

Rootkits & Hacking Assignment 3

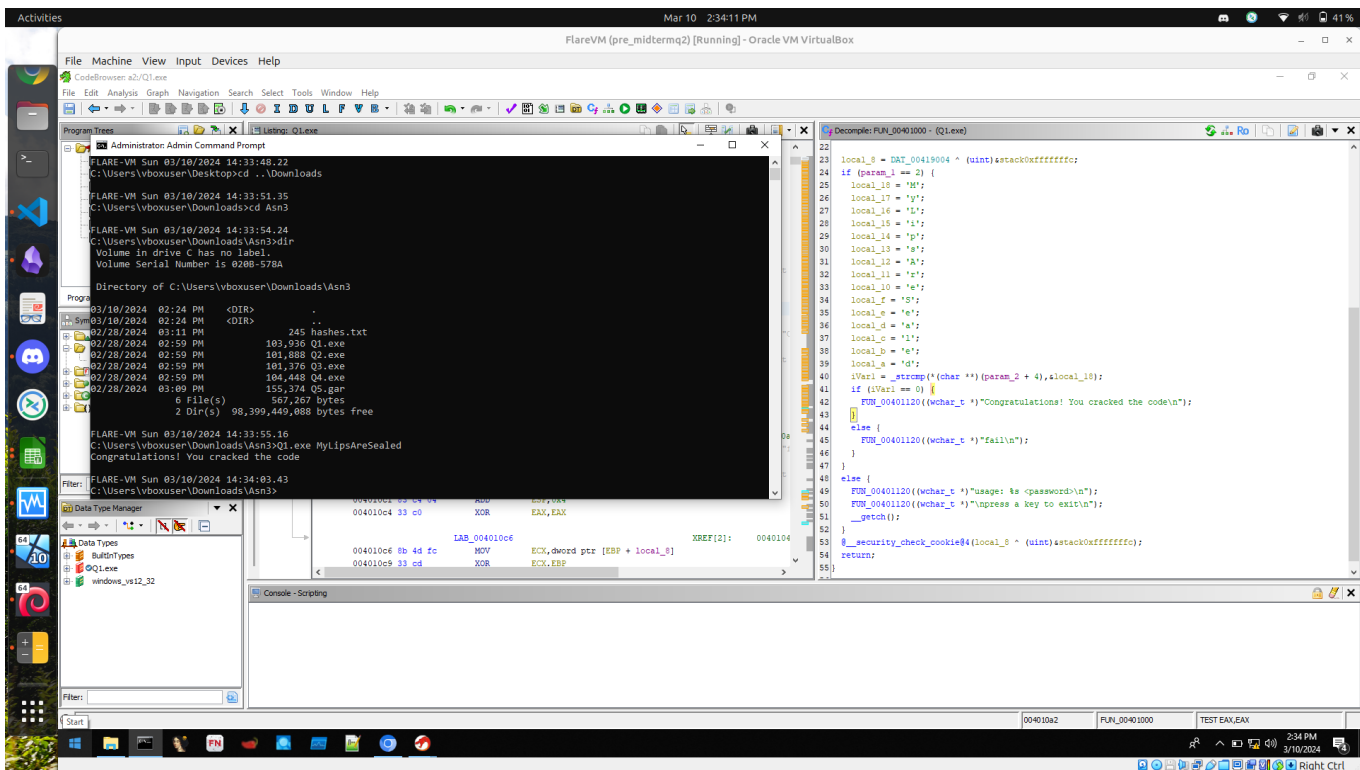
#deliverable

#conestoga

#rootkits-and-hacking

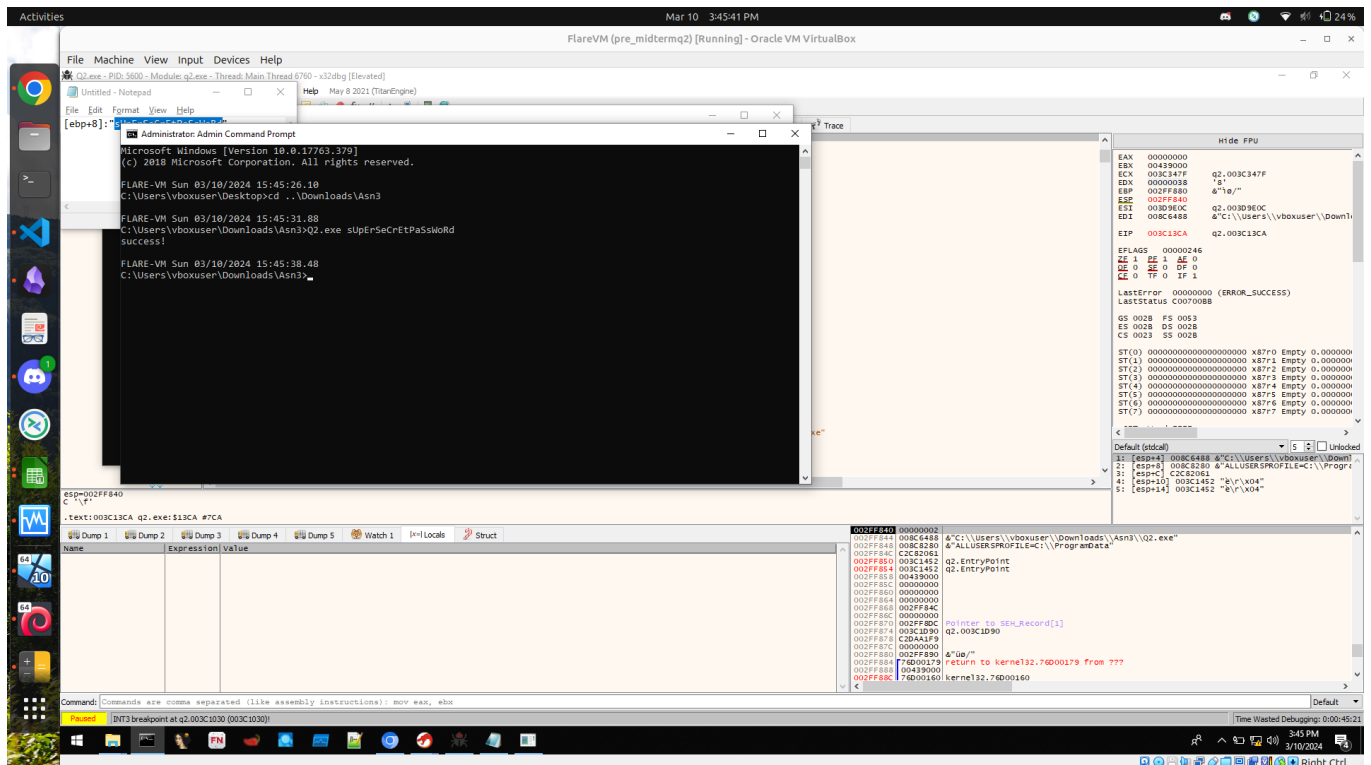
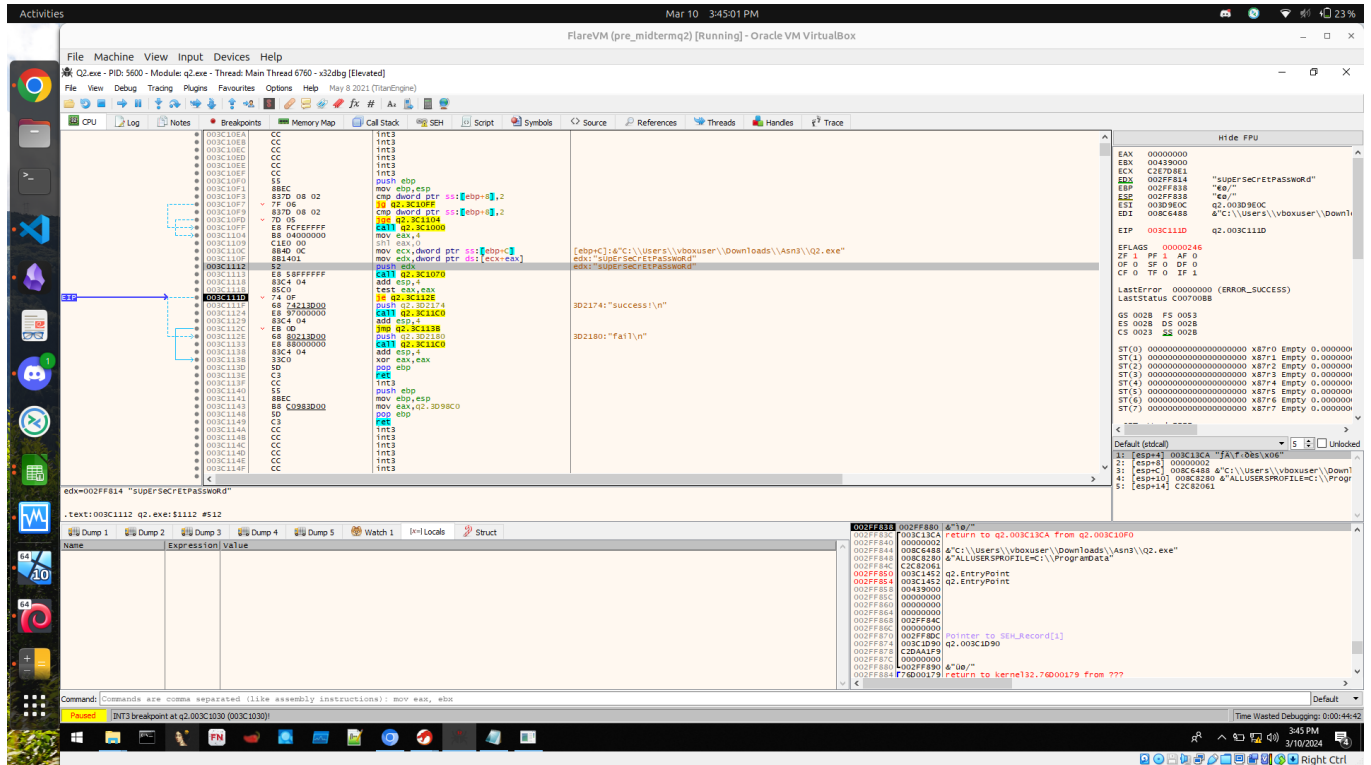
Course	SECU74000: Rootkits & Hacking
Student	Emil Harvey
Instructor	Dr. Steve Hendrikse
Assignment	3
Date	2024-04-05

Question 1

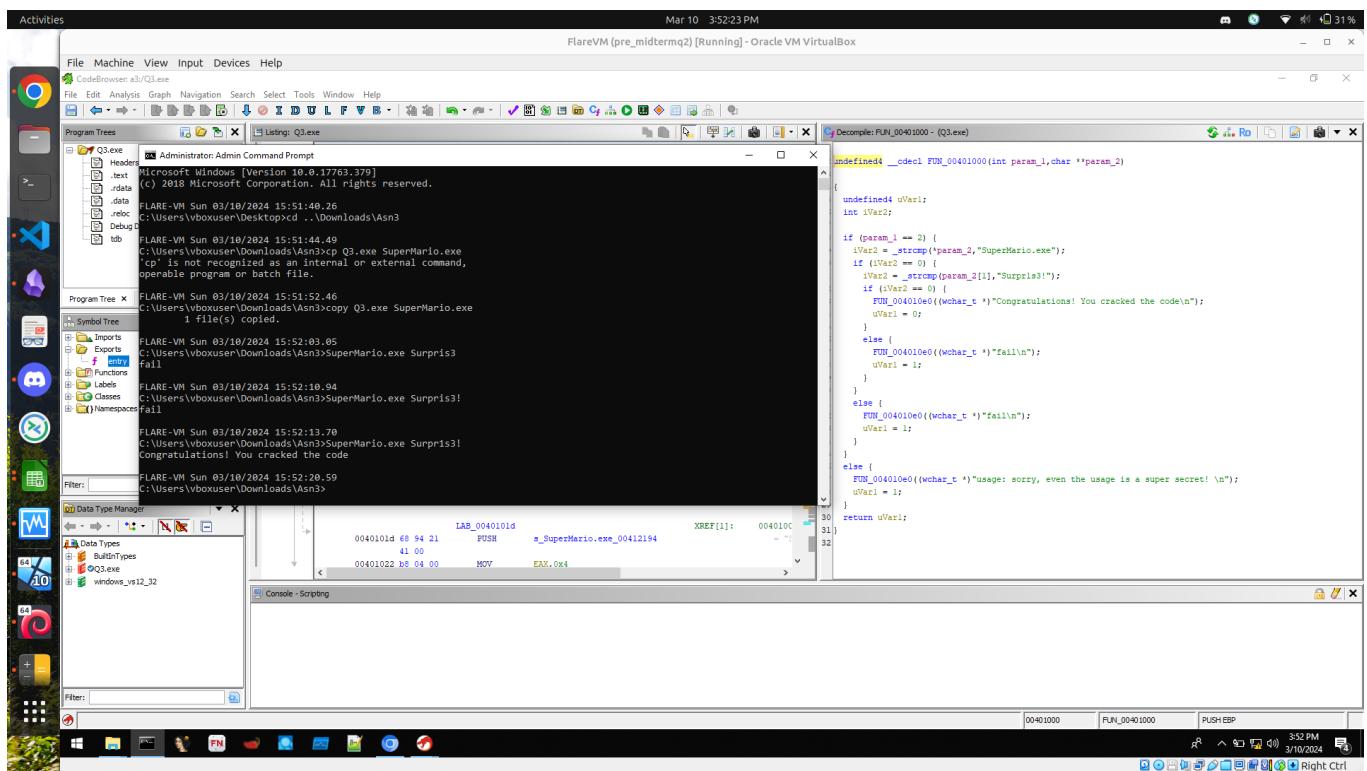
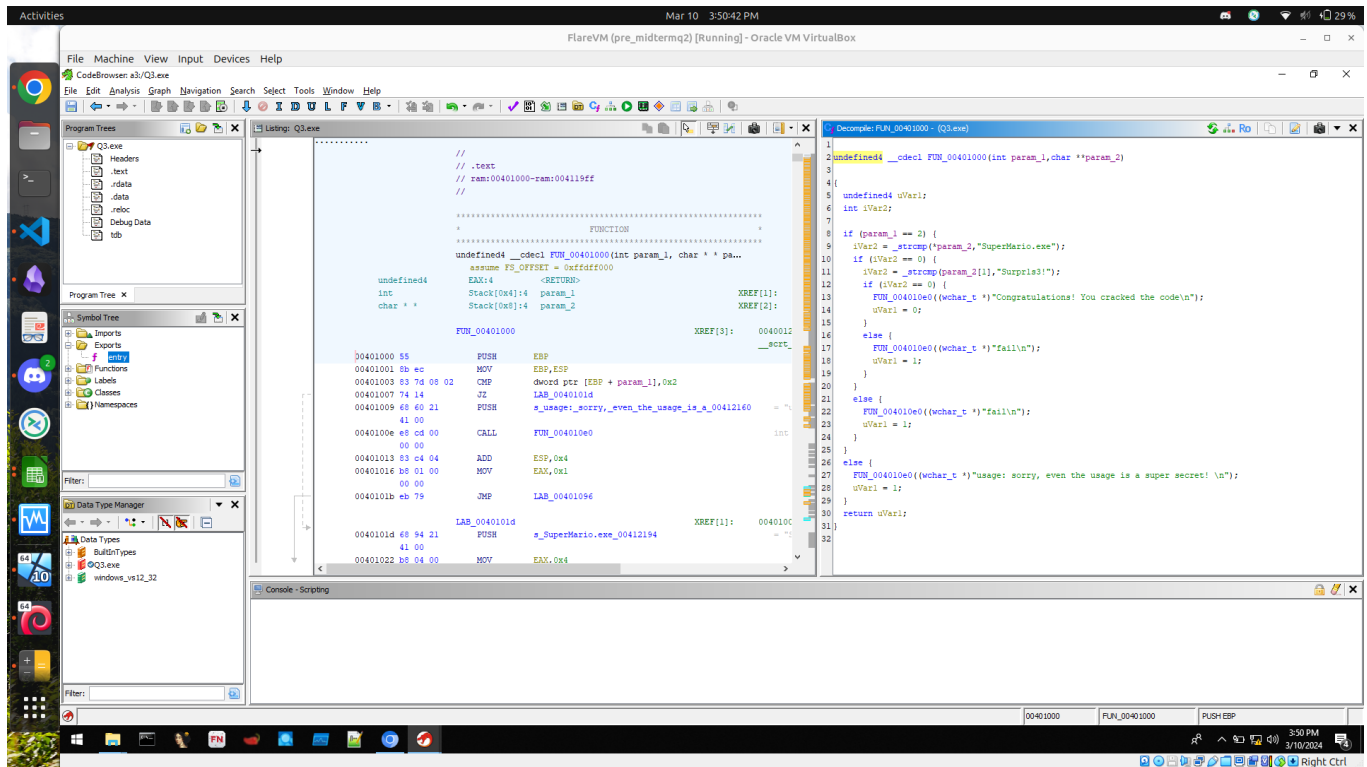


(Password is defined from line 25 to line 39).

Question 2



Question 3



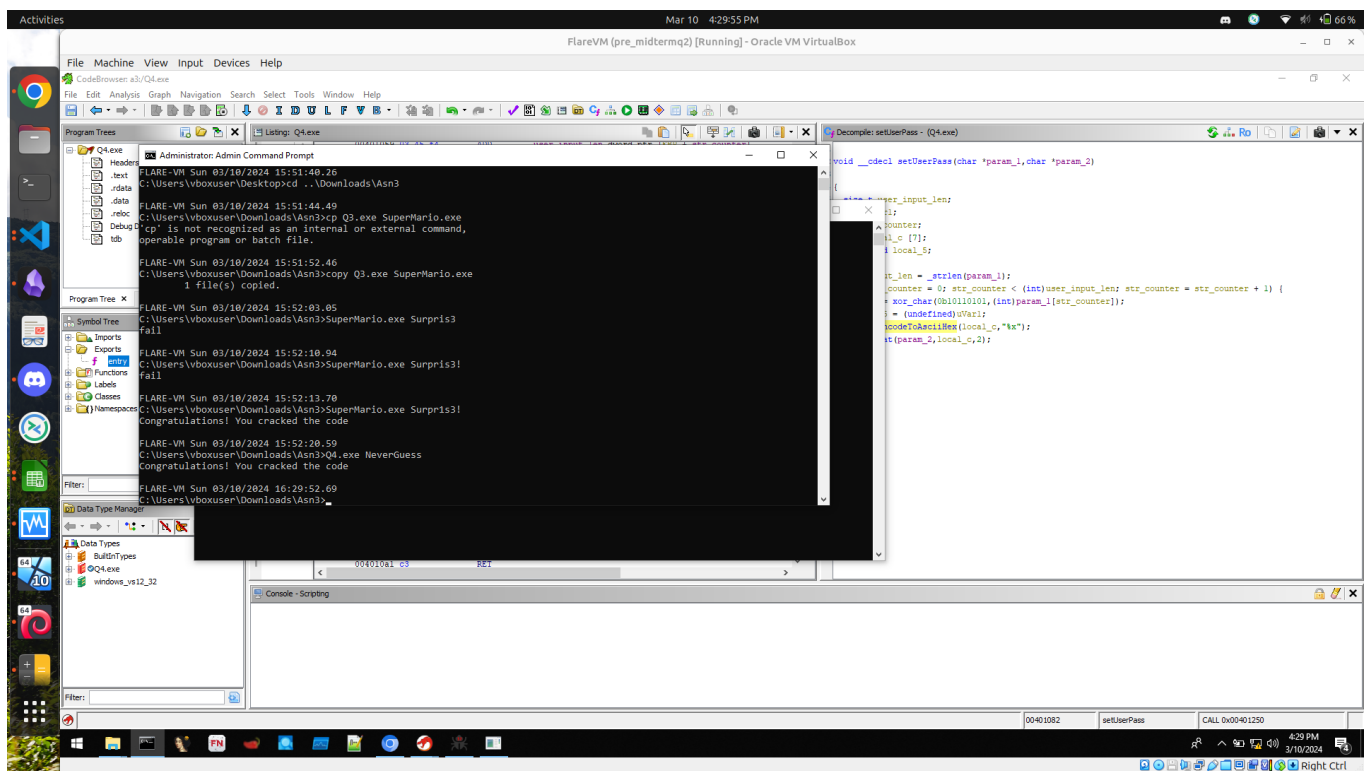
(Had to rename the executable to SuperMario.exe first).

Question 4

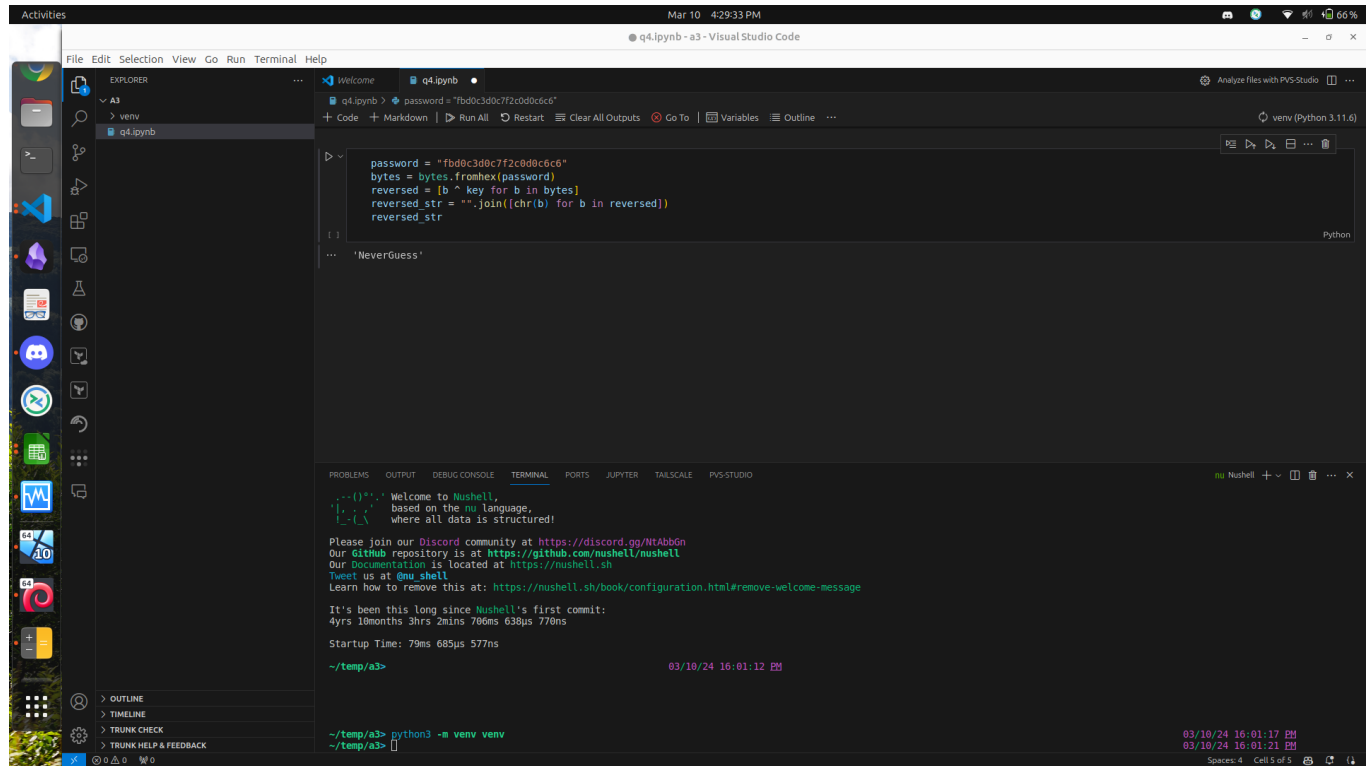
After opening the sample in Ghidra, I saw that some operations were performed on the user's input before being compared to a stored value (a string starting with `fb...`). When I looked at the obfuscation logic, I saw a formatter string `"%x"` as well as some XOR logic. So I ran the program in x32dbg using the input `"abc"`. It transformed this input to `"d4d7d6"`, which matched the encoding approach of XORing the input (character by character) with the stored key and then printing the result, byte by byte, as hex. So, I wrote some Python to that reversed the process:

1. take the `fb...` string,
2. get the byte representation by interpreting it as hex
3. XOR it with the stored key (note that the inverse of XOR is XOR)
4. Print the result, interpreting it as ASCII

This gave me the password `"NeverGuess"`, which I confirmed to be correct.



(The python code I used)



The screenshot shows a Visual Studio Code editor window with a Python script in the main editor and a terminal window at the bottom. The script defines a password, converts it to bytes, reverses it, and prints the reversed string. The terminal window shows the output of the script, which is 'NeverGuess'.

```
password = "fbd0c3d0c7f2c0d0c6c6"
bytes = bytes.fromhex(password)
reversed = [b ^ key for b in bytes]
reversed_str = ''.join([chr(b) for b in reversed])
reversed_str

'NeverGuess'
```

```
...()... Welcome to Nushell,
'...' based on the nu language,
'...' where all data is structured!

Please join our Discord community at https://discord.gg/NtAbbdn
Our Github repository is at https://github.com/nushell/nushell
Our Documentation is located at https://nushell.sh
Tweet us at @nu_shell
Learn how to remove this at: https://nushell.sh/book/configuration.html#remove-welcome-message

It's been this long since Nushell's first commit:
4yrs 10months 3hrs 2mins 706ms 630µs 770ns

Startup Time: 79ms 685µs 577ns

~/temp/a3>

03/10/24 16:01:12 PM

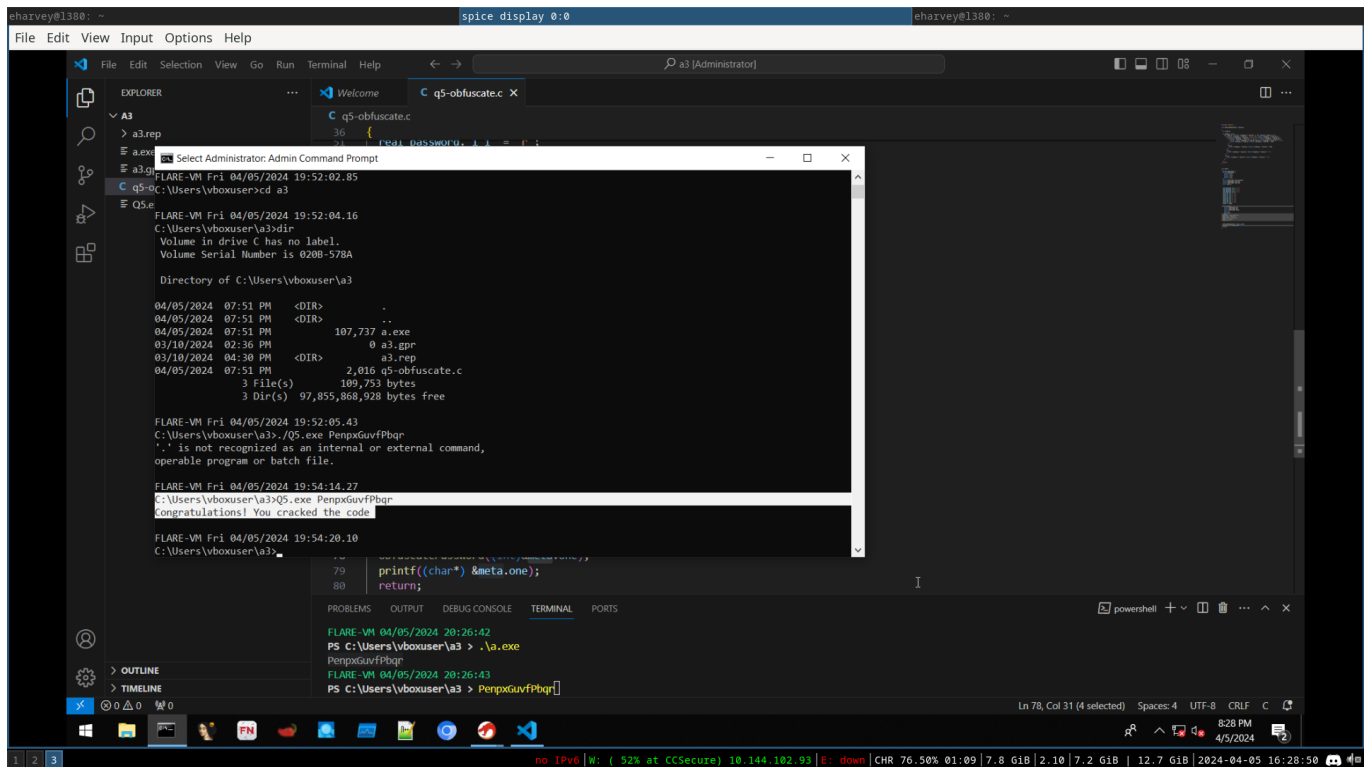
~/temp/a3> python3 -m venv venv
~/temp/a3>
```

(I had defined `key` earlier in this notebook).

Question 5

(In the time between Question 1-4 and Question 5, I migrated to Debian , hence why the UI looks different)

After opening the sample in Ghidra, I was able to determine a function that checked the user input (command line argument). Ghidra decided to decompile the "real" password (stored on stack) as a series of 4 and 2 byte variables declared next to each other. I assumed that Ghidra decompiles binaries "straight", not trying to reverse engineer layout optimizations, padding, etc, making me believe that the concatenation of each variable, in order of declaration, was the password. This password ("CrackThisCode") was modified by a function before comparison, so I copied this function into VS Code as well as the "CrackThisCode" variables declarations. I did not take the time to fully understand this function, but I did see that it operates character by character (this made me know that the result should be 13 characters, the same length as "CrackThisCode"). After a few modifications to prevent padding and layout changes, I created a small C program that printed the password, which is `PenpxGuvfPbqr` :



(Below is the source code I used to discover the password. I tried to modify Ghidra's output as little as possible, so excuse the poor style). I compiled using `gcc -m32 -O0 .\q5-obfuscate.c`

```
#include <stdint.h>

void obfuscatePassword(int param_1)
```

```

{
    int local_8;

    if (param_1 != 0) {
        for (local_8 = 0; *(char *)(param_1 + local_8) != '\0'; local_8 =
local_8 + 1) {
            if ((* (char *) (param_1 + local_8) < 'a') || ('m' < * (char *) (param_1 +
local_8))) {
                if ((* (char *) (param_1 + local_8) < 'A') || ('M' < * (char *) (param_1
+ local_8))) {
                    if ((* (char *) (param_1 + local_8) < 'n') || ('z' < * (char *)
(param_1 + local_8))) {
                        if (('M' < * (char *) (param_1 + local_8)) && (* (char *) (param_1 +
local_8) < '[')) {
                            * (char *) (param_1 + local_8) = * (char *) (param_1 + local_8) +
-0xd;
                        }
                    }
                }
            }
            else {
                * (char *) (param_1 + local_8) = * (char *) (param_1 + local_8) +
-0xd;
            }
        }
        else {
            * (char *) (param_1 + local_8) = * (char *) (param_1 + local_8) +
'\r';
        }
    }
    else {
        * (char *) (param_1 + local_8) = * (char *) (param_1 + local_8) + '\r';
    }
}
return;
}

```

```

void main()
{
    #pragma pack(push, 1)
    // ^^ prevents padding, see https://stackoverflow.com/a/40643512
    struct undefined4 {
        uint8_t _0_1_;
        uint8_t _1_1_;
    }
}

```

```

    uint8_t _2_1_;
    uint8_t _3_1_;
};
struct undefined4 real_password;
struct undefined4 local_14;
struct undefined4 local_10;
short local_c;

real_password._0_1_ = 'C';
real_password._1_1_ = 'r';
real_password._2_1_ = 'a';
real_password._3_1_ = 'c';
local_14._0_1_ = 'k';
local_14._1_1_ = 'T';
local_14._2_1_ = 'h';
local_14._3_1_ = 'i';
local_10._0_1_ = 's';
local_10._1_1_ = 'C';
local_10._2_1_ = 'o';
local_10._3_1_ = 'd';
local_c = 0x65; // e in ASCII

// This struct is to ensure the ^^ string is not
// rearranged by Windows GCC
// Before this change, my program printed only 4 characters
// which I knew to be incorrect (I expected 13)
struct meta {
    struct undefined4 one;
    struct undefined4 two;
    struct undefined4 three;
    short c;
    char null;
} meta;
meta.one = real_password;
meta.two = local_14;
meta.three = local_10;
meta.c = local_c;
meta.null = 0;

obfuscatePassword((int)&meta.one);
printf((char*) &meta.one);
return;
}

```