

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!

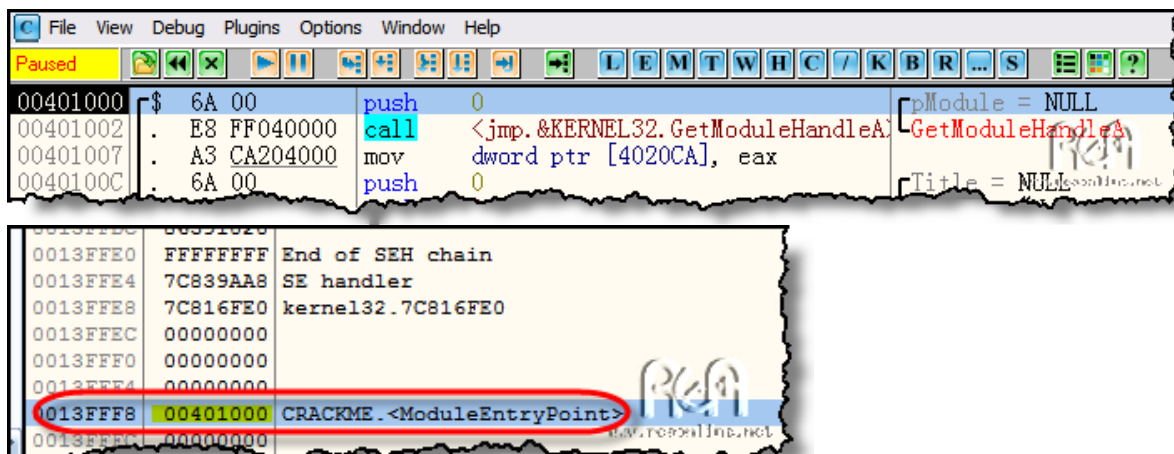
I. Giới thiệu chung

Vậy là chúng ta đã trải qua tám bài viết trong loạt bài viết về OllyDbg, trong tám bài viết này tôi đã hoàn thành phần việc đầu tiên là giới thiệu và giải thích sơ bộ về các lệnh asm thường được sử dụng nhất khi chúng ta làm việc với OllyDbg. Trong các phần tiếp theo tới đây chúng ta sẽ dần dần tiếp cận những kiến thức mới mẻ hơn, sẽ có nhiều đất để cho chúng ta tìm tòi, học hỏi và thực hành. Chúng ta sẽ tìm hiểu dần dần từng phần một một cách chậm rãi, song song với việc đọc lý thuyết thì chúng ta sẽ thực hành luôn những gì chúng ta đã tìm hiểu được và tự bổ sung những mảng mà chúng ta còn khiếm khuyết. Bài viết này tôi sẽ trình bày tới các bạn một số thuật ngữ cơ bản, cách thức làm việc với các hàm APIs, cách patch thông qua các cờ và cuối cùng là các edit trực tiếp code của chương trình. NOW....L3t's GO!!!!!!!!!!

II. Thuật ngữ cơ bản, làm việc với APIs và patch thông qua cờ

Trong phần 9 này chúng ta vẫn tiếp tục sử dụng crackme của CRUEHEAD để demo, Load crackme vào trong Olly chúng ta dừng lại tại entrypoint của Crackme. Vậy **entrypoint** nó là cái gì? Có khá nhiều câu hỏi của các bạn liên quan đến nó, tôi không phải là dân lập trình chính gốc nên tôi hiểu thế nào sẽ giải thích cho các bạn.

Về cơ bản thuật ngữ **EntryPoint (EP)** ám chỉ điểm bắt đầu của một chương trình nơi mà tại đó trở đi chương trình sẽ được thực thi một cách bình thường. Không nên bị nhầm lẫn giữa **EP** và **OEP (Original Entry Point)**, OEP là một thuật ngữ khác mà chúng ta sẽ tìm hiểu ở các phần tiếp theo sau của bộ tuts này. Sau khi chúng đã open một chương trình trong Olly, đợi cho quá trình phân tích kết thúc thì Olly sẽ đưa chúng ta dừng lại tại EntryPoint của chương trình đó.



Cụ thể trong trường hợp của chúng ta, crackme này có EP là 0x401000 và Olly cũng đã chỉ cho chúng ta thấy sau khi analyze crackme trên nó đang dừng lại tại EP như hình minh họa mà các bạn đã thấy ở trên. Hầu hết tất cả các chương trình (tức là khoảng 99% các trường hợp) khi chúng ta load nó bằng Olly thì đều dừng lại tại EP của chương trình đó, ngoài trừ một số trường hợp đặc biệt có sự “**can thiệp**” khiến cho sau khi load chươn trình vào Olly ta lại không dừng lại tại EP, đây cũng là mà thủ thuật đặc biệt mà chúng ta có thể sẽ có dịp tìm hiểu sau này. Còn trong lúc này nó mới chỉ là khái niệm mà thôi ☺, chúng ta còn nhiều thời gian để mò mẫm lắm!

Tiếp theo là một khái niệm khác nữa mà chúng ta cũng cần xem xét đến đó chính là các hàm **Application Programming Interface (APIs)** và thư viện **DLL**.

Lý thuyết cũng như kiến thức về API và DLL các bạn có thể tham khảo quyển **PE File Format** mà tôi đã dịch hoặc các nguồn từ Internet. Theo như hình minh họa ở trên các bạn thấy chỗ khoanh đỏ chính là một lời gọi tới hàm API.

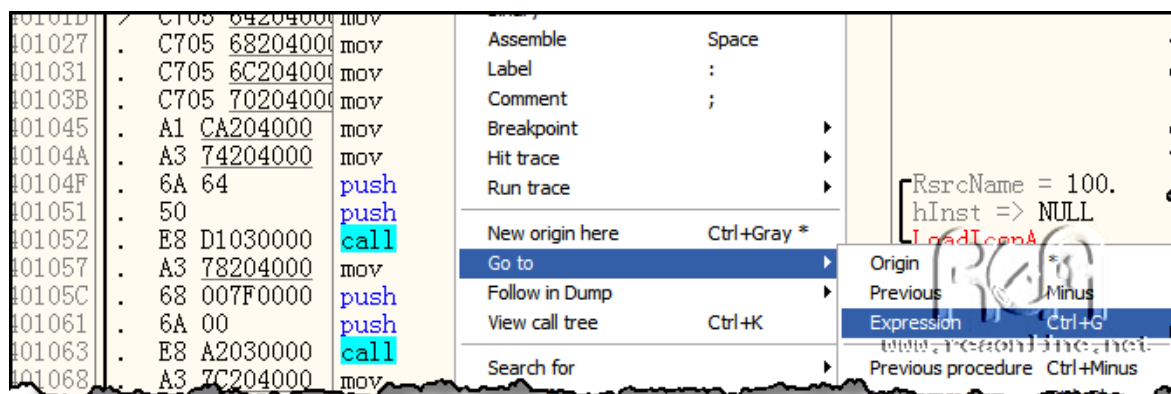
CALL LoadIconA

Có thể nói nôm na về API như sau, hệ điều hành Windows xây dựng nên một tập hợp rất nhiều các hàm/thủ tục, những hàm/thủ tục này sẽ giúp bạn thực hiện những công việc mà bạn phải lặp đi lặp lại hàng ngày, rất nhàm chán trong quá trình coding. Tập hợp những hàm/thủ tục mà Windows xây dựng được đặt cho cái tên chung là API, với sự có mặt của API các lập trình viên không phải phí công sức cho những công việc vốn đã được xây dựng sẵn. Các API này tuy theo nhóm công việc, mục đích thực hiện sẽ được tập hợp vào trong một file thư viện DLL để khi cần đến người lập trình chỉ cần tra từ thư viện đó xem hàm đó có nằm trong thư viện đó không, nếu có thì chỉ việc gọi ra và sử dụng mà thôi.

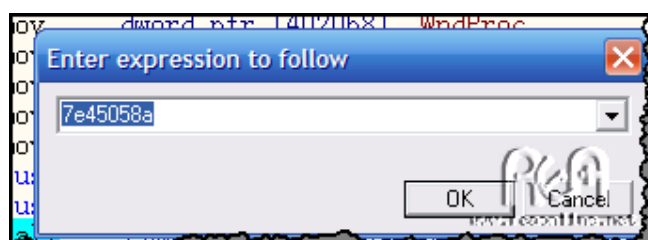
Nhìn vào hình minh họa ở trên, các bạn thấy Olly đã chỉ cho ta thấy hàm **LoadIconA** nằm trong DLL là **User32.dll**.

Ta lấy một ví dụ đơn giản với hàm **MessageBoxA** như sau, tôi không hề biết hàm này nằm ở thư viện dll nào và cũng chẳng biết địa chỉ của nó là gì? Vậy tôi làm thế nào đây để có được thông tin về hàm này, rất đơn giản Olly đã hỗ trợ cho chúng ta khả năng tìm kiếm địa chỉ theo tên hàm. Tại chỗ **Command Bar** của Olly ta gõ tên hàm vào như sau :

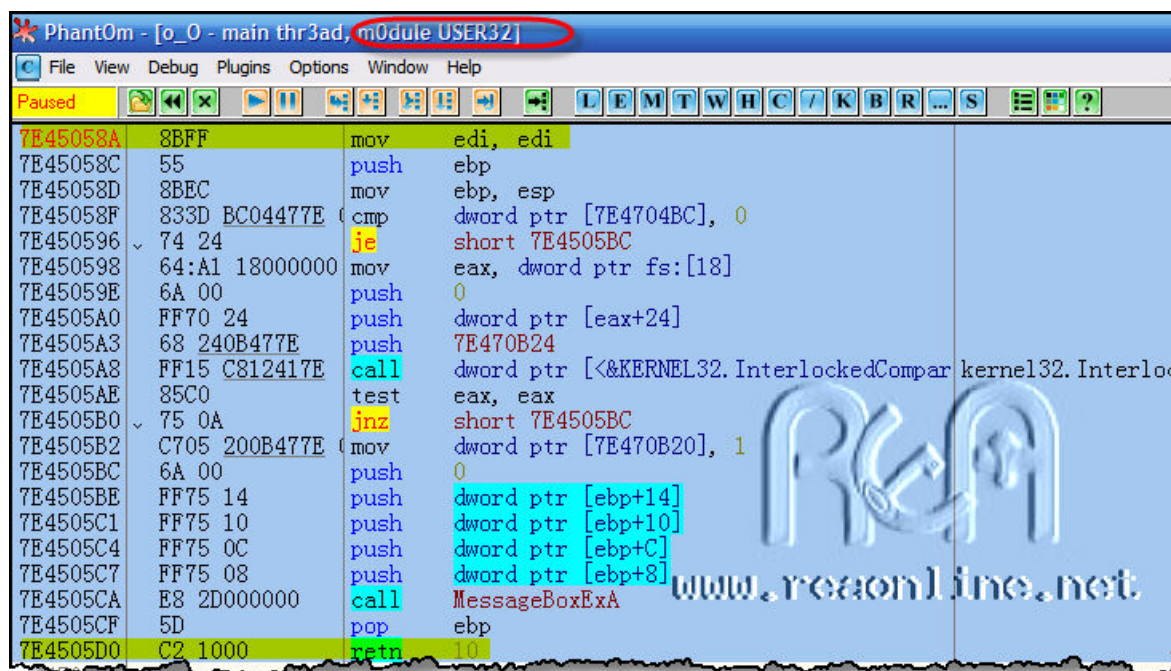
Wow, ngay lập tức Olly tìm cho ta ngay địa chỉ của hàm **MessageBoxA**, bây giờ ta đi tới địa chỉ này để xem hàm mà chúng ta tìm nằm trong thư viện nào. Tại Olly, nhấn chuột phải và chọn **Go to > Expression :**



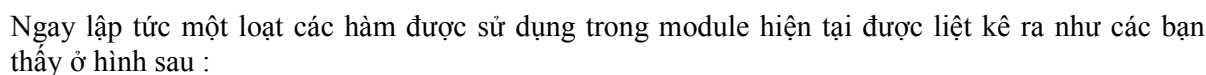
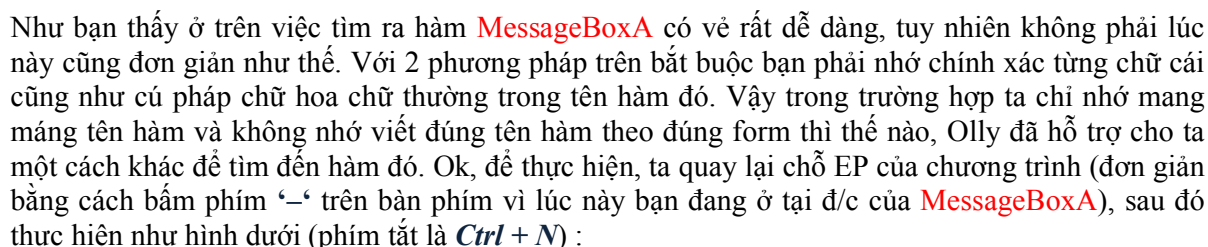

Nhập địa chỉ đã tìm được vào textbox và nhấn OK :



Olly sẽ đưa ta tới địa chỉ của hàm **MessageBoxA** :



Theo như hình trên thì ta thấy ngay rằng hàm **MessageBoxA** thuộc về thư viện DLL là **User32.dll**. Hàm này bắt đầu tại 0x7e45058a và kết thúc bằng lệnh **Retn 10** tại 0x7e4505d0.



Address	Section	Type	Name	Comment
004031E4	.idata	Import	USER32.BeginPaint	
00403230	.idata	Import	KERNEL32.CloseHandle	
00403250	.idata	Import	COMCTL32.CreateToolBar	
0040324C	.idata	Import	COMCTL32.CreateToolBarEx	
004031E8	.idata	Import	USER32.CreateWindowExA	
004031EC	.idata	Import	USER32.DefWindowProcA	
00403278	.idata	Import	GDI32.DeleteDC	
00403274	.idata	Import	GDI32.DeleteObject	
004031F0	.idata	Import	USER32.DialogBoxParamA	
004031F4	.idata	Import	USER32.DispatchMessageA	
004031F8	.idata	Import	USER32.DrawMenuBar	
004031FC	.idata	Import	USER32.EndDialog	
00403270	.idata	Import	GDI32.EndDoc	
0040326C	.idata	Import	GDI32.EndPage	
00403200	.idata	Import	USER32.EndPaint	
00403240	.idata	Import	KERNEL32.ExitProcess	
00403204	.idata	Import	USER32.FindWindowA	
00403208	.idata	Import	USER32.GetDC	
0040320C	.idata	Import	USER32.GetDlgItem	
00403210	.idata	Import	USER32.GetDlgItemTextA	
0040321C	.idata	Import	KERNEL32.GetLocalTime	
00403214	.idata	Import	USER32.GetMessageA	
00403238	.idata	Import	KERNEL32.GetModuleHandleA	
00403284	.idata	Import	COMCTL32.GetOpenFileNameA	

Nhìn như trên thì rồi quá ta không biết phải mò ra **MessageBoxA** ở đâu trong một rừng tên như thế này, để tìm kiếm được đúng hàm cần tìm trước tiên tại chính cửa sổ trên ta gõ chữ cái đầu của tên hàm mà cụ thể ở đây là chữ **M**. Olly sẽ đưa chúng ta đến vị trí của những hàm bắt đầu bằng chữ **M**

0040318C	.idata	Import	USER32.LoadCursorA
004031A0	.idata	Import	USER32.LoadIconA
004031C8	.idata	Import	USER32.LoadMenuA
0040319C	.idata	Import	USER32.LoadStringA
0040322C	.idata	Import	KERNEL32.lstrlen
00403194	.idata	Import	USER32.MessageBeep
004031AC	.idata	Import	USER32.MessageBoxA
00401000	CODE	Export	<ModuleEntryPoint>
004031C0	.idata	Import	USER32.MoveWindow
00403220	.idata	Import	KERNEL32.OpenFile
004031B0	.idata	Import	USER32.PostQuit2

Tiếp tục gõ những chữ cái tiếp theo trong tên hàm Olly sẽ đưa ta đến đúng vị trí cần tìm :

Phantom - [Find: MESSAGEBOX]			
File View Debug Plugins Options Window Help			
Paused [Icons]			
Address	Section	Type	Name
0040321C	.idata	Import	KERNEL32.GetLocalTime
00403214	.idata	Import	USER32.GetMessageA
00403238	.idata	Import	KERNEL32.GetModuleHandleA

0040322C	.idata	Import	KERNEL32.lstrlen
00403194	.idata	Import	USER32.MessageBeep
004031AC	.idata	Import	USER32.MessageBoxA
00401000	CODE	Export	<ModuleEntryPoint>
004031C0	.idata	Import	USER32.MoveWindow
00403220	.idata	Import	KERNEL32.OpenFile

Tại hàm tìm được ta nhấn chuột phải và chọn **Follow import in Disassembler** :

0040322C	.idata	Import	KERNEL32.lstrlen
00403194	.idata	Import	USER32.MessageBeep
004031AC	.idata	Import	USER32.MessageBoxA
00401000	CODE	Export	<ModuleEntryPoint>
004031C0	.idata	Import	USER32.MoveWin
00403220	.idata	Import	KERNEL32.OpenF
004031B0	.idata	Import	USER32.PostQui
00403288	.idata	Import	COMDLG32.Print
0040323C	.idata	Import	KERNEL32.ReadF
004031E0	.idata	Import	USER32.Registe

7E45058A	8BFF	mov	edi, edi
7E45058C	55	push	ebp
7E45058D	8BEC	mov	ebp, esp
7E45058F	833D BC04477E	cmp	dword ptr [7E4704BC], 0
7E450596	74 24	je	short 7E4505BC
7E450598	64:A1 18000000	mov	eax, dword ptr fs:[18]
7E45059F	6A 00	mov	byte ptr [edi], al

Ok, vậy là chúng ta đã trải qua một số phương pháp khác nhau để tìm kiếm thông tin về một hàm API, bây giờ chúng ta tiếp tục trở lại với phần tiếp theo của bài viết. Sau khi tìm kiếm được thông tin về hàm **MessageBoxA** như hình minh họa ở trên, ta tiến hành đặt một điểm ngắt hay còn gọi với một thuật ngữ là **Break Point (BP)** . Ta làm như sau :

004031AC	.idata	Import	USER32.MessageBoxA
00401000	CODE	Export	<ModuleEntryPoint>
004031C0	.idata	Import	USER32.MoveWind
00403220	.idata	Import	KERNEL32.OpenFi
004031B0	.idata	Import	USER32.PostQuit
00403288	.idata	Import	COMDLG32.PrintD
0040323C	.idata	Import	KERNEL32.ReadFi
004031E0	.idata	Import	USER32.Register
004031D0	.idata	Import	USER32.SendMess

Việc thiết lập BP như trên cũng tương tự với cách làm khác như sau, tại cửa sổ **Command Bar** ta gõ vào : **Bp MessageBoxA**

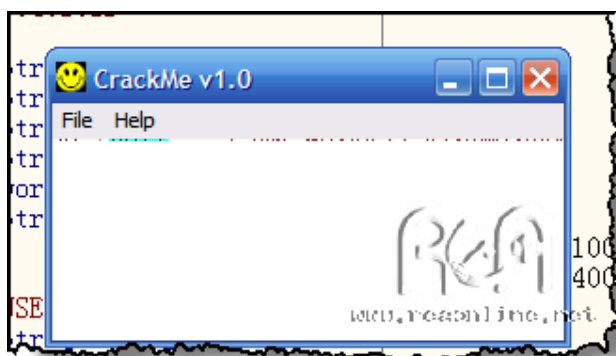
00402030	00000000	
00402034	00000000	
00402038	00000000	

Command **bp MessageBoxA** BP address, string -- Break with condition
Start:402000 End:401FFF Value:0

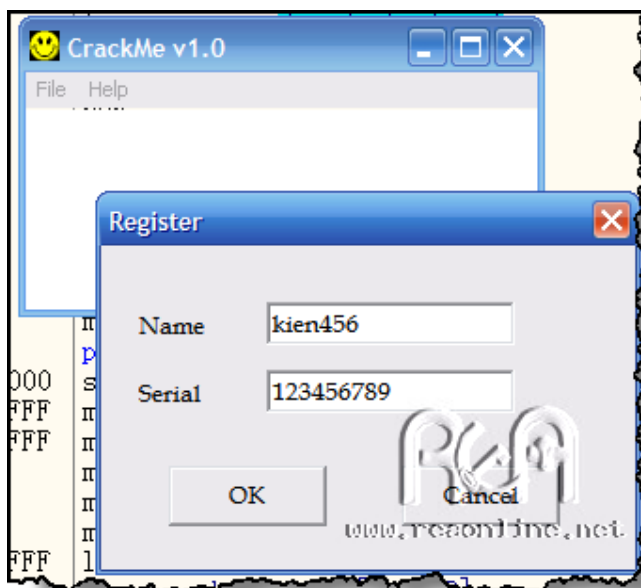
Ok ta vừa mới đặt BP, giờ ta kiểm tra xem kết quả ta đặt như thế nào. Chuyển qua cửa sổ BP bằng cách nhấn phím tắt (**Alt + B**) :

Address	Module	Active	Disassembly
7E45058A	USER32	Always	mov edi, edi

Như bạn thấy ở trên, tôi đã đặt một BP tại địa chỉ bắt đầu của hàm **MessageBoxA**, bây giờ khi tôi cho thực thi crackme này trong Olly nếu như có bất kì một thông báo nào bắn ra thì ta sẽ dừng lại tại vị trí mà ta đã đặt BP. Để kiểm nghiệm điều này, ta tiến hành thực thi crackme bằng cách nhấn **F9** :



Vào menu Help và chọn Register, cửa sổ yêu cầu nhập User Name và Serial hiện ra :



Ta nhập thử Fake info vào những text box, sau đó nhấn Ok. Ngay lập tức Olly sẽ dừng lại và dừng đúng chỗ mà chúng ta đặt BP :

7E45058A	8BFF	mov	edi, edi
7E45058C	55	push	ebp
7E45058D	8BEC	mov	ebp, esp
7E45058F	833D BC04477E	cmp	dword ptr [7E45058F], 0

Vậy ta đoán ngay lúc ta nhấn Ok sẽ có một thông báo bắn ra, tuy nhiên ta đã cho Olly bắt hành động này nên Olly đã dừng lại tại đầu hàm. Bây giờ ta chuyển qua cửa sổ **Stack** ta sẽ có được những thông tin sau :

Address	Value	Comment
0013FE8C	004013C1	CALL to MessageBoxA from CRACKME.004013BC
0013FE90	006E057E	hOwner = 006E057E ('CrackMe v1.0', class='No need to disasm the code')
0013FE94	00402169	Text = "No luck there, mate!"
0013FE98	00402160	Title = "No luck!"
0013FE9C	00000030	Style = MB_OK MB_ICONEXCLAMATION MB_APPLMODAL
0013FEA0	00401232	RETURN to CRACKME.00401232 from CRACKME.0040137E
0013FEA4	0040218E	ASCII "KIEN456"

Theo thông tin mà hình cung cấp các bạn có thể thấy rằng mỗi hàm Api trước khi chuẩn bị được gọi thì các tham số của hàm sẽ được đẩy lên Stack. Các tham số này bạn có thể tham khảo tại file Help là **Win32.hlp**. Ok giờ tại cửa sổ Stack ta chọn như sau :

Address	Value	Comment
0013FE8C	004013C1	CALL to MessageBoxA from CRACKME.004013BC
0013FE90	006E057E	hOwner = 006E057E ('CrackMe v1.0', class='No need to disasm the code')
0013FE94	00402169	Text = "No luck there, mate!"
0013FE98	00402160	Title = "No luck!"
0013FE9C	00000030	Style = MB_OK MB_ICONEXCLAMATION MB_APPLMODAL
0013FEA0	00401232	RETURN to CRACKME.00401232 from CRACKME.0040137E
0013FEA4	0040218E	ASCII "KIEN456"
0013FEA8	00000000	
0013FEAC	0013FF1C	
0013FEB0	00401128	RETURN to CRACKME.00401128 from CRACKME.0040137E
0013FEB4	0013FEE0	
0013FEB8	7E418734	RETURN to USER32.ExitProcess from CRACKME.0040137E
0013FEBE	006E057E	
0013FEC0	00000111	
0013FEC4	00000066	

Ta sẽ quay lại cửa sổ code của chương trình và dừng tại vị trí sau :

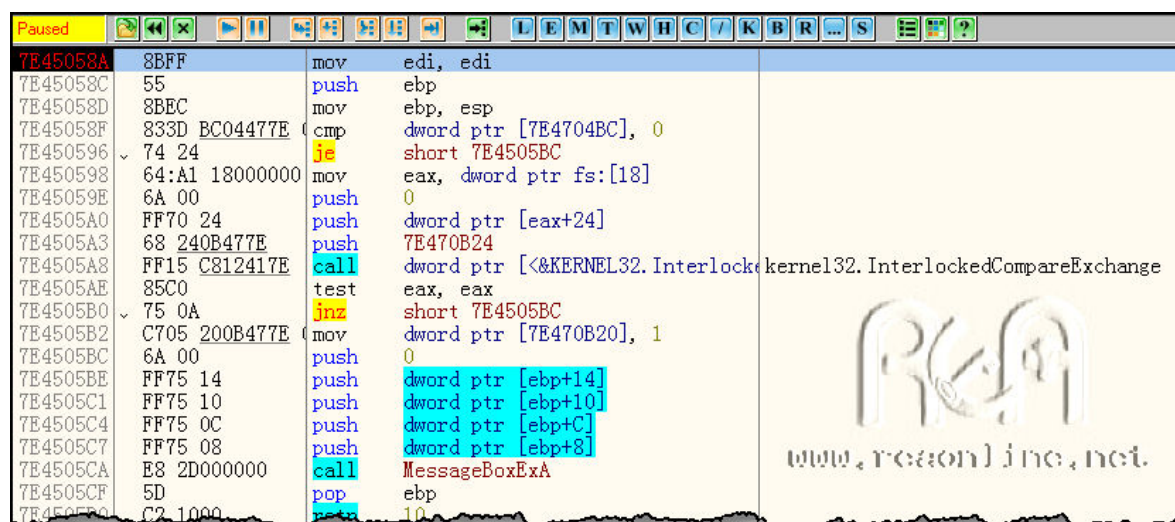
004013AA	EB 15	jmp	short 004013C1
004013AC	5E	pop	esi
004013AD	6A 30	push	30
004013AF	68 60214000	push	00402160
004013B4	68 69214000	push	00402169
004013B9	FF75 08	push	dword ptr [ebp+8]
004013BC	E8 79000000	call	<jmp.&USER32.MessageBoxA>
004013C1	C3	retn	
004013C7	33FF	xor	edi, edi

Theo như lý thuyết về hai lệnh CALL và RET mà tôi đã giới thiệu ở phần trước thì chúng ta sẽ không ngạc nhiên lắm khi ta Follow theo địa chỉ trên thì Olly lại đưa ta đến lệnh Ret mà không phải là lệnh Call.

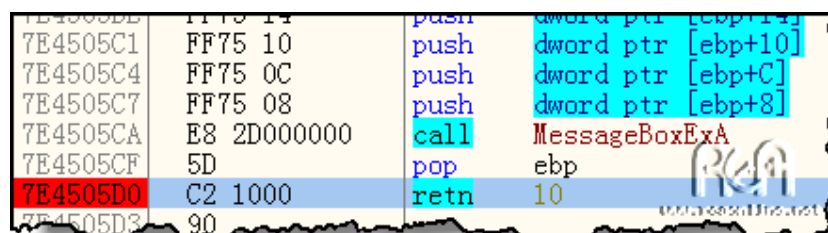
Ok vậy là như các bạn đã thấy, khi thông tin về User name và Serial mà chúng ta nhập vào không đúng thì chúng ta sẽ nhận được một thông báo với nội dung như sau :

```
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
```

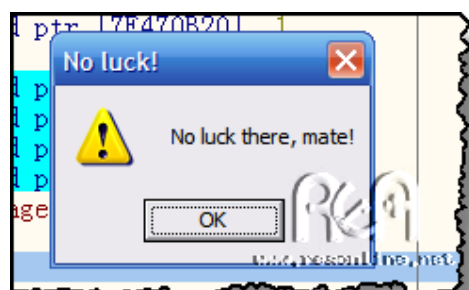
Giờ ta quay trở lại chỗ đặt BP để kiểm chứng thông tin mà ta vừa nói ở trên.



Ta đặt một BP tại lệnh Ret 10 :



Sau đó nhấn F9 để thực thi chương trình, chúng ta nhận được thông báo xong :



Khả khả đúng như thông tin mà ta có được ở trên, giờ ta nhấn OK ngay lập tức sẽ dừng lại tại lệnh Ret 10 :

7E4505C7	FF75 08	push	dword ptr [ebp]
7E4505CA	E8 2D000000	call	MessageBoxExA
7E4505CF	5D	pop	ebp
7E4505D0	C2 1000	retn	10
7E4505D3	90	nop	
7E4505D4	90	nop	

Lệnh Ret 10 này cho ta biết ta đang ở điểm kết thúc của hàm API **MessageBoxA**, khi lệnh này được thực thi thì ta sẽ quay trở lại đoạn code chính của chương trình. Nhưng trước khi thực hiện lệnh này, ta nhìn qua cửa sổ Stack sẽ có được thông tin địa chỉ mà khi thực hiện lệnh Ret 10 ta sẽ quay về đó :

Address	Value	Comment
0013FE8C	004013C1	RETURN to CRACKME.004013C1 from <jmp.&USER32.MessageBoxA>
0013FE90	006E057E	
0013FE94	00402169	ASCII "No luck there, mate!"
0013FE98	00402160	ASCII "No luck!"
0013FE9C	00000030	
0013FEA0	00401232	RETURN to CRACKME.00401232 from CRACKME.0040137E
0013FEA4	0040218E	ASCII "KIEN456"
0013FEA8	00000000	

Địa chỉ mà tôi khoanh đỏ ở trên chính là địa chỉ của lệnh bên dưới lời gọi tới hàm **MessageBoxA**. Ta nhấn **F7** để trace qua lệnh Retn 10, khi đó ta sẽ trở về địa chỉ 0x004013C1 nhưng đồng thời khi ta thực hiện lệnh này thì thanh ghi Esp cũng tự động được cộng thêm 0x10 vào, tức là $ESP = ESP + 0x10 = 0x0013FE90 + 0x10 = 0x0013FEA0$. Ok, sau khi nhấn F7 như đã nói ta sẽ tới đây :

004013AC	>	5E	pop	esi	
004013AD	.	6A 30	push	30	
004013AF	.	68 60214000	push	00402160	
004013B4	.	68 69214000	push	00402169	
004013B9	.	FF75 08	push	dword ptr [ebp+8]	
004013BC	.	E8 79000000	call	<jmp.&USER32.MessageBoxA>	
004013C1	>	C3	retn		

Để ý cửa sổ Stack :

Address	Value	Comment
0013FEA0	00401232	RETURN to CRACKME.00401232 from CRACKME.0040137E
0013FEA4	0040218E	ASCII "KIEN456"
0013FEA8	00000000	
0013FEAC	0013FF1C	
0013FEB0	00401128	RETURN to CRACKME.WndProc from <jmp.&KERNEL32.ExitProcess>

Như hình trên ta đang ở 0x004013C1, bên trên nó là một lời gọi tới hàm **MessageBoxA**, hình này cho chúng ta biết được chúng ta đã nhập thông tin về Name và Serial bị sai cho nên thông báo "No luck..." sẽ bắn ra!! Bây giờ ta tiếp tục nhấn F9 thêm một lần nữa :

7E45058A	8BFF	mov	edi, edi	
7E45058C	55	push	ebp	
7E45058D	8BEC	mov	ebp, esp	
7E45058F	833D BC04477E	cmp	dword ptr [7E4704BC], 0	
7E450596	74 24	je	short 7E4505BC	
7E450598	64:A1 18000000	mov	eax, dword ptr fs:[18]	
7E45059E	6A 00	push	0	
7E4505A0	FF70 24	push	dword ptr [eax+24]	
7E4505A3	68 240B477E	push	7E470B24	
7E4505A8	FF15 C812417E	call	dword ptr [&KERNEL32.InterlockedKernel32.	
7E4505AE	85C0	test	eax, eax	
7E4505B0	75 0A	jnz	short 7E4505BC	

Bùm...ta lại break tại MessageBoxA, tiếp tục dòm qua cửa sổ Stack :

Address	Value	Comment
0013FE8C	0040137D	CALL to MessageBoxA from CRACKME.00401378
0013FE90	006E057E	hOwner = 006E057E ('CrackMe v1.0',class='No need to disasm the code!')
0013FE94	00402169	Text = "No luck there, mate!"
0013FE98	00402160	Title = "No luck!"
0013FE9C	00000030	Style = MB_OK MB_ICONEXCLAMATION MB_APPLMODAL
0013FEA0	0040124A	RETURN to CRACKME.0040124A from CRACKME.00401362
0013FEA4	0040218E	ASCII "KIEN"

Chuột phải tại dòng đầu tiên và chọn *Follow in Disassembler* :

Address	Value	Comment
0013FE8C	0040137D	CALL to Me
0013FE90	006E057E	hOwner = 0
0013FE94	00402169	Text = "No
0013FE98	00402160	Title = "N
0013FE9C	00000030	Style = MB
0013FEA0	0040124A	RETURN to C
0013FEA4	0040218E	ASCII "KIEN
0013FEA8	00000000	
0013FEAC	0013FF1C	
0013FEB0	00401128	RETURN to C
0013FEB4	0013FEE0	
0013FEB8	7E418734	RETURN to U
0013FEBE	006E057E	
0013FEC0	00000111	
0013FEC4	00000066	

Oly đưa ta đến địa chỉ 0x0040137D, bên trên tại 0x00401378 tiếp tục là một lời gọi tới hàm **MessageBoxA** :

00401369	6A 30	push	30	Style = MB_OK MB_ICONEXCLAMATION MB_APPLMODAL Title = "No luck!" Text = "No luck there, mate!" hOwner = 006E057E ('CrackMe v1.0',class='No need to disasm the code!')
0040136B	68 60214000	push	00402160	
00401370	68 69214000	push	00402169	
00401375	FF75 08	push	dword ptr [ebp+8]	
0040137A	E8 BD000000	call	<jmp.&USER32.MessageBoxA>	
0040137D	C3	ret		

Như vậy, tổng kết lại chúng ta thấy rằng có hai đoạn code đều Show ra cái Nag là "No luck...", vậy ta phỏng đoán rằng vậy chúng ta sẽ có hai đoạn check liên quan đến Username và Serial nhập

vào. Có thể cái Nag đầu tiên mà chúng ta nhận là cái Nag liên quan tới việc Check Name, còn cái Nag tiếp theo mà chúng ta thấy ở trên hình là cái Nag liên quan đến check Serial ☺. Chà chà có vẻ mệt đây!!

Tại vị trí trên, dịch lên một chút bạn sẽ thấy có thêm một lời gọi tới hàm **MessageBoxA** nữa :

0040134D	6A 30	push	30	[Style = MB_OK MB_ICONEXCLAMATION MB_APPLMODAL Title = "Good work!" Text = "Great work, mate!", CR, "Now try the next hOwner MessageBoxA
0040134F	68 29214000	push	00402129	
00401354	68 34214000	push	00402134	
00401359	FF75 08	push	dword ptr [ebp+8]	
0040135C	E8 D9000000	call	<jmp.&USER32.MessageBoxA>	MessageBoxA
00401361	C3	retn		
00401362	6A 00	push	0	[BeepType = MB_OK MessageBeep Style = MB_OK MB_ICONEXCLAMATION MB_APPLMODAL Title = "No luck!" Text = "No luck there, mate!" hOwner MessageBoxA
00401364	E8 AD000000	call	<jmp.&USER32.MessageBeep>	
00401369	6A 30	push	30	
0040136B	68 60214000	push	00402160	
00401370	68 69214000	push	00402169	
00401375	FF75 08	push	dword ptr [ebp+8]	
00401378	E8 BD000000	call	<jmp.&USER32.MessageBoxA>	MessageBoxA
0040137D	C3	retn		

Hình trên sẽ cho ta biết có 2 Nag liên quan đến việc nhập Serial, nếu ta nhập đúng thì hiện thông báo ở chỗ được tô vàng, nếu nhập sai thì sẽ hiện thông báo ở chỗ được tô xanh. Để ý trong hình trên ta thấy được có dấu '\$', dấu này thông báo cho chúng ta biết ta đang ở trong thân của một lời gọi hàm/thủ tục, vậy để biết được lời gọi này xuất phát ở đâu chúng ta chỉ việc chọn dòng có chứa dấu \$ và nhìn xuống cửa sổ bên dưới :

00401362	\$ 6A 00	push	0	[BeepType = MB_OK MessageBeep Style = MB_OK MB Title = "No luck Text = "No luck hOwner MessageBoxA
00401364	E8 AD000000	call	<jmp.&USER32.MessageBeep>	
00401369	6A 30	push	30	
0040136B	68 60214000	push	00402160	
00401370	68 69214000	push	00402169	
00401375	FF75 08	push	dword ptr [ebp+8]	
00401378	E8 BD000000	call	<jmp.&USER32.MessageBoxA>	MessageBoxA
0040137D	C3	retn		
0040137E	\$ 8B7424 04	mov	esi, dword ptr [esp+4]	[www.reconline.net]
00401382	56	push	esi	
00401383	> 8A06	mov	al, byte ptr [esi]	
00401385	84C0	test	al, al	
00401387	< 74 13	je	short 0040139C	
Local call from 00401245				

Vậy là Olly đã giúp chúng ta biết được địa chỉ nơi mà có lời gọi gọi tới đoạn code trên đó chính là tại 0x00401245, nhấn chuột phải tại dòng tô màu xanh ở trên và chọn **Go to Call from 00401245** :

00401387	< 74 13	je	short 0040139C
Local call from 00401245			
<div> Copy pane to clipboard Go to CALL from 00401245 Appearance </div>			
00402000	00 00 00		


```

00401241 . 3BC3      cmp     eax, ebx
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
00401253 . C8 000000  enter   0, 0
00401257 . 53         push    ebx
00401259 . 56         push    esi

```

Hmm, có vẻ như chúng ta đang đứng tại vị trí chứa đoạn code so sánh. Vậy ta lập luận như sau, nếu kết quả so sánh là đúng thì chúng ta sẽ nhảy tới địa chỉ 0x0040124C và thực hiện lời gọi hàm tại địa chỉ này, mà theo hình trên thì lời gọi này sẽ hiện thông báo “Greate work...”, còn ngược lại nếu kết quả so sánh là sai thì ta sẽ đi tới lời gọi tại 0x00101245 và thực hiện lời gọi hàm hiện thông báo “No luck ...”. Ta đặt thử một BP tại lệnh JE :

```

00401241 . 3BC3      cmp     eax, ebx
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

```

Xóa hết các BP liên quan đến API **MessageBoxA** đi, mở cửa sổ Break Point (**Alt + B**) :

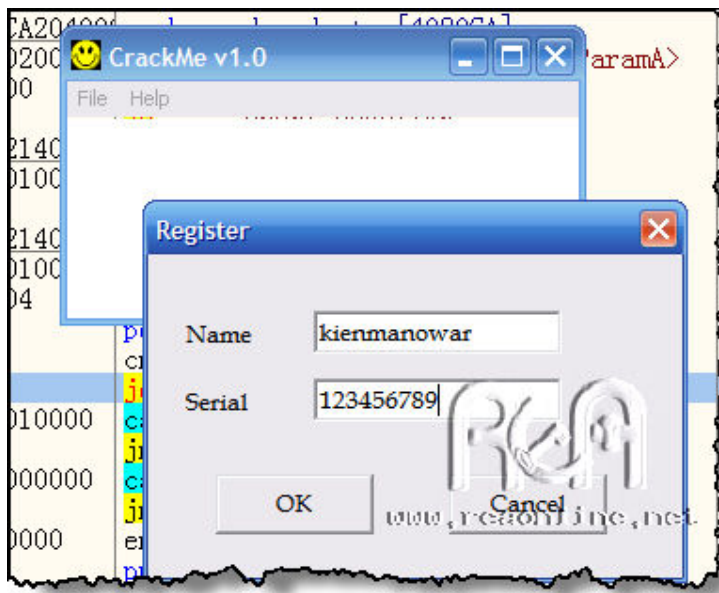


Address	Module	Active	Disassembly
00401243	CRACKME	Always	je short 0040124C
7E45058A	USER32	Always	mov edi, edi
7E4505D0	USER32	Always	retn

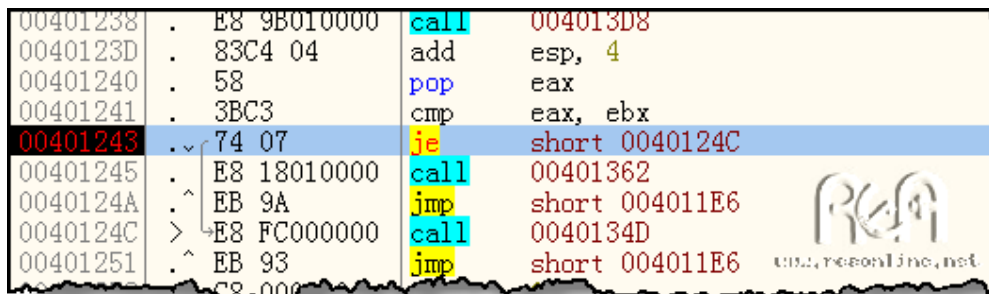
Ta loại bỏ hai BP mà ta đặt tại Module User32 đi chỉ để lại BP mà ta đặt tại lệnh JE, để loại bỏ một BP thì chỉ việc nhấn chuột phải tại BP đó và chọn Remove hoặc nhấn phím tắt Del.

Address	Module	Active	Disassembly
00401243	CRACKME	Always	je short 0040124C

Sau khi remove BP chỉ để lại một BP duy nhất như hình trên, ta nhấn F9 để thực thi chương trình. Lúc này cái Nag “No luck...” sẽ xuất hiện, lý do là vì lúc trước ta set BP tại **MessageBoxA** nhưng ta vừa bỏ đi rồi nên nhận cái Nag đó là đúng thôi ☺. Nhấn Ok, sau đó tiến hành nhập lại thông tin về Name và Serial, lần này ta nhập thử một cái name khác thử xem :



Nhấn OK, ta sẽ dừng lại tại BP. Oh... như vậy kết luận sơ bộ ban đầu của tôi về việc crackme này có tới hai chỗ check liên quan tới Name và Serial là đúng. Vì nếu như bạn nhập Name như tôi nhập lần đầu ở trên thì khi chúng ta nhấn Ok chúng ta sẽ nhận Nag "No luck..." trước khi chúng ta break tại điểm đặt BP. Sau khi chúng ta nhấn Ok tại Nag thì chúng ta mới dừng lại tại BP mà ta đã set :



Chúng ta thấy rằng dựa theo kết quả so sánh ở lệnh CMP, nếu như giá trị của eax không bằng giá trị của ebx thì lệnh nhảy sẽ không được thực hiện. Khi lệnh nhảy không được thực hiện thì lệnh Call bên dưới nó tại địa chỉ 0x00401245 sẽ được thực hiện tiếp theo, và chúng ta biết rằng lệnh CALL 401362 chính là hàm show ra Nag là "No luck..", nếu như bạn không nhớ thì bạn chọn lệnh Call đó và nhấn Enter để Follow :



Như chúng ta thấy lệnh JE không được thực hiện vì Serial mà chúng ta nhập vào là sai. Ngoài ra ta biết rằng lệnh nhảy JE này phụ thuộc vào cờ Z, vậy muốn cho lệnh này thực hiện thì ta có một mẹo nhỏ là thay đổi giá trị của cờ bằng cách nhấp đúp vào cờ Z :

```

C 1 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 1 FS 003B 32bit 7FFDD000(FFF)
T 0 GS 0000 32bit 0(FFFFFFFF)

```

Theo hình trên thì cờ Z đã chuyển thành 1, điều này đồng nghĩa với việc là giá trị của thanh ghi eax bằng giá trị của thanh ghi ebx, cũng đồng nghĩa với nếu như thực hiện lệnh Cmp thì lệnh này sẽ thực hiện phép trừ hai toán hạng bằng nhau cho ra kết quả là 0. Mà khi kết quả bằng 0 thì cờ Z được bật lên ☺ và chúng ta cùng “nhảy” lolz ... như hình minh họa dưới đây :

```

00401240 . 58          pop     eax
00401241 . 3BC3        cmp     eax, ebx
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
00401253 . C8 000000   enter   0, 0
00401257 . 53          push    ebx
00401258 . 55          push    esi

```

Vậy là nếu ta nhấn F8 để trace thì lệnh JE sẽ đưa chúng ta tới lệnh tiếp theo tại địa chỉ 0x0040124C. Ta biết rằng lệnh Call tại địa chỉ này chính là show Nag “Great work ...”, nếu bạn không nhớ hãy thử Follow tại lệnh Call này bằng cách nhấn Enter :

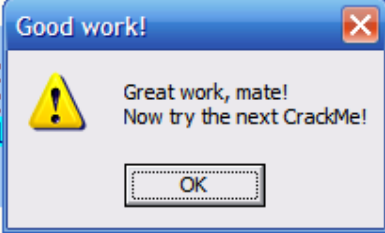
```

0040134D . 6A 30       push    30
0040134F . 68 29214000 push    00402129
00401354 . 68 34214000 push    00402134
00401359 . FF75 08     push    dword ptr [ebp+8]
0040135C . E8 D9000000 call     <jmp.&USER32.MessageBoxA>
00401361 . C3         retn

```

Style = MB_OK|MB_ICONEXCLAMATION|MB_...
Title = "Good work!"
Text = "Great work, mate!", CR, "Now try the next CrackMe!"
hOwner = ...
MessageBoxA
BeepType = MB_OK

Nào ta cùng thử xem có đúng không nhé, nhấn F9 để run và bùm :

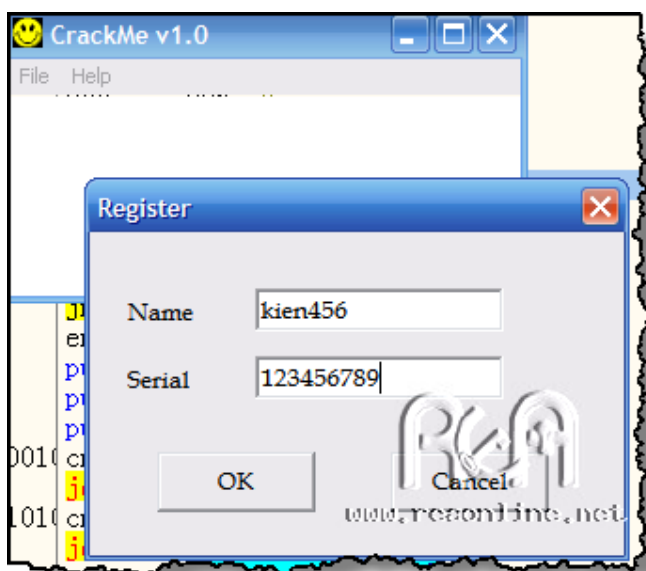


```

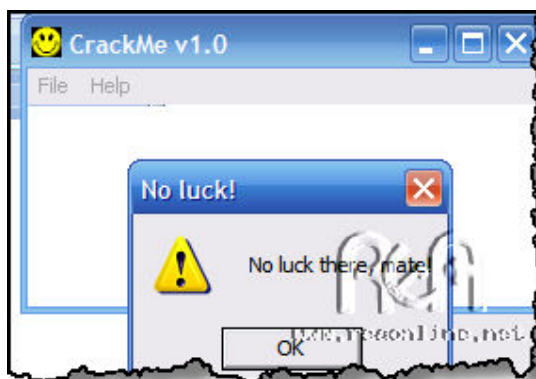
Style = MB_OK|MB_ICONEXCLAMATION|MB_...
Title = "Good work!"
Text = "Great work, mate!", CR, "Now try the next CrackMe!"
hOwner = ...
MessageBoxA
BeepType = MB_OK

```

Vậy tại đây chúng ta có thể khẳng định lệnh Cmp eax, ebx chính là lệnh so sánh liên quan đến Serial, phụ thuộc vào kết quả đầu ra tức là (eax = ebx?) mà lệnh nhảy sẽ quyết định việc show Bad boy hay Show good boy. Cũng từ đây ta kết luận crackme này có hai đoạn check riêng biệt, một đoạn liên quan đến Name và một đoạn liên quan đến Serial. Vì nếu chúng ta nhập Name mà trong Name nhập vào có số thì chúng ta sẽ nhận Nag luôn. Ok ta kiểm tra thử nhé :



Nhấn Ok để xác nhận thông tin nhập vào và ...:



Ta nhấn Ok một lần nữa và lần này ta mới dừng lại tại BP mà ta đã thiết lập ở trên :

00401241	. 3BC3	cmp	eax, ebx
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

Nếu như ta bypass nốt đoạn check trên thì mới vượt qua được Nag cuối cùng của Crackme này, tức là khúc này sẽ check serial, đúng thì sẽ hiện "Greate work..." mà sai thì tiếp tục hiện "No luck..". Ok, giờ ta quay lại khúc mà ta break khi ta đặt BP tại hàm api **MessageBoxA** :

Address	Value	Comment
0013FE8C	004013C1	CALL to MessageBoxA from CRACKME.004013BC
0013FE90	0005062E	hOwner = 0005062E ('CrackMe v1.0',class='No need to disasm the code!')
0013FE94	00402169	Text = "No luck there, mate!"
0013FE98	00402160	Title = "No luck!"
0013FE9C	00000030	Style = MB_OK MB_ICONEXCLAMATION MB_APPLMODAL
0013FEA0	00401232	RETURN to CRACKME.00401232 from CRACKME.0040137E
0013FEA4	0040218E	ASCII "KIEN456"

Quan sát ta thấy địa chỉ trở về là 0x004013C1, follow theo địa chỉ này

0040137D	C3	retn	
0040137E	\$ 8B7424 04	mov esi, dword ptr [esp+4]	
00401382	56	push esi	
00401383	> 8A06	mov al, byte ptr [esi]	
00401385	84C0	test al, al	
00401387	~ 74 13	je short 0040139C	
00401389	3C 41	cmp al, 41	
0040138B	~ 72 1F	jb short 004013AC	
0040138D	3C 5A	cmp al, 5A	
0040138F	~ 73 03	jnb short 00401394	
00401391	46	inc esi	
00401392	^ EB EF	jmp short 00401383	
00401394	> E8 39000000	call 004013D2	
00401399	46	inc esi	
0040139A	^ EB E7	jmp short 00401383	
0040139C	> 5E	pop esi	
0040139D	E8 20000000	call 004013C2	
004013A2	81F7 78560000	xor edi, 5678	
004013A8	8BC7	mov eax, edi	
004013AA	~ EB 15	jmp short 004013C1	
004013AC	> 5E	pop esi	
004013AD	6A 30	push 30	
004013AF	68 60214000	push 00402160	
004013B4	68 69214000	push 00402169	
004013B9	FF75 08	dword ptr [ebp+8]	
004013BC	E8 79000000	call <jmp.&USER32.MessageBoxA>	
004013C1	> C3	retn	

kiểm tra chuỗi Name mà ta nhập vào

Tính toán với Name nhập vào nếu Name hợp lệ

Style = MB_OK|MB_ICONEXCLAM
Title = "No luck!"
Text = "No luck there, mate"
hOwner
MessageBoxA

Tại đây, chúng ta thấy rằng Olly chỉ cho ta biết đoạn routine này bắt đầu từ 0x0040137E và kết thúc tại 0x004013C1. Để ý một chút chúng ta cũng thấy rằng tại địa chỉ 0x004013AC có một dấu ">". Kí hiệu này sẽ chỉ cho ta biết lệnh nhảy ở vị trí nào nhảy tới nó, để biết chi tiết ta chỉ việc chọn vị trí có dấu trên và quan sát cửa sổ Tip Window :

0040138B	~ 72 1F	jb short 004013AC	
0040138D	3C 5A	cmp al, 5A	
0040138F	~ 73 03	jnb short 00401394	
00401391	46	inc esi	
00401392	^ EB EF	jmp short 00401383	
00401394	> E8 39000000	call 004013D2	
00401399	46	inc esi	
0040139A	^ EB E7	jmp short 00401383	
0040139C	> 5E	pop esi	
0040139D	E8 20000000	call 004013C2	
004013A2	81F7 78560000	xor edi, 5678	
004013A8	8BC7	mov eax, edi	
004013AA	~ EB 15	jmp short 004013C1	
004013AC	> 5E	pop esi	
004013AD	6A 30	push 30	
004013AF	68 60214000	push 00402160	
004013B4	68 69214000	push 00402169	
004013B9	FF75 08	dword ptr [ebp+8]	
004013BC	E8 79000000	call <jmp.&USER32.MessageBoxA>	
004013C1	> C3	retn	
004013C2	\$ 33FF	xor edi, edi	
004013C4	33DB	xor ebx, ebx	

Style = MB_...
Title = "No...
Text = "No...
hOwner
MessageBoxA

www.reasonline.net

Jump from 0040138B

Lại một lần nữa chúng ta để ý thấy rằng có một lệnh so sánh và một lệnh nhảy phụ thuộc vào kết quả so sánh, vậy ta đặt một BP tại lệnh nhảy tại địa chỉ 0x0040138B :

00401385	. 84C0	test	al, al
00401387	. 74 13	je	short 0040139C
00401389	. 3C 41	cmp	al, 41
0040138B	. 72 1F	jb	short 004013AC
0040138D	. 3C 5A	cmp	al, 5A
0040138F	. 73 03	jnb	short 00401394
00401391	. 46	inc	esi
00401392	. EB EF	jmp	short 00401383
00401394	> E8 39000000	call	004013D2
00401399	. 46	inc	esi

Ok nhấn F9 để run chương trình và nhập thông tin như lúc trước ta nhập :

Name	<input type="text" value="kien456"/>
Serial	<input type="text" value="123456789"/>

Sau đó nhấn Ok, ta sẽ break tại chỗ ta vừa set BP :

00401387	. 74 13	je	short 0040139C
00401389	. 3C 41	cmp	al, 41
0040138B	. 72 1F	jb	short 004013AC
0040138D	. 3C 5A	cmp	al, 5A
0040138F	. 73 03	jnb	short 00401394

Sau khi break, ta quan sát thấy rằng lần đầu tiên break lệnh nhảy này sẽ không nhảy. Lý do là vì nó sẽ kiểm tra dần dần từng kí tự của chuỗi Name ta nhập vào, nếu có chứa chữ số thì nó mới nhảy. Giờ ta nhấn F9 thêm 4 lần nữa lúc này ta quan sát thấy rằng nó kiểm tra đến kí tự thứ 5, vị trí này chính là số "4" cho nên lệnh nhảy sẽ được thực thi :

00401387	. 74 13	je	short 0040139C
00401389	. 3C 41	cmp	al, 41
0040138B	. 72 1F	jb	short 004013AC
0040138D	. 3C 5A	cmp	al, 5A
0040138F	. 73 03	jnb	short 00401394
00401391	. 46	inc	esi

Registers (FPU)	
EAX	00000034
ECX	0013F000
EDX	7C90BB94
EBX	00000000

eax = 0x34 <=> số 4 trong bảng mã ASCII

Quan sát các giá trị của các cờ ta thấy rằng cờ C được bật lên :

```

EIP 0040138B CRACKME.0040138B
C 1 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 1 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)

```

Chọn giá trị của cờ C và nhấn đúp để set về giá trị 0, quan sát sang cửa sổ CPU ta thấy lệnh nhảy không còn hiệu lực nữa :

```

C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 1 FS 003B 32bit 7FFDF000(FFF)

```

00401389	. 3C 41	cmp	al, 41
0040138B	. 72 1F	jb	short 004013AC
0040138D	. 3C 5A	cmp	al, 5A
0040138F	. 73 03	jnb	short 00401394
00401391	. 46	inc	esi
00401392	. ^ EB EF	jmp	short 00401383
00401394	> E8 39000000	call	004013D2
00401399	. 46	inc	esi
0040139A	. ^ EB E7	jmp	short 00401383
0040139C	> 5E	pop	esi
0040139D	. E8 20000000	call	004013C2
004013A2	. 81F7 78560000	xor	edi, 5678
004013A8	. 8BC7	mov	eax, edi
004013AA	. ^ EB 15	jmp	short 004013C1

Ta tiếp tục nhấn F9 và bypass tương tự cho hai số tiếp theo. Sau khi vượt qua đoạn check này chúng ta sẽ break tại đây :

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
00401253	. C8 000000	enter	0, 0

Tiếp tục sử dụng tiểu xảo để active lệnh nhảy bằng cách nhấp đúp chuột tại cờ Z ☺ :

```

C 1 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)

```

00401241	. 3BC5	cmp	eax, ebx
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

Cuối cùng ta nhấn F9 thêm một lần nữa và lần này là những gì chúng ta mong đợi :



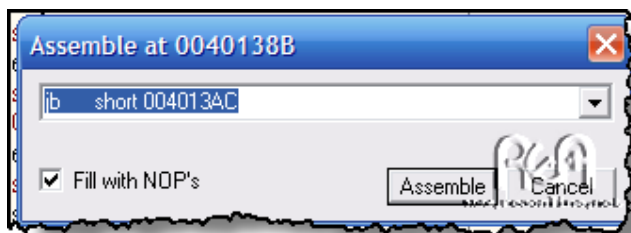
Như các bạn thấy rằng để đến được cái thông báo Great work như trên chúng ta đã thực hiện bằng cách thay đổi các cờ, tuy nhiên các bạn biết rằng các cờ này thay đổi liên tục do đó ta không có cách nào để lưu lại những gì chúng ta làm được. Vì vậy, để có thể ép chương trình của chúng ta chấp nhận bất kì username và serial nào mà chúng ta nhập vào thì ta buộc phải thay đổi code của chương trình!! Ta sẽ làm như thế nào?

III. Patch bằng cách sửa code

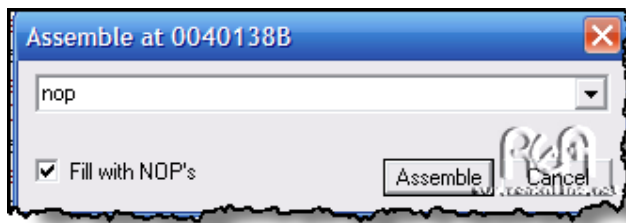
Đầu tiên ta xử lý cái lệnh nhảy ở chỗ check liên quan đến Name nhập vào để nó chấp nhận trong Name có số :

00401389	. 3C 41	cmp	al, 41
0040138B	72 1F	jb	short 004013AC
0040138D	3C 5A	cmp	al, 5A
0040138F	73 03	jnb	short 00401394

Hehe giờ ta sẽ không thay đổi cờ nữa mà edit lại code luôn, tại địa chỉ 0x0040138B chứa lệnh nhảy ta nhấn phím Space bar :



Thay bằng lệnh NOP :



Sau đó nhấn Assemble ta có kết quả như sau :

00401389	. 3C 41	cmp al, 41
0040138B	90	nop
0040138C	90	nop
0040138D	. 3C 5A	cmp al, 5A
0040138F	73 02	jmp short 00401399

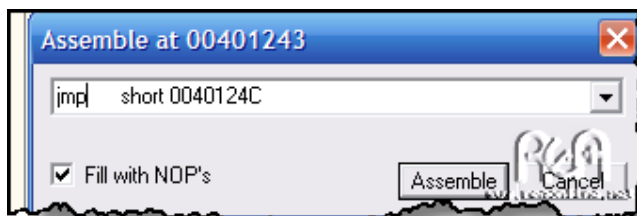
Ok như vậy chúng ta đã loại bỏ được lệnh nhảy này, giờ ta nhấn F2 để bỏ việc thiết lập BP tại vị trí trên :

0040138B	90	nop
0040138C	90	nop
0040138D	. 3C 5A	cmp al, 5A

Tiếp theo ta đi tới chỗ lệnh nhảy tại chỗ kiểm tra Serial :

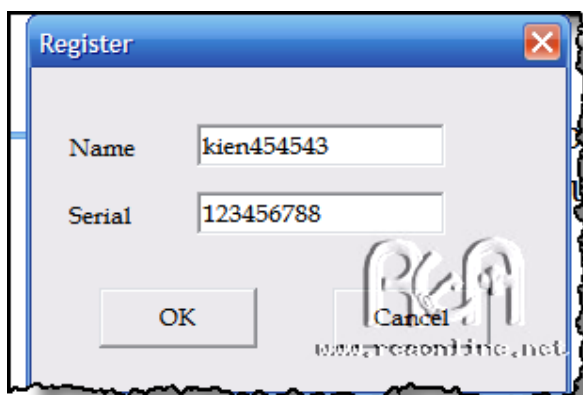
00401241	. 3BC3	cmp eax, ebx
00401243	. 74 07	je short 0040124C
00401245	. E8 18010000	call 00401362
0040124A	. EB 9A	jmp short 004011E6
0040124C	. E8 FC000000	call 0040134D

Tại đây ta cũng không cần tác động cờ Z để active lệnh nhảy nữa. Ta nhấn Space Bar và thay bằng lệnh sau :

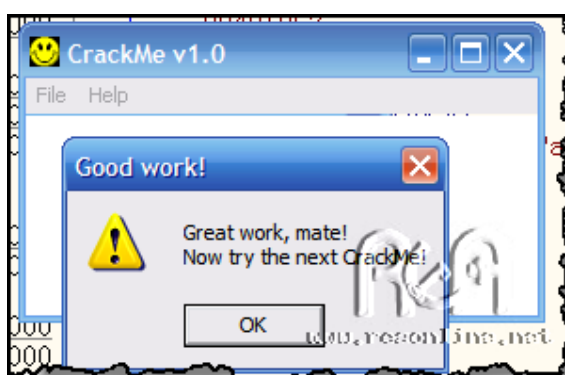


00401241	. 3BC3	cmp eax, ebx
00401243	. EB 07	jmp short 0040124C
00401245	. E8 18010000	call 00401362
0040124A	. EB 9A	jmp short 004011E6
0040124C	. E8 FC000000	call 0040134D
00401251	. EB 93	jmp short 004011E6

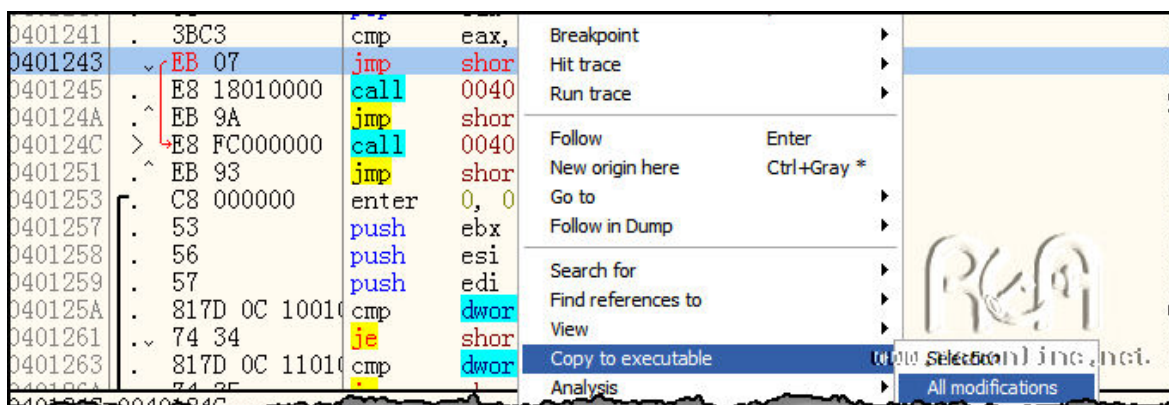
Thay bằng jmp tức là ta ép cho nó luôn nhảy dù có đúng hay sai đi nữa, ok sau khi edit ta bỏ BP tại vị trí trên. Nhấn F9 để run và kiểm tra kết quả :



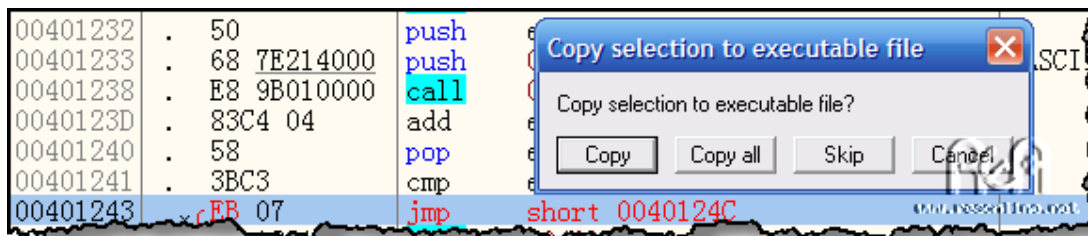
Chọn Ok và khả khả :



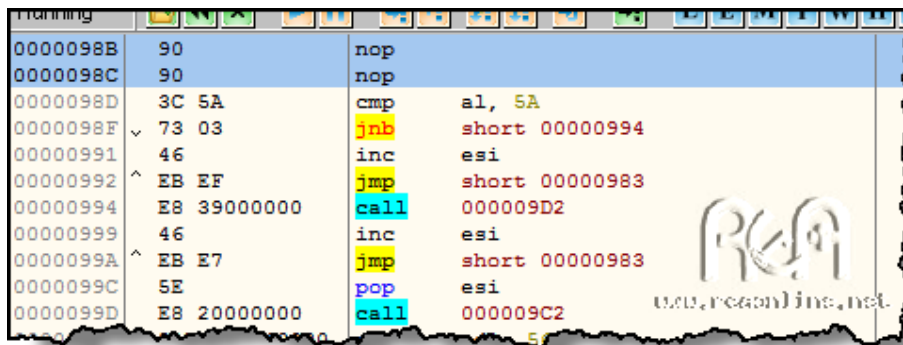
Như các bạn thấy vừa rồi tôi đã hướng dẫn các bạn cách edit lệnh trực tiếp trong Olly, nhưng nếu chúng ta restart chương trình thì những thay đổi này sẽ không còn tác dụng có nghĩa là nó ko lưu lại những gì ta đã làm. Vậy để lưu các thay đổi vừa rồi trước khi đóng Olly bạn làm như sau :



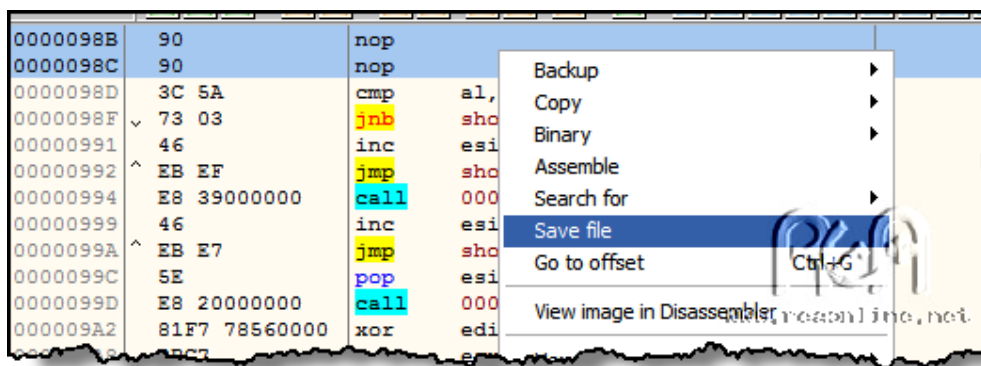
Nhấn chuột phải sau đó chọn **Copy to executable > All modifications** , một cửa sổ bật ra :

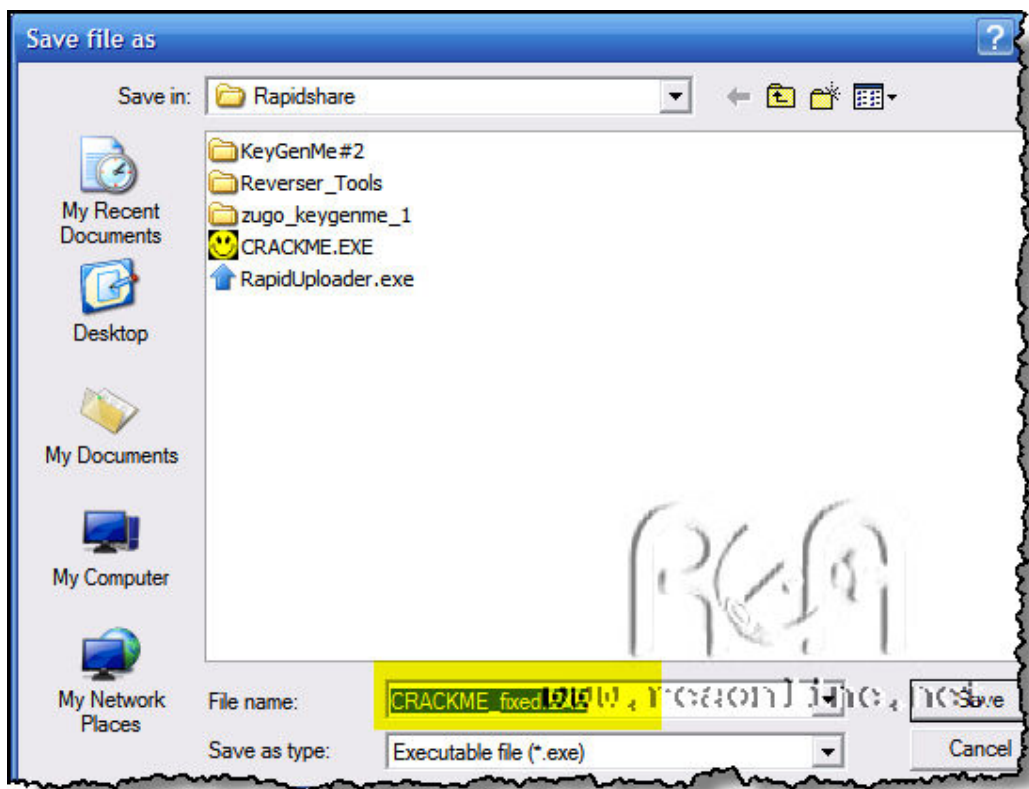


Ta chọn Copy all để lưu lại toàn bộ những gì ta đã thay đổi :

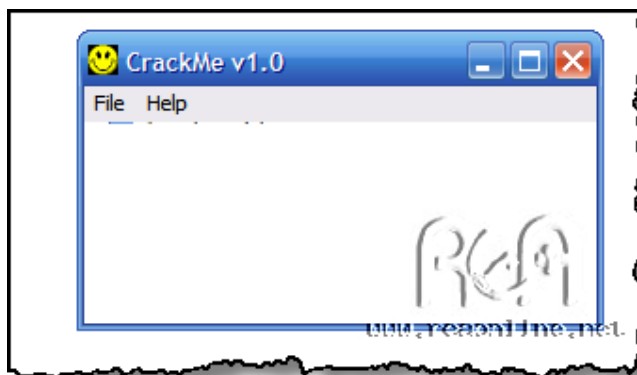
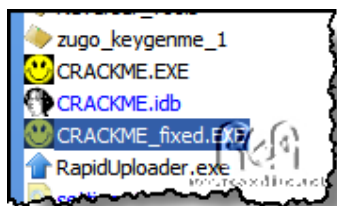


Một cửa sổ khác bật ra như trên, tại đây chúng ta nhấn chuột phải và chọn : Save file

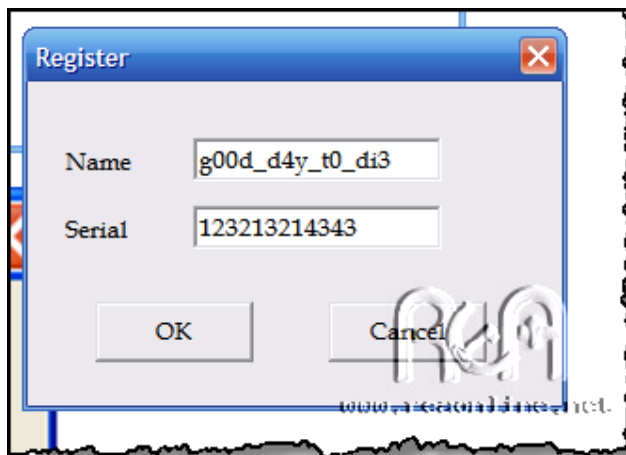




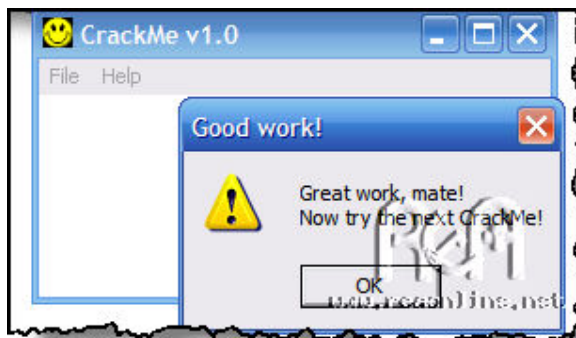
Đặt cho file mà chúng ta Save dưới một tên mới, ví dụ Crackme_fixed.exe và chọn Save. Đóng Olly lại và test thử kết quả bằng cách run file mới :



Nhập thông tin để đăng kí :



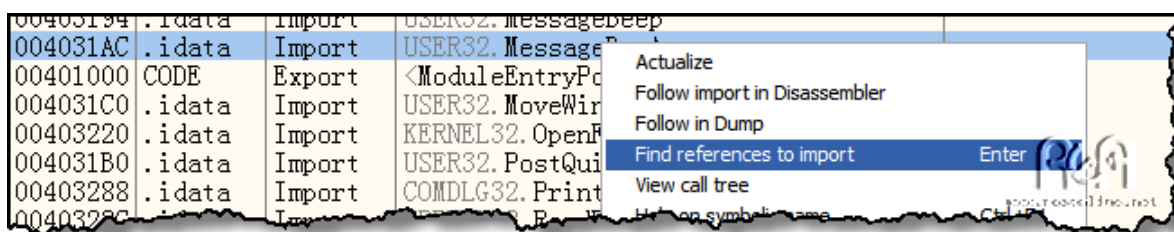
Nhấn Ok ☺ :



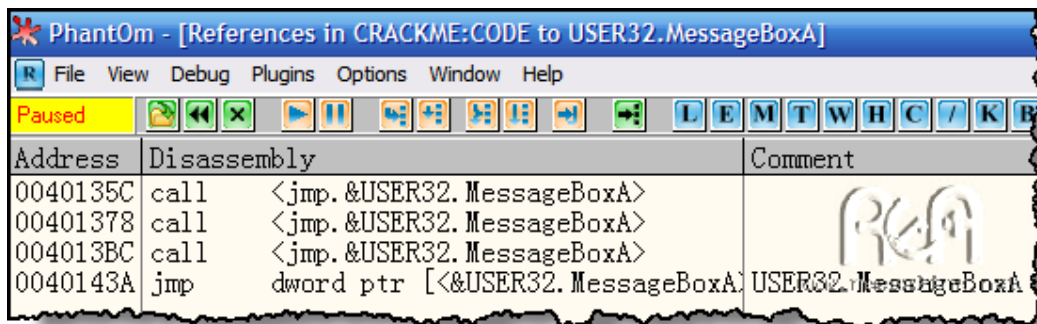
Khà khà vậy là file mới mà ta vừa save hoạt động rất tốt!! Quá khỏe, giờ ta nhập loạn từng phèo nó cũng nói là “Greate work ...”

Phu lục

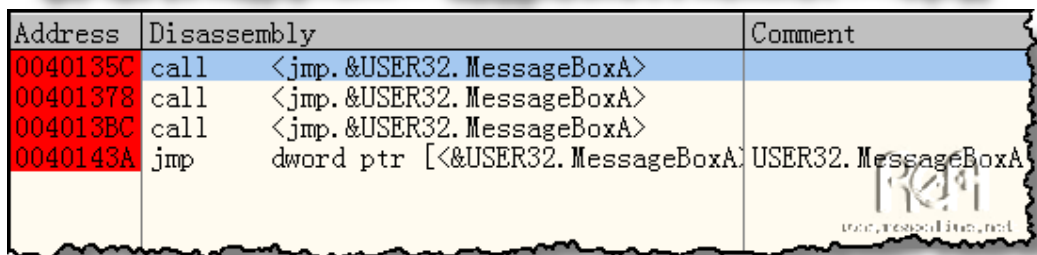
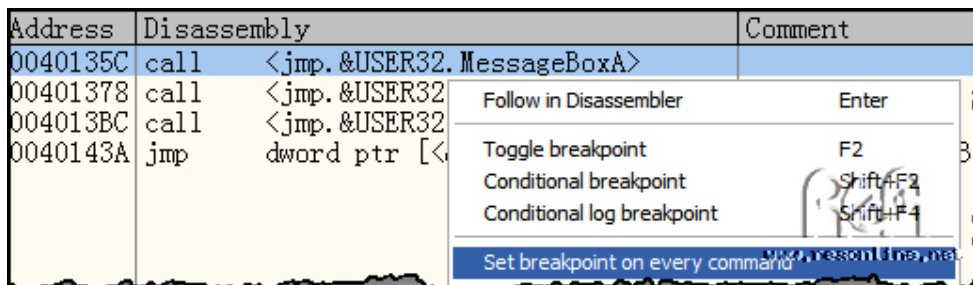
Phần phụ lục này ta tìm hiểu thêm một cách đặt BP để dò ra tử huyệt, đầu tiên load crackme vào Olly và nhấn **Ctrl + N**. Tìm đến hàm **MessageBoxA**, nhấn chuột phải và chọn như sau :



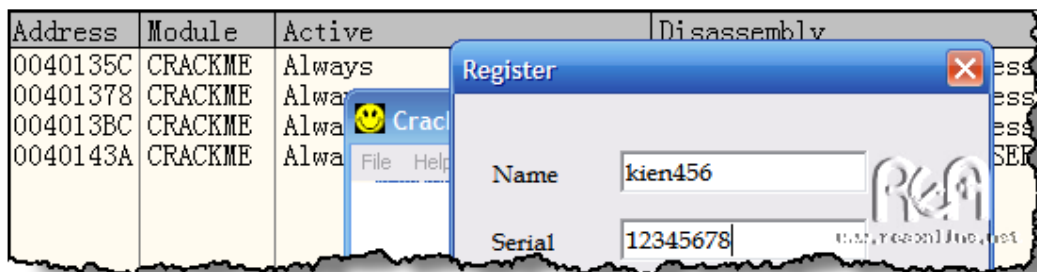
Cửa sổ **References** sẽ xuất hiện :



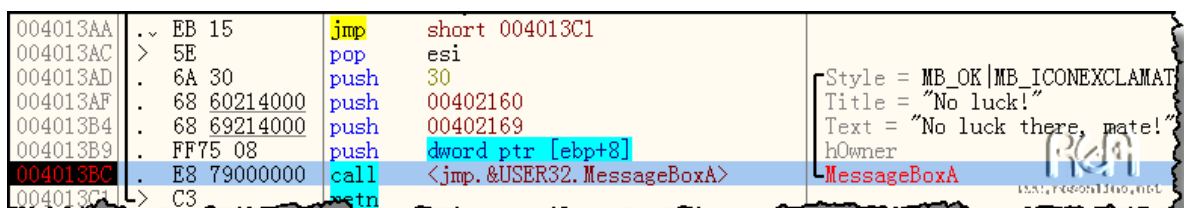
Nhấn chuột phải tại vị trí bất kì và chọn :



Sau khi đã đặt BP như trên, tiến hành nhấn F9 để run crackme và nhập thông tin :



Sau đó nhấn Ok, ta sẽ break tại đây :



Khả khả vậy là đúng chỗ ta cần tìm rồi nhé, chỉ việc lần theo lệnh Retn là ta biết được nơi sẽ gọi tới đoạn code này. Lúc này ta làm tương tự như những gì mà tôi đã viết ở trên.

Ok vậy là phần 9 của loạt tuts về Ollydbg đến đây là kết thúc, bài viết này đã hướng dẫn sơ bộ các cách cơ bản để bạn tìm một hàm API, cách thiết lập BP, cách edit các cò và hơn cả là patch chương trình bằng cách edit code. Đây mới chỉ là những gì cơ bản nhất thôi, phía trước còn nhiều điều thú vị lắm. Hi vọng trong thời gian tới tôi sẽ 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, QHQCkrker, the_Lighthouse, Merc, Hoadongnoi, Nini ... all REA's members, TQN, HacNho, RongChauA, Deux, tlandn, light.phoenix, dqtn, ARTEAM all my friend, and YOU.

--++--==[**Thanks To**]==--++--

iamidiot, WhyNotBar, trickyboy, dzungltn, takada, hurt_heart, haule_nth, hytkl, moth, XIANUA, nhc1987, 0xdie 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-Nigurath**, **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