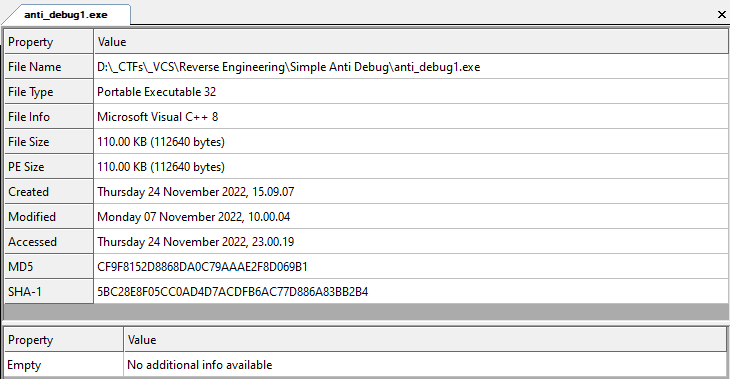
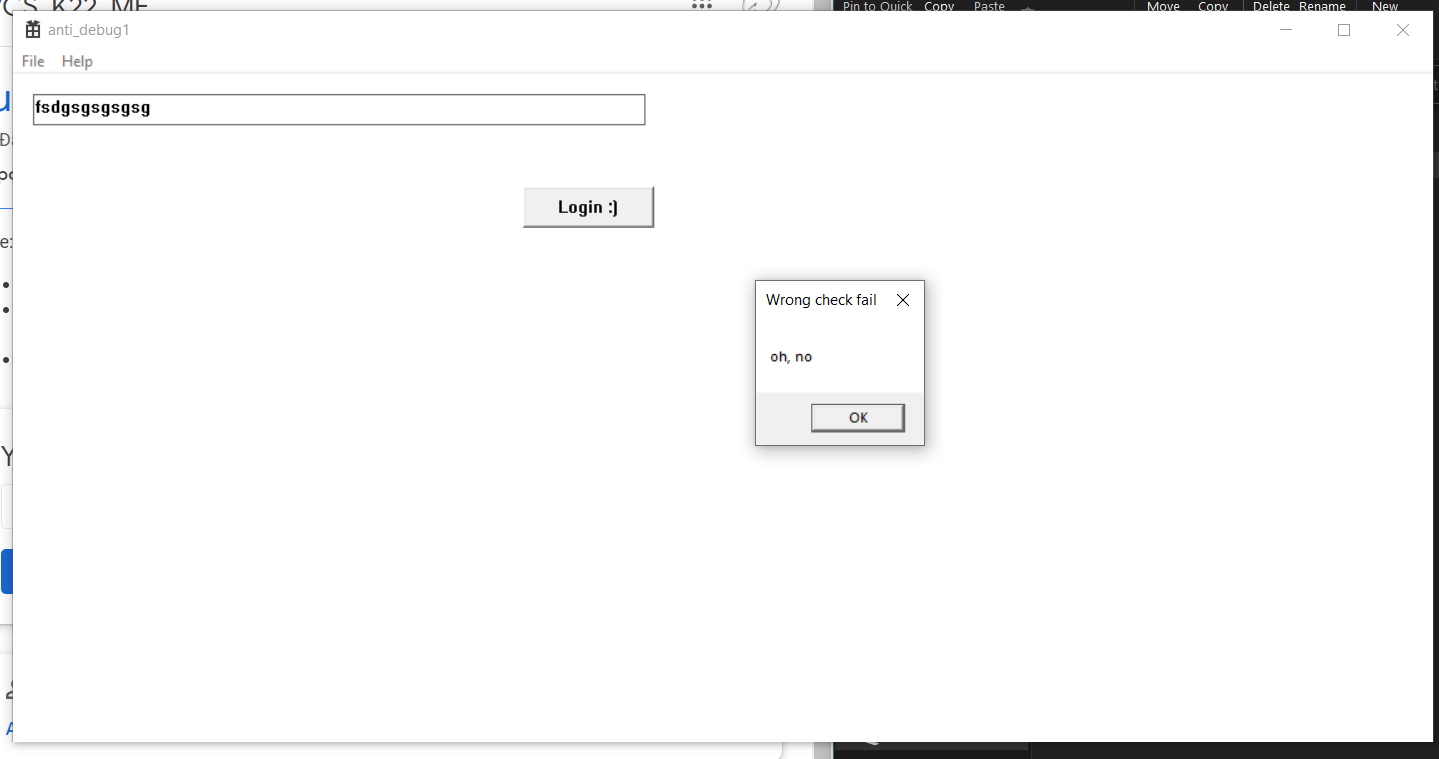
**Week2 – Simple Anti Debug**

1. **Simple Anti Debug**

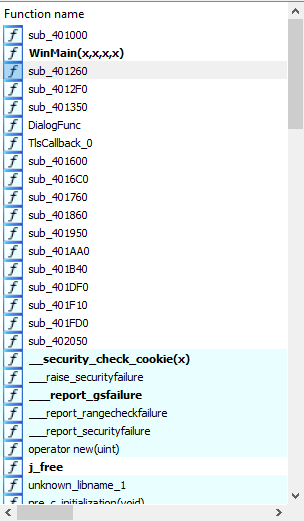
****

Chương trình yêu cầu chạy file với quyền administrator, ta có một chương trình đơn giản nhập vào chuỗi ký tự password, nếu sai thì sẽ thông báo “wrong check fail” .



Có thể thấy luồng thực thi của chương trình khi chạy bình thưởng như này khá là đơn giản nhưng theo tên bài thì ở đây chúng ta có một bài Anti Debug.

Đầu tiên mình kiểm tra các chuỗi có trong chương trình thông qua view substrings và không có chuỗi nào dạng thông báo dạng kiểu thành công hay nhập mật khẩu đúng cả.



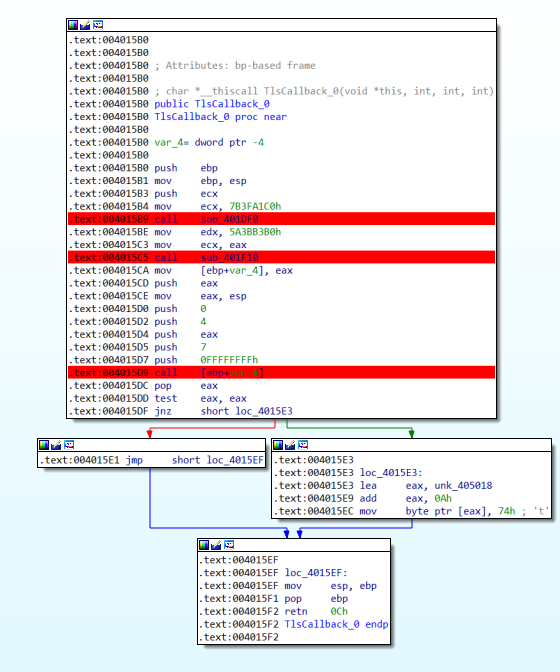
Vì theo đề bài đây là một bài Anti Debug, có một phương pháp antidebug là sử dụng **TlsCallback**.

*Checking for the presence of a debugger in the main function is not the best idea, as this is the first place a reverser will look when viewing a disassembler listing. Checks implemented in main can be erased by NOP instructions thus disarming the protection. If the CRT library is used, the main thread will already have a certain call stack before transfer of control to the main function. Thus a good place to perform a debugger presence check is in the TLS Callback. Callback function will be called before the executable module entry point call.*

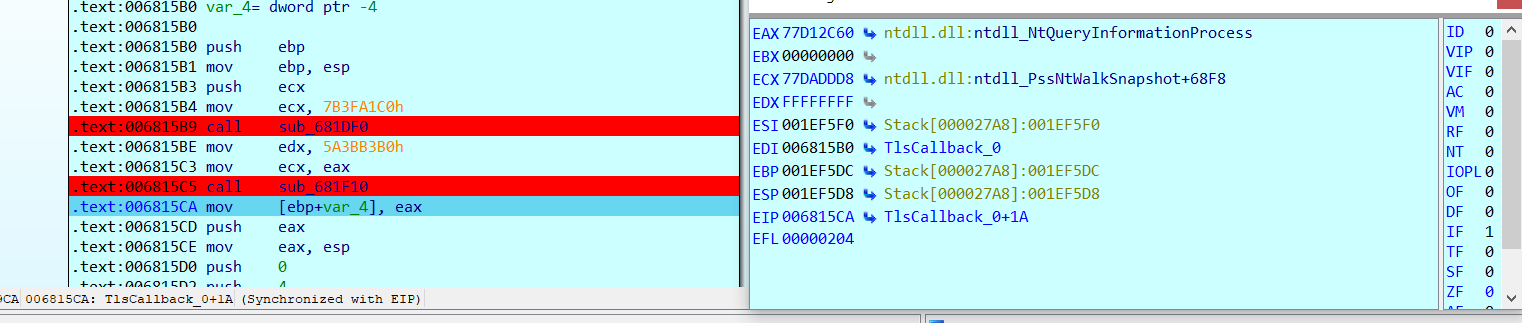
Tóm tắt lại thì là các hàm kiểm tra xem chương trình có đang ở trạng thái Debug không như **IsDebuggerPresent** sẽ được đặt ở hàm TlsCallback thay vì main (WinMain) và chương trình sẽ gọi hàm Callback trước khi gọi điểm vào thực thi của chương trình.

Ta thấy hàm **TlsCallback** trong danh sách các hàm của chương trình khi phân tích chương trình bằng IDA, ta sẽ bắt đầu phân tích chương trình từ đó.

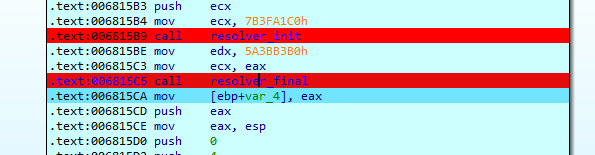
1. **Hàm TLSCallBack**



Chương trình tiến hành gọi hai hàm **sub\_401DF0** và **sub\_401F10,** sau khi gọi hai hàm này ra với các đối số tương ứng thì chúng trả lại cho chúng ta giá trị trong thanh ghi eax rồi lưu tại [ebp-4] để call tới ngay bên dưới, tiến hành debug xem giá trị được trả về là gì ?



* Vậy là sau khi chạy xong hai hàm này chương trình trả lại cho chúng ta hàm **ZwQueryInformationProcess** được lấy từ cấu trúc **PEB** của chương trình.
* Đổi lại tên hai hàm được gọi đến để dễ dàng phân tích hơn.



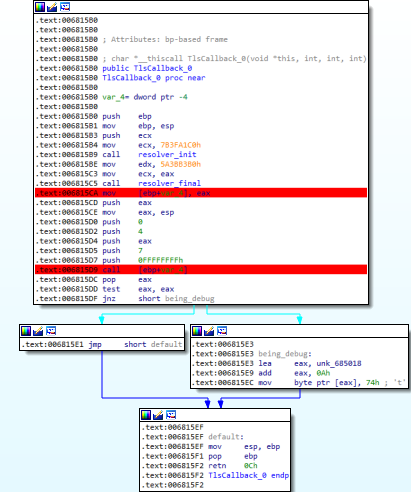
1. **Hàm ZwQueryInformationProcess:**

Ta tiếp tục phân tích chương trình sau khi đã có hai hàm resolver API, **ZwQueryInformationProcess** là một hàm AntiDebug, hàm này sẽ lấy thông tin của process (có thể là chính hàm đó) đễ check xem process có đang được Debug hay không ?

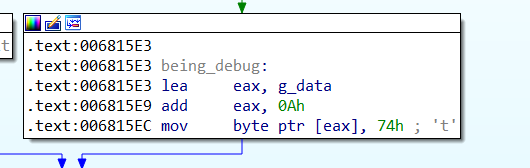
Hàm **ZwQueryInformationProcess** cũng giống hàm **NtQueryInformationProcess** nhưng chỉ khác là **Zw** sẽ được gọi từ **kernel mode.**

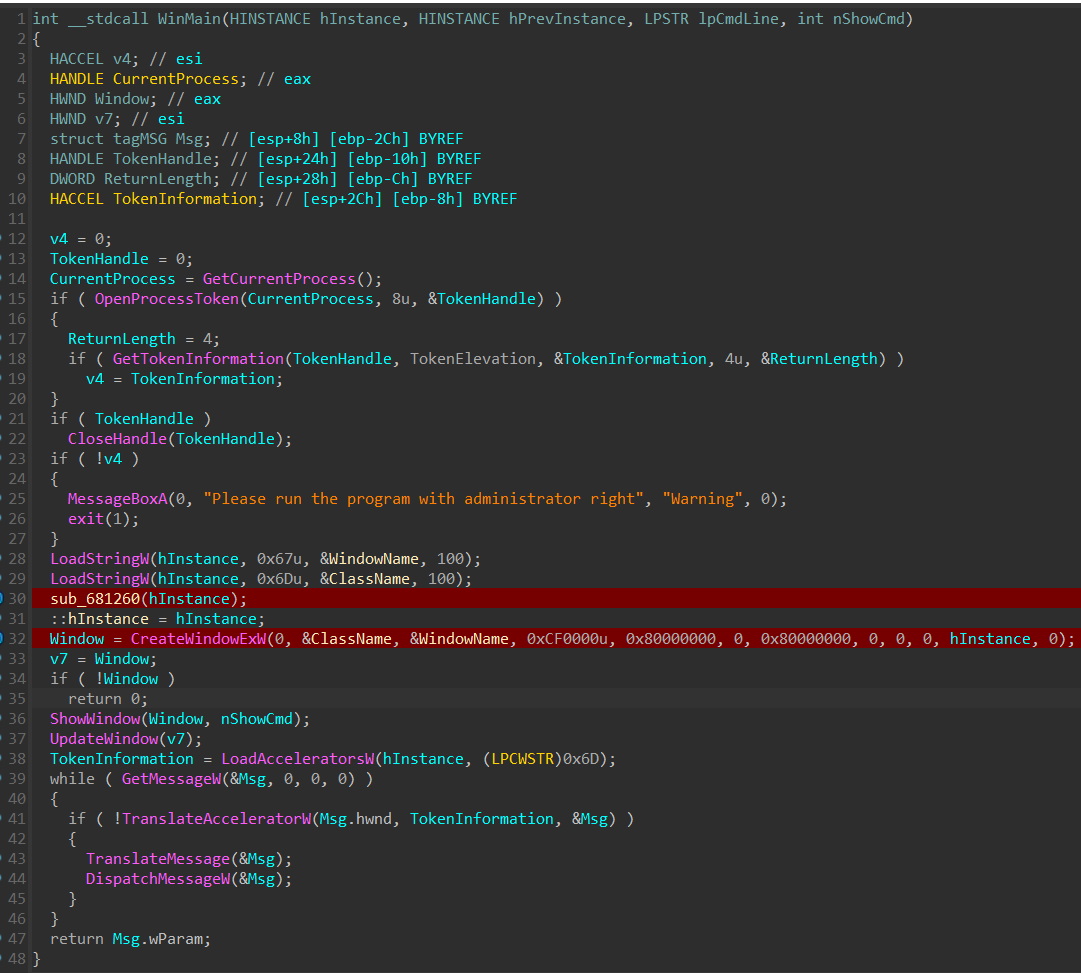


Code TLSCallBack gọi **ZwQueryInformationProcess** của chương trình:

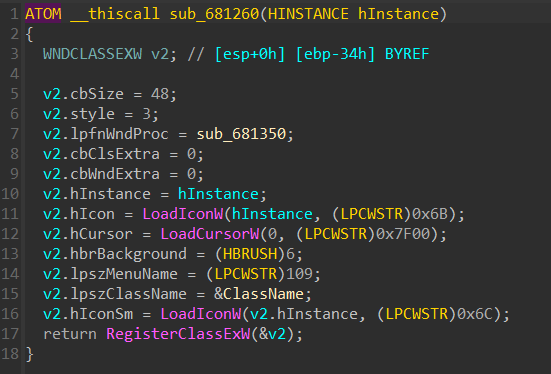


* Vậy hướng chạy khi thực thi của chương trình sẽ không đi qua hàm **being\_debug**.
* Nếu chương trình đang được debug thì dữ liệu tại **unk\_685018+0Ah** sẽ mang giá trị 74h. Đổi tên thành **g\_data** để xem chương trình làm gì với dữ liệu này.



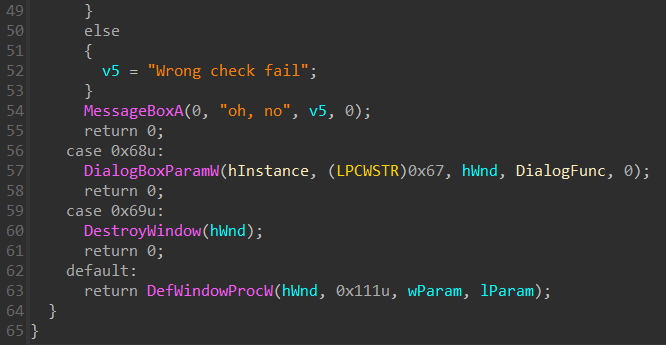
1. **Hàm WinMain:** ****

Hàm **WinMain** gọi hàm **CreateWindowExw** với đối số **hInstance** được khởi tạo từ hàm **sub\_681260**,ta sẽ phân tích theo luồng chính này.

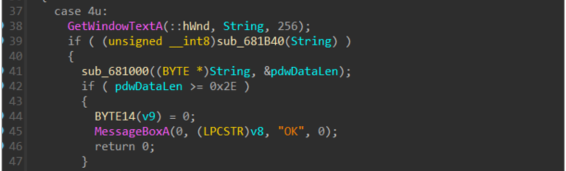
****

Tiếp tục phân tích luồng chính theo hàm **sub\_681350**.

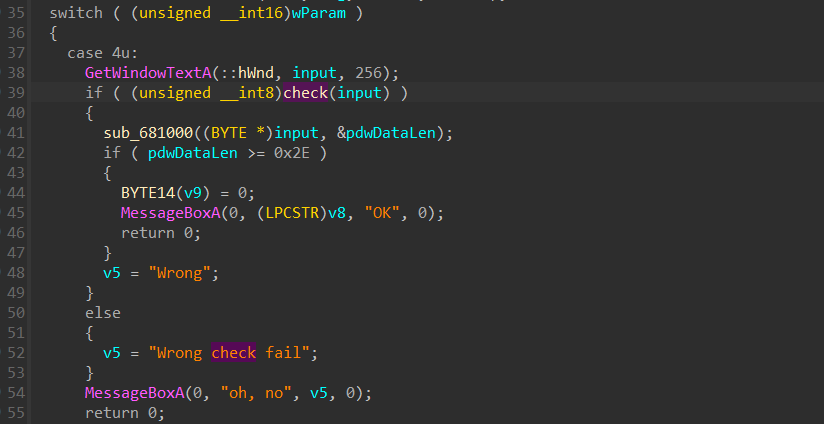




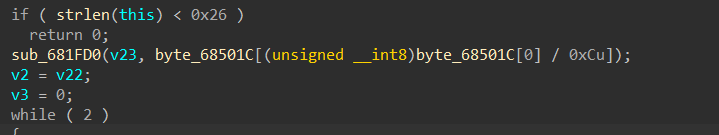
Ở hàm này tại case4 ta thấy có 1 **MessageBoxA** hiển thị tiêu đề Window là OK => Ta đang đi đúng hướng và sẽ phân tích kỹ trường hợp tại case4.



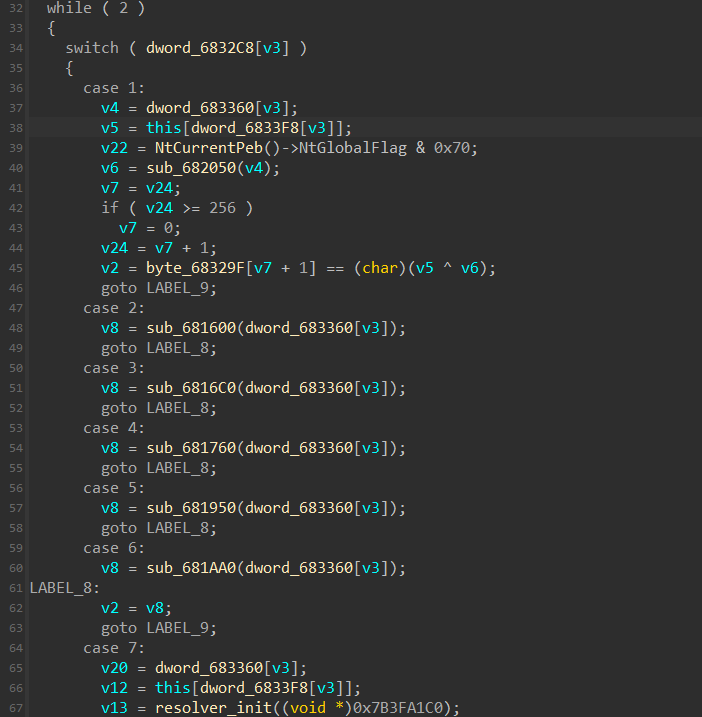
Tại đây ta thấy chương trình sao chép dữ liệu từ cửa sổ ::hWnd là đối số đầu tiên của hàm **sub\_681350** này vào chuỗi String – **lấy input** và kiểm tra dữ liệu đó. Ta đổi lại tên hàm và biến để phân tích.



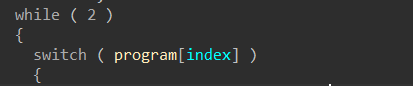
Ta tiến hành phân tích hàm check.



Chương trình gọi đến một hàm có tác động tới byte dữ liệu **g\_data** mà ta đổi tên ở TLSCallBack. Output của hàm này là biến v23. Ta tiến hành đổi tên thành **default\_const** để dễ dàng nhận dạng và phân tích.

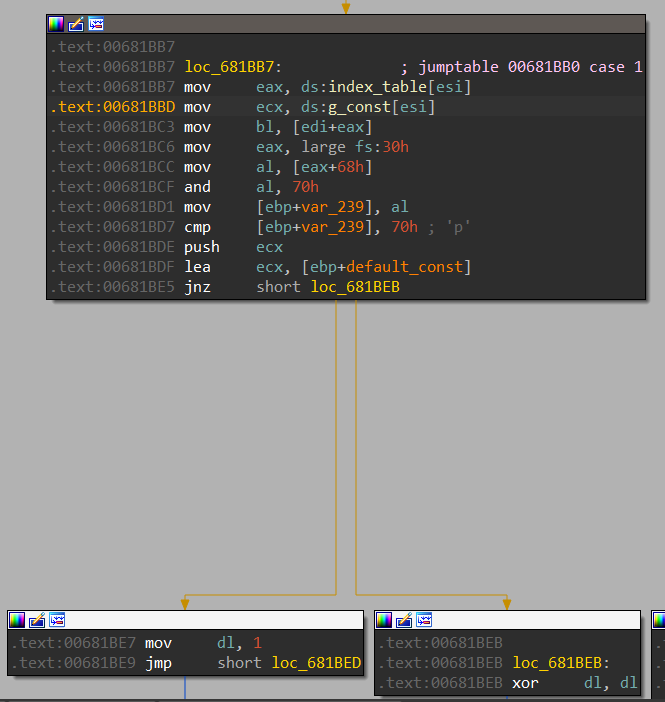


Chương trình sẽ xử lý dữ liệu với 7 case khác nhau trong vòng lặp while, tiến hành đổi tên để dễ dàng phân tích.

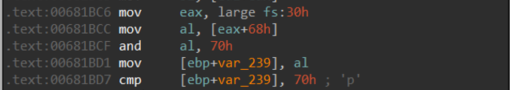


Đọc tiếp các case của chương trình thì ta nhận ra rằng mỗi case của switch case sẽ sử dụng một ký thuật antidebug khác nhau.

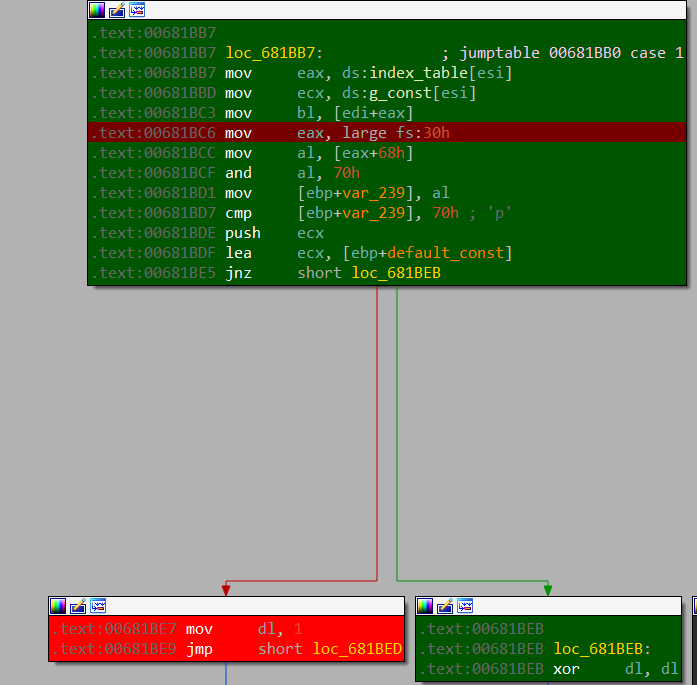
**Case1:**



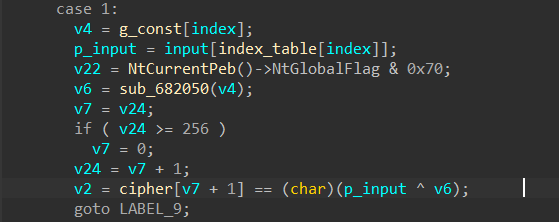
Nếu đọc theo luồng của chương trình thì địa chỉ Input của chúng ta nhập vào vẫn đang nằm ở thanh ghi edi và chương trình sẽ lấy từng kí tự chúng ta nhập vào ví trí là data tại ds:index\_table[esi] , esi đang là index mà chúng ta đã đặt tên cho mã giả ở bên trên.



Trong case 1 này thì 1 kỹ thuật antidebug được sử dụng đến là **NtGlobalFlag** khi so sánh với 0x70 để kiểm tra xem chương trình có đang được debug, nếu được debug thì **NtGlobalFlag** sẽ được set gái trị 0x70. Vậy luồng thực thi đúng là đường màu xanh.



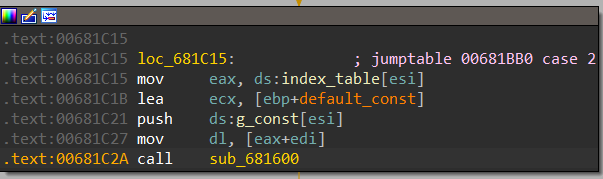
Lúc này dl set giá trị bằng 0 và gọi hàm sub\_682050.



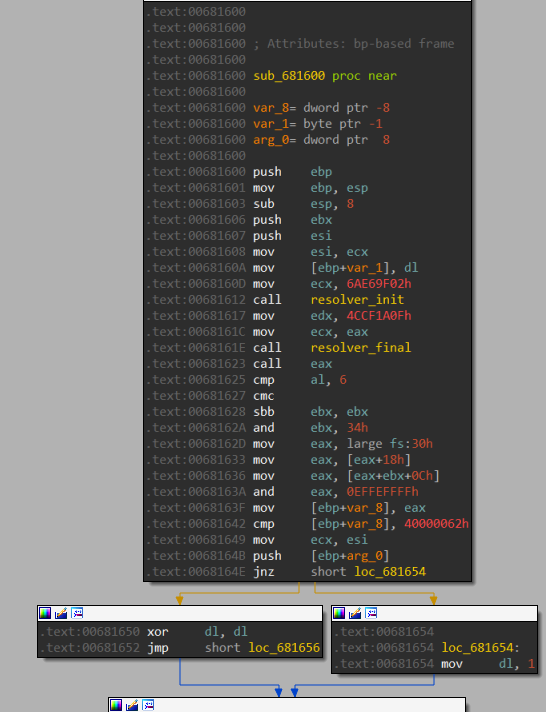
Cơ bản thì case 1 kiểm tra NtGlobalFlag, nếu chương trình không debug thì dl = 0 và được sử dụng tới trong hàm con sub\_682050 sau đó thì giá trị trả về biến v6 được sử dụng tới để XOR với đầu vào input của hàm **check** xem có bằng với gía trị của cipher không.

* Vậy ta kết luận rằng nếu ta set giá trị của v22 hoặc là thanh ghi dl khi debug = 0 thay vì 0x70 thì ta sẽ có được gía trị trả về **v6 đúng** đó XOR lại với cipher thì ta sẽ có input đúng.

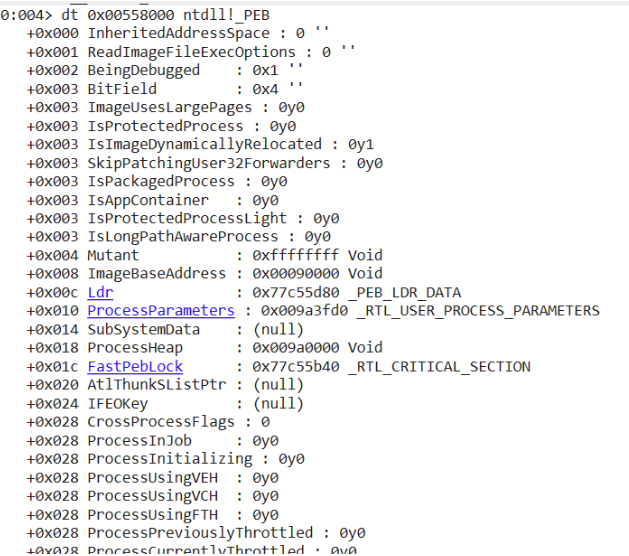
**CASE2:**



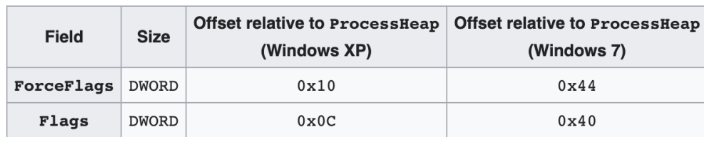
Gọi tới hàm sub\_681600,



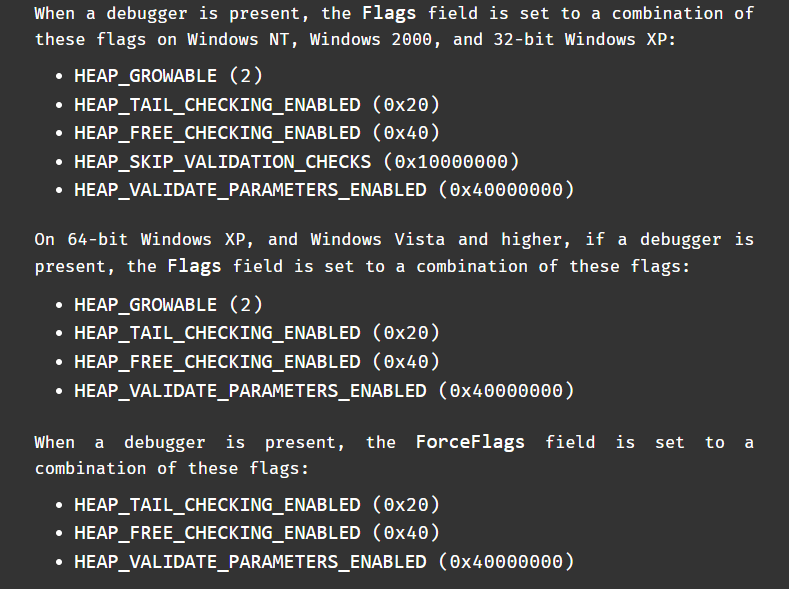
Ta thấy chương trình sẽ tiến hành các bước để resolver API như đã làm trong TLSCallBack nhưng là để gọi tới 1 hàm khác thay vì ZwQueryInformationDebug. Sau khi debug ta check được hàm được call tới là **GetVersion**, sau đó lấy **PEB** và lấy giá trị tại vùng +18h.



Vậy là chương trình sẽ lấy **ProcessHeap** và chương trình sử dụng hàm **GetVersion** để lấy **Flags** và **ForceFlags** tùy vào version **=> đây là kỹ thuật antidebug Heap Flags .**

****

Khi thử debug chúng ta sẽ thấy Flags nằm ở 0x40 và ForceFlags nằm ở 0x44.

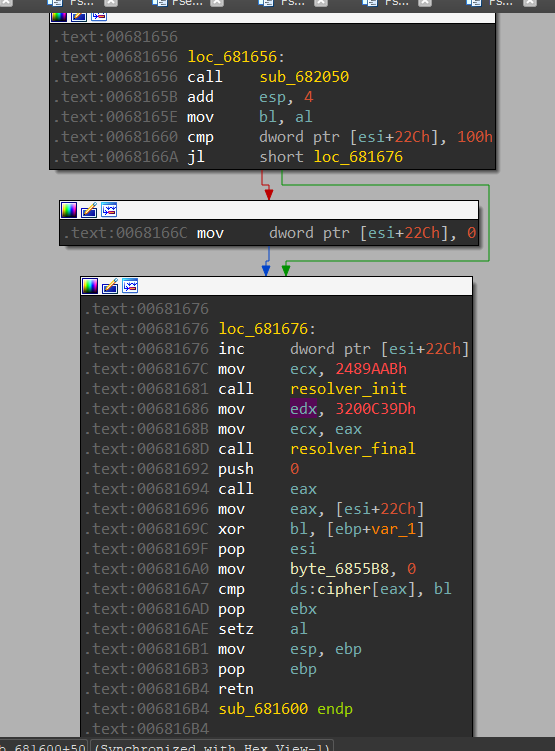


Trong Case thứ 2 này, Chúng ta có thể thấy chương trình đang check Flags(0x40) của chúng ta và phiên bản window của mình mới hơn nên nó sẽ giống như :

Flags(0x40) = HEAP\_GROWABLE(2) | HEAP\_TAIL\_CHECKING\_ENABLED(0x20) | HEAP\_FREE\_CHECKING\_ENABLED(0x40) | HEAP\_VALIDATE\_PARAMETERS\_ENABLED (0x40000000)

=>>> Flags(0x40) = 0x40000062

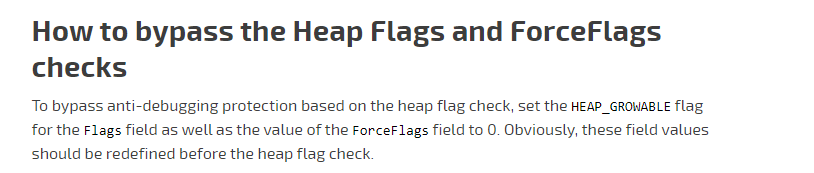
* Để bypass ta chỉ cần set giá trị không bằng 0x40000062 và rồi sau đó thì chương trình lại làm y hệt như với case1 đó là so gái trị đúng thì sẽ có được key đúng.

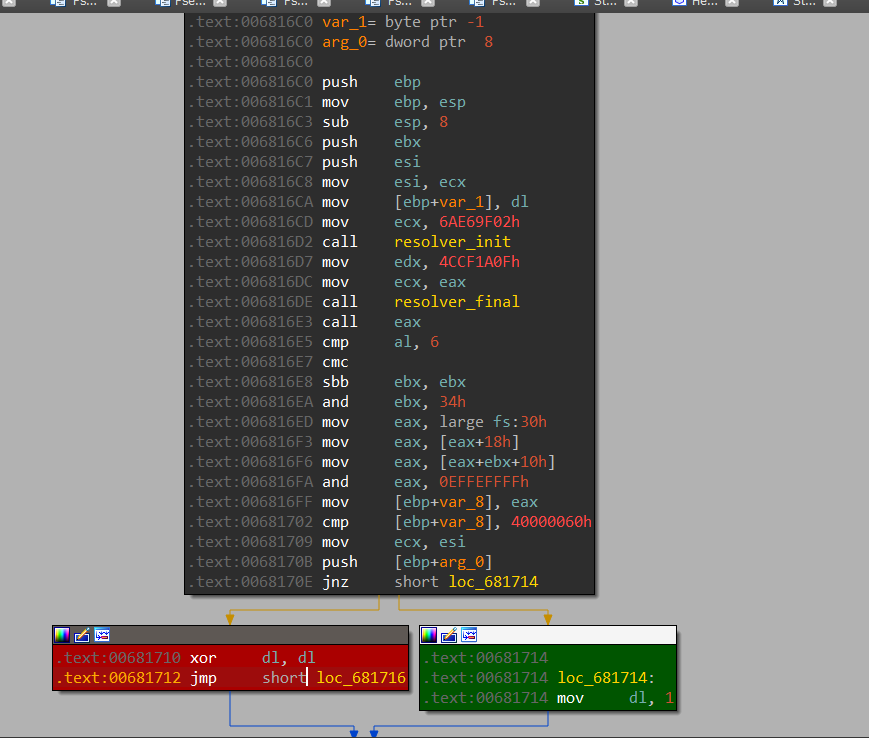


**CASE3**

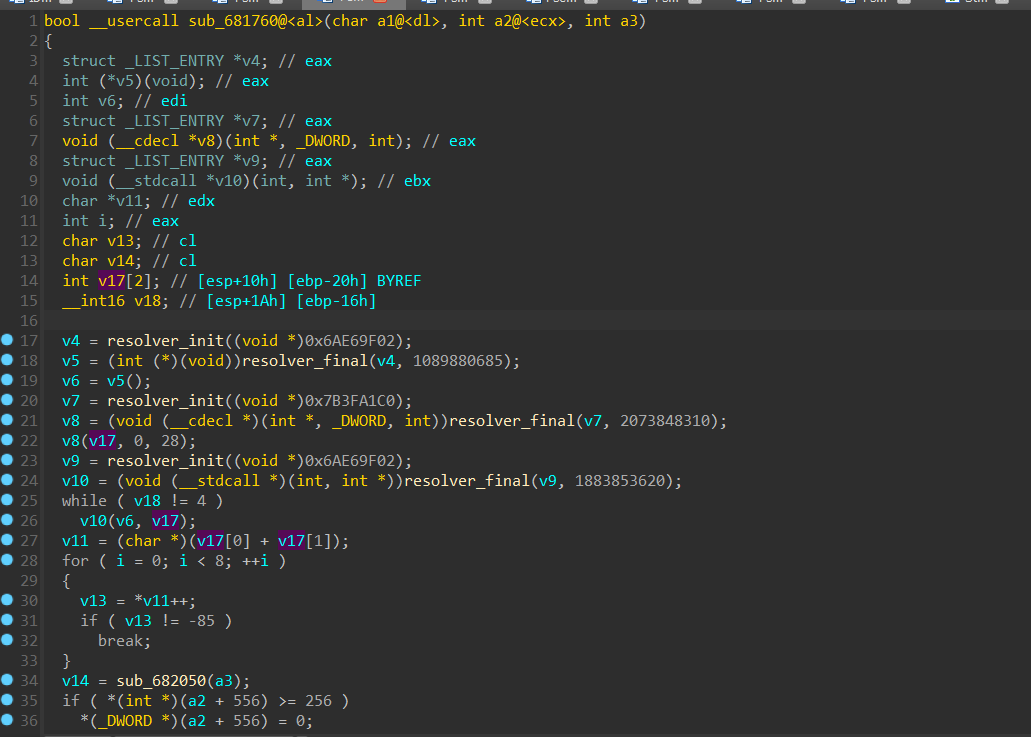
Hàm ở trường hợp cũng giống như case2, cũng là ký thuật Heap Flags nhưng thay vì là sử dụng Flags(0x40) thì ở đây ta sử ForceFlags(0x44).

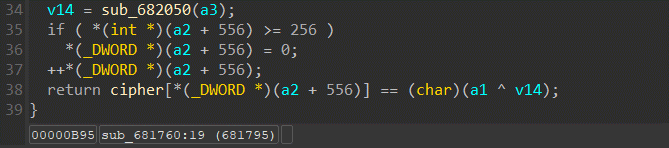
Cách bypass giống y hệt với case2:





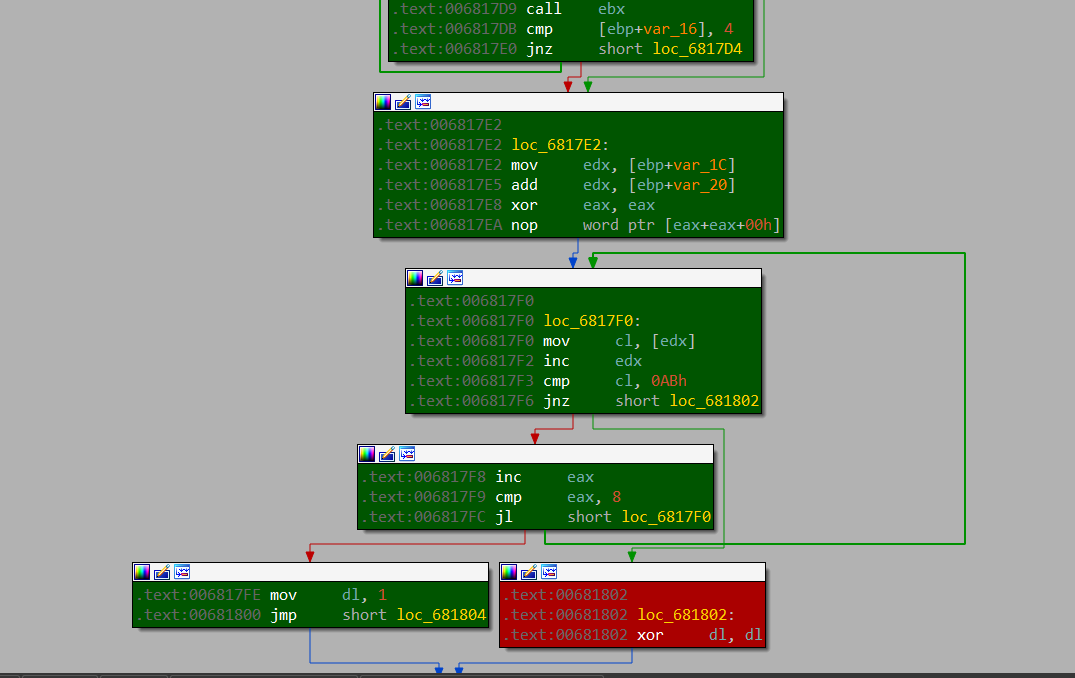
**CASE4**





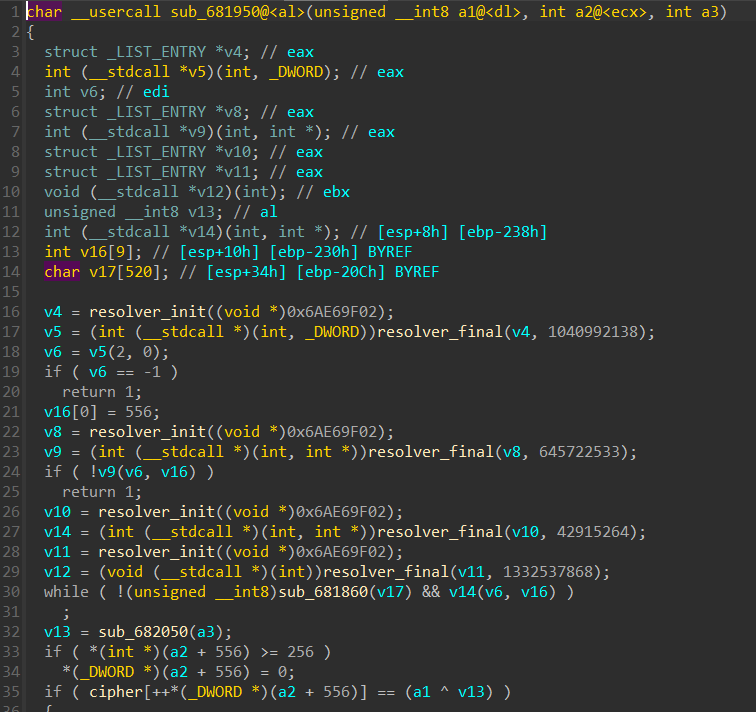
Đoạn cuối vẫn là giá trị đúng thì cho key đúng xor ngược lại với cipher của chương trình thì sẽ được input đúng mà ta tìm nhưng có lẽ là một ký thuật antidebug khác.

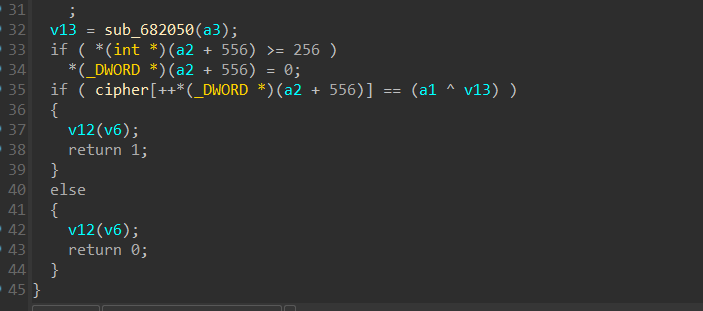
3 hàm được resolver ta check được trong lúc debug lần lượt là kernel32\_GetProcessHeap, ntdll\_memset, kernel32\_HeapWalk.



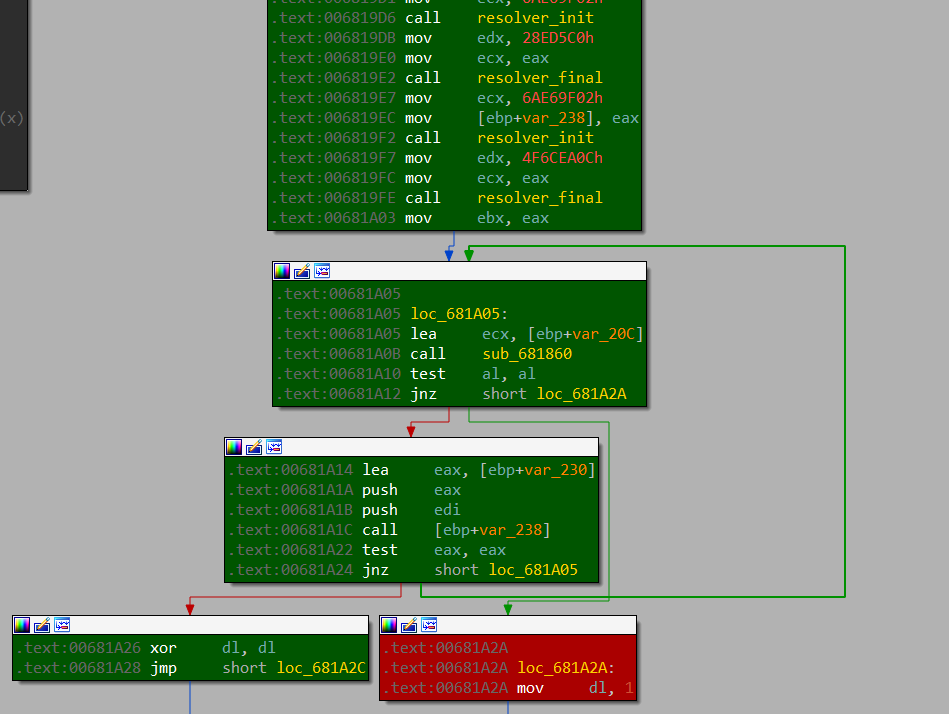
Thực sự thì đến đoạn này làm mãi không hiểu đây là antidebug kỹ thuật gì nhưng kinh nghiệm ở ba case trước thì nếu dl được set để chương trình đi theo mũi tên màu đỏ thì sẽ là luồng thực thi khi run không debug và sẽ sinh ra được key đúng từ hàm nhận dl là đối số nên mình set thử và nó cho ra kết quả đúng thật =)))) .

**CASE5:**

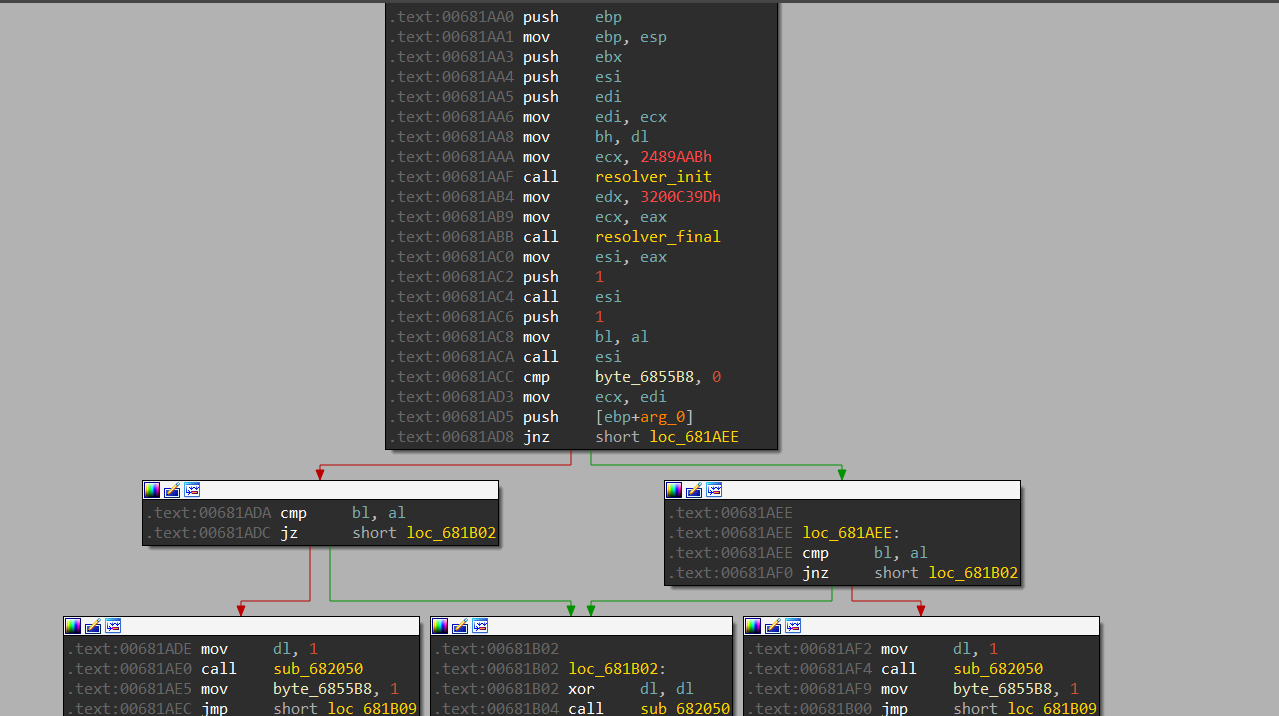




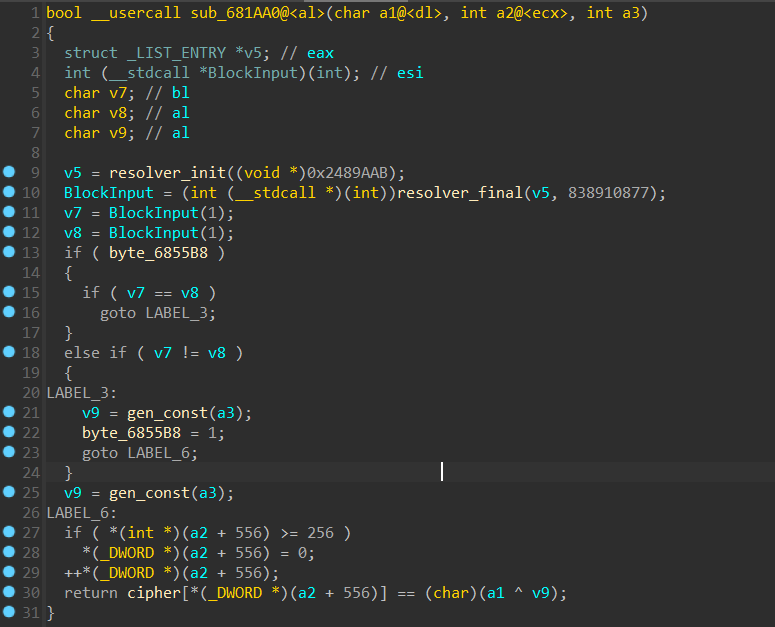
Lại tiếp tục là resolver các hàm API để kiểm tra chương trình có đang debug không, ta tsex bypass tương tự các case trên.

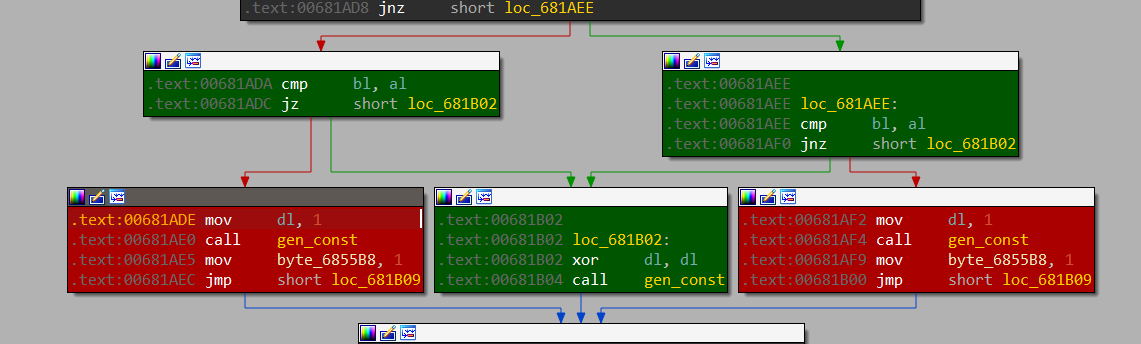


**CASE6:**

****

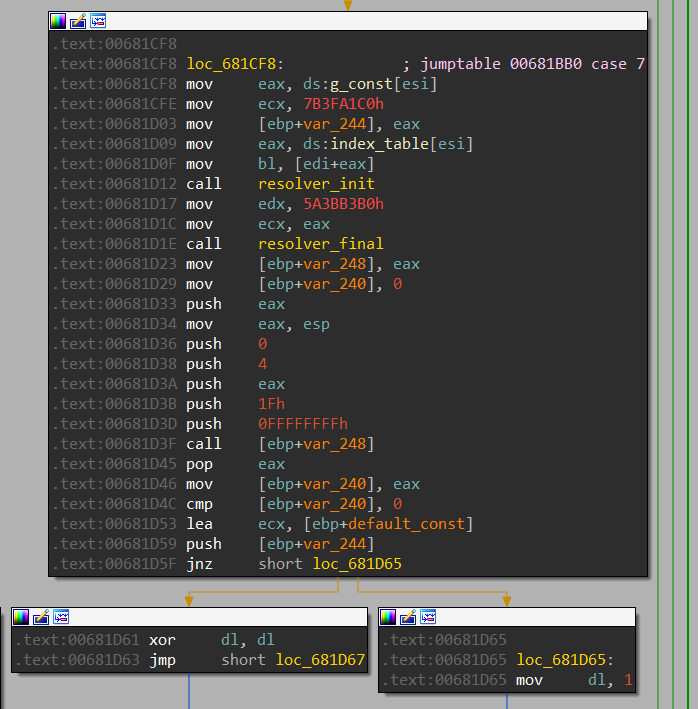
Case này thì hàm được resolver là BlockInput.



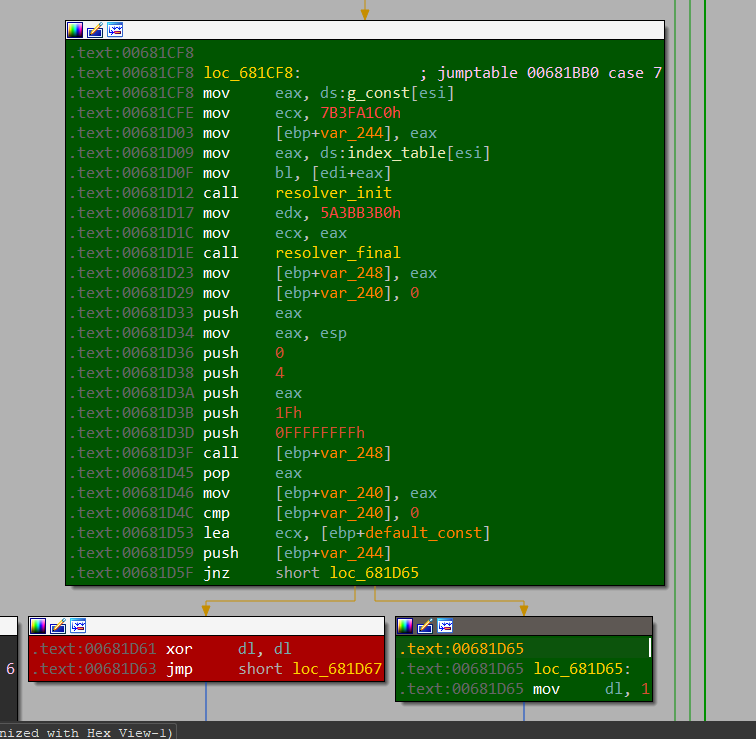


Set đúng giá trị của dl khi debug thì ta sẽ đi đúng hướng chương trình khi không debug.

**CASE7:**

****

Hàm này resolver NtQueryInformationProcess, ta xử lý như đã làm trong TLSCallBack.

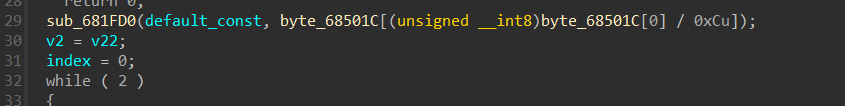


**Kết luận:**

Sau tất cả các case chúng ta thấy rằng chương trình sẽ đều gọi hàm gen\_const và trả về giá trị để xor với input của chúng ta, sau đó so sánh với cipher. Vậy **gen\_const** nhận vào tham số thứ 1:

* Data check debug (1 hoặc 0).
* G\_const
* default\_const(Sinh ra ở bên trên hàm check - Hàm đã sử dụng data được thay đổi TLSCallback).

Vậy thì ta sẽ debug và lấy nguyên **default\_const** được sinh ra – v2.



Sau đó code lại các switch – case và sử dụng hàm gen\_const của chương trình – file **script.cpp**.

Run chương trình ta được flag

**I\_10v3-y0U\_\_wh3n Y0u=c411..M3 Senor1t4 .**