**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

MÔN HỌC: KIẾN TRÚC MÁY TÍNH VÀ HỢP NGỮ

LỚP CỬ NHÂN TÀI NĂNG 2017

BÁO CÁO ĐỒ ÁN 3

**CRACK PHẦN MỀM**

***Nhóm thực hiện:***

1712152 | Nguyễn Thị Mai Thanh

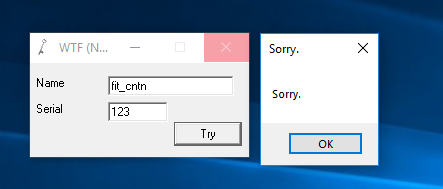
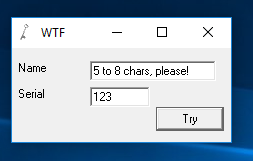
1712228 | Phạm Việt Nga

1712807 | Nguyễn Thị Minh Thùy

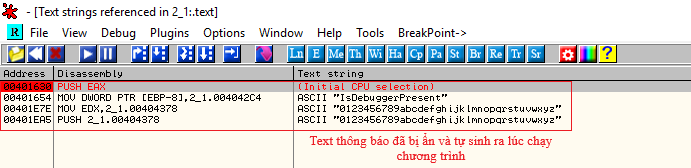
**THÔNG TIN NHÓM & PHÂN CÔNG**

|  |  |  |  |
| --- | --- | --- | --- |
| Thành viên | MSSV | Phân công | Mức độ hoàn thành |
| Nguyễn Thị Mai Thanh | 1712152 | 2\_1 | 100% |
| Phạm Việt Nga | 1712228 | 2\_3 | 100% |
| Nguyễn Thị Minh Thùy | 1712807 | 2\_2 | 100% |

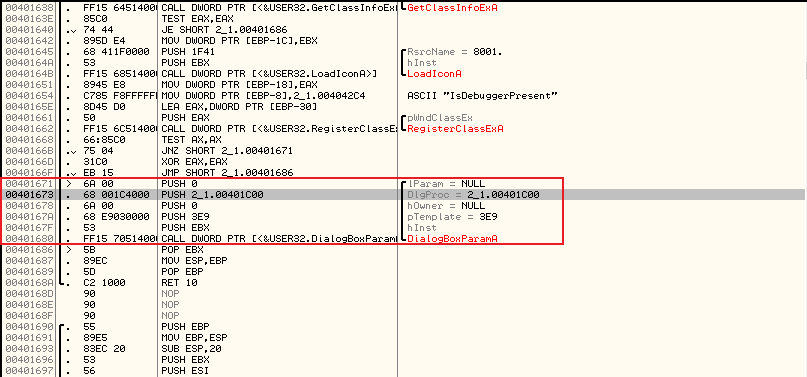
**2.1**

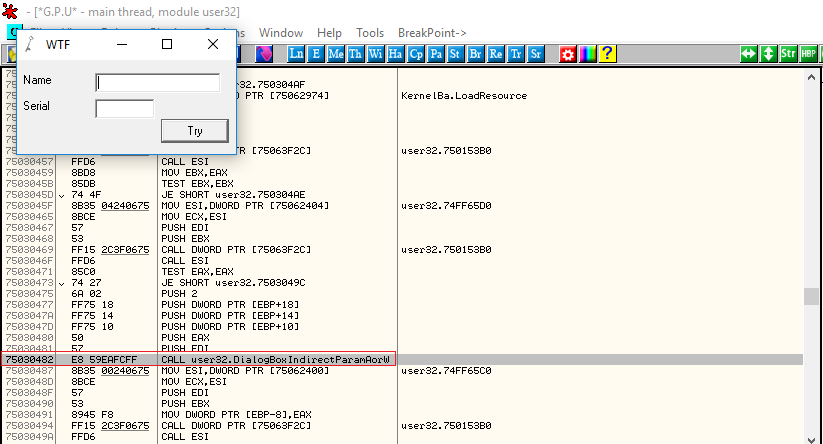
1. **Chạy thử chương trình:**
2. **Quá trình debug:**

* Chọn Search for => All referenced text strings để xem các chuỗi kí tự trong chương trình.

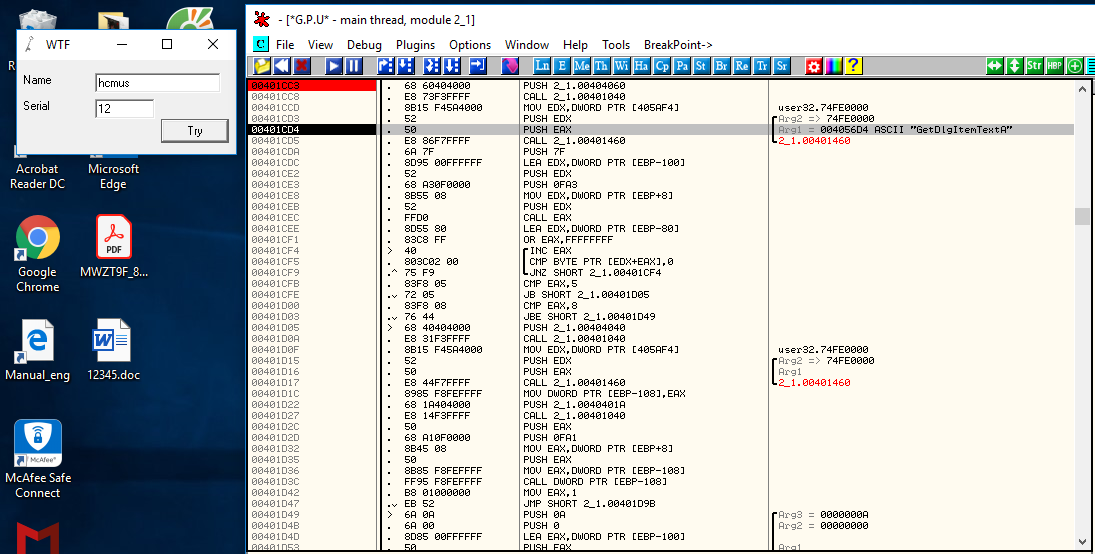


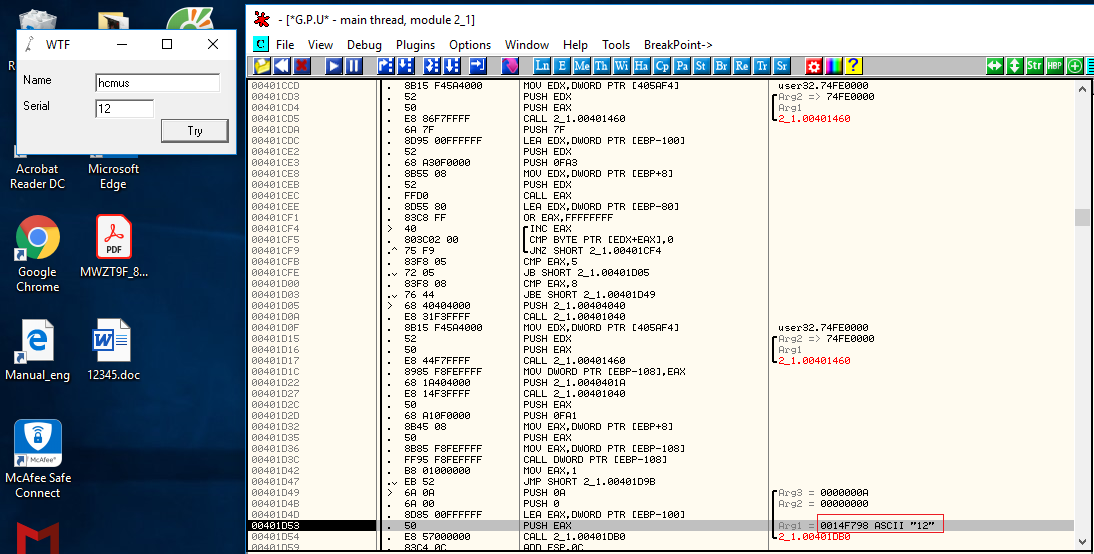
* Không tìm thấy text “Sorry” => phải tìm đoạn sinh ra DialogBox nhập dữ liệu
* Khi debug đến dòng 00401680 thì DialogBox được tạo

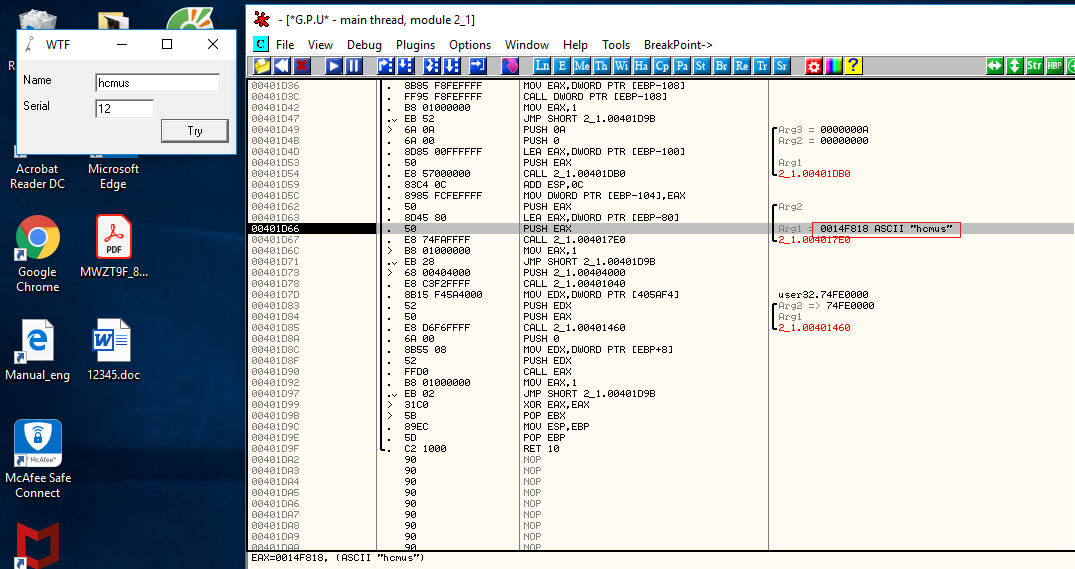




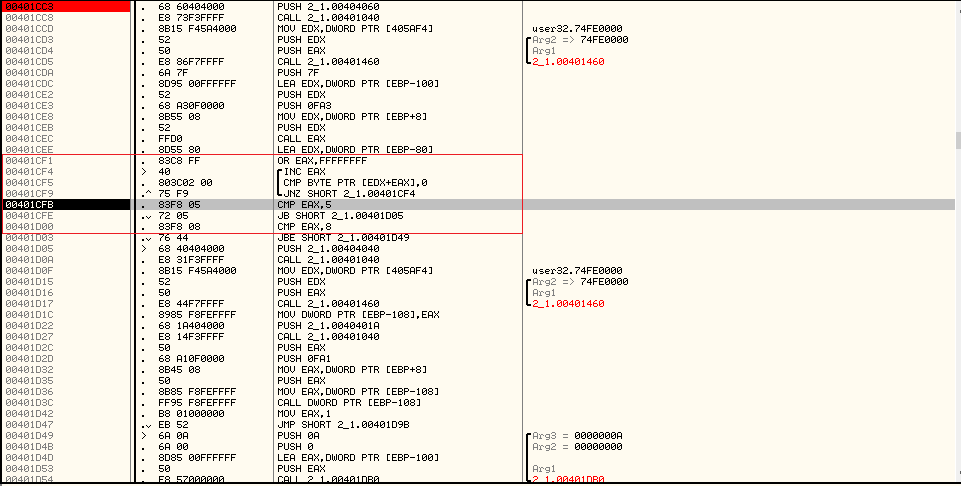
* Tiếp tục debug thì tìm thấy đoạn code lấy chuỗi dữ liệu nhập vào từ chương trình (Name: hcmus, Serial: 12)



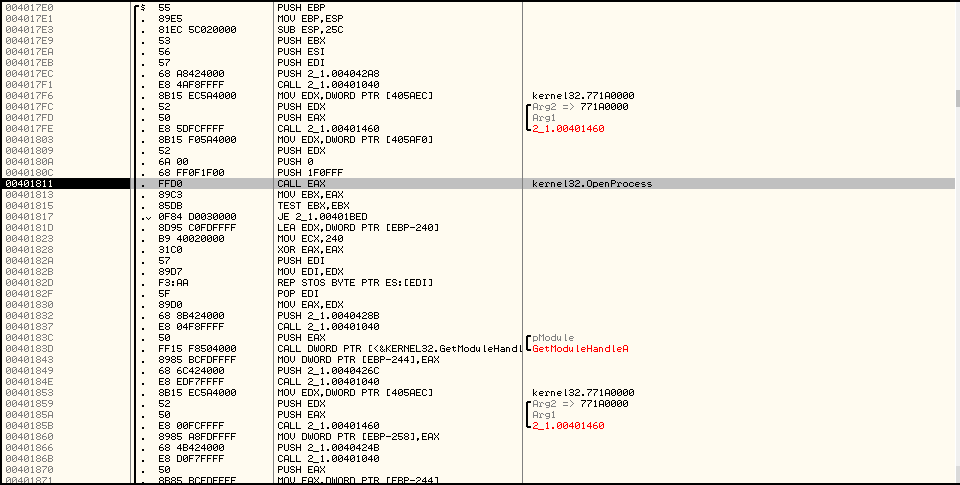




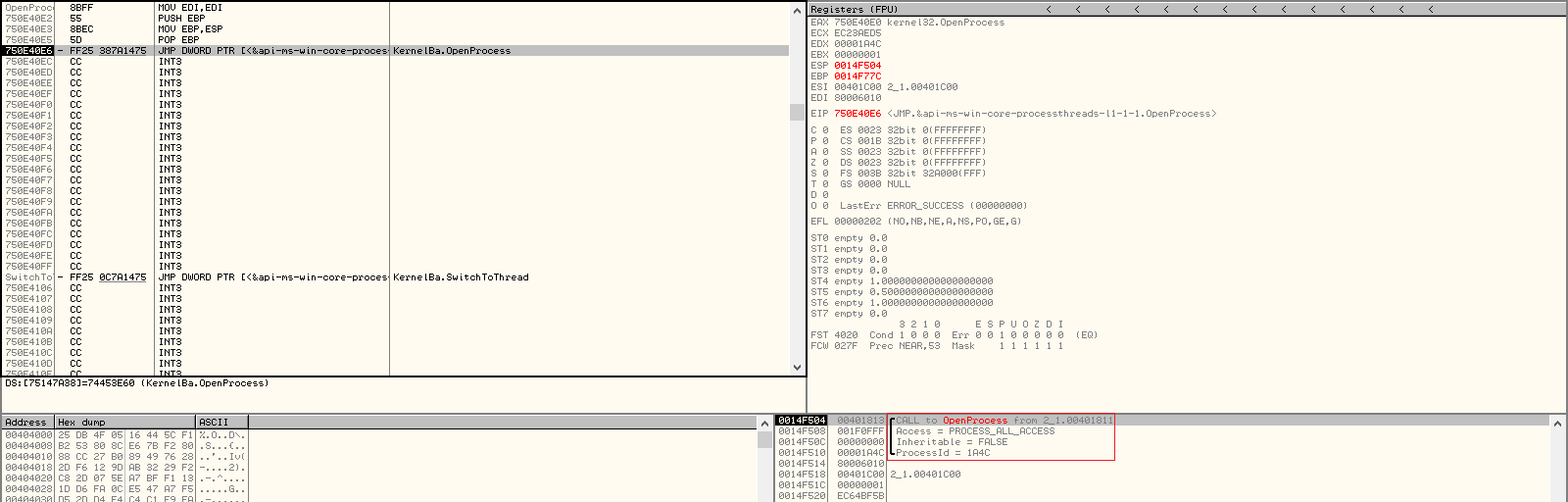
* Phát hiện thấy đoạn code kiểm tra độ dài của chuỗi Name nhập vào (phải có độ dài từ 5 đến 8 kí tự)

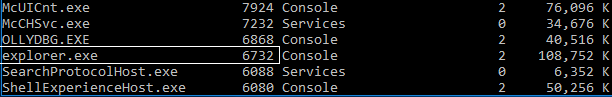


* Sau đó, tìm thấy được hàm kernel32.OpenProcess

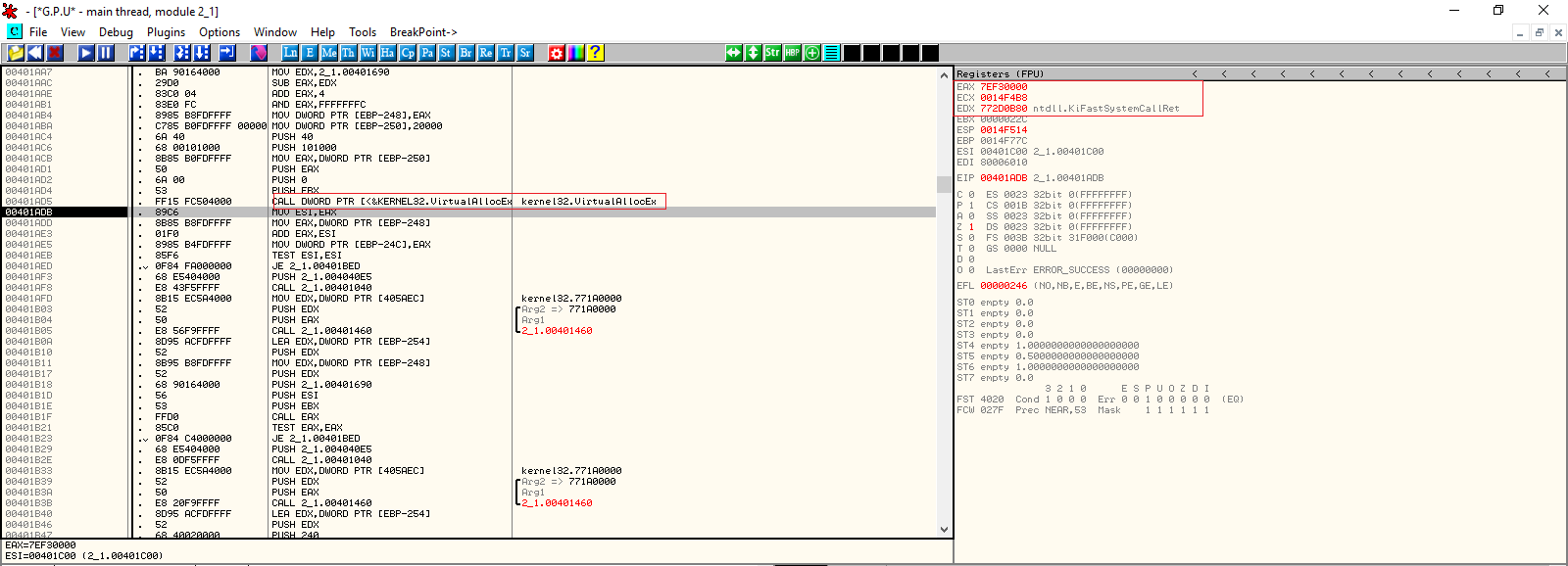


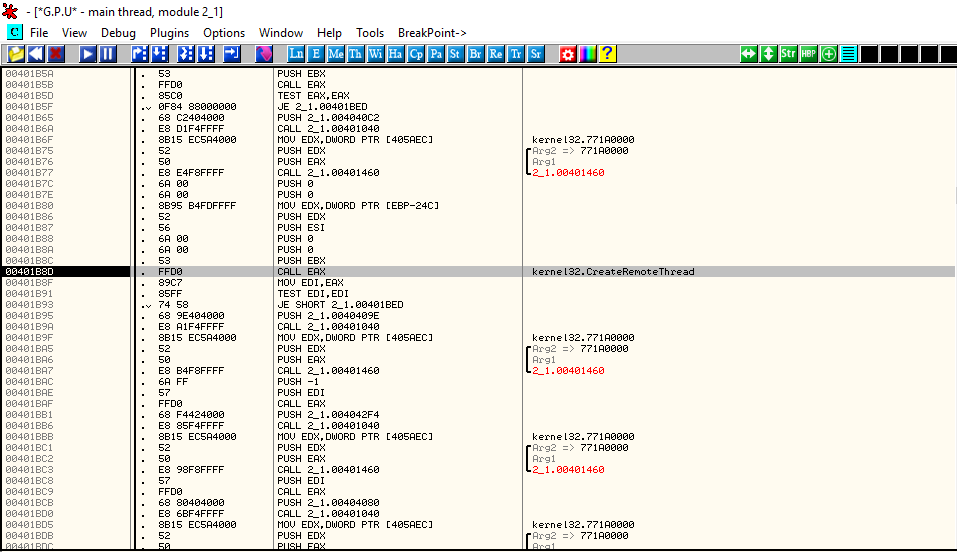
* Tìm thấy ProcessId được đẩy vào bộ nhớ là 1A4C16 = 673210, dựa vào tasklist thì xác định được đó là ProcessId của explorer.exe



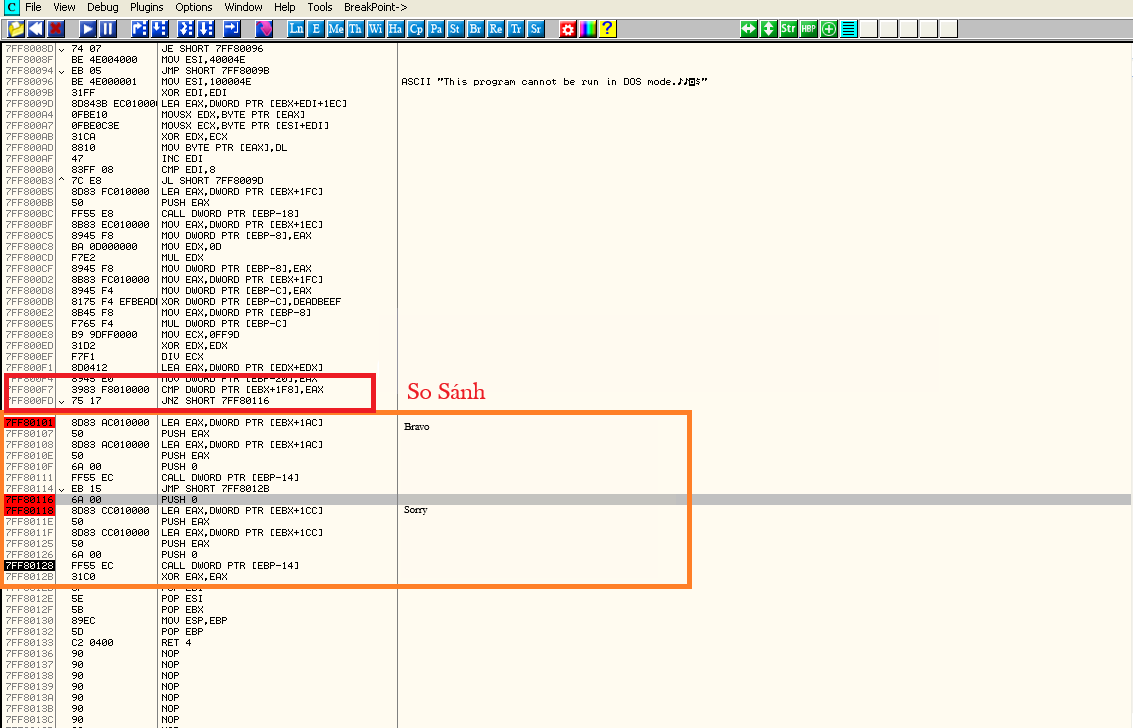


* Tiếp theo là gọi hàm kernel32.VirtualAllocEx và kernel32.CreateRemoteThread để nhúng explorer.exe vào đoạn code

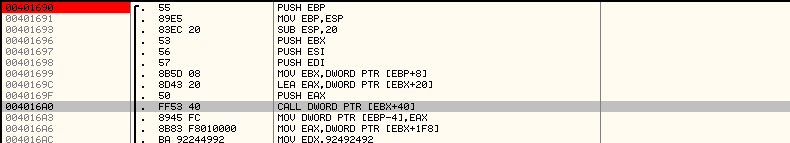




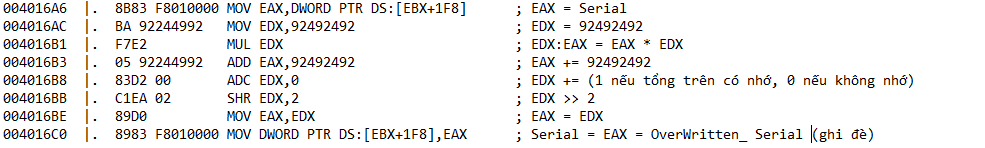
* Mở file Olly mới và Attach explorer.exe vào, ta dựa vào địa chỉ được nhúng vào code (7EF3000) để tìm ra phần được đưa vào



* Trong quá trình debug thì tìm được code xử lý chính của dữ liệu đầu vào

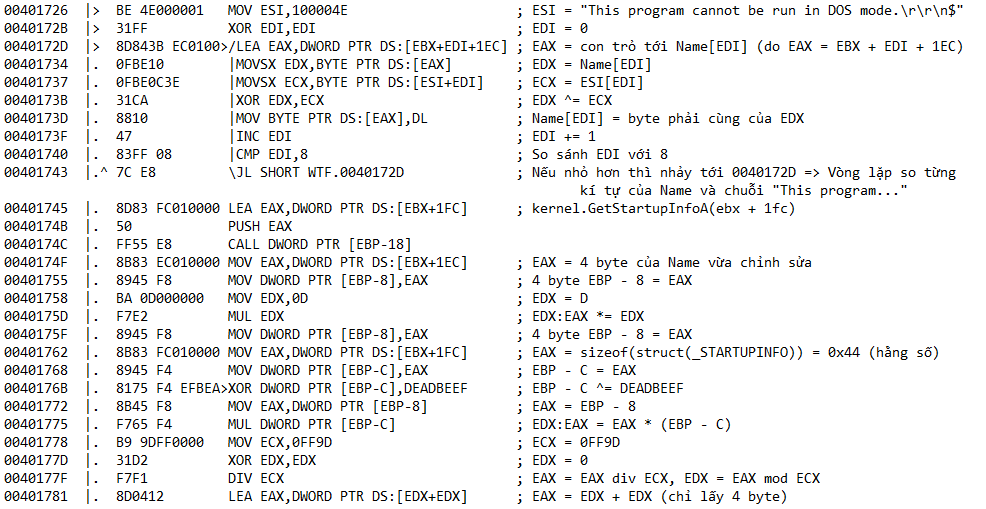


* Xử lý phần Serial nhập vào:





* Xử lý phần Name nhập vào:



* 100004E là địa chỉ của chuỗi ASCII "This program cannot be run in DOS mode.\r\r\n$"
* Dù lúc đầu xử lý toàn bộ 8 kí tự trong vòng lặp (từ 0040172D đến 00401743), nhưng kết quả xử lý lúc sau chỉ còn lại 4 kí tự đầu (0040174F) => Serial chỉ phụ thuộc vào 4 kí tự đầu của Name
* So sánh 2 chuỗi vừa xử lý với nhau:



* Như vậy ta có thể suy ngược Serial từ chuỗi Name mà người dùng nhập vào

1. **Xử lý Name để tìm ra Serial phù hợp:**
   1. Kiểm tra tính hợp lệ của chuỗi Name nhập vào (phải từ 5 – 8 kí tự)
   2. Xử lý phần Name (tương tự chương trình)

* XOR từng kí tự của chuỗi Name với chuỗi “This pro” (do Name chỉ có từ 5 – 8 kí tự)
* Trích lấy 4 kí tự đầu của kết quả vừa nhận được
* Sau đó nhân chuỗi đó với D
* Tiếp theo đem nhân kết quả với (0x44 xor DEADBEEF)
* Sau đó đem chia kết quả cho 0FF9D
* Kết quả cuối cùng là thương của phép chia trên \* 2
  1. Từ Kết quả tìm được suy ngược ra Serial (làm ngược so với chương trình)
* Gọi R là kết quả vừa tìm được
* R >>= 1 (chia 2)
* R ^= 1
* R -= D
* R <<= 2 (nhân 4)
* R = R \* 0x100000000
* R div 92492492
* Ta có:

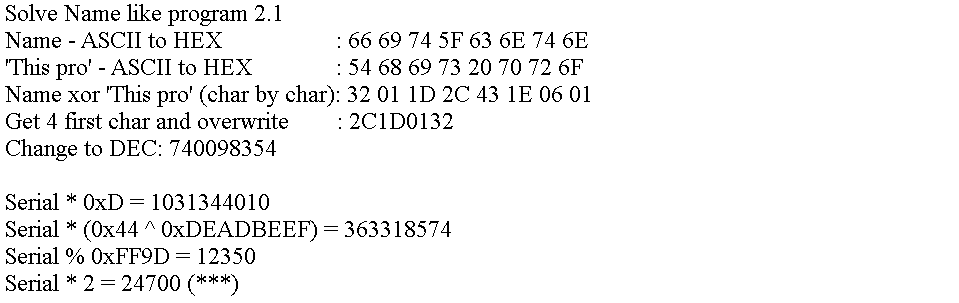
Gọi a = 0x100000000 và c = 92492492

Theo chương trình 2.1 thì: R \* a + b = dc + c với b có dạng XXXXXXXX (do chương trình lấy dự liệu ở EDX bỏ qua EAX)

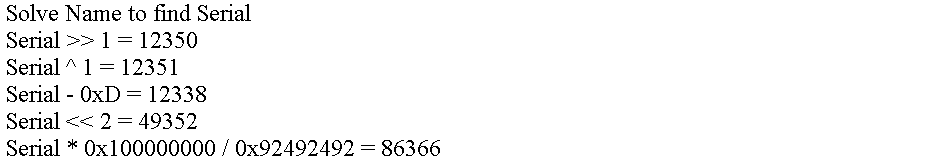
Như vậy, kết quả cần tìm sẽ là: d = (R \* a + (b – c)) div c = (R \* a) div c (do (b – c) < c nên không thể làm thay đổi giá trị của d => Kết quả luôn đúng)

1. **Ví dụ:**

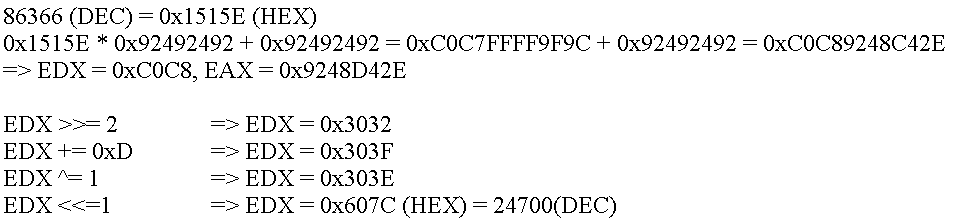
* Nhập vào Name là fit\_cntn
* Xử lý phần Name nhập vào (như chương trình 2.1)



* Xử lý phần Name để tìm Serial (làm ngược lại chương trình 2.1)

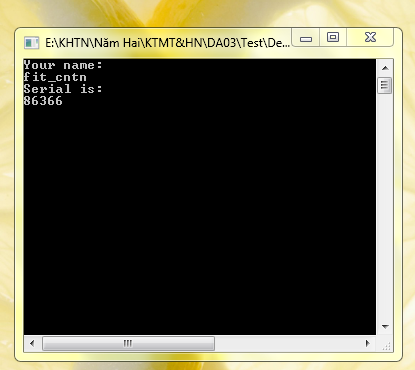


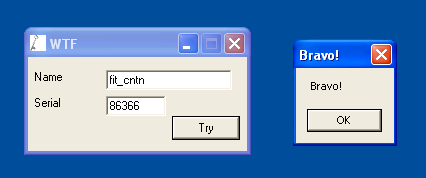
* Thử lại Serial vừa tìm được theo chương trình 2.1



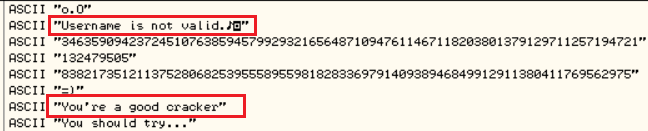
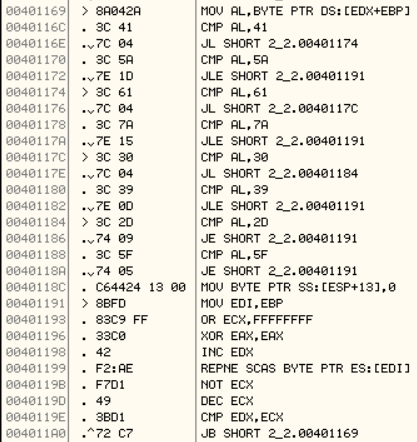
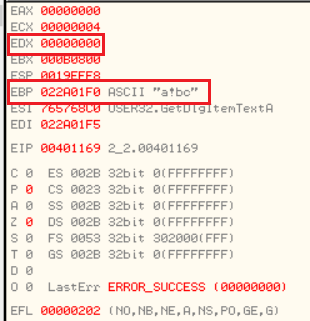
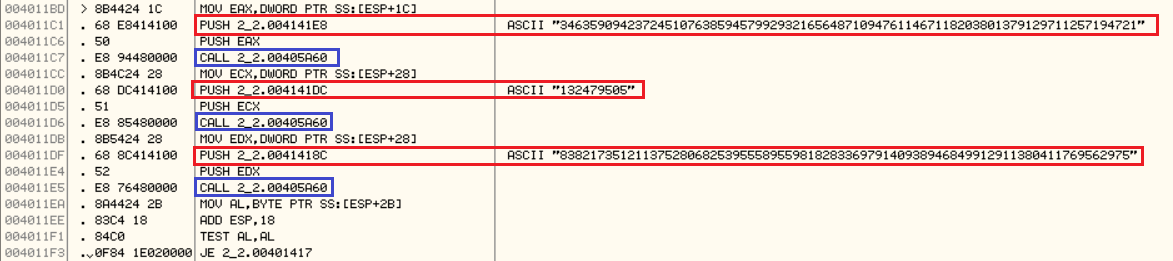
Như vậy, kết quả tìm được bằng với phần kết quả xử lý Name

* Chạy thử chương trình





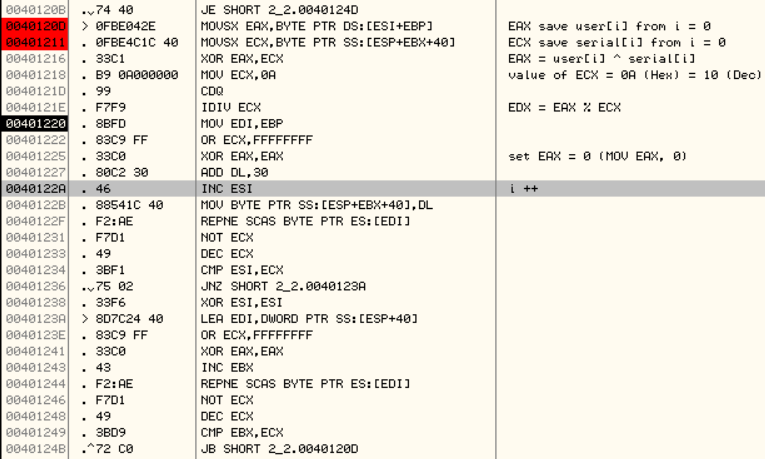
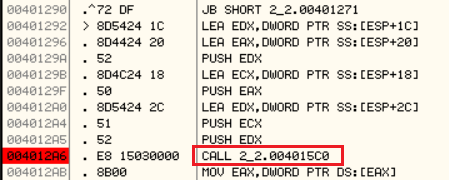
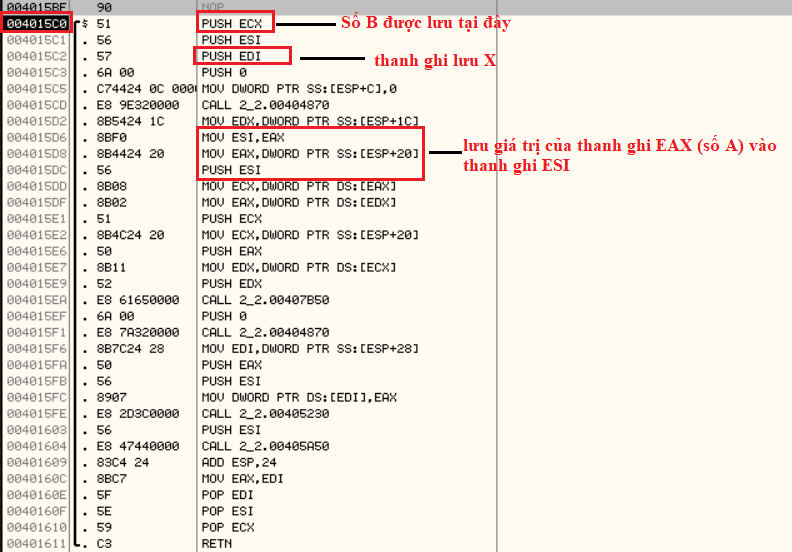
**2.2**

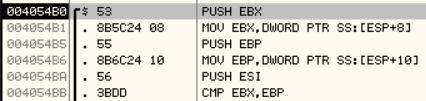
* Mở file 2.2.exe bằng phần mềm OllyDbg, tại cửa sổ CPU của phần mềm, nhấn chuột phải, chọn **Search for** 🡪 **All referenced text strings**.
* Từ các string tìm kiếm được, có thể nhận thấy Goodboy = **“You’re a good cracker”** và Badboy = **“Username is not valid”**.
* Xử lý Badboy
* Nháy đúp chuột vào dòng **“Username is not valid”** để hiện tập dòng code dẫn đến xảy ra badboy. Có thể thấy, tại dòng 004011B1 thông báo ra text của badboy, như vậy, từ dòng lệnh đó trở về trước sẽ là lệnh dẫn đến username không hợp lệ.
* Với username = “Admin” mặc định sẵn, nhập serial key bất kỳ hoặc để trống thì chương trình không hiển thị bất kỳ thông báo nào. Vậy, đó là một username hợp lệ và serial key sai (do trong các string kiếm được trong chương trình, không có string nào đề cập đến vấn đề lỗi serial).
* Thử với username = “a!bc”, đặt BreakPoint tại dòng code chứa badboy và chạy chương trình.
* Đoạn code lặp dẫn đến message badboy.
* Phân tích đoạn code:
* Từ hình trên (kết quả trên thanh ghi) cho biết thanh ghi EBP lưu username nhập vào, EDX lưu biến chạy (i) và bắt đầu i = 0.
* Dòng code 00401169 **MOV AL, BYTE PTR DS: [EDX + EBP]**: AL sẽ lần lượt lấy từng ký tự của username trong thanh ghi EBP để so sánh với các giá trị định sẵn theo hệ hexa (‘A’ = 41, ‘a’ = 61, ‘Z’ = 5A, ‘z’ = 7A, ‘0’ = 30, ‘9’ = 39, ‘-’ = 2D, ‘\_’ = 5F). Dòng lệnh từ 0040116C đến 00401188 để kiểm tra xem ký tự có vượt khỏi phạm vi quy định hay không? ([A, Z], [a, z], [0, 9], ‘-’, ‘\_’). Sau mỗi vòng lặp, EDX sẽ tăng lên 1 (INC EDX) đến khi hết chuỗi. Khi vi phạm tính hợp lệ, messagebox sẽ được thông báo ra.
* Xử lý Goodboy
* Khi username được kiểm tra hợp lệ, chương trình tiếp tục thực hiện kiểm tra serial được nhập vào bởi người sử dụng.
* Các lệnh tiếp theo có 3 dãy số lớn được biểu diễn dưới dạng chuỗi kí tự ASCII như sau. Có thể thấy, lệnh **CALL 2\_2.00405A60** thực hiện việc lưu các chuỗi số vào bộ nhớ.
* Đặt ba chuỗi số lớn lần lượt là
* A = "34635909423724510763859457992932165648710947611467118203

801 379129711257194721" (lưu trong thanh ghi EAX)

* B = "132479505" (lưu trong thanh ghi ECX)
* C = "8382173512113752806825395558955981828336979140938946849

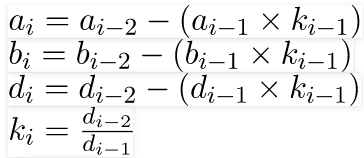
9129 11380411769562975" (lưu trong thanh ghi EDX)

* Message thông báo goodboy tại địa chỉ 004012E2. Do đó, khối lệnh tìm ra serial theo username nhập vào sẽ thực hiện từ địa chỉ 004011C1 (nối tiếp việc kiểm tra hợp lệ username).
* Tiếp theo là vòng lặp thực hiện việc tính toán giữa từng ký tự trong username và serial.
* Phân tích nhiệm vụ của các lệnh trên
* Đặt BreakPoint tại dòng 0040120D và 00401211 để xem giá trị mà hai thanh ghi EAX và ECX lưu trữ tại thời điểm này.
* Thử với username = “Admin” và serial = “123”, ta có kết quả bên thanh ghi bắt đầu như sau: thanh ghi ESI được khởi tạo bằng 0 (i = 0), EAX, ECX nhận kết quả  cho thấy đó là 2 ký tự đầu của username và serial ở dạng Hexa (00000041(hex) = ‘A’(char) và 0000001(hex) = ‘1’(char).
* Tiếp theo **00401216 XOR EAX, ECX** và **0040121E IDIV ECX**. Kết quả phép XOR được lưu vào thanh ghi EAX, phần dư của phép IDIV lưu vào thanh ghi EDX (xem giá trị trên thanh ghi khi chạy chương trình).
* Cứ tiếp tục như vậy, đoạn chương trình trên thực hiện phép ((username[i] XOR serial[i]) MOD 10).
* Kết quả nhận được (X) sẽ lưu vào bộ nhớ và thực hiện phép toán tiếp theo.
* Như hình trên, việc gọi lệnh 2\_2.004015C0 để thực hiện tính XB mod A với các giá trị X, B, A đã được lưu vào bộ nhớ (chạy xem giá trị thay đổi trên các thanh ghi).
* Kết quả nhận được nếu bằng chuỗi C thì sẽ nhận thông báo message = **“You’re a good cracker”.** Việc thực hiện so sánh được thực hiện bên dưới.



* Giải mã bài toán bằng thuật toán RSA
* Có C = XB mod A và X = CD mod A. Từ hai hệ thức trên, yêu cầu tìm ra X thỏa mãn với A, B, C đã biết (A, B, C là số nguyên lớn tương ứng với ba chuỗi ASCII trên).
* A = P \* Q (với P, Q là hai số nguyên tố lớn và P khác Q. Phân tích A ra thừa số nguyên tố, ta tìm được:
* P = 193003303516993676033890977166201071299
* Q = 179457598873041395723006544257000707979
* Khi đó, giá trị hàm số Ơle φ = (P – 1) \* (Q – 1)

φ = 346359094237245107638594579929321656483384867090770831 32044481608288055415444

* B = 132479505. Ta có gcd(φ, B) = φ\*X + B\*D = 1 (gcd: ước chung lớn nhất)
* Khởi tạo bảng với 4 cột a, b, c, k như sau
* Dòng i =1: a = 1, b = 0, d = φ, k = -
* Dòng i = 2: a = 0, b = 1, d = B, k = φ/B
* ****Kể từ i = 3:
* Điều kiện dừng: d = 1. Khi đó giá trị D được tính như sau

+ Nếu b > φ thì D = b mod φ

+ Nếu b < 0 thì D = b + φ

* Bảng kết quả như sau:

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | d | k |
| 1 | 0 | 346359094237245  107638594579929  321656483384867  090770831320444  81608288055415444 | - |
| 0 | 1 | 132479505 | 26144352987825  93636944783118  93844754691214  21241026740801  9410108818628 |
| … | … | … | … |

* Suy ra D = 1417556257111350875114490777755485627731407812521537

8969346711107195327734757.

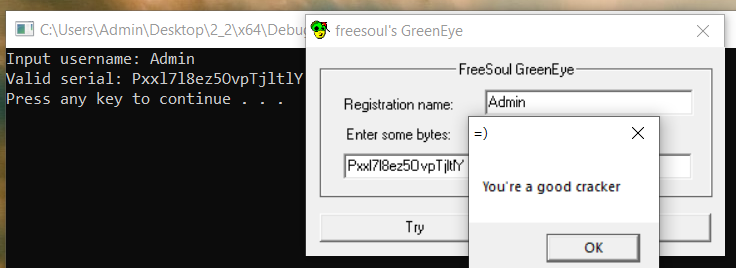
* X = CD mod A 🡪 X = 7815952891489145618.
* Tính toán giá trị trong bảng tại: <https://defuse.ca/big-number-calculator.htm>
* Phân tích thừa số nguyên tố tại: <https://www.alpertron.com.ar/ECM.HTM>
* Hướng dẫn giải bằng thuật toán RSA tại:

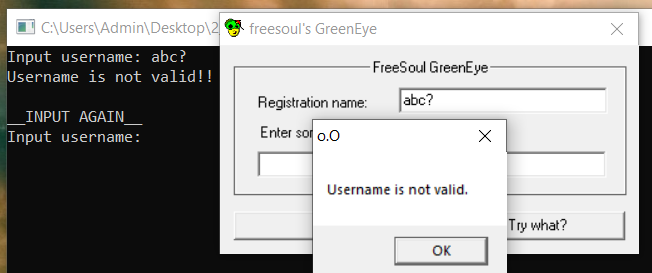
<https://www.geeksforgeeks.org/how-to-solve-rsa-algorithm-problems/?fbclid=IwAR3gdqcRAbdWymApj4XpMeJ2N3OhpjvAY490WXnVPcjDHxZRAEwtlIlCEok>

* Sau khi có kết quả X, thực hiện phép XOR giữa X và username để tìm ra serial tương ứng. Như phép tính đã phân tích phía trên **(username[i] XOR serial[i]) MOD 10 = x (x là một ký tự trong chuỗi X)**. Do đó, serial[i] sẽ được chọn ngẫu nhiên trong bảng ký tự bất kỳ. Bảng ký tự chọn như sau

Table = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz012

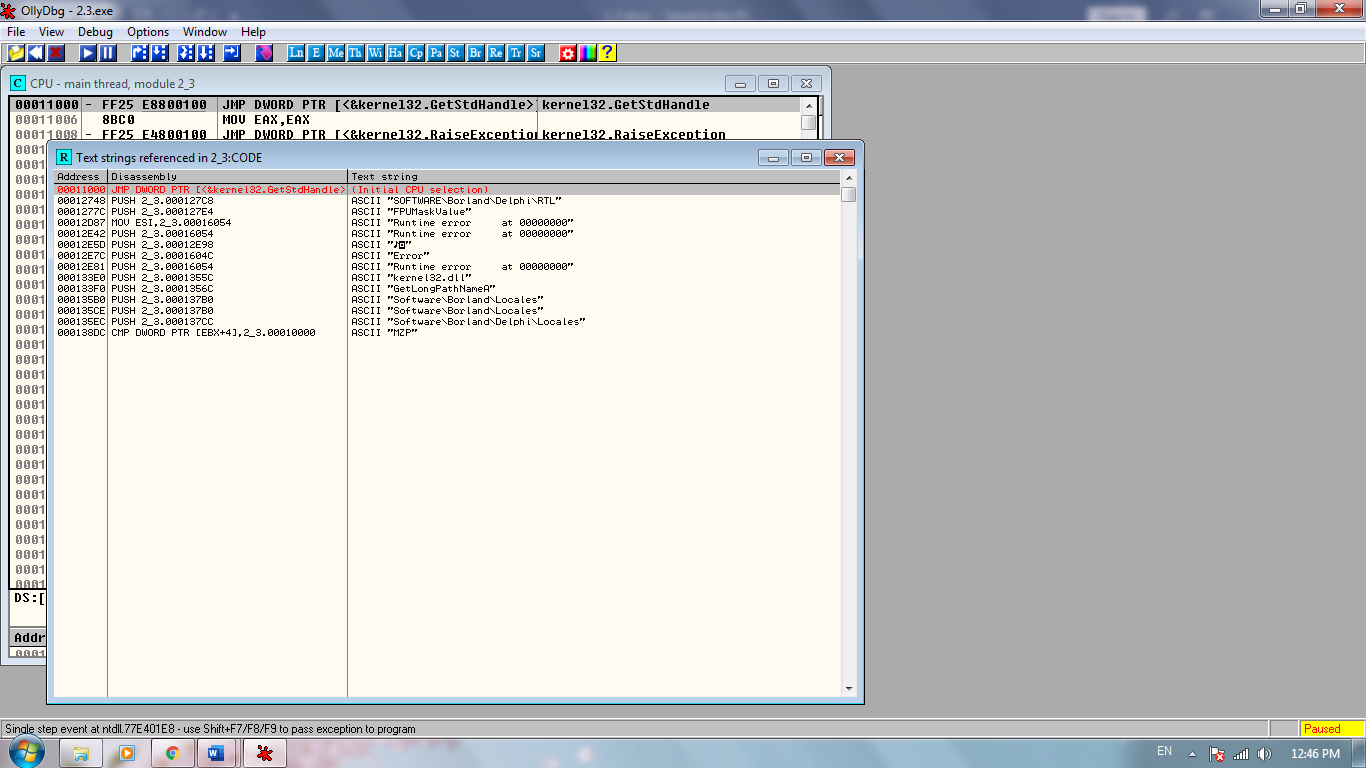
3456789-\_.,". (Bảng không cố định). Từ đó sẽ tìm ra được chuỗi serial.

* Ví dụ:

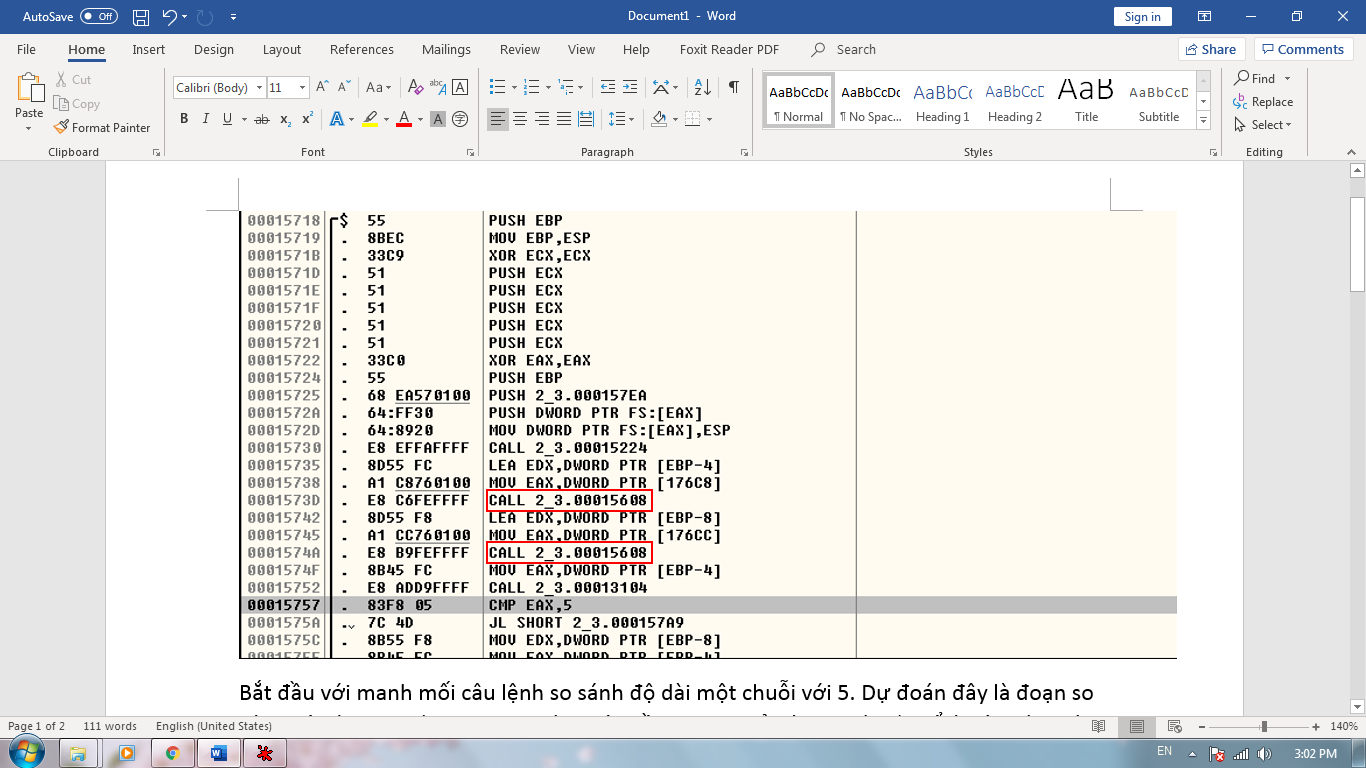


**2.3**

Thử tìm các thông báo được xuất ra trong chương trình:

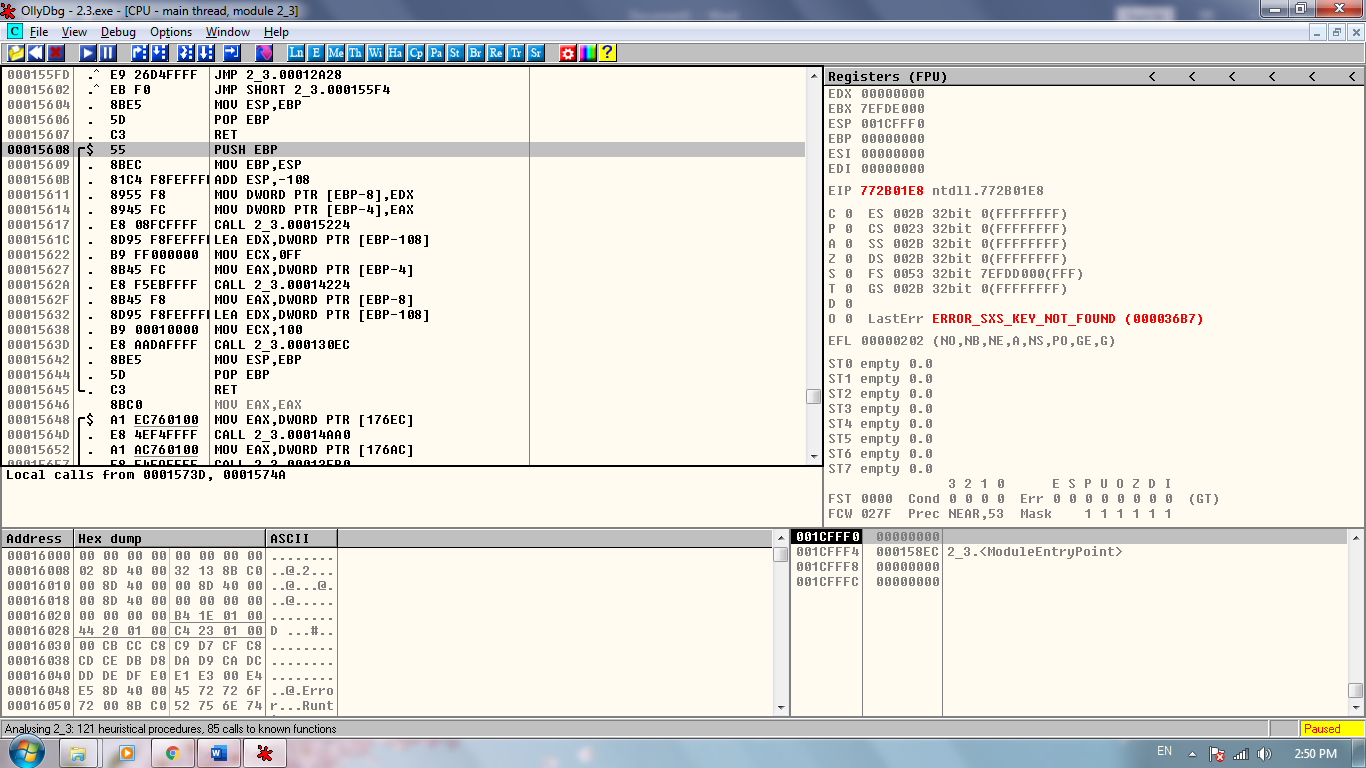


Các thông báo đúng/ sai serial đã được giấu đi, do đó phải bắt đầu từ manh mối khác.

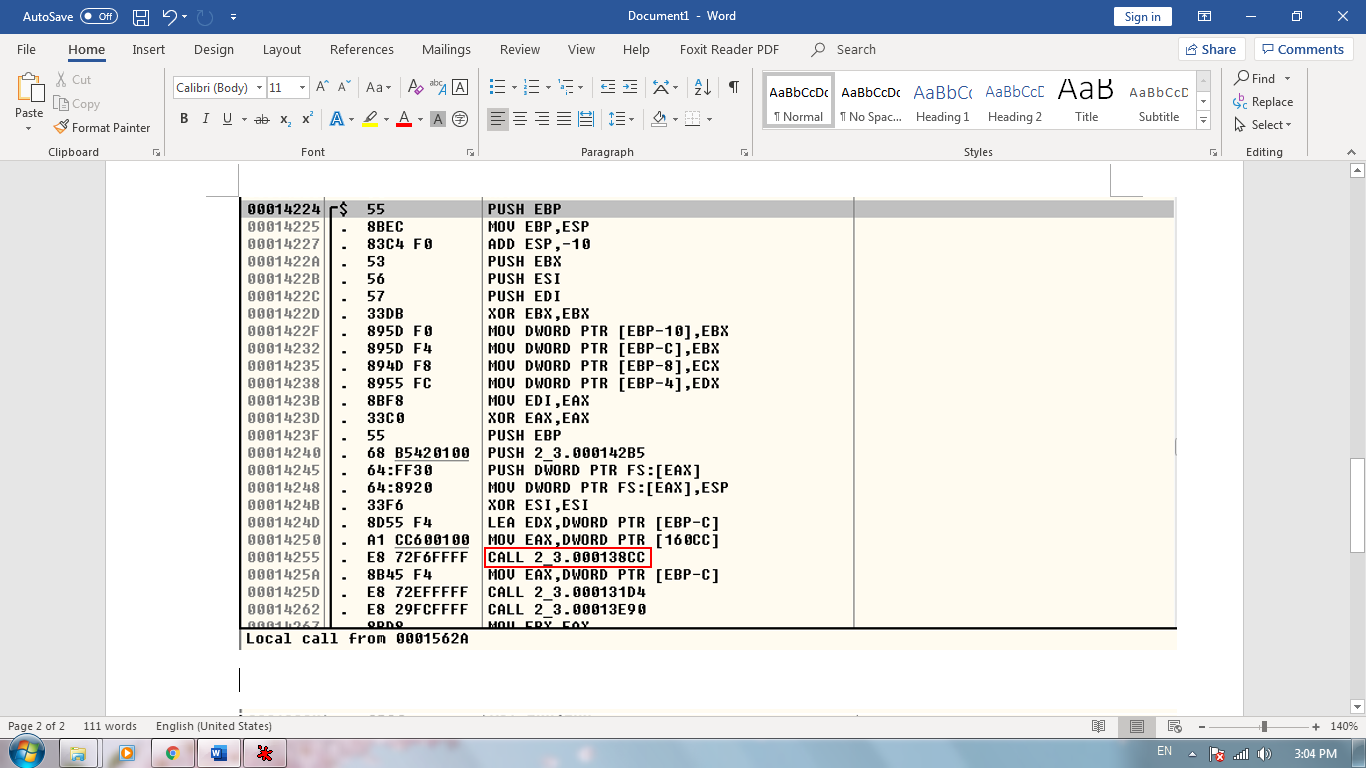


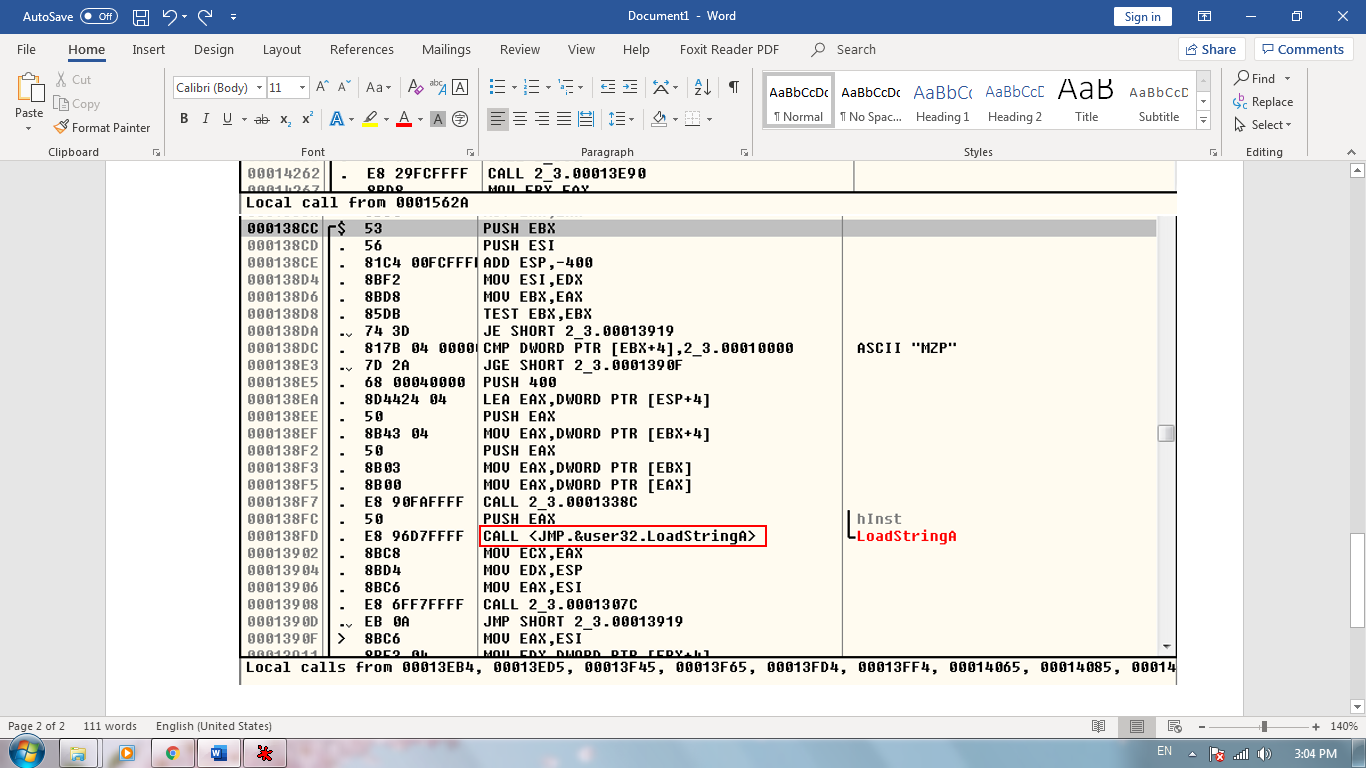
Bắt đầu với manh mối câu lệnh 00015757 so sánh một biến với 5. Dự đoán đây là đoạn so sánh độ dài name (do chương trình yêu cầu name phải từ 5-8 ký tự) và hàm 00013104 là hàm tính độ dài một chuỗi. Để ý trên hàm này có gọi hàm 00015608 hai lần 🡪 dự đoán hàm 00015608 thực hiện nhận chuỗi người dùng nhập vào, gồm hai chuỗi name và serial.

Hàm 00015608:

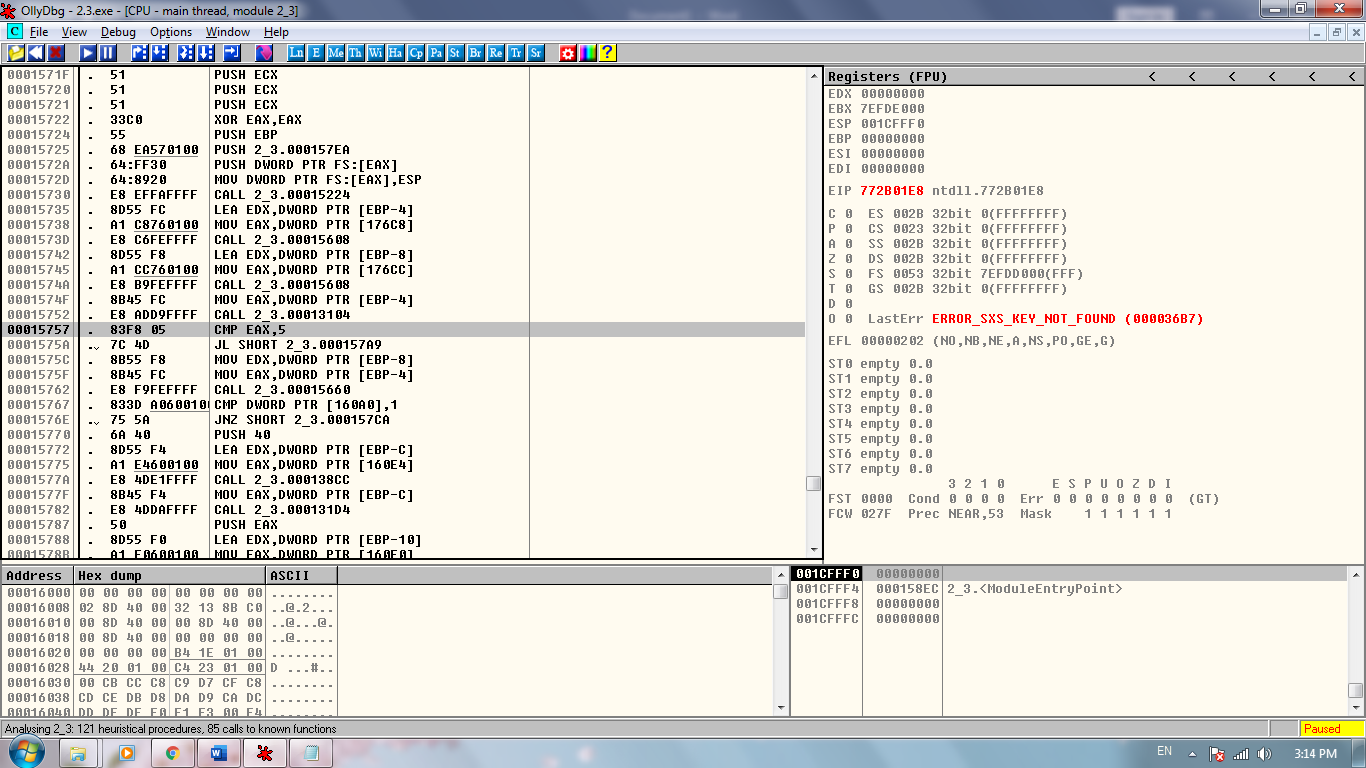


Hàm này gọi 3 hàm con là 00015224, 00014224 và 000130EC. Đi sâu vào hàm 00014224, ta thấy hàm này gọi hàm 000138CC, mà trong hàm 000138CC có lệnh thực hiện Load String 🡪 hàm 00015608 thực hiện nhận chuỗi người dùng nhập.



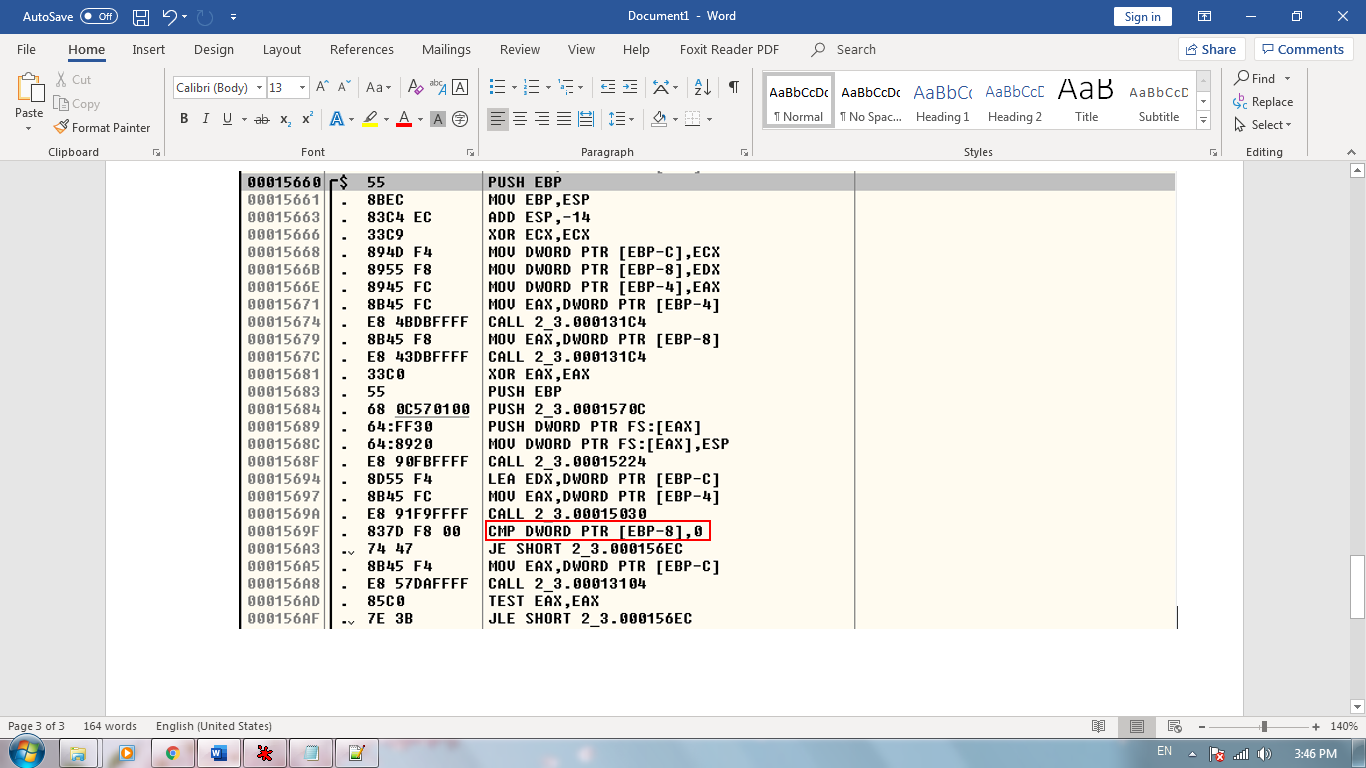


***Quay lại đoạn lệnh đọc name và serial:***



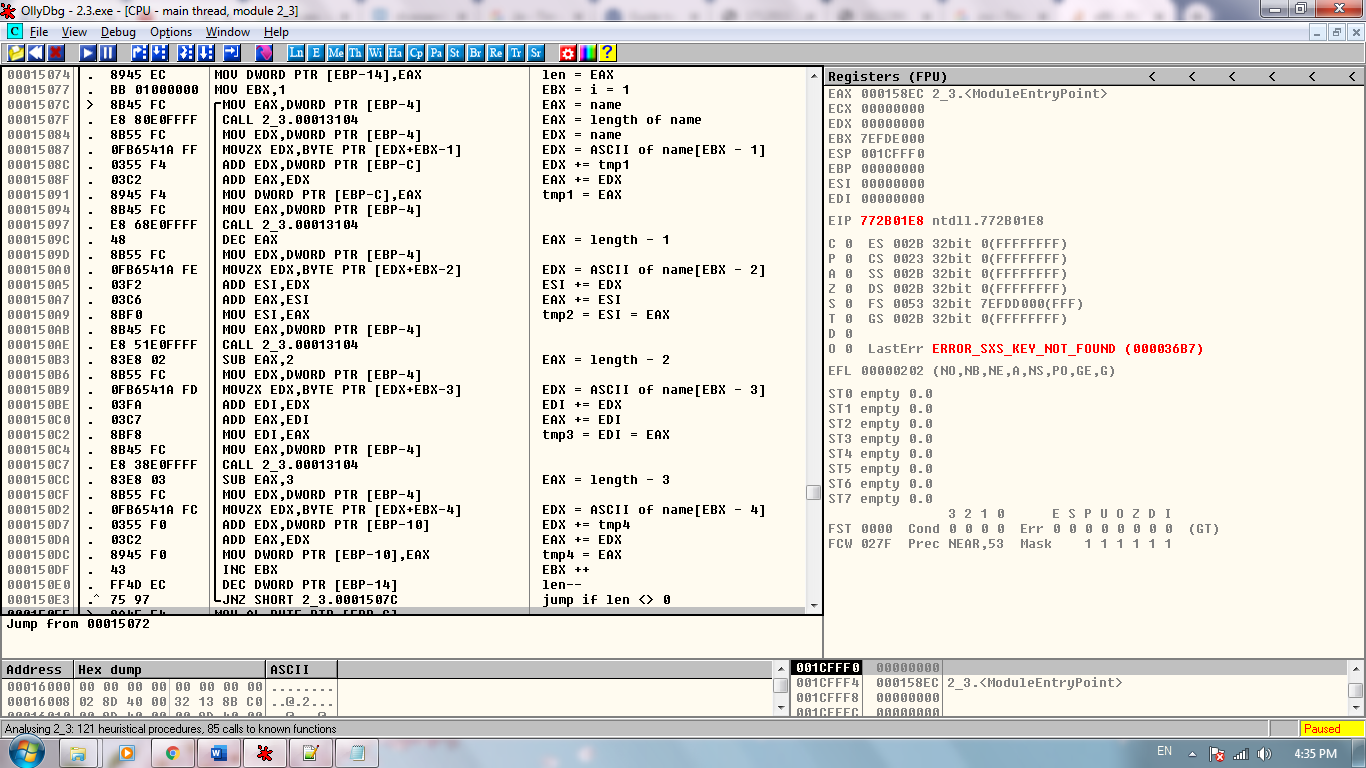
Name được lưu ở [EBP – 4] và serial do người dùng nhập được lưu ở [EBP – 8]. Nếu độ dài name >= 5: lưu name vào EAX, serial vào EDX và gọi hàm 00015660.

Hàm 00015660: trong hàm này có một câu lệnh so sánh giá trị lưu ở [EBP – 8] (chính là serial do người dùng nhập) 🡪 hàm 00015030 được gọi trước khi so sánhlà hàm phát sinh serial đúng của chương trình.



Hàm 00015030:





Mỗi biến tmp được lưu vào 1 vùng nhớ 8 bit (tương ứng với 1 thanh ghi), do đó tmp sẽ có kiểu unsigned char trong C++. Vậy các lệnh trên tương ứng với đoạn code C++ sau:

unsigned char tmp1 = 0, tmp2 = 0, tmp3 = 0, tmp4 = 0;

for (int i = 0; i < length; i++) //i = EBX - 1

{

//EAX = length;

//EDX = name;

tmp1 += length;

tmp1 += int(name[i - 0]);

tmp2 += length - 1;

if (i > 0) tmp2 += int(name[i - 1]);

tmp3 += length - 2;

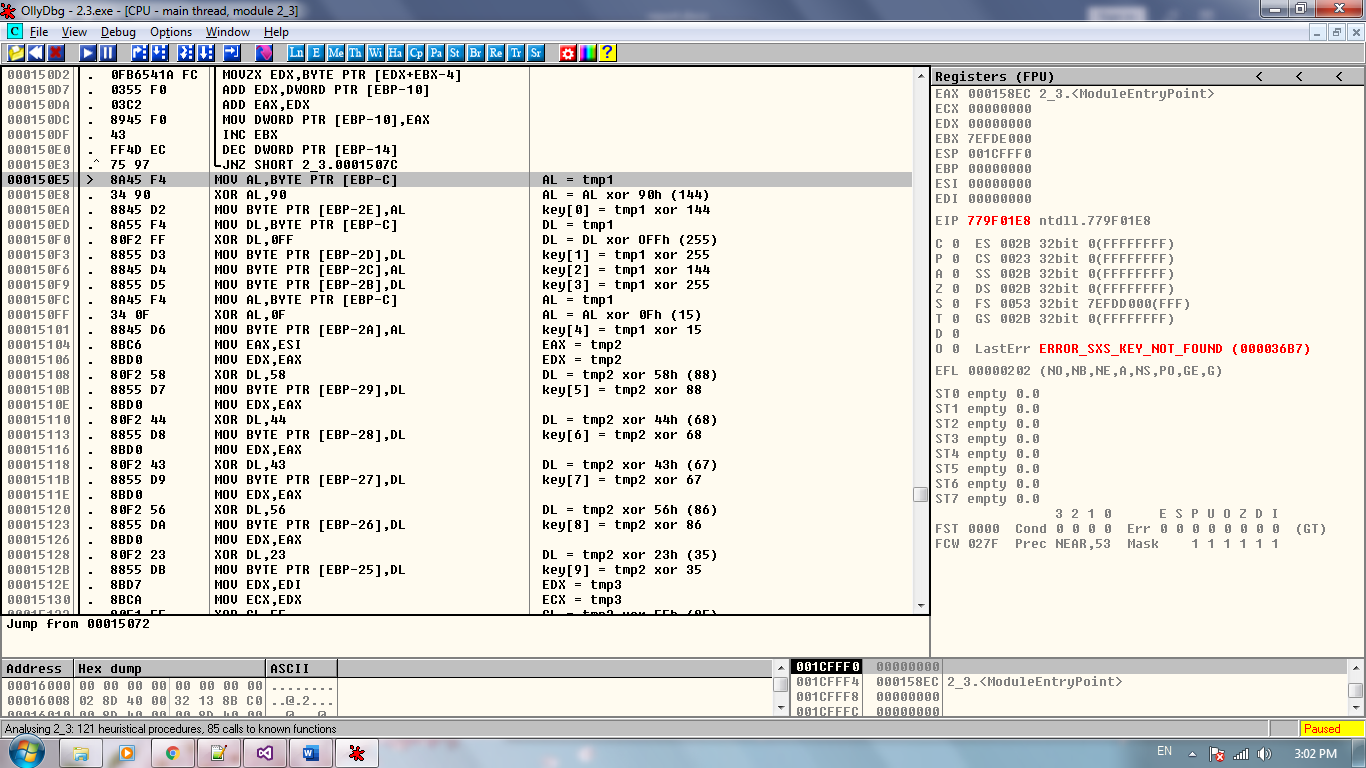
if (i > 1) tmp3 += int(name[i - 2]);

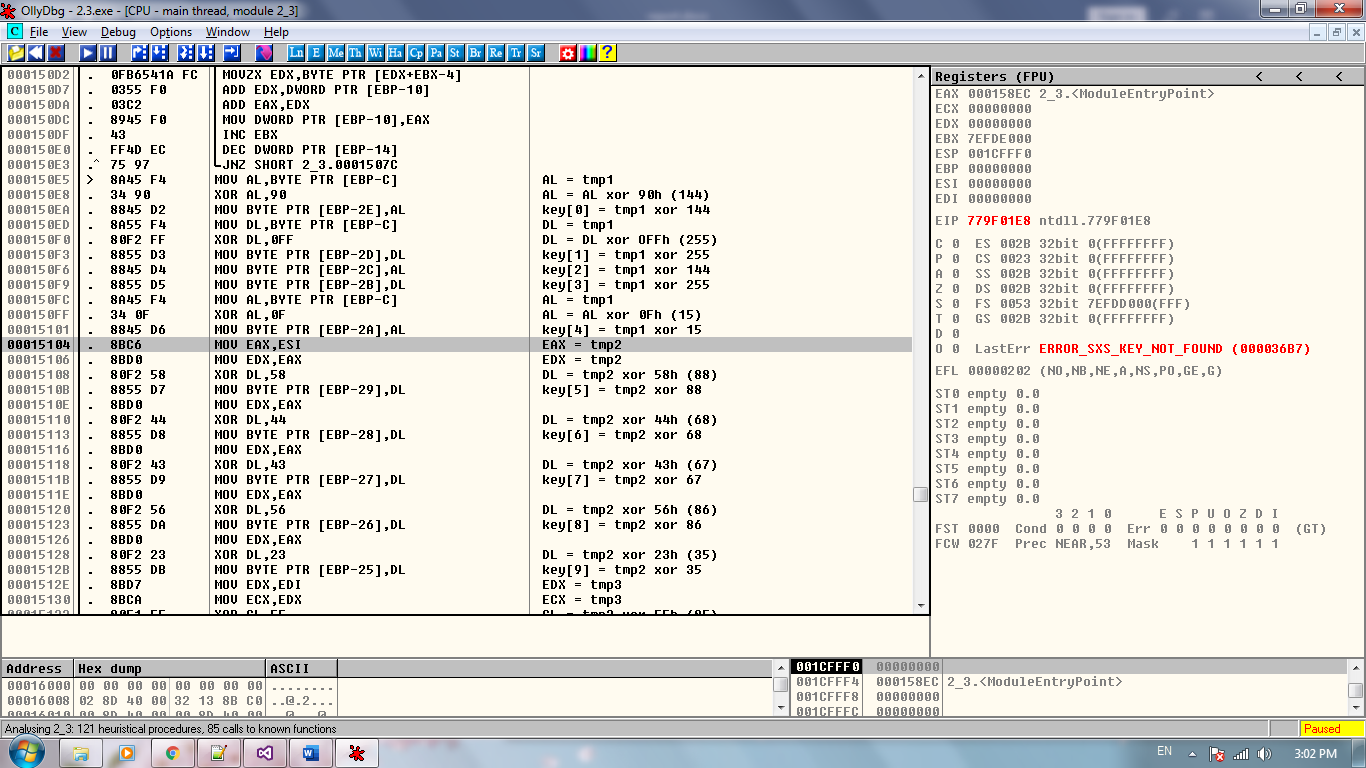
tmp4 += length - 3;

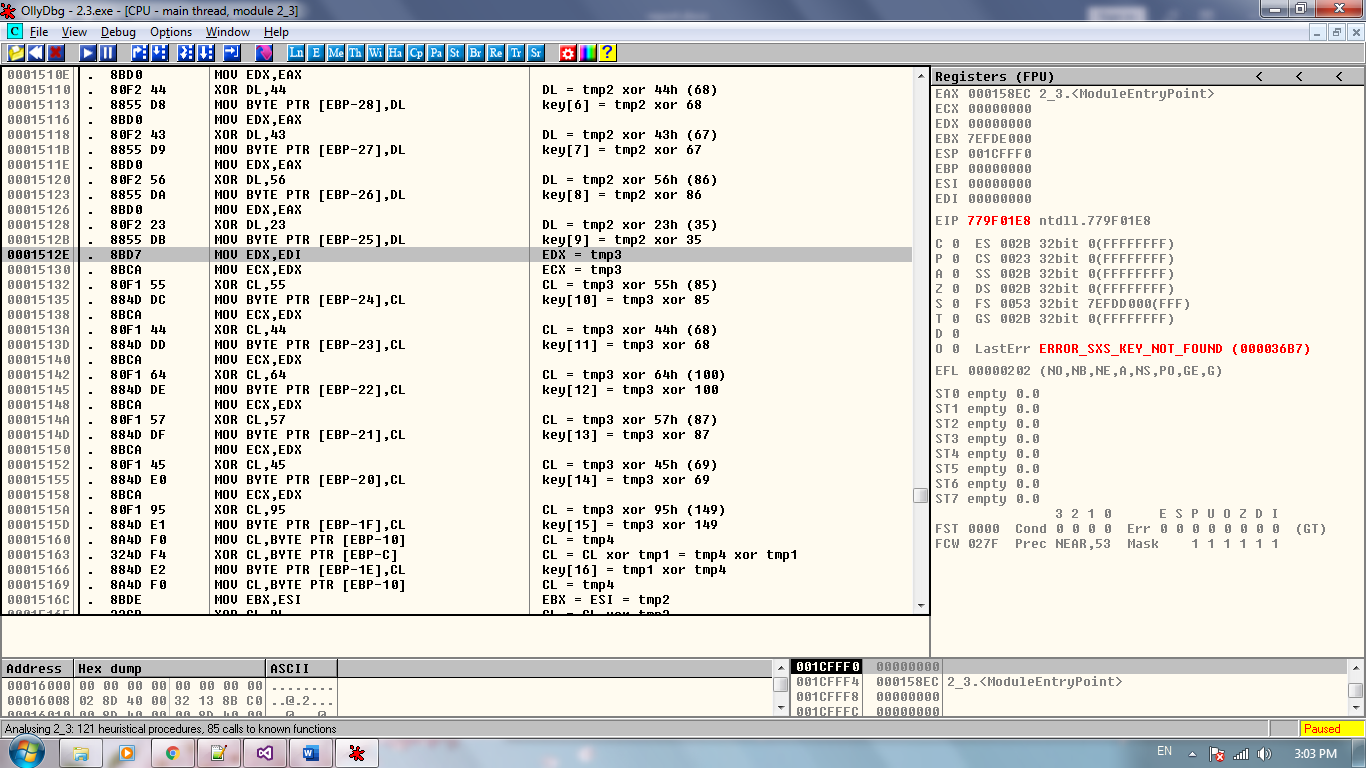
if (i > 2) tmp4 += int(name[i - 3]);

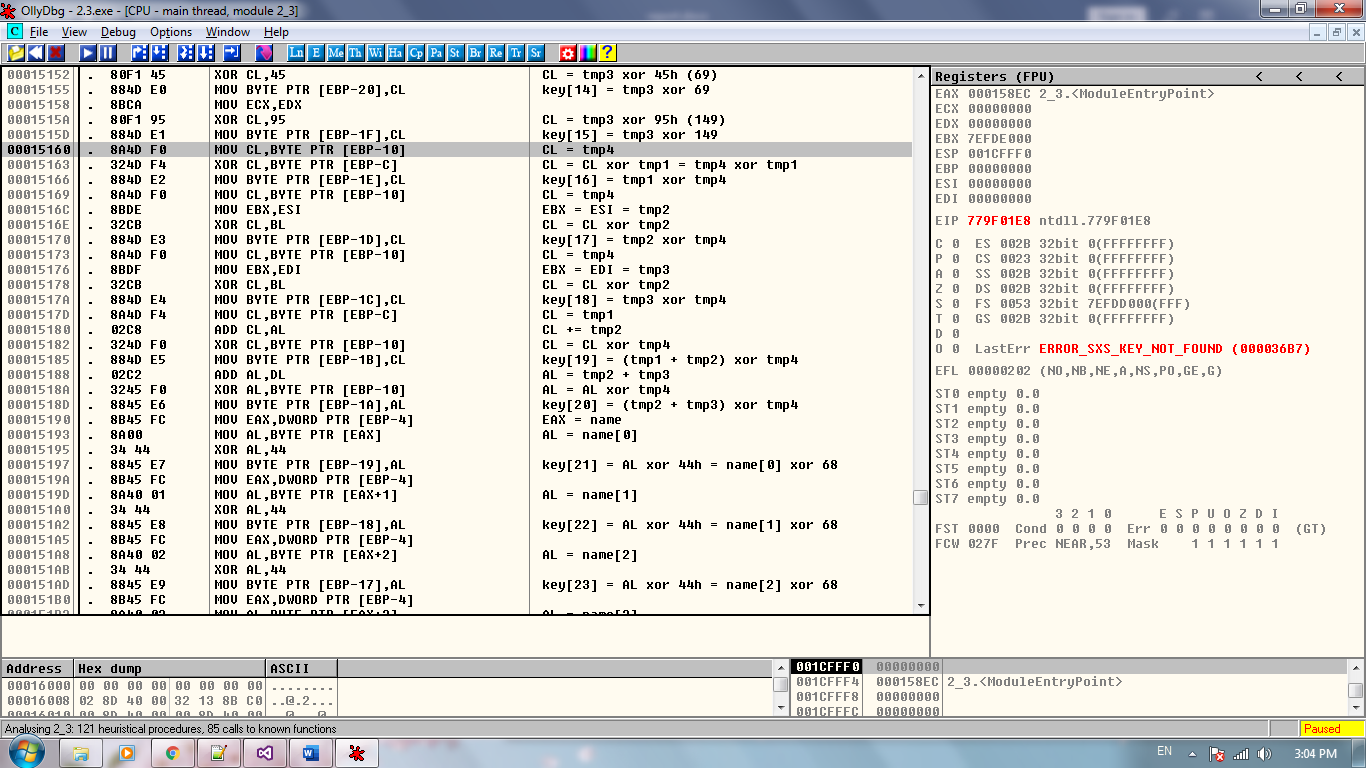
}

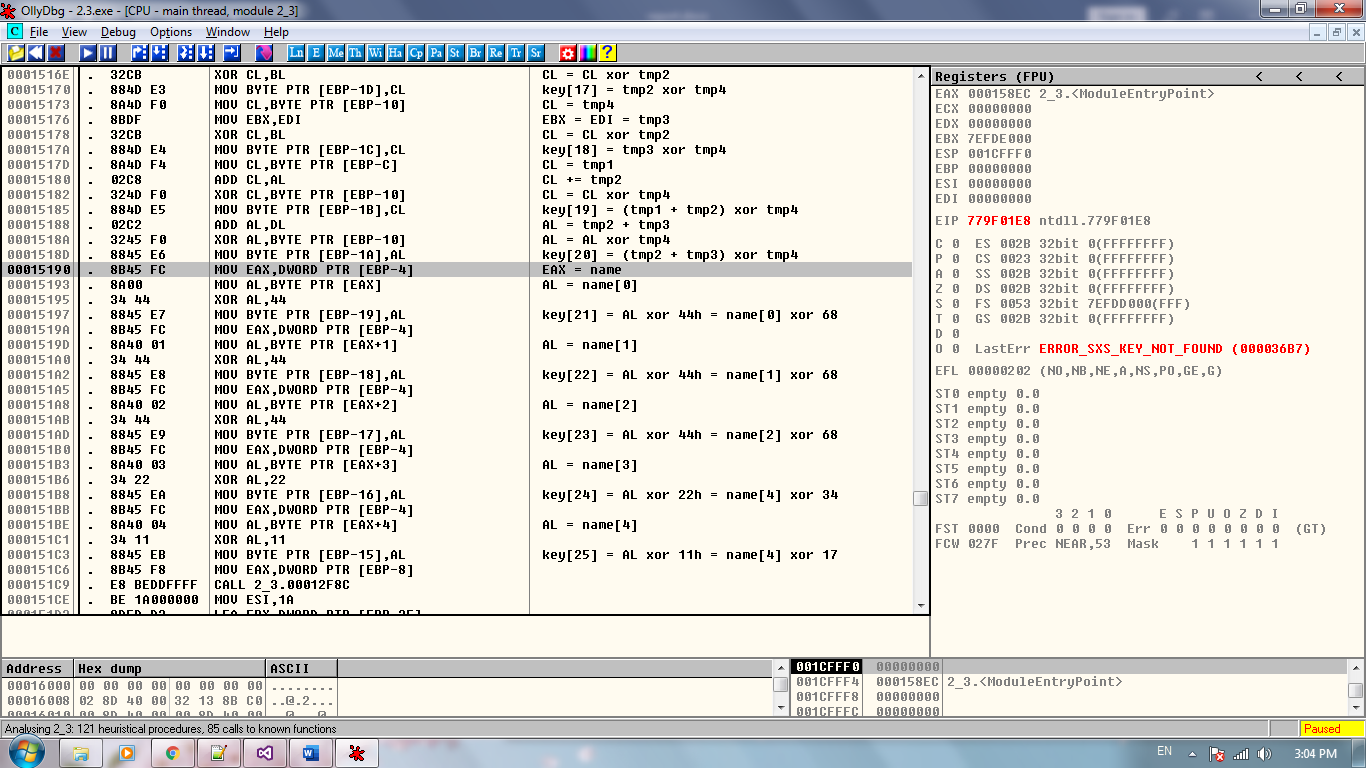
Serial được ghép lại từ 26 giá trị (được lưu vào mảng unsigned char key[26] trong C++) với mỗi giá trị được xác định như sau:



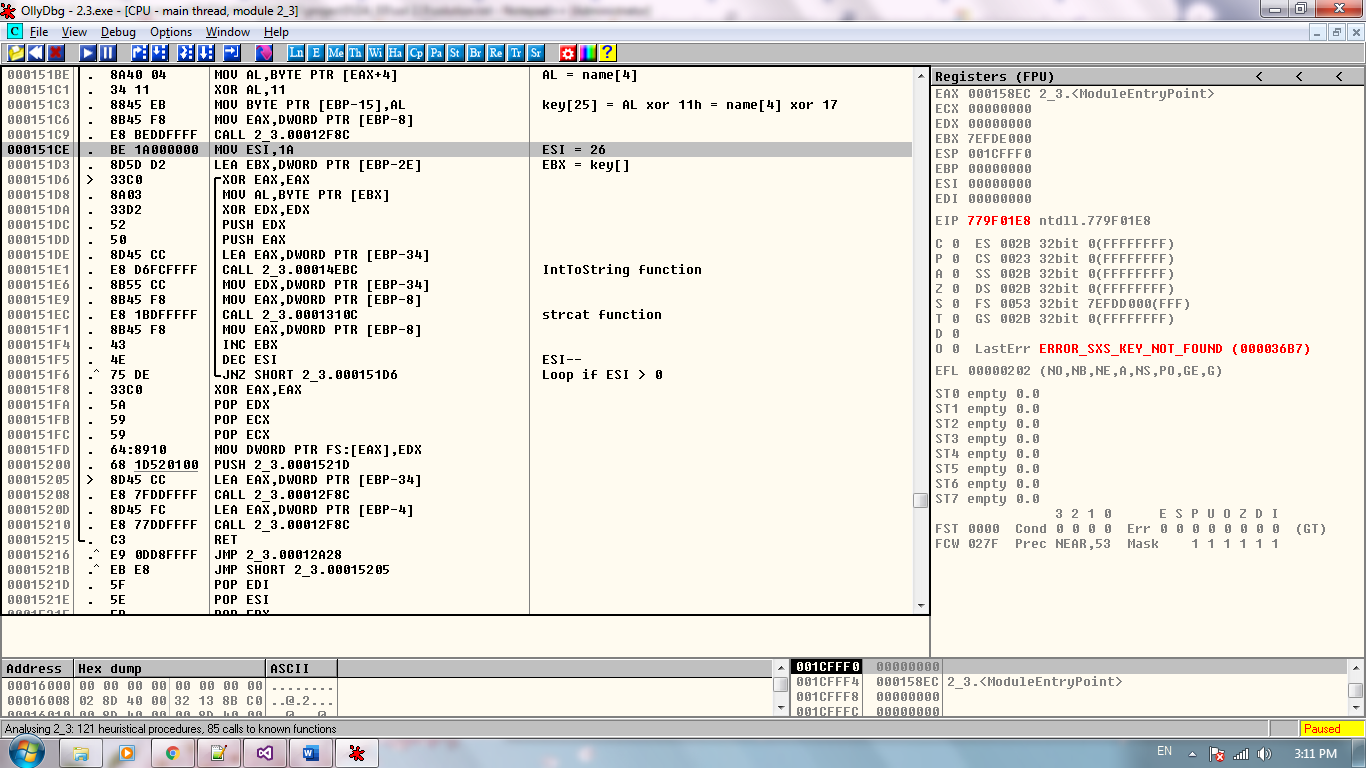








Sau khi xác định 26 giá trị key, đoạn lệnh tiếp theo dùng để ghép 26 giá trị này thành chuỗi serial hoàn chỉnh:



Ví dụ 1 case đúng:

|  |  |
| --- | --- |
| Name: hcmus  Serial: 169198169198541531331301512261833516221023620146472214439418798 |  |