2009

[Cracking with OllyDbg]

Based on OllyDbg tuts of Ricardo Narvaja (CrackLatinos Team)

www.reaonline.net

kienmanowar

12/24/2009

Mục Lục

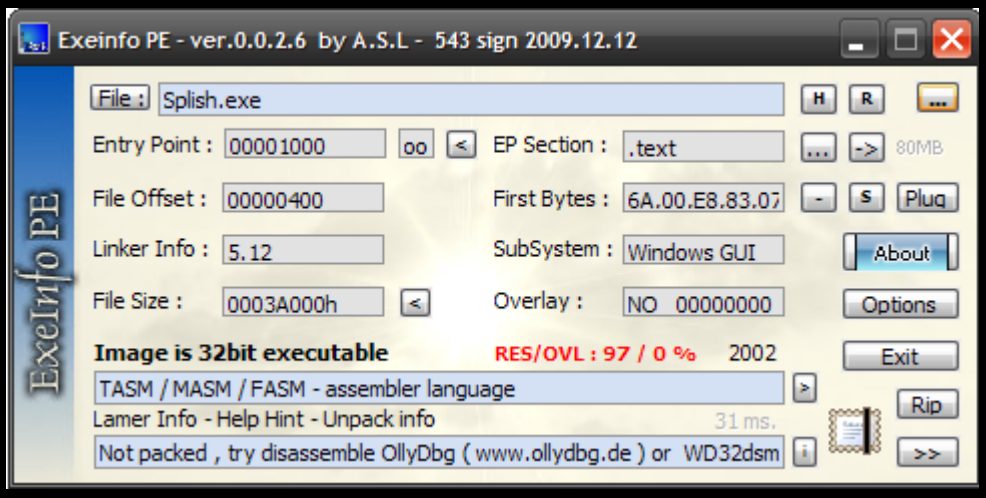
I. Giới thiệu chung

Chào anh em, kể từ ngày mất laptop bao nhiêu dữ liệu quý hiếm mất sạch, thêm nữa tôi bận lung tung thứ việc nên bộ **"OllyDbg tutorial"** này đã bị tạm hoãn trong một thời gian khá dài. Tôi nhận được rất nhiều câu hỏi đặt ra liên quan tới bộ tuts này, hầu như xoay quanh việc mong muốn tôi viết tiếp, thú thực là nhiều lúc thấy nản muốn bò béng cho rồi ☺. Đợt này nhân dịp Noel, bã xã mới sinh hạ đại ca và một năm mới sắp đến nên quyết định đặt bút tiếp tục bộ tutor dài kì này. Ngoài mục đích là refresh lại chính mình, quay trở lại nghiên cứu những gì mà tôi đã từng yêu thích cũng nhưng hi vọng rằng sau này đại ca của tôi sẽ có hứng thú với những gì mà tôi đã và đang viết ở đây ☺. Lục lọi và đọc lại 15 bài trước, thấy còn nhiều thiếu sót và có thể chưa đáp ứng được hết những mong muốn, yêu cầu của các bạn, hehe sức người có hạn…tôi biết gì thì viết thế thôi, không dám ôm đồm nhiều thứ.

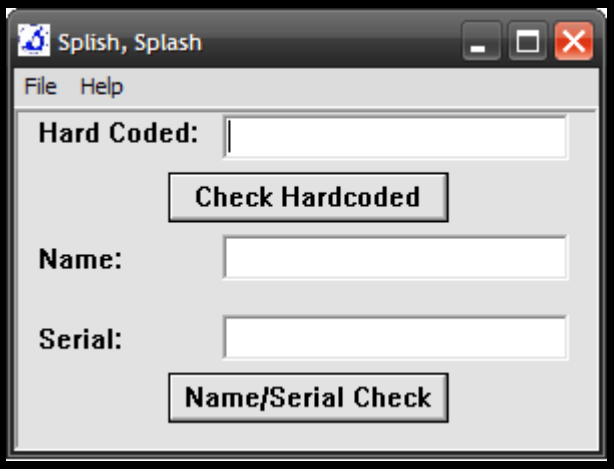
Ở bài viết thứ 16 này, như tôi đã nói ở trên đó là refresh lại chính mình và là bàn đạp hứng thú để tôi tiếp tục viết tiếp, cho nên tôi sẽ viết rất đơn giản, ngắn gọn và thực hiện demo trên target là Splish.exe mà tôi đã đính kèm trong bài 15. Nếu bạn nào giải quyết được nó rồi (và tôi tin là các bạn làm được) thì có thể bỏ qua bài viết này và chờ đợi những điều mới hơn ở bài 17. N0w let's g0……

II. Phân tích và xử lý target

Trước tiên, như thường lệ khi ta làm việc với bất kì target nào cũng cần kiểm tra sơ bộ thông tin về target đó trước. Ở đây tôi sử dụng chương trình ExeInfo của A.S.L :



Theo thông tin mà Exeinfo check được, ta thấy target này không bị pack, khả năng được code bằng MASM32. Ta chạy thử xem mặt mũi nó thế nào :

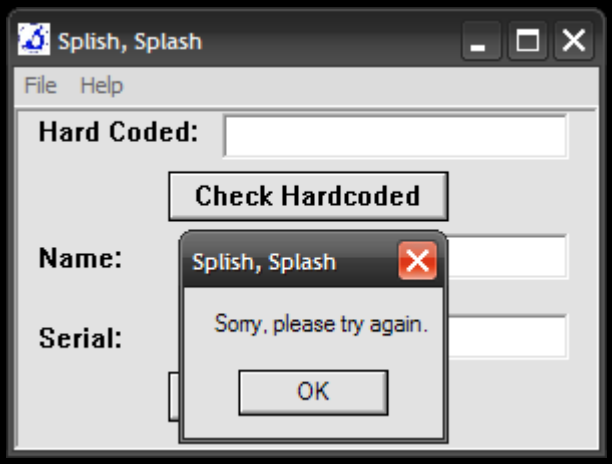


Chà như vậy ở đây chúng ta cần giải quyết hai mục tiêu :

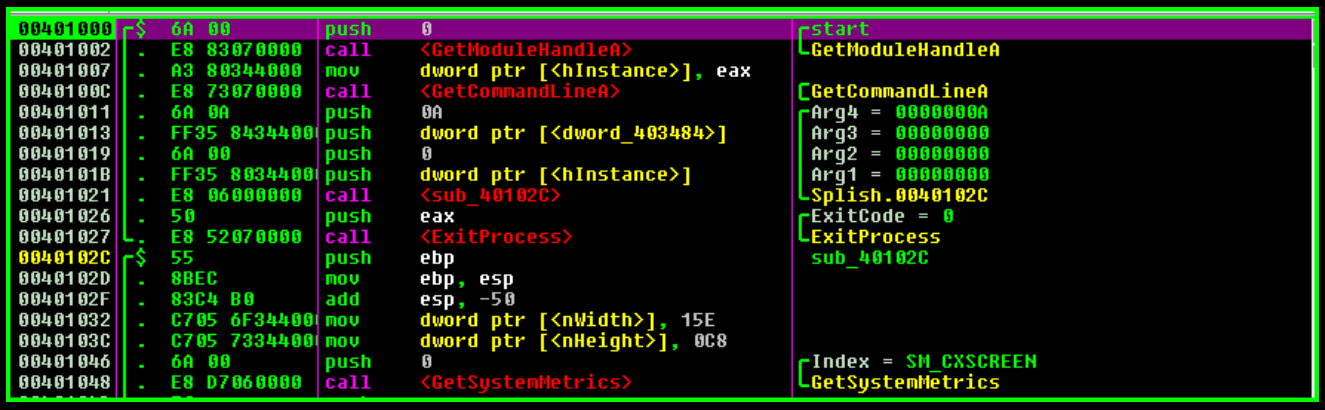
a. Tìm Hardcoded serial.
b. Tìm Valid serial dựa trên việc tính toán Username hoặc là một cách tính nào đó mà ta chưa biết.

1. Tìm Hardcoded Serial

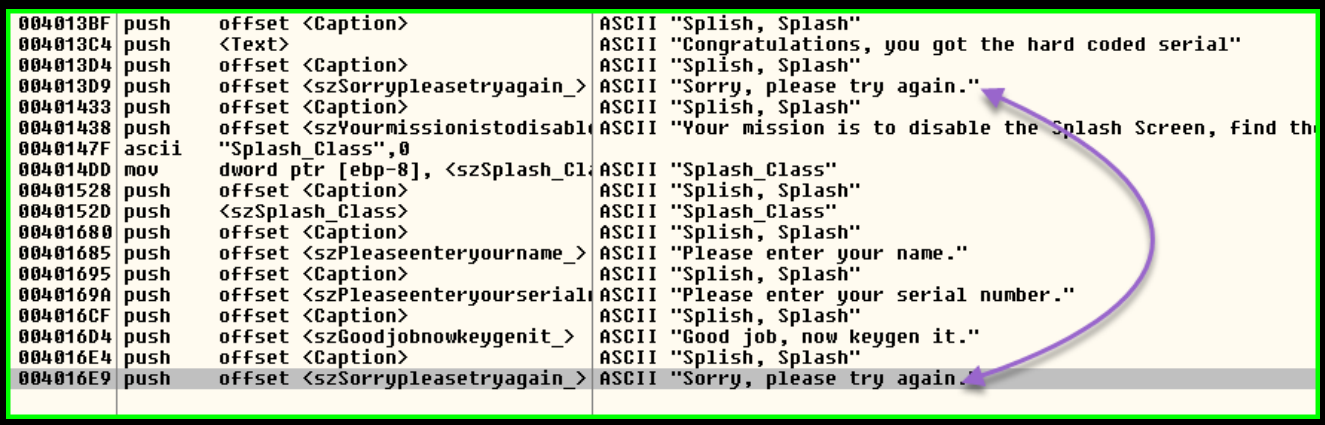
Để tìm Hardcoded serial, ta bấm thử vào nút **Check Hardcoded** để xem điều gì diễn ra, từ đó có thông tin mà lần theo :



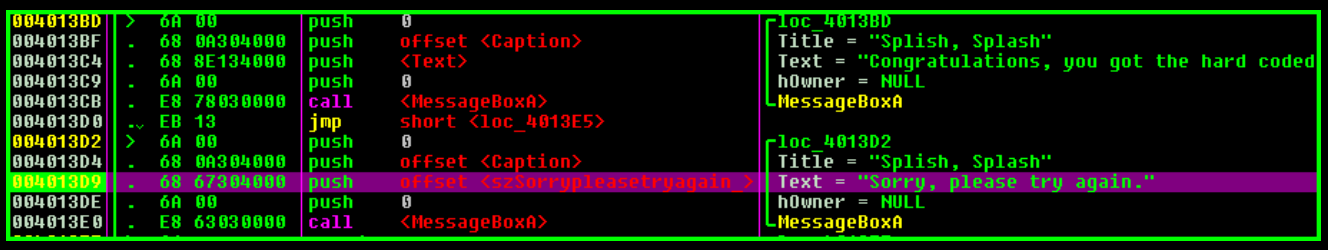
Như vậy là ta nhận được thông báo : *"Sorry, please try again.".* OK, tạm thời có chút manh mối, mở Olly lên và load target vào :



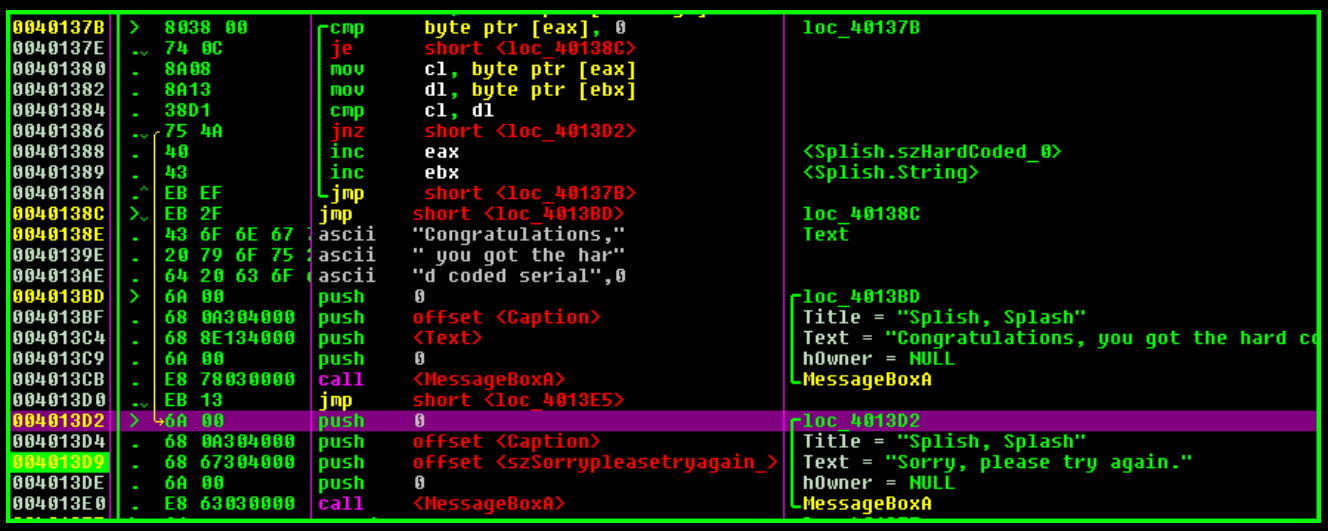
Sau khi load vào Olly, ta dừng lại tại EP của target. Chuột phải chọn **Search for > All referenced text strings**, ta tìm kiếm chuỗi trên. Ở đây ta thấy hai thông báo xuất hiện ở hai địa chỉ khác nhau :



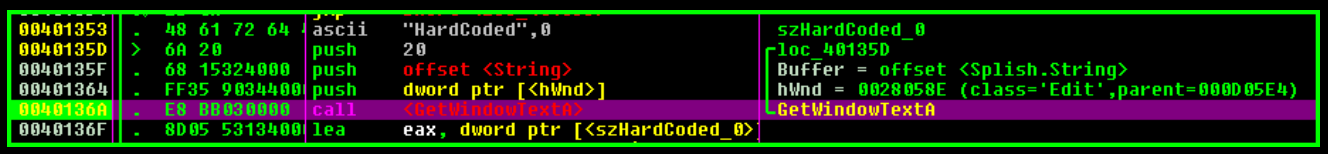
Ta có thể đoán được một cái sẽ dùng cho phần check Hardcoded Serial, cái kia sẽ dùng cho phần liên quan tới việc check Username và Serial. Để biết chính xác thông báo nào sẽ xuất hiện khi ta nhấn nút Check Hardcoded ta đặt BP tại hai địa chỉ liên quan tới chuỗi trên. Sau khi đặt BP xong, nhấn F9 để thực thi chương trình và nhấn vào nút Check Hardcoded, ta sẽ dừng tại đây :



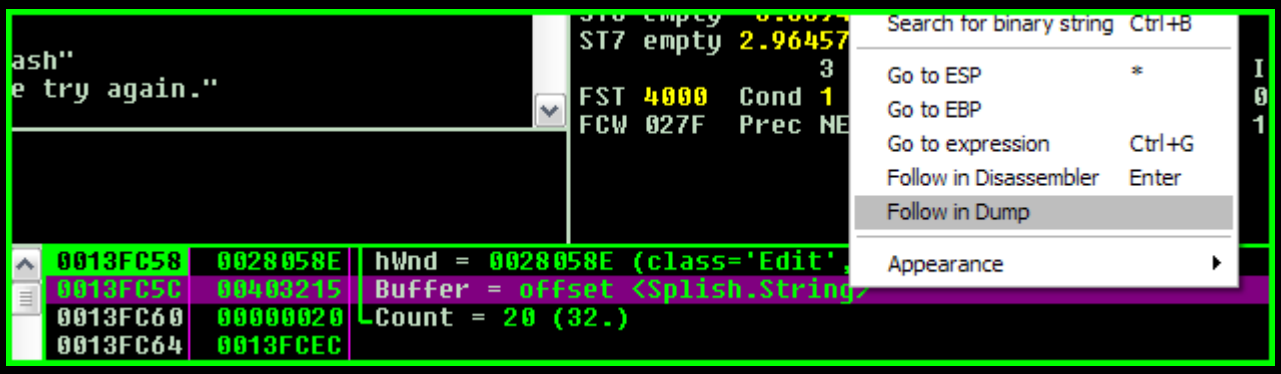
Đó chính là thông tin mà ta cần tìm, dịch lên một chút tại đoạn code `004013D2 >|> \6A 00 push 0 ; /loc_4013D2` sẽ cho ta biết lệnh nhảy nào sẽ nhảy tới nó :



Để ý ở trên lệnh nhảy tại `00401386 |. /75 4A |jnz short <loc_4013D2>` ta thấy có các lệnh so sánh. Có thể kết luận đó là những đoạn code liên quan tới việc so sánh sẽ Serial mà ta nhập vào có trùng với Hardcoded Serial của target hay không. Việc ta cần làm là tìm ra cái Hardcoded Serial đó là gì, ta tiến hành đặt BP tại `0040136A |. E8 BB030000 call <GetWindowTextA> ; \GetWindowTextA` . Nhấn F9 để run chương trình và nhập đại một Serial vào và nhấn nút Check, Olly sẽ break tại địa chỉ mà ta đặt BP :



Follow in Dump tại địa chỉ Buffer, sau đó nhấn F8 để trace qua lệnh gọi tới hàm GetWindowTextA ta sẽ có được chuỗi fake serial mà ta nhập vào được lưu tại vùng buffer như sau :



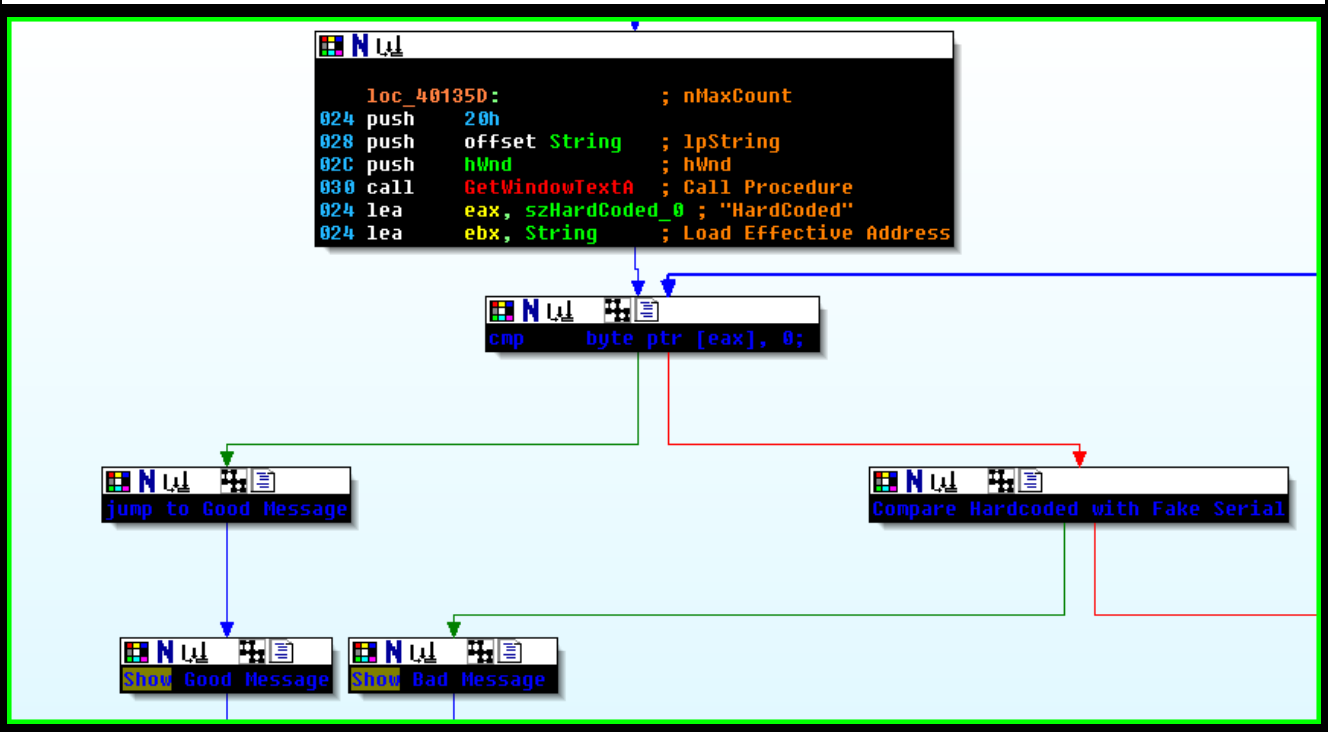
Sau khi trace qua ta sẽ dừng lại ở lệnh kế tiếp :

Info :

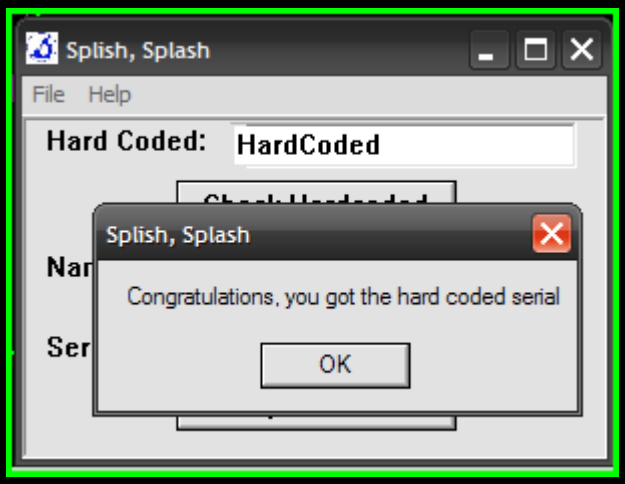
```
LEA - Load Effective Address
        Usage:  LEA     dest,src
        Modifies flags: None
        Transfers offset address of "src" to the destination register.
```

Như vậy ta biết được lệnh đầu tiên sẽ là lệnh truyền địa chỉ chứa chuỗi Hardcoded Serial vào thanh ghi eax, lệnh thứ hai là truyền địa chỉ chứa chuỗi fake serial (chính là địa chỉ buffer ở trên) vào thanh ghi ebx. Sau đó hai chuỗi này sẽ được so sánh với nhau. Ta đang dừng lại ở lệnh : `0040136F  |.  8D05 53134000 lea     eax, dword ptr [<szHardCoded_0>]` , nhìn xuống cửa sổ Tip Window ta sẽ có được thông tin về địa chỉ chứa chuỗi Hardcoded và chuỗi Hardcoded serial mà ta cần tìm :

Wow quá rõ ràng, chuỗi mà ta cần tìm chính là : **"HardCoded"**. Như vậy nhiệm vụ đi tìm chuỗi Hardcoded serial của chúng ta đã hoàn thành, ta không cần quan tâm tới các lệnh so sánh ở bên dưới làm gì nữa. Nếu chúng ta dùng IDA để xem code ta sẽ thấy như sau :

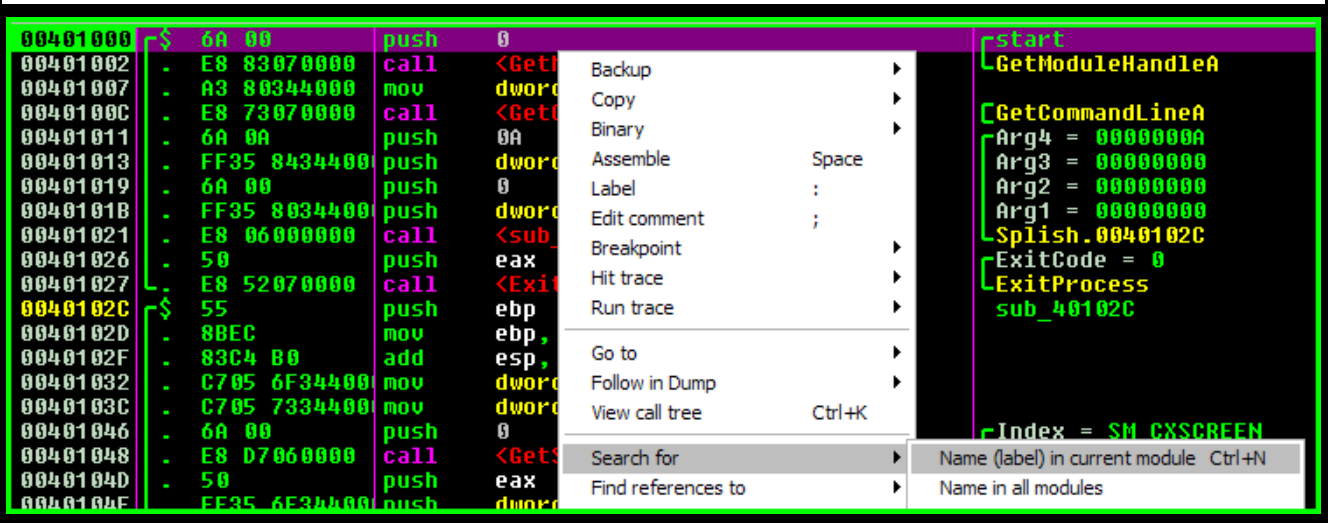


Tiến hành kiểm tra xem chuỗi ta tìm được có chính xác hay không :

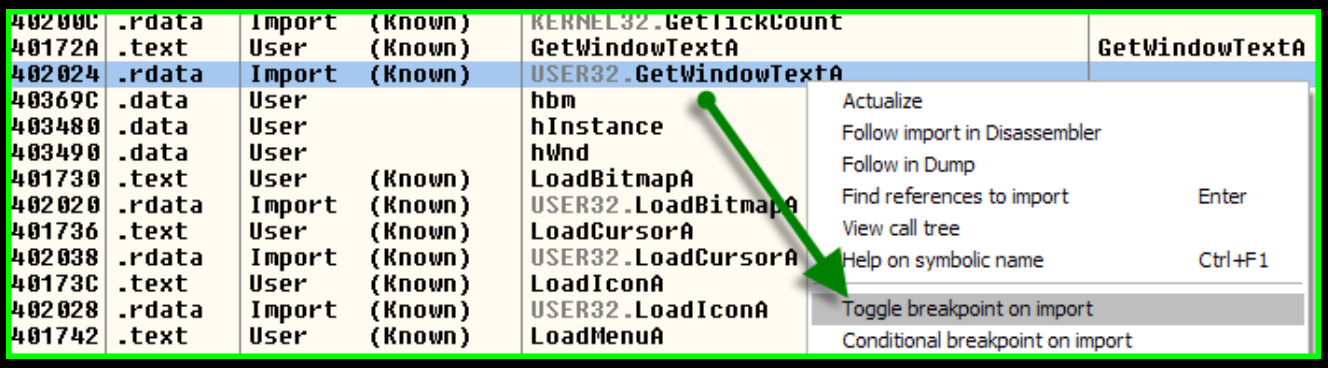


2. *Tìm Serial dựa trên việc tính toán Name nhập vào*

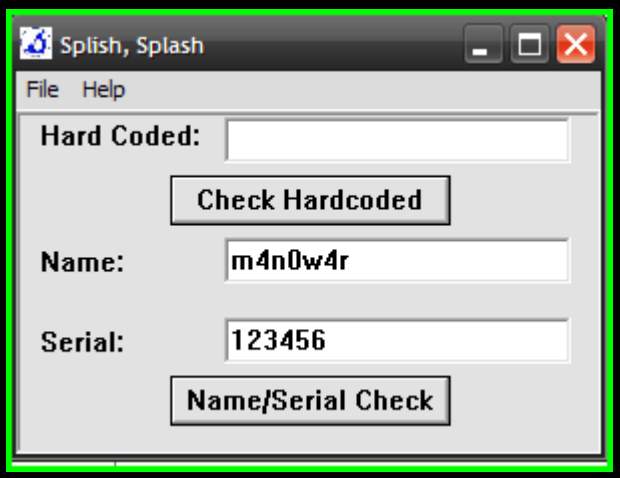
Phần này ta sẽ giải quyết nốt nhiệm vụ còn lại. Các bạn còn nhớ địa chỉ của chuỗi *"Sorry, please try again."* thứ hai mà ta đã tìm được ở trên chứ? Các bạn có thể theo đó để tìm ra đoạn code liên quan tới việc tính toán và so sánh. Để cho bài viết hấp dẫn hơn tôi chọn cách khác để tìm ra mục tiêu chính. Giờ bạn bỏ hết các BP đã đặt ở trên đi, nhấn Ctrl+F2 để restart lại. Sau đó nhấn chuột phải và chọn như sau :



Tìm tới hàm GetWindowTextA và đặt BP tại hàm này :



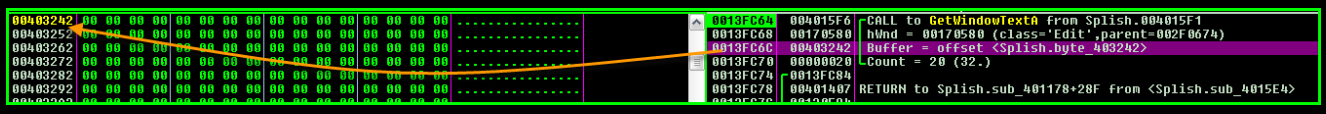
Nhấn F9 để thực thi chương trình, nhập đại Name và Serial vào :



Sau đó nhấn nút **"Name/Serial Check",** Olly ngay lập tức sẽ break tại địa chỉ mà ta vừa đặt BP (Note: địa chỉ các hàm APIs trên máy của tôi có thể khác với máy các bạn):

```
77D62144   6A 0C           push    0C
77D62146   68 A821D677      push    77D621A8
77D6214B   E8 8064FEFF      call    77D485D0
77D62150   8B7D 0C          mov     edi, dword ptr [ebp+C]
77D62153   33DB             xor     ebx, ebx
77D62155   3BFB             cmp     edi, ebx
77D62157   0F84 388F0000    je      77D6B095
77D6215D   395D 10          cmp     dword ptr [ebp+10], ebx
77D62160   0F84 2F8F0000    je      77D6B095
```

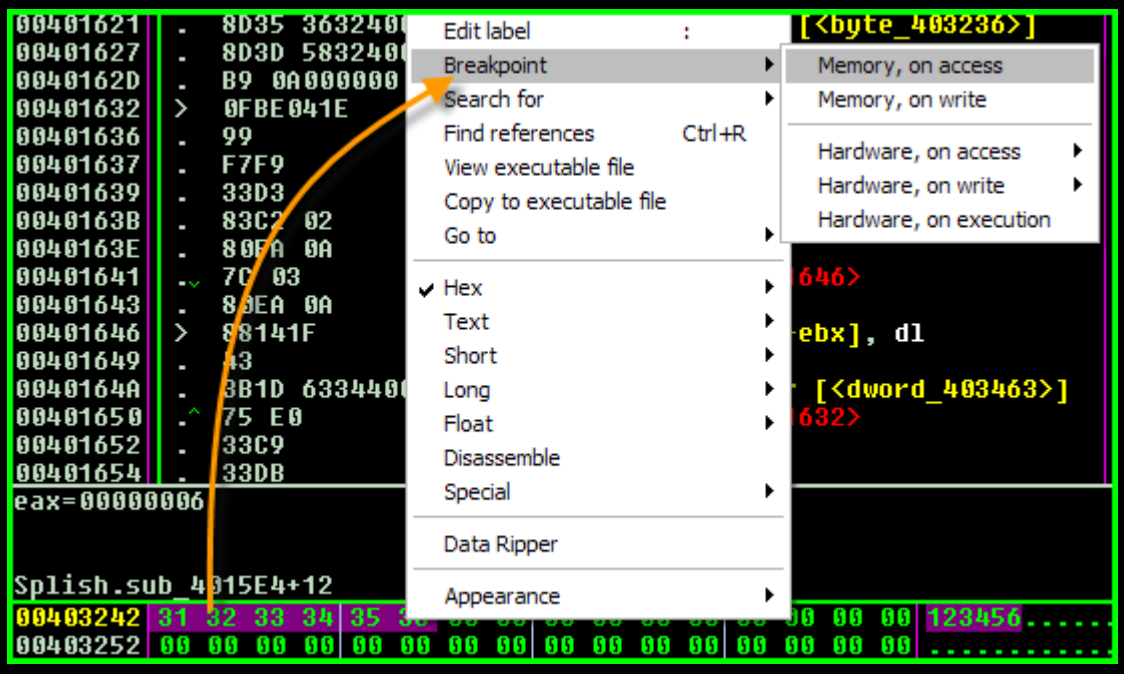
Quan sát vùng Buffer tại cửa sổ Stack và *Follow in Dump* tại vùng Buffer này :



Nhấn **Alt+F9** để trở về code chính của chương trình, đồng thời quan sát vùng Buffer ta sẽ nhận được thông tin về Serial đã nhập vào :

```
Splish.sub_4015E4+12
00403242  31 32 33 34 35 36 00 00  00 00 00 00 00 00 00 00  123456..........
00403252  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00403262  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00403272  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00403282  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00403292  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
004032A2  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Vậy là lời gọi hàm GetWindowTextA đầu tiên lại nhận chuỗi Serial mà ta nhập vào trước chứ không phải là chuỗi Name. Tại vùng Buffer chứa chuỗi Fake Serial ở trên, ta đánh dấu chuỗi này và chọn :

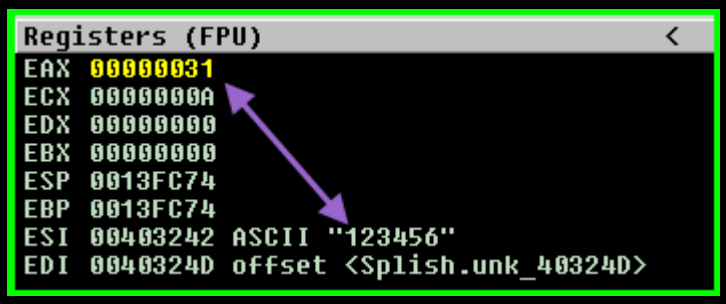


Mục đích của việc này là ta muốn nhanh chóng tìm ra đoạn code sẽ sử dụng tới chuỗi Fake Serial.Nhấn F9 để run chương trình, ta sẽ lại dừng lại tại GetWindowTextA .. lần

này là nhận chuỗi Name nhập vào và đặt vào vùng Buffer nào đó. Ở đây ta không cần quan tâm Name lưu ở đâu, tiếp tục nhấn **F9** để run chương trình ta sẽ dừng lại tại đây :

```
00401656  .  33D2          xor     edx, edx
00401658  .  8D35 4232400( lea     esi, dword ptr [<byte_403242>]
0040165E  .  8D3D 4D32400( lea     edi, dword ptr [<unk_40324D>]
00401664  .  B9 0A000000   mov     ecx, 0A
00401669  >  0FBE041E      movsx   eax, byte ptr [esi+ebx]          loc_401669
0040166D  .  99            cdq
0040166E  .  F7F9          idiv    ecx
00401670  .  88141F        mov     byte ptr [edi+ebx], dl
00401673  .  43            inc     ebx
00401674  .  3B1D 6734400( cmp     ebx, dword ptr [<dword_403467>]
0040167A  .^ 75 ED         jnz     short <loc_401669>
0040167C  .v EB 2A         jmp     short <loc_4016A8>
0040167E  >  6A 00         push    0                                loc_40167E
00401680  .  68 0A304000   push    offset <Caption>                 Title = "Splish, Splash"
00401685  .  68 A0304000   push    offset <szPleaseenteryourname_>  Text = "Please enter your name."
0040168A  .  6A 00         push    0                                hOwner = NULL
0040168C  .  E8 B7000000   call    <MessageBoxA>                    MessageBoxA
```

Phân tích đoạn code trên, đầu tiên là lệnh movsx sẽ chuyển byte đầu tiên (số đầu tiên) của chuỗi Fake Serial vào thanh ghi eax. Nhấn **F8** để trace qua lệnh này ta sẽ nhận được thông tin như sau :



Lệnh tiếp theo là lệnh :

```
0040166D  |.  99            |cdq
0040166E  |.  F7F9          |idiv    ecx
```

Tìm kiếm chút thông tin về lệnh CDQ xem nó làm gì ☺ :

```
CDQ - Convert Double to Quad (386+)
        Usage:  CDQ
        Modifies flags: None
        Converts signed DWORD in EAX to a signed quad word in EDX:EAX by
        extending the high order bit of EAX throughout EDX
```

The cdq instruction sign extends the 32 bit value in eax to 64 bits and places the result in edx:eax by copying bit 31 of eax throughout bits 0..31 of edx. This instruction is available only on the 80386 and later. You would normally use this instruction before a long division operation

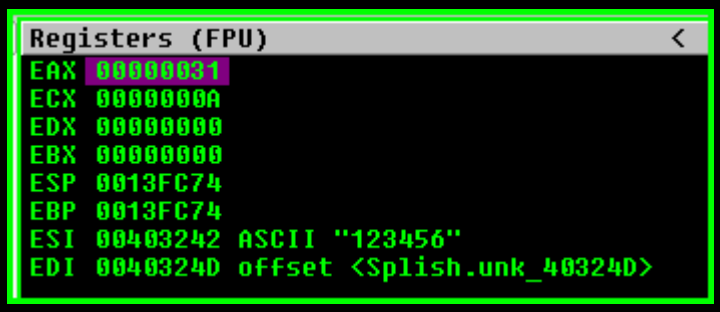
Chốt lại lệnh CDQ làm nhiệm vụ copy bit dấu của EAX (bit 31) vào mọi bit của thanh ghi EDX. Nếu bit 31 = 0 thì EDX có giá trị là 0x00000000, nếu bit 31 = 1 thì EDX sẽ là 0xFFFFFFFF. Ở đây ta thấy lệnh movsx mỗi lần một giá trị vào EAX cho nên giá trị bit dấu luôn là 0, và do đó thanh ghi EDX luôn có giá trị là 0x00000000.

Lệnh tiếp theo là idiv :

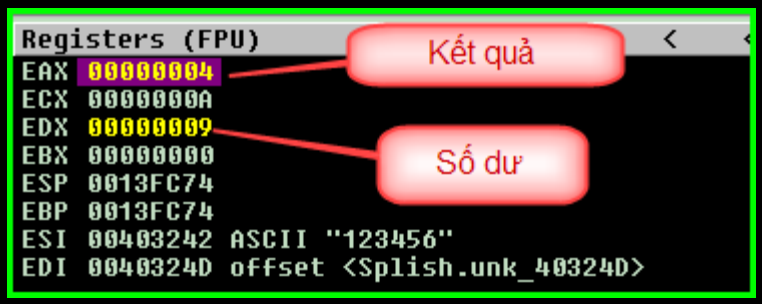
```
idiv — Integer Division

    The idiv instruction divides the contents of the 64 bit integer
EDX:EAX (constructed by viewing EDX as the most significant four bytes and
EAX as the least significant four bytes) by the specified operand value.
The quotient result of the division is stored into EAX, while the
remainder is placed in EDX.
```

Lệnh idiv là lệnh chia số có dấu, nó thực hiện chia số 64 bit trong thanh ghi EDX:EAX cho toán hạng nguồn (ở đây là thanh ghi ECX). Kết quả phép chia sẽ lưu vào EAX, còn số dữ sẽ lưu vào EDX.

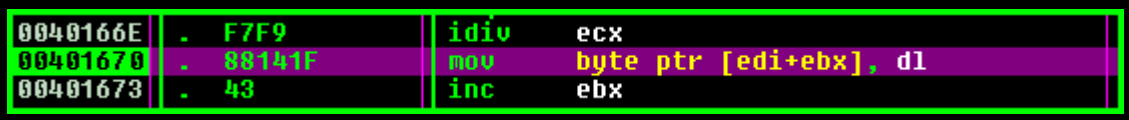
Phần phân tích sơ bộ thế là đủ rồi, quay trở lại phần chính. Hiện tại trên cửa sổ thanh ghi của tôi có những thông tin như sau (sau khi tôi thực hiện lệnh CDQ) :



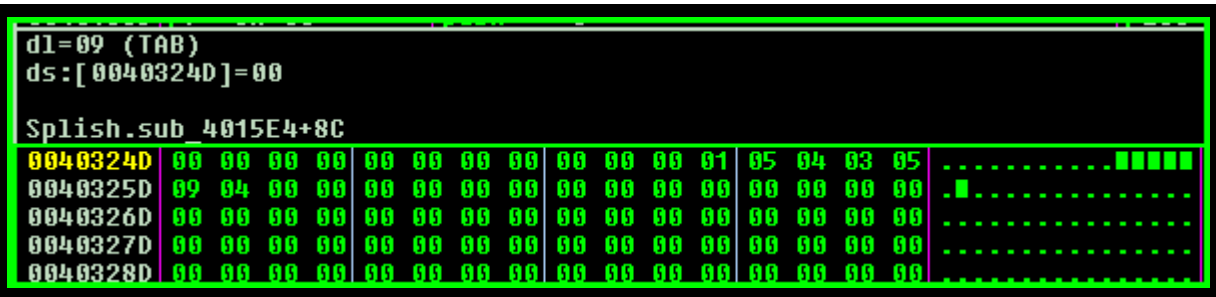
Thực hiện phép chia và quan sát kết quả thủ được :



Thanh ghi EAX lưu kết quả của phép chia là 0x4, EDX lưu phần dư là 0x9. Tiếp theo phần dư này sẽ được lưu vào một vùng nhớ khác :



Xem thêm thông tin tại cửa sổ Tip và Dump :



Nhấn F8 để trace qua lệnh mov, quan sát kết quả ta thu được như sau :

Đoạn code tiếp theo :

```
00401673  |.  43           |inc     ebx ; ebx++
00401674  |.  3B1D 67344000 |cmp     ebx, dword ptr [<dword_403467>]; while
(ebx < 6)
0040167A  |.^ 75 ED        \jnz     short <loc_401669> ; then continue loop
```

Thực chất là một vòng lặp, trong đó ebx là biến đếm. Sau mỗi lần tính toán ebx được tăng lên 1, sau đó so sánh với số lần lặp là 6.

Mọi quá trình tính toán cho tới khi kết thúc vòng lặp các bạn từ mình thực hiện tiếp. Sau khi kết thúc quá trình tính toàn tôi thu được kết quả của toàn bộ phần dư được lưu trong vùng nhờ (tôi đặt là Remain_buf) như sau :

```
0040167A  .^ 75 ED        jnz     short <loc_401669>          <== then continue loop
0040167C  .v EB 2A        jmp     short <loc_4016A8>
0040167E  > 6A 00         push    0                           loc_40167E
00401680  . 68 0A304000   push    offset <Caption>            Title = "Splish, Splash"
00401685  . 68 A0304000   push    offset <szPleaseenteryourname_>  Text = "Please enter your name."
0040168A  . 6A 00         push    0                           hOwner = NULL
0040168C  . E8 B7000000   call    <MessageBoxA>               MessageBoxA
00401691  .v EB 62        jmp     short <locret_4016F5>
00401693  > 6A 00         push    0                           loc_401693
00401695  . 68 0A304000   push    offset <Caption>            Title = "Splish, Splash"
0040169A  . 68 B8304000   push    offset <szPleaseenteryourserialnum  Text = "Please enter your serial number."
0040169F  . 6A 00         push    0                           hOwner = NULL
004016A1  . E8 A2000000   call    <MessageBoxA>               MessageBoxA
004016A6  .v EB 4D        jmp     short <locret_4016F5>
004016A8  > 8D35 4D32400( lea     esi, dword ptr [<Remain_Buf>]   loc_4016A8
004016AE  . 8D3D 5832400( lea     edi, dword ptr [<unk_403258>]
004016B4  . 33DB          xor     ebx, ebx
```

Chúng ta thực hiện lệnh JMP và bắt đầu đoạn code mới :

```
004016A8  > 8D35 4D32400( lea     esi, dword ptr [<Remain_Buf>]   loc_4016A8
004016AE  . 8D3D 5832400( lea     edi, dword ptr [<unk_403258>]
004016B4  . 33DB          xor     ebx, ebx
004016B6  > 3B1D 6334400( cmp     ebx, dword ptr [<dword_403463>]  loc_4016B6
004016BC  .v 74 0F        je      short <loc_4016CD>
004016BE  . 0FBE041F      movsx   eax, byte ptr [edi+ebx]
004016C2  . 0FBE0C1E      movsx   ecx, byte ptr [esi+ebx]
004016C6  . 3BC1          cmp     eax, ecx
004016C8  .v 75 18        jnz     short <loc_4016E2>
004016CA  . 43            inc     ebx
004016CB  .^ EB E9        jmp     short <loc_4016B6>
004016CD  > 6A 00         push    0                           loc_4016CD
004016CF  . 68 0A304000   push    offset <Caption>            Title = "Splish, Splash"
004016D4  . 68 42304000   push    offset <szGoodjobnowkeygenit_>  Text = "Good job, now keygen it."
004016D9  . 6A 00         push    0                           hOwner = NULL
004016DB  . E8 68000000   call    <MessageBoxA>               MessageBoxA
004016E0  .v EB 13        jmp     short <locret_4016F5>
004016E2  > 6A 00         push    0                           loc_4016E2
004016E4  . 68 0A304000   push    offset <Caption>            Title = "Splish, Splash"
004016E9  . 68 67304000   push    offset <szSorrypleasetryagain_>  Text = "Sorry, please try again."
004016EE  . 6A 00         push    0                           hOwner = NULL
004016F0  . E8 53000000   call    <MessageBoxA>               MessageBoxA
004016F5  > C9            leave                               locret_4016F5
004016F6  . C2 0800       retn    8
```

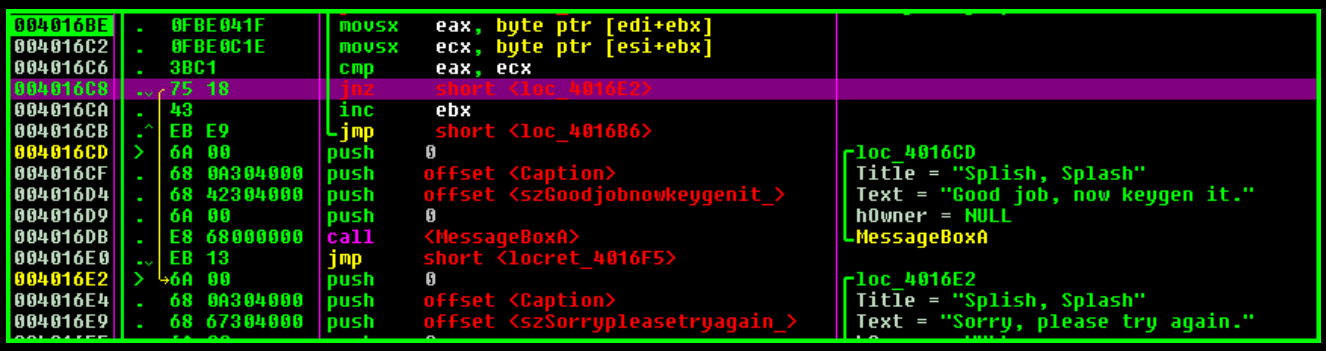
Có hai lệnh LEA nạp vào esi và edi :

```
Registers (FPU)                                    <
EAX 00000005
ECX 0000000A
EDX 00000004
EBX 00000006
ESP 0013FC74
EBP 0013FC74
ESI 0040324D offset <Splish.Remain_Buf>
EDI 00403258 offset <Splish.unk_403258>
```

ESI thì trỏ tới vùng Remain_Buf còn EDI trỏ tới vùng nhớ nào đó mà ta chưa rõ.OK chúng ta tiếp tục :

```
004016B4  . 33DB          xor     ebx, ebx                    <== ebx=0
004016B6  > 3B1D 6334400( cmp     ebx, dword ptr [<dword_403463>]  <== ebx = 0x7?
004016BC  .v 74 0F        je      short <loc_4016CD>          <== yes, jump out of routine
004016BE  . 0FBE041F      movsx   eax, byte ptr [edi+ebx]
004016C2  . 0FBE0C1E      movsx   ecx, byte ptr [esi+ebx]
004016C6  . 3BC1          cmp     eax, ecx
004016C8  .v 75 18        jnz     short <loc_4016E2>
004016CA  . 43            inc     ebx
004016CB  .^ EB E9        jmp     short <loc_4016B6>
```

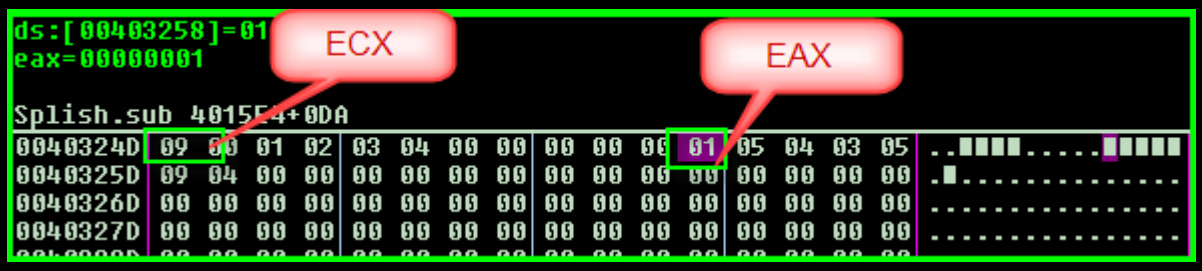
Ta thấy thanh ghi ebx bị clear, đem so sánh với 0x7, nếu bằng sẽ thoát khỏi vòng lặp.Tiếp theo ta gặp đoạn code sau :



Ta thấy có hai lệnh movsx chuyển hai giá trị :

a. giá trị thứ nhất từ vùng nhớ do edi trỏ tới, chuyển vào eax
b. giá trị thứ hai là giá trị của vùng Remain_Buf mà esi trỏ tới, chuyển vào ecx.

Sau đó hai giá trị này được so sánh với nhau, nếu chúng không bằng nhau thì sẽ thoát khỏi vòng lặp so sánh ngay. Vậy ta xem hai giá trị được chuyển vào eax và ecx là gì :



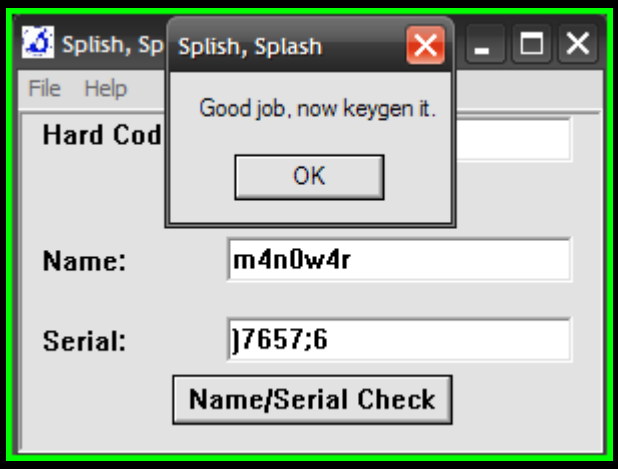
Như vậy, ta thấy giá trị đầu tiên của vùng Remain_Buf sẽ được so sánh với giá trị đầu tiên của vùng nhớ mà edi trỏ tới (ta gọi là vùng Real_Buf). Do đó để quá trình so sánh thành công thì giá trị đầu tiên của Remain_Buf sẽ phải là 0x1. Vậy làm thế nào để tính ra được giá trị đó, và Serial đầu vào ta phải nhập thế nào thì mới tính toán ra được. Quay ngược trở lại bài toán, ta có như sau :

1. Fake Serial nhập vào là "123456"
2. Số 1 tương ứng với 0x31. Số này được lấy ra và chia cho 0xA.
3. Phần dư lúc này là 0x9, còn kết quả có được là 0x4.

Vậy ta có công thức : 0x31 = 0x4*0xA + 0x9.

Giờ do yêu cầu bài toán thì 0x9 phải là 0x1 thì quá trình so sánh mới đúng. Vậy khi thay thế vào ta được : 0x4*0xA + 0x1 = 0x29 (ASCII: ")"). Làm các phép tính toán với lập

luận tương tự như trên, tôi có được chuỗi Serial chính xác nhập vào là : "**)7657;6**". Chạy thử chương trình và nhập chuỗi Serial mà ta tìm được :



Tuy nhiên nếu các bạn để ý một chút thì trong quá trình tính toán ta có thể suy ra được nhiều Serial khác nhau ứng với Name mà chúng ta nhập vào. Nếu các bạn muốn tìm hiểu quá trình tính toán và tạo keygen thì nên tham khảo thêm đoạn code sau :

```
0040162D  .  B9 0A000000   mov     ecx, 0A
00401632  >  0FBE041E      movsx   eax, byte ptr [esi+ebx]      loc_401632
00401636  .  99            cdq
00401637  .  F7F9          idiv    ecx
00401639  .  33D3          xor     edx, ebx
0040163B  .  83C2 02       add     edx, 2
0040163E  .  80FA 0A       cmp     dl, 0A
00401641  .v 7C 03         jl      short <loc_401646>
00401643  .  80EA 0A       sub     dl, 0A
00401646  >  88141F        mov     byte ptr [edi+ebx], dl       loc_401646
00401649  .  43            inc     ebx
0040164A  .  3B1D 6334400  cmp     ebx, dword ptr [<dword_403463>]
00401650  .^ 75 E0         jnz     short <loc_401632>
```

Đoạn code này tính toán dựa trên Name mà ta nhập vào, sau đó lưu các giá trị vào một vùng nhớ. Và vùng nhớ đó chính là vùng Real_Buf mà tôi đã đề cập ở trên.

V. Kết luận

Cuối cùng cũng hoàn thành bài viết, tốn 17 trang giấy cho việc giải quyết một bài toán mà có thể nhiều bạn cho là đơn giản.Như đã nói bài này mục đích để Refresh lại cũng như làm đà để tôi có cảm hứng viết tiếp, hi vọng sau một thời gian khá dài ngừng viết, câu cú trong bài viết này không quá lủng củng. Rất cảm ơn anh em và các bạn đã dành thời gian quí báu để đọc tài liệu này. Hẹn gặp lại ở bài 17.

PS: Tài liệu này chỉ mang tính tham khảo, tác giả không chịu trách nhiệm nếu người đọc sử dụng nó vào bất kì mục đích nào.

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, 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). 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