Vector Veil Challenge Walkthrough

The difficulty of this challenge mainly arises from understanding and debugging code obfuscated with AVX2. For example, recognizing the `atoi` function written in AVX2 can be difficult. Additionally, values are not stored in general-purpose registers like `rax`, but instead in SSE and AVX registers such as `xmm0` and `ymm0`. Mathematical calculations are also done with AVX2 instructions instead of conventional x64 instructions.

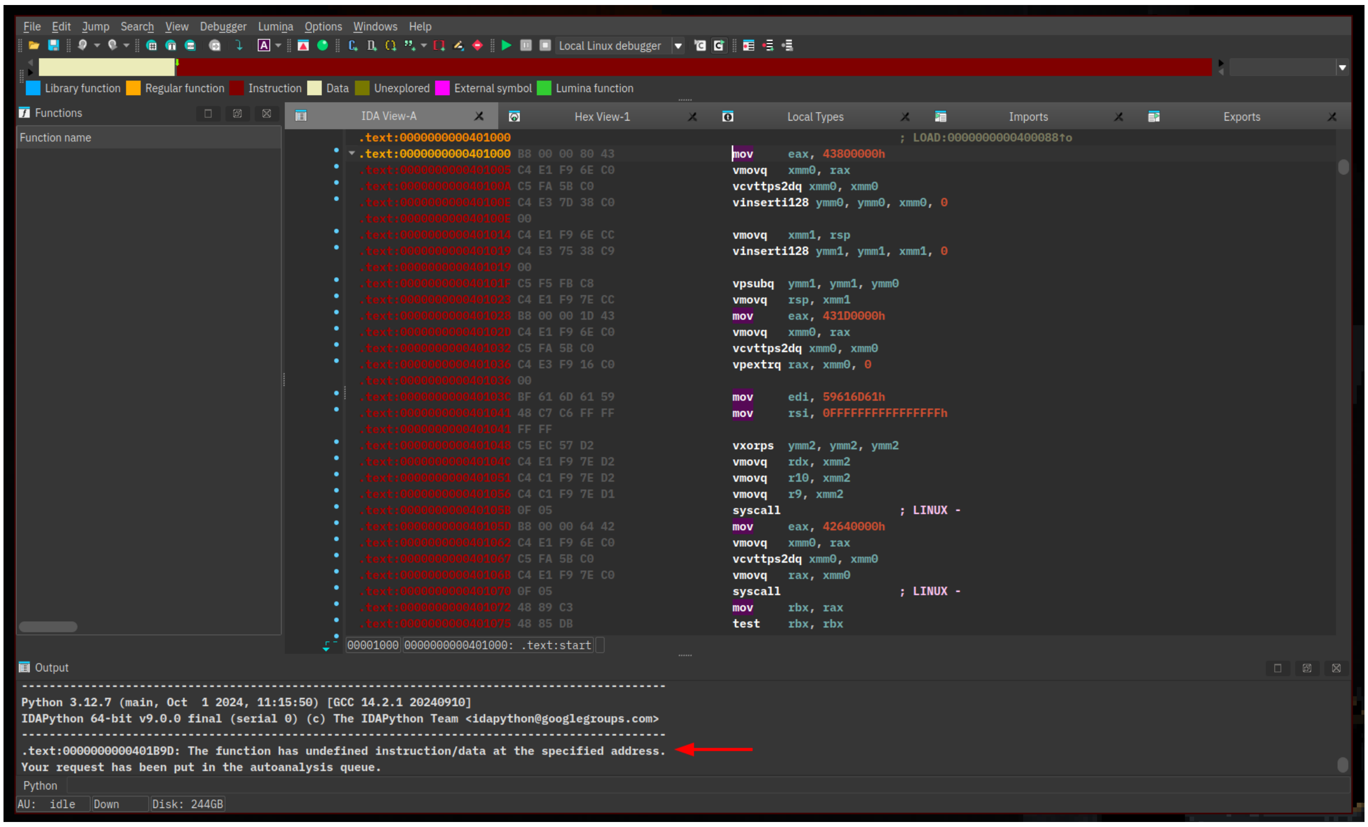
The following are some critical points for solving this challenge:

- (Optional) Fix the null byte at the end of the file so that the program can be recognized as a function
- (Optional) Defeat the anti-debug mechanism to enable debugging and stepping through the code
- Find the FNV-1a loop responsible for calculating the registration code
- Understand how the registration code input is parsed (`atoi`)

Detailed Walkthrough

This walkthrough will use IDA as the analysis environment as an example. This challenge can be solved following the same principle with any other environments.

Upon opening the binary in IDA, it can be seen that IDA is unable to recognize the code as as functions. This means we cannot get a control flow graph, which is inconvenient. When we try to manually define a function and the program's entry point using `p`, IDA produces an error:

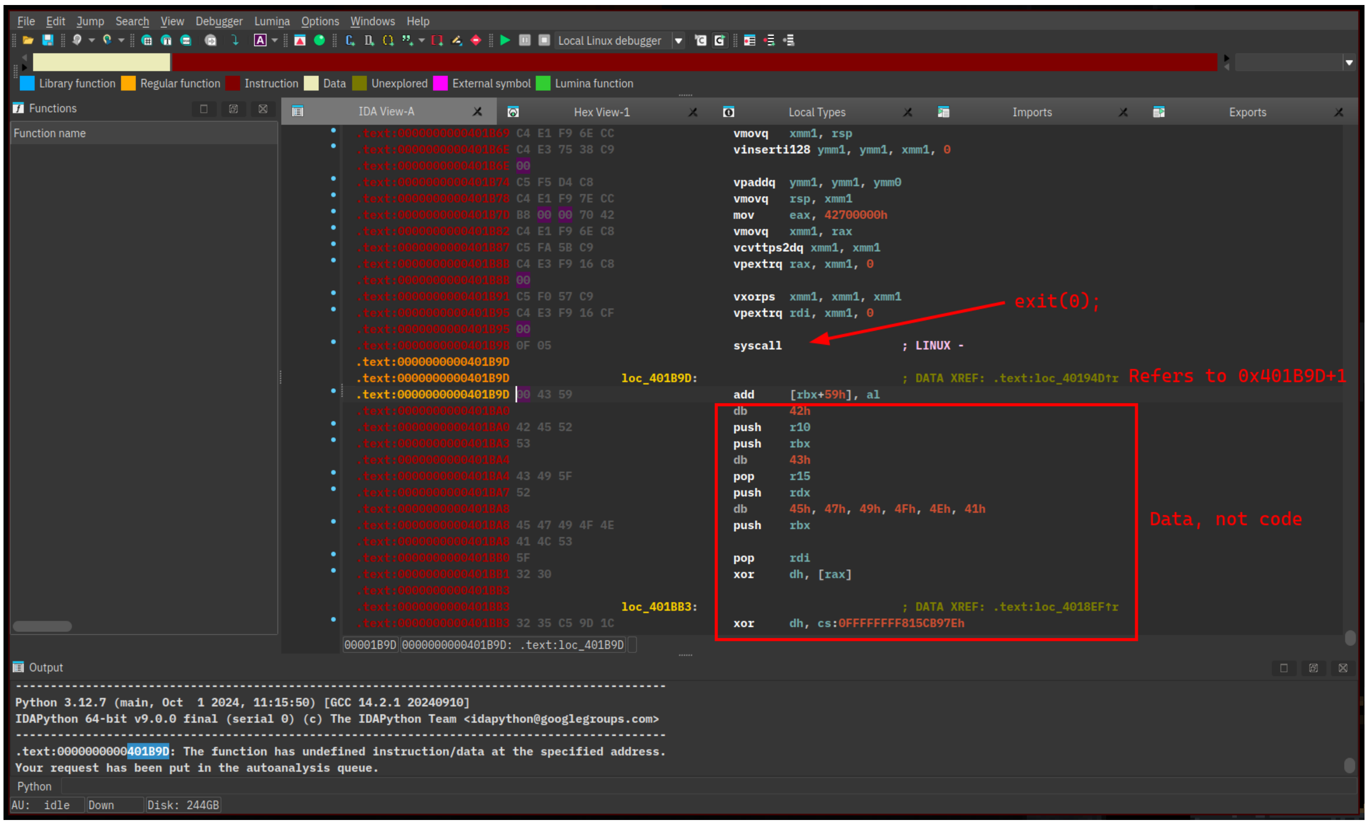


```
.text:0000000000401000                                      ; LOAD:0000000000400088↑o
.text:0000000000401000 B8 00 00 80 43       mov     eax, 43800000h
.text:0000000000401005 C4 E1 F9 6E C0       vmovq   xmm0, rax
.text:000000000040100A C5 FA 5B C0          vcvttps2dq xmm0, xmm0
.text:000000000040100E C4 E3 7D 38 C0       vinserti128 ymm0, ymm0, xmm0, 0
.text:000000000040100E 00
.text:0000000000401014 C4 E1 F9 6E CC       vmovq   xmm1, rsp
.text:0000000000401019 C4 E3 75 38 C9       vinserti128 ymm1, ymm1, xmm1, 0
.text:0000000000401019 00
.text:000000000040101F C5 F5 FB C8          vpsubq  ymm1, ymm1, ymm0
.text:0000000000401023 C4 E1 F9 7E CC       vmovq   rsp, xmm1
.text:0000000000401028 B8 00 00 1D 43       mov     eax, 431D0000h
.text:000000000040102D C4 E1 F9 6E C0       vmovq   xmm0, rax
.text:0000000000401032 C5 FA 5B C0          vcvttps2dq xmm0, xmm0
.text:0000000000401036 C4 E3 F9 16 C0       vpextrq rax, xmm0, 0
.text:0000000000401036 00
.text:000000000040103C BF 61 6D 61 59       mov     edi, 59616D61h
.text:0000000000401041 48 C7 C6 FF FF       mov     rsi, 0FFFFFFFFFFFFFFFFh
.text:0000000000401041 FF FF
.text:0000000000401048 C5 EC 57 D2          vxorps  ymm2, ymm2, ymm2
.text:000000000040104C C4 E1 F9 7E D2       vmovq   rdx, xmm2
.text:0000000000401051 C4 C1 F9 7E D2       vmovq   r10, xmm2
.text:0000000000401056 C4 C1 F9 7E D1       vmovq   r9, xmm2
.text:000000000040105B 0F 05                syscall                 ; LINUX -
.text:000000000040105D B8 00 00 64 42       mov     eax, 42640000h
.text:0000000000401062 C4 E1 F9 6E C0       vmovq   xmm0, rax
.text:0000000000401067 C5 FA 5B C0          vcvttps2dq xmm0, xmm0
.text:000000000040106B C4 E1 F9 7E C0       vmovq   rax, xmm0
.text:0000000000401070 0F 05                syscall                 ; LINUX -
.text:0000000000401072 48 89 C3             mov     rbx, rax
.text:0000000000401075 48 85 DB             test    rbx, rbx
00001000 0000000000401000: .text:start
```

```
---------------------------------------------------------------------------------------
Python 3.12.7 (main, Oct  1 2024, 11:15:50) [GCC 14.2.1 20240910]
IDAPython 64-bit v9.0.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
---------------------------------------------------------------------------------------
.text:0000000000401B9D: The function has undefined instruction/data at the specified address.
Your request has been put in the autoanalysis queue.
```

```
AU:  idle   Down   Disk: 244GB
```

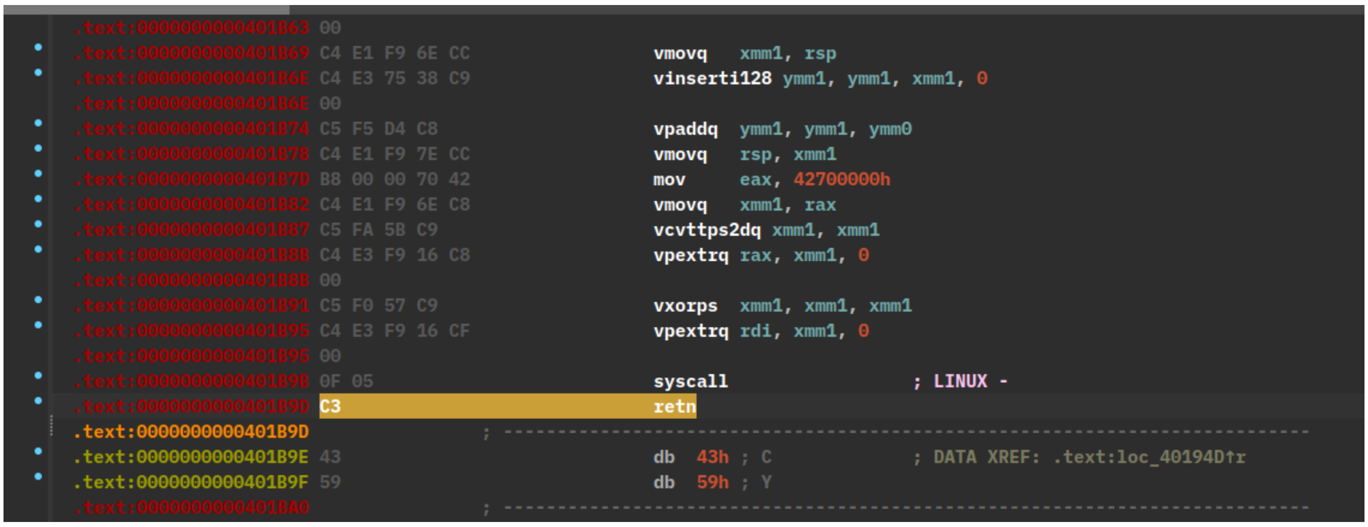
The error leads us to the end of the program. The code immedialy above `loc_401B9D` is equivalent to `exit(0);`. The bytes after that syscall is never reached and interpreted as code. The code above only refers to `0x401B9D+1`, which means the null byte at `0x401B9D` is never used.



```
.text:0000000000401B69 C4 E1 F9 6E CC          vmovq   xmm1, rsp
.text:0000000000401B6E C4 E3 75 38 C9          vinserti128 ymm1, ymm1, xmm1, 0
.text:0000000000401B6E 00
.text:0000000000401B74 C5 F5 D4 C8             vpaddq  ymm1, ymm1, ymm0
.text:0000000000401B78 C4 E1 F9 7E CC          vmovq   rsp, xmm1
.text:0000000000401B7D B8 00 00 70 42          mov     eax, 42700000h
.text:0000000000401B82 C4 E1 F9 6E C8          vmovq   xmm1, rax
.text:0000000000401B87 C5 FA 5B C9             vcvttps2dq xmm1, xmm1
.text:0000000000401B8B C4 E3 F9 16 C8          vpextrq rax, xmm1, 0
.text:0000000000401B8B 00
.text:0000000000401B91 C5 F0 57 C9             vxorps  xmm1, xmm1, xmm1
.text:0000000000401B95 C4 E3 F9 16 CF          vpextrq rdi, xmm1, 0
.text:0000000000401B95 00
.text:0000000000401B9B 0F 05                   syscall                 ; LINUX -
.text:0000000000401B9D
.text:0000000000401B9D            loc_401B9D:                          ; DATA XREF: .text:loc_40194D↑r
.text:0000000000401B9D 00 43 59                add     [rbx+59h], al
.text:0000000000401BA0                         db      42h
.text:0000000000401BA0 42 45 52                push    r10
.text:0000000000401BA3 53                      push    rbx
.text:0000000000401BA4                         db      43h
.text:0000000000401BA4 43 49 5F                pop     r15
.text:0000000000401BA7 52                      push    rdx
.text:0000000000401BA8                         db      45h, 47h, 49h, 4Fh, 4Eh, 41h
.text:0000000000401BA8 45 47 49 4F 4E          push    rbx
.text:0000000000401BA8 41 4C 53
.text:0000000000401BB0 5F                      pop     rdi
.text:0000000000401BB1 32 30                   xor     dh, [rax]
.text:0000000000401BB3
.text:0000000000401BB3            loc_401BB3:                          ; DATA XREF: .text:loc_4018EF↑r
.text:0000000000401BB3 32 35 C5 9D 1C          xor     dh, cs:0FFFFFFFF815CB97Eh
00001B9D 0000000000401B9D: .text:loc_401B9D
```

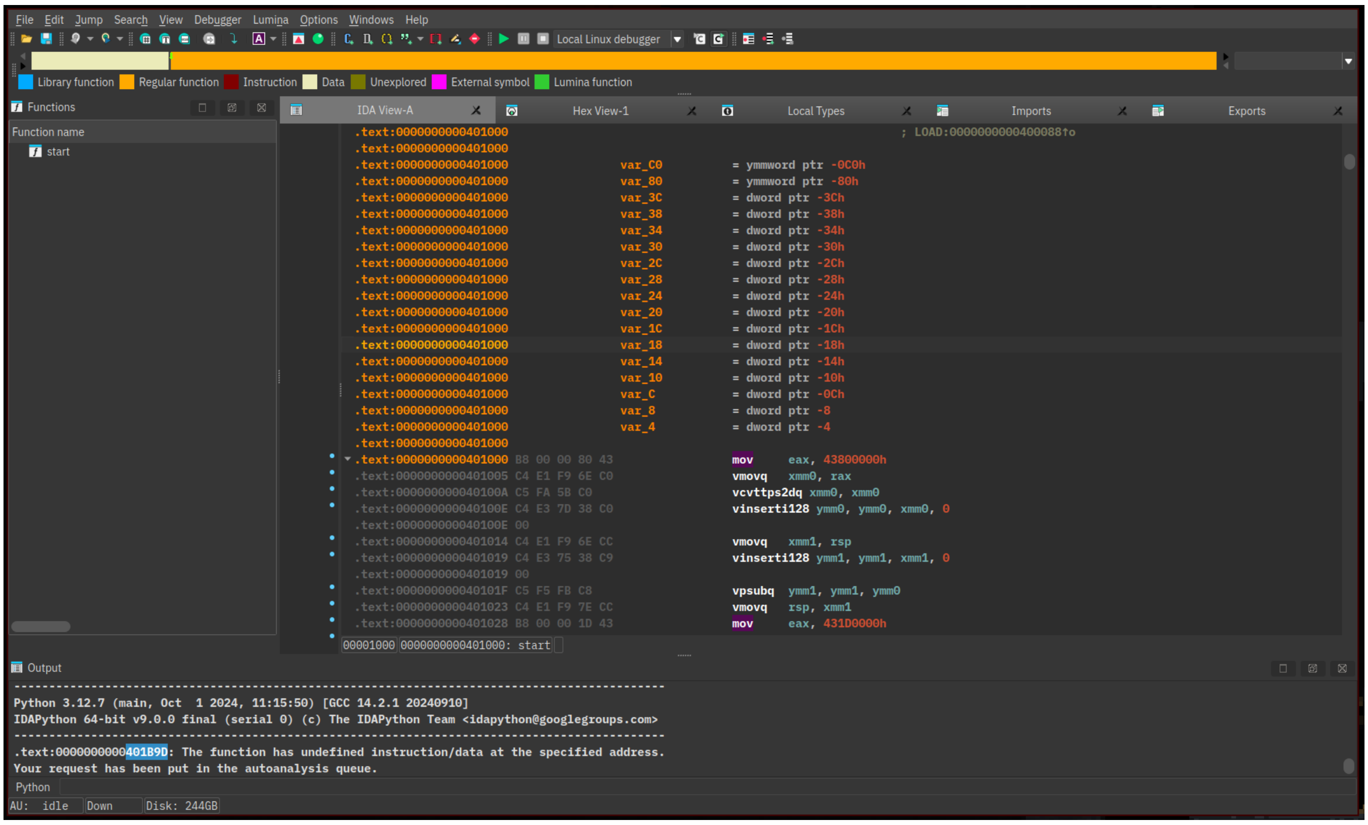
```
---------------------------------------------------------------------------------------
Python 3.12.7 (main, Oct  1 2024, 11:15:50) [GCC 14.2.1 20240910]
IDAPython 64-bit v9.0.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
---------------------------------------------------------------------------------------
.text:0000000000401B9D: The function has undefined instruction/data at the specified address.
Your request has been put in the autoanalysis queue.
```

We can leverage this and change the byte to a `ret` instruction so IDA can recognize the code above as a function. This is not required, but will help make the following analysis easier.



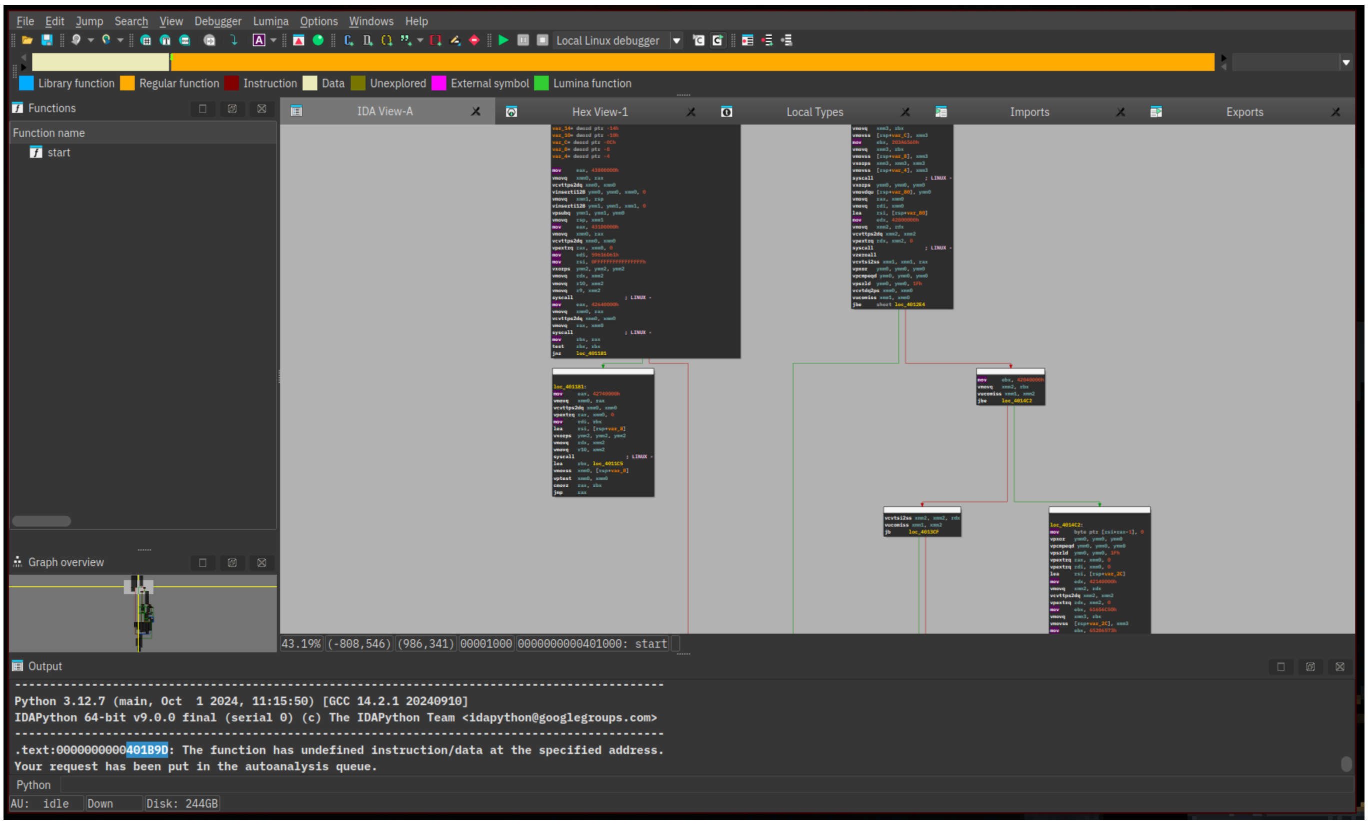
```
.text:0000000000401B63 00
.text:0000000000401B69 C4 E1 F9 6E CC          vmovq   xmm1, rsp
.text:0000000000401B6E C4 E3 75 38 C9          vinserti128 ymm1, ymm1, xmm1, 0
.text:0000000000401B6E 00
.text:0000000000401B74 C5 F5 D4 C8             vpaddq  ymm1, ymm1, ymm0
.text:0000000000401B78 C4 E1 F9 7E CC          vmovq   rsp, xmm1
.text:0000000000401B7D B8 00 00 70 42          mov     eax, 42700000h
.text:0000000000401B82 C4 E1 F9 6E C8          vmovq   xmm1, rax
.text:0000000000401B87 C5 FA 5B C9             vcvttps2dq xmm1, xmm1
.text:0000000000401B8B C4 E3 F9 16 C8          vpextrq rax, xmm1, 0
.text:0000000000401B8B 00
.text:0000000000401B91 C5 F0 57 C9             vxorps  xmm1, xmm1, xmm1
.text:0000000000401B95 C4 E3 F9 16 CF          vpextrq rdi, xmm1, 0
.text:0000000000401B95 00
.text:0000000000401B9B 0F 05                   syscall                 ; LINUX -
.text:0000000000401B9D C3                      retn
.text:0000000000401B9D               ; ---------------------------------------------------------
.text:0000000000401B9E 43                      db  43h ; C             ; DATA XREF: .text:loc_40194D↑r
.text:0000000000401B9F 59                      db  59h ; Y
.text:0000000000401BA0               ; ---------------------------------------------------------
```

After changing the null byte to a `C3`, we can now manually define a function at the program's entrypoint.


```
.text:0000000000401000                                         ; LOAD:0000000000400088↑o
.text:0000000000401000
.text:0000000000401000                 var_C0          = ymmword ptr -0C0h
.text:0000000000401000                 var_80          = ymmword ptr -80h
.text:0000000000401000                 var_3C          = dword ptr -3Ch
.text:0000000000401000                 var_38          = dword ptr -38h
.text:0000000000401000                 var_34          = dword ptr -34h
.text:0000000000401000                 var_30          = dword ptr -30h
.text:0000000000401000                 var_2C          = dword ptr -2Ch
.text:0000000000401000                 var_28          = dword ptr -28h
.text:0000000000401000                 var_24          = dword ptr -24h
.text:0000000000401000                 var_20          = dword ptr -20h
.text:0000000000401000                 var_1C          = dword ptr -1Ch
.text:0000000000401000                 var_18          = dword ptr -18h
.text:0000000000401000                 var_14          = dword ptr -14h
.text:0000000000401000                 var_10          = dword ptr -10h
.text:0000000000401000                 var_C           = dword ptr -0Ch
.text:0000000000401000                 var_8           = dword ptr -8
.text:0000000000401000                 var_4           = dword ptr -4
.text:0000000000401000
.text:0000000000401000 B8 00 00 80 43                  mov     eax, 43800000h
.text:0000000000401005 C4 E1 F9 6E C0                  vmovq   xmm0, rax
.text:000000000040100A C5 FA 5B C0                     vcvttps2dq xmm0, xmm0
.text:000000000040100E C4 E3 7D 38 C0                  vinserti128 ymm0, ymm0, xmm0, 0
.text:000000000040100E 00
.text:0000000000401014 C4 E1 F9 6E CC                  vmovq   xmm1, rsp
.text:0000000000401019 C4 E3 75 38 C9                  vinserti128 ymm1, ymm1, xmm1, 0
.text:0000000000401019 00
.text:000000000040101F C5 F5 FB C8                     vpsubq  ymm1, ymm1, ymm0
.text:0000000000401023 C4 E1 F9 7E CC                  vmovq   rsp, xmm1
.text:0000000000401028 B8 00 00 1D 43                  mov     eax, 431D0000h
00001000 0000000000401000: start
```

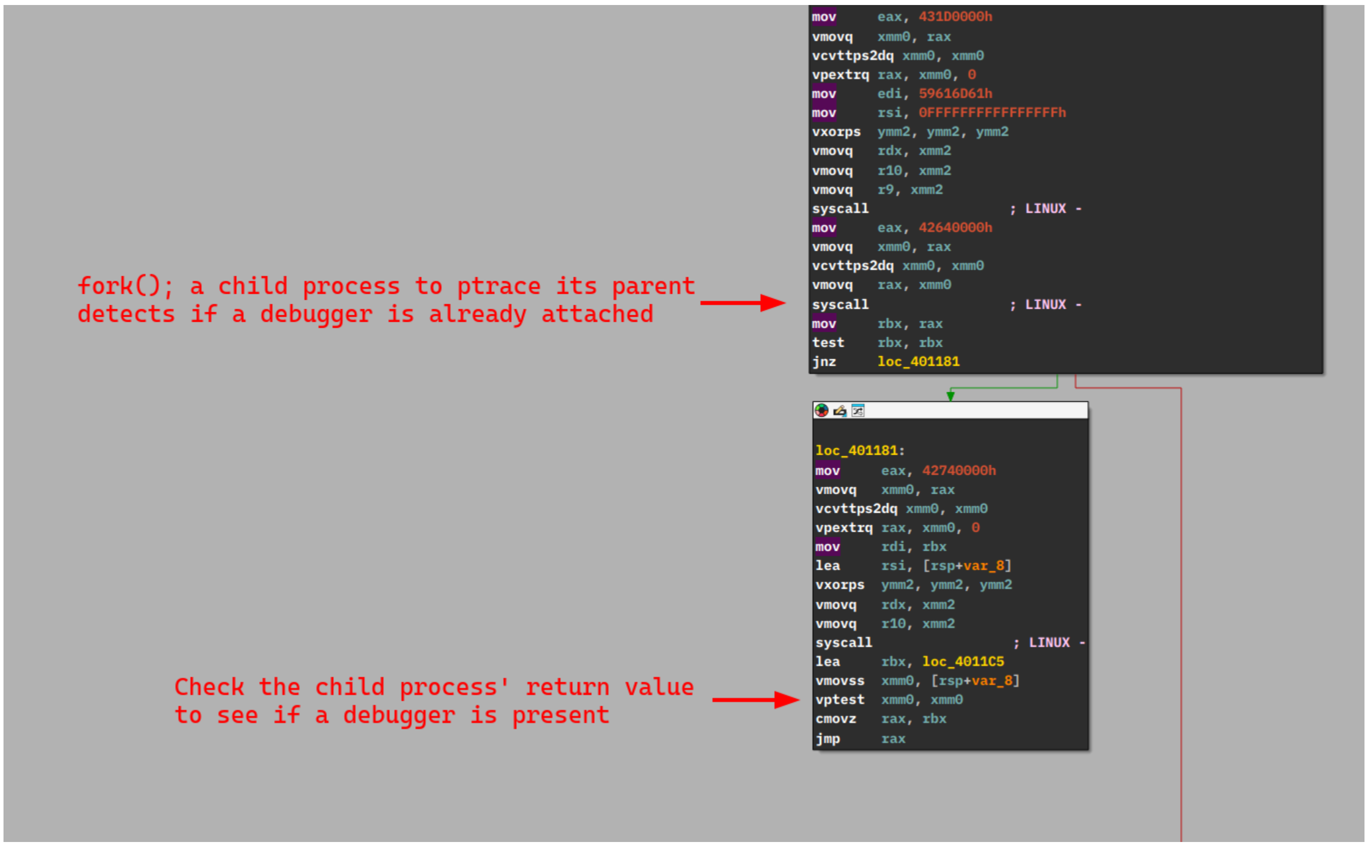
```
---------------------------------------------------------------------------------------
Python 3.12.7 (main, Oct  1 2024, 11:15:50) [GCC 14.2.1 20240910]
IDAPython 64-bit v9.0.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
---------------------------------------------------------------------------------------
.text:000000000040189D: The function has undefined instruction/data at the specified address.
Your request has been put in the autoanalysis queue.
```

Now that IDA recognizes the code as a function, we can tell it to make a control graph.


```
43.19% (-808,546) (986,341) 00001000 0000000000401000: start
```

```
---------------------------------------------------------------------------------------
Python 3.12.7 (main, Oct  1 2024, 11:15:50) [GCC 14.2.1 20240910]
IDAPython 64-bit v9.0.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
---------------------------------------------------------------------------------------
.text:000000000040189D: The function has undefined instruction/data at the specified address.
Your request has been put in the autoanalysis queue.
```

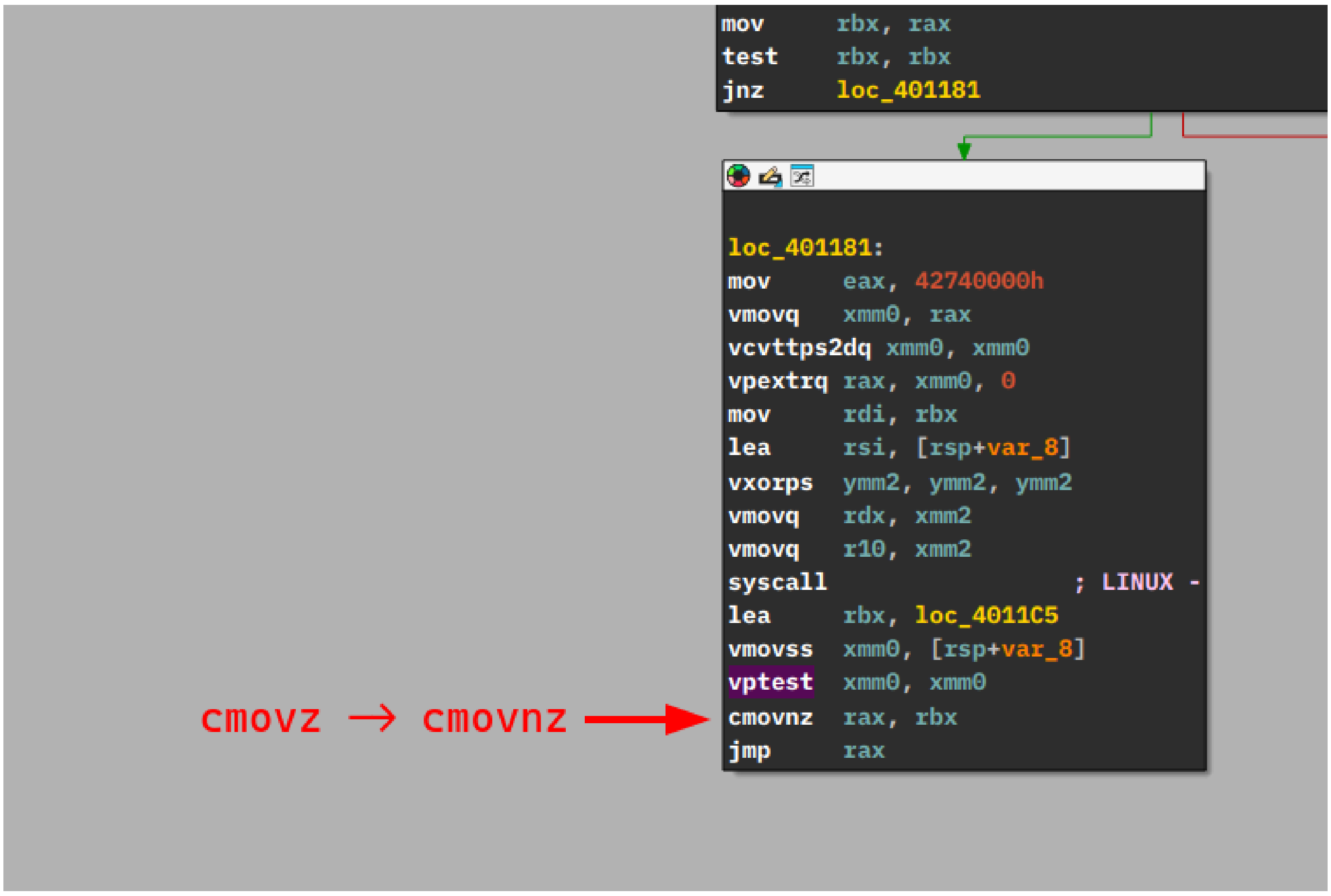
If we debug the code, it will produce a segfault. This is because this program contains an anti-debug mechanism. It forks a child process which attaches its parent with `ptrace`. If the attach fails, it means a debugger is present and will not jump to the correct code afterwards.


fork(); a child process to ptrace its parent
detects if a debugger is already attached
Check the child process' return value
to see if a debugger is present

```
mov     eax, 431D0000h
vmovq   xmm0, rax
vcvttps2dq xmm0, xmm0
vpextrq rax, xmm0, 0
mov     edi, 59616D61h
mov     rsi, 0FFFFFFFFFFFFFFFFh
vxorps  ymm2, ymm2, ymm2
vmovq   rdx, xmm2
vmovq   r10, xmm2
vmovq   r9, xmm2
syscall                 ; LINUX -
mov     eax, 42640000h
vmovq   xmm0, rax
vcvttps2dq xmm0, xmm0
vmovq   rax, xmm0
syscall                 ; LINUX -
mov     rbx, rax
test    rbx, rbx
jnz     loc_401181

loc_401181:
mov     eax, 42740000h
vmovq   xmm0, rax
vcvttps2dq xmm0, xmm0
vpextrq rax, xmm0, 0
mov     rdi, rbx
lea     rsi, [rsp+var_8]
vxorps  ymm2, ymm2, ymm2
vmovq   rdx, xmm2
vmovq   r10, xmm2
syscall                 ; LINUX -
lea     rbx, loc_4011C5
vmovss  xmm0, [rsp+var_8]
vptest  xmm0, xmm0
cmovz   rax, rbx
jmp     rax
```

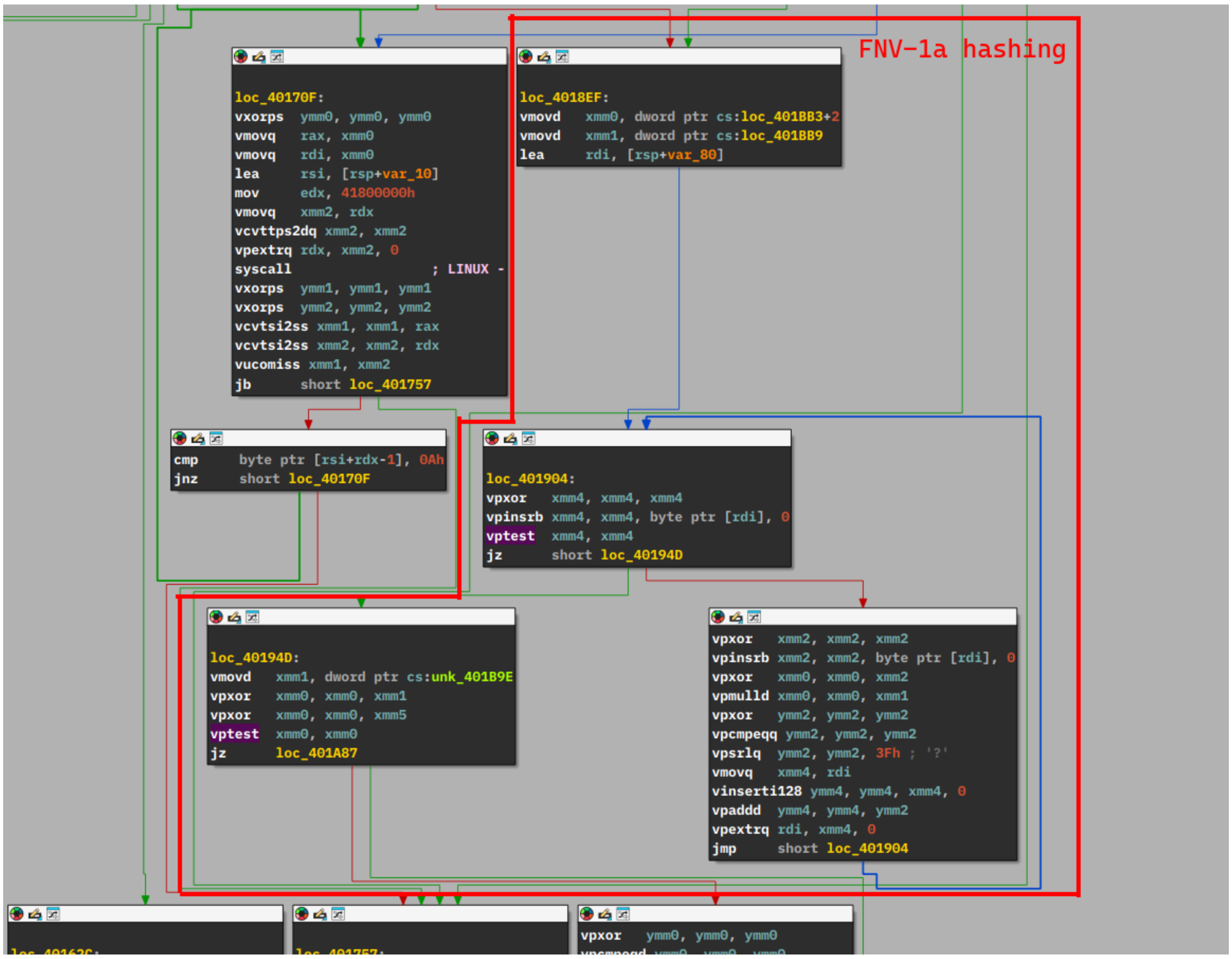
We can reverse the logic here to allow stepping through the program with a debugger.



```
mov     rbx, rax
test    rbx, rbx
jnz     loc_401181

loc_401181:
mov     eax, 42740000h
vmovq   xmm0, rax
vcvttps2dq xmm0, xmm0
vpextrq rax, xmm0, 0
mov     rdi, rbx
lea     rsi, [rsp+var_8]
vxorps  ymm2, ymm2, ymm2
vmovq   rdx, xmm2
vmovq   r10, xmm2
syscall                 ; LINUX -
lea     rbx, loc_4011C5
vmovss  xmm0, [rsp+var_8]
vptest  xmm0, xmm0
cmovnz  rax, rbx
jmp     rax
```

If we step through the code, at some point the program will print the line `Please enter your name:` and read the voter's name. We can follow the data read by the syscall and see that it's being processed by the instructions below. The registration code is verified by hashing the voter's name with FNV-1a hash then xor it with the bytes `CYBERSCI`.



```
loc_40170F:
vxorps   ymm0, ymm0, ymm0
vmovq    rax, xmm0
vmovq    rdi, xmm0
lea      rsi, [rsp+var_10]
mov      edx, 41800000h
vmovq    xmm2, rdx
vcvttps2dq xmm2, xmm2
vpextrq  rdx, xmm2, 0
syscall                   ; LINUX -
vxorps   ymm1, ymm1, ymm1
vxorps   ymm2, ymm2, ymm2
vcvtsi2ss xmm1, xmm1, rax
vcvtsi2ss xmm2, xmm2, rdx
vucomiss xmm1, xmm2
jb       short loc_401757
```

```
loc_4018EF:
vmovd    xmm0, dword ptr cs:loc_401BB3+2
vmovd    xmm1, dword ptr cs:loc_401BB9
lea      rdi, [rsp+var_80]
```

```
cmp      byte ptr [rsi+rdx-1], 0Ah
jnz      short loc_40170F
```

```
loc_401904:
vpxor    xmm4, xmm4, xmm4
vpinsrb  xmm4, xmm4, byte ptr [rdi], 0
vptest   xmm4, xmm4
jz       short loc_40194D
```

```
loc_40194D:
vmovd    xmm1, dword ptr cs:unk_401B9E
vpxor    xmm0, xmm0, xmm1
vpxor    xmm0, xmm0, xmm5
vptest   xmm0, xmm0
jz       loc_401A87
```

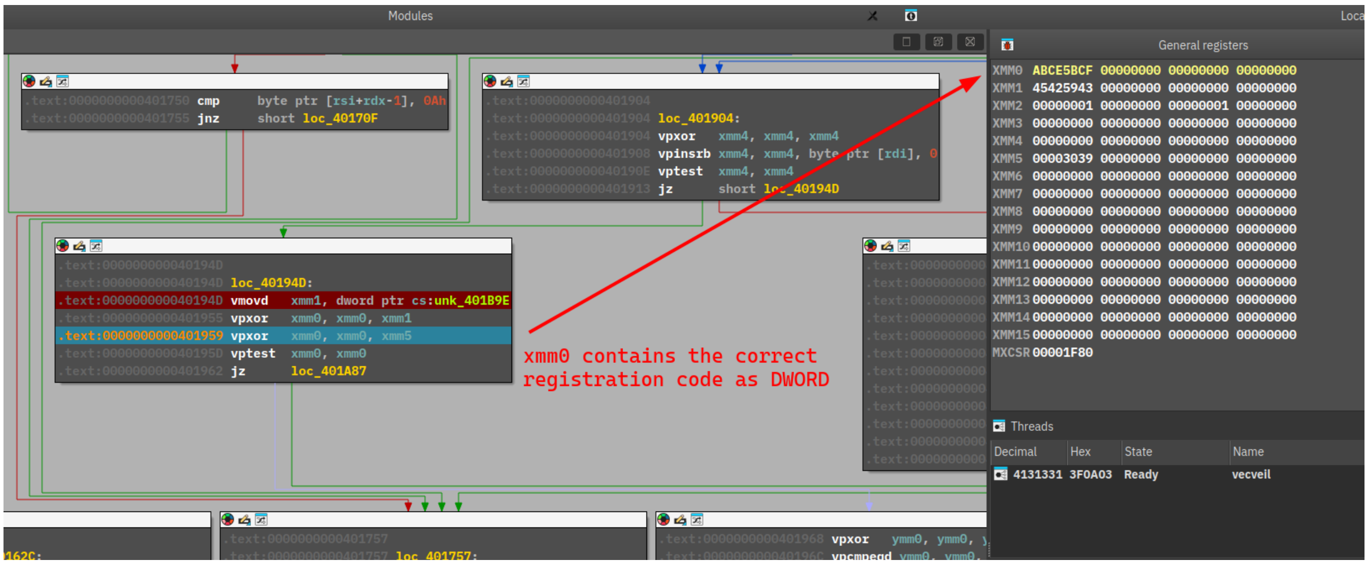
```
vpxor    xmm2, xmm2, xmm2
vpinsrb  xmm2, xmm2, byte ptr [rdi], 0
vpxor    xmm0, xmm0, xmm2
vpmulld  xmm0, xmm0, xmm1
vpxor    ymm2, ymm2, ymm2
vpcmpeqq ymm2, ymm2, ymm2
vpsrlq   ymm2, ymm2, 3Fh ; '?'
vmovq    xmm4, rdi
vinserti128 ymm4, ymm4, xmm4, 0
vpaddd   ymm4, ymm4, ymm2
vpextrq  rdi, xmm4, 0
jmp      short loc_401904
```

```
loc_40162C:
```

```
loc_401757:
```

```
vpxor    ymm0, ymm0, ymm0
vpcmpeqd ymm0, ymm0, ymm0
```

If we enter `Mateo Alvarez` as the name and a random registration code, we can catch the correct calculated hash in memory stored in the `xmm0` register here.



```
.text:0000000000401750 cmp      byte ptr [rsi+rdx-1], 0Ah
.text:0000000000401755 jnz      short loc_40170F
```

```
.text:0000000000401904
.text:0000000000401904 loc_401904:
.text:0000000000401904 vpxor    xmm4, xmm4, xmm4
.text:0000000000401908 vpinsrb  xmm4, xmm4, byte ptr [rdi], 0
.text:000000000040190E vptest   xmm4, xmm4
.text:0000000000401913 jz       short loc_40194D
```

```
.text:000000000040194D
.text:000000000040194D loc_40194D:
.text:000000000040194D vmovd    xmm1, dword ptr cs:unk_401B9E
.text:0000000000401955 vpxor    xmm0, xmm0, xmm1
.text:0000000000401959 vpxor    xmm0, xmm0, xmm5
.text:000000000040195D vptest   xmm0, xmm0
.text:0000000000401962 jz       loc_401A87
```

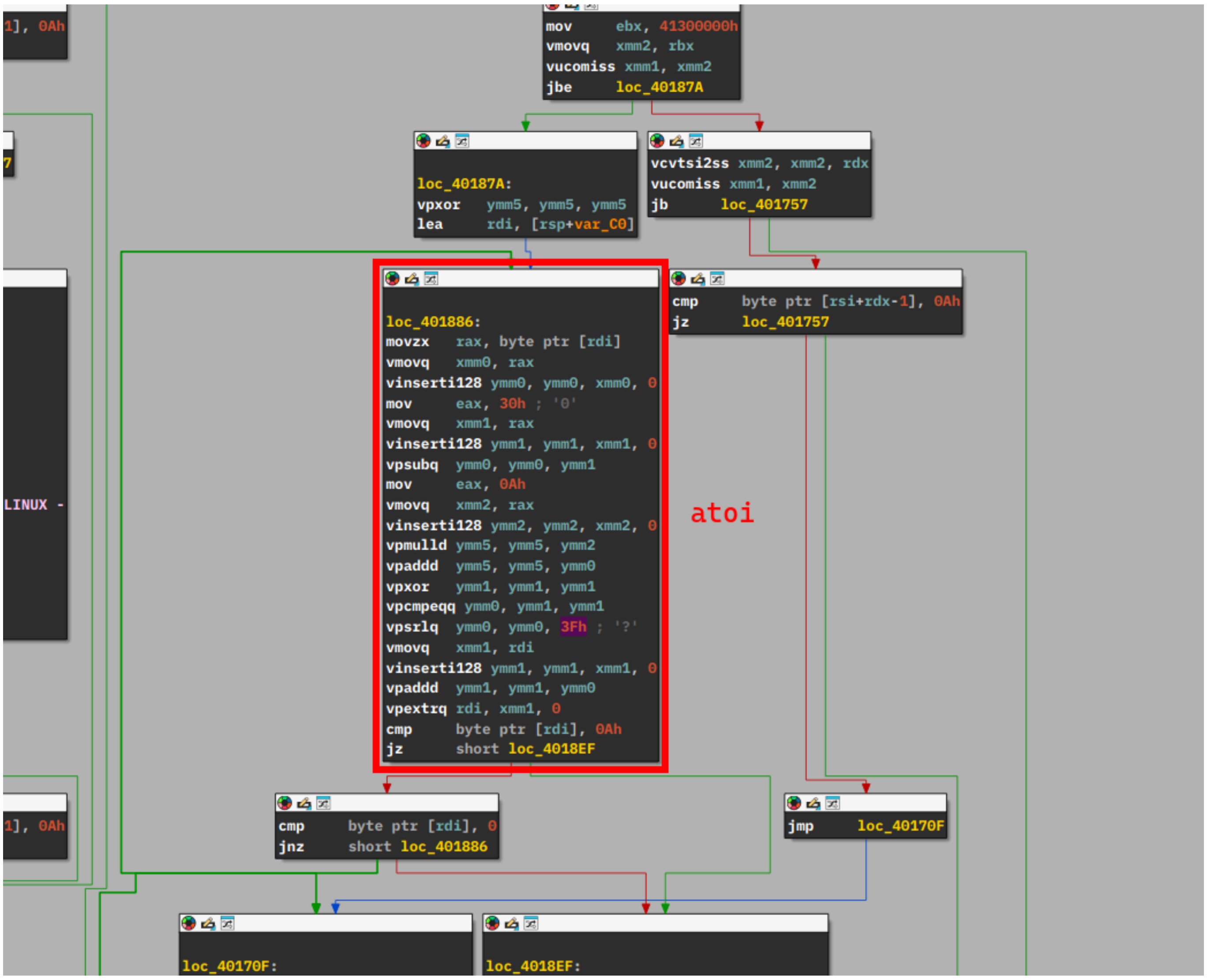
```
.text:0000000000401757
.text:0000000000401757 loc_401757:
```

```
.text:0000000000401968 vpxor    ymm0, ymm0,
.text:000000000040196C vpcmpeqd ymm0, ymm0,
```

```
XMM0  ABCE5BCF 00000000 00000000 00000000
XMM1  45425943 00000000 00000000 00000000
XMM2  00000001 00000000 00000001 00000000
XMM3  00000000 00000000 00000000 00000000
XMM4  00000000 00000000 00000000 00000000
XMM5  00003039 00000000 00000000 00000000
XMM6  00000000 00000000 00000000 00000000
XMM7  00000000 00000000 00000000 00000000
XMM8  00000000 00000000 00000000 00000000
XMM9  00000000 00000000 00000000 00000000
XMM10 00000000 00000000 00000000 00000000
XMM11 00000000 00000000 00000000 00000000
XMM12 00000000 00000000 00000000 00000000
XMM13 00000000 00000000 00000000 00000000
XMM14 00000000 00000000 00000000 00000000
XMM15 00000000 00000000 00000000 00000000
MXCSR 00001F80
```

Decimal	Hex	State	Name
4131331	3F0A03	Ready	vecveil

Now if we follow how the input registration code is processed, it is parsed by this code which is the `atoi` function written in AVX2. This means the registration code we need to enter is the number `0xABCE5BCF` in base 10.



```
mov     ebx, 41300000h
vmovq   xmm2, rbx
vucomiss xmm1, xmm2
jbe     loc_40187A
```

```
loc_40187A:
vpxor   ymm5, ymm5, ymm5
lea     rdi, [rsp+var_C0]
```

```
vcvtsi2ss xmm2, xmm2, rdx
vucomiss xmm1, xmm2
jb      loc_401757
```

```
loc_401886:
movzx   rax, byte ptr [rdi]
vmovq   xmm0, rax
vinserti128 ymm0, ymm0, xmm0, 0
mov     eax, 30h ; '0'
vmovq   xmm1, rax
vinserti128 ymm1, ymm1, xmm1, 0
vpsubq  ymm0, ymm0, ymm1
mov     eax, 0Ah
vmovq   xmm2, rax
vinserti128 ymm2, ymm2, xmm2, 0
vpmulld ymm5, ymm5, ymm2
vpaddd  ymm5, ymm5, ymm0
vpxor   ymm1, ymm1, ymm1
vpcmpeqq ymm0, ymm1, ymm1
vpsrlq  ymm0, ymm0, 3Fh ; '?'
vmovq   xmm1, rdi
vinserti128 ymm1, ymm1, xmm1, 0
vpaddd  ymm1, ymm1, ymm0
vpextrq rdi, xmm1, 0
cmp     byte ptr [rdi], 0Ah
jz      short loc_4018EF
```

```
cmp     byte ptr [rsi+rdx-1], 0Ah
jz      loc_401757
```

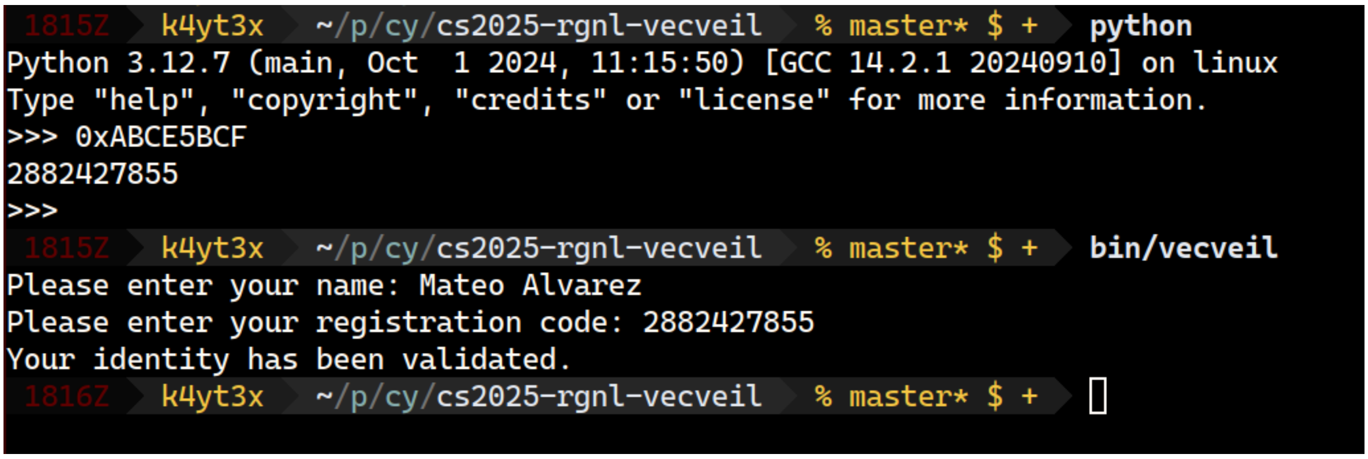
```
cmp     byte ptr [rdi], 0
jnz     short loc_401886
```

```
jmp     loc_40170F
```

```
loc_40170F:
```

```
loc_4018EF:
```

If we convert the number from hex to base 10 and enter it as the registration code, we will pass the identity validation, solving this challenge. The answer is 2882427855.



```
1815Z  k4yt3x  ~/p/cy/cs2025-rgnl-vecveil  % master* $ +  python
Python 3.12.7 (main, Oct  1 2024, 11:15:50) [GCC 14.2.1 20240910] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 0xABCE5BCF
2882427855
>>>
1815Z  k4yt3x  ~/p/cy/cs2025-rgnl-vecveil  % master* $ +  bin/vecveil
Please enter your name: Mateo Alvarez
Please enter your registration code: 2882427855
Your identity has been validated.
1816Z  k4yt3x  ~/p/cy/cs2025-rgnl-vecveil  % master* $ +
```