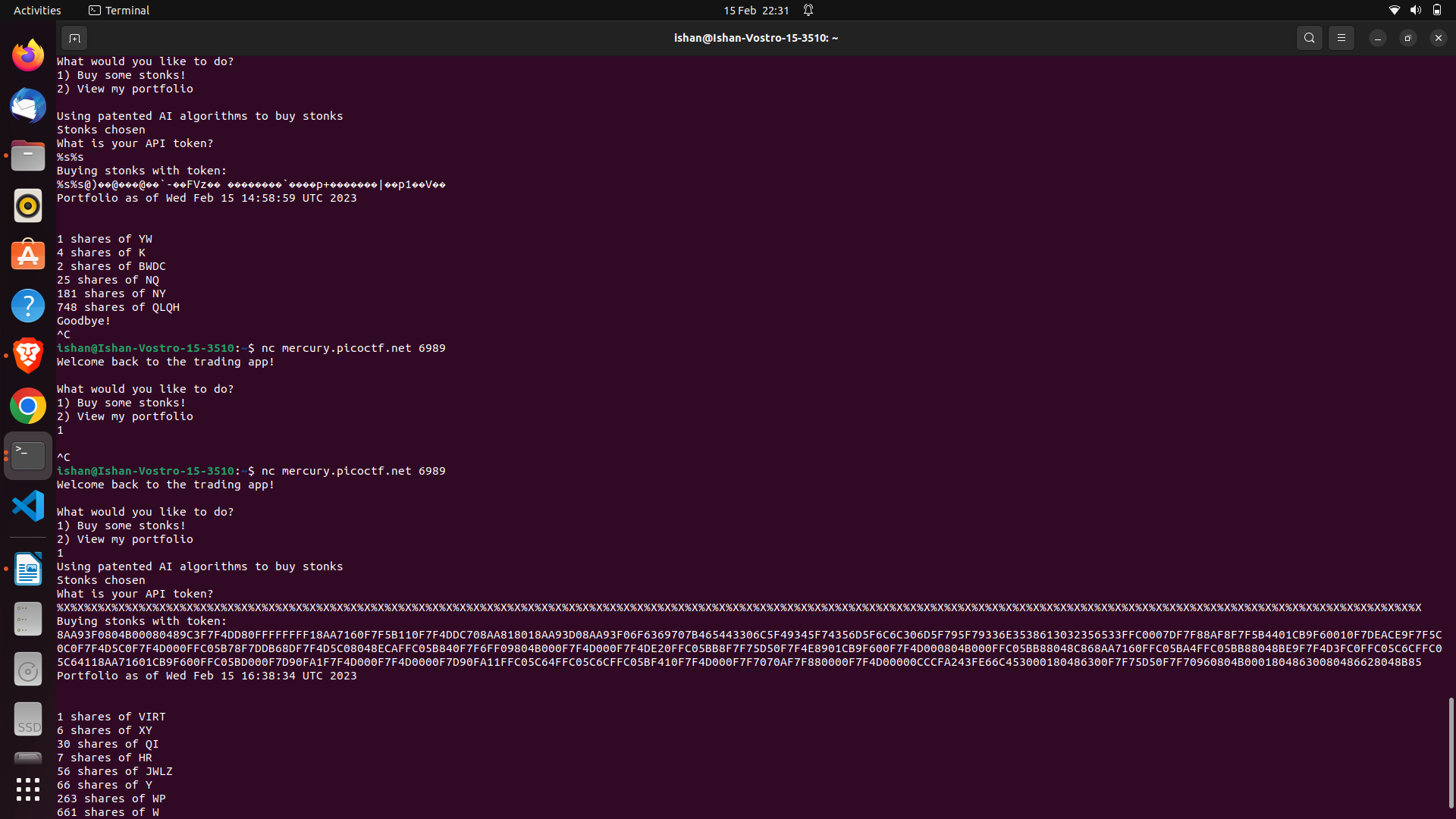
*(image showing bash line starting from end of ouput itsel fas no newline character is added, but here we don’t increase size as string data type accounts for null character)*

Challenge 7:- **Stonks**

This is my first challenge in binary exploitation. I downloaded the code and read it to try to understand what is happening, and I also ran the netcat link given in the challenge. I tried using lot of different entries whereit asked me my API, but wasn’t able to make much progress.

So, I tried to look on the internet for help an dfound a video which showed me that if I input a format specifier instead of an inout string, it will list out its memory contents to me. It used the %X specifier for hex data type and I received an output (I later on tried with %s and %d too, but didn’t get much comprehensive results).

I received a lot of output in hex which I converted to ASCII using CyberChef, and then I saw that the flag was printed but sets of four characters were reversed. Hence, I corrected the output and submitted the correct flag.

*(image showing my result with %X format specifier typed 100 times vs string format specifier typed a few times*

*(typing %s too many times gives memory error)*