CS165 Project 2 - Reverse Engineering

Fall 2021
Professor Zhiyun Qian
University of California, Riverside.

By Sanchit Goel and Jiacheng Hou.

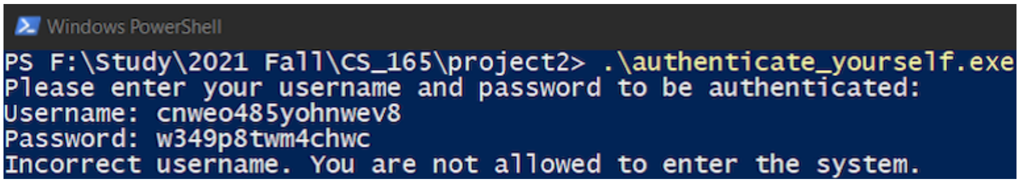
Part 1 - Authenticate Yourself

In this part, our main goal is to bypass the authentication used in a toy application created for this class and get a unique flag string using the disassembler IDA.

Program Assessment

Running the program

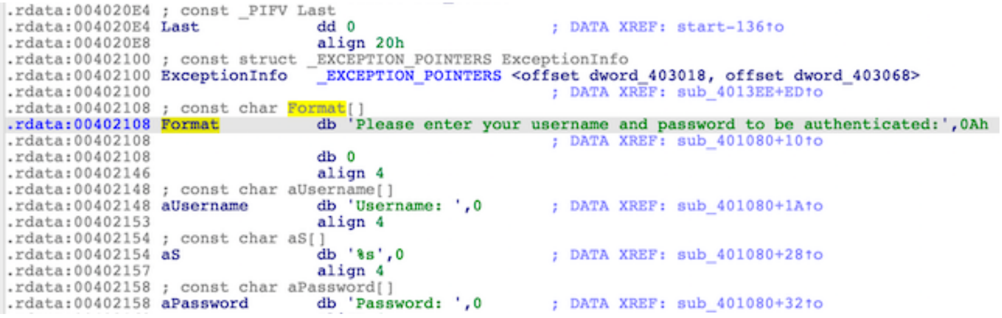
Running the program in PowerShell, and entering random strings as username and password, we got:



```
PS F:\Study\2021 Fall\CS_165\project2> .\authenticate_yourself.exe
Please enter your username and password to be authenticated:
Username: cnweo485yohnwev8
Password: w349p8twm4chwc
Incorrect username. You are not allowed to enter the system.
```

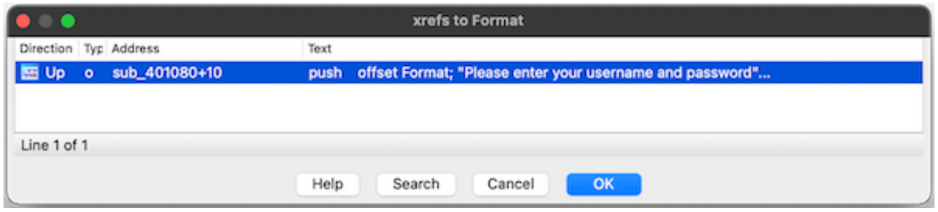
Locating the function

To locate the function where the authentication is performed, we searched "username" using the search text function of IDA and found the string 'Please enter your username and password to be authenticated:'.



```
.rdata:004020E4 ; const _PIFV Last
.rdata:004020E4 Last            dd 0                    ; DATA XREF: start-136↑o
.rdata:004020E8                 align 20h
.rdata:00402100 ; const struct _EXCEPTION_POINTERS ExceptionInfo
.rdata:00402100 ExceptionInfo   _EXCEPTION_POINTERS <offset dword_403018, offset dword_403068>
.rdata:00402100                                         ; DATA XREF: sub_4013EE+ED↑o
.rdata:00402108 ; const char Format[]
.rdata:00402108 Format          db 'Please enter your username and password to be authenticated:',0Ah
.rdata:00402108                                         ; DATA XREF: sub_401080+10↑o
.rdata:00402108                 db 0
.rdata:00402146                 align 4
.rdata:00402148 ; const char aUsername[]
.rdata:00402148 aUsername       db 'Username: ',0       ; DATA XREF: sub_401080+1A↑o
.rdata:00402153                 align 4
.rdata:00402154 ; const char aS[]
.rdata:00402154 aS              db '%s',0               ; DATA XREF: sub_401080+28↑o
.rdata:00402157                 align 4
.rdata:00402158 ; const char aPassword[]
.rdata:00402158 aPassword       db 'Password: ',0       ; DATA XREF: sub_401080+32↑o
```

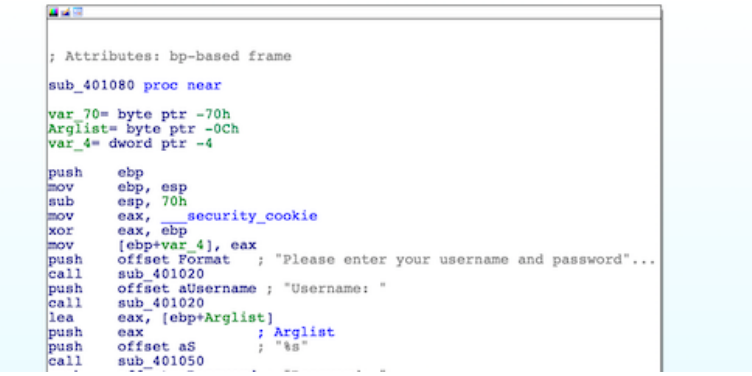
Then we use "Jump to xref operand..." and saw the string is used in address `sub_401080+10`



Direction	Typ	Address	Text
Up	o	sub_401080+10	push offset Format; "Please enter your username and password"...

Line 1 of 1

And clicking on "OK" brought us to function `sub_401080` and this is the function that performs the authentication.



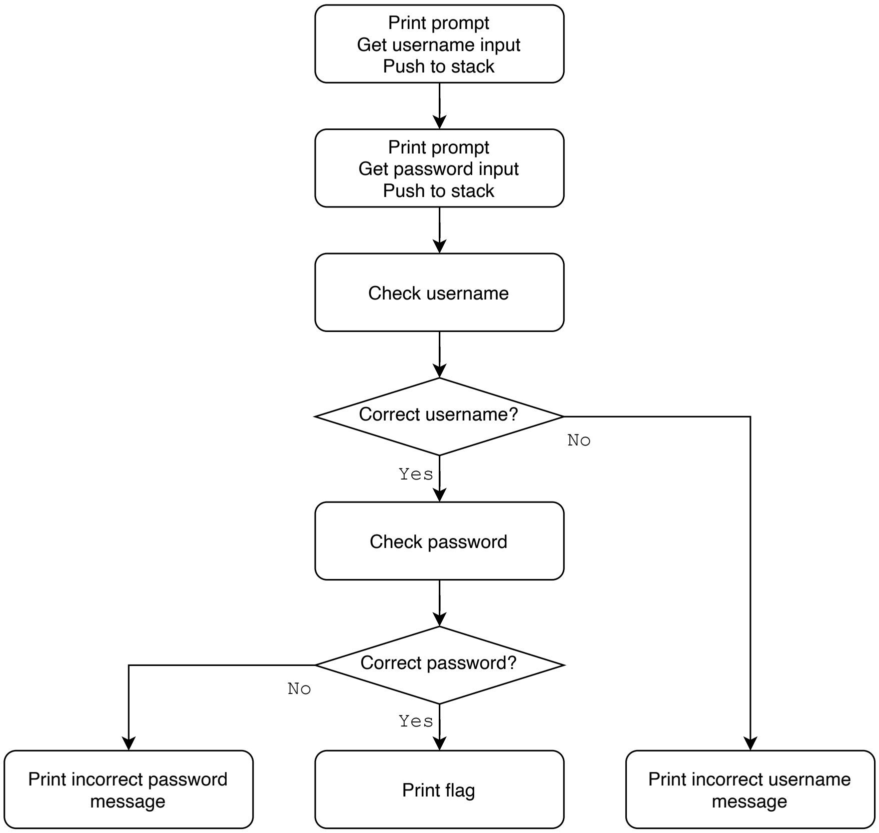
```
; Attributes: bp-based frame

sub_401080 proc near

var_70= byte ptr -70h
Arglist= byte ptr -0Ch
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 70h
mov     eax, ___security_cookie
xor     eax, ebp
mov     [ebp+var_4], eax
push    offset Format   ; "Please enter your username and password"...
call    sub_401020
push    offset aUsername ; "Username: "
call    sub_401020
lea     eax, [ebp+Arglist]
push    eax             ; Arglist
push    offset aS       ; "%s"
call    sub_401050
```

Now we can work on bypassing the authentication.

Implementation

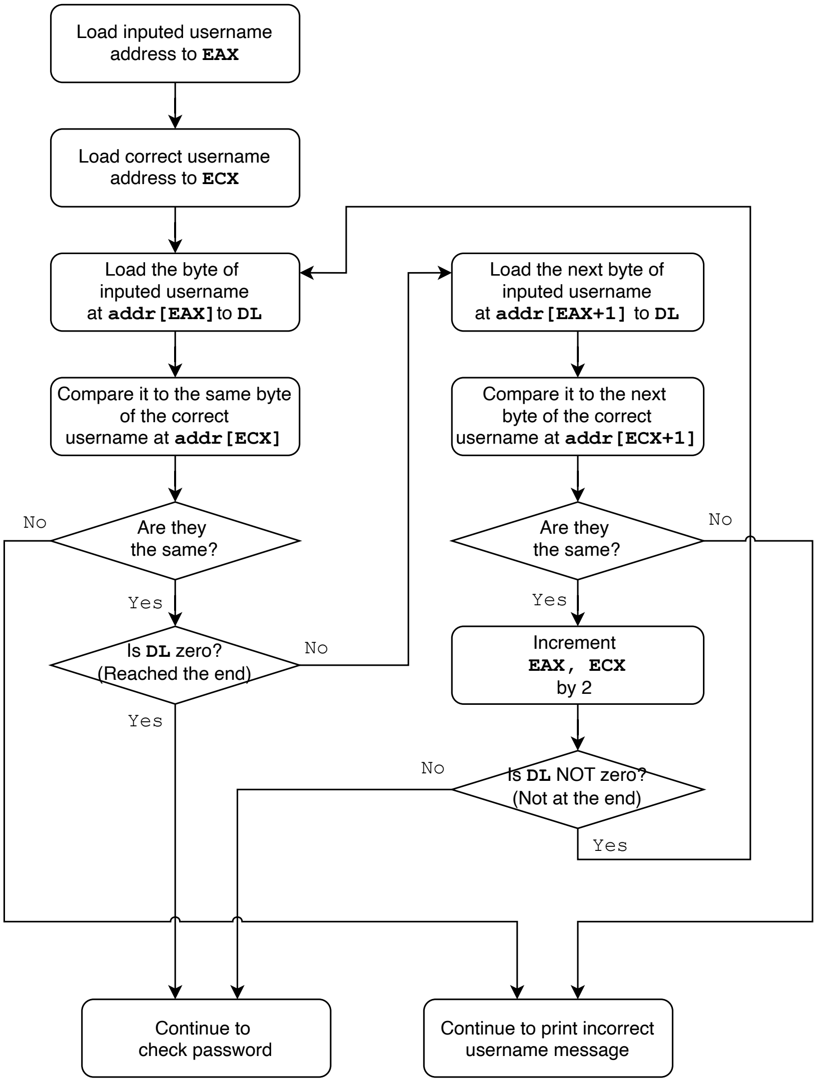
Overview

Here is an overview of the logic of the function:



Bypassing Username Check

Here is a closer look on how the function checks if the user enters the correct username:


Load inputed username address to EAX
Load correct username address to ECX
Load the byte of inputed username at addr[EAX] to DL
Compare it to the same byte of the correct username at addr[ECX]
Are they the same?
No
Yes
Is DL zero? (Reached the end)
No
Yes
Load the next byte of inputed username at addr[EAX+1] to DL
Compare it to the next byte of the correct username at addr[ECX+1]
Are they the same?
No
Yes
Increment EAX, ECX by 2
Is DL NOT zero? (Not at the end)
No
Yes
Continue to check password
Continue to print incorrect username message

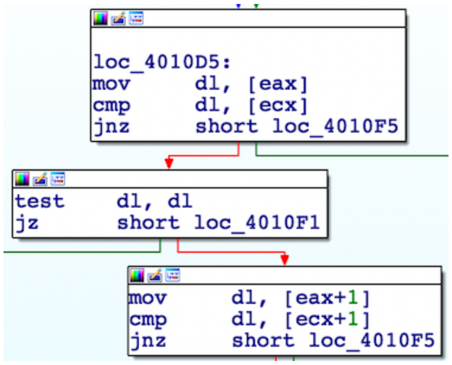
We can see, the function compares the inputted username against the correct username Using a loop. Each loop, it checks if the bytes are the same and if the byte in the inputted username is zero (indication it has reached the end.) [1]

Using "Patch Program" function in IDA, we can edit the program and apply our changes.

To bypass the username check, instead of comparing the byte from the inputted username in `DL` to the byte from the correct username at `addr[ECX]`, we changed it to comparing to itself, which always yields true.

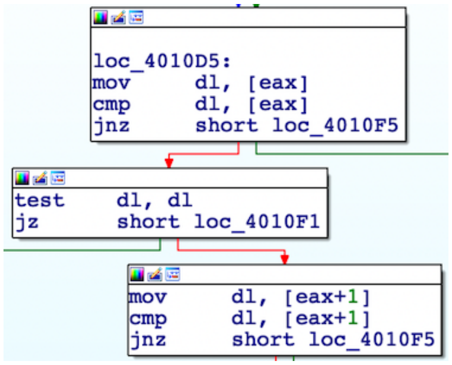
Similarly, we changed the comparison on the next byte of the username to always be true.

Before



```
loc_4010D5:
mov     dl, [eax]
cmp     dl, [ecx]
jnz     short loc_4010F5

test    dl, dl
jz      short loc_4010F1

mov     dl, [eax+1]
cmp     dl, [ecx+1]
jnz     short loc_4010F5
```

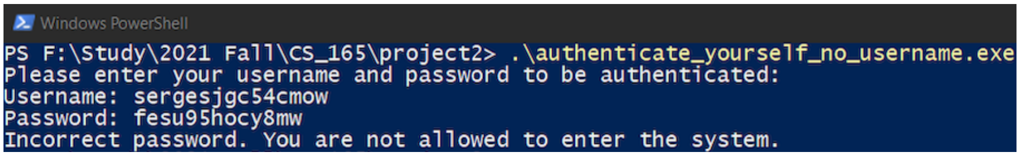
After



```
loc_4010D5:
mov     dl, [eax]
cmp     dl, [eax]
jnz     short loc_4010F5

test    dl, dl
jz      short loc_4010F1

mov     dl, [eax+1]
cmp     dl, [eax+1]
jnz     short loc_4010F5
```

By making the changes above, we successfully bypassed the program's username check and the program now always accepts the inputted username.

After entering a random string as username, we got:



```
PS F:\Study\2021 Fall\CS_165\project2> .\authenticate_yourself_no_username.exe
Please enter your username and password to be authenticated:
Username: sergesjgc54cmow
Password: fesu95hocy8mw
Incorrect password. You are not allowed to enter the system.
```

Next was to bypass password check.

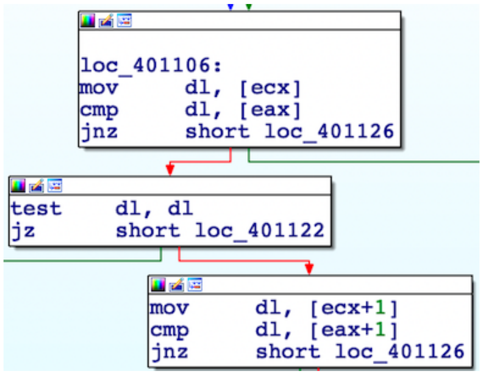
Bypassing Password Check

We found the mechanism for checking password is the same as for checking username, with only some differences:

1. Load the correct password address and inputted password into EAX and ECX respectively.
2. If the bytes are different, then print incorrect password message.
3. If the end of the inputted password is reached and all its bytes are the same as those of the correct one, the flag string is printed.

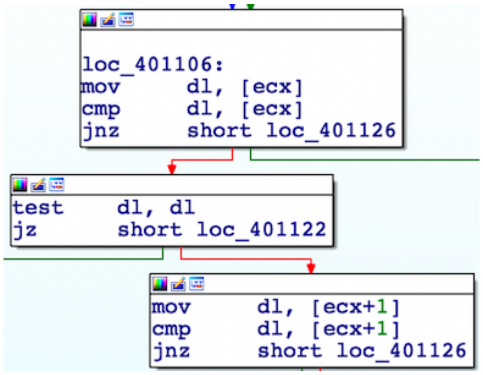
We made the similar changes to the comparisons so they would always be true.

Before



```
loc_401106:
mov     dl, [ecx]
cmp     dl, [eax]
jnz     short loc_401126

test    dl, dl
jz      short loc_401122

mov     dl, [ecx+1]
cmp     dl, [eax+1]
jnz     short loc_401126
```

After



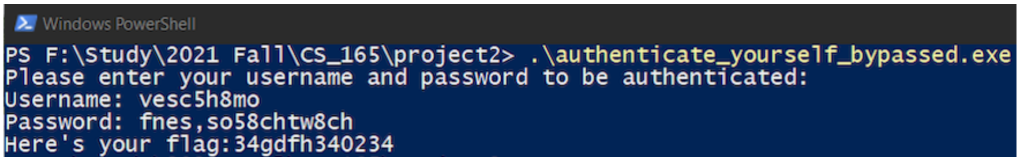
```
loc_401106:
mov     dl, [ecx]
cmp     dl, [ecx]
jnz     short loc_401126

test    dl, dl
jz      short loc_401122

mov     dl, [ecx+1]
cmp     dl, [ecx+1]
jnz     short loc_401126
```

We successfully bypassed the program's password check as well.

Outcome

Since both username and password checks are bypassed, the toy program's entire authentication is bypassed. The program now gives out the flag string no matter what username or password we give.

Running the patched program with random strings as username and password, we got:



```
PS F:\Study\2021 Fall\CS_165\project2> .\authenticate_yourself_bypassed.exe
Please enter your username and password to be authenticated:
Username: vesc5h8mo
Password: fnes,so58chtw8ch
Here's your flag:34gdfh340234
```

The flag is:

```
34gdfh340234
```

[1] Thanks to John Dvorak at https://stackoverflow.com/a/13064985 for explaining the combination of TEST and JZ.

```
TEST    EAX, EAX
JZ      short loc_123456
```

This combo means "jump if EAX is zero." ↩