Given a executable to reverse:

First we would try to debug and understand the program flow of the executable. To do so, I would like to use GDB.

gdb filename

set disassembly-flavor intel

break main

This will set the breakpoint in the main function of the program, as this crackme is an unstripped binary we know where the main starts had it been stripped, we had to use some tools like Ghidra, radare2 etc.

But it is a very simple program which is understandable and very human-like readability. I'm going to stick with gdb.

We see following program flow in main function,

0x00401460 <+0>: lea ecx,[esp+0x4]

0x00401464 <+4>: and esp,0xfffffff0

0x00401467 <+7>: push DWORD PTR [ecx-0x4]

0x0040146a <+10>: push ebp

0x0040146b <+11>: mov ebp,esp

0x0040146d <+13>: push ecx

0x0040146e <+14>: sub esp,0x24

0x00401471 <+17>: call 0x401a70 <\_\_main>

0x00401476 <+22>: mov DWORD PTR [esp+0xc],0x24

0x0040147e <+30>: mov DWORD PTR [esp+0x8],0x405064

0x00401486 <+38>: mov DWORD PTR [esp+0x4],0x405078

0x0040148e <+46>: mov DWORD PTR [esp],0x0

0x00401495 <+53>: call 0x403b98 <MessageBoxW@16>

0x0040149a <+58>: sub esp,0x10

0x0040149d <+61>: mov DWORD PTR [ebp-0xc],eax

0x004014a0 <+64>: cmp DWORD PTR [ebp-0xc],0x6

0x004014a4 <+68>: jne 0x4014cf <main()+111>

0x004014a6 <+70>: mov DWORD PTR [esp+0xc],0x40

0x004014ae <+78>: mov DWORD PTR [esp+0x8],0x4050a2

0x004014b6 <+86>: mov DWORD PTR [esp+0x4],0x4050ac

0x004014be <+94>: mov DWORD PTR [esp],0x0

0x004014c5 <+101>: call 0x403b98 <MessageBoxW@16>

0x004014ca <+106>: sub esp,0x10

0x004014cd <+109>: jmp 0x401525 <main()+197>

0x004014cf <+111>: cmp DWORD PTR [ebp-0xc],0x7

0x004014d3 <+115>: jne 0x4014fe <main()+158>

0x004014d5 <+117>: mov DWORD PTR [esp+0xc],0x10

0x004014dd <+125>: mov DWORD PTR [esp+0x8],0x4050e0

0x004014e5 <+133>: mov DWORD PTR [esp+0x4],0x405100

0x004014ed <+141>: mov DWORD PTR [esp],0x0

0x004014f4 <+148>: call 0x403b98 <MessageBoxW@16>

0x004014f9 <+153>: sub esp,0x10

0x004014fc <+156>: jmp 0x401525 <main()+197>

0x004014fe <+158>: mov DWORD PTR [esp+0xc],0x20

0x00401506 <+166>: mov DWORD PTR [esp+0x8],0x40513a

0x0040150e <+174>: mov DWORD PTR [esp+0x4],0x40514c

0x00401516 <+182>: mov DWORD PTR [esp],0x0

0x0040151d <+189>: call 0x403b98 <MessageBoxW@16>

0x00401522 <+194>: sub esp,0x10

0x00401525 <+197>: mov eax,0x0

0x0040152a <+202>: mov ecx,DWORD PTR [ebp-0x4]

0x0040152d <+205>: leave

0x0040152e <+206>: lea esp,[ecx-0x4]

0x00401531 <+209>: ret

We are seeing the main function is calling the message function <MessageBoxW@16> 4 times, but when we ran the program, the message box popped up only thrice?

Are we getting a hint for hidden\_option?

Lets analyze,

**METHOD 1:Skip and set**

The actual message box starts at \*1495(0x00401495)(checked)



We then chose Yes to message box skipping instructions we get at \*14c5 (checked) by ni we get to end of program 1525



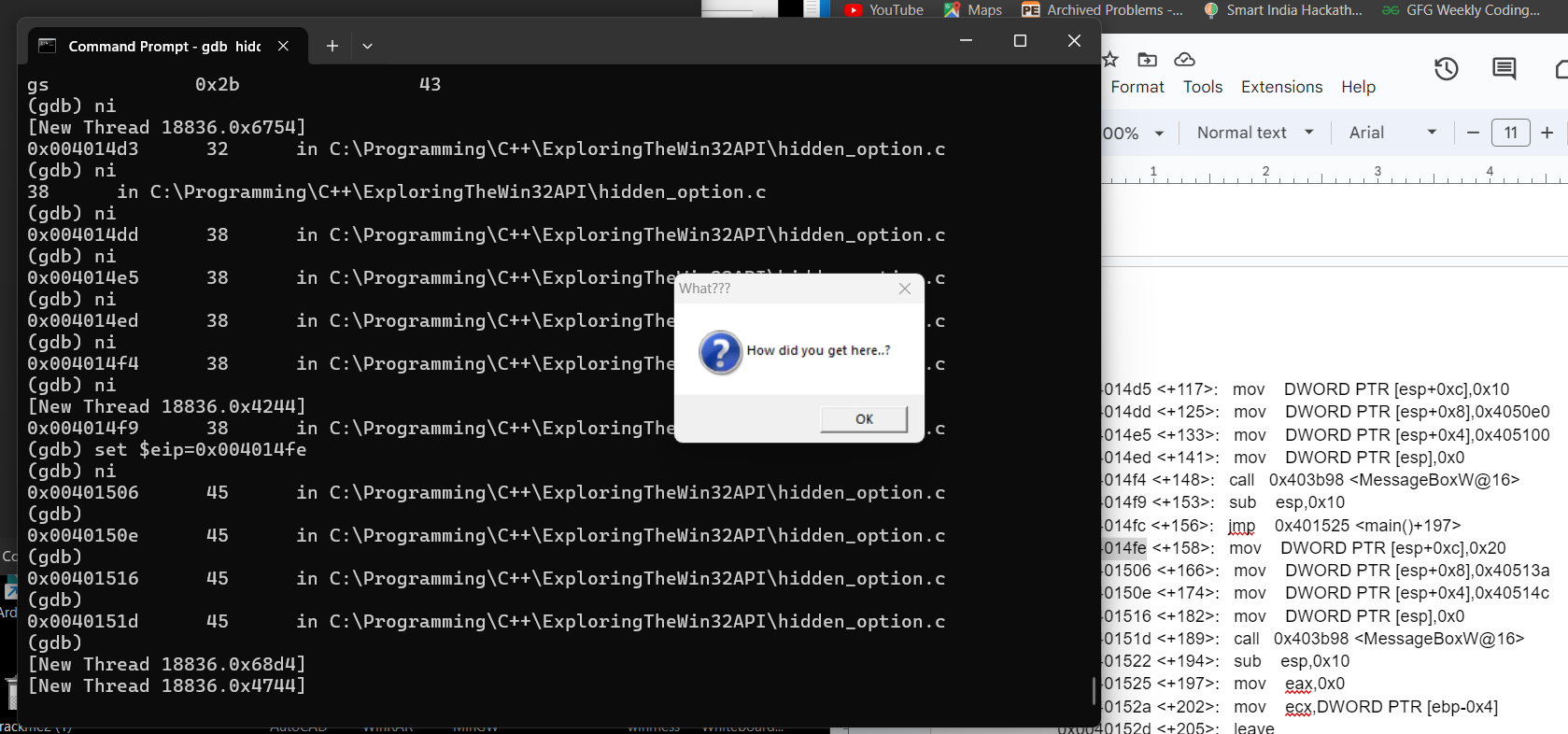
Then we choose No, address is pointing to \*14f4(checked) and then end.



Finally, we are remaining we last MessageBox \*151d

Now lets escalate there by ni to do this lets set address to

set $eip= 0x004014fe



By doing this we are skipping the jump and reaching at required address(0x0040151d) by ni’s:

Finally we the get hidden\_option!

**Method 2:Manipulating the Flag**

0x00401460 <+0>: lea ecx,[esp+0x4]

0x00401464 <+4>: and esp,0xfffffff0

0x00401467 <+7>: push DWORD PTR [ecx-0x4]

0x0040146a <+10>: push ebp

0x0040146b <+11>: mov ebp,esp

0x0040146d <+13>: push ecx

0x0040146e <+14>: sub esp,0x24

0x00401471 <+17>: call 0x401a70 <\_\_main>

0x00401476 <+22>: mov DWORD PTR [esp+0xc],0x24

0x0040147e <+30>: mov DWORD PTR [esp+0x8],0x405064

0x00401486 <+38>: mov DWORD PTR [esp+0x4],0x405078

0x0040148e <+46>: mov DWORD PTR [esp],0x0

0x00401495 <+53>: call 0x403b98 <MessageBoxW@16>

0x0040149a <+58>: sub esp,0x10

0x0040149d <+61>: mov DWORD PTR [ebp-0xc],eax

0x004014a0 <+64>: cmp DWORD PTR [ebp-0xc],0x6

**0x004014a4 <+68>: jne 0x4014cf <main()+111>**

0x004014a6 <+70>: mov DWORD PTR [esp+0xc],0x40

0x004014ae <+78>: mov DWORD PTR [esp+0x8],0x4050a2

0x004014b6 <+86>: mov DWORD PTR [esp+0x4],0x4050ac

0x004014be <+94>: mov DWORD PTR [esp],0x0

0x004014c5 <+101>: call 0x403b98 <MessageBoxW@16>

0x004014ca <+106>: sub esp,0x10

0x004014cd <+109>: jmp 0x401525 <main()+197>

0x004014cf <+111>: cmp DWORD PTR [ebp-0xc],0x7

**0x004014d3 <+115>: jne 0x4014fe <main()+158>**

0x004014d5 <+117>: mov DWORD PTR [esp+0xc],0x10

0x004014dd <+125>: mov DWORD PTR [esp+0x8],0x4050e0

0x004014e5 <+133>: mov DWORD PTR [esp+0x4],0x405100

0x004014ed <+141>: mov DWORD PTR [esp],0x0

0x004014f4 <+148>: call 0x403b98 <MessageBoxW@16>

0x004014f9 <+153>: sub esp,0x10

0x004014fc <+156>: jmp 0x401525 <main()+197>

*0x004014fe <+158>: mov DWORD PTR [esp+0xc],0x20*

0x00401506 <+166>: mov DWORD PTR [esp+0x8],0x40513a

0x0040150e <+174>: mov DWORD PTR [esp+0x4],0x40514c

0x00401516 <+182>: mov DWORD PTR [esp],0x0

0x0040151d <+189>: call 0x403b98 <MessageBoxW@16>

0x00401522 <+194>: sub esp,0x10

0x00401525 <+197>: mov eax,0x0

0x0040152a <+202>: mov ecx,DWORD PTR [ebp-0x4]

0x0040152d <+205>: leave

0x0040152e <+206>: lea esp,[ecx-0x4]

0x00401531 <+209>: ret

Now, carefully again [highlighted parts], if we somehow manipulate those jne , then we are very able to jump and jump to required address that’s 0x0040151d <+189>:<MessageBoxW@16>

Lets try again,

**0x004014a4 <+68>: jne 0x4014cf <main()+111>**

JNE means JUMP NOT EQUAL[to zero], it essentially checks the ZF[zero flag] it checks whether zero flag is set to zero or not if it's zero it would jump meaning two operands are not same.If they were the same it would be equal to One.

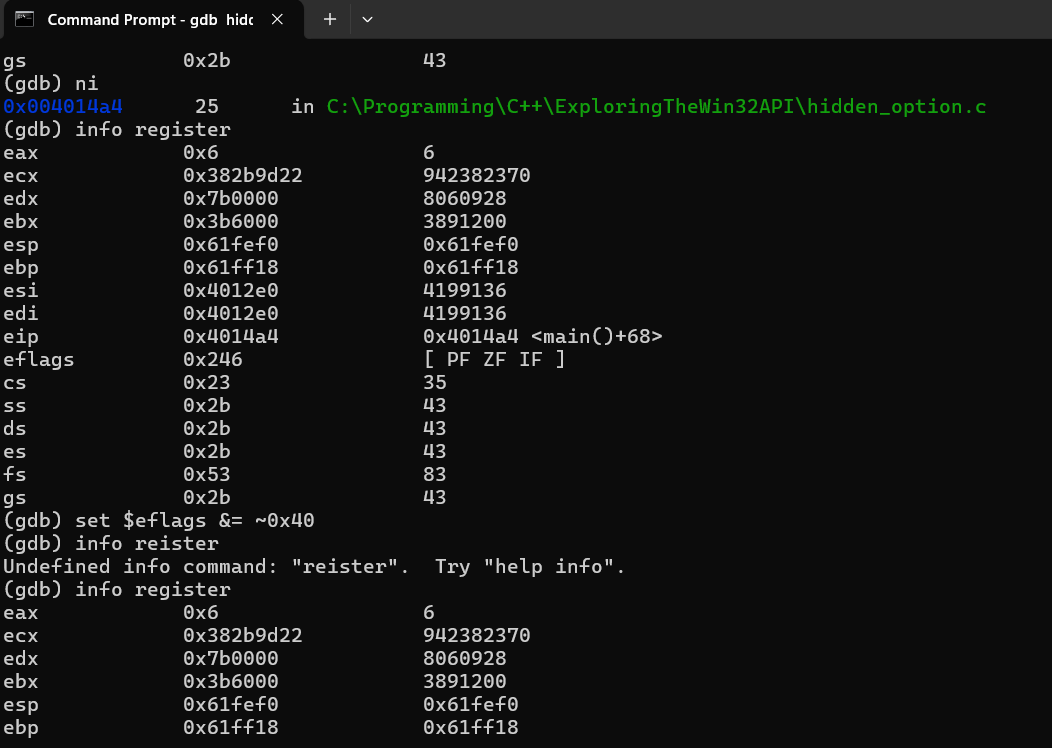
We are seeing

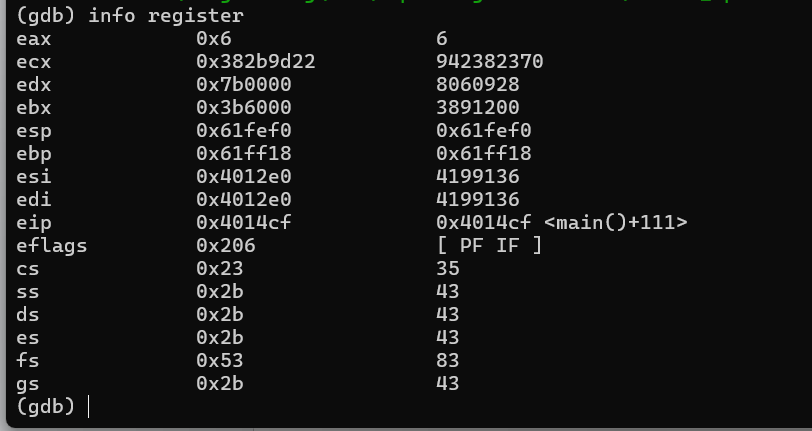
0x004014a0 <+64>: cmp DWORD PTR [ebp-0xc],0x6

Eax is 6 and the comparison value is 0x6 too! It would mean that cmp will store 1 in its ZF and would not jump like in the previous case but now we would manipulate it :)

For that lets click yes to that message box and ni until jne, then do

set $eflags &= ~0x40





Explanation:

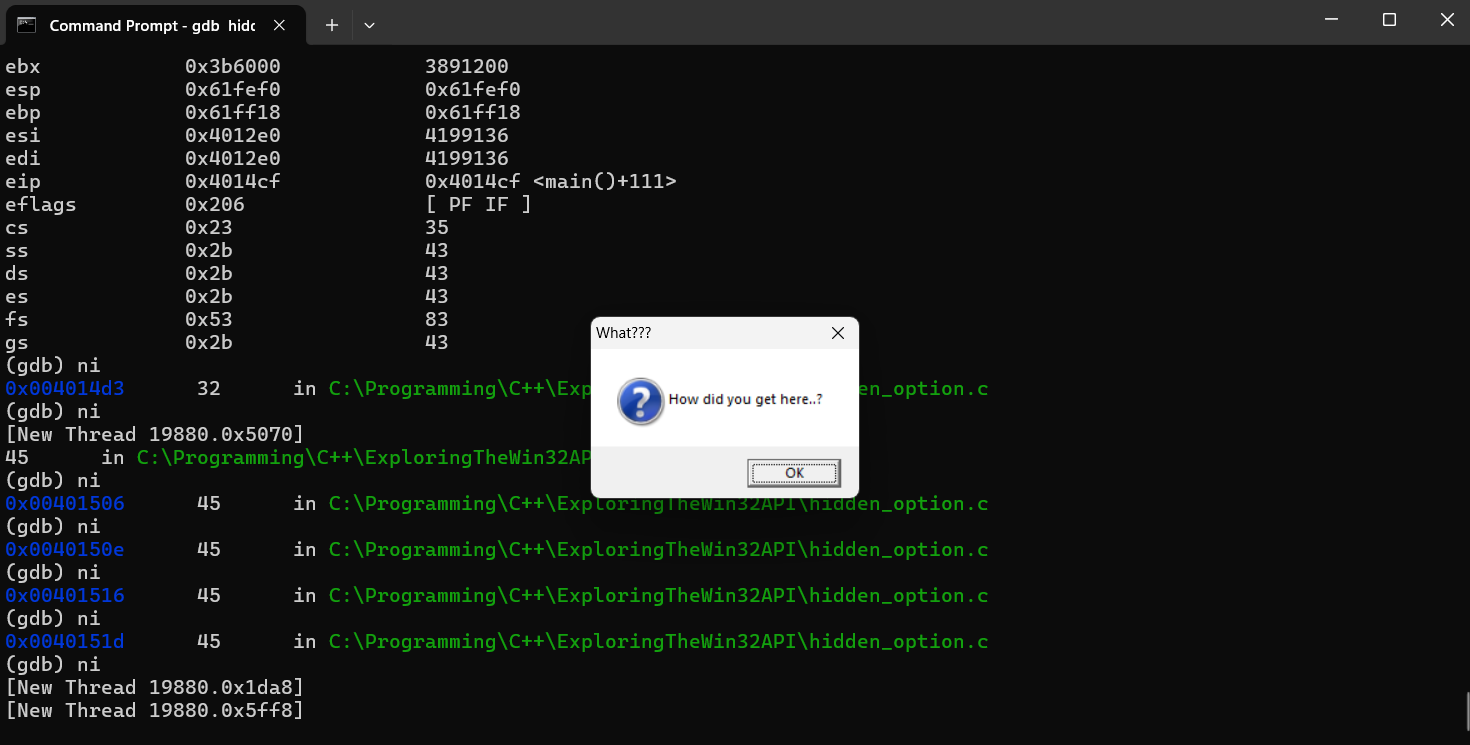
Originally, by info register the flag must be stored as 0X246

But we want it to be 0x206 [clear or zero]

For that we need to use bitwise AND operator(to only change zf and leaving other flags untouched)

This would change the 6th bit(in EFlags, the ZF bit is stored in 6th from MSB) leaving others as they were originally.

Let's do this and



voila we got it!!!

Next time i will try to bring something more challenging cause it helps me to push through Reverse Engineering T~T

I hope you all learned something :3