

**ĐẠI HỌC QUỐC GIA TP.HCM**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

## KIẾN TRÚC MÁY TÍNH VÀ HỢP NGỮ

## BÁO CÁO ĐỒ ÁN 3

## CRACKING X86

Giáo viên hướng dẫn: **Lê Viết Long, Phạm Tuấn Sơn, Nguyễn Thị Thanh Huyền**

**Thành viên nhóm**TRẦN QUỐC BẢO – 18120111   
ĐOÀN PHÚ ĐỨC – 18120117   
LÊ MINH KHOA – 18120415  
Lớp : 18CNTN

**Mục Lục**

**[I.](#_Toc45998367)****[Đánh giá mức độ hoàn thành đồ án:](#_Toc45998367)** [3](#_Toc45998367)

**[II.](#_Toc45998368)****[DIABLO CRACKING(LV03)](#_Toc45998368)** [3](#_Toc45998368)

**[1.](#_Toc45998369)****[Mô tả các bước thực hiện Crack](#_Toc45998369)** [3](#_Toc45998369)

**[2.](#_Toc45998370)****[Mô tả các bước tạo key](#_Toc45998370)** [9](#_Toc45998370)

**[III.](#_Toc45998371)****[KEYGENME1 CRACKING(LV03)](#_Toc45998371)** [11](#_Toc45998371)

**[1.](#_Toc45998372)****[Mô tả các bước thực hiện Crack](#_Toc45998372)** [11](#_Toc45998372)

**[2.](#_Toc45998373)****[Mô tả các bước tạo key](#_Toc45998373)** [18](#_Toc45998373)

**[IV.](#_Toc45998374)****[PHOENIX3 CRACKING(LV04)](#_Toc45998374)** [19](#_Toc45998374)

**[V.](#_Toc45998375)****[Tài liệu tham khảo](#_Toc45998375)** [22](#_Toc45998375)

1. **Đánh giá mức độ hoàn thành đồ án:**

|  |  |
| --- | --- |
| **PROBLEM** | **Mức độ hoàn thành** |
| Diablo(lv3) | **100%** |
| Keygenme1(lv3) | **100%** |
| Phoenix3(lv4) | **30%** |
| Đánh giá hoàn thành của cả đồ án | **70%** |

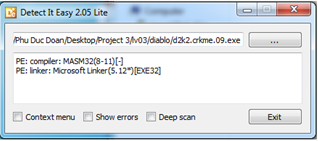
Phần đã hoàn thành của nhóm chúng em:

* Crack, tìm thuật toán và cài đặt phần key gen của 2 problem lv3.

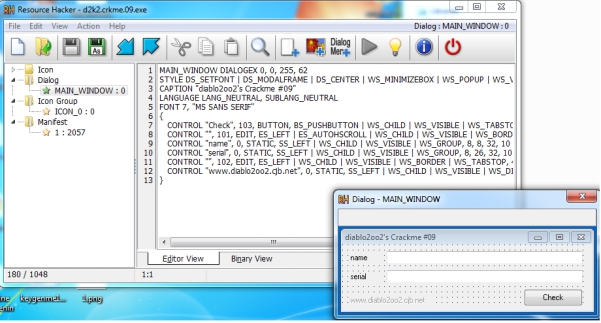
Ngược lại nhóm chúng em cũng có phần chưa hoàn thành:

* Crack Phoenix(lv4) chưa hoàn thiện.

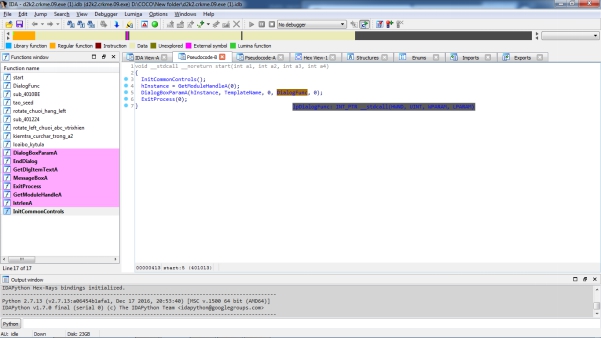
1. **DIABLO CRACKING(LV03)**
2. **Mô tả các bước thực hiện Crack**



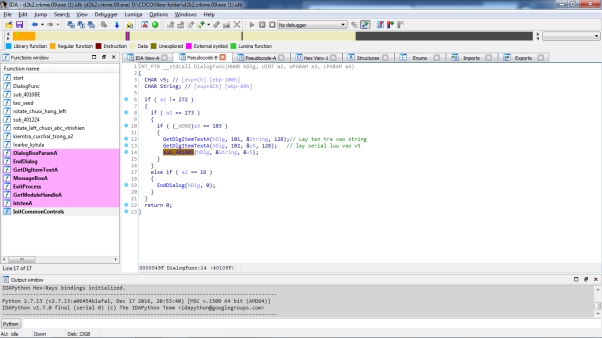
* **Bước 1:** Mở file diablo.exe muốn crack bằng Detect It Easy, ta sẽ thấy chương trình này ở chế độ EXE32. Ta sẽ dùng IDA32 bit để mở file này.



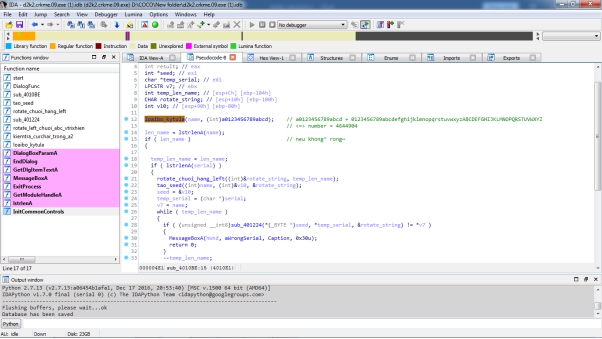
* **Bước 2:** Mở diablo.exe bằng Resource Hacker để biết các Button tương ứng với các mã code nào.



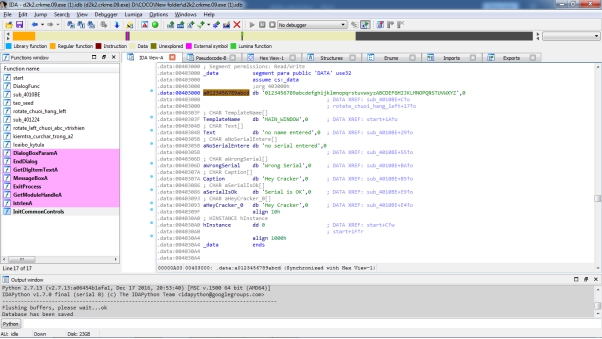
* **Bước 3:** Mở diablo.exe với IDA32 bit và bấm F5 để decompile code Assembly ra code C, lúc này ta cần quan tâm hàm DialogFunc làm việc gì, ta click đúp vào hàm đấy để xem chi tiết.



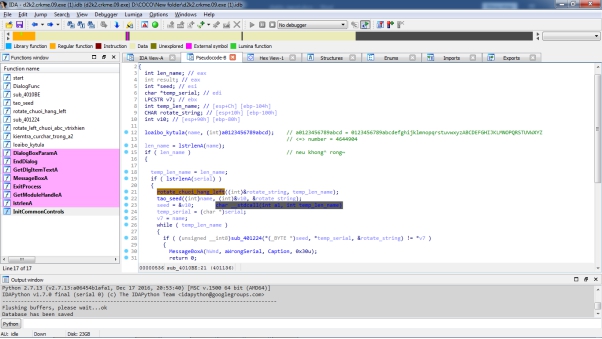
* **Bước 4:** Ta thấy chương trình gọi hàm GetDlgItemTextA để lấy name lưu và biến String và Serial lưu vào biến v5. Sau đó gọi hàm sub\_4010BE để kiểm tra liệu có hợp lệ hay không ? Ta click đúp vào hàm đấy để biết chi tiết.



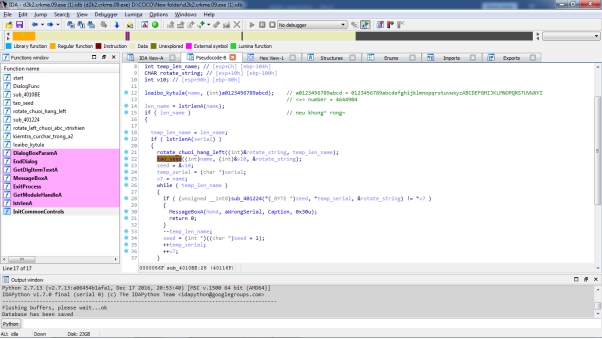
* **Bước 5:** Ta thấy hàm sub\_4012C8 làm gì đấy với chuỗi name nhập vào, click vào đọc chi tiết thì có thể biết được là hàm này thực hiện chức năng chỉ giữ lại các ký tự có trong chuỗi a0123456789abcd.



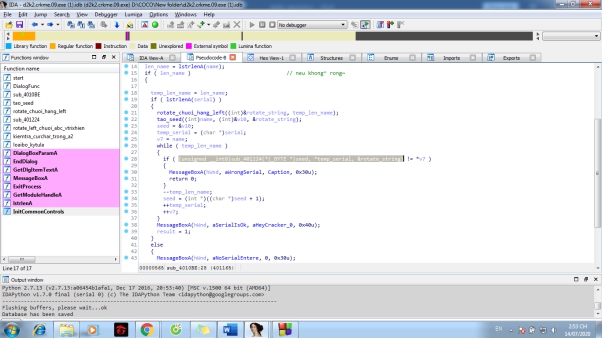
* **Bước 6:** Ta click đúp vào chuỗi a0123456789abcd thì sẽ thấy chuỗi này chỉ gồm các chữ cái Latin và chữ số. Tức là ta có thể biết được hàm đấy dùng để loại bỏ ký tự đặc biệt.



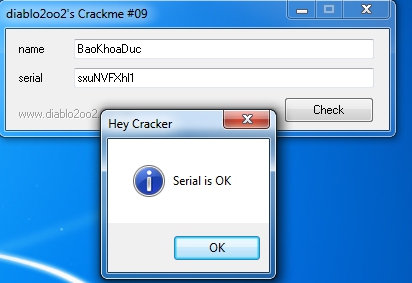
* **Bước 7:** Ta thấy hàm sub\_4011FF thực hiện dịch trái chuỗi a0123456789abcd một lượng là min(30, len(name) \* 4) lần và lưu vào chuỗi rotate\_string. Trong hình ta đã đổi tên hàm.



* **Bước 8:** Tiếp theo, hàm sub\_4011C0 là hàm tạo seed dựa trên name và rotate\_string ở trên và lưu vào v10



* **Bước 9**: Sau đó vòng lặp while sẽ lặp len(name) lần để kiểm tra từng cặp ký tự (seed[i], serial[i], rotate\_string[i]). Nếu có bất kỳ cặp nào sai sẽ xuất ra thông báo sai Serial.



* **Bước 10**: Thử name và Serial được sinh ra từ key gen

1. **Mô tả các bước tạo key**

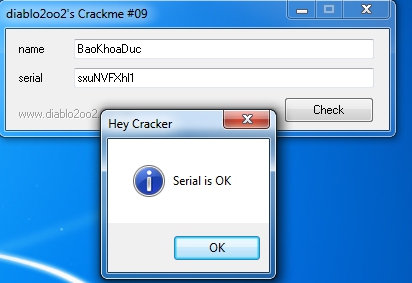
* **Bước 1**: Chuỗi name sẽ được loại bỏ các ký tự lạ, chỉ giữ lại các ký tự nằm trong chuỗi a0123456789abcd = “0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ”
* **Bước 2**: Chuỗi rorate\_string sẽ được dịch trái min(30, len(name) \* 4) lần từ chuỗi a0123456789abcd
* **Bước 3**: Sau đó seed sẽ được tạo dựa vào name và rotate\_string theo thuật toán sau:

|  |
| --- |
| string get\_seed(string name, string rotate\_string) {  int len = rotate\_string.size();  string res = "";  int v6 = ROL3(name[0]); // rol 3 bit ky tu dau tien  for (int i = 0; i < name.size(); ++i) {  char pre\_char = name[i];  char cur\_char = i + 1 < name.size() ? name[i + 1] : 0;  int temp = (v6 + (cur\_char ^ pre\_char)) & 0xFF;  v6 = (v6 + temp) & 0xFF;  res.push\_back(rotate\_string[temp % len]);  }  return res;  } |

* **Bước 4:** Sau đó ta tạo Serial key dựa vào seed, name và rotate string theo thuật toán sau:

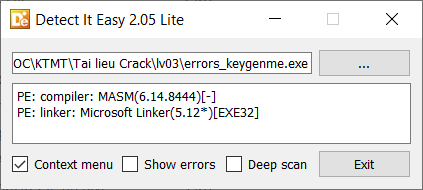
|  |
| --- |
| void chk(char char\_seed, char char\_name, string rot\_string) {  // can tim char\_ser = char\_name  int idx = find\_char(char\_seed, rot\_string);  int need\_idx = find\_char(char\_name, rot\_string);  string rot2 = Rot(rot\_string, idx); // dich sang trai voi idx lần  cout << rot2[need\_idx];  } |

* **Bước 5:** Thử kiểm tra thuật toán với name = “BaoKhoaDuc” ta được Serial = “sxuNVFXhl1”

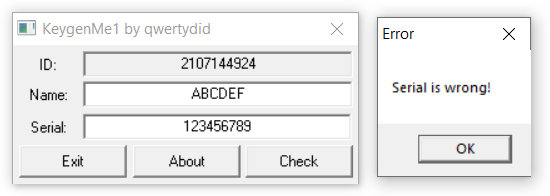


1. **KEYGENME1 CRACKING(LV03)**
2. **Mô tả các bước thực hiện Crack**

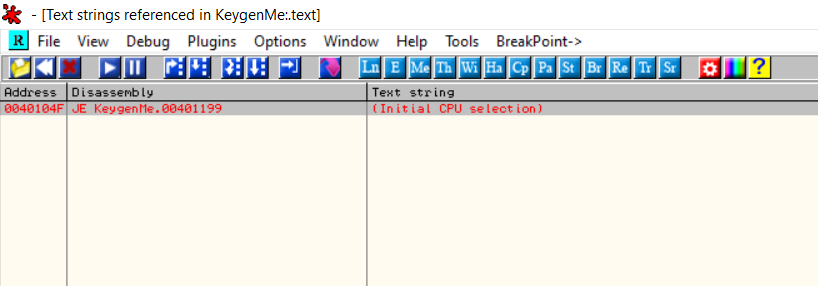
* **Bước 1:** Đầu tiên, ta cần kiểm tra xem chương trình trên là 32bit hay 64bit và có được nén bằng các phần mềm chuyên dụng hay không. Để kiểm tra, ta sử dụng phần mềm Detect It Easy.



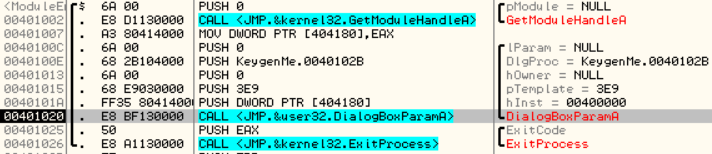
* **Bước 2:** Thông qua các thông tin được hiển thị, ta biết được rằng chương trình trên không được nén và đang ở dạng 32bit. Ta tiến hành chạy thử chương trình để xem cách thức hoạt động và tìm kiếm manh mối.



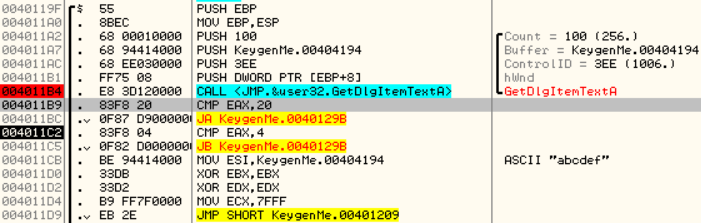
* **Bước 3:** Nhận thấy khi nhập Name và Serial vào và thực hiện check, chương trình trả về chuỗi là “bad boy”, ta tiến hành mở chương trình OllyDbg và tìm kiếm các chuỗi string.



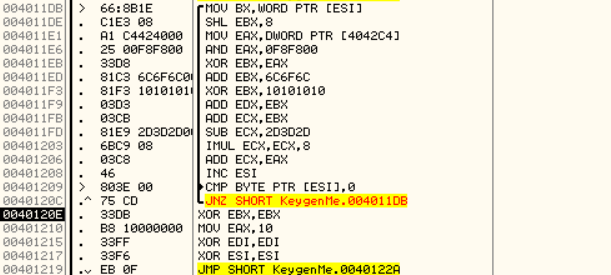
* **Bước 4:** Nhận thấy không tìm được bất kỳ chuỗi ký tự nào trong chương trình, ta có thể đoán được trong quá trình thực hiện, chuỗi sẽ được sinh ra dựa vào một hàm nào đó. Tiến hành thực hiện chạy từng dòng lệnh assembly để tìm kiếm manh mối cho việc bẻ khóa.



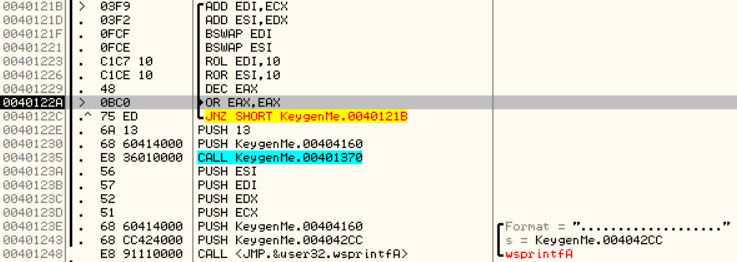
* **Bước 5:** Khi gọi xong hàm DialogBoxParamA, giao diện của chương trình được hiển thị. Để tiếp tục debug, ta tiến hành nhập vào Name và Serial. Thông thường, để lấy nội dung của các ô ta đã nhập, chương trình cần phải gọi hàm GetDlgItemTextA với 4 biến theo thứ tự: địa chỉ cửa sổ, địa chỉ của ô cần lấy, biến chứa chuỗi giá trị được lấy, độ dài chuỗi tối đa. Ta tiến hành tìm và đặt breakpoint tại các hàm trên.



* **Bước 6:** Khi thực hiện check, ta sẽ đi vào breakpoint tại địa chỉ 004011B4. Dựa vào lệnh Push, ta biết được: địa chỉ chuỗi trả về: 00404194, số lượng ký tự tối đa: 100, địa chỉ ô được lấy chuỗi: 1006. Dựa vào phần mềm Resource Hacker, ta biết được 1006 chính là địa chỉ của ô Name. Vậy hàm này thực hiện việc lấy giá trị Name lưu vào vùng nhớ 00404194 và số lượng ký tự được lấy ra sẽ lưu vào thanh ghi EAX. Sau đó, chương trình thực hiện so sánh xem nếu độ dài chuỗi lớn hơn 32 hoặc bé hơn 4 thì sẽ nhảy vào địa chỉ 0040129B. Thực hiện nhập thử các chuỗi có độ dài nhỏ hơn 4 hoặc lớn hơn 32, ta thấy chương trình hiện ra hộp thoại “Name is wrong!”. Vậy có thể xem địa chỉ 0040129B là “bad boy” của ta. Sau đó, gán địa chỉ của vùng nhớ của chuỗi Name vào thanh ghi ESI, gán EBX = EDX = 0, ECX = 32767 và nhảy vào địa chỉ 00401209.



* **Bước 7:** Tại đây, chương trình thực hiện duyệt qua từng ký tự của chuỗi Name. Với mỗi vòng lặp sẽ tính toán các giá trị trong thanh ghi EAX, EBX, ECX, EDX. Điều làm ta thắc mắc chính là việc lưu 4 byte tính từ địa chỉ 4042C4 vào EAX. Vậy địa chỉ 4042C4 là gì? Sau khi duyệt xong, gán EBX = EDI = ESI = 0, EAX = 16 và nhảy đến địa chỉ 0040122A.



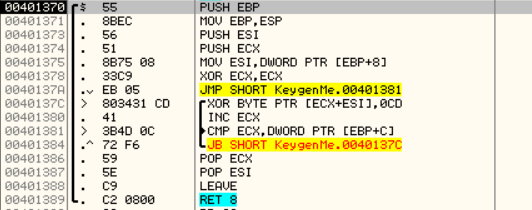
* **Bước 8:** Tại đây, thực hiện lặp đến khi giá trị thanh ghi EAX bằng 0. Với mỗi vòng lặp sẽ thực hiện giảm giá trị EAX đi 1 và tính toán:

EDI = ROL(BSWAP(EDI + ECX), 16)

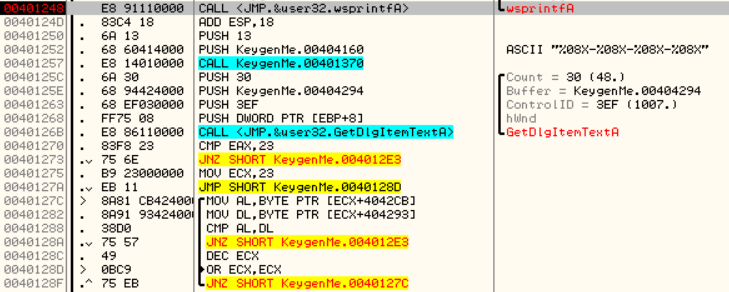
ESI = ROL(BSWAP(ESI + EDX), 16)



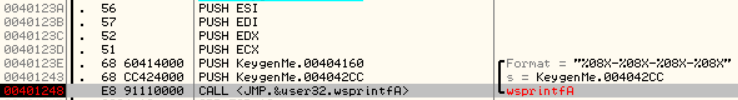
* **Bước 9:** Thực hiện xong vòng lặp, chương trình tiếp tục gọi hàm ở địa chỉ 00401370.



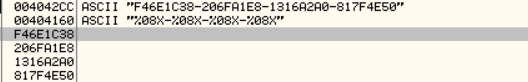
* **Bước 10:** Hàm trên thực hiện lặp i từ 0 đến nhỏ hơn giá trị của con trỏ tại địa chỉ EBP + C. Với mỗi vòng lặp sẽ thực hiện XOR byte thứ i của con trỏ ESI với 0CD.   
    
  Thoát khỏi hàm, ta sẽ nhảy đến dòng lệnh ở địa chỉ 00401248. Dòng lệnh này sẽ gọi hàm wsprintfA trong thư hiện Winuser.h. Tìm hiểu, ta biết hàm trên có cú pháp int WINAPIV wsprintfA(LPSTR, LPCSTR, v1, v2, ...). Hàm sẽ thực hiện chức năng ghi vào biến LPSTR một chuỗi chứa các giá trị của v1, v2,... với định dạng theo biến LPCSTR.



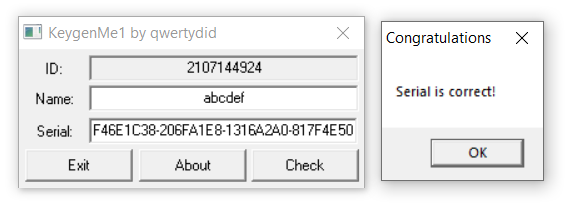
* **Bước 11:** Để kiếm tra số lượng tham số được truyền vào hàm wsprintfA, ta xem các thanh ghi vào biến nào sẽ được lưu vào stack trước khi thực hiện lệnh CALL.



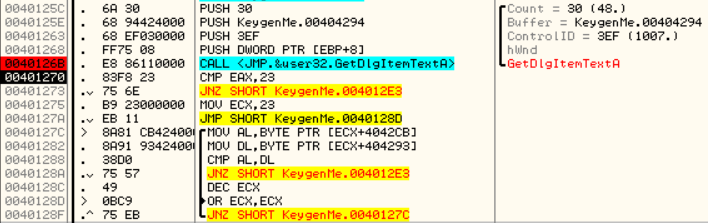
* **Bước 12:** Theo dấu các lệnh ở trên, ta rút ra được nhận xét:
  + Biến LPSTR: là bộ nhớ ở địa chỉ 004042CC. Đây chính là địa chỉ của vùng nhớ sẽ được lưu chuỗi khi hàm wsprintfA được thực hiện xong.
  + Biến LPCSTR: là bộ nhớ ở địa chỉ 00404160. Ta còn có nhận xét được định dạng của chuỗi được lưu trong LPSTR sau khi thực hiện hàm: XXXXXXXX- XXXXXXXX- XXXXXXXX- XXXXXXXX. Chuỗi trên khá giống serial trong các phần mềm, phải chăng đây chính là chuỗi ta cần tìm? Ta còn nhớ hàm 00401370 sau khi gọi đã thay đổi giá trị của chuỗi ở địa chỉ 00404160. Vậy có thể kết luận rằng hàm 00401370 chính là hàm có chức năng tạo ra định dạng của serial. Việc sử dụng các phép XOR trong hàm có thể là một thuật toán tạo định dạng serial nào đó. Sau hàm wsprintfA, hàm 00401370 lại được gọi lần nữa với chính tham số là địa chỉ 00404160. Sau khi gọi, dữ liệu tại 00404160 bị mất. Có thể kết luận người lập trình đã gọi hàm này để xóa dữ liệu chứa định dạng serial.
  + Các biến v1, v2, v3, v4: ECX, EDX, EDI, ESI.



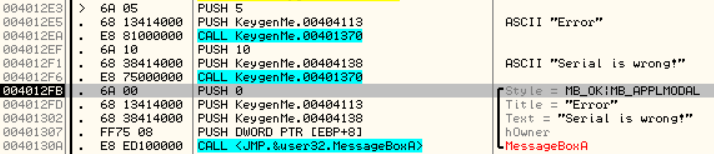
* **Bước 13:** Tìm trong data segment, ta thấy được giá trị của biến LPSTR. Tiến hành nhập thử Name và serial vào trong chương trình và điều kỳ diệu xảy ra!



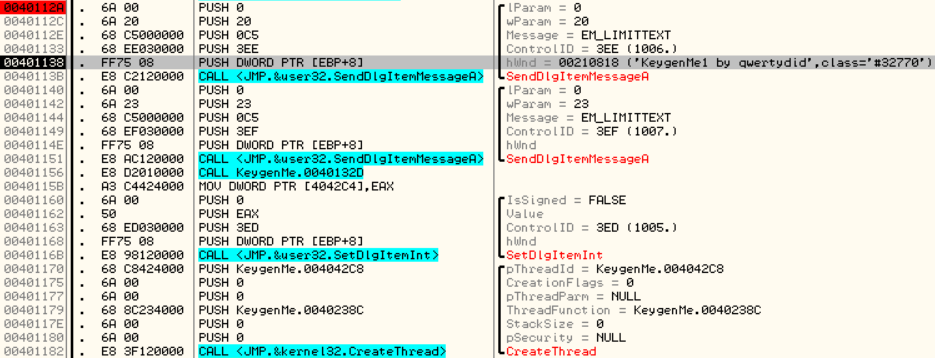
* **Bước 14:** Ta kết luận hàm wsprintfA chính là hàm sẽ tạo ra serial. Khi đó, ta cần tìm giá trị của các thanh ghi ECX, EDX, EDI, ESI. Đã tim được địa chỉ vòng lặp tính toán giá trị của EDI và ESI tại địa chỉ 0040122A và nơi tính toán giá trị của ECX và EDX tại địa chỉ 00401209. Ta chỉ cần tìm nội dung được lưu tại địa chỉ 4042C4 là có thể biết được đầy đủ cách tính toán ra serial.   
  Tiến hành chạy tiếp chương trình, ta tìm được vị trí lấy ra serial và gán vào vùng nhớ tại địa chỉ 00404294.



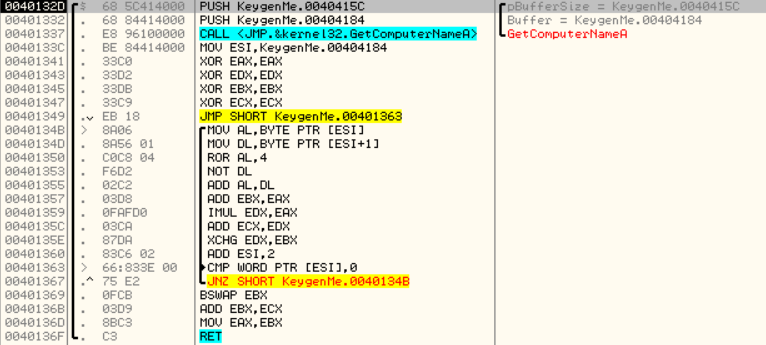
* **Bước 15:** Sau đó, chương trình so sánh nếu độ dài serial lấy ra khác 35 thì sẽ nhảy vào địa chỉ 00401280. Ta nhận thấy sau khi gọi hàm 00401370, các vùng nhớ là tham số đầu vào sẽ sinh ra nội dung chuỗi ASCII của “bad boy” và hàm CALL <JMP.&user32.MessageBoxA> sẽ sinh ra hội thoại báo serial is wrong. Vậy đây chính là “bad boy” của chúng ta.



* **Bước 16:** Nhận thấy rằng có một trường rất quan trọng mà ta chưa đụng đến là ID. Rất có thể đó là ID của máy tính chúng ta. Để có thể hiện được nội dung như chuỗi, ID lên cửa sổ, ta cần phải dùng các hàm có dạng SendDlgItemMessage. Tiến hành đặt breakpoint tại trước các hàm SendDlgItemMessage để tìm kiếm manh mối.



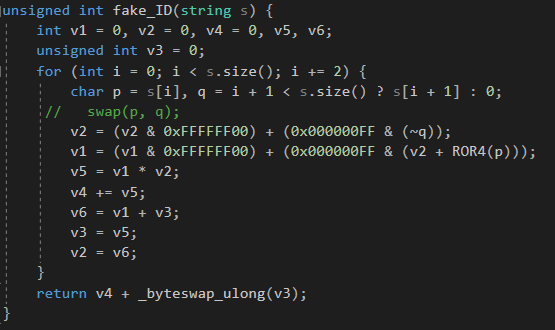
* **Bước 17:** Duyệt qua, ta thấy hàm SendBlgItemMessageA ở địa chỉ 0040113B là nơi tạo tiêu đề cho cửa sổ. Không những thế, chương trình thực hiện gọi hàm ở địa chỉ 0040132D.



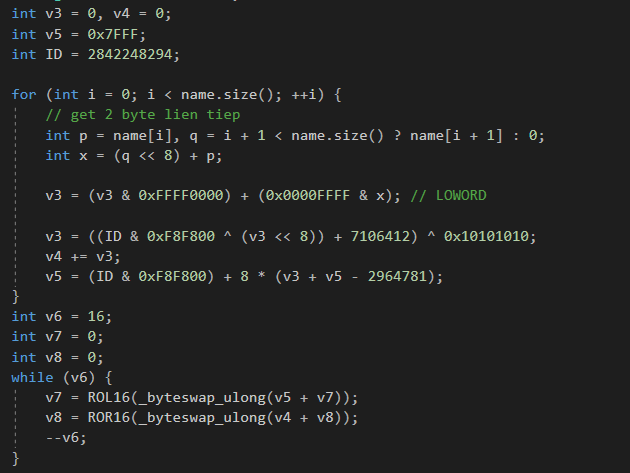
* **Bước 18:** Thấy hàm GetComputerNameA có tác dụng lấy ra tên máy tính được gọi, ta nghi ngờ hàm này có tác dụng thực hiện tính ID cho chương trình. Sau khi gọi hàm, địa chỉ vùng nhớ chứa tên máy tính được lưu vào thanh ghi ESI. Sau đó lần lượt duyệt đến từng cặp ký tự của tên máy tính, với mỗi vòng lặp thực hiện tính toán trên các thanh ghi AL, DL, EBX, EDX, ECX. Sau đó, thực hiện trả về EAX = ECX + ByteSwap(EBX). Kiếm tra giá trị trả về EAX = 7D987ADC = 2107144924 được lưu vào vùng nhớ ở địa chỉ 4042C4 chính là ID được hiển thị trên phần mềm. Vậy hàm tại địa chỉ 0040132D là nơi tạo ra ID. Nên ta đã tìm được nội dung lưu tại địa chỉ 4042C4 giúp có đầy đủ thông tin để tính toán ra giá trị của các thanh ghi ECX, EDX.

1. **Mô tả các bước tạo key**

* **Bước 1:** Tính ID của từng máy tính dựa vào tên máy tính.



* **Bước 2:** Tính toán giá trị của các thanh ghi ECX, EDX dựa trên ID và tên nhập vào.
* **Bước 3:**Tính toán giá trị các thanh ghi EDI, ESI dựa vào giá trị các thanh ghi ECX và EDX.



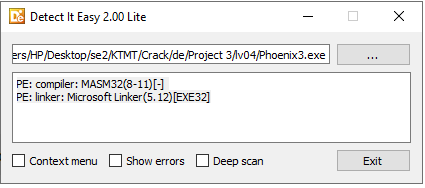
* **Bước 4:** Tạo ra chuỗi serial bằng hàm wsprintfA và giá trị các thanh ghi EDI, ESI, ECX, EDX.



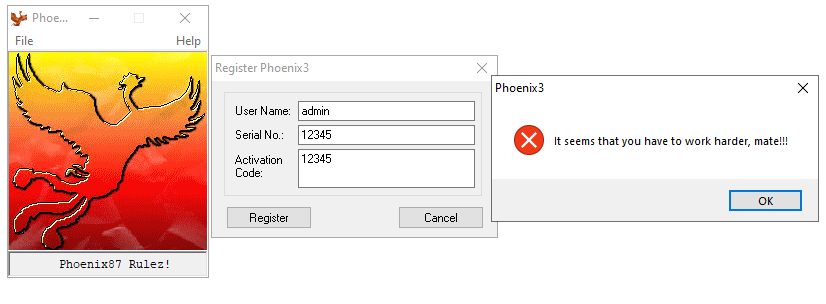
* **Bước 5:** So sánh từng kí tự của chuỗi serial vừa tạo và chuỗi serial nhập vào. Nếu đúng hiện ra “good boy”.

1. **PHOENIX3 CRACKING(LV04)**

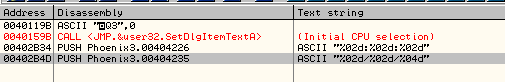
* **Bước 1:** Mở file phoenix3.exe muốn crack bằng Detect It Easy, ta sẽ thấy chương trình này ở chế độ EXE3.



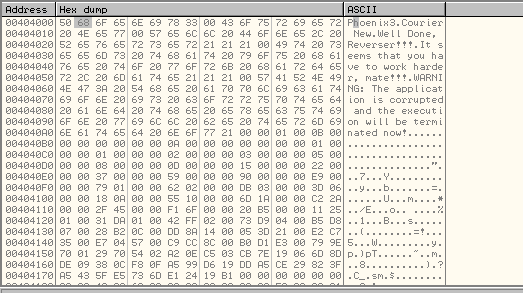
* **Bước 2:** Mở file Phoenix3.exe lên, tiến hành chạy thử chương trình và tìm kiếm manh mối để tiếp tục:



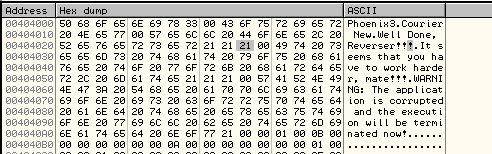
* **Bước 3:** Sử dụng Ollydbg, mở file Phoenix3.exe. Ta nhận thấy không thể tìm được bất kì “bad boy” hay “good boy” nào bằng cách search string.



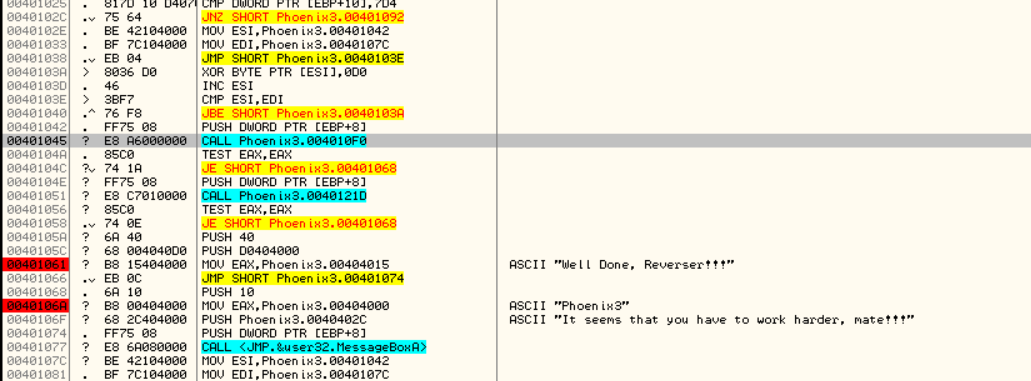
* **Bước 4:** Mở Ollydbg, lúc đầu ta xem thử phần memory ta thấy:



Sau khi chạy chương trình, ta thấy phần memory thay đổi và có những message đáng lưu ý:



* **Bước 5:** Đặt BreakPoint tại 0041061, 40106A, 401077 tại MessageBoxA ta phát hiện ra đc một số “bad boy” , tuy nhiên đến đây nhóm em bị bí ý tưởng và vẫn chưa phát hiện ra được manh mối nào để đi tiếp.



* Ngoài ra, chúng em đã thử đặt Break Point tại dialogfunc, thử debug trong các hàm Call Phoenix3.xxx,… cũng như xem sự thay đổi memory, nhưng vẫn không thể tìm ra được goodboy.

1. **Tài liệu tham khảo**

* Tài liệu tham khảo trên trang Moodle môn Kiến trúc máy tính và hợp ngữ.
* [IDA Pro - Hex Rays](https://www.hex-rays.com/products/ida/)
* [Detect it easy](http://ntinfo.biz/index.html)
* [Resource Hacker](http://www.angusj.com/resourcehacker/)