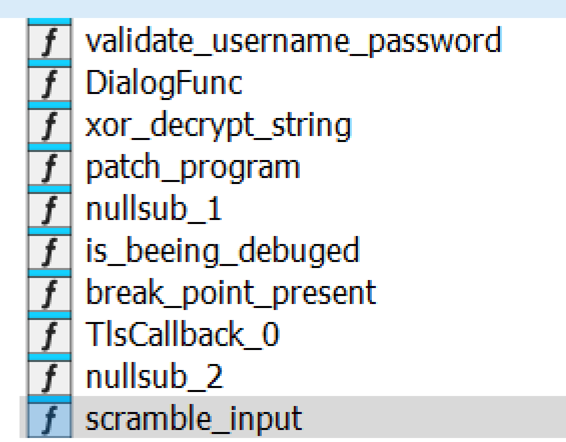
AntiDebug Assignment 06

- Daniel Attali 328780879

This assignment is to try and find the username and password combo that get us to the success screen.

In this assignment the first we have to understand what are the anti-debug behaviors and find a way to disable them.

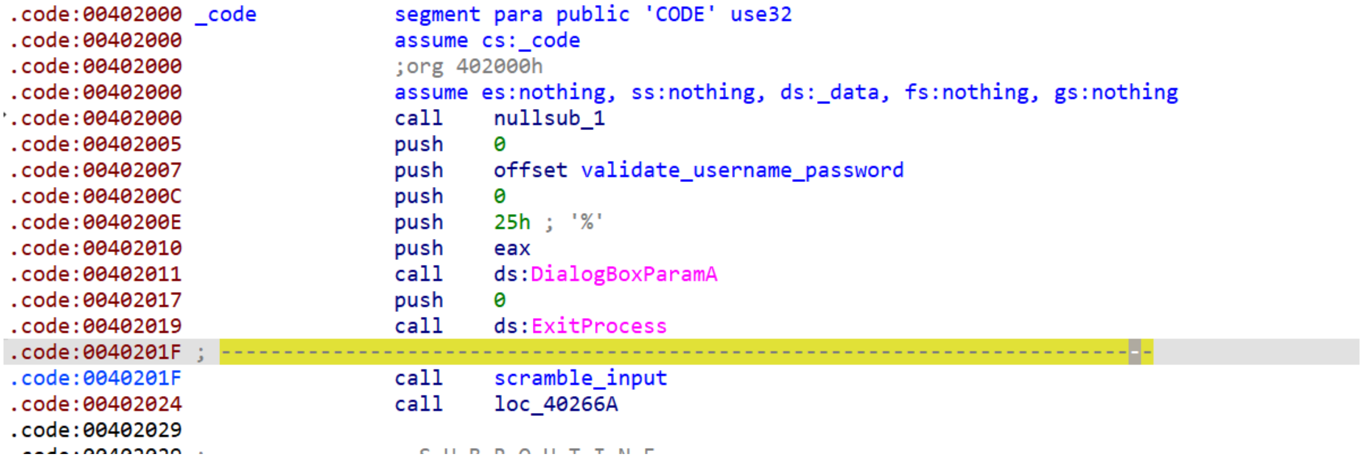
We started by finding all the function and giving them appropriate names to understand what is going on:



```
validate_username_password
DialogFunc
xor_decrypt_string
patch_program
nullsub_1
is_beeing_debuged
break_point_present
TlsCallback_0
nullsub_2
scramble_input
```

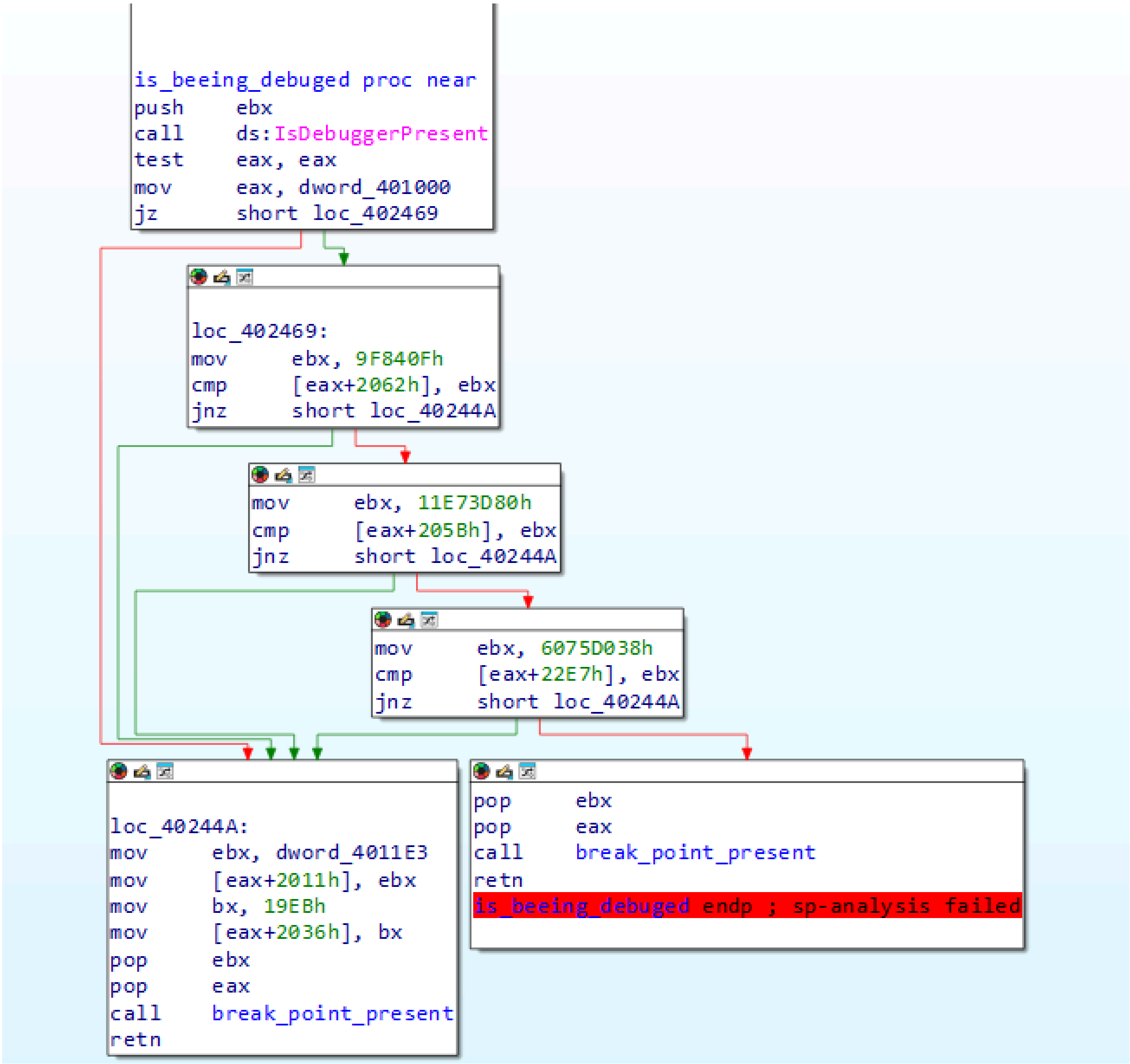
As we can see we found functions that validate the username and password one that uses some kind of xor encryption decryption scheme but also a function that we have found to check for break points in the code (i.e. `0xCC`) and also a function to check if the program is being debugged with all the anti-debug scheme we learned in class.

The program starts as follows:



```
.code:00402000 _code           segment para public 'CODE' use32
.code:00402000                 assume cs:_code
.code:00402000                 ;org 402000h
.code:00402000                 assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
.code:00402000                 call    nullsub_1
.code:00402005                 push    0
.code:00402007                 push    offset validate_username_password
.code:0040200C                 push    0
.code:0040200E                 push    25h ; '%'
.code:00402010                 push    eax
.code:00402011                 call    ds:DialogBoxParamA
.code:00402017                 push    0
.code:00402019                 call    ds:ExitProcess
.code:0040201F ; ---------------------------------------------------------------------------
.code:0040201F                 call    scramble_input
.code:00402024                 call    loc_40266A
.code:00402029
```

but when trying to put a break point and using dynamic analysis the call at `.code0040211` is change from `call ds:DialogBoxParamA` to `call ds:ExitProcess` so we understood that we need to find a way to either disable all the anti debug functions or to jump over them if possible.



```
is_beeing_debuged proc near
push    ebx
call    ds:IsDebuggerPresent
test    eax, eax
mov     eax, dword_401000
jz      short loc_402469
```

```
loc_402469:
mov     ebx, 9F840Fh
cmp     [eax+2062h], ebx
jnz     short loc_40244A
```

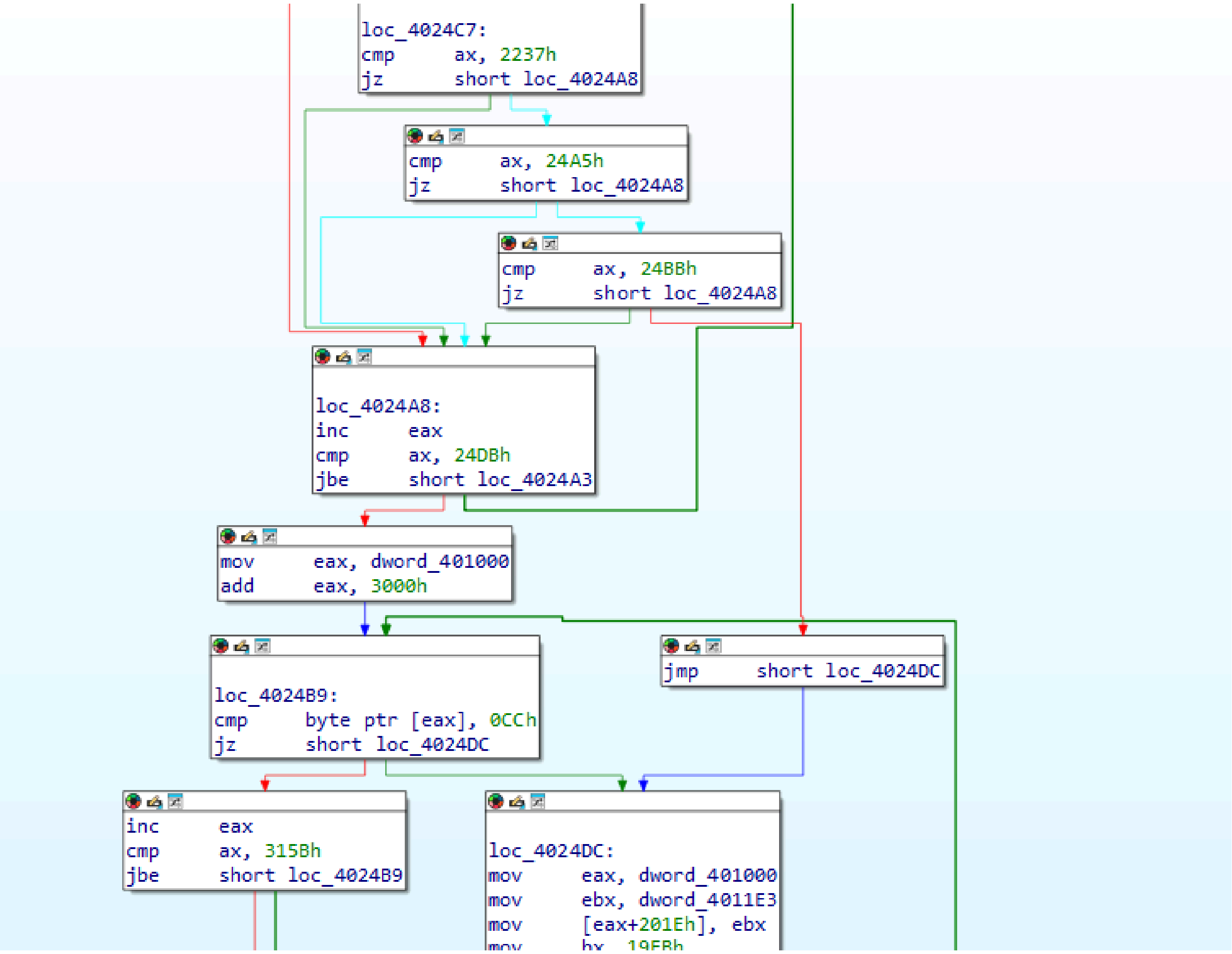
```
mov     ebx, 11E73D80h
cmp     [eax+205Bh], ebx
jnz     short loc_40244A
```

```
mov     ebx, 6075D038h
cmp     [eax+22E7h], ebx
jnz     short loc_40244A
```

```
loc_40244A:
mov     ebx, dword_4011E3
mov     [eax+2011h], ebx
mov     bx, 19EBh
mov     [eax+2036h], bx
pop     ebx
pop     eax
call    break_point_present
retn
```

```
pop     ebx
pop     eax
call    break_point_present
retn
is_beeing_debuged endp ; sp-analysis failed
```

The image above is the `is_beeing_debubed` function and has you can see it has a structure of `if` `else if` `else` program and we start by checking if the program is being debugged by calling the `IsDebuggerPresent` function and then we check the special `BYTE` present like we learn in class (i.e. the special byte in each program stub) and if so we call the `break_point_present` function.



```
loc_4024C7:
cmp     ax, 2237h
jz      short loc_4024A8
```

```
cmp     ax, 24A5h
jz      short loc_4024A8
```

```
cmp     ax, 24BBh
jz      short loc_4024A8
```

```
loc_4024A8:
inc     eax
cmp     ax, 24DBh
jbe     short loc_4024A3
```

```
mov     eax, dword_401000
add     eax, 3000h
```

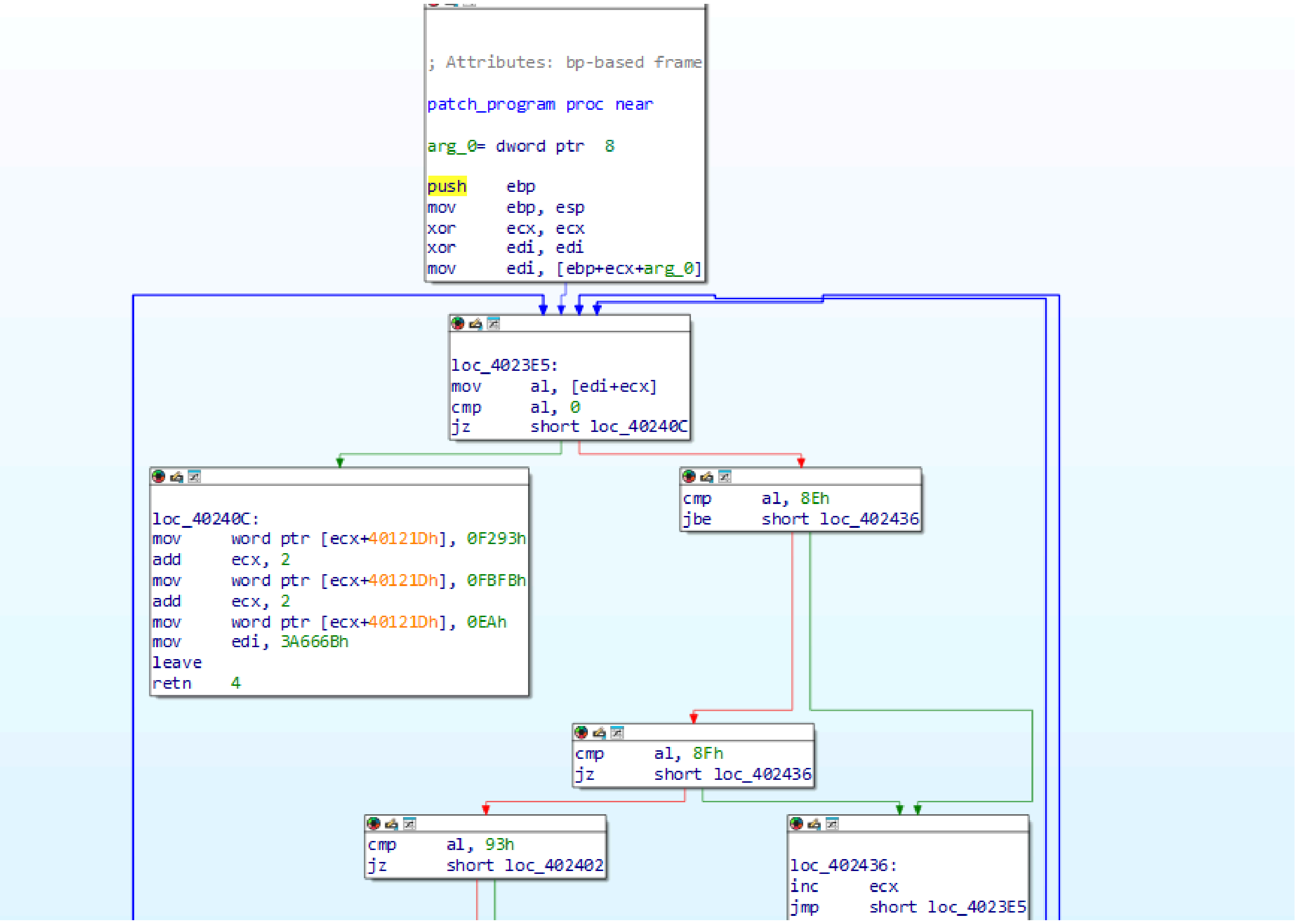
```
loc_4024B9:
cmp     byte ptr [eax], 0CCh
jz      short loc_4024DC
```

```
jmp     short loc_4024DC
```

```
inc     eax
cmp     ax, 315Bh
jbe     short loc_4024B9
```

```
loc_4024DC:
mov     eax, dword_401000
mov     ebx, dword_4011E3
mov     [eax+201Eh], ebx
mov     bx, 19EBh
```

This is part of the `is_breakpoint_present` function and has we can see we search the `exe` to check if there is a `0xCC` byte present and if so the program is overwriting some addresses with a value (in runtime we saw the this function patch that program to hide the function of message box etc.).



```
; Attributes: bp-based frame

patch_program proc near

arg_0= dword ptr  8

push    ebp
mov     ebp, esp
xor     ecx, ecx
xor     edi, edi
mov     edi, [ebp+ecx+arg_0]
```

```
loc_4023E5:
mov     al, [edi+ecx]
cmp     al, 0
jz      short loc_40240C
```

```
loc_40240C:
mov     word ptr [ecx+40121Dh], 0F293h
add     ecx, 2
mov     word ptr [ecx+40121Dh], 0FBFBh
add     ecx, 2
mov     word ptr [ecx+40121Dh], 0EAh
mov     edi, 3A666Bh
leave
retn    4
```

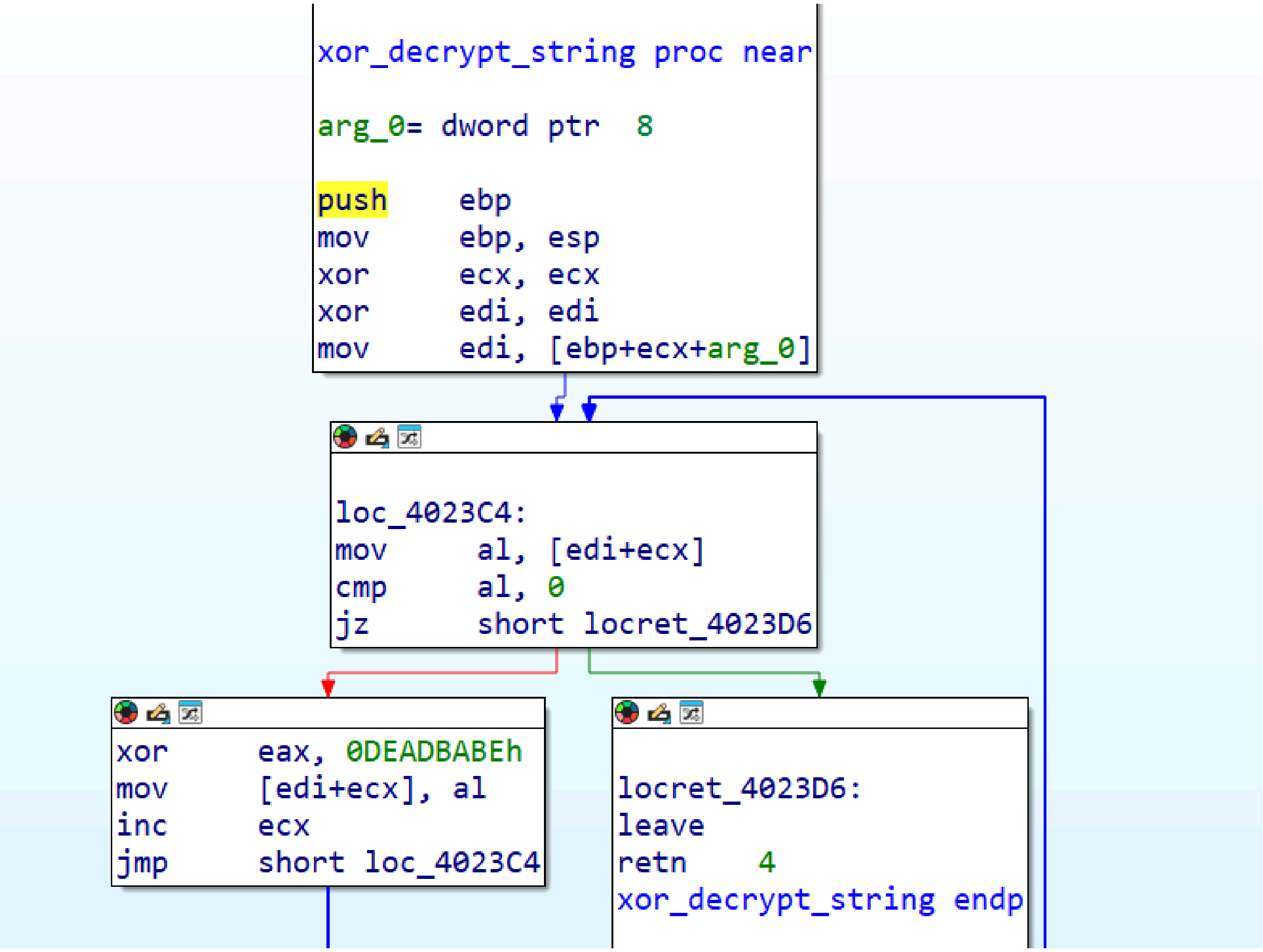
```
cmp     al, 8Eh
jbe     short loc_402436
```

```
cmp     al, 8Fh
jz      short loc_402436
```

```
cmp     al, 93h
jz      short loc_402402
```

```
loc_402436:
inc     ecx
jmp     short loc_4023E5
```

The image above is the `patch_program` function and we we can see it replaces value at `0x40121D` which are lines of code (assembly) in which we make certain function call.



```
xor_decrypt_string proc near

arg_0= dword ptr  8

push    ebp
mov     ebp, esp
xor     ecx, ecx
xor     edi, edi
mov     edi, [ebp+ecx+arg_0]
```

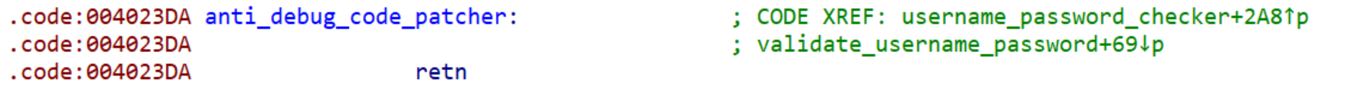
```
loc_4023C4:
mov     al, [edi+ecx]
cmp     al, 0
jz      short locret_4023D6
```

```
xor     eax, 0DEADBABEh
mov     [edi+ecx], al
inc     ecx
jmp     short loc_4023C4
```

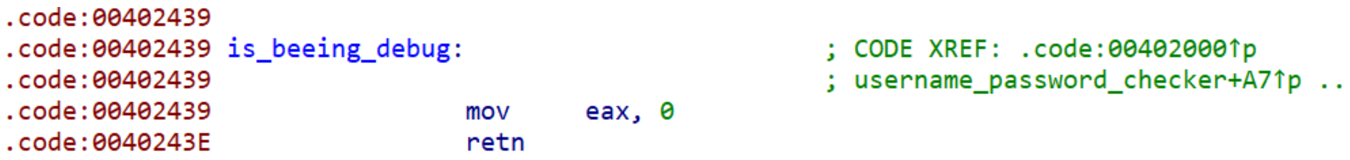
```
locret_4023D6:
leave
retn    4
xor_decrypt_string endp
```

The function above is the `xor_decrypt_string` function that uses some kind of xor scheme to shred the value at some point in memory.

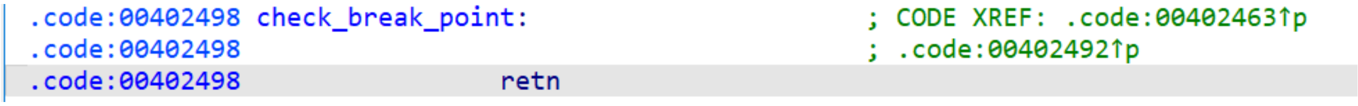
To remove any anti debug behavior we decided to patch the program our selves.



```
.code:004023DA anti_debug_code_patcher:                ; CODE XREF: username_password_checker+2A8↑p
.code:004023DA                                         ; validate_username_password+69↓p
.code:004023DA                 retn
```



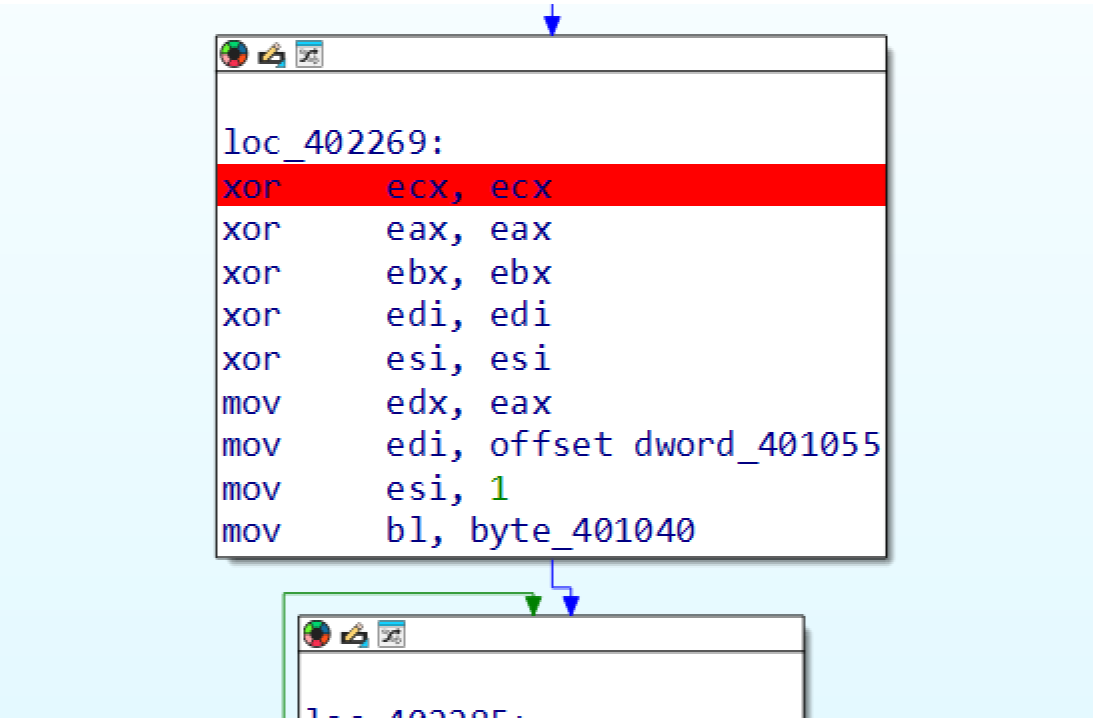
```
.code:00402439
.code:00402439 is_beeing_debug:                        ; CODE XREF: .code:00402000↑p
.code:00402439                                         ; username_password_checker+A7↑p ..
.code:00402439                 mov     eax, 0
.code:0040243E                 retn
```



```
.code:00402498 check_break_point:                      ; CODE XREF: .code:00402463↑p
.code:00402498                                         ; .code:00402492↑p
.code:00402498                 retn
```

This change make all the anti-debug function just do nothing.

next part was to use dynamic analysis to see what happens when we input something into the program:

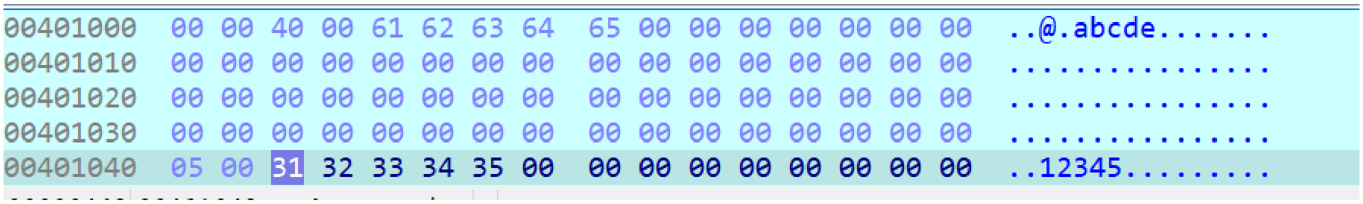


```
loc_402269:
xor     ecx, ecx
xor     eax, eax
xor     ebx, ebx
xor     edi, edi
xor     esi, esi
mov     edx, eax
mov     edi, offset dword_401055
mov     esi, 1
mov     bl, byte_401040
```

We putted some break point throughout the program and we inputted the following:

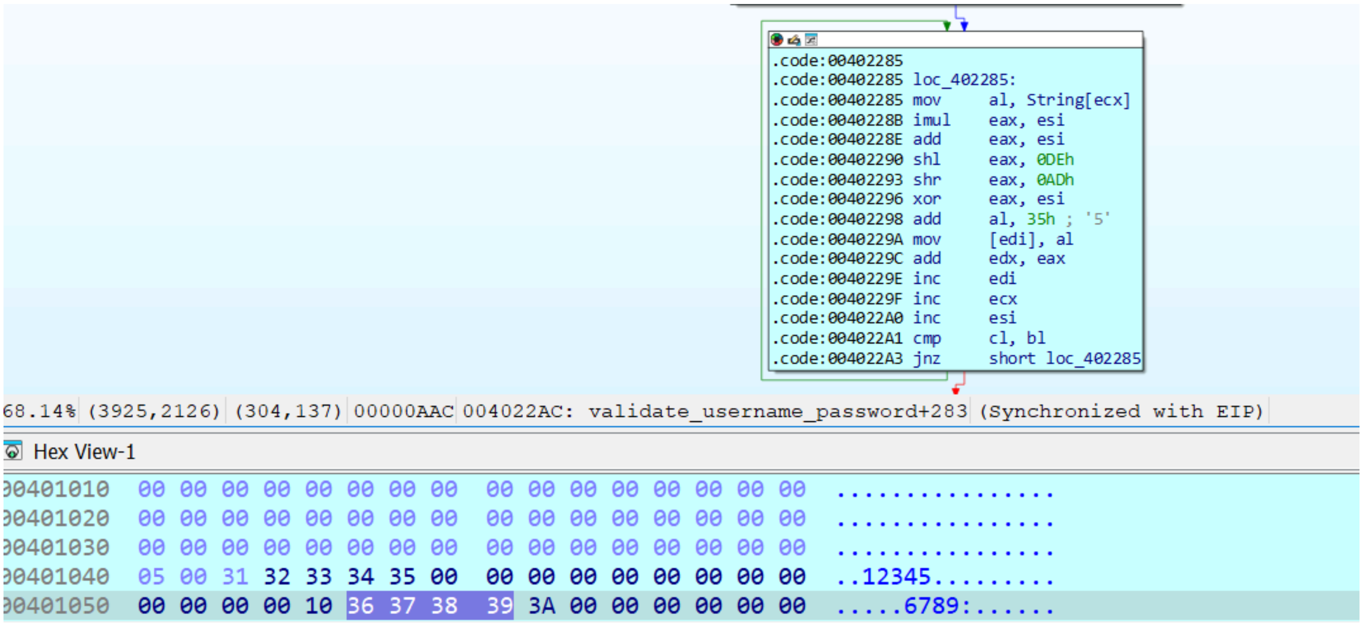
- `12345` as username
- `abcde` as password

And we ran the program to the break point and search in memory to see what is happening:



```
00401000  00 00 40 00 61 62 63 64  65 00 00 00 00 00 00 00  ..@.abcde.......
00401010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00401020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00401030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00401040  05 00 31 32 33 34 35 00  00 00 00 00 00 00 00 00  ..12345.........
```

We can see both the username and the password in memory has we inputted and we ran to program further



```
.code:00402285
.code:00402285 loc_402285:
.code:00402285 mov     al, String[ecx]
.code:0040228B imul    eax, esi
.code:0040228E add     eax, esi
.code:00402290 shl     eax, 0DEh
.code:00402293 shr     eax, 0ADh
.code:00402296 xor     eax, esi
.code:00402298 add     al, 35h ; '5'
.code:0040229A mov     [edi], al
.code:0040229C add     edx, eax
.code:0040229E inc     edi
.code:0040229F inc     ecx
.code:004022A0 inc     esi
.code:004022A1 cmp     cl, bl
.code:004022A3 jnz     short loc_402285
```

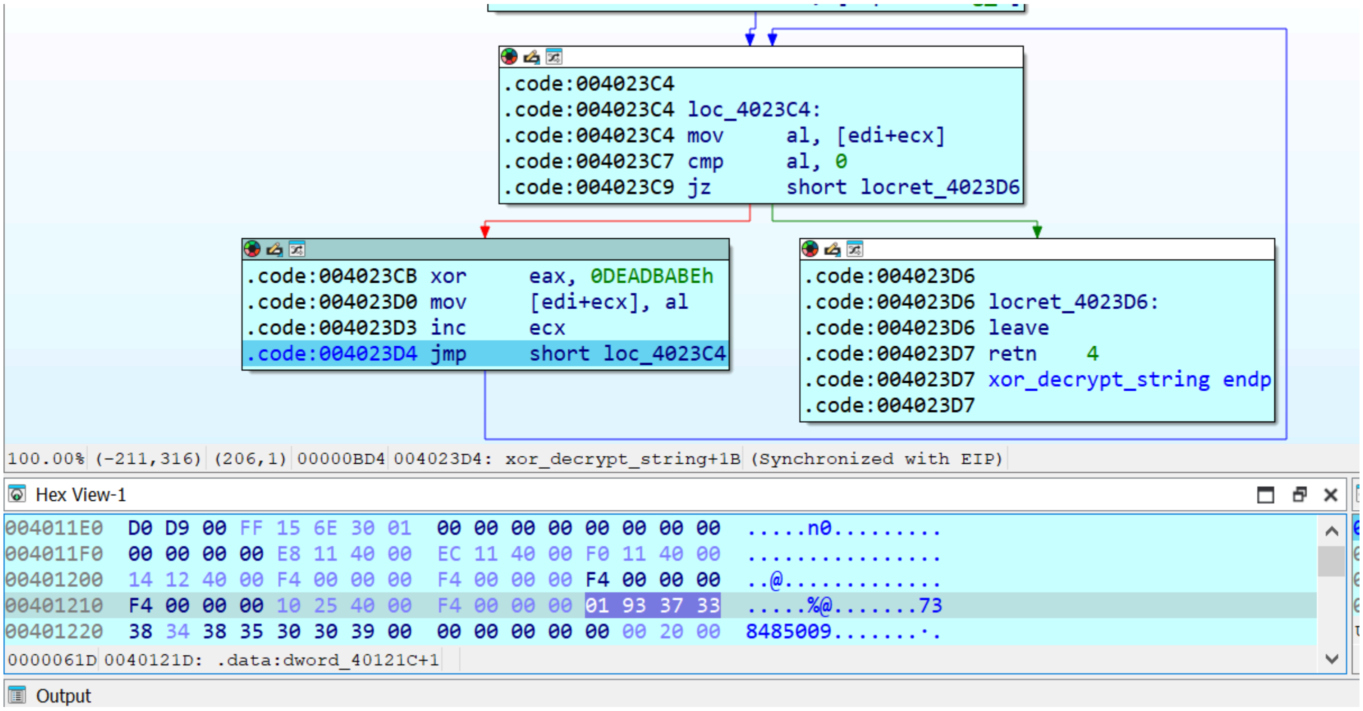
```
68.14% (3925,2126) (304,137) 00000AAC 004022AC: validate_username_password+283 (Synchronized with EIP)
```

Hex View-1

```
00401010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00401020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00401030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00401040  05 00 31 32 33 34 35 00  00 00 00 00 00 00 00 00  ..12345.........
00401050  00 00 00 00 10 36 37 38  39 3A 00 00 00 00 00 00  .....6789:......
```

And we saw that `edi` is pointing to memory at `0x0040106` which is the start of the string `6789:`

we try to put it in as input with no success so we continued investigating



```
.code:004023C4
.code:004023C4 loc_4023C4:
.code:004023C4 mov     al, [edi+ecx]
.code:004023C7 cmp     al, 0
.code:004023C9 jz      short locret_4023D6
```

```
.code:004023CB xor     eax, 0DEADBABEh
.code:004023D0 mov     [edi+ecx], al
.code:004023D3 inc     ecx
.code:004023D4 jmp     short loc_4023C4
```

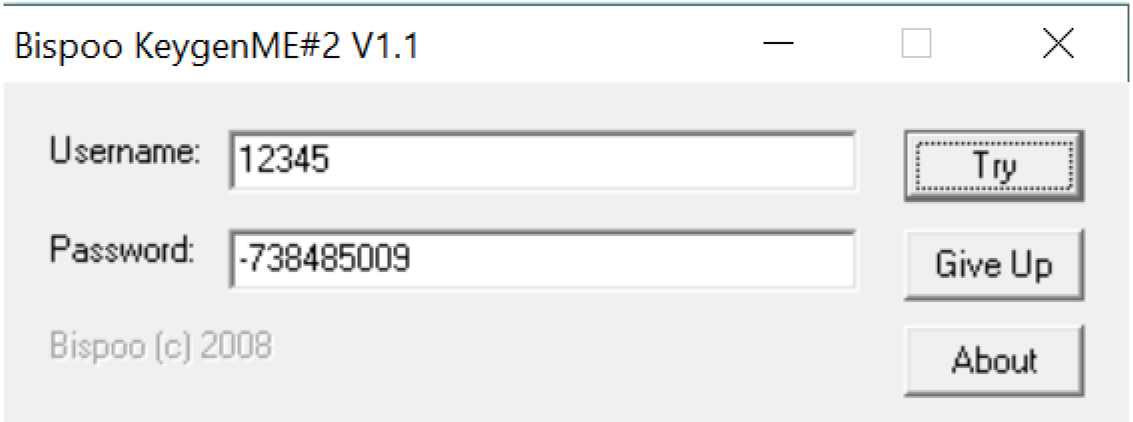
```
.code:004023D6
.code:004023D6 locret_4023D6:
.code:004023D6 leave
.code:004023D7 retn    4
.code:004023D7 xor_decrypt_string endp
.code:004023D7
```

```
100.00% (-211,316) (206,1) 00000BD4 004023D4: xor_decrypt_string+1B (Synchronized with EIP)
```

Hex View-1

```
004011E0  D0 D9 00 FF 15 6E 30 01  00 00 00 00 00 00 00 00  .....n0.........
004011F0  00 00 00 00 E8 11 40 00  EC 11 40 00 F0 11 40 00  ................
00401200  14 12 40 00 F4 00 00 00  F4 00 00 00 F4 00 00 00  ..@.............
00401210  F4 00 00 00 10 25 40 00  F4 00 00 00 01 93 37 33  .....%@.......73
00401220  38 34 38 35 30 30 39 00  00 00 00 00 00 00 20 00  8485009.......·.
0000061D 0040121D: .data:dword_40121C+1
```

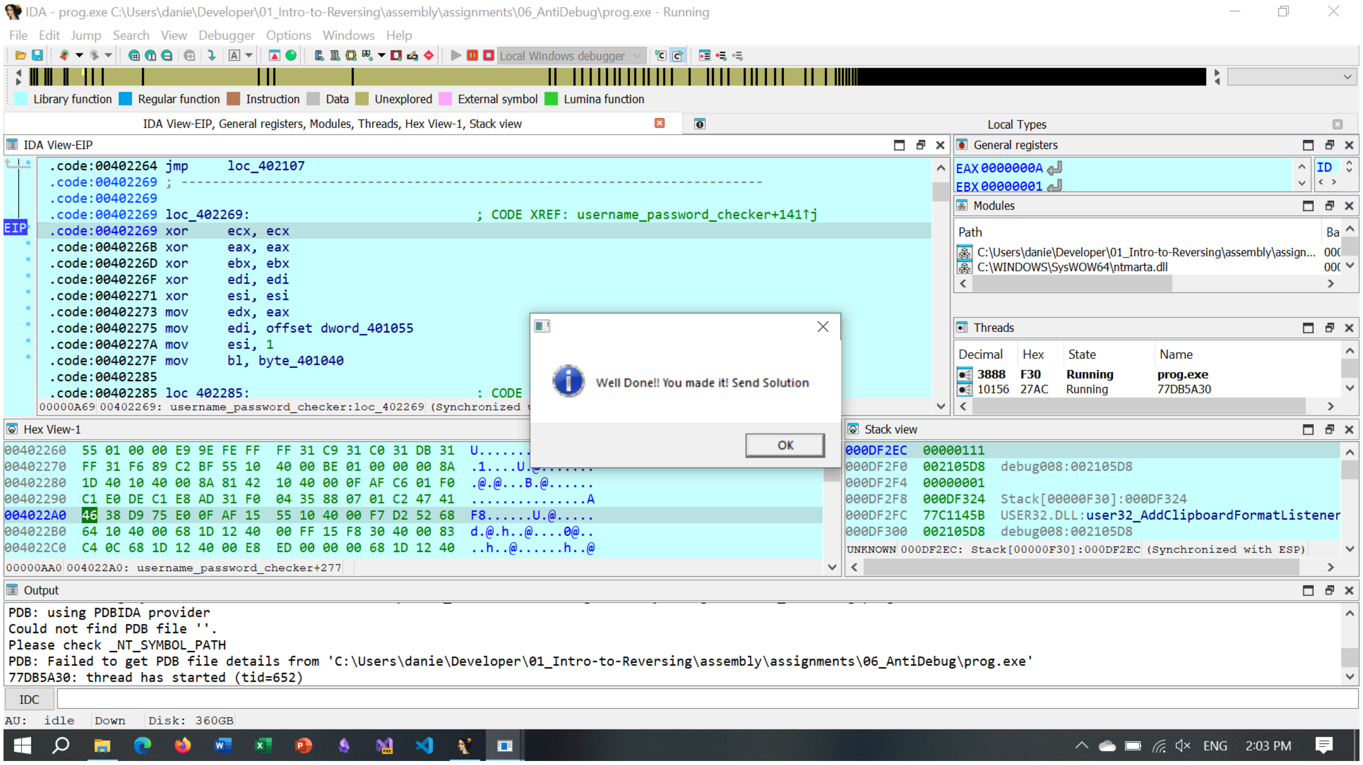
we saw that the program is calling the xor function and we saw that iteration by iteration the string that started at `ecx+edi` is fading away `"-738485009"` so we tried this:



Username: 12345

Password: -738485009

And got the success message


IDA - prog.exe C:\Users\danie\Developer\01_Intro-to-Reversing\assembly\assignments\06_AntiDebug\prog.exe - Running
File Edit Jump Search View Debugger Options Windows Help
Local Windows debugger
Library function
Regular function
Instruction
Data
Unexplored
External symbol
Lumina function
IDA View-EIP, General registers, Modules, Threads, Hex View-1, Stack view
Local Types
IDA View-EIP
General registers
Modules
Threads
Hex View-1
Stack view
Output
OK
IDC
AU: idle Down Disk: 360GB
ENG 2:03 PM

```
.code:00402264 jmp     loc_402107
.code:00402269 ; ---------------------------------------------------------------------------
.code:00402269
.code:00402269 loc_402269:                             ; CODE XREF: username_password_checker+141↑j
.code:00402269 xor     ecx, ecx
.code:0040226B xor     eax, eax
.code:0040226D xor     ebx, ebx
.code:0040226F xor     edi, edi
.code:00402271 xor     esi, esi
.code:00402273 mov     edx, eax
.code:00402275 mov     edi, offset dword_401055
.code:0040227A mov     esi, 1
.code:0040227F mov     bl, byte_401040
.code:00402285
.code:00402285 loc 402285:                             ; CODE
00000A69 00402269: username_password_checker:loc_402269 (Synchronized
```

Well Done!! You made it! Send Solution

```
EAX 0000000A
EBX 00000001
```

Path	Ba
C:\Users\danie\Developer\01_Intro-to-Reversing\assembly\assign...	00(
C:\WINDOWS\SysWOW64\ntmarta.dll	00(

Decimal	Hex	State	Name
3888	F30	Running	prog.exe
10156	27AC	Running	77DB5A30

```
00402260  55 01 00 00 E9 9E FE FF  FF 31 C9 31 C0 31 DB 31  U.......
00402270  FF 31 F6 89 C2 BF 55 10  40 00 BE 01 00 00 00 8A  .1....U.@.......
00402280  1D 40 10 40 00 8A 81 42  10 40 00 0F AF C6 01 F0  .@.@...B.@......
00402290  C1 E0 DE C1 E8 AD 31 F0  04 35 88 07 01 C2 47 41  ...............A
004022A0  46 38 D9 75 E0 0F AF 15  55 10 40 00 F7 D2 52 68  F8......U.@.....
004022B0  64 10 40 00 68 1D 12 40  00 FF 15 F8 30 40 00 83  d.@.h..@....0@..
004022C0  C4 0C 68 1D 12 40 00 E8  ED 00 00 00 68 1D 12 40  ..h..@......h..@
00000AA0 004022A0: username_password_checker+277
```

```
000DF2EC  00000111
000DF2F0  002105D8  debug008:002105D8
000DF2F4  00000001
000DF2F8  000DF324  Stack[00000F30]:000DF324
000DF2FC  77C1145B  USER32.DLL:user32_AddClipboardFormatListener
000DF300  002105D8  debug008:002105D8
UNKNOWN 000DF2EC: Stack[00000F30]:000DF2EC (Synchronized with ESP)
```

```
PDB: using PDBIDA provider
Could not find PDB file ''.
Please check _NT_SYMBOL_PATH
PDB: Failed to get PDB file details from 'C:\Users\danie\Developer\01_Intro-to-Reversing\assembly\assignments\06_AntiDebug\prog.exe'
77DB5A30: thread has started (tid=652)
```