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

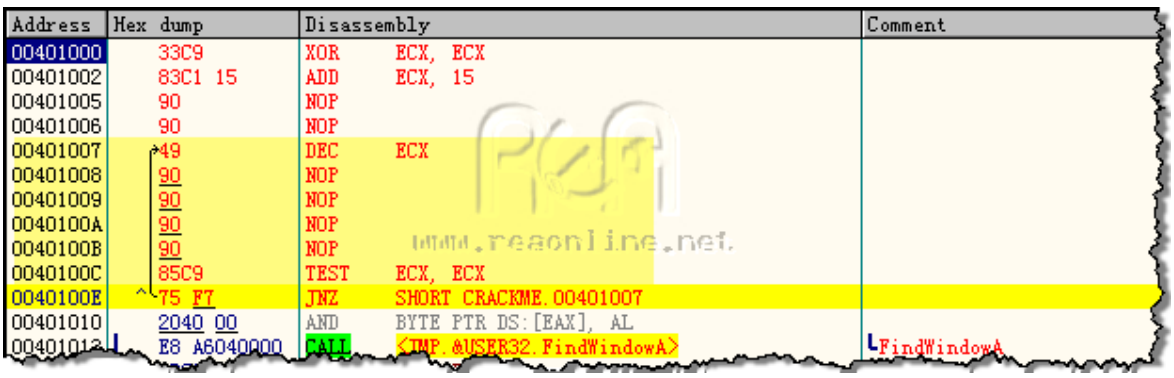
Đợt rồi bận quá, mải cài đặt và config hệ thống để phục vụ cho dự án upgrade, xém chút nữa thì quên mất viết tiếp bộ tut về Olly. Trong phần 7 tôi đã tập trung giới thiệu tới các bạn về lệnh CALL và RET. Xét một cách tổng quan thì đây là 2 lệnh đơn giản, tuy nhiên khi đi vào chi tiết nhiều người thường khó hiểu đặc biệt là những bạn mới làm quen với ASM. Hôm nay, chúng ta tiếp tục với phần 8 của loạt tuts này, phần này tập trung vào giới thiệu về vòng lặp và các câu lệnh liên quan tới chuỗi. Bài viết sẽ không sa đà vào giải thích cặn kẽ từng cấu trúc của vòng lặp mà sẽ đi nhanh và tập trung nhấn mạnh vào những phần quan trọng. Phần 8 này cũng là phần kết thúc việc giới thiệu các lệnh cơ bản, các phần tiếp theo sẽ tập trung vào những chủ đề, khía cạnh thú vị hơn.

II. Giới thiệu về vòng lặp

Một vòng lặp là một chuỗi các lệnh được lặp lại. Số lần lặp có thể đã được xác định trước hoặc phụ thuộc vào điều kiện. Thông thường trong Asm thì bộ đếm vòng lặp thường là thanh ghi ECX, nó được khởi tạo bằng *số lần lặp*. Sau khi thực hiện xong các lệnh bên trong vòng lặp giá trị của thanh ghi ECX sẽ tự động giảm đi 1. Nếu như giá trị của ECX còn khác 0 thì các lệnh trong vòng lặp còn tiếp tục được thực hiện , còn nếu như ECX bằng 0 thì sẽ thoát khỏi vòng lặp và thực hiện các lệnh tiếp theo bên dưới vòng lặp. Lấy ví dụ minh họa:

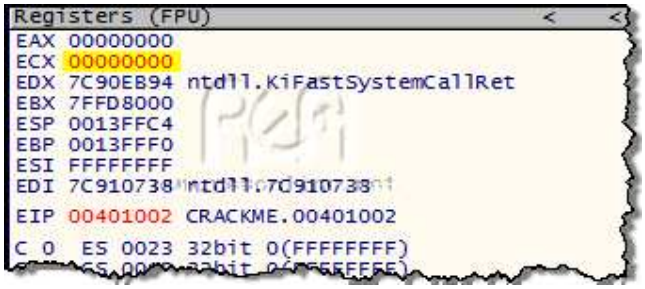
```
XOR ECX, ECX\ Đoạn lệnh này thực hiện nhiệm vụ khởi gán cho thanh ghi
ADD ECX, 15 / ECX.Số lần lặp lúc này sẽ là 15 và được lưu trong thanh ghi ECX.
LOOP:
DEC ECX // Giá trị của thanh ghi ECX sẽ được giảm đi 1
…… \   Thực hiện các lệnh trong thân vòng lặp
…… /
TEST ECX, ECX \ Chừng nào giá trị của ECX còn khác 0
JNE LOOP      / thì còn tiếp tục thực hiện
```

Minh họa ví dụ trên trong Olly như sau :



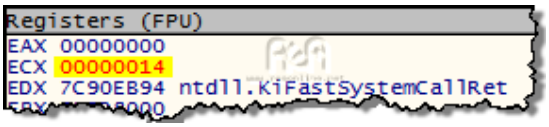
Vùng mà tôi khoanh vàng ở trên hình đó chính là vòng lặp, các lệnh trong vòng lặp này sẽ được thực hiện lặp lại cho tới khi nào thanh ghi ECX có giá trị 0. Các câu lệnh bắt đầu từ `0x00401008` cho tới `0x0040100C` chính là thân của vòng lặp. Do chỉ mang tính chất minh họa nên tôi để toàn lệnh NOP, các bạn có thể thay thế bằng các lệnh khác. Hiện tại chúng ta đang ở tại câu lệnh `XOR ECX, ECX` , nhấn F7 để trace và quan sát giá trị của thanh ghi `ECX` .

Lệnh `XOR ECX, ECX` sẽ xóa thanh ghi `ECX` về 0 :

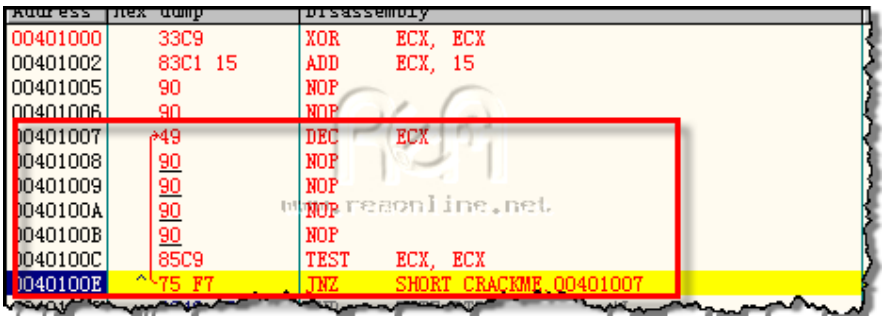


Lệnh `ADD ECX, 15` tương đương với việc gán `ECX = 0x15` :

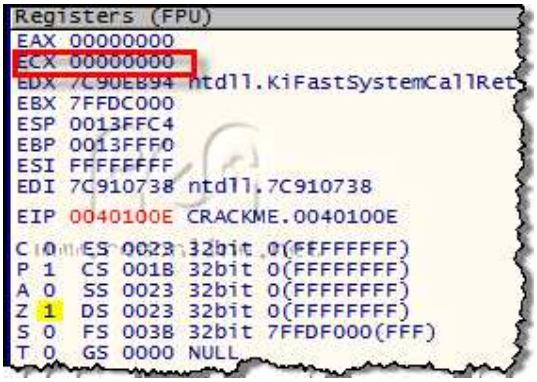
Nhấn F7 và trace qua lệnh `DEC ECX`. Giá trị của `ECX` lúc này sẽ đươc giảm đi 1 (`ECX = ECX -1 = 0x15 - 0x1 = 0x14`) :



Tiếp tục thực hiện cho tới khi chúng ta dừng lại tại lệnh `TEST ECX, ECX`, câu lệnh này sẽ kiểm tra xem giá trị của ECX đã bằng 0 chưa ? Nếu bằng 0 thì cờ Z sẽ được bật lên. Tuy nhiên lúc này giá trị ECX đang là 0x14, do đó khi trace qua lệnh TEST thì cờ Z không được active và vì vậy lệnh JNZ ở bên dưới sẽ nhảy lại về vị trí `0x00401007` (nơi bắt đầu của vòng lặp)



Trace tiếp để thực hiện các lệnh cho tới khi giá trị của thanh ghi ECX =0x0, lúc này lệnh TEST kiểm tra thấy giá trị của ECX = 0x0 cho nên nó sẽ active cờ Z. Khi cờ Z được active nên câu lệnh JNZ sẽ dựa vào cờ Z và đưa ra quyết định sẽ không nhảy về vị trí `0x00401007`.



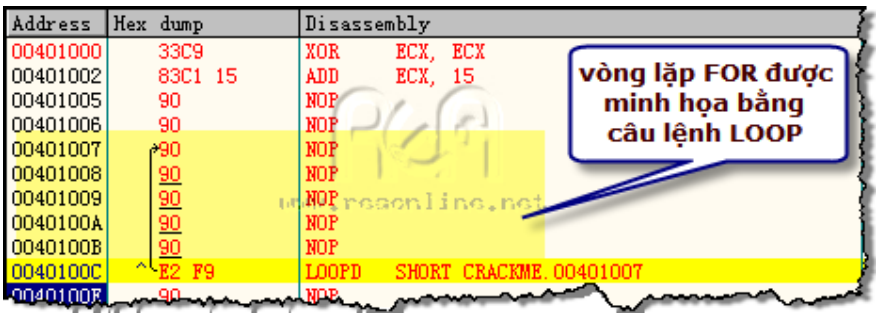
Các bạn sẽ dễ dàng nhận thấy mũi tên tại lệnh JNZ đã chuyển sang màu xám, điều đó có nghĩa là lệnh nhảy sẽ không được thực hiện. Còn nếu như mũi tên này có màu đỏ như hình bên trên thì tức là lệnh nhảy sẽ được thực hiện. Do mũi tên đã chuyển sang màu xám cho nên khi ta tiếp tục trace, chúng ta sẽ thoát ra khỏi vòng lặp và thực hiện các lệnh tiếp theo bên dưới. Đây chỉ là một ví dụ minh họa rất đơn giản để các bạn hình dùng về vòng lặp, các cấu trúc lặp như For, While và Repeat các bạn có thể đọc thêm trong các tài liệu về lập trình ASM.

Lệnh LOOP :

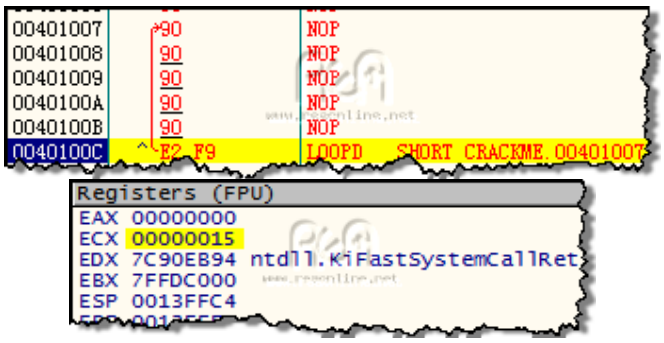
Câu lệnh này thường được sử dụng để thực hiện vòng lặp FOR. Nó có dạng như sau :

```
LOOP   Nhãn
```

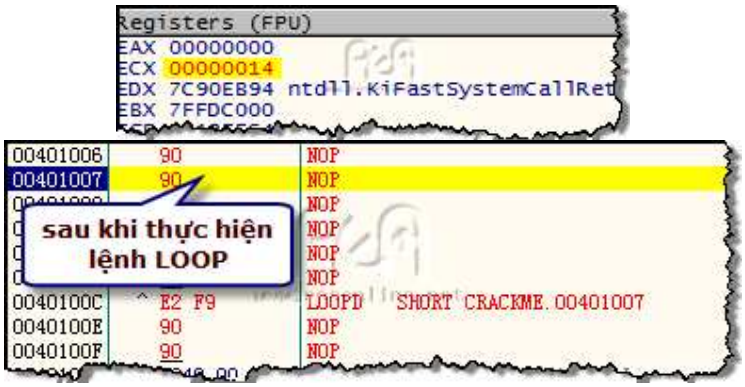
Bộ đếm vòng lặp được khởi tạo trong thanh ghi ECX. Mỗi lần thực hiện lệnh LOOP, thanh ghi ECX sẽ tự động được giảm đi 1, nếu thanh ghi ECX khác 0 thì điều khiển được chuyển tới *Nhãn* (tức là lặp lại lệnh bắt đầu từ vị trí của Nhãn). Nếu ECX = 0, lệnh tiếp theo sau lệnh LOOP sẽ được thực hiện. Với câu lệnh LOOP này ta có thể thay thế các lệnh đã minh họa ở mục trước như sau :



Như đã nói, vì chỉ mang tính chất minh họa cho nên tôi để toàn lệnh NOP cho đơn giản và dễ hiểu. Tượng tự như ở trên ta thực hiện lệnh XOR và lệnh ADD để khởi tạo giá trị cho thanh ghi ECX. Sau đó ta trace cho tới câu lệnh LOOP, giá trị của ECX lúc này đang là 0x15.



Bây giờ ta trace để thực hiện lệnh LOOP, ngay lập tức thanh ghi ECX sẽ được giảm xuống (trừ đi 1) và quyền điều khiển được chuyển tới *Nhãn* (địa chỉ `0x401007`).



Tiếp tục thực hiện cho tới khi giá trị của thanh ghi ECX được giảm xuống còn 0x1, và trace tới lệnh LOOP. Lúc này khi ta thực hiện lệnh LOOP, ECX sẽ được giảm xuống là 0x0, quyền điều khiển sẽ không được chuyển cho *Nhãn* nữa mà chuyển cho câu lệnh bên dưới lệnh LOOP.
Ngoài lệnh LOOP trên các bạn có thể tham khảo thêm về các lệnh `LOOPE/LOOPZ`, `LOOPNE/LOOPNZ` trong các tài liệu.

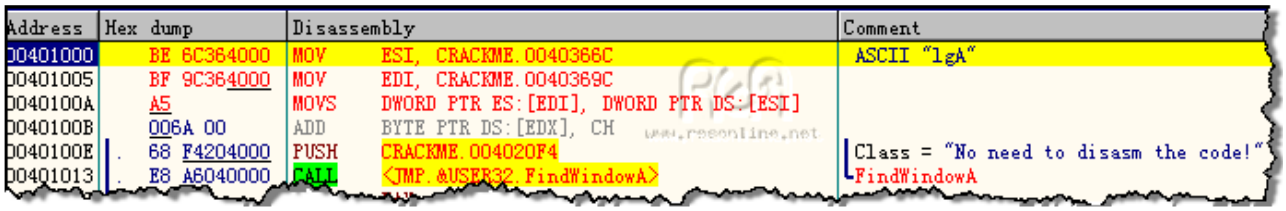
III. Các lệnh liên quan đến chuỗi

Lệnh MOVS :

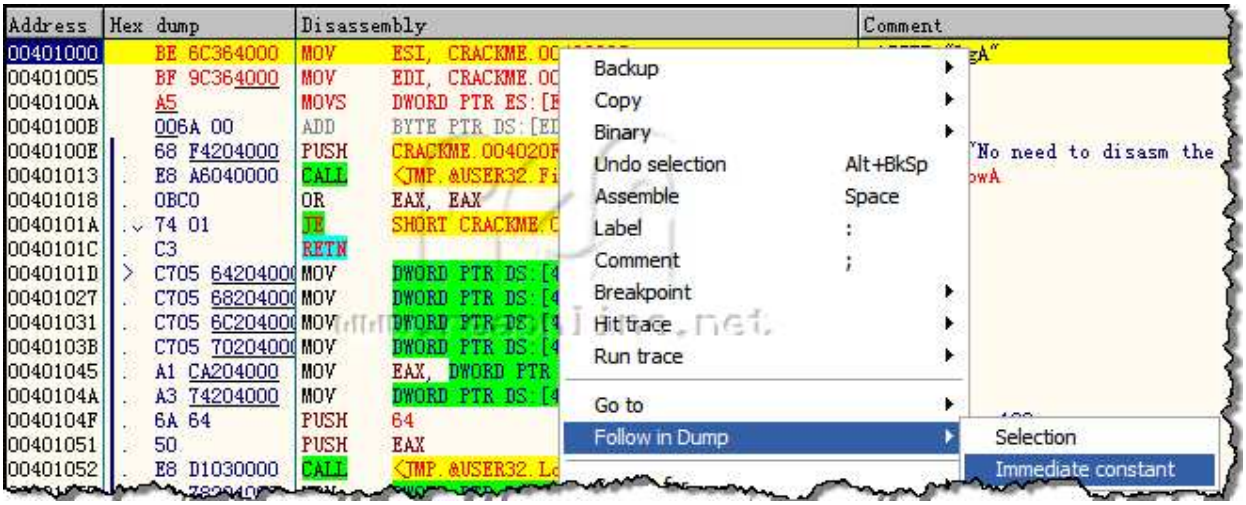
Câu lệnh này sẽ sao chép nội dung dữ liệu được định địa chỉ bởi DS:ESI đến vùng dữ liệu được định địa chỉ bởi ES:EDI. Nội dung của chuỗi gốc không bị thay đổi. Sau khi chuỗi được chuyển cả hai thanh ghi ESI và EDI đều được tự động tăng lên 1 nếu như cờ DF = 0 hay giảm đi nếu DF = 1.

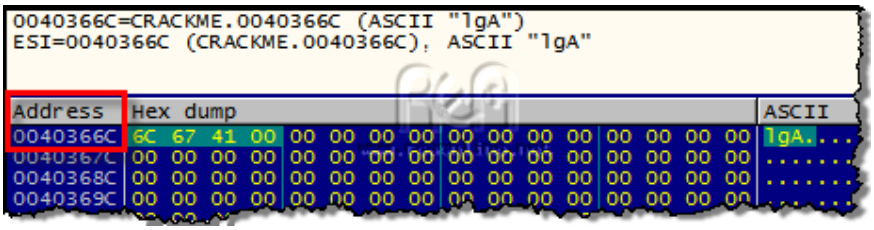
Nếu DF = 0, ESI và EDI được xử lý theo chiều tăng của các địa chỉ trong bộ nhớ : từ trái qua phải trong chuỗi. Ngược lại nếu DF = 1, ESI và EDI được xử lý theo chiều giảm dần các địa chỉ bộ nhớ : từ phải qua trái trong chuỗi.

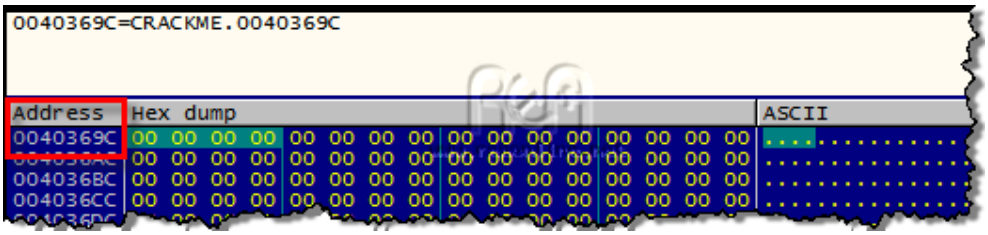
Lệnh MOVS thường không cần bất kì tham số nào, tuy nhiên khi viết trong Olly chúng ta phải chỉ rõ ra như sau : `MOVS DWORD PTR IS:[EDI], DWORD PTR DS:[ESI]`. Lấy một ví dụ nhỏ để minh họa :



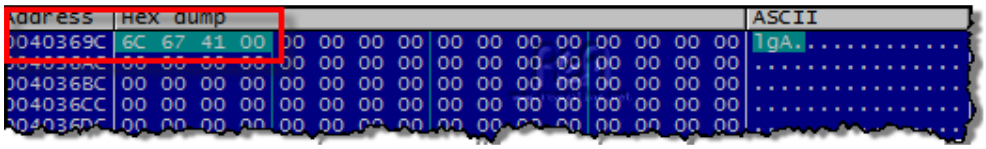
Câu lệnh đầu tiên sẽ khởi tạo giá trị cho ESI và nội dung dữ liệu sẽ được đọc ra từ đó. EDI cũng được khởi tạo một giá trị, và tại đó dữ liệu sẽ được copy vào. Chúng ta có thể xem nội dung tại hai địa chỉ trên bằng cách vào cửa sổ Dump, nhấn Ctrl+G và gõ vào địa chỉ cần xem nội dung hoặc tại cửa sổ CPU, chuột phải chọn **Follow in Dump > Immediate Constant :**







OK vậy là ta đã biết được nội dung của chuỗi gốc và chuỗi đích (Chuỗi gốc là nơi mà ta đọc dữ liệu ra và chuỗi đích là nơi mà ta copy dữ liệu vào) Bây giờ nấn F7 để trace qua lệnh MOVS, quan sát cửa sổ dump ta sẽ thấy được 4bytes của chuỗi gốc được copy vào chuỗi đích :



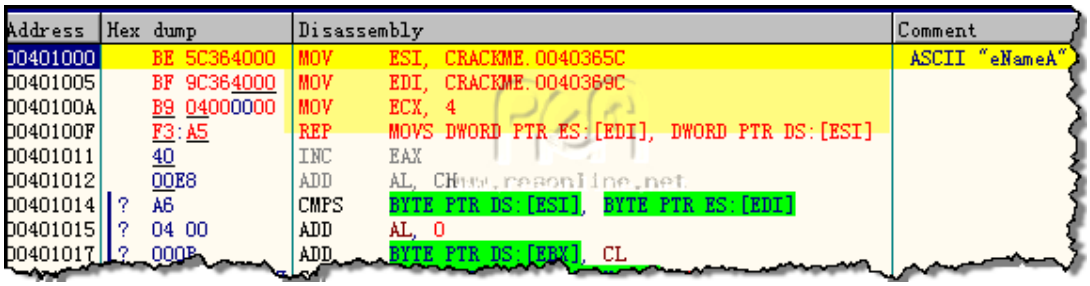
Các bạn tự tìm hiểu thêm các lệnh `MOVSB` và `MOVSW`.

Lệnh REP :

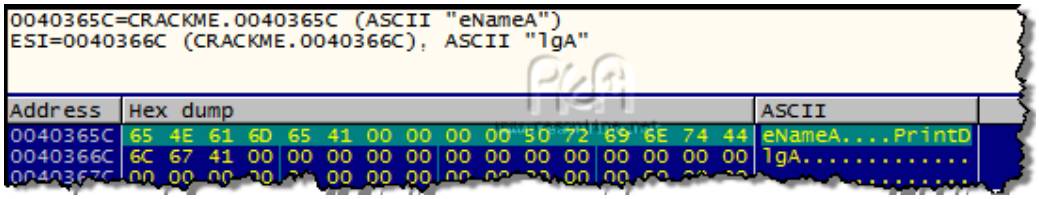
Không giống như lệnh MOVS chỉ chuyển một doubleword tại địa chỉ DS:(E)SI tới địa chỉ ES:(E)DI, lệnh REP cho chúng ta chuyển cả chuỗi . Để chuyển cả chuỗi đầu tiên chúng ta phải khởi tạo thanh ghi (E)CX với số N bằng số doubleword trong chuỗi nguồn và thực hiện lệnh như sau :

```
REP MOVS
```

Lệnh REP có tác dụng làm cho MOVS đươc thực hiện N lần. Sau mỗi lệnh MOVS thì (E)CX được giảm đi 1 cho đến khi nó bằng 0. Tôi lấy ví dụ minh họa như sau :

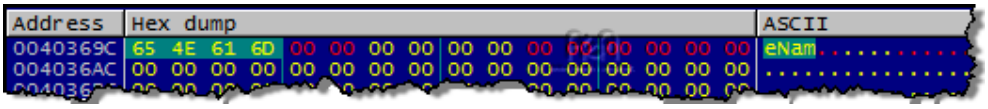


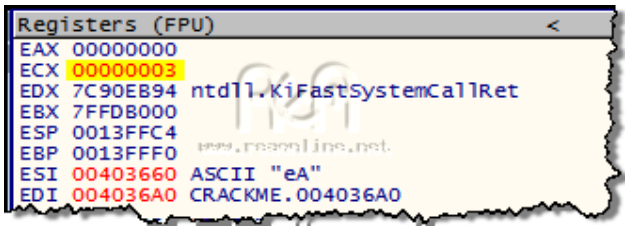
Ta quan sát nội dung tại `0x0040365C` , nơi chứa chuỗi gốc :



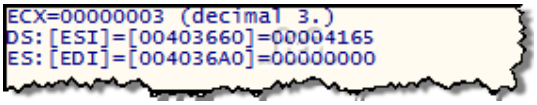
Trong đoạn code trên, bạn đế ý thấy ECX được khởi tạo giá trị là 4. Vậy tức là sẽ có 4 lần copy từ chuỗi nguồn vào chuỗi đích, mỗi lần copy một doubleword. Bây giờ ta Trace từ từ tới lệnh REP :

Các thông tin hiện ra trong Tip Window rất đầy đủ, giờ ta nhấn F7 và quan sát nội dung tại chuỗi đích, tức là tại `0x0040369C`. Tại sao lại nhấn F7 mà không phải là F8, vì nếu các bạn nhấn F8 lệnh REP MOVS sẽ copy thẳng một mạch toàn bộ 4 doubleword từ chuỗi gốc sang chuỗi đích,đồng thời thanh ghi ECX sẽ được giảm về 0. Ta sử dụng F7 để quan sát từng lần copy một :

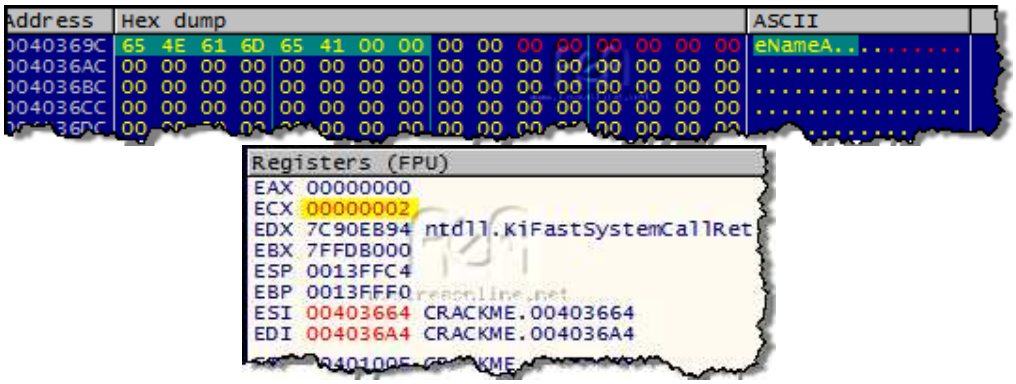




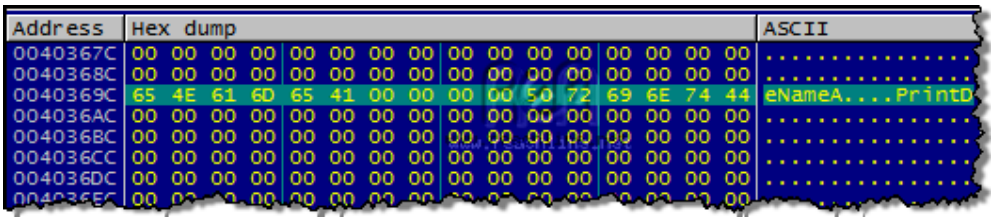
Như quan sát trên hình minh họa, ta thấy 4 bytes đầu tiên được chuyển tới và thanh ghi ECX lúc này bị giảm đi 1 (`ECX = 0x4 -0x1 = 0x3`). Thêm vào đó 2 thanh ghi ESI và EDI được tăng thêm 4 để trỏ vào vị trí tiếp theo :



Tiếp tục nhấn F7 và quan sát :

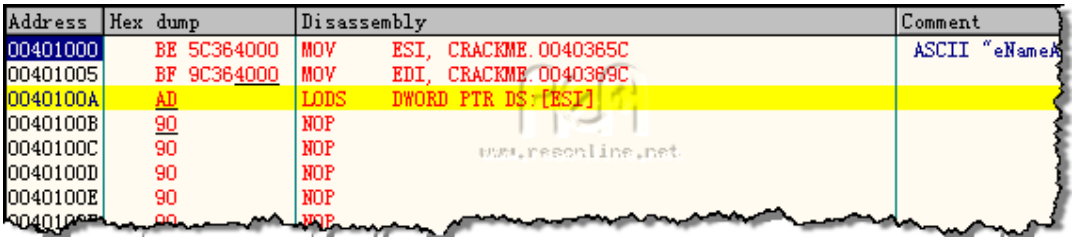


Cuối cùng nhấn F7 cho tới khi ECX = 0x0, lúc này toàn bộ chuỗi gốc đã được copy sang chuỗi đích. Rất rõ ràng và dễ hiểu!



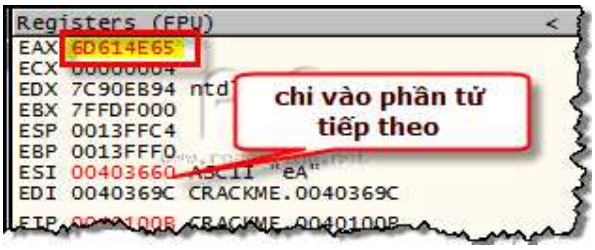
Lệnh LODS :

Lệnh LODS nạp vào EAX một double word do ESI chỉ ra trong đoạn DS, sau đó ESI tự động tăng/ giảm để chỉ vào phần tử tiếp theo tùy theo cờ hướng DF. Ví dụ minh họa :



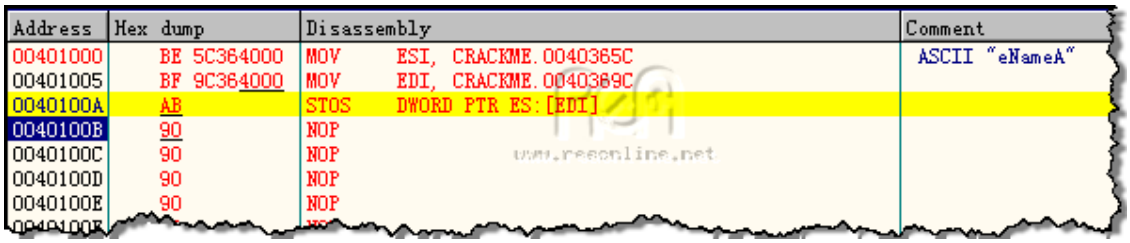
Trace từ từ đến lệnh LODS và quan sát :

Nhấn F7 để thực hiện lệnh LODS, kết quả ta sẽ có được như sau :

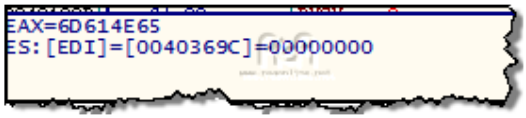


Lệnh STOS :

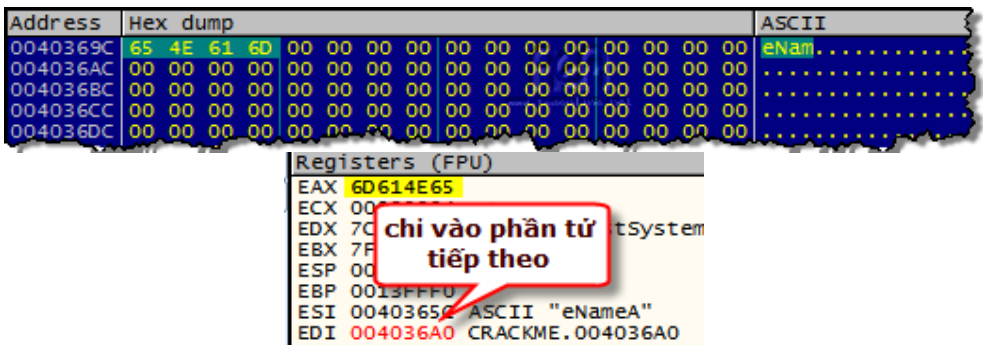
Lệnh này ngược lại với lệnh LODS, có tác dụng chuyển nội dụng của thanh EAX đến vị trí đã được đĩnh nghĩa bởi ES:EDI. Sau khi thực hiện xong thì EDI tự động tăng/giảm để chỉ vào phần tử tiếp theo tùy theo cờ hướng. Ví dụ :



Trace tới lệnh STOS và quan sát cửa sổ TIP :



Sau khi thực hiện lệnh STOS, kết quả ta có được như sau :

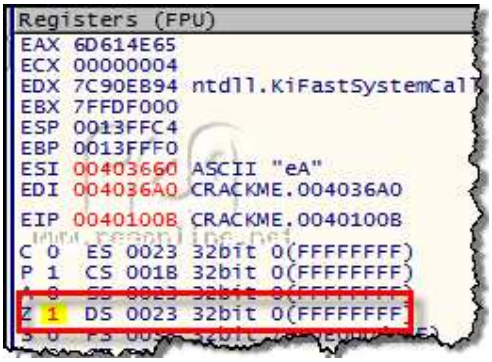


Lệnh CPMS :

Cú pháp : `CPMS Chuỗi đích, Chuỗi gốc`
Lệnh này dùng để so sánh nội dung của ESI với nội dung của EDI. Lệnh này chỉ tạo cờ, không lưu kết quả so sánh, sau khi so sánh các toán hạng không bị thay đổi.

Ta trace tới lệnh CMPS và quan sát :

Về bản chất của lệnh CPMS là lệnh này trừ dw tại địa chỉ DS:ESI cho dw tại địachỉ ES:EDI và thiếp lập cờ. Nếu kết quả của phép trừ bằng 0x0 thì cờ Z được bật cũng đồng nghĩa hai chuỗi này giống nhau :



Ok như vậy qua 8 phần có thể nói tôi đã chỉ cho các bạn đã phần nào nắm được các lệnh cơ bản và quan trọng trong quá trình làm việc với Olly.Các phần tiếp theo của loạt tuts này sẽ tập trung vào các vấn đề khác, do đó các bạn tự tìm hiểu thêm về các lệnh trong các tài liệu về lập trình ASM.

Các chế độ địa chỉ

DIRECT (Trực tiếp) :

Trong chế độ địa chỉ trực tiếp, một toán hạng chứa địa chỉ lệch của ô nhớ dùng chứa dữ liệu còn toán hạng kia chỉ có thể là thanh ghi (không được là ô nhớ). Đây là cách chung được sử dụng để tham chiếu tới một địa chỉ trong bộ nhớ, một số ví dụ minh họa :

```
mov dword ptr [00513450], ecx
mov ax, word ptr [00510A25]
mov al, byte ptr [00402811]
CALL 452200
JMP 421000
```

Trong trường hợp này chúng ta sẽ không gặp vấn đề gì trong việc biểu diễn địa chỉ cũng như biết được nơi mà lệnh nhảy sẽ nhảy tới v..v... Chúng ta chỉ việc chỉ rõ các địa chỉ này trong các lệnh mov, lệnh jmp và cả lệnh Call.

INDIRECT (Gián tiếp) :

Thường được gọi là chế độ địa chỉ gián tiếp qua thanh ghi, trong chế độ địa chỉ này một toán hạng là một thanh ghi được sử dụng để chứa địa chỉ lệch của ô nhớ chứa dữ liệu, còn toán hạng kia chỉ có thể là thanh ghi mà không được là ô nhớ.

```
mov dword ptr [eax], ecx
CALL EAX
JMP [ebx + 4]
```

Như các ban thấy với cơ chế đánh địa chỉ như trên chúng ta sẽ gặp khó khăn trong việc xác định vị trí, không biết được vị trí mà câu lệnh nhảy sẽ nhảy đến, đoạn routine nào sẽ được gọi bởi lệnh call. Chỉ có sử dụng Olly để debug thì chúng ta mới biết được chính xác thông qua việc xem xét các giá trị của các thanh ghi.

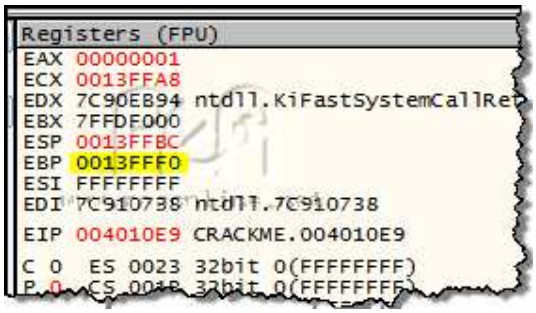
Rất nhiều chương trình sử dụng cơ chế địa chỉ này để ngăn cản hay hạn chế quá trình phân tích chương trình của của cracker/reverser. Lấy một ví dụ để dễ hiểu, khi bạn cho Olly phân tích một chương trình bằng cách load target vào trong Olly, bạn sẽ gặp câu lệnh tương tự như sau :

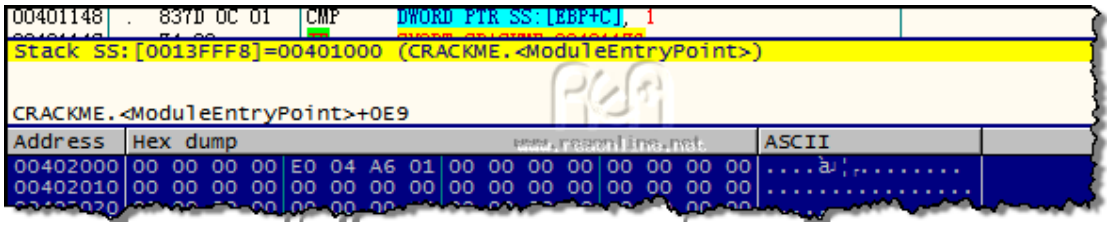
```
PUSH    DWORD PTR SS:[EBP+8]                    ; |hWnd
```

Nếu như bạn không tiến hành debug chương trình, thì thử hỏi khi bạn nhìn vào lệnh trên bạn làm cách nào để biết được nội dung gì sẽ được đẩy lên Stack ?

Câu trả lời như sau, để biết được nội dung gì sẽ được đẩy lên Stack bạn có thể debug trong Olly bằng cách trace từ từ tới câu lệnh trên hoặc đặt một Break point tại câu lệnh này và thực thi chương trình để chương trình dừng tại chỗ mà chúng ta đã set BP. Qua đó chúng ta sẽ quan sát được sự thay đổi và giá trị của thanh ghi EBP, dựa vào thông tin thu được ta sẽ xác định được nôi dung của ô nhớ [EBP + 8].Cụ thể như sau :

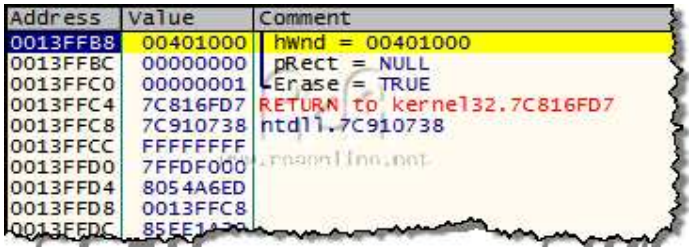
Nhấn F9 để run chương trình, ngay lập tức chúng ta sẽ break tại chỗ mà ta vừa set BP. Quan sát trên cửa sổ Registers, ta thấy được giá trị của EBP là `0x0013FFF0`, vậy EBP + 8 sẽ là `0x0013FFF8`. Cửa sổ tip sẽ cho ta biết nội dung tại ô nhớ `0x0013FFF8` là gì :





Vậy tóm lại lệnh `PUSH [EBP+8]` sẽ PUSH nội dung của `0x0013FFF8` lên Stack. Nếu cẩn thận các bạn có thể vào cửa sổ DUMP, nhấn Ctrl + G, nhập địa chỉ trên vào để quan sát và kiểm tra :

Giờ ta nhấn F7 để thực hiện lệnh Push, đồng thời quan sát cửa sổ Stack để thấy được kết qủa :



Okie tôi nghĩ đến đây là đã đủ cho phần 8, 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 thể không tránh khỏi những sai sót trong quá trình viết, rất mong các bạn góp ý để tôi chỉnh sửa lại. Tôi tin là trải qua 8 bài viết vừa qua tôi đã cung cấp cho các bạn những kiến thức cơ bản nhất để các bạn có thể hiểu và làm việc với Olly. Mặc dù 8 phần này có vẻ hơi nhàm chán với những ai đã có kinh nghiệm trong lập trình ASM tuy nhiên nó lại là những viên gạch đầu tiên cho những người mới làm quen. Những phần tiếp theo của loạt tuts này hứa hẹn sẽ mang đến cho bạn nhiều thông tin bổ ích hơn nữa.Hẹn gặp lại các bạn trong các phần tiếp theo, 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