INTRODUCTION TO THE CRACKING WITH OLLYDBG

FROM CRACKLATINOS

(_kienmanowar_)

Một cái đầu lạnh để vững vàng, một trái tim đỏ lửa để yêu và làm việc hết mình!

I. Giới thiệu chung

Chào các bạn, hôm nay chúng ta lại gặp nhau ở phần 13 của loạt bài viết về Olly ☺. Vẫn còn rất nhiều phần khác nữa tới đây, chỉ sợ sức lực tôi có hạn không thể viết hết được thôi.. khà khà. Trong toàn bộ 12 bài viết trước, tôi đã lần lượt giới thiệu cho các bạn về Ollydbg, các kiến thức cơ bản về ASM, các câu lệnh thường được sử dụng, cách patch chương trình cũng như các kiểu BP từ cơ bản đến nâng cao trong Olly và còn nhiều thông tin khác nữa.... Tôi hi vọng rằng qua 12 bài viết đó các bạn đã tự trang bị cho mình những kĩ năng cơ bản nhất để làm việc với Olly, cũng như tích lũy được những kinh nghiệm để có thể làm việc tiếp với những bài viết chuyên sâu tiếp theo của loạt tutor này. Vậy ở phần 13 này chúng ta sẽ làm gì nhỉ? Thực ra là phần 13 này không có trong kịch bản của lão Rincardo đâu, mà là tự tôi viết. Vì trong phần 12 của lão, cuối bài lão có nói lão sẽ xử lý Crackme Cruehead để tóm được Serial nhưng rồi lão lại không viết ở phần 13, thay vào đó lão đi xử lý các crackme khác. Cho nên tôi quyết định tự tay xử lý Crackme này để phục vụ các bạn ☺. Rất nhiều điều thú vị đang nằm ở phía trước…. N0w….L3t's F1nish H1M !!!!!!!

II. Let's Finish Him ☺

Nói là xử lý nhưng chúng ta phải làm thế nào nhỉ? Người có kiến thức và kinh nghiệm thì bảo : *"Hãy kiểm tra chương trình trước xem có bị pack bởi packer nào không? Nếu bị pack thì giải quyết packer trước rồi tính tiếp. Còn nếu không bị pack thì quá khỏe, chạy thử chương trình xem nó hoạt động ra sao và tìm kiếm thông tin. Sau khi có được những thông tin quan trọng thì load chương trình vào Olly, tìm các cách để tiếp cận, đặt BP ở những điểm mấu chốt, sau đó trace code, comment những chỗ quan trọng, nếu fish được serial thì tốt, còn không thì tìm ra thuật toán và code keygen v..v..Bạn hãy tự mình thực hành đi đã, nếu bị bí chỗ nào hãy post lên để hỏi!"*

Những người biết thì thưa thớt nhưng lại thích khoe khoang cũng phán đại :*"Thì load vào Olly, tìm cái chuỗi liên quan đến Nag ấy, rồi đặt BP chứ còn làm gì nữa! Không làm được thì show code lên đây tôi giúp cho v..v.."*

Riêng cá nhân của tôi thì thấy rằng: *"Phải tự mình đúc kết các kinh nghiệm trước khi lâm trận cái đã, khi bạn chưa biết gì mà đã vội nhảy vào trận chiến thì chẳng khác nào lấy trứng chọi đá. Vậy kinh nghiệm ở đâu ra? Kinh nghiệm có được khi bạn đọc những bài viết của người khác, có được khi bạn thực hành với những trường hợp tương tự nhưng bạn thử nghiệm những hướng tiếp cận khác ,kinh nghiệm có được khi bạn tham gia thảo luận một chủ đề kĩ thuật v..v.. Để rồi từ đó bạn rút tỉa dần dần và tích lũy lại thành kinh nghiệm của riêng mình.Rồi sẽ đến một lúc nào đó, lại có người muốn ta chia sẻ kinh nghiệm của mình. Không ai có đủ thời gian và kiên nhẫn để chỉ dạy từng bước cho bạn, bạn phải tự mình tìm tòi và khám phá, khi nào bạn cảm thấy thực sự cần đến sự giúp đỡ tôi nghĩ lúc đó sẽ có người sẵn sàng giúp bạn ☺"*

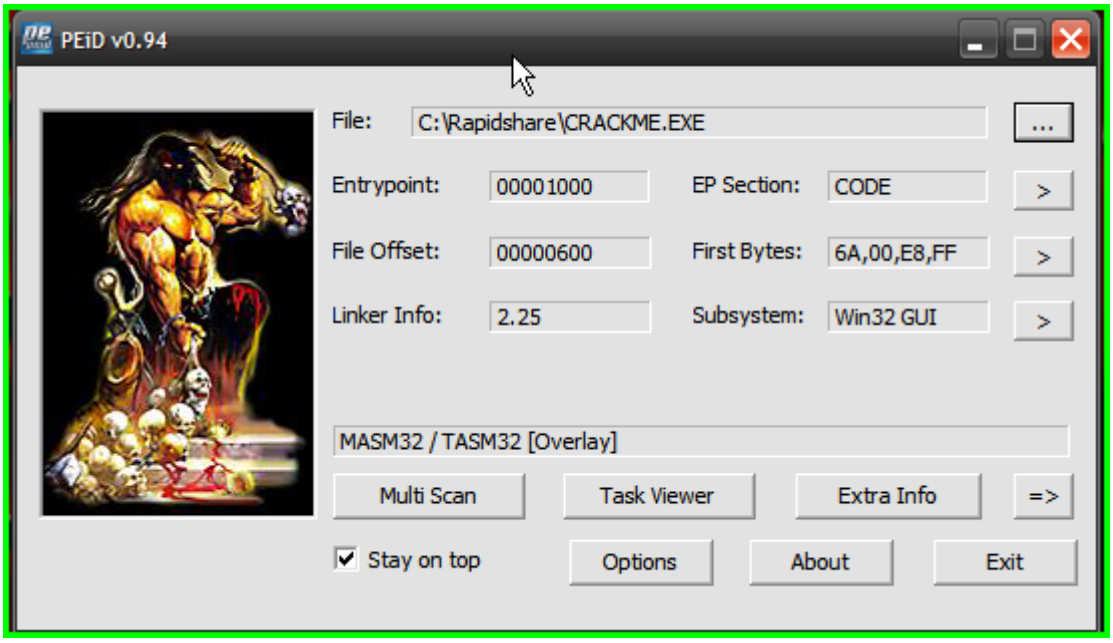
Quay trở lại phần chính của bài viết này là giải quyết crackme CrueHead để tìm ra một valid serial. Một hướng tiếp cận cơ bạn sẽ như tôi trình bày bên dưới đây, đương nhiên không nằm ngoài khả năng có những cách tiếp cận khác, điều đó nằm ở sự khám phá của các bạn ☺.

1. Kiểm tra xem chương trình có bị pack hay không?

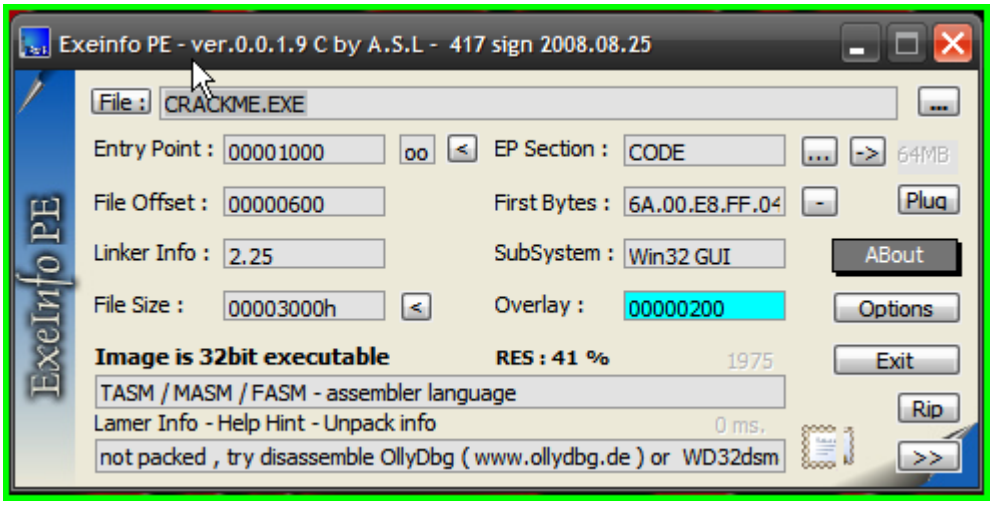
Pack file nghĩa là như thế nào? Tại sao phải kiểm tra xem có bị pack? Hiểu một cách đơn giản thì pack file là nén file thực thi (PE file : .dll, .exe, .ocx, v..v..) để làm giảm kích thước của file, việc nèn này ngoài việc nén code, data của chương trình thì trình packer còn thêm cả đoạn decompress stub vào PE file để làm nhiệm vụ unpack chương trình trong memory. Việc nén này không nên hiểu như ta dùng Winrar/Winzip để nén file, vì Winrar/Winzip sau khi nén file xong ta không thể thực thi file đó được mà ta phải làm một bước là extract file, sau đó mới run file.

Khi một file không bị pack thì lúc ta load chương trình vào Olly ta sẽ dừng lại tại EP của chương trình (hay còn gọi là OEP gốc). Còn nếu chương trình đã bị pack, khi ta load vào Olly ta sẽ dừng lại tại EP của packer chứ không phải là EP của chương trình. Do đó nhiệm vụ của chúng ta là phải unpack chương trình trước đã (tức là ta đi tìm lại OEP gốc), rồi mới thực hiện các hướng tiếp cận khác. Đó chính là lý do tại sao ta phải kiểm tra chương trình. Vậy ta kiểm tra như thế nào? Tôi thường sử dụng một số chương trình sau để check :

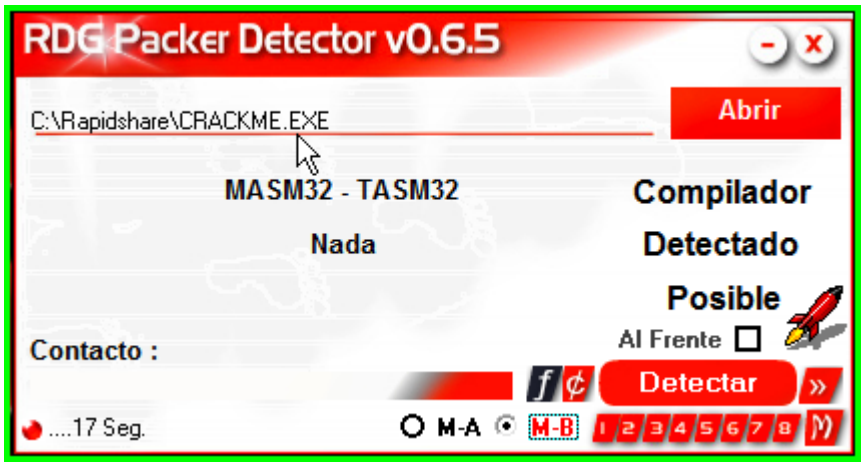
a) **PeiD v0.94/v0.95** :



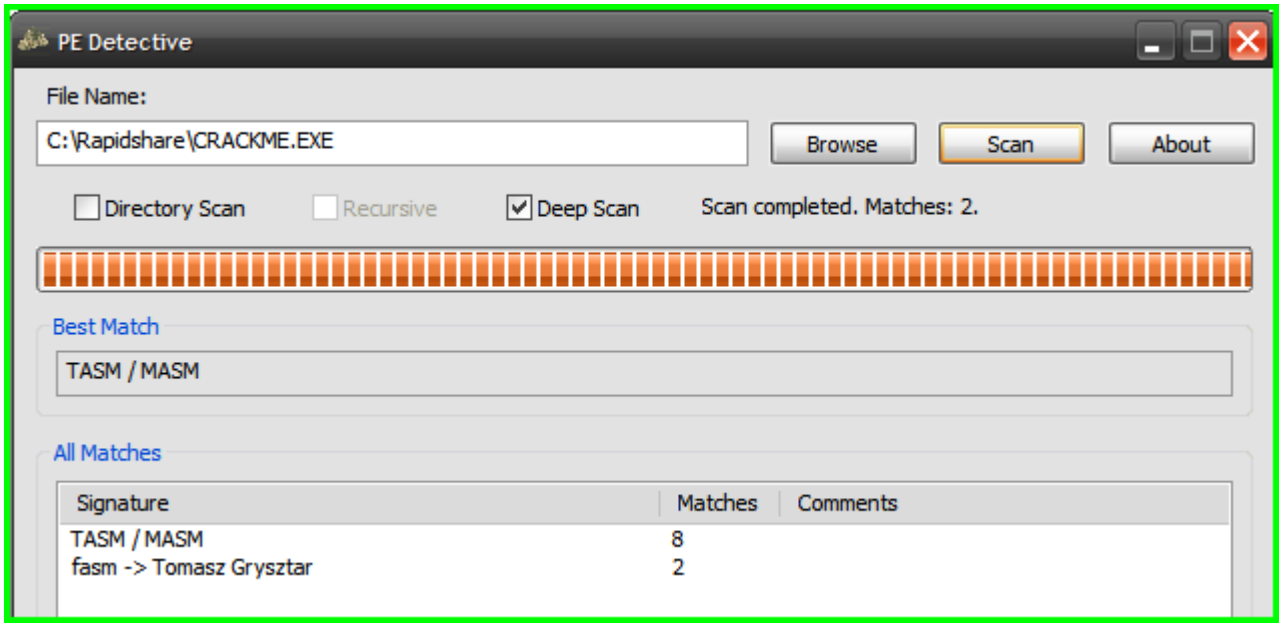
b) **ExeInfo PE** :



c) **RDG Packer Detector**:



d) **PE Detective** :



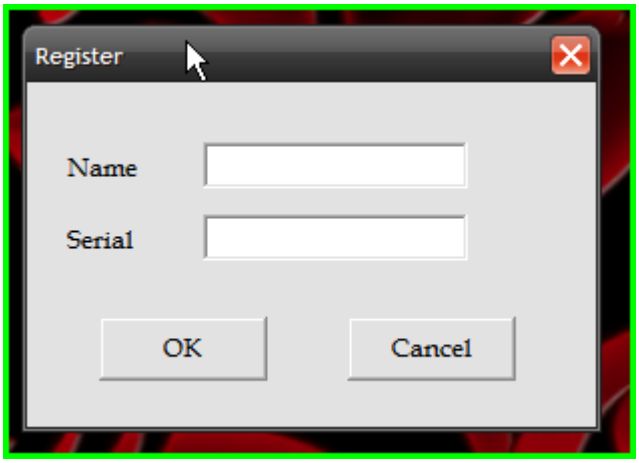
Kết quả như các bạn đã thấy, sau khi sử dụng một loạt các chương trình PE detector ta nhận được kết quả là :

- Chương trình không bị pack bởi bất kì packer nào.
- Chương trình có thể được code bằng Masm32 hoặc Tasm32

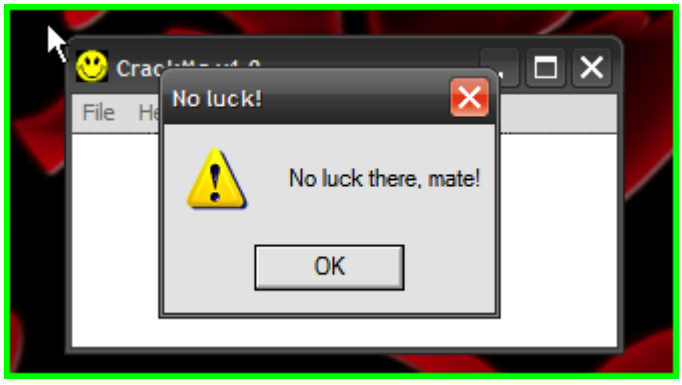
Việc chương trình không bị pack cũng đồng nghĩa với việc ta không cần phải unpack chương trình nữa, vậy là nhẹ được một bước.Ta chuyển qua bước kế tiếp ☺

2. Chạy thử chương trình để tìm kiếm mục tiêu cần tiếp cận.

Việc chạy thử chương trình nhằm mục đích giúp cho ta có một cái nhìn tổng quan về hoạt động của chương trình, biết được nhưng mục tiêu mà ta cần giải quyết. Song song với đó ta sẽ tìm kiếm các cách thức để tiếp cận mục tiêu. Ok giờ tôi chạy thử chương trình xem thế nào đã. Sau khi run tôi thấy nó không show nag gì cả. Nhìn sơ qua cũng chưa biết là mục tiêu tiếp cận ở đâu. Nếu các bạn để ý khi sử dụng các chương trình thì chức năng **Register** thường được đặt ở Menu Help. Do đó tôi nhấn chuột thử vào menu Help xem thế nào, blah blah..tôi thấy có phần Register, nhấn vào đó tôi nhận được :



Chà, tiếp theo làm gì nữa đây? Chúng ta nhập đại Name và Serial vào và hi vọng là nó đúng ☺. Sau khi nhập và nhấn OK tôi nhận được nguyên cái Nag!



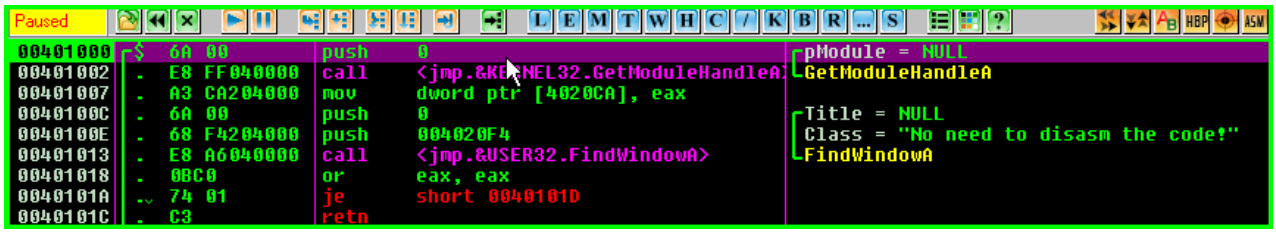
Ok, thông qua việc thực thi chương trình và nhập thông tin đăng kí như trên ta rút ra được một số mục tiêu tiếp cận như sau :

- Tiếp cận thông qua việc tìm kiếm chuỗi ***"No luck there, mate!"***
- Tiếp cận thông qua hàm **MessageBoxA**.
- Tiếp cận thông qua việc nhận Name và Serial (thông qua **GetDlgItemTextA**)
- v..v…

Như vậy sơ sơ ta cũng đã có 3 hướng tiếp cận rồi, việc chọn cách nào là tùy ở bạn. Bạn cũng có thể tự tìm ra một hướng khác với những hướng đã liệt kê ở trên. Ở đây tôi chọn cách thứ 3 là dụng hàm **GetDlgItemTextA** để tiếp cận và giải quyết crackme này!

3. Dùng Olly + Brain để giải quyết bài toán

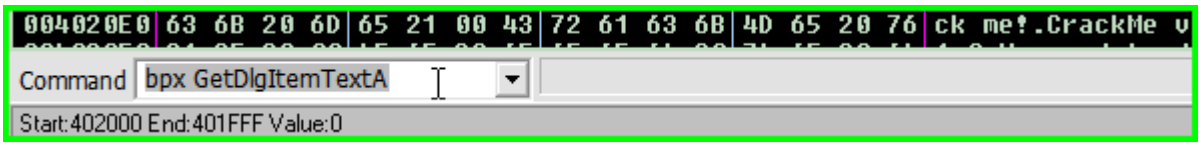
Sau hai bước 1 và 2 chúng ta đã thu lượm được những thông tin cần thiết cho việc giải quyết bài toán. Ở bước 3 này chúng ta sẽ nhờ đến **Ollydbg + Brain** để giải quyết bài toán hóc búa này ☺. Tại sao lại là Olly và Brain nhi? Vì đơn giản Ollydbg chỉ là một công cụ phục vụ cho mục đích của chúng ta, việc làm chủ và sử dụng thành thạo nó là kĩ năng tối thiểu và cần thiết khi chúng ta muốn debug một chương trình. Còn việc phải làm ra sao, tìm các tuyệt chiêu thế nào, phương hướng tiếp cận làm sao nhanh nhất có thể v..v.. lại nằm ở trí thông minh và sức sáng tạo của chúng ta. Quay trở lại bài toán của chúng ta, load Crackme vào trong Olly đã :



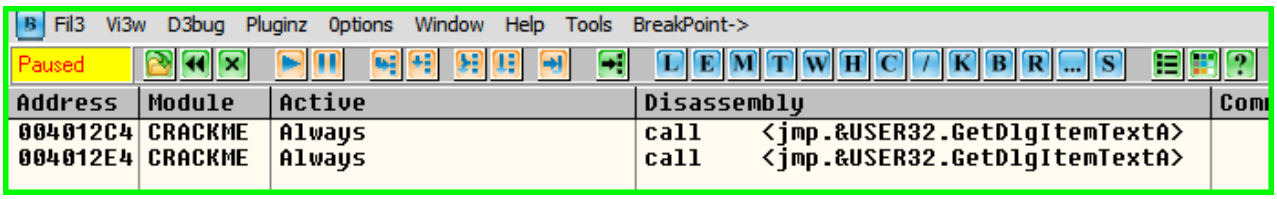
Như đã nói ở trên, hướng tiếp cận của tôi lúc này là đi theo hàm **GetDlgItemTextA**. Thông tin về hàm này như sau :

```
The GetDlgItemText function retrieves the title or text associated with a
control in a dialog box.

UINT GetDlgItemText(
    HWND hDlg,          // handle of dialog box
    int nIDDlgItem,     // identifier of control
    LPTSTR lpString,    // address of buffer for text
    int nMaxCount       // maximum size of string
   );

Return Values
If the function succeeds, the return value specifies the number of characters
copied to the buffer, not including the terminating null character.

If the function fails, the return value is zero.
```

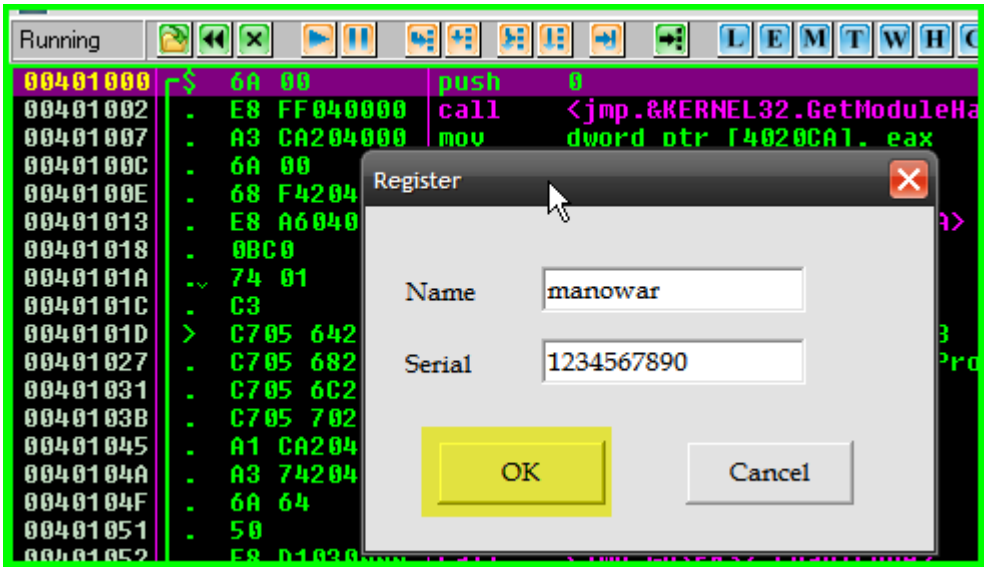
Trờ lại Olly, tôi tiến hành đặt BP tại hàm **GetDlgItemTextA** :

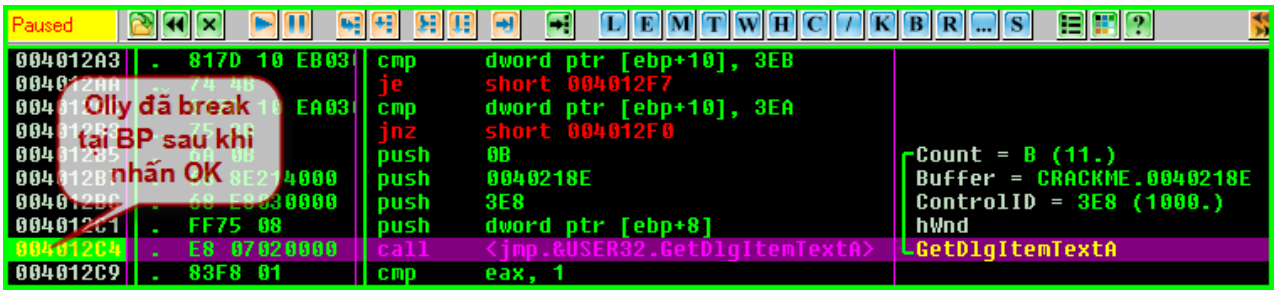


Chuyển tới cửa sổ Breakpoints ta thấy có hai BP được thiết lập, vậy ta phỏng đoán hai hàm này tương ứng với hai lần Get Name và Get Serial :

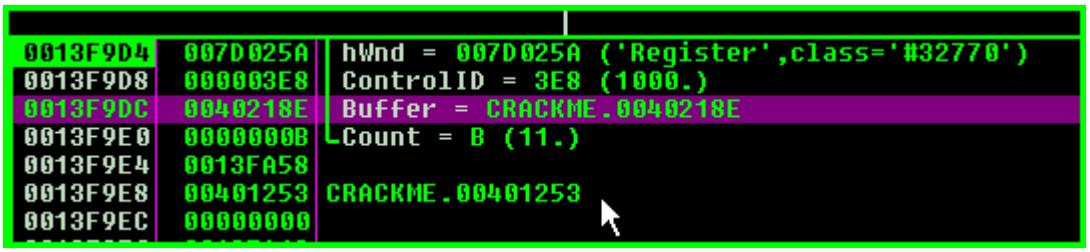


Tiếp theo ta nhấn **F9** để thực thi chương trình, nhập Name và Serial vào sau đó nhấn Ok và hi vọng Olly sẽ break tại nơi ta vừa thiết lập BP.

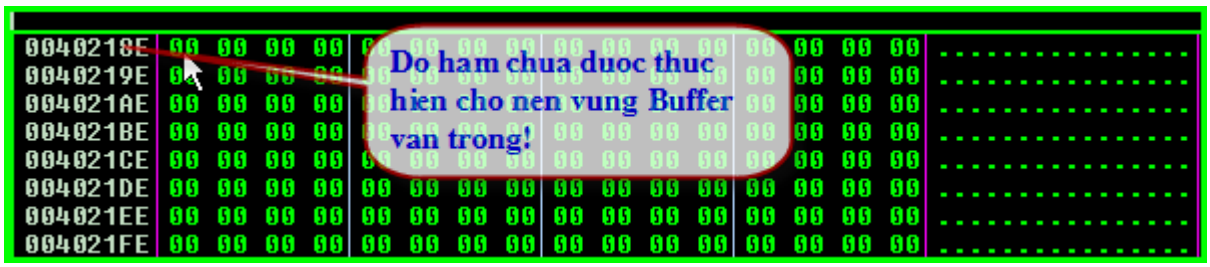




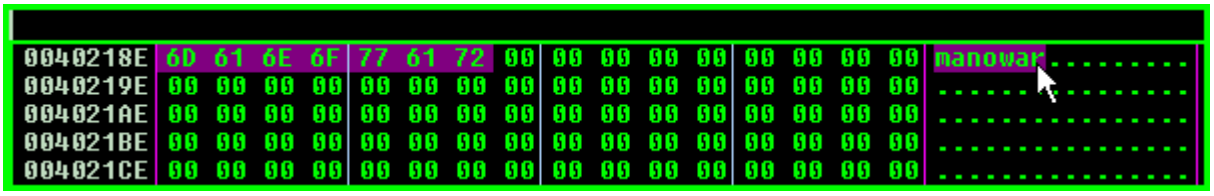
Như các bạn thấy trên hình, ta đang dừng lại tại lời gọi tới hàm **GetDlgItemTextA** đầu tiên. Nhìn sang cửa sổ Stack ta có thông tin sau :



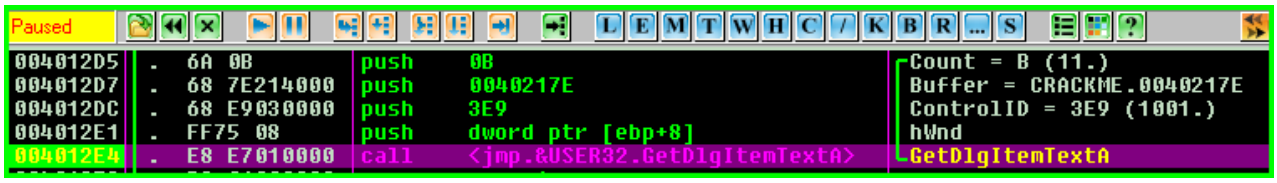
Không cần giải thích chắc các bạn cũng hiểu các thông tin này, chúng tương ứng với các tham số truyền vào cho hàm trước khi hàm được thực hiện. Ở đây chúng ta quan tâm tới vùng **Buffer**, vì nếu hàm thành công thì đoạn text nhập vào sẽ được lưu tại vùng **Buffer** này. Ta chọn **Buffer**, chuột phải và chọn **Follow in Dump** :

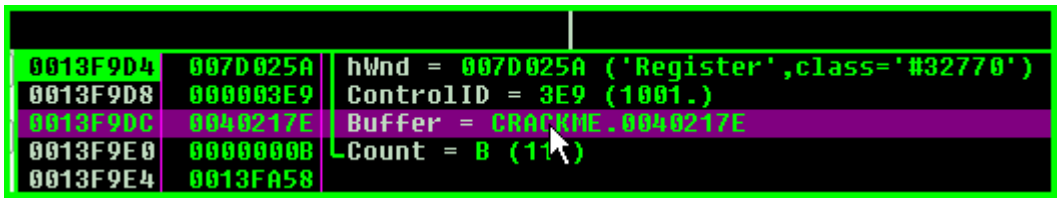


Bây giờ ta nhấn **F8** để trace và thực hiện hàm **GetDlgItemTextA**, sau đó quan sát kết quả tại vùng **Buffer** :



Như vậy là ta đã có được chuỗi Name rồi ☺. Để ý một chút bên cửa sổ Register ta sẽ thấy thanh ghi eax lưu độ dài của chuỗi Name (`eax=00000007`). Sau khi có được chuỗi Name như trên, tiếp tục nhấn **F9** để thực thi chương trình, ta sẽ Break tại lời gọi tới hàm **GetDlgItemTextA** tiếp theo :



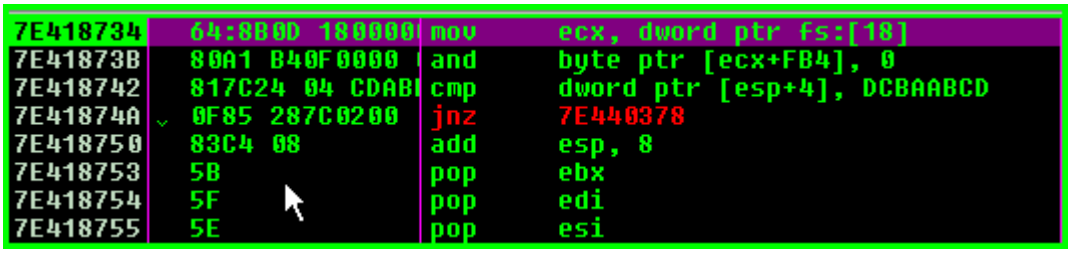


Tiếp tục **Follow in Dump** tại vùng Buffer : `0x0040217E`. Sau đó nhấn **F8** để trace và thực hiện hàm **GetDlgItemTextA** thứ hai :

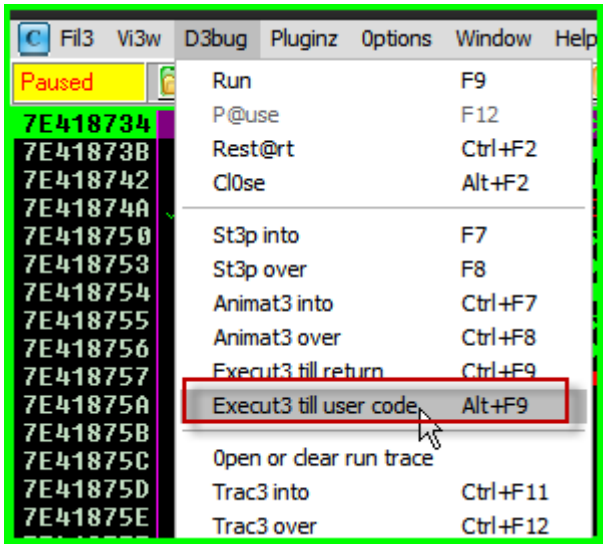
Sau khi thu được hai chuỗi Name và Serial, ta tiếp tục nhấn **F8** để trace code.Trace qua mấy đoạn code ta tới đây :

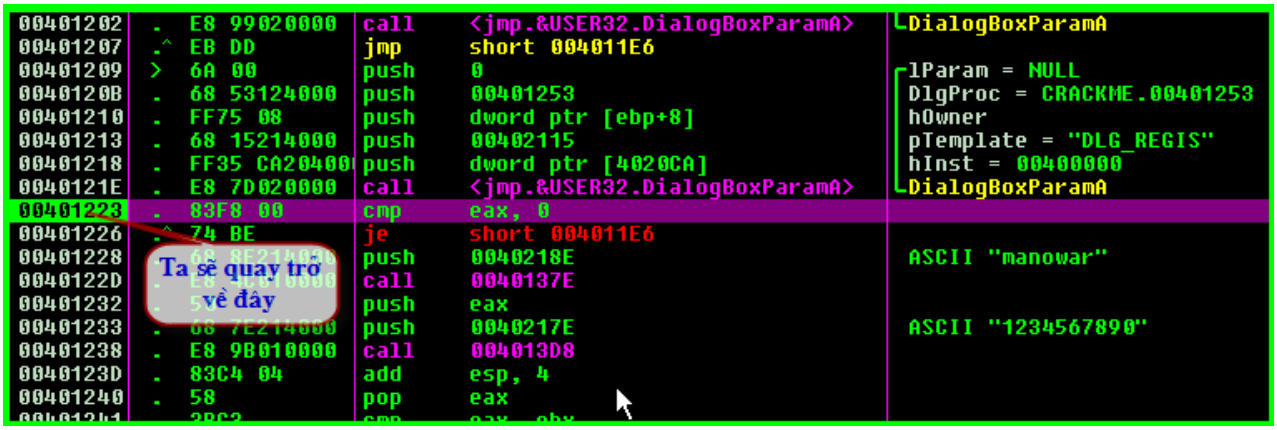
```
00401284   |> /5F            pop     edi
00401285   |. |5E            pop     esi
00401286   |. |5B            pop     ebx
00401287   |. |C9            leave
00401288   |. |C2 1000       retn    10
```

Tiếp tục trace và thực hiện lệnh `Retn 10`, ta bị return tới vùng code không phải là code chính của crackme :



Nếu các bạn tiếp tục trace tiếp thì sẽ vẫn luẩn quẩn trong đám code này, vậy để cho nhanh chóng về vùng code chính của chương trình ta chọn :





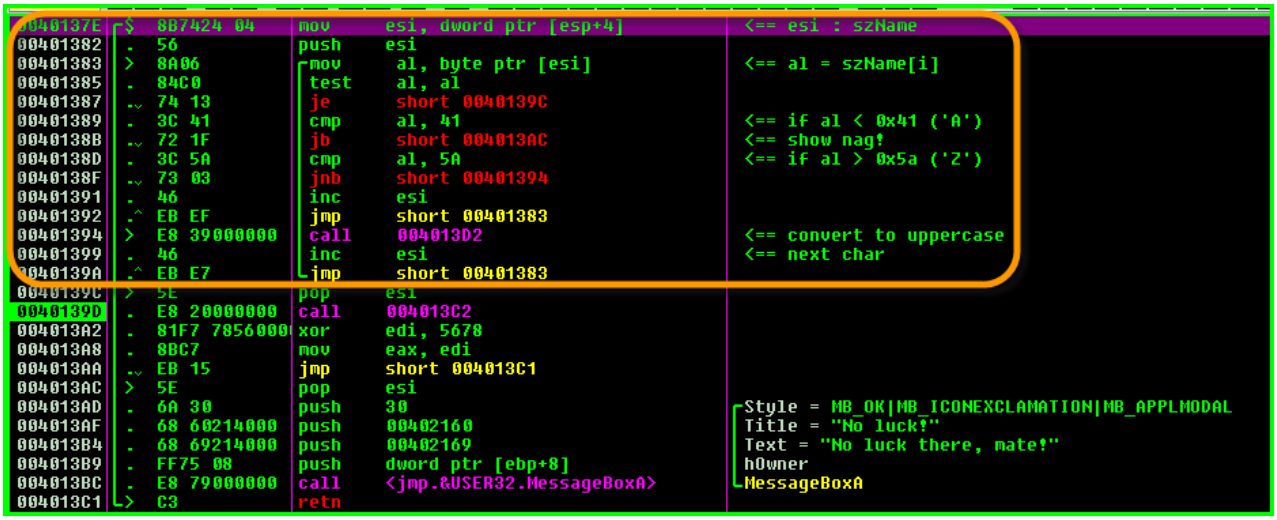
Tại đây ta dừng lại một chút và suy nghĩ đã, quan sát đoạn code :

```
00401223    .   83F8 00        cmp     eax, 0
00401226    .^ 74 BE           je      short 004011E6
00401228    .   68 8E214000    push    0040218E    ;  ASCII "manowar"
0040122D    .   E8 4C010000    call    0040137E (1)
00401232    .   50             push    eax
00401233    .   68 7E214000    push    0040217E    ;  ASCII "1234567890"
00401238    .   E8 9B010000    call    004013D8 (2)
0040123D    .   83C4 04        add     esp, 4
00401240    .   58             pop     eax
00401241    .   3BC3           cmp     eax, ebx (3)
00401243    .   74 07          je      short 0040124C
```

```
00401245    .  E8 18010000    call    00401362
0040124A    .^ EB 9A          jmp     short 004011E6
0040124C    >  E8 FC000000    call    0040134D
00401251    .^ EB 93          jmp     short 004011E6
```

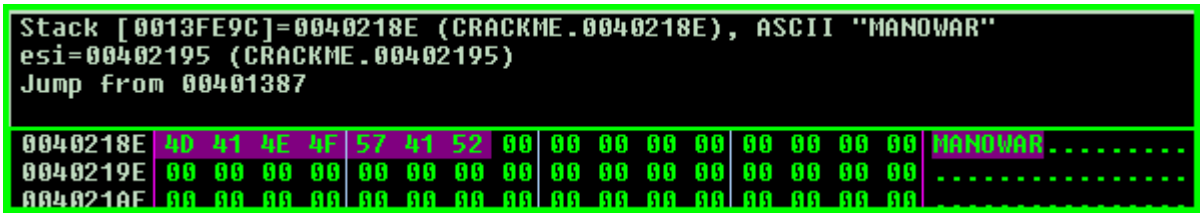
Đầu tiên, để ý đến lệnh **call 0040137E (1)** trước nó là lệnh push chuỗi Name vào Stack. Vậy ta đoán khả năng lệnh call này sẽ dùng chuỗi Name để tính toán gì đó với chuỗi Name. Kết quả tính toán này sẽ được lưu vào thanh ghi eax vì ta thấy thanh ghi eax sau đó được lưu vào Stack và được sử dụng lại trong quá trinh so sánh. Tiếp theo là lệnh **call 004013D8 (2),** trước nó là lệnh push chuỗi Serial vào Stack, khả năng lệnh call này cũng tính toán gì đó với Serial , kết quả sau đó chắc là được lưu vào thanh ghi ebx. Vì tay thấy rằng sau đó thanh ghi eax được đem so sánh với thanh ghi ebx. Phụ thuộc vào kết quả so sánh này sẽ là một lệnh nhảy đưa ta đến Good/Bad boy ☺.

Quá trình suy nghĩ và phân tích sơ bộ đã xong, giờ ta trace vào từng đoạn code để hiểu thêm xem về cơ chế tính toán ra sao. Trước tiên ta làm việc với lệnh **call 0040137E (1)** :



Tôi phân tích từng phần một để các bạn dễ hiểu, đầu tiên là khúc tôi khoanh màu vàng. Nó có nhiệm vụ như sau :

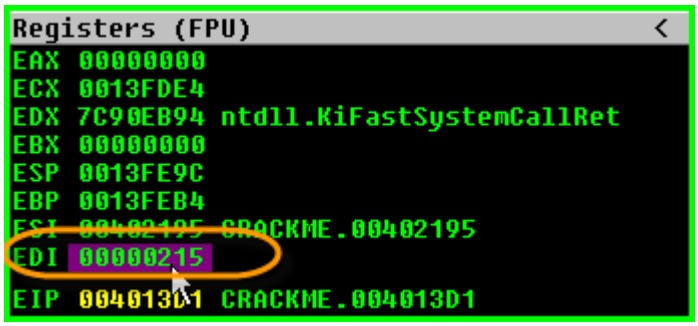
1. Đọc chuỗi Name từ Buffer và lưu vào thanh ghi esi.
2. Kiểm tra từng kí tự trong chuỗi Name xem có nằm trong khoảng từ ‘A’ – ‘Z’ không?
3. Nếu có kí tự nào có mã Ascii < 0x41 (‘A) thì sẽ show Nag (tức là Name không chứa chữ số v..v.)
4. Nếu kí tự nào là chữ hoa thì thôi, nếu không là chữ hoa thực hiện convert sang chữ hoa (`call 004013D2`).
5. Kết quả cuối cùng được lưu lại vào vùng Buffer.



Tiếp theo ta sẽ trace into vào lệnh call tại : `0040139D  |.  E8 20000000  call  004013C2`, mục đích để tìm hiểu xem lệnh call này sẽ tính toán gì tiếp theo với chuỗi buffer.

```
004013C2  $  33FF      xor     edi, edi        <== edi = 0x0
004013C4  .  33DB      xor     ebx, ebx        <== ebx = 0x0
004013C6  >  8A1E      mov     bl, byte ptr [esi]   <== bl = szName[i]
004013C8  .  84DB      test    bl, bl
004013CA  .v 74 05     je      short 004013D1
004013CC  .  03FB      add     edi, ebx        <== edi = edi + ebx
004013CE  .  46        inc     esi             <== next char of szName
004013CF  .^ EB F5     jmp     short 004013C6
004013D1  L> C3        retn
```

Đoạn code này làm nhiệm vụ cộng dồn toàn bộ các kí tự trong chuỗi Name lại vào lưu vào thanh ghi edi, kết quả cuối cùng của thanh ghi edi đối với chuỗi Name mà tôi nhập vào là :

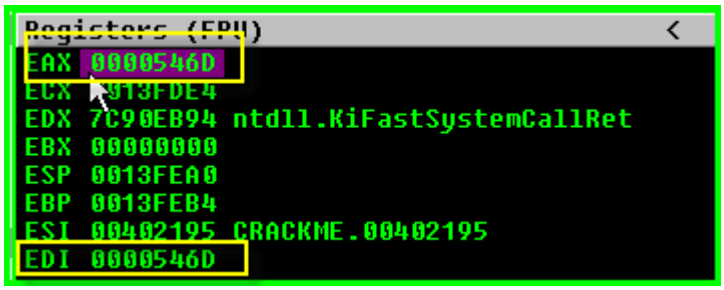


Chuyển giá trị này về dạng decimal xem là bao nhiêu :

Sau khi có được kết quả tại thanh ghi edi ta thực hiện lệnh `004013D1  \> \C3  retn` để trở về, ta tới đây :

```
004013A2  .  81F7 7856000  xor     edi, 5678       <== edi = edi ^ 0x5678
004013A8  .  8BC7          mov     eax, edi        <== eax = edi
004013AA  .v EB 15         jmp     short 004013C1  <== return to main code
004013AC  >  5E            pop     esi
004013AD  .  6A 30         push    30              Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
004013AF  .  68 60214000   push    00402160        Title = "No luck!"
004013B4  .  68 69214000   push    00402169        Text = "No luck there, mate!"
004013B9  .  FF75 08       push    dword ptr [ebp+8]   hOwner
004013BC  .  E8 79000000   call    <jmp.&USER32.MessageBoxA>   MessageBoxA
004013C1  L> C3            retn
```

Tại đoạn code này, giá trị của edi sau khi tính toán được ở trên được đem đi xor với một giá trị mặc định của chương trình là `0x5678`. Kết quả được bao nhiêu sẽ lưu lại vào thanh ghi `eax`. Sau đó quay về main code của chương trình.

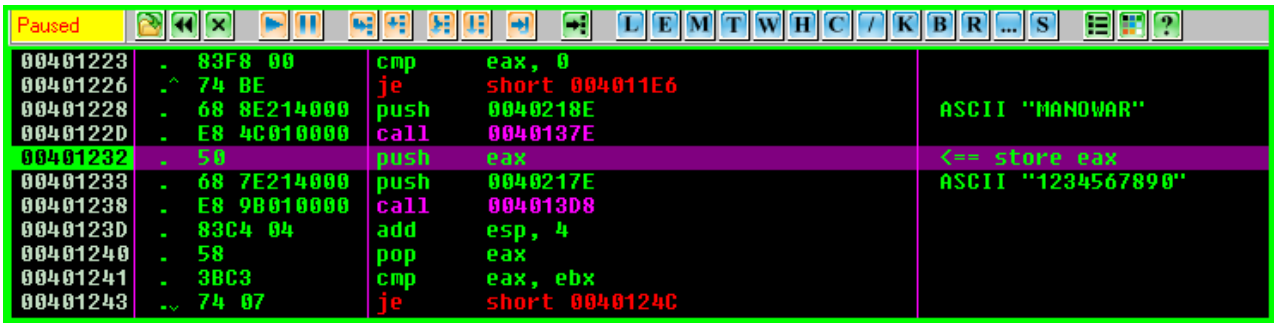


Vậy tổng kết lại ta có được quá trình tính toán liên quan tới chuỗi Name như sau :

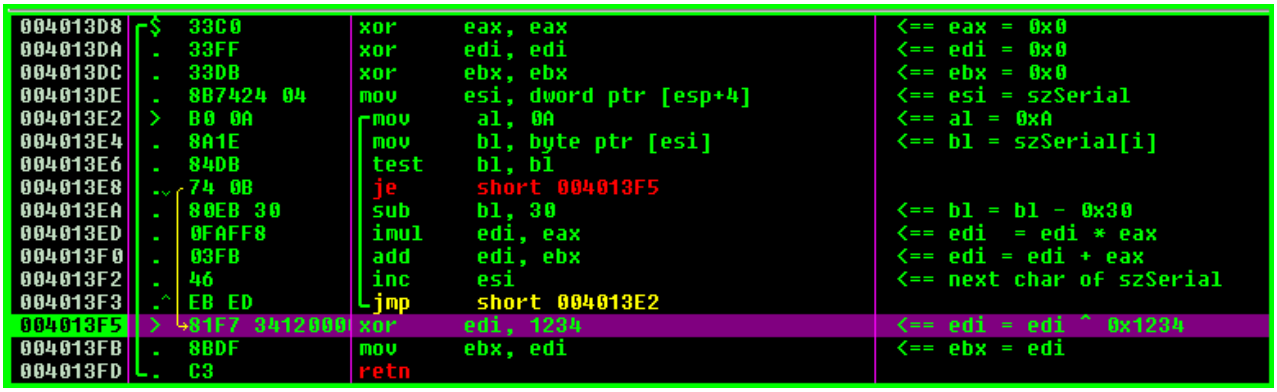
1) Chuỗi Name nhập vào phải là các chữ cái trong khoảng ‘A’-‘Z’, ‘a’-‘z’.
2) Toàn bộ chuỗi sau đó sẽ được convert thành chữ in hoa.

3) Sau khi được convert, đem các kí tự trong chuỗi cộng dồn lại ở dạng hexa, và lưu vào edi.
4) Thanh ghi edi tiếp tục được xor với giá trị mặc định là `0x5678`.
5) Cuối cùng được kết quả bao nhiêu sẽ lưu vào eax.

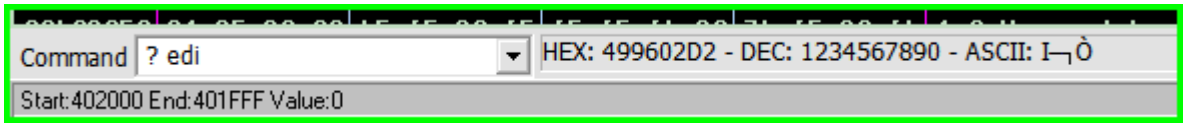
Sau khi trở về main code của crackme ta ở đây :



Nhìn vào hình trên ta thấy, thanh ghi eax lưu kết quả của quá trình tính toán liên quan đến chuỗi Name sẽ được cất tạm vào Stack. Tiếp theo chương trình sẽ đẩy chuỗi Serial lên Stack và tính toán gì đó với chuỗi này. Việc tiếp theo ta cần làm là tìm hiểu xem nó tính toán gì tại : **`call 004013D8 (2)`**. Trace into vào lệnh call này ta tới đây :



Ngay đầu tiên ta đã thấy thanh ghi eax bị clear, chính vì thế các bạn thấy tác giả đã lưu lại thanh ghi eax trước. Tổng thể toàn bộ đoạn code trên làm nhiệm vụ : chuyển chuỗi Serial về dạng hexa. Trong ví dụ của tôi nhập vào là *1234567890*, qua đoạn code trên nó sẽ được convert về thành giá trị là 0x*499602D2* và lưu vào thanh ghi edi :



Giá trị tại thanh ghi edi sau đó lại được đem xor với một giá trị mặc định khác là `0x1234` (hehe ở trên thì là `0x5678`). Kết quả được bao nhiêu sẽ được lưu lại tại thanh ghi ebx. Sau đó trở về main code để tới quá trình so sánh tại :

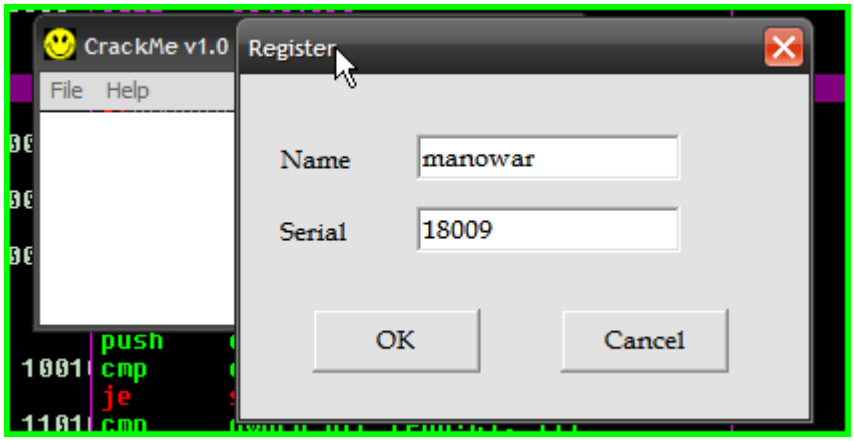
```
00401241    .   3BC3            cmp     eax, ebx (3)
00401243    .   74 07           je      short 0040124C
```

Vậy là ta đã tìm hiểu xong phần tính toán liên quan đến chuỗi Name và Serial. Bây giờ ta cần phải suy nghĩ và lập luận để tìm ra real serial cho chuỗi Name của chúng ta. Như các bạn thấy, chuỗi Name của ta nhập vào được chuyển sang chữ hoa, sau đó cộng dồn, cuối cùng đem xor với `0x5678` để cho ra kết quả và lưu vào eax. Trong trường hợp của tôi giá trị sau khi tính toán là `0x0000546D`,

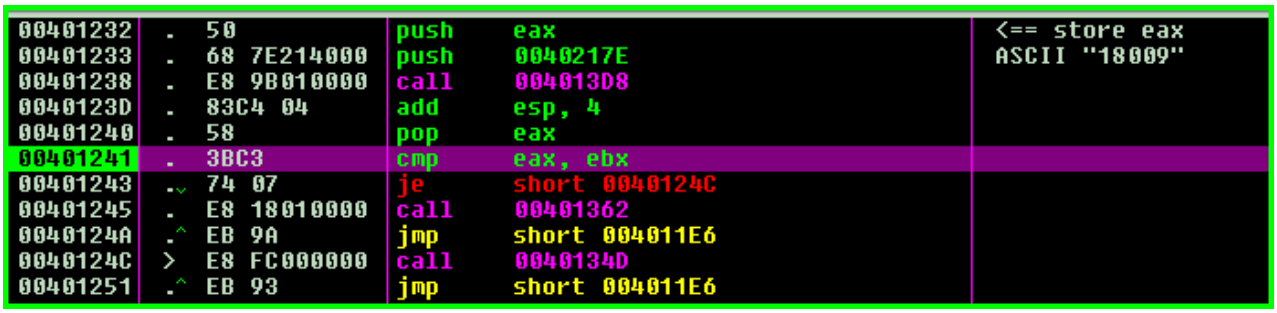
chuyển giá trị này sang dạng decimal tôi có **21613**. Tiếp theo tôi thấy rằng chuỗi Serial mà tôi nhập vào cũng được convert thành dạng hexa (Giả sử nếu tôi nhập vào là 21613, thì tức là chuyển giá trị 21613 về dạng hexa là 0x546D). Sau đó giá trị hexa này được đem đi xor với giá trị mặc định là 0x1234. Mà tôi thì biết rằng lệnh XOR có ý nghĩa như sau :

Lệnh XOR có thể được dùng để đảo các bit xác định của toán hạng đích trong khi vẫn giữ nguyên những bit còn lại. Bit 1 của mặt nạ làm đảo bit tương ứng còn bit 0 giữ nguyên bit tương ứng của toán hạng đích.

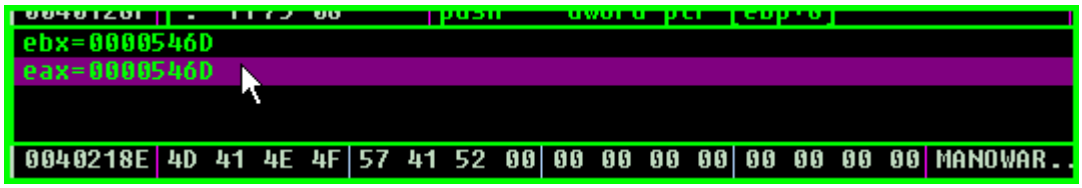
Vậy từ đó ta kết luận rằng Serial của chúng ta sẽ là giá trị tính toán được của chuỗi Name và đem xor với 0x1234. Trong trường hợp cụ thể của tôi thì real serial sẽ là : **0x546D xor 0x1234 = 0x4659** (chuyển sang decimal là : **18009**). Thử kiểm chứng lại xem có đúng không nhé, tôi nhấn F9 để run chương trình. Nhập Name và Serial như sau :



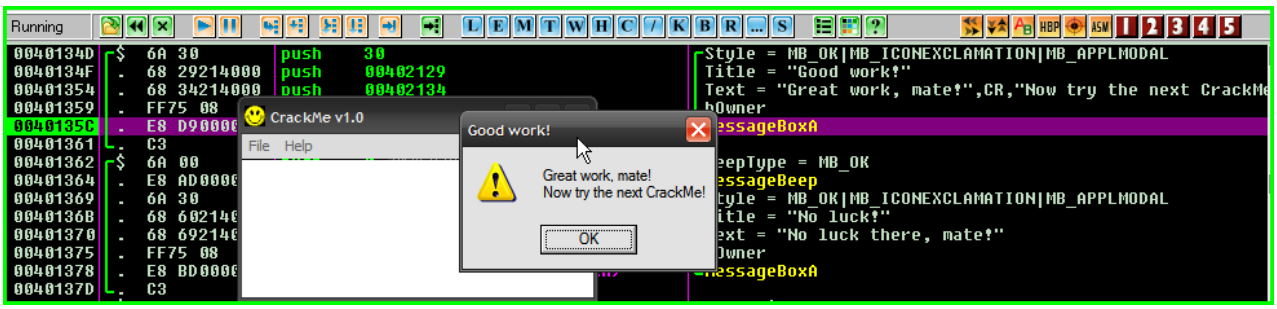
Nhấn Ok và trace tới đoạn code :



Ta dừng lại tại đoạn so sánh, để ý cửa sổ Tip Window ta sẽ thấy ☺ :



Khà khà quá chuẩn rồi, nhấn F9 để thực thi chương trình các bạn sẽ nhận được good boy :



Như vậy các bạn đã thấy tầm quan trọng của việc phân tích và đọc hiểu code của chương trình sẽ giúp chúng ta rất nhiều. Bên cạnh đó tính kiên nhẫn trong việc trace code cũng không thể thiếu được khi ta làm việc với Olly. Sau khi phân tích được thuật toán của Crackme như trên, bạn hoàn toàn có thể code một chương trình nhỏ làm nhiệm vụ Genkey từ chuỗi Name nhập vào, chương trình đó người ta gọi là keygen hay keymaker ☺. Công việc đó xin nhường lại cho các bạn tự mình khám phá tiếp nhé, còn tôi thì mệt rồi!!

Ok vậy là phần 13 của loạt tuts về Ollydbg đến đây là kết thúc, qua bài viết này tôi đã hướng dẫn các bạn cách kiểm tra xem file có bị pack hay không, cách tìm các điểm quan trọng để tiếp cận mục tiêu, phân tích chi tiết hoạt động của Crackme CrueHead thông qua việc trace và analyze code để từ đó tìm ra một real serial cho chuỗi Name nhập vào. Hi vọng qua bài viết này tôi đã truyền tải tới các bạn những kinh nghiệm thực tế khi làm việc với một crackme đơn giản nhưng cũng sẽ là tiền đề cho các bạn khi gặp các crackme hoặc các chương trình khác. Hẹn gặp lại các bạn trong các phần tiếp theo của loạt tutor này, By3 By3!! ☺

Best Regards
[Kienmanowar]

--++--==[Greatz Thanks To]==--++--
My family, Computer_Angel, Moonbaby , Zombie_Deathman, Littleboy, Benina, QHQCrker, the_Lighthouse, Merc, Hoadongnoi, Nini ... all REA's members, TQN, HacNho, RongChauA, Deux, tlandn, light.phoenix, dump, dqtln, ARTEAM .... all my friend, and YOU.
--++--==[Thanks To]==--++--
iamidiot, WhyNotBar, trickyboy, dzungltvn, takada, hurt_heart, haule_nth, hytkl, moth, XIANUA, nhc1987, 0xdie, Unregistered!, akira, mranglex v..v.. các bạn đã đóng góp rất nhiều cho REA. Hi vọng các bạn sẽ tiếp tục phát huy ☺

I want to thank **Teddy Roggers** for his great site, Reversing.be folks(especially **haggar**), Arteam folks(**Shub-Nigurrath, MaDMAn_H3rCuL3s**) and all folks on crackmes.de, thank to all members of **unpack.cn** (especially **fly** and **linhanshi**). Great thanks to **lena151**(I like your tutorials).Thanx to Orthodox, kanxue, TiGa and finally, thanks to **RICARDO NARVAJA** and all members on **CRACKSLATINOS.**

>>>> If you have any suggestions, comments or corrections email me:
kienmanowar[at]reaonline.net