INTRODUCTION TO THE CRACKING WITH OLLYDBG

FROM CRACKLATINOS

(_kienmanowar_)

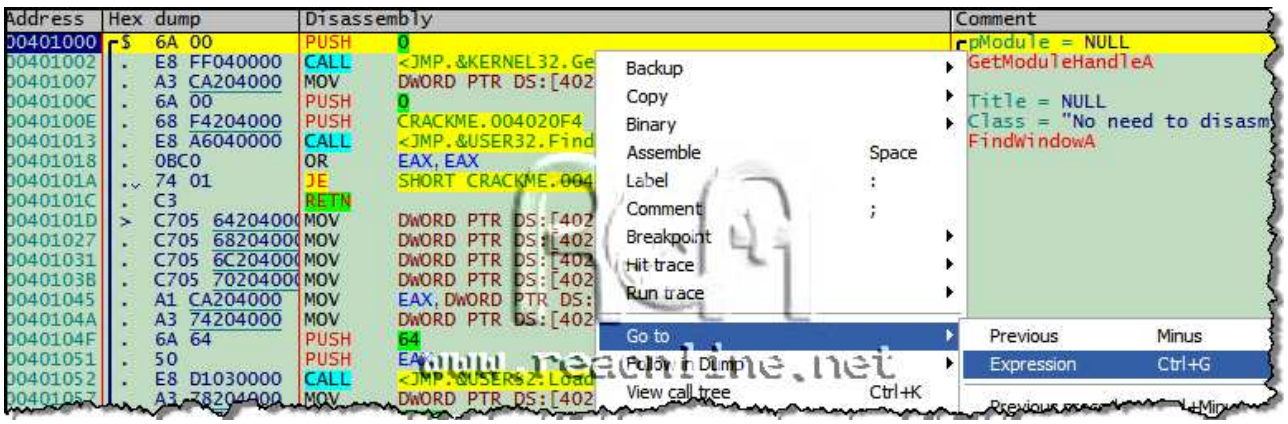
Một cái đầu lạnh để vững vàng, một trái tim đỏ lửa để yêu và làm việc hết mình!

I. Giới thiệu chung

Công việc bề bộn và ngập đầu trong dự án nhưng tôi vẫn ở đây, vẫn dành trọn tình cảm cho những đam mê của riêng mình. Vẫn muốn đóng góp thật nhiều cho dù là nhỏ bé nhất cho ngôi nhà REA thân yêu. Trong sáu phần trước tôi đã tập trung giới thiệu một cách tổng quan những câu lệnh cơ bản và thường được sử dụng nhất. Tuy nhiên vẫn còn rất nhiều các câu lệnh khác nhưng trong khuôn khổ có hạn của bài viết không thể giới thiệu hết được, và thiết nghĩ việc liệt kê hết ra sẽ rất nhàm chán cho nên chi tiết về các câu lệnh sẽ được đề cập tới khi chúng ta gặp phải trong quá trình làm việc với Olly. Trong phần 7 của loạt tuts này sẽ tập trung giới thiệu tới các bạn về lệnh CALL và RET. Xét một cách tổng quan thì đây là 2 lệnh đơn giản, tuy nhiên khi đi vào chi tiết nhiều người thường khó hiểu đặc biệt là những bạn mới làm quen với ASM. Chính vì lý do này mà phần 7 sẽ đi sâu vào giải quyết 2 lệnh này.

II.CALL và RET

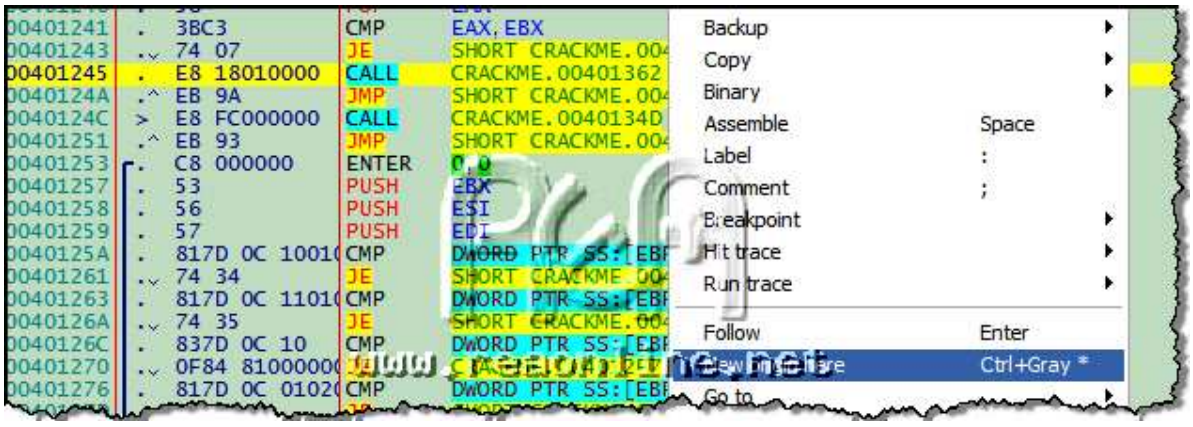
Trước tiên chúng ta mở Olly lên và load crackme vào. Tại cửa sổ CPU, bạn nhấn chuột phải và chọn như sau (hoặc nhấn **Ctrl + G**) :



Tại cửa sổ vừa mới xuất hiện bạn gõ vào địa chỉ như sau : `0x401245` và nhấn OK

Olly sẽ đưa chúng ta đến địa chỉ mà chúng ta vừa gõ. Tại đây chúng ta sẽ thấy một lệnh CALL :

Đây là lệnh CALL mà tôi dùng để thực hành trong bài viết này. Để có thể thực thi được lệnh CALL này thì chúng ta phải thiết lập lại thanh ghi EIP trỏ đúng tới địa chỉ `0x401245`. Điều này được thực hiện rất đơn giản bằng cách nhấn chuột phải lên lệnh CALL và chọn *New origin here* như sau :



Quan sát thanh ghi EIP các bạn sẽ thấy nó thay đổi (xem lại về thanh ghi EIP ở phần trước) :

Ok, giờ quay trở lại lệnh CALL của chúng ta :

Về bản chất thì lệnh CALL được dùng để gọi một thủ tục (chương trình con). Có hai kiểu gọi thủ tục đó là gọi trực tiếp và gọi gián tiếp. Cú pháp của lệnh gọi thủ tục trực tiếp :

CALL Name (trong đó Name là tên của thủ tục)

Cú pháp của lệnh gọi thủ tục gián tiếp :

CALL addr_expresstion (trong đó addr_expression là một thanh ghi hay ô nhớ chứa địa chỉ của thủ tục)

Note : Xem thêm hình minh họa về lệnh CALL trong file CaLL_stack.gif.

Lệnh CALL dùng để chuyển hoạt động của bộ vi xử lý từ chương trình chính sang chương trình con. Có nghĩa là khi chúng ta trace qua một lệnh CALL trong Olly thì các câu lệnh bên trong của câu lệnh CALL này sẽ được thực hiện, điều này giống như việc các bạn lập trình trên các ngôn ngữ bậc cao bằng cách tạo ra các thủ tục hay các hàm. Khi thực hiện chương trình thì ở đâu có lời gọi hàm hay thủ tục thì sẽ thực hiện các lệnh bên trong hàm/thủ tục đó đã rồi mới trả lại quyền điều khiển cho chương trình chính. Một ví dụ minh họa để hiểu rõ vấn đề :

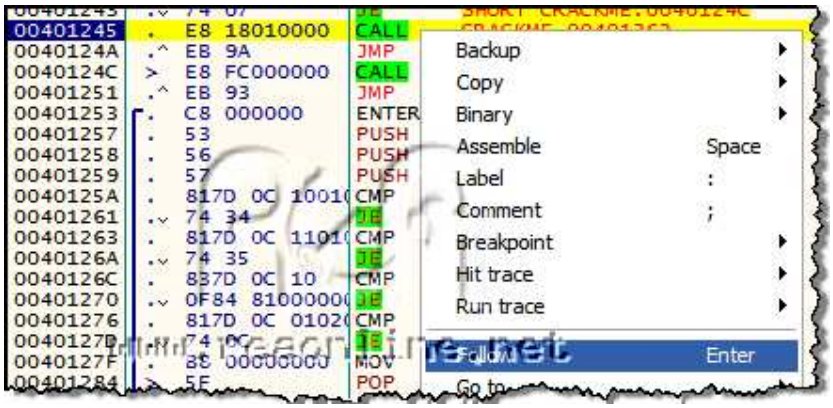
Câu lệnh `CALL CRACKME.00401362` có nghĩa là câu lệnh tiếp theo sẽ được thực thi bắt đầu tại địa chỉ `0x00401362` và sau khi thực hiện xong chương trình con (thủ tục/hàm) bắt đầu từ địa chỉ đó, nó sẽ trở về (trả lại quyền điều khiển cho chương trình chính) thực thi câu lệnh tiếp theo bên dưới lệnh CALL. Trong ví dụ của chúng ta thì sau khi thực hiện các lệnh bên trong lệnh `CALL CRACKME.00401362` xong thì nó sẽ trở về lệnh `JMP SHORT CRACKME.004011E6` tại địa chỉ `0x0040124A` và thực thi câu lệnh này theo đúng trình tự thực hiện của chương trình chính. Vậy trong Ollydbg hỗ trợ cho tôi chức năng gì để tôi có thể làm việc với lệnh CALL?
Nếu như tôi muốn xem nội dung của lệnh CALL và muốn trace để debug các lệnh bên trong lệnh CALL thì Ollydbg hỗ trợ cho tôi tính năng Step Into (F7). Tức là tại bất kì câu lệnh CALL nào bạn chỉ việc nhấn F7 thì bạn sẽ chui vào trong lòng lệnh CALL đó để xem các lệnh. Tuy nhiên, nếu lệnh CALL nào cũng nhấn F7 thì e rằng nhiều lúc thực sự không cần thiết vì tôi chỉ muốn có cái nhìn tổng quan về lệnh CALL đó mà chưa cần quan tâm đến nội dung bên trong lệnh CALL có những gì, về mặt này Olly hỗ trợ cho chúng ta tính năng Step Over (F8). Tính năng này giúp ta thực thi luôn lệnh CALL mà không cần phải chui vào thực hiện

từng lệnh bên trong lệnh CALL như chúng ta nhấn F7, sau khi thực hiện xong lệnh CALL nó sẽ đưa chúng ta thẳng tới lệnh tiếp theo bên dưới lệnh CALL.
Vậy từ đây ta đi tới kết luận, khi ta đứng tại một lệnh CALL bất kỳ ta sẽ có hai lựa chọn :

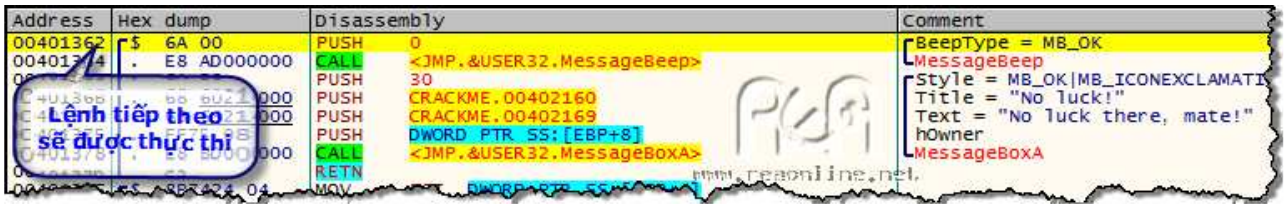
1. Nếu như lệnh CALL đó là quan trọng và chúng ta muốn quan sát cách hành xử bên trong lệnh CALL đó thì chúng ta sử dụng chức năng Step Into.

2. Nếu như lệnh CALL đó không có gì để ta phải quan tâm và ta chỉ cần kết quả trả về của nó mà không cần biết bên trong nó làm những gì thì ta sử dụng chức năng Step Over.

Tuy nhiên hai chức năng này vẫn phải thực thi các lệnh, nhưng ở đây tôi chỉ muốn quan sát bên trong lệnh CALL có những lệnh gì mà không muốn phải thực thi bất kì lệnh nào thì Olly có tính năng hỗ trợ không?? Với một lệnh CALL mà ta quan tâm, Olly hỗ trợ cho chúng ta chức năng Follow. Nhấn chuột phải lên lệnh CALL và chọn Follw, xem hình minh họa dưới đây :

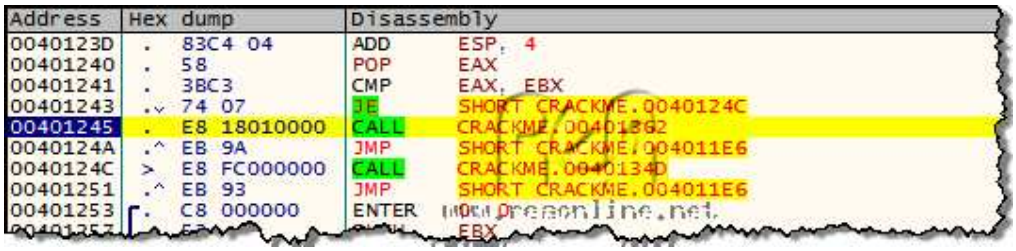


Tính năng này chỉ đơn giản là cho phép ta xem câu lệnh tiếp theo sẽ được thực hiện mà bản thân nó không hề thực thi bất kì câu lênh nào của chương trình. Nó cũng không làm thay đổi thanh ghi EIP, quan sát hình minh họa dưới đây :

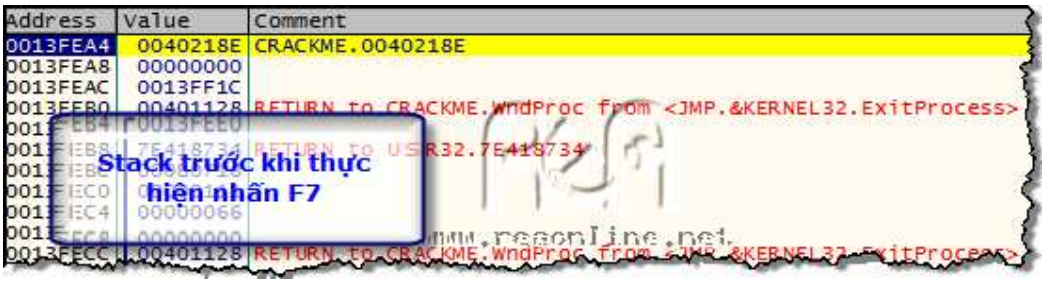


Khi tôi Follow theo lệnh CALL tại địa chỉ `0x00401362` tôi sẽ quan sát được các lệnh bên trong lệnh CALL này và rõ ràng rằng địa chỉ của lệnh tiếp theo sẽ được thực thi sẽ bắt đầu tại `0x00401362`. Kết thúc lệnh CALL này để trở về chương trình chính, các bạn quan sát sẽ thấy có một lệnh đặc biệt đó chính là RET/ RETN. Lệnh RET thường được đặt ở cuối chương trình con để bộ vi xử lý lấy lại địa chỉ trở về (địa chỉ của lệnh tiếp theo bên dưới lệnh CALL),

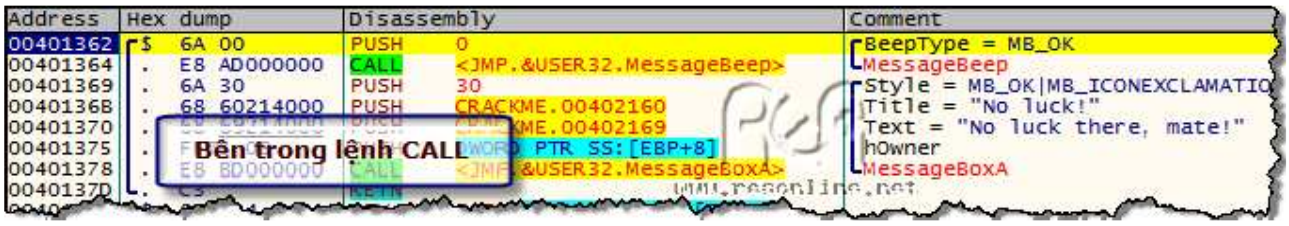
địa chỉ này được tự động cất vào stack mỗi khi có lời gọi đến một chương trình con. Theo như ví dụ của chúng ta thì sau khi thực hiện lệnh RET tại `0040137D  \.  C3  RETN` ta sẽ trở về địa chỉ 0x0040124A.
Đây là những kiến thức rất quan trọng các bạn cần phải nắm được về lệnh CALL. Ở trên các bạn mới chỉ làm những công việc là nhìn và quan sát những gì bên trong lệnh CALL mà chưa thực thi bất kì một câu lệnh nào. Phần tiếp theo dưới đây chúng ta sẽ áp dụng các chức năng Step Into và Step Over. Ok, quay trở lại ví dụ chúng ta đang dừng lại tại lệnh CALL :



Bây giờ chúng ta nhấn F7 để đi vào trong lệnh CALL, khà khà nhưng trước tiên chúng ta cần quan sát cửa sổ Stack trước khi thực hiện nhấn F7. Cửa sổ Stack sẽ cung cấp cho chúng ta một thông tin rất quan trọng đó là địa chỉ trở về (địa chỉ của lệnh bên dưới lệnh CALL), địa chỉ này sẽ được cất vào Stack để từ đó chương trình sẽ biết được nơi mà nó cần trở về khi thực hiện câu lệnh RET.



Lưu ý giá trị trên Stack của tôi có thể khác với máy của các bạn và điều này không quan trọng.Bây giờ chúng ta nhấn F7 để chui vào trong lệnh CALL đông thời quan sát cửa sổ Stack xem có gì xảy ra :



Khác với chức năng Follow ở trên lúc này thanh ghi EIP sẽ thay đổi và điều này có nghĩa là chúng ta đang thực thi chương trình con :

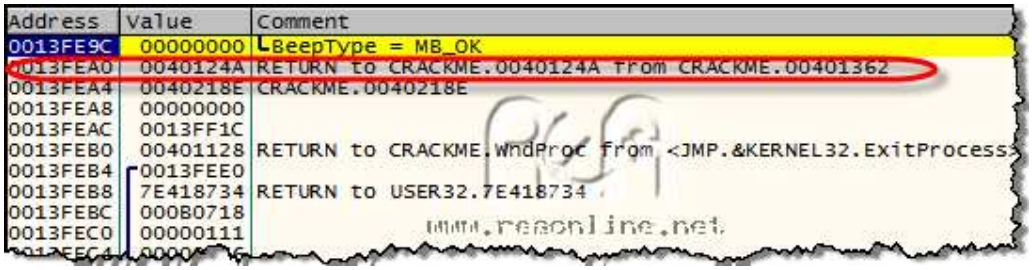
Cửa số Stack lúc này thay đổi như sau :

Như bạn thấy trên hình minh họa thanh ghi ESP trỏ lên đỉnh của Stack đã được giảm đi thành `0x0013FEA0` và địa chỉ của câu lệnh tiếp theo sau lệnh CALL (địa chỉ trở về) được đẩy vào đỉnh của Stack.Và đây chính là cơ sở cho lệnh RET thoát khỏi chương trình con để quay trở về chương trình chính ☺. Olly cũng đã cung cấp cho chúng ta thông tin : `RETURN to CRACKME.0040124A from CRACKME.00401362`

Quay trở lại công việc chúng ta đang làm, các bạn nhấn tiếp F7 để thực hiện lệnh tại địa chỉ 0x00401362 đó là :

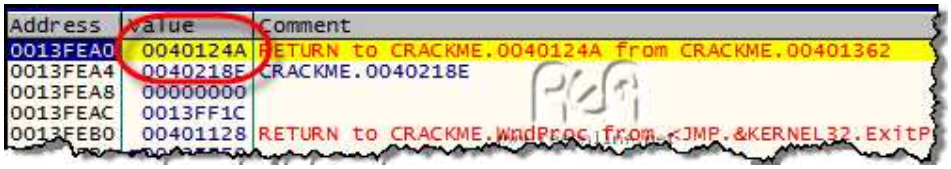
```
00401362   /$   6A 00   PUSH    0          ; /BeepType = MB_OK
```

Lệnh PUSH 0 được thực thi tức là đẩy giá trị 0 vào đỉnh của Stack. Lúc này địa chỉ trở về sẽ được đẩy lùi xuống nhường chỗ cho giá trị 0x0 :



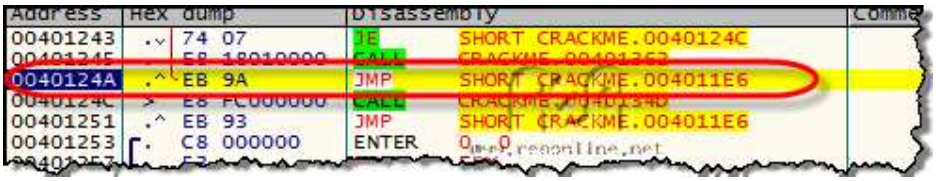
Chương trình có thể kiểm soát hàng nghìn câu lệnh bên trong một chương trình con, bản thân bên trong chương trình con có thể thực hiện hàng nghìn lệnh PUSH và POP nhưng khi đã ở tại câu lệnh RET thì lúc đó trên stack sẽ chỉ có địa chỉ trở về (chương trình gọi tới chương trình con), để từ đó câu lệnh RET sẽ căn cứ vào địa chỉ trở về này để quay về lệnh tiếp theo dưới lệnh CALL. Tiếp theo ví dụ trên, chúng ta nhấn F8 để thực hiện các lệnh tiếp theo cho tới khi chúng ta dừng lại tại câu lệnh RET.

Thông tin trên cửa sổ Stack :



Như chúng ta đã đề cập về lệnh RET ở phía trên, khi ta thực hiện lệnh RET thì nó sẽ căn cứ vào địa chỉ trở về trên Stack để quay về lệnh tiếp theo dưới lệnh CALL. Đồng thời thêm vào

đó nó sẽ xóa địa chỉ trở về này khỏi Stack vì giá trị này không còn hữu ích nữa. OK ta nhấn F7 để thực hiện lệnh RETN, ta sẽ được đưa tới địa chỉ `0x0040124A` :

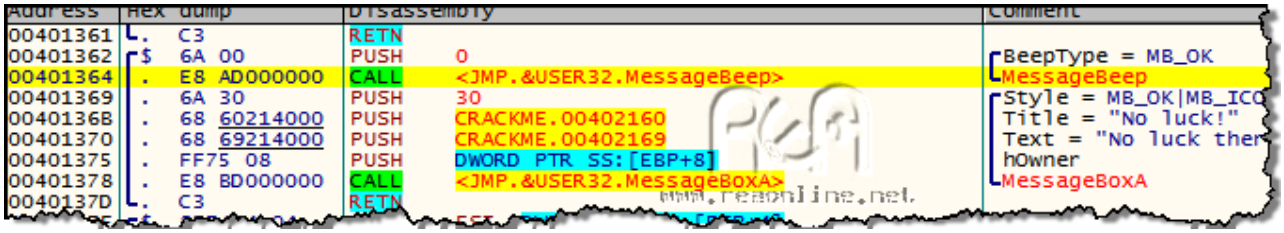


Tuy nhiên, trong một số trường hợp ta có thể thực thi lệnh RET mà không cần phải có lệnh CALL. Một ví dụ như sau :

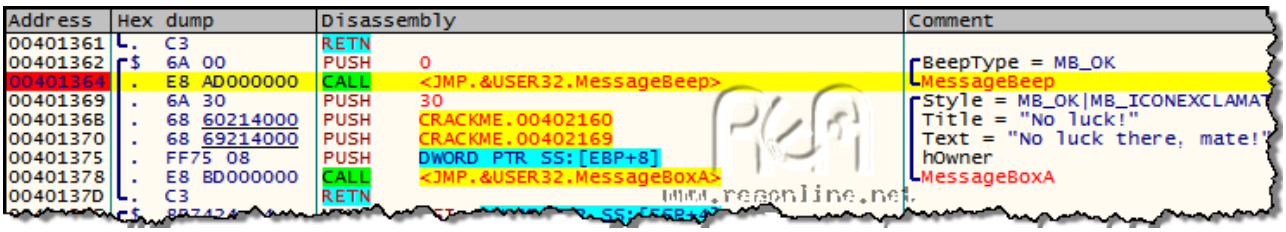
```
PUSH 401256
RET
```

Trong ví dụ trên, lệnh PUSH sẽ đẩy giá trị `0x401256` vào đỉnh của Stack và bên dưới lệnh PUSH này là một lệnh RET. Theo như lý thuyết thì lệnh RET sẽ căn cứ vào địa chỉ trở về được cất trên Stack để quay trở về vậy cho nên khi thực hiện lệnh RET chúng ta sẽ được đưa tới địa chỉ của lệnh tiếp theo sẽ được thực hiện là `0x401256`. Qua ví dụ này ta rút ra một nhận xét là cặp lệnh PUSH và RET tương đương với lệnh JMP ☺.

Bây giờ quay trở lại ví dụ chính của chúng ta, tôi sẽ minh họa một trường hợp khác, trong Olly nhấn **Ctrl + G** và gõ vào địa chỉ `0x401364`. Ta sẽ ở đây trong Olly :



Quá trình này không làm thay đổi thanh ghi EIP, nhấn F2 để đặt một BP tại địa chỉ trên. Mục đích của việc đặt BP là tôi muốn chương trình sau khi thực thi sẽ dừng lại tại nơi mà tôi đã đặt BP.



Sau khi đặt BP giống như trên xong, lúc này ta nhấn F9 để Run chương trình :

Tại Crackme v1.0, ta nhấn Help và chọn Register :

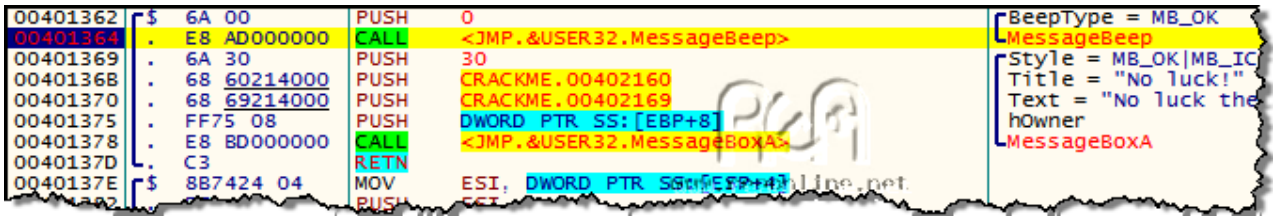
Cửa sổ Register sẽ hiện ra yêu cầu ta nhập thông tin :

1

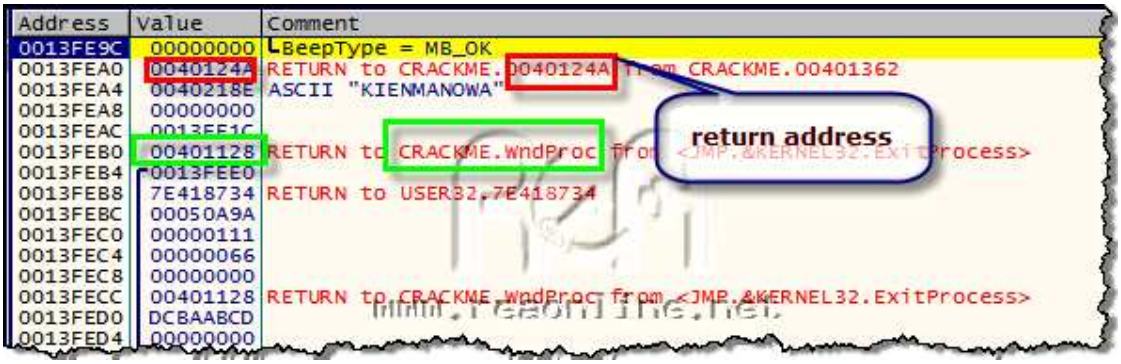
Ta nhập thông tin vào :

1

Nhấn OK, ngay lập tức chúng ta sẽ dừng tại điểm mà chúng ta đặt BP trong Olly :



Lúc này quan sát cửa sổ Stack ta có được như sau :



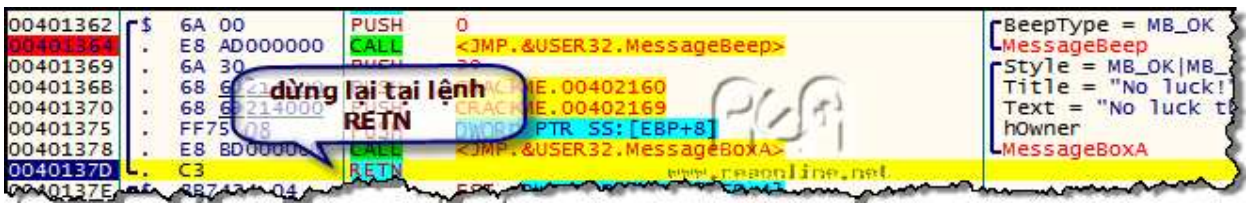
Như các bạn quan sát thấy trên cửa sổ Stack lúc này cố một số dòng "RETURN TO…", nhưng theo lý thuyết mà chúng ta đã lập luận ở trên thì dòng nào ở vị trí cao nhất trong tất cả các dòng sẽ chính là nơi chứa địa chỉ trở về chương trình chính khi chương trình con thực hiện câu lệnh RET bên trong nó. Vì khi chúng ta ở tại câu lệnh RET thì dòng "RETURN TO…" cao nhất sẽ được đẩy lên đỉnh của Stack và lúc này địa chỉ trở về sẽ là `0x0040124A`.

Ta đang dừng lại tại BP đã thiết lập, bây giờ nhấn F8 để trace tới lệnh RET.Tuy nhiên bạn để ý trên hình minh họa trước khi tới được lệnh RET thì sẽ phải đi qua một lệnh CALL :

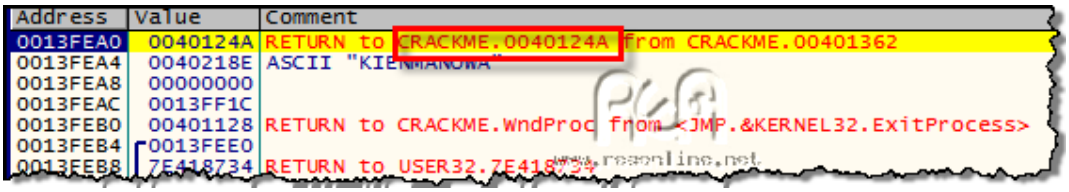
```
00401378 |. E8 BD000000  CALL MP.&USER32.MessageBoxA> ; \MessageBoxA
```

Ta không cần quan tâm bên trong nó ra sao chỉ cần biết rằng khi trace over qua nó sẽ hiện ra một thông báo :

Nhấn Ok để tiếp tục, lúc này chúng ta sẽ dừng lại tại câu lệnh RETN và chuẩn bị thực hiện câu lệnh này :

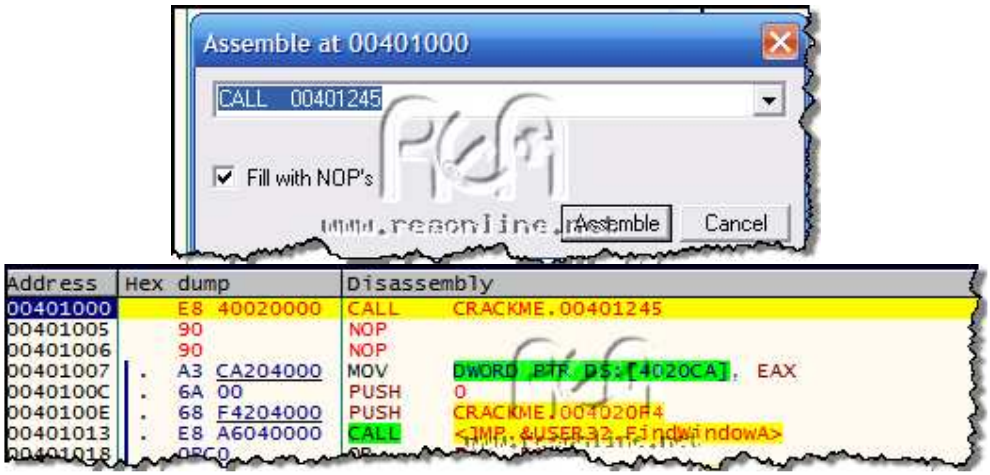


Quan sát trên cửa sổ Stack, ta sẽ có được thông tin của địa chỉ trở về :

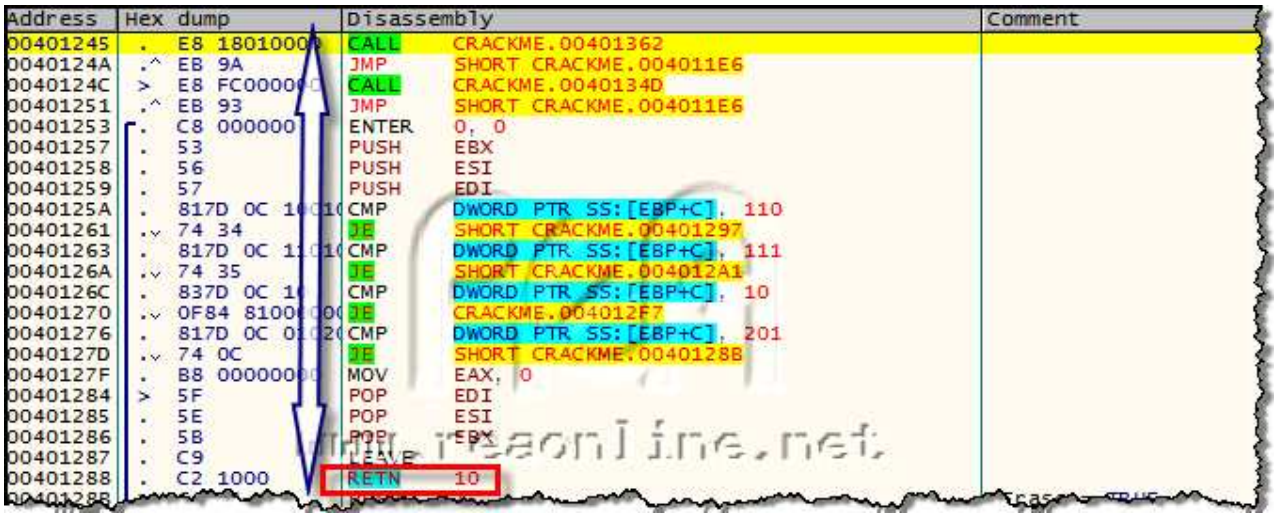


Điểm khác biệt của lần phân tích này so với lần phân tích ở trên chính là việc thay đổi thanh ghi EIP, và việc chúng ta thực hiện một câu lệnh CALL (là một phần của chương trình) mà không phải là toàn bộ chương trình.Thanh ghi EIP thay đổi thông qua việc thực thi chương trình và dừng lại tại điểm mà chúng ta đặt BP. Nếu chúng ta tiếp tục thực thi chương trình bằng việc nhấn F9 thì nó sẽ hoàn toàn chạy một cách bình thường như không có gì xảy ra ☺.
Tuy nhiên điều mà tôi muốn nhấn mạnh ở đây đó là công dụng của cửa sổ Stack, khi chúng ta đang thực thi chương trình và bị dừng lại vì một lý do nào đó thì thông tin mà cửa sổ Stack cung cấp cho chúng ta sẽ cho ta biết được nơi mà thủ tục được gọi cũng như địa chỉ mà nó sẽ trở về. Tuy nhiên các chương trình thường sử dụng các lệnh CALL lồng nhau thì qua cửa sổ Stack này chúng ta cũng có được các thông tin để lần ngược về nơi gọi.

Phần tiếp theo tôi sẽ lấy một ví dụ minh họa khác, nhấn Ctrl+F2 để restart Olly và edit lại như hình minh họa dưới đây :

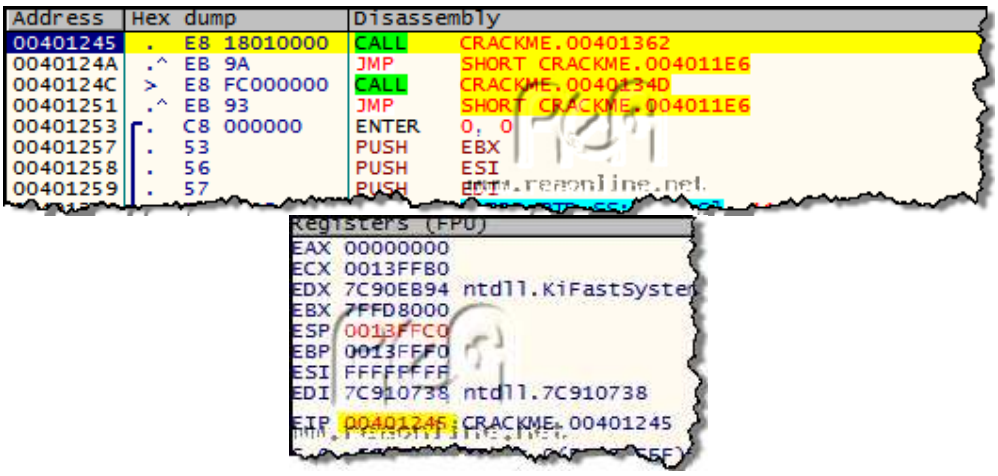


Nhấn Enter để Follow vào quan sát nội dung bên trong lệnh CALL :

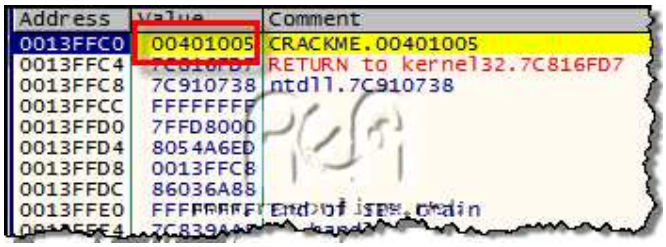


Như quan sát ở trên thì sau khi chúng ta sửa đổi lúc này câu lệnh đầu tiên sẽ được thực hiện sẽ bắt đầu tại địa chỉ `0x00401245` và kết thục tại câu lệnh RETN tại địa chỉ `0x00401288` (mà ở đây chúng ta thấy là lệnh RETN 10 hơi khác một chút với lệnh RETN).

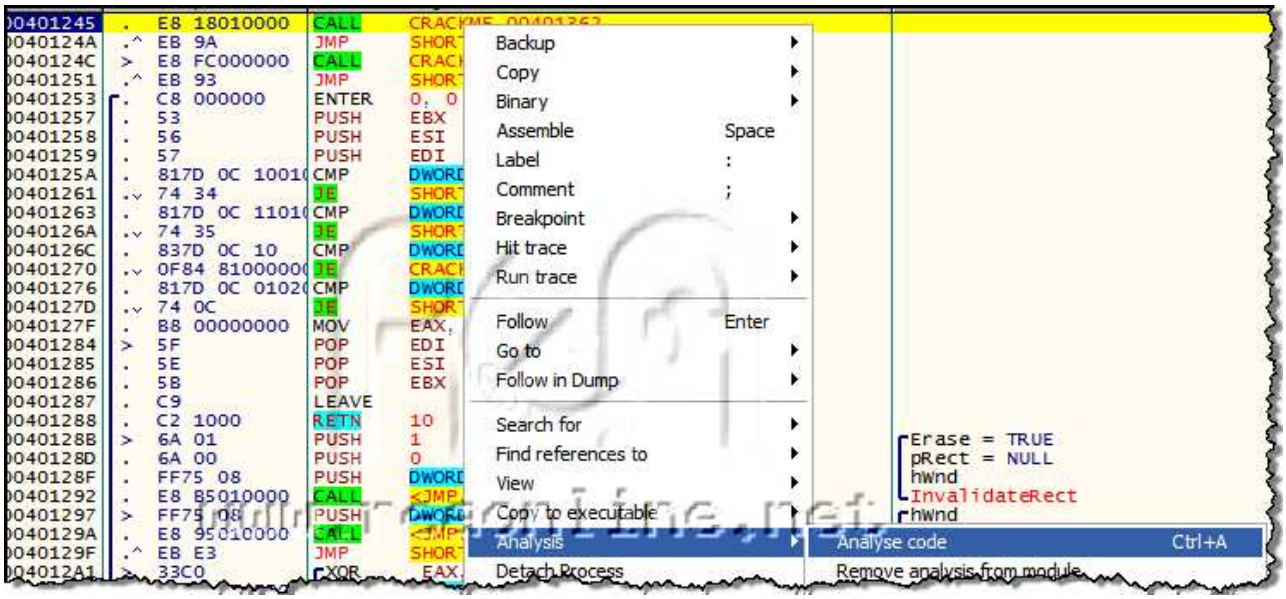
Tại địa chỉ mà chúng ta vừa mới Follow tới là `0x00401245` ta nhấn "-" để quay về nơi phát sinh ra lời gọi tới nó, sau đó chúng ta nhấn F7 để trace into vào trong lệnh CALL, lú đó thành ghi EIP sẽ thay đổi :



Đồng thời trên cửa sổ Stack ta sẽ quan sát thấy địa chỉ trờ về sau khi thực hiện xong :



Chúng ta nhìn thấy giá trị là `0x00401005`, điều này là rất đúng và hợp lý. Tuy nhiên ở đây tôi không khẳng định rằng sau khi thực hiện xong Ollydbg sẽ đưa chúng ta trở về đúng nơi mà chúng ta đã thấy ở trên.Lý do là vì chúng ta đã sửa lại hướng thực thi của cả chương trình ☺ điều này chẳng khác gì trong một cỗ máy đang hoạt động trơn tru bị bạn làm thay đổi qui trình hoạt động của nó.
Để có thể fix được nó thì trong cửa sổ CPU nhấn chuột phải và chọn :



Hoặc có thể nhấn phím tắt là **Ctrl + A**. Sau khi analyze xong quan sát cửa sổ Stack :

Ok nó thông báo cho chúng ta như sau :

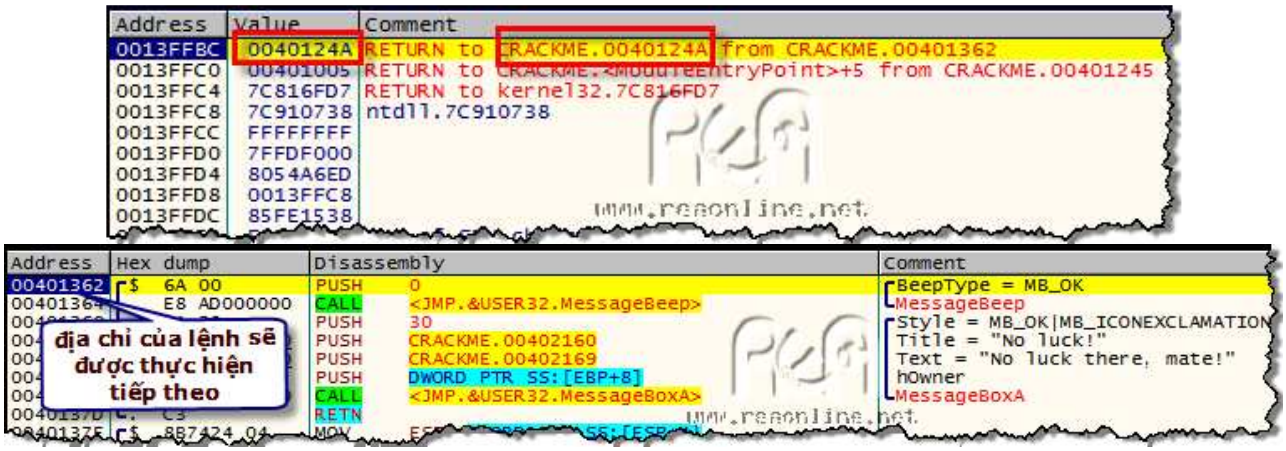
```
0013FFC0    00401005    RETURN    to    CRACKME.<ModuleEntryPoint>+5    from
CRACKME.00401245
```

Điều này chính là giá trị 0x00401000 `(EntryPoint) + 0x5 = 0x00401005`. Rõ ràng là thông tin sau khi Olly analyze mang lại đầy đủ ý nghĩa hơn là thông tin ban đầu mà chúng ta có được. Sau khi nhấn F7 chúng ta sẽ ở đây trong Olly :

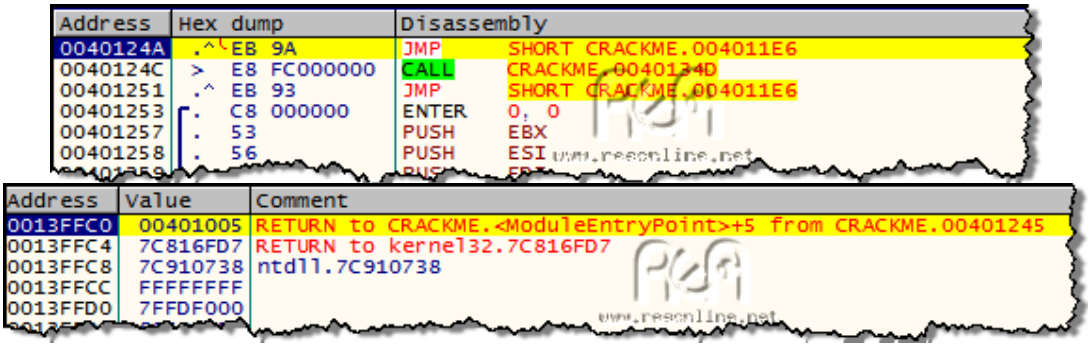
Tại đây chúng ta nhấn tiếp F7 để trace vào bên trong lệnh :

```
00401245    $    E8 18010000    CALL    CRACKME.00401362
```

Quan sát trên cửa sổ Stack ta có được như sau :



Thông qua hình minh họa ở trên ta thấy rằng trước khi đi vào mỗi lệnh CALL địa chỉ của lệnh tiếp theo (dưới lệnh CALL) đều được đẩy lên đỉnh của Stack _ đó chính là địa chỉ trở về. Điều này sẽ giúp cho chúng ta hiểu được cơ chế quản lý khi có các lệnh CALL lồng nhau. Trong ví dụ trên, ta đang ở tại địa chỉ `0x00401362` và đây chính là nơi mà lệnh tiếp theo được thực hiện.Sau khi thực hiện toàn bộ các lệnh bên trong và tới lệnh RETN tại địa chỉ `0x0040137D`, khi ta thực hiện lệnh RETN này nó sẽ căn cứ vào địa chỉ trở về nằm trên Stack để quay ra ngoài, và lúc này địa chỉ trở về đó là `0x0040124A`. Thực hiện lệnh RETN ta sẽ ở đây theo đúng lý thuyết đã lập luận :



Và trên cửa sổ Stack ta có thông tin giống như hình minh họa ở trên, điều này có nghĩa là ta vẫn đang nằm trong lòng lệnh CALL (`00401000 >/$ E8 40020000 CALL CRACKME.00401245`) và địa chỉ trở về sẽ `là 0x00401005` bên dưới lệnh CALL này ☺. Tuy nhiên như tôi nói ở trên nếu bạn tiếp tục thực hiện các lệnh tiếp theo thì chương trình sẽ không return về đúng địa chỉ, đó là bởi vì chúng ta đã phá vỡ cấu trúc của chương trình. Nếu bạn cố tình thực hiện thì sẽ crash đó, ráng chịu!! ☺.

Okie tôi nghĩ đến đây là đã đủ cho phần 7 về Ollydbg, bài viết xin được kết thúc tại đây. Trong phần này tôi có tham khảo thêm một số tài liệu liên quan tới lập trình ASM nhưng có thế không tránh khỏi những sai sót trong quá trình viết, rất mong các bạn góp ý để tôi chỉnh sửa lại. Tôi tin là với những ví dụ minh họa trực quan như ở trên các bạn sẽ nắm được vấn đề nhanh hơn. Ngoài ra các bạn có thể tham khảo thêm các tài liệu liên quan để có được một cái nhìn sâu hơn. Hẹn gặp lại các bạn trong phần 8 của loạt bài viết về Olly, 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, dqtln, ARTEAM .... all my friend, and YOU.

--++--==[Thanks To]==--++--
iamidiot, WhyNotBar, trickyboy, dzungltvn, takada, hurt_heart, haule_nth, hytkl 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