Reverse Engineering Documentation

Name: Elvan Alandi

Table of Contents

Introduction

In this reverse engineering task, I am using IDA Freeware as a tool to disassemble the program. The executable (.exe) file of the program will be inserted into IDA Freeware to modify the program.

There are three required modifications to the program:

- Bypassing the student ID check range
- Saving unencrypted password to an output file
- Showing a warning message on the console

Before delving into the sub-tasks, it is necessary to create a project in IDA. When the IDA dialog box appears, click **New** and navigate to the program's executable file, then click **Open**. Select **Portable executable**, set the processor type to **MetaPC**, and click **OK.** When the confirmation dialog box appears, select **No**. The project setup is now complete, and we are ready to modify the program.

Sub-Task 1 – Bypassing Student ID Check Range

- The first step in reverse engineering is to examine the strings view. To access the strings view, navigate to the top toolbar, select **View**, then **Open subviews**, and finally choose **Strings**.

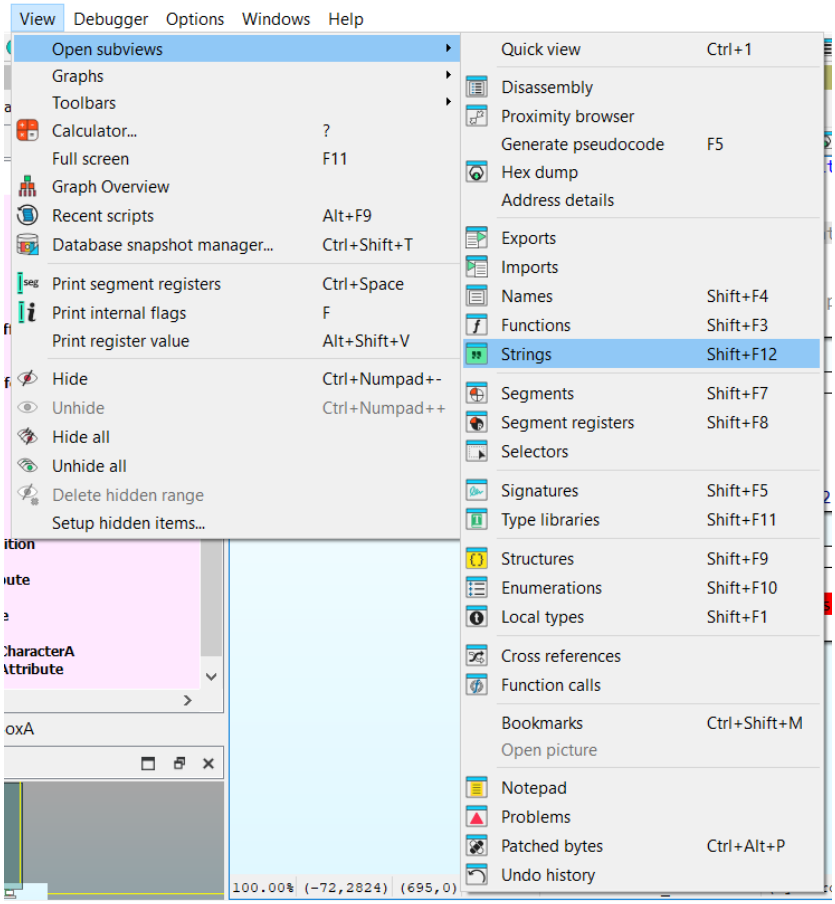

Figure 1. Strings View

- Since I am going to bypass the student ID check, I looked for the student ID prompt message. In the figure below, we can see the student ID prompt.

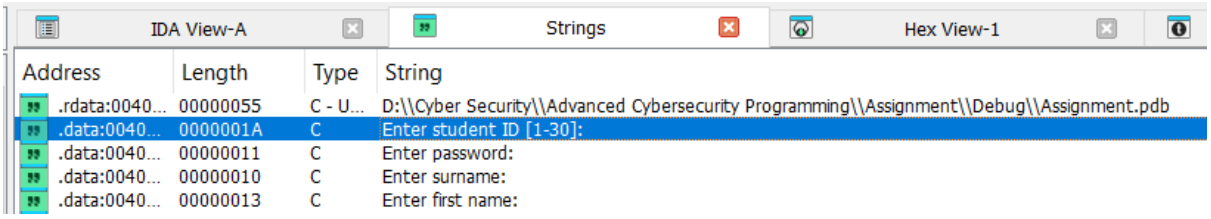

Figure 2. Student ID Prompt

After that, I **double-clicked** on it to view the disassembly.

- In the disassembly view, I could not determine the overall procedure used by the prompt. Therefore, I examined the **Data Cross Reference** (**DATA XREF**).

```
.data:00406000                 ;org 406000h
.data:00406000 aEnterStudentId db 'Enter student ID [1-30]: ',0
.data:00406000                                         ; DATA XREF: sub_403721:loc_403722↑o
```

Figure 3. Student ID Prompt Disassembly

- Here, I found the procedure for the Student ID Prompt. There is a comparison instruction (**cmp eax, 0**) that could be modified, but I need more clues.

```
loc_403722:
mov     edx, offset aEnterStudentId ; "Enter student ID [1-30]: "
call    sub_4010C8
call    sub_401037
mov     dword_406159, eax
call    sub_4037AB
cmp     eax, 0
jz      short loc_403722
```

Figure 4. Student ID Prompt Cross Reference

- Next, I examined the procedures that are called by the call instructions. I discovered a procedure related to the ID check range, represented by **sub_4037AB**.

```
sub_4037AB proc near
mov     eax, 1
cmp     eax, dword_406159
jg      short loc_4037CB

mov     eax, 1Eh
cmp     eax, dword_406159
jl      short loc_4037CB

mov     eax, 1
retn

loc_4037CB:
mov     edx, offset aIncorrectStude ; "Incorrect student ID"
call    sub_4010C8
call    sub_40113B
xor     eax, eax
retn
sub_4037AB endp
```

Figure 5. Student ID Check Procedure

I identified this procedure as the ID range check because it includes comparison instructions involving the values 1 and 30 (indicated by 1Eh) and is linked to the incorrect student ID message. In this procedure, there are **mov eax, 1** and **xor eax, eax** instructions before the return instruction, which hint at what needs to be modified to

bypass the check. These instructions mean that the value of eax is returned as either 1 or 0 (from **xor eax,eax**). As we can see, xor eax,eax is executed after the incorrect student ID message. Thus, 0 indicates an invalid ID, and 1 indicates a valid ID.

After identifying these clues, I referred to Figure 3 and noticed the **jz**, which jumps back to the student ID prompt. Therefore, I decided to modify jz into jl because the program compares eax against 0. By assigning jl, the program will continue to another procedure since neither 0 nor 1 can be less than 0.

- To modify the instruction, go to **Edit → Patch Program → Assemble**.

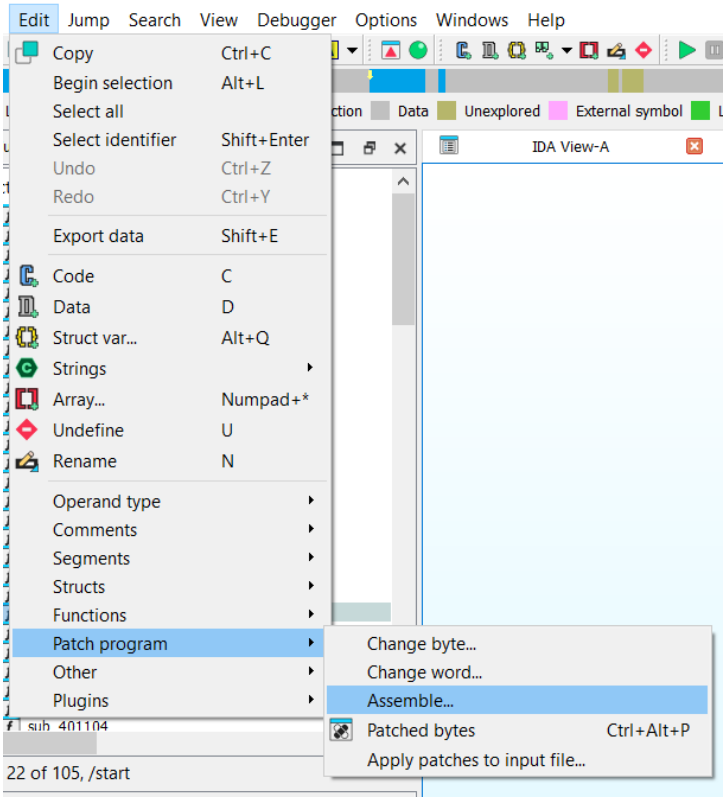

Figure 6. Edit Program by Changing the Instruction

Then, I changed the instruction from **jz** to **jl**. Once done, I clicked **OK**.

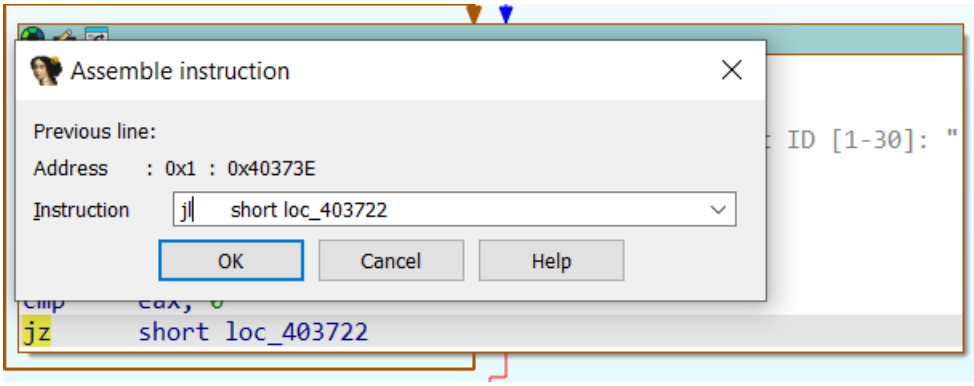

Figure 7. Instruction to Bypass ID Check Range

Sub-Task 2 – Saving Unencrypted Password to File

- In this sub-task, I used the same method as before, searching for clues in the strings view. In the figure below, I looked for the pre-defined password (student123).

```
.data:0040...  00000035  C  Your data has been successfully written to a file!\r\n
.data:0040...  00000022  C  To exit the program, press Enter!
.data:0040...  0000000A  C  logged in
.data:0040...  0000000B  C  student123
.data:0040...  0000001B  C  %0.2d/%0.2d/%d %0.2d:%0.2d
```

Figure 8. Password Strings View

- At the disassembly view, I discovered the cross-reference of the predefined password.

```
.data:00406144                                          ; sub_403836+2A4↑o
.data:0040614E aStudent123      db 'student123',0       ; DATA XREF: sub_4037DD+2↑o
.data:00406159 dword_406159     dd 0                    ; DATA XREF: sub_403721+10↑w
```

Figure 9. Pre-defined Password Disassembly View

- Inside the procedure, there is an instruction to compare the strings (**cmpsb**) and the invalid password message. However, we will ignore the left block containing the invalid password message since we are not attempting to bypass the password. Therefore, I searched all the procedures that are being called in the right block.

```
sub_4037DD proc near
pusha
cld
mov     esi, offset aStudent123 ; "student123"
mov     eax, offset unk_40637C
mov     edi, eax
mov     ecx, 0Bh
repe cmpsb
jecxz   short loc_40380F
```

```
mov     edx, offset aIncorrectPassw ; "Incorrect password, please try again"
call    sub_4010C8
call    sub_40113B
call    sub_403742
call    sub_403B65
popa
retn
```

```
loc_40380F:
push    offset unk_40637C
call    sub_403C85
mov     nNumberOfBytesToWrite, ecx
push    nNumberOfBytesToWrite
push    offset unk_40637C
call    sub_403B90
call    sub_403836
sub_4037DD endp
```

Figure 10. Password Check Procedure

- Then, I found a procedure called **sub_403B90**. It contains a loop instruction that alters every character in the esi register. It utilizes **xor** to transform a character against **0FAh** to another character. This represents a XOR encryption method applied to the program.

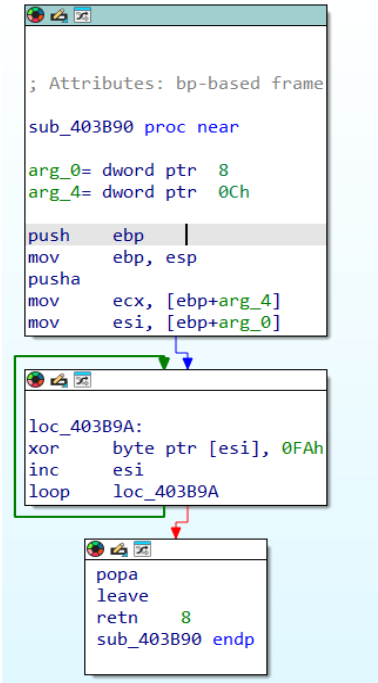

Figure 11. XOR Encryption Procedure

To disable the encryption procedure, we need to change **0FAh** to **0**. Zero in XOR operation is the solution to disable the encryption because when we compare a bit with 0, the result will be the bit itself. Consequently, no bits are changed.

- To change the byte in IDA, Select **Edit → Patch Program**, and choose **Change byte**.

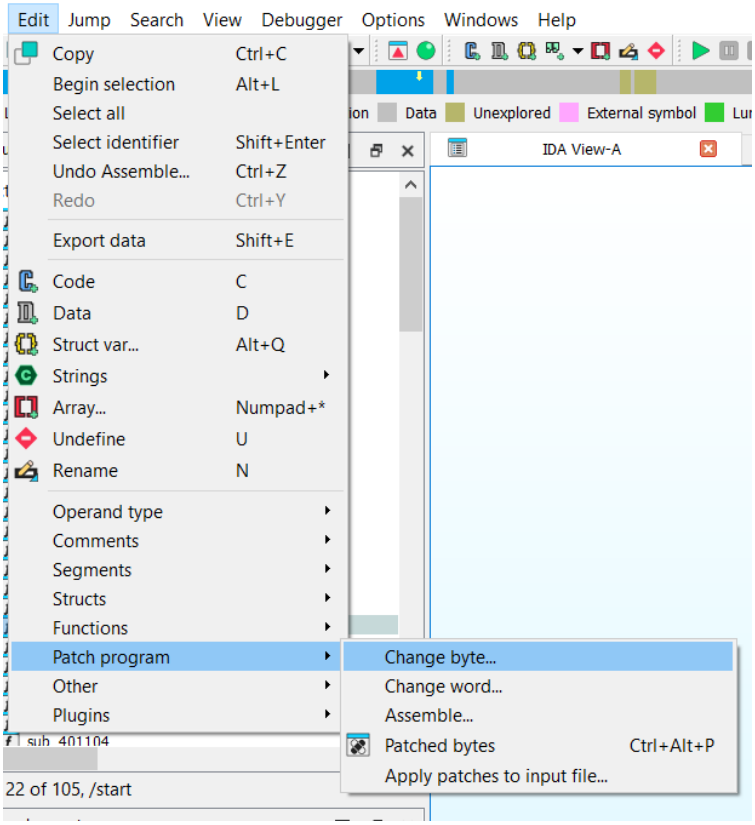

Figure 12. Changing Byte in IDA

- Then, change the FA byte to 00. The FA byte that needs to be changed is the first one because the **xor** and **byte ptr [esi]** are represented by the bytes "**80 36**".

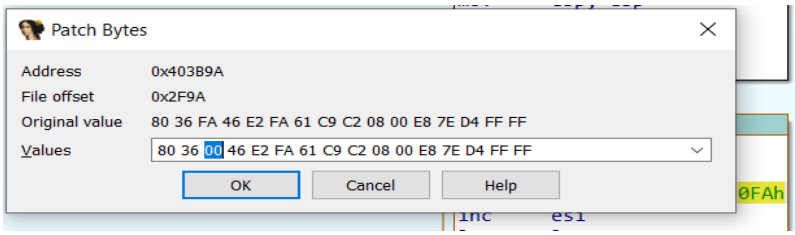

Figure 13. Changing Byte Value

- The figure below is the result of modification.

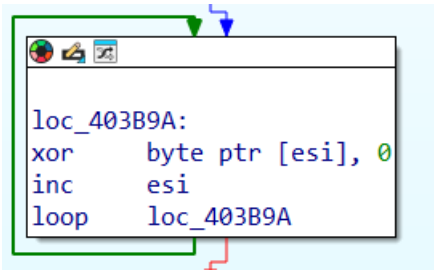

Figure 14. Byte Changes Result

Sub-Task 3 – Display a Warning Message on the Console

- When I was looking at the strings view, I found an exit program message. Then, I followed it to its procedure.

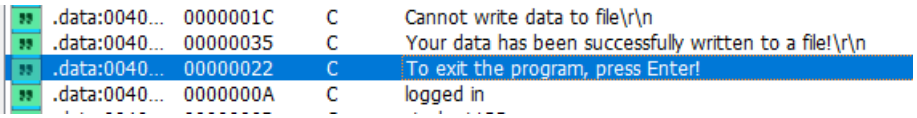

Figure 15. Exit Program Message String

- Inside the procedure, there is another feedback message before the exit program message from the **aYourDataHasBee** variable. Consequently, I thought the data inside this variable could be changed to a crafted warning message.

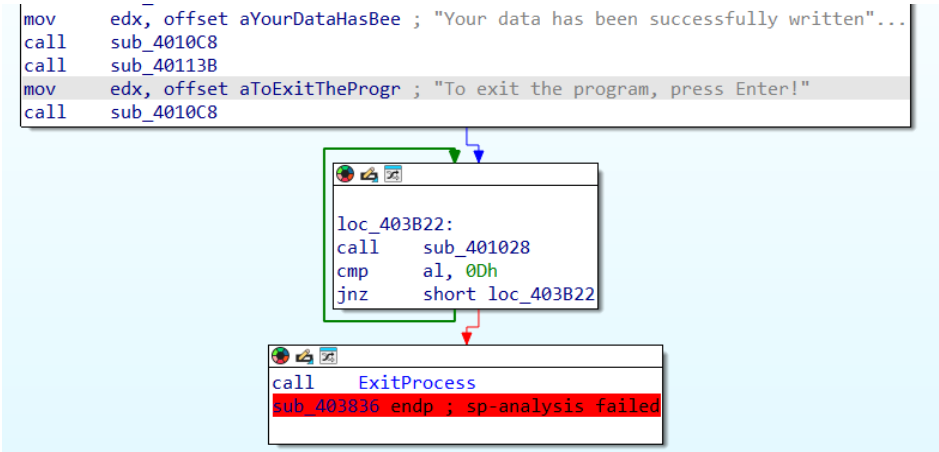

Figure 16. Write Data to File Procedure

- I double-clicked the variable to view its disassembly.

```
.data:004060D1                                          ; DATA XREF: sub_403BBE:loc_403C6B↑o
.data:004060ED aYourDataHasBee db 'Your data has been successfully written to a file!',0Dh,0Ah,0
.data:004060ED                                          ; DATA XREF: sub_403836+2D3↑o
```

Figure 17. Written Data Message Disassembly

- To modify the data, I found that the string should first be converted into a data format. Subsequently, right-click on the string, select **Data** and click **Yes** if a confirmation prompt appears.

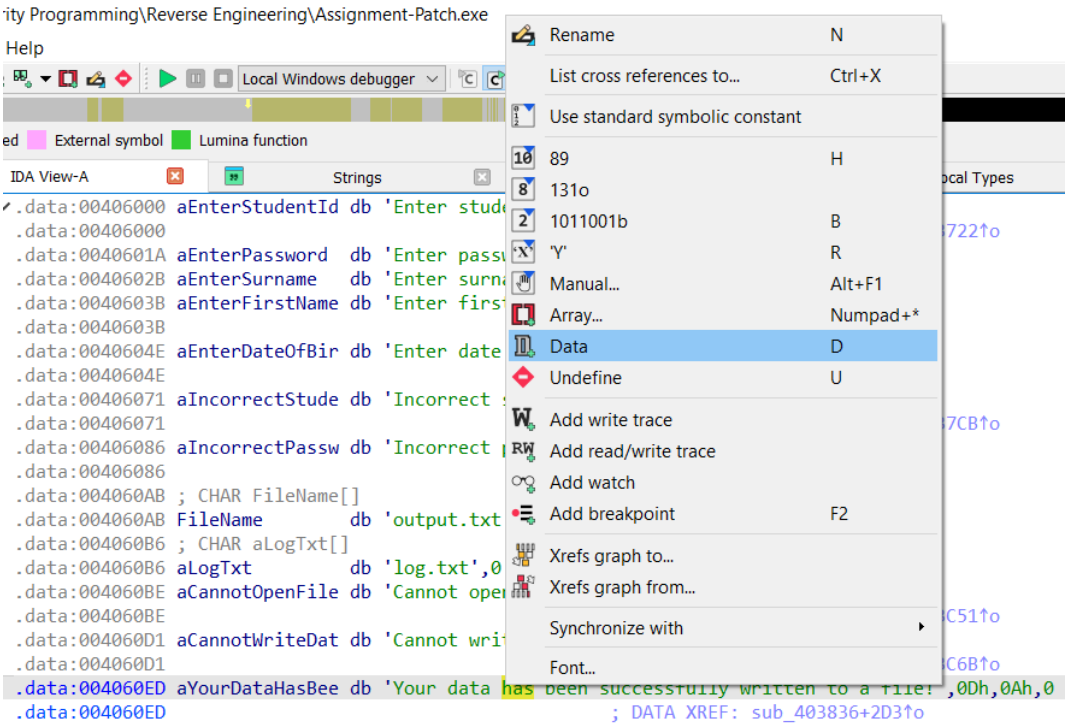

Figure 18. Change String to Data

The data will look like the figure below.

```
.data:004060ED byte_4060ED      db 59h                  ; DATA XREF: sub_403836+2D3↑o
.data:004060EE                  db  6Fh ; o
.data:004060EF                  db  75h ; u
.data:004060F0                  db  72h ; r
.data:004060F1                  db  20h
.data:004060F2                  db  64h ; d
.data:004060F3                  db  61h ; a
.data:004060F4                  db  74h ; t
.data:004060F5                  db  61h ; a
.data:004060F6                  db  20h
.data:004060F7                  db  68h ; h
.data:004060F8                  db  61h ; a
.data:004060F9                  db  73h ; s
.data:004060FA                  db  20h
.data:004060FB                  db  62h ; b
.data:004060FC                  db  65h ; e
.data:004060FD                  db  65h ; e
.data:004060FE                  db  6Eh ; n
.data:004060FF                  db  20h
.data:00406100                  db  73h ; s
```

Figure 19. String in Data Format

- The message we want to display on the console is “Your information has been disclosed!”, but we cannot put the string directly into the data. We need to convert it first to hexadecimal format. I used RapidTables website to quickly convert the string.

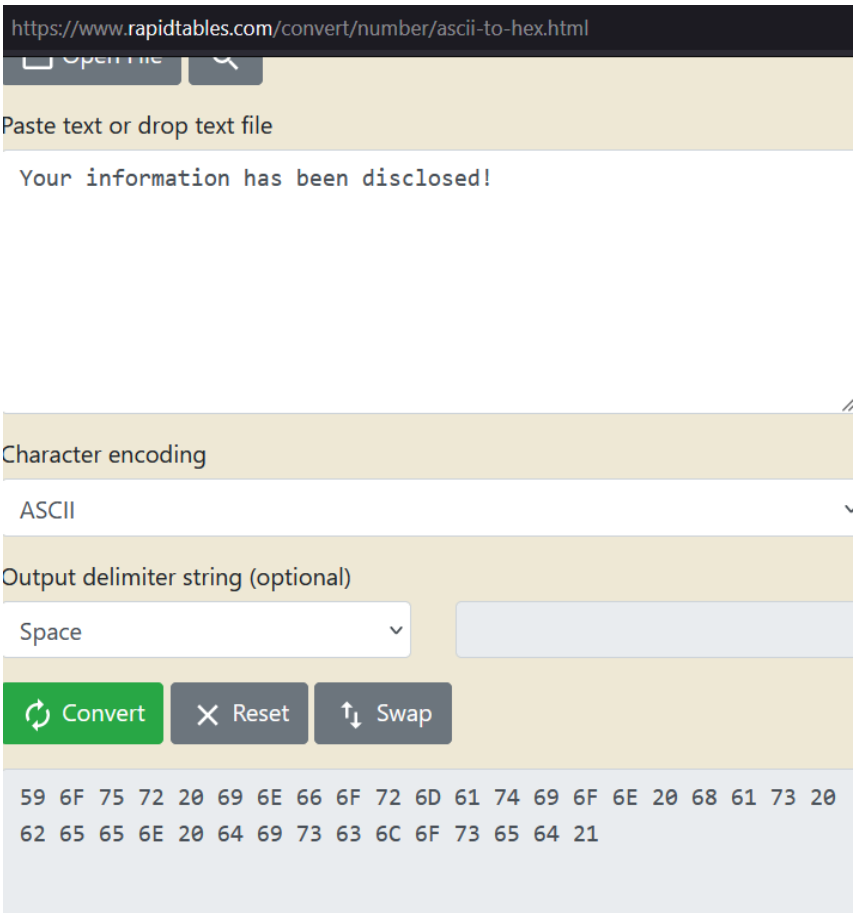

Figure 20. ASCII to Hexadecimal Conversion

- After that, we can change the bytes of the string using the same method on sub-task 2. The patching bytes can only accommodate 16 bytes per patch, so we will need to perform about four patches to modify the entire string.

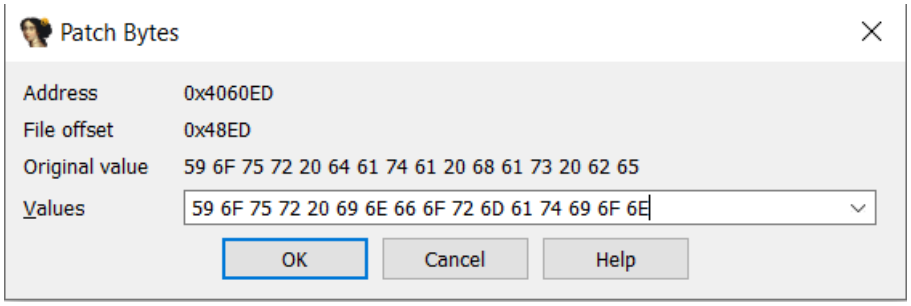

Figure 21. First Patch

- Then, we continue with the second patch by highlighting the next byte to be patched.

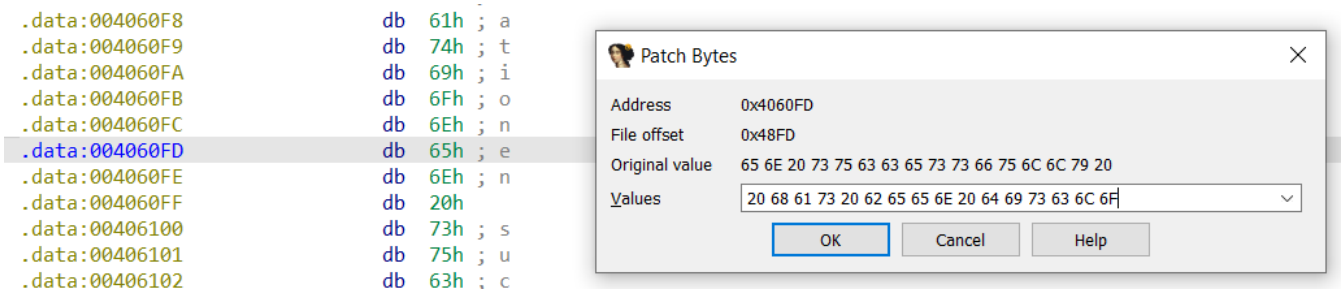

Figure 22. Second Patch

- For the third patch, we need to fill the bytes following the string with **00** to indicate that the string has ended and to prevent the printing of any additional characters beyond the intended string. I added "**0D 0A**" to print a new line.

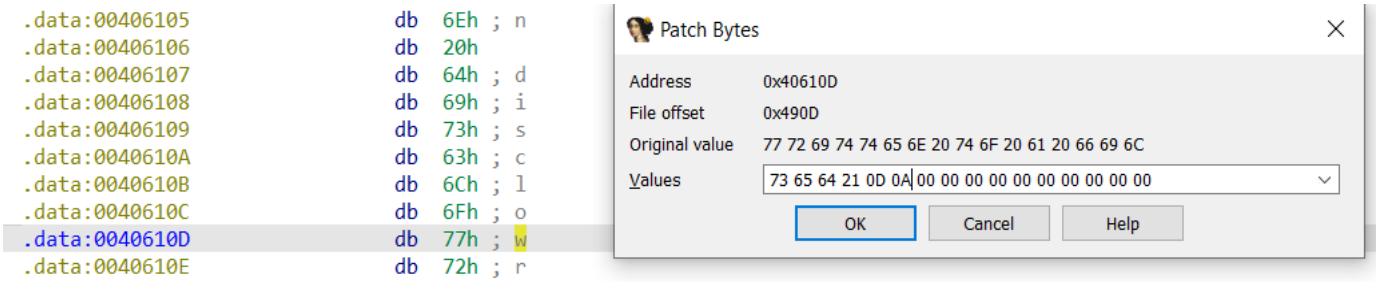

Figure 23. Third Patch

- For the fourth patch, I only changed some bytes before byte **00** because after that, it belongs to another variable.

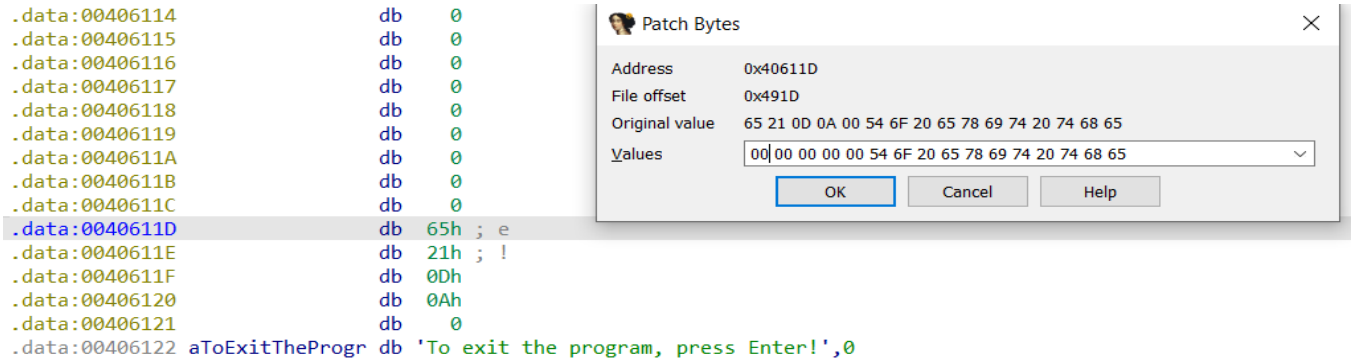

Figure 24. Fourth Patch

- Finally, return the format back to a string again. This can be done by selecting the data, right-clicking on it, and then choosing the strings logo (indicated by double quotes). The result can be seen on figure 26.

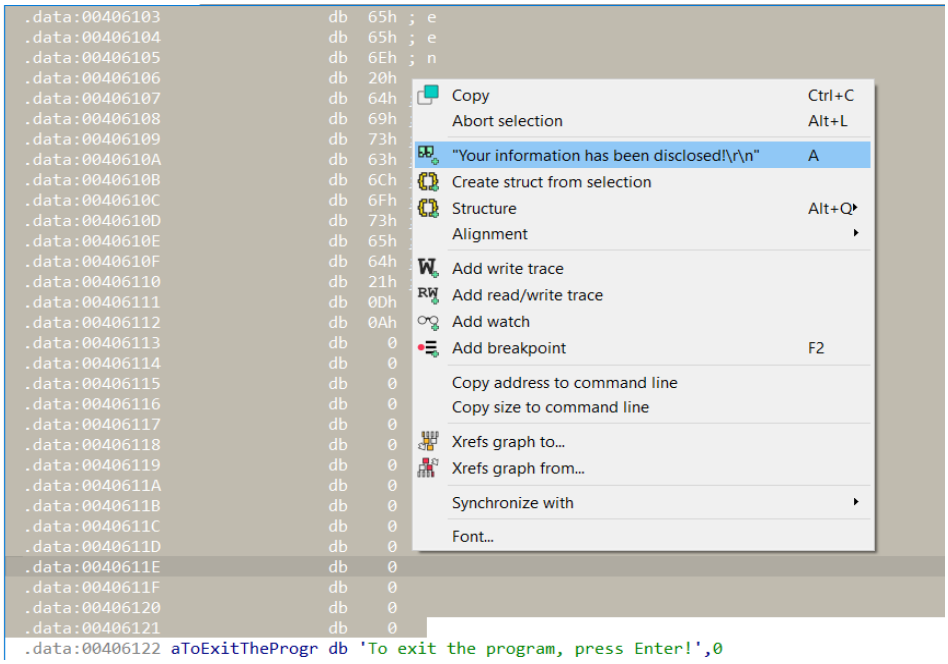

Figure 25. Changing Data to String Format

```
.data:004060ED aYourInformatio db 'Your information has been disclosed!',0Dh,0Ah
.data:004060ED                                          ; DATA XREF: sub_403836+2D3↑o
.data:00406113                 db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Figure 26. Modified String

Apply All Patches

The final step is to apply all patches to the application. This can be done by navigating to **Edit → Patch Program**, and selecting **Apply patches to input file**. When a prompt appears, click **OK**.

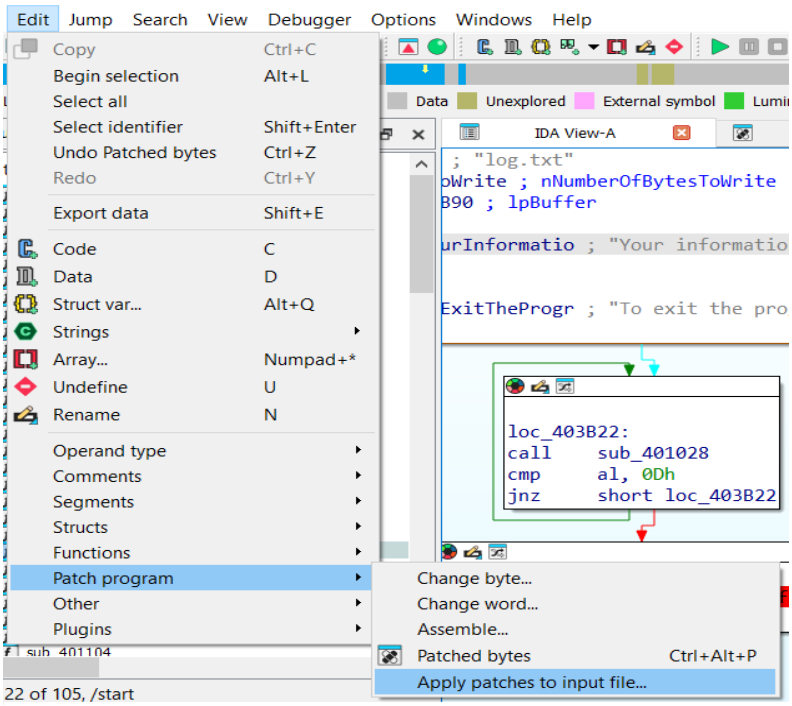

Figure 27. Apply Patches

Modification Results

- Bypassing student ID check range: The program can accept IDs outside of the expected range, even though displaying an invalid student ID message.

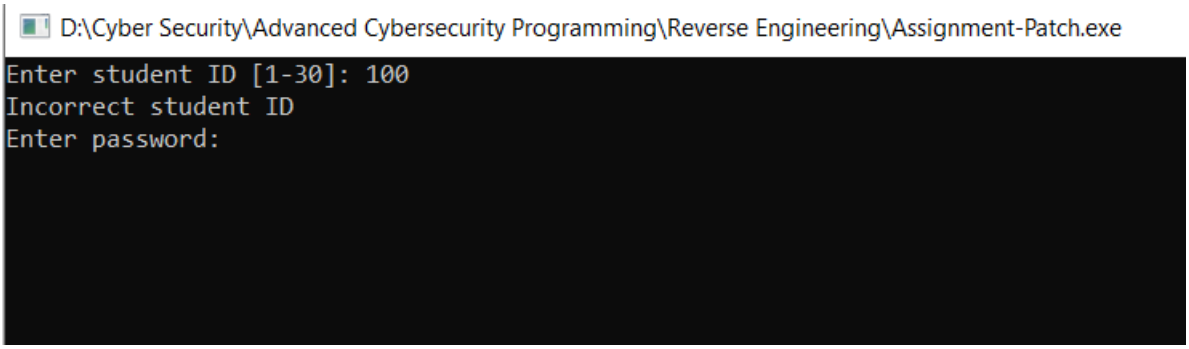

Figure 28. Bypass Result

- Saving unencrypted password to file: the modification successfully saves the unencrypted password to the output file.

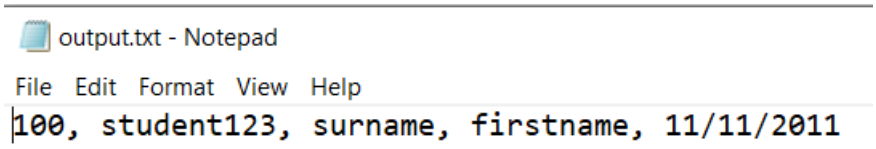

Figure 29. Save Unencrypted Password Result

- Display a warning message on the console: the message "Your data has been successfully written to a file" has been changed to "Your information has been disclosed!" in the modified program.

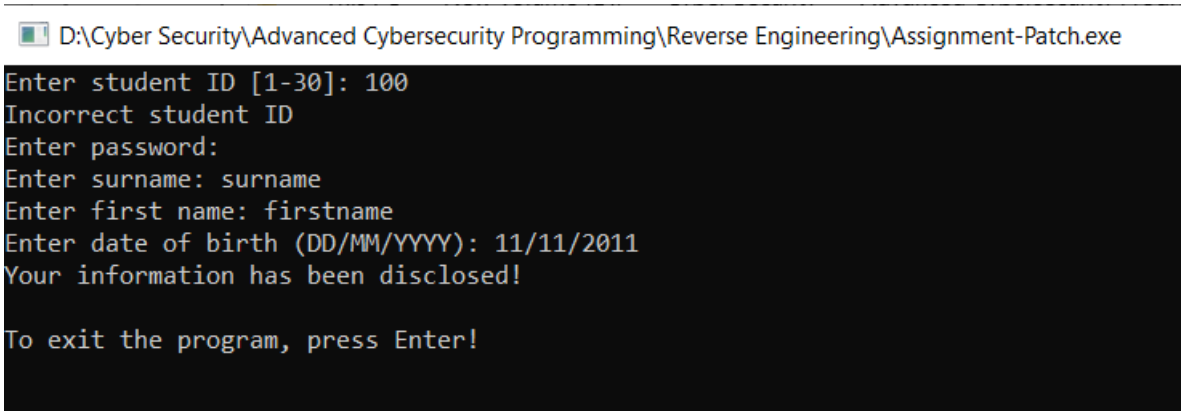

Figure 30. Warning Message Result