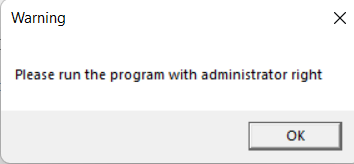
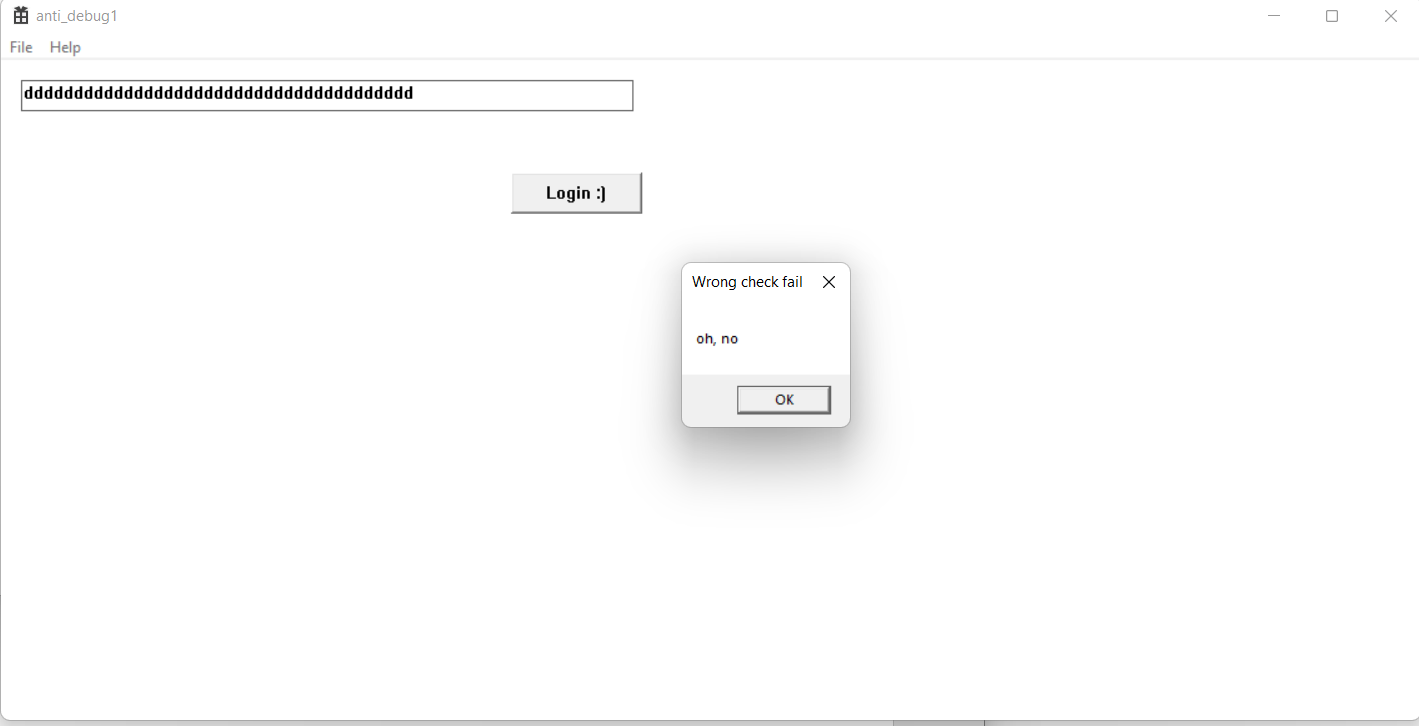
Chạy thử chương trình



Chương trình yêu cầu phải chạy dưới quyền admin

Chạy chương trình ta có

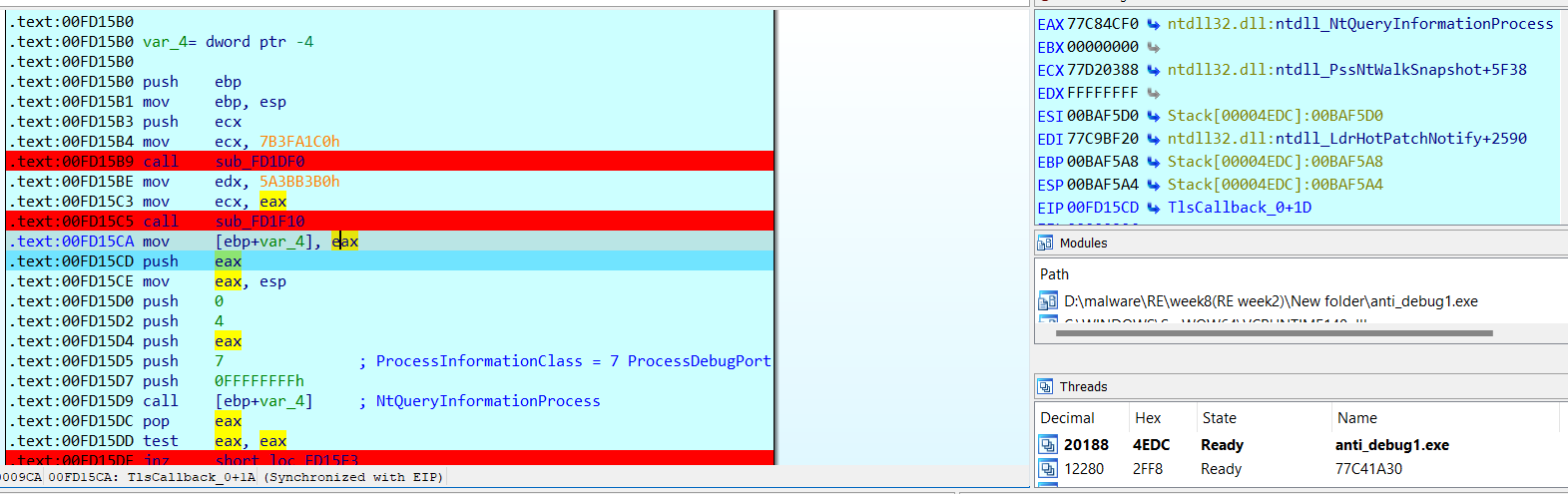


Dự đoán cần nhập vào 1 chuỗi nếu nhập đúng pass

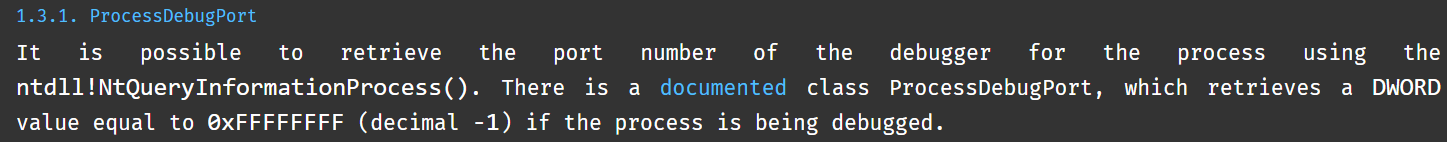
Dùng phân tích ida chạy chương trình

Có hàm TlsCallback

TLS (Thread Local Storage) CallBack là 1 cơ chế của cho phép hàm này được gọi khi 1 luồng được tạo. Đây là 1 mẹo để chống debug Code trong hàm này sẽ được thực thi trước thời điểm tệp thực thi được thực hiện nó sẽ kiểm tra sự hiện diện của debugger.

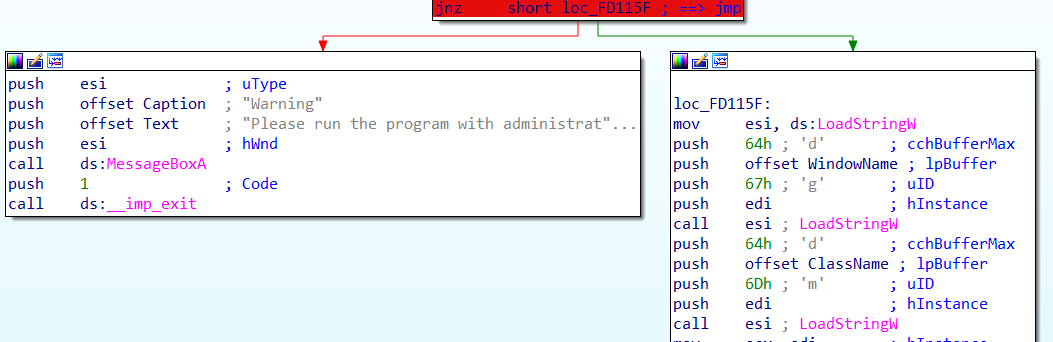


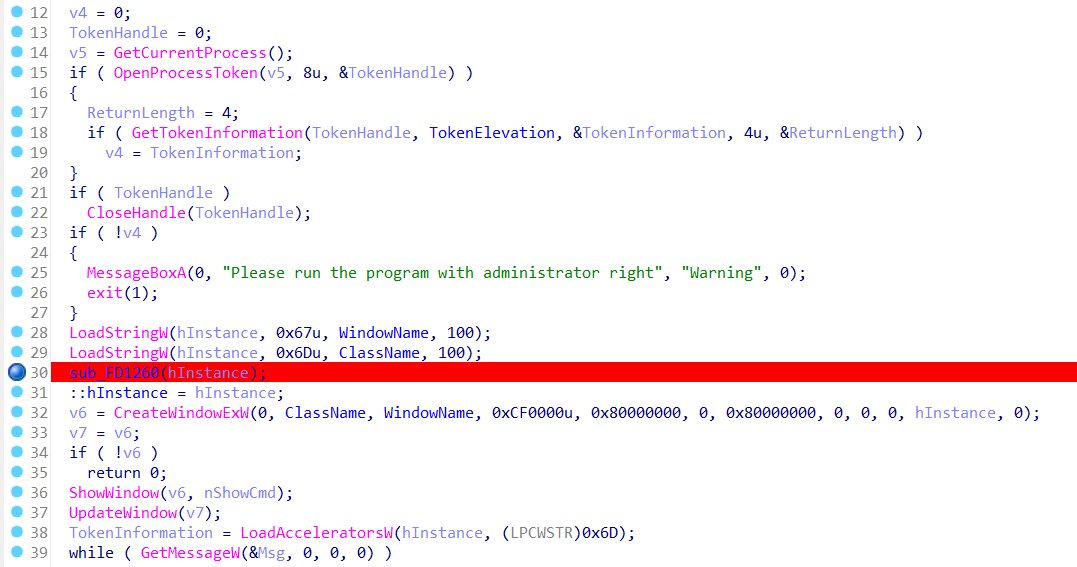
Debug tại đây ta xác định được kĩ thuật sử dụng ở đây **NtQueryInformationProcess** với trường **ProcessInformationClass** = 7 ProcessDebugPort

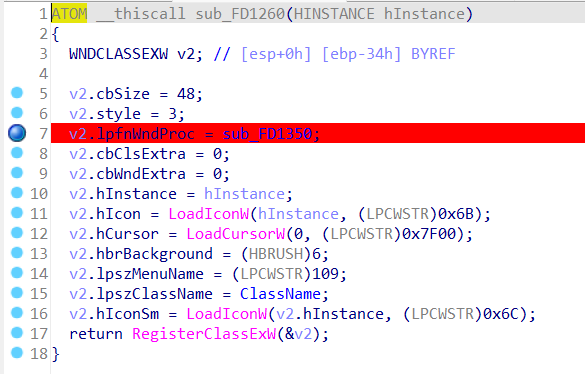


https://anti-debug.checkpoint.com/techniques/debug-flags.html#using-win32-api-ntqueryinformationprocess

Hàm main của chương trình có đoạn yêu cầu phải dạy dưới quyền admin



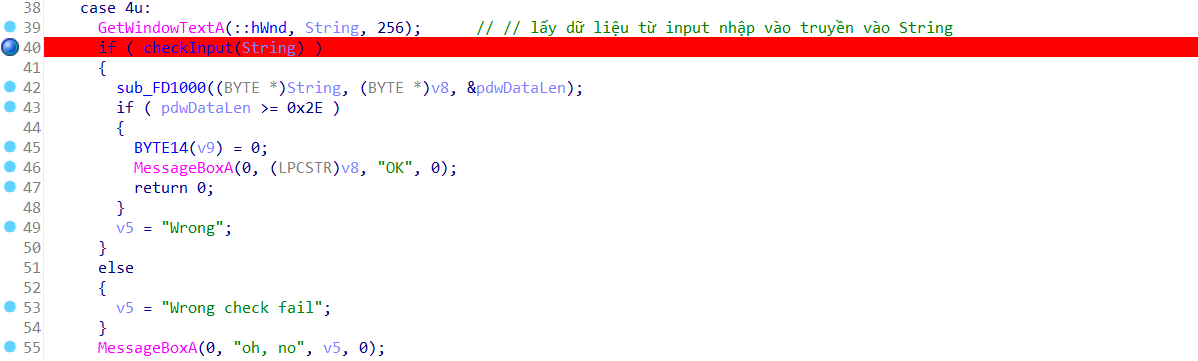


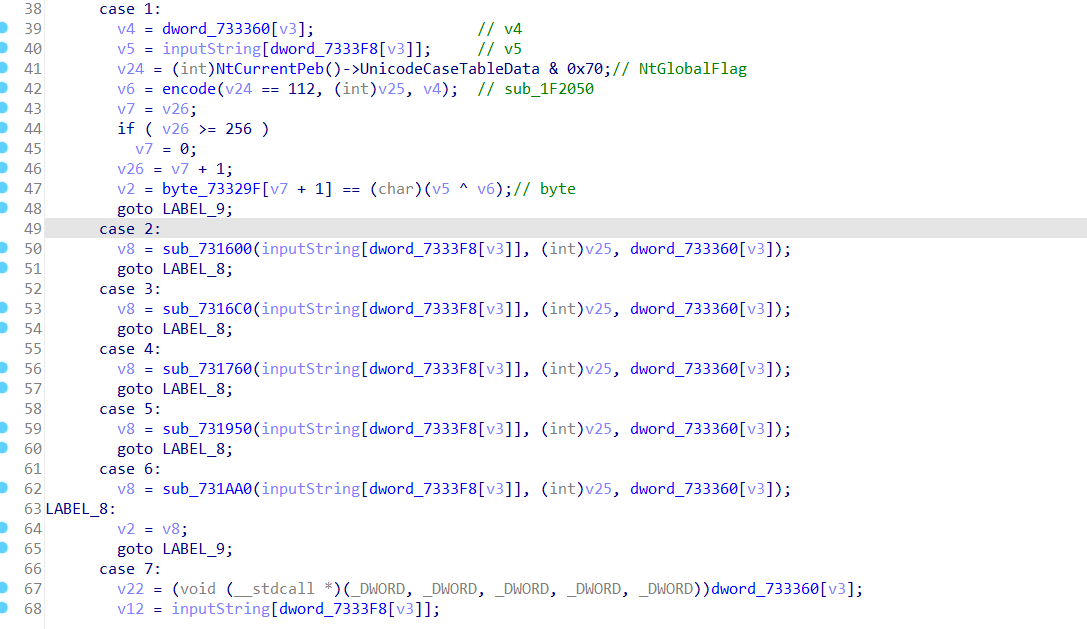


Đây là hàm xác định Struct WDVCLASSEX để truyền vào RegisterClassExw

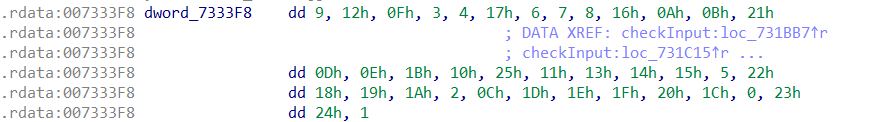
Từ đây ta xác định được luồng thực thi chính của chương trình

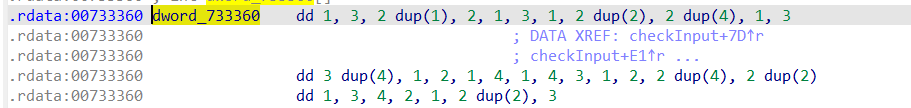






Cả 8 case đều có điểm chung là thông qua 3 tham số **inputString[dword\_7333F8[v3]], (int)v25, dword\_733360[v3]** để tính v2

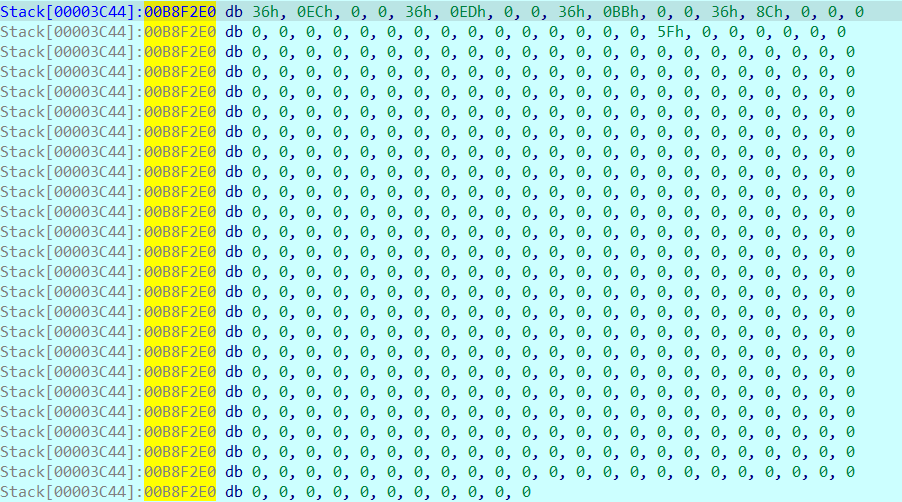






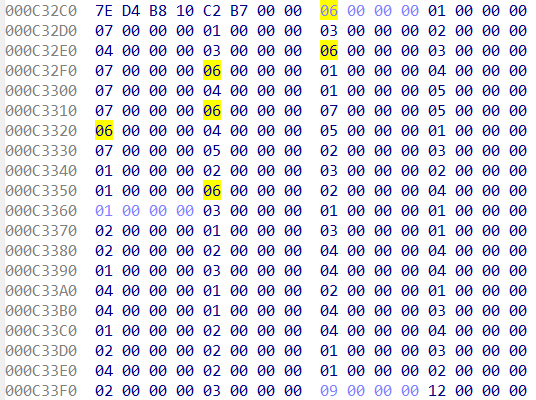
Mảng **dword\_7333F8 và dword\_733360** ta dễ dàng tìm ra còn mảng **v25** phải thông qua hàm **sub\_731FD0** cách đơn giản nhất là debug rồi tìm giá trị của mảng này

Debug để tìm v25 ta được



Mảng kiểu dword C32C8 có giá trị trong đoạn [1;7] sẽ quyết định case nào được thực hiện

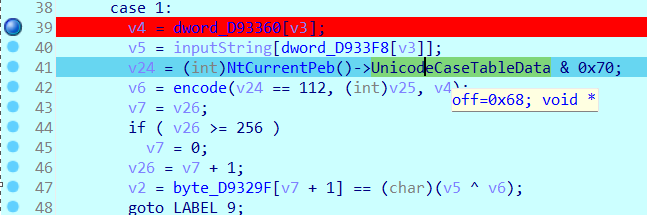


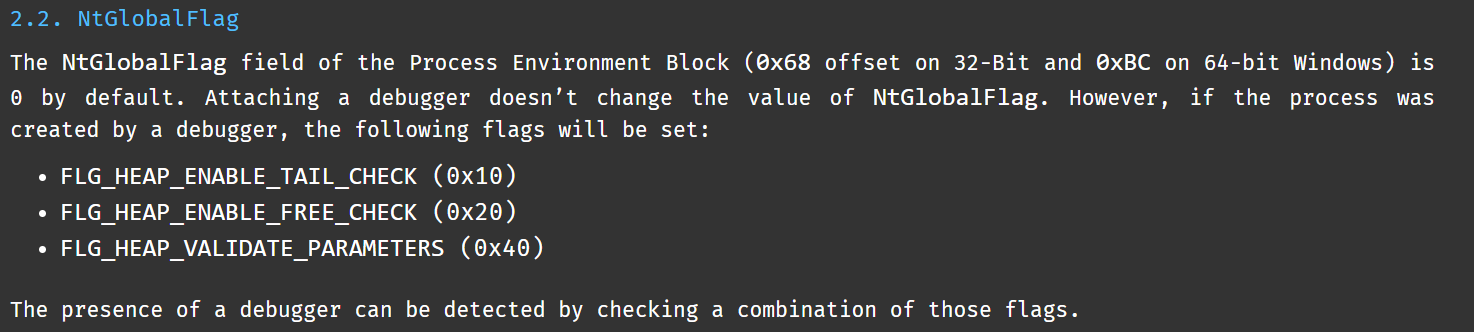


Hình giá trị của mảng dword\_C32C8

Anti-Debug

* Case1: **NtGlobalFlag**

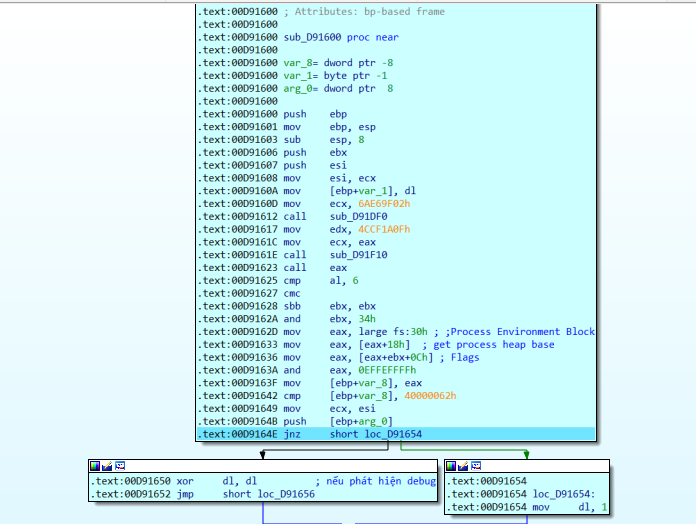


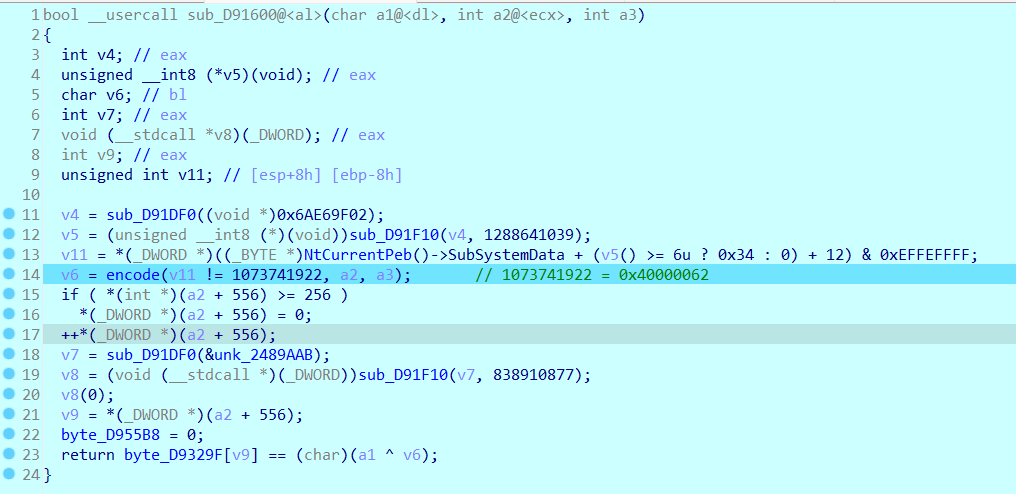


chương trình kiểm tra có phải nằm trong môi trường debug hay ko nếu có thì **NtGlobalFlag** được đặt giá trị là 0x70 (= 112) => v24==112 true hay nếu là phát hiện debug tham số thứ nhất có **giá trị là 1**

* Case2: **Flags – Heap Flag**







Heap flag chứa 2 cờ được khởi tạo bởi NtGlobalFlag : Flags và Force Flags. Giá trị của 2 trường này không chỉ phủ thuộc vào debugger mà còn phiên bản windows. Ở đây chương trình kiểm tra debugger bằng **Flags (** độ lệch 0Ch so với **heap base** ) với **Force Flags** là lệch 10h

Check với mỗi bản win thì giá trị là như nào

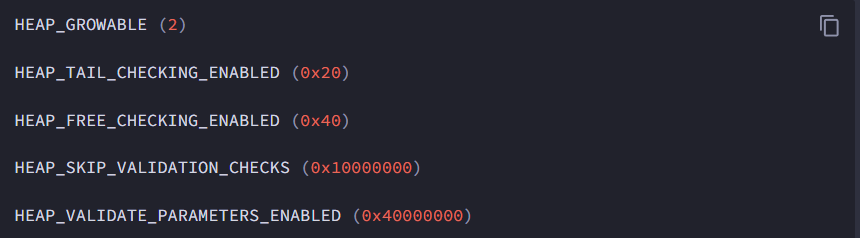
Nếu phát hiện debug thì v11= 0x40000062 hay v11!= 1073741922 false => tham số thứ nhất có **giá trị là 0**

Heap flag chứa 2 cờ được khởi tạo bởi NtGlobalFlag : Flags và Force Flags. Giá trị của 2 trường này không chỉ phủ thuộc vào debugger mà còn phiên bản windows.

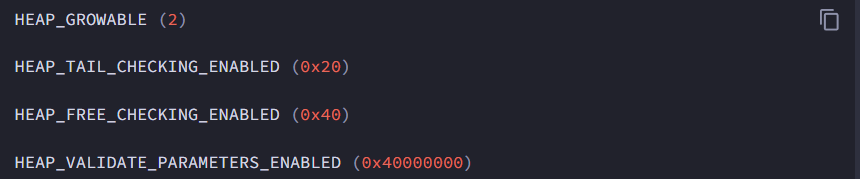
Check với mỗi bản win thì giá trị là như nào

**Với Flags**

* Với Windows NT, Windows 2000 và Windows XP 32 bit, trường Flags (offset 0x0C) sẽ đặt các cờ sau:

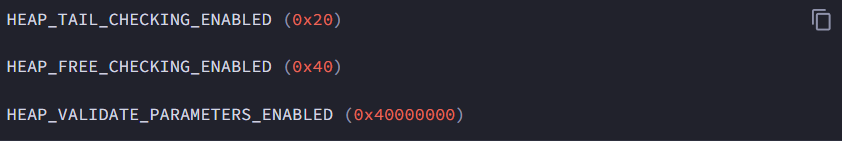


* Trên hệ thống Windows XP 64 bit, Windows Vista và các phiên bản hệ thống mới hơn, trường Flags (offset 0x40) sẽ đặt các cờ sau (ít hơn HEAP\_SKIP\_VALIDATION\_CHECKS (0x10000000)):

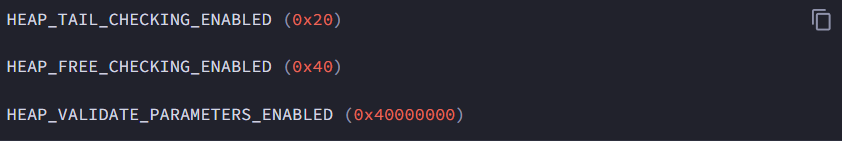


**Với Force Flags**

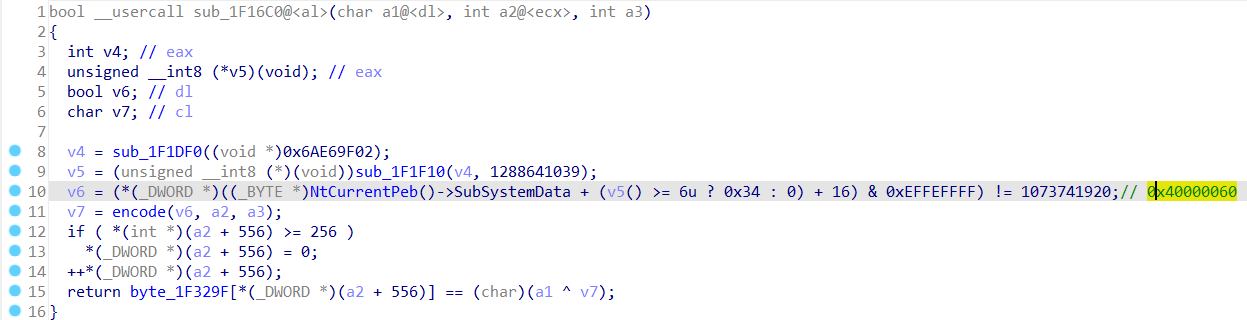
* Với Windows NT, Windows 2000 và Windows XP 32 bit, trường ForceFlags (offset 0x10) sẽ đặt các cờ sau:



* Trên hệ thống Windows XP 64 bit, Windows Vista và các phiên bản hệ thống mới hơn, trường Flags (offset 0x44) sẽ đặt các cờ sau



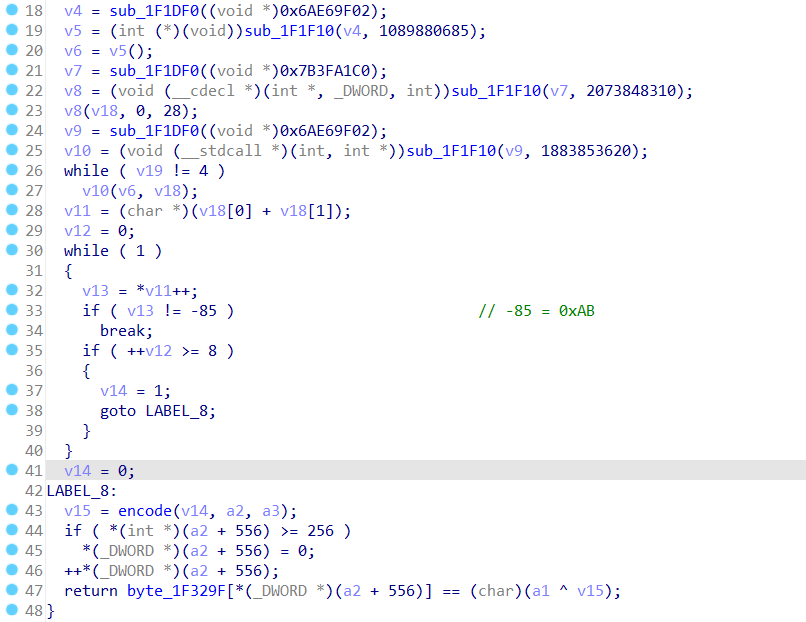
* Case 3: **ForgeFlags - Heap Flag**



Kĩ thuật tương tự case 2 nhưng ở đây kiểm tra **ForgeFlags**

Nếu debug phát hiện debug thì v11= 0x40000060 hay v11!= 1073741922 false => tham số thứ nhất có **giá trị là 0**

* Case 4: **The heap**



Kiểm tra debug bằng cách kiểm tra nếu cờ **HEAP\_TAIL\_CHECKING\_ENABLED** được đặt thì sẽ thêm 2 chuỗi 0xABABABAB (tổng cộng 8 AB) vào cuối heap được phân bổ . nếu phát hiện debug thì v14=0 => tham số **thứ nhất có giá trị là 1.**

**Tại sao đặt ABABABAB**

0xabababab là mẫu bit cho khối bảo vệ sau khi bộ nhớ được cấp phát bằng HeapAlloc(), LocalAlloc(LMEM\_FIXED), GlobalAlloc(GMEM\_FIXED) or CoTaskMemAlloc()

Nếu thấy mẫu bit 0xabababab, điều đó có nghĩa là đang đọc bộ nhớ sau một khối bộ nhớ đã được cấp phát bởi HeapAlloc(), LocalAlloc(LMEM\_FIXED), GlobalAlloc(GMEM\_FIXED) hoặc CoTaskMemAlloc().

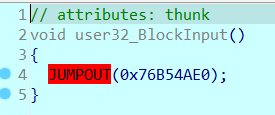
* Case 5: **CreateToolhelp32Snapshot**

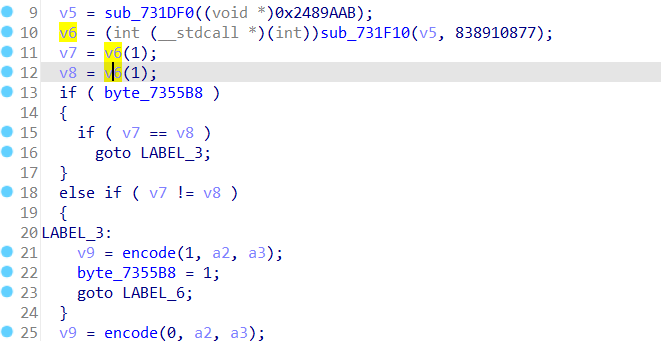


Lấy id tiến trình gốc và tên tiến trình gốc và kiểm tra nếu debug thì giá **trị tham số thứ nhất là 1**

Mô ta đoạn code này- kiểm tra cái j

* Case 6: **BlockInput**



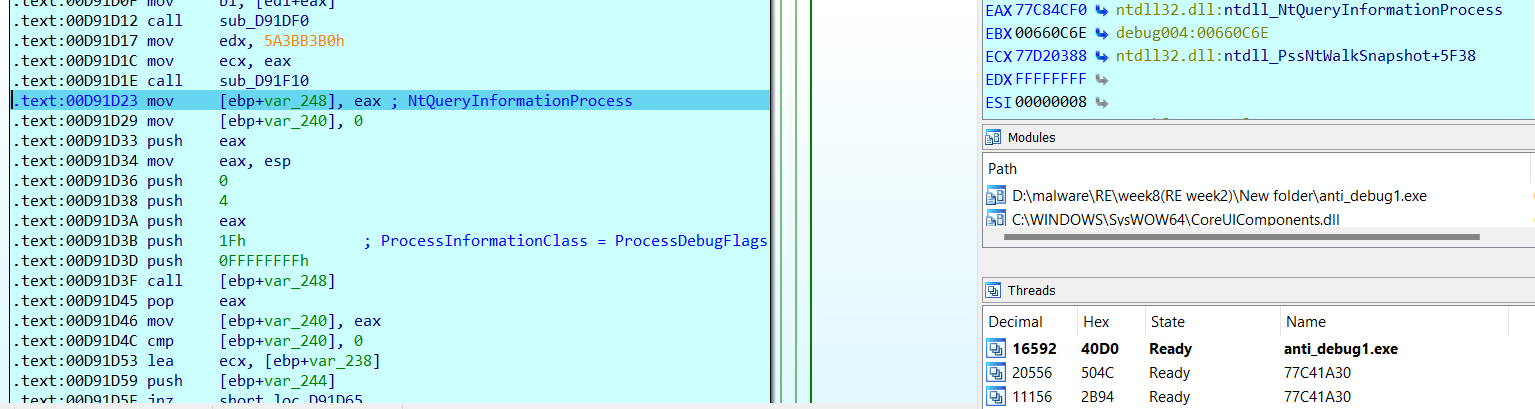


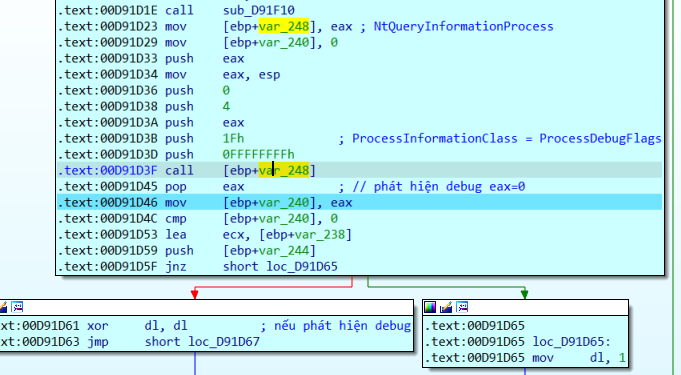
Gọi **BlockInput** 2 lần với tham số đầu vào đều là 1 (true). v7 sẽ trả về true (mặc định) còn v8 nếu có debug thì v8 sẽ trả về true (bình thường sẽ trả về false) ngăn chặn debug bằng cách thực hiện BlockInput

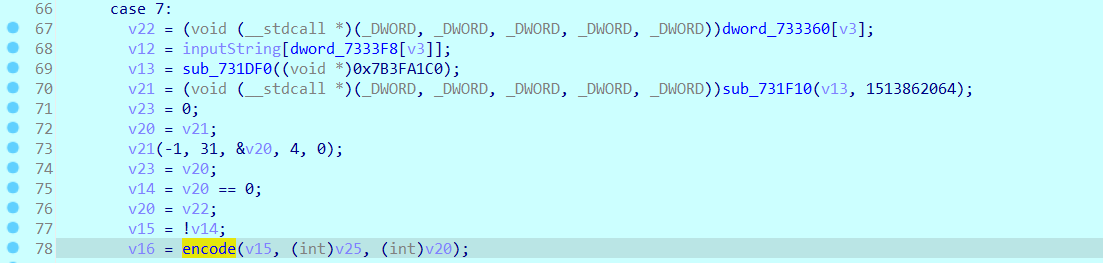
Ở đây **byte\_7355B8** =0 nên ko thực hiện dòng 13->17

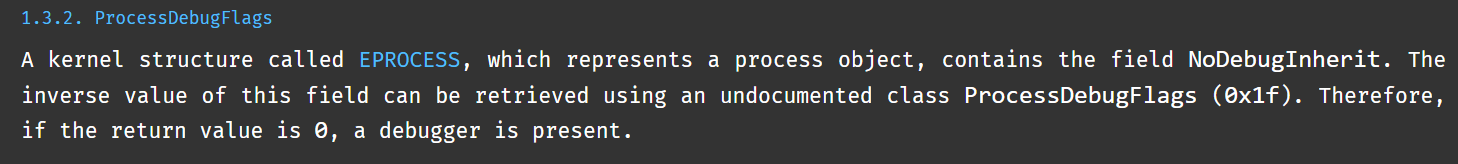
Vì vậy nếu có debug thì v7==v8 hay tham số đầu thứ nhất có giá trị là 0. Nếu có debug trả về tham số **thứ nhất có giá trị là 0**

* **Case 7 NtQueryInformationProcess**









Tìm hiểu lại

Có thể kiểm tra số cổng của trình gỡ lỗi giá trị khác 0 cho biết chương trình chạy dưới trình gỡ rối. Nếu có debug thì v21 !=0 => v20!=0 => v14 !=0 => v15=0. Vậy nếu có debug tham số **thứ nhất có giá trị là 0**

Sử dụng **NtQueryInformationProcess**  để phát hiện debug.

Với trường **ProcessInformation = 0x1F** 1 cấu trúc **EPROCESS** được gọi đến. Trong cấu trúc này có trường **NoDebugInherit** giá trị của trường này được trả về tại **ProcessInformation** nếu giá trị trả về tại đây khác 0 báo hiệu chương trình đang chạy dưới debugger

Cấu trúc **EPROCESS** :https://www.nirsoft.net/kernel\_struct/vista/EPROCESS.html

Cách 2 :

Cách này đơn giản hơn là lúc debug, đặt điểm breakpoint tại đoạn encode xem ứng với mỗi case thì tham số đầu vào là bao nhiêu (ko cần hiểu các kĩ thuật anti debug sử dụng) làm cách này lần lượt có được

Case 6 = 0

Case 1 = 1

Case 7 = 0

Case 1 =0

Case 3 =0

Case 2 =0

Case 4 =1

Case 3 =0

Case 6 =0

Case 3 =0

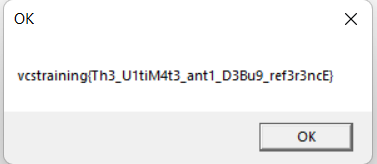
Case 7=0

Tới đây viết chương trình để tìm ra input phù hợp

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



Hình : key



Hình : flag

Tài liệu tham khảo

* <https://ctf-wiki.mahaloz.re/reverse/windows/anti-debug/int-3/>
* https://anti-debug.checkpoint.com/techniques/debug-flags.html