

# INTRODUCTION TO THE CRACKING WITH OLLYDBG

## FROM CRACKLATINOS

([\\_kienmanowar\\_](#))



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

### I. Giới thiệu chung

Chào các bạn, trong phần bài viết trước ở phần 11 tôi đã hướng dẫn các bạn tìm hiểu về hai dạng Break Points là **Hardware Breakpoints** và **Conditional Breakpoints**, cũng như các cách thiết lập cho hai dạng BP này. Lẽ ra trong phần 11 tôi cũng định giới thiệu luôn về dạng **Message BreakPoints**, nhưng thiết nghĩ nếu gộp luôn vào trong phần 11 sẽ khiến các bạn quá tải và dẫn đến “tẩu hỏa” mất ☺. Cho nên tôi cắt riêng phần **Message Breakpoints** ra và quyết định giới thiệu về nó trong phần 12 này. Rất nhiều điều thú vị đang nằm ở phía trước.... N0w....L3t's G0!!!!!!!

### II. BreakPoints in OllyDbg

#### 1. Message Breakpoints :

**Message Breakpoint** là gì nhỉ? Sao lại lảm loại BP đến thế ☺. Khi gặp bất kì những vấn đề nào mới, trong đầu tôi luôn xuất hiện những câu hỏi liên quan đến vấn đề đó. Sau đó, tôi tìm cách tiếp cận thông tin để giải quyết cho những câu hỏi của tôi. Trước tiên, tôi chẳng biết Message Breakpoint nó là cái gì, cho nên tôi tra help của Olly trước, hi vọng sẽ tìm ra chút thông tin nào đó về nó. Đọc trong help file của Olly thì chỉ nhận được một chút thông tin như sau :

*Message breakpoint is same as conditional logging except that OllyDbg automatically generates condition allowing to break on some message (like WM\_PAINT) on the entry point to window procedure. You can set it in the Windows window*

Thông tin đầu tiên gợi mở cho tôi là thằng Message breakpoint này gần tương tự như Conditional Log BP ngoài trừ việc khác ở chỗ Olly hoàn toàn tự động tạo ra các điều kiện cho phép dừng lại tại các Message. Mà cái Message ở đây chính là các Windows Message. Tôi không phải là dân chuyên lập trình cho nên tôi tìm hiểu tiếp các thông tin khác trên Net về Windows Message :

*“The messages in Windows are used to communicate most of the events, at least in the basic levels. If you want a window or control (which is a specialized window) does something, you must send a message to him. If another window wants you do something, then it sends a message to you. If an event occurs, such as when the user moves the mouse, presses the keyboard, etc... the system sends a message to the affected window. This window receives the message and acts suitably.”*

-Message là một trong những phương tiện giao tiếp quan trọng nhất trong môi trường Windows.  
-Lập trình trong Windows chủ yếu là đáp ứng lại những sự kiện.  
-Message có thể báo hiệu nhiều sự kiện gây ra bởi người sử dụng, hệ điều hành, hoặc chương trình khác.

-Có hai loại message: window message và thread message.

### Window Message:

-Tất cả các message đều được trữ trong một Message Queue(một nơi trong bộ nhớ). Những message này sau đó sẽ được luân chuyển giữa các ứng dụng.

### Message Loop:

-Bạn có thể gọi những message bằng cách tạo ra một Message Queue.

-Message Loop là một vòng lặp kiểm tra những message trong Message Queue.

-Khi một message được nhận, Message Loop giải quyết nó bằng cách gọi Message Handler (một hàm được thiết kế để giúp Message Loop xử lý message)<hầu hết công việc lập trình của mình sẽ tập trung vào đây>

-Message Loop sẽ kết thúc khi nhận được message WM\_QUIT (lúc người dùng chọn File/Exit || click vào nút Close || bấm Alt+F4)

-Khi bạn tạo cửa sổ (ứng dụng) Windows sẽ tạo cho bạn một Message Handler mặc định. Bạn sẽ vào đây để sửa chữa giúp ứng dụng phản hồi lại những sự kiện theo ý bạn muốn ->chương trình của bạn.

-Tất cả những control chuẩn đều như thế. Lấy một button làm ví dụ: khi nó nhận WM\_PAINT message nó sẽ vẽ button; khi bạn click chuột trái lên nó sẽ nhận WM\_LBUTTONDOWN message và nó sẽ vẽ hình button bị nhấn xuống; khi buông chuột ra nó nhận WM\_LBUTTONUP message và sẽ vẽ lại button bình thường.

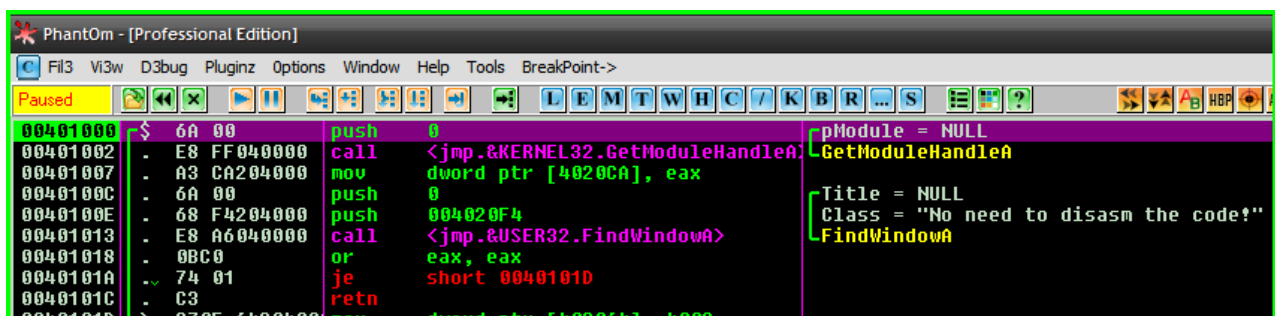
-Tên của window message thường có dạng WM\_ và hàm để xử lý message đó thường có dạng On. Ví dụ hàm xử lý WM\_SIZE message là OnSize.

-Message thường có hai tham số lưu trữ thông tin về sự kiện(32 bit): lParam kiểu LONG và wParam kiểu WORD. Ví dụ: WM\_MOUSEMOVE sẽ trữ tọa độ chuột trong một tham số còn tham số kia sẽ có cờ hiệu để ghi nhận trạng thái của phím ATL, Shift, CTRL và những nút trên con chuột.

-Message có thể trả về một giá trị giúp bạn gửi dữ liệu ngược trở về chương trình gửi nó. Ví dụ: WM\_CTLCOLOR message chờ bạn trả về một HBRUSH (khi dùng AppWizard để tạo nhanh ứng dụng bạn chú ý những hàm có dạng On... và nhận xét những kiểu trả về của nó, bạn sẽ hiểu hơn về cơ chế liên kết giữa các thành phần ứng dụng với nhau)

Nguồn : <http://www.eco-blue.net/index.php?showtopic=571&mode=threaded&pid=2527>

Ok, qua những thông tin tổng hợp ở trên tôi chắc rằng tôi và các bạn cũng đã phần nào hiểu được ý nghĩa và mục đích của Windows Message. Bây giờ chúng ta sẽ cùng nhau làm một ví dụ, như các bạn đã đọc ở các phần trước, tôi đã hướng dẫn các bạn cách đặt BP tại các hàm APIs để dừng sự thực thi của chương trình cũng như tìm ra những điểm mấu chốt để patch chương trình. Lần này ta thực hành như sau : để tìm ra Serial của chương trình chúng ta muốn chương trình dừng sự thực thi khi chúng ta tiến hành nhập fake Username và Serial và nhấn nút OK, cụ thể hơn có nghĩa là sau khi ta nhấn nút OK thì chương trình sẽ dừng lại theo đúng mong muốn của ta. Chà nghe có vẻ phức tạp quá phải không, vậy thì ta load crackme vào OllyDbg cái đã ☺ :



The screenshot shows the OllyDbg interface with the following assembly code and comment:

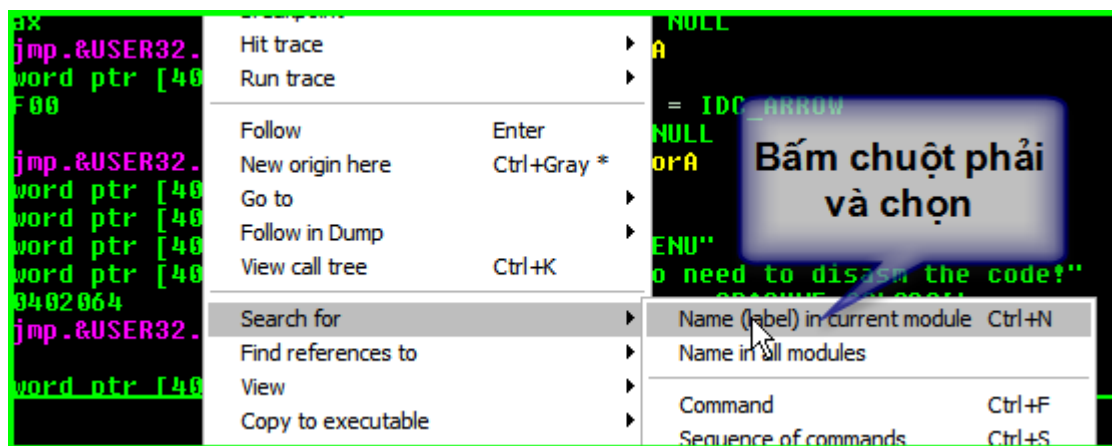
Address	Disassembly	Comment
00401000	6A 00 push 0	pModule = NULL
00401002	E8 FF040000 call <jmp.&KERNEL32.GetModuleHandleA>	GetModuleHandleA
00401007	A3 C0204000 mov dword ptr [4020C0], eax	
0040100C	6A 00 push 0	
0040100E	68 F4204000 push 004020F4	Title = NULL
00401013	E8 A6040000 call <jmp.&USER32.FindWindowA>	Class = "No need to disasm the code!"
00401018	0BC0 or eax, eax	FindWindowA
0040101A	74 01 je short 0040101D	
0040101C	C3 retn	

Nhưng trước khi thực hiện phương pháp Message BP ta ôn lại phương pháp đặt BP tại APIs cái đã. Lần này tôi không dùng hàm **MessageBoxA** nữa mà sẽ dùng hàm **GetDlgItemTextA**, hàm này có nhiệm vụ như sau :

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

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

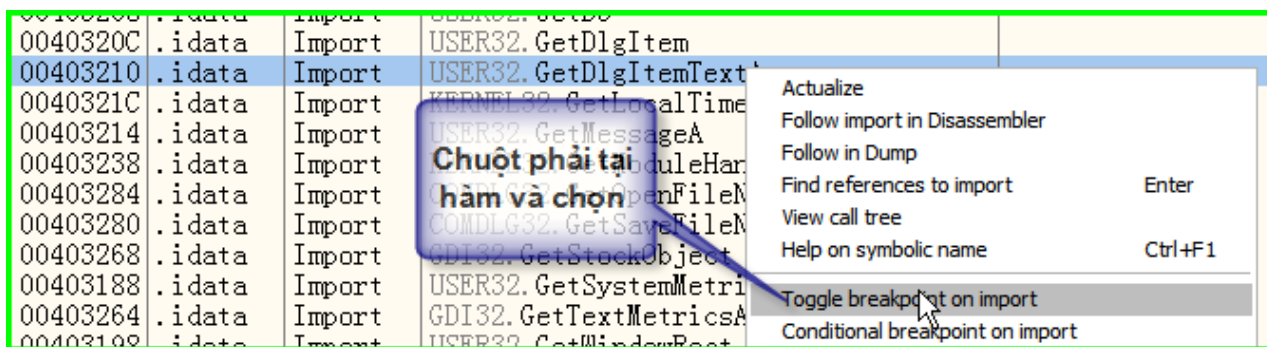
Mục đích của tôi là, thông qua hàm này tôi tìm ra nơi lưu thông tin mà tôi nhập vào để dùng cho các quá trình xử lý tiếp theo trong chương trình. Vậy để đặt BP tại hàm **GetDlgItemTextA** tôi làm như sau, trước tiên tôi tìm nó cái đã :



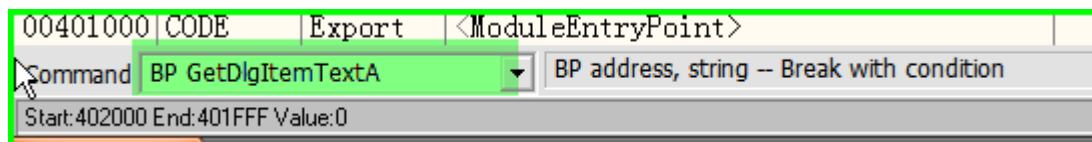
Các bạn còn nhớ cách tìm hàm API trong cửa sổ Names mà tôi đã hướng dẫn ở các phần trước chứ, nếu không nhớ thì các bạn chỉ việc : tại cửa sổ Names gõ đúng tên hàm API cần tìm là Olly sẽ đưa chúng ta đến đúng hàm ta cần. Tôi có được như sau :

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	COMDLG32.GetOpenFileNameA	
00403280	.idata	Import	COMDLG32.GetSaveFileNameA	

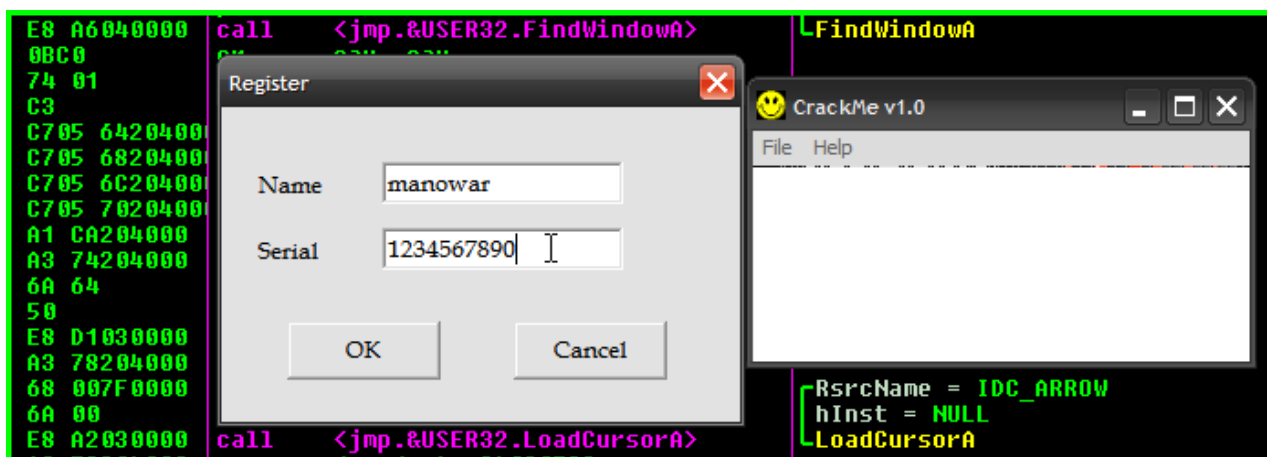
Thực hiện đặt BP tại hàm này như sau :



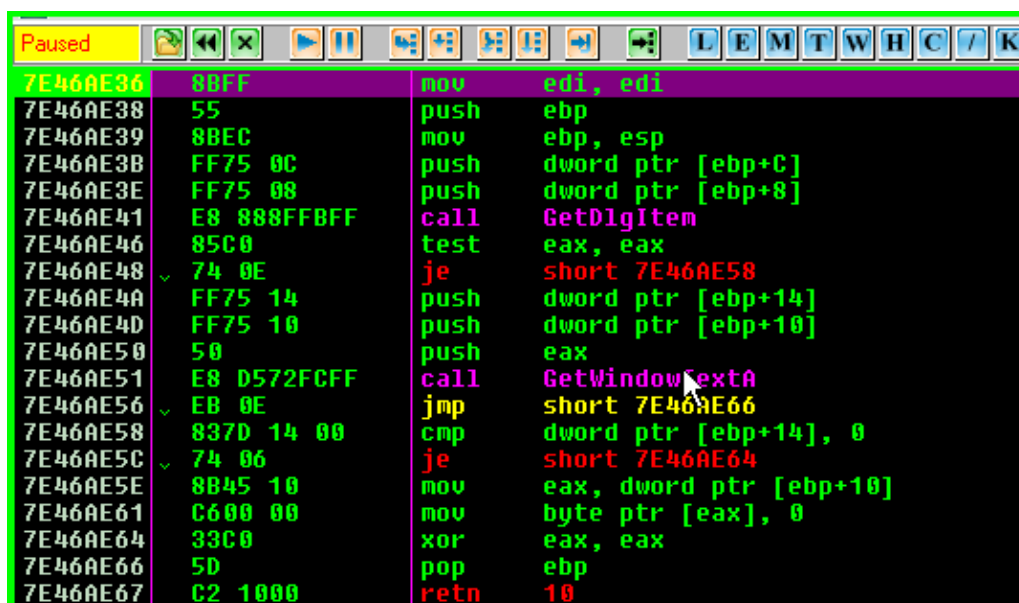
Hoặc đơn giản hơn, tại cửa sổ Command Bar Plug-in ta gõ :



Đặt BP xong, cho thực thi chương trình bằng cách nhấn **F9**. Sau đó điền thông tin về UserName và Serial vào :



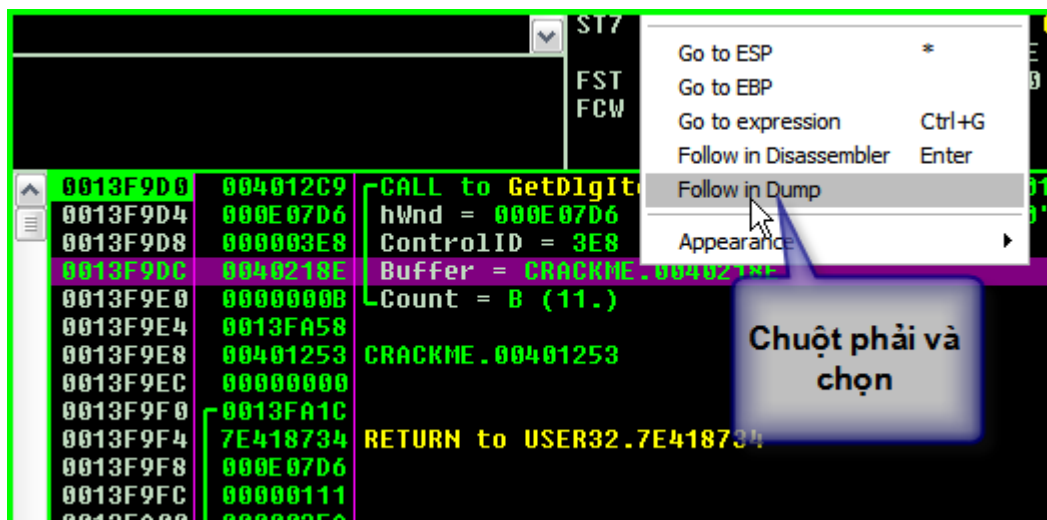
Nhập xong ta nhấn OK, ngay lập tức Olly sẽ break tại API **GetDlgItemTextA** mà chúng ta đã đặt BP ở trên.



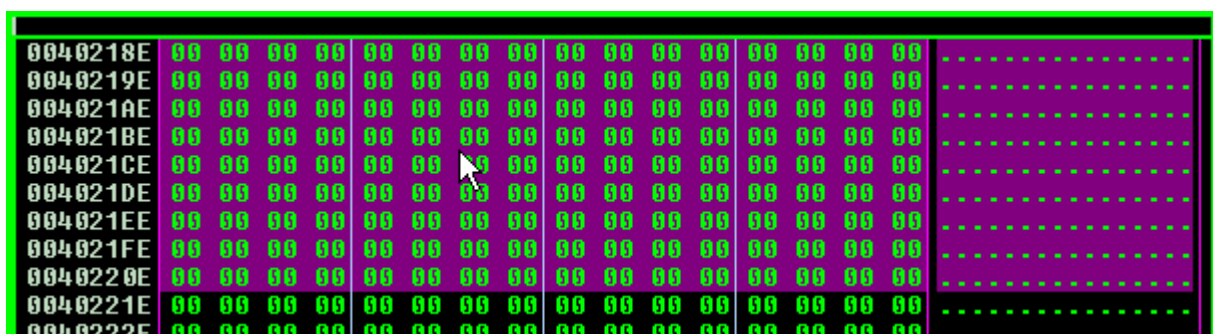
Đào mắt qua cửa sổ Stack để xem ta thu được những thông tin gì ☺ :

0013F9D0	004012C9	CALL to GetDlgItemTextA From CRACKME.004012C4
0013F9D4	000E07D6	hWnd = 000E07D6 ('Register',class='#32770')
0013F9D8	000003E8	ControlID = 3E8 (1000.)
0013F9DC	0040218E	Buffer = CRACKME.0040218E
0013F9E0	0000000B	Count = B (11.)
0013F9E4	0013FA58	
0013F9E8	00401253	CRACKME.00401253
0013F9EC	00000000	
0013F9F0	0013FA1C	
0013F9F4	7E418734	RETURN to USER32.7E418734
0013F9F8	000E07D6	
0013F9FC	00000111	

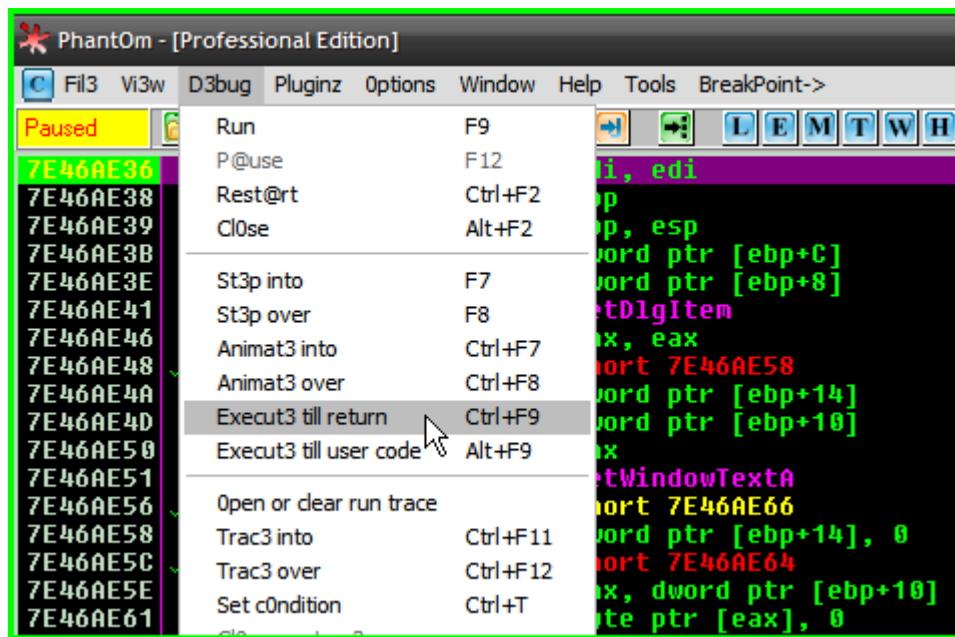
Wow, như các bạn thấy thông tin các tham số truyền vào cho hàm **GetDlgItemTextA** là hết sức rõ ràng, kết hợp với mô tả về hàm này ở trên các bạn sẽ thấy rằng tham số Buffer sẽ là nơi lưu đoạn text mà chúng ta nhập vào, nhưng lúc này chúng ta chưa biết là vùng Buffer này sẽ lưu Name hay Serial. Ta phỏng đoán theo thứ tự lần lượt từ trên xuống thì nó sẽ lấy UserName trước. Để xác định chính xác ta làm như sau, chuột phải tại chỗ Buffer và chọn **Follow in Dump** hoặc chuyển qua cửa sổ Dump, nhấn **Ctrl + G** và nhập vào địa chỉ của vùng Buffer là 0x0040218E :



Lúc này tại cửa sổ Dump các bạn nhận được kết quả là toàn byte 0x00. Đơn giản là vì ta đã thực thi hàm đâu mà lấy được đoạn text nhập vào ☺



Bây giờ ta cho thực thi hàm **GetDlgItemTextA** và quan sát kết quả thu được tại cửa sổ Dump :



0040218E	6D 61 6E 6F	77 61 72 00	00 00 00 00	00 00 00 00	manowar.....
0040219E	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
004021AE	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
004021BE	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
004021CE	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....

Ok đúng như ta phỏng đoán, chương trình sẽ lấy UserName nhập vào trước và lưu nó vào vùng Buffer có địa chỉ là 0x0040218E. Tiếp theo ta lại cho thực thi chương trình bằng cách nhấn **F9** :

7E46AE36	8BFF	mov	edi, edi
7E46AE38	55	push	ebp
7E46AE39	8BEC	mov	ebp, esp
7E46AE3B	FF75 0C	push	dword ptr [ebp+C]
7E46AE3E	FF75 08	push	dword ptr [ebp+8]
7E46AE41	E8 88FFBFF	call	GetDlgItem

0013F9D0	004012E9	CALL to GetDlgItemTextA from CRACKME.004012E4
0013F9D4	000E07D6	hWnd = 000E07D6 ('Register',class='#32770')
0013F9D8	000003E9	ControlID = 3E9 (1001.)
0013F9DC	0040217E	Buffer = CRACKME.0040217E
0013F9E0	0000000B	Count = B (11.)
0013F9E4	0013F058	

Bùm, Olly một lần nữa lại Break tại hàm **GetDlgItemTextA**. Không cần nói chắc các bạn cũng có thể đoán ra ngay được là nó sẽ lấy Serial mà ta nhập vào. Nhưng lúc này Serial sẽ được lưu vào một vùng Buffer khác mà các bạn thấy ở trên, đó là 0x0040217E. Ta follow theo vùng Buffer này :

0040217E	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
0040218E	6D 61 6E 6F	77 61 72 00	00 00 00 00	00 00 00 00	manowar.....
0040219E	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
004021AE	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
004021BE	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
004021CE	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....

Cho thực thi hàm và kiểm tra kết quả có được tại vùng Buffer :



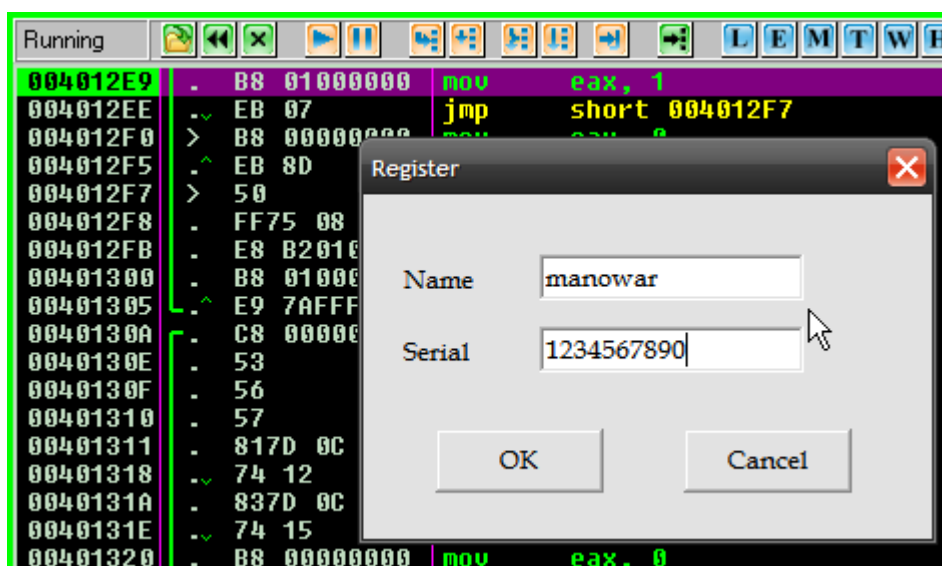
eax=0000000A															
0040217E	31	32	33	34	35	36	37	38	39	3A	00	00	00	00	1234567890.....
0040218E	6D	61	6E	6F	77	61	72	00	00	00	00	00	00	00	manowar.....
0040219E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
004021AE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
004021BE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
004021CE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
004021DE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

Hehe Serial mà chúng ta nhập vào kia!! Như vậy là với việc đặt BP tại hàm **GetDlgItemTextA** tôi đã dừng lại tại nơi lấy những thông tin mà tôi nhập vào, bao gồm UserName và Serial, bước tiếp theo tôi sẽ trace dần dần từng đoạn code cho tới khi nào tôi có thể tìm ra một Serial hợp lệ. Nhưng có lẽ tạm thời ta nên dừng lại tại đây để chuyển sang phần thực hành với Message BP, xem có thể thu được những thông tin tương tự như những gì ta vừa mới làm với việc đặt BP tại hàm APIs hay không? Trước khi thực hành ta xóa hết các BP đã đặt, vào cửa sổ BreakPoint và xóa hết :

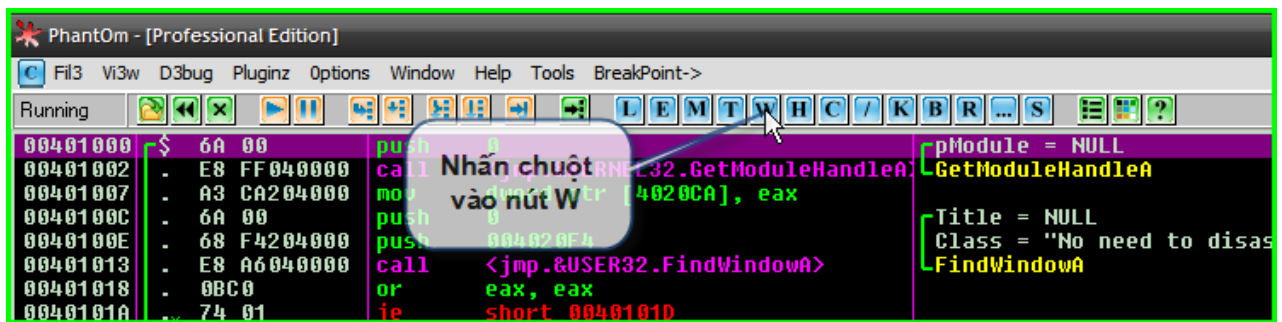
Address	Module	Active	Disassembly
7E46AE36	USER32	Always	mov edi

Remove Del  
Disable Space  
Edit condition  
Follow in Disassembler Enter  
Disable all  
Delete All BreakPoints  
Copy to clipboard  
Appearance

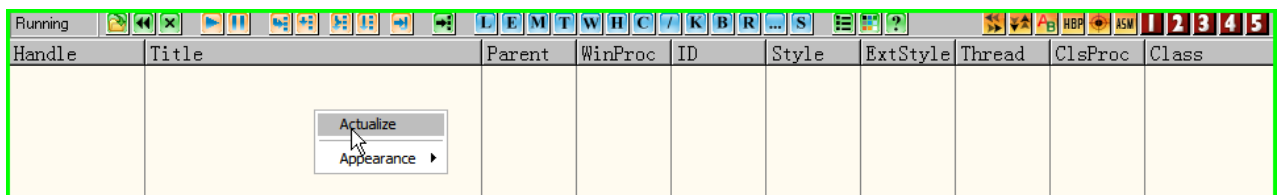
Sau khi xóa xong, ta restart lại Olly và cho thực thi chương trình , tiến hành nhập thông tin mà chương trình yêu cầu :



Không giống như đặt BP tại các hàm API là ta có thể đặt BP tại đầu hàm thì để đặt Message BP chúng ta phải làm việc với cửa sổ Windows. Hiện tại chương trình của chúng ta đang thực thi, chúng ta chuyển qua cửa sổ Window bằng cách nhấn vào nút **W** :



Sau khi bạn nhấn vào nút W thì cửa sổ Windows sẽ hiện ra. Nếu như bạn thấy nó trống trơn không có gì cả thì nhấn chuột phải và chọn **Actualize** :



Kết quả ta có được như sau :

Handle	Title	Parent	WinProc	ID	Style	ExtStyle	Thread	ClsProc	Class
000A01D4	Register	Topmost			14C800C4	00010101	Main	7E43E53F	#32770
000A0224		000A01D4		000003E8	50010080	00000204	Main	7E43B3B4	Edit
000C024E	Cancel	000A01D4		000003EB	50010000	00000004	Main	7E43AFFE	Button
000D01A2	Serial	000A01D4		0000FFFF	50020000	00000004	Main	7E43E582	Static
000D0214	Default IME	000A01D4			8C000000		Main	7E46C6F6	IME
0009022E	M	000D0214			8C000000		Main	FFFF02D5	MSCTIME UI
000F01EA	OK	000A01D4		000003EA	50010000	00000004	Main	7E43AFFE	Button
001001A4	Name	000A01D4		0000FFFF	50020000	00000004	Main	7E43E582	Static
001902A8		000A01D4		000003E9	50010080	00000204	Main	7E43B3B4	Edit
000B01CE	CrackMe v1.0	Topmost		02DF01F3	1CCF0000	00000100	Main	00401128	No need to disasm the code!

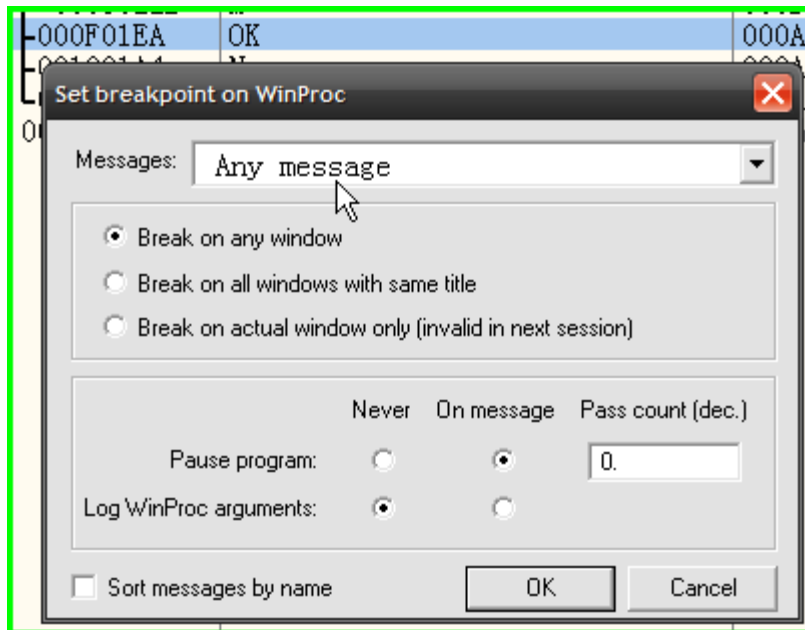
Như đã nói ở trên, mục đích của chúng ta là sau khi bấm Button Ok thì chương trình sẽ dừng lại. Vậy quan sát trong cửa sổ Windows, ở phần Class ta thấy có dòng Button, sau đó nhìn qua phần Title ta thấy được tên của Button là OK. Vậy đây chính là mục tiêu của chúng ta. Để đặt Message BP ta làm như sau, chuột phải tại nơi của Button OK và chọn **Message Breakpoint on ClassProc**:

000D0214	Default IME	000A01D4			8C000000		Main	7E46C6F6	IME
0009022E	M	000D0214			8C000000		Main	FFFF02D5	MSCTIME UI
000F01EA	OK	000A01D4		000003EA	50010000	00000004	Main	7E43AFFE	Button
001001A4	Name			0000FFFF	50020000	00000004	Main	7E43E582	Static
001902A8				000003E9	50010080	00000204	Main	7E43B3B4	Edit
000B01CE	CrackMe v1.0			02DF01F3	1CCF0000	00000100	Main	00401128	No need to

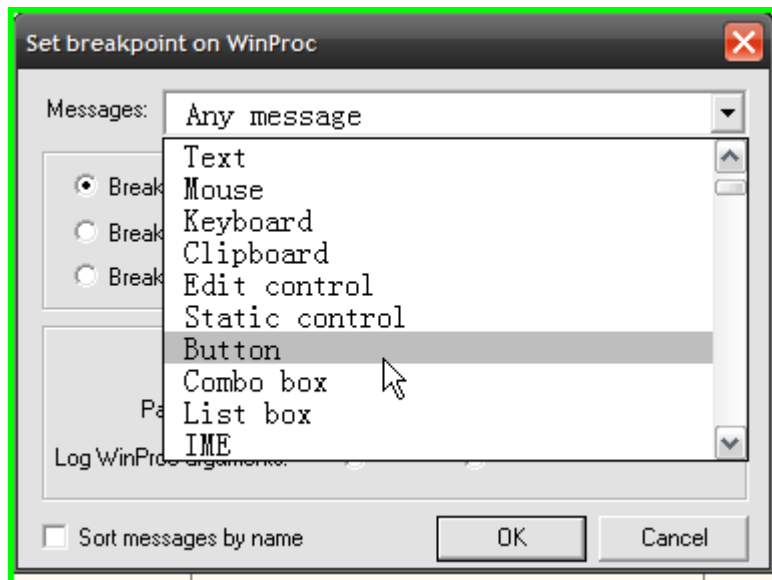
Actualize  
 Follow ClassProc  
 Toggle breakpoint on ClassProc  
 Conditional log breakpoint on ClassProc  
 Message breakpoint on ClassProc  
 Copy to clipboard  
 Sort by  
 Appearance

Cửa sổ cho phép ta thiết lập BP hiện ra :

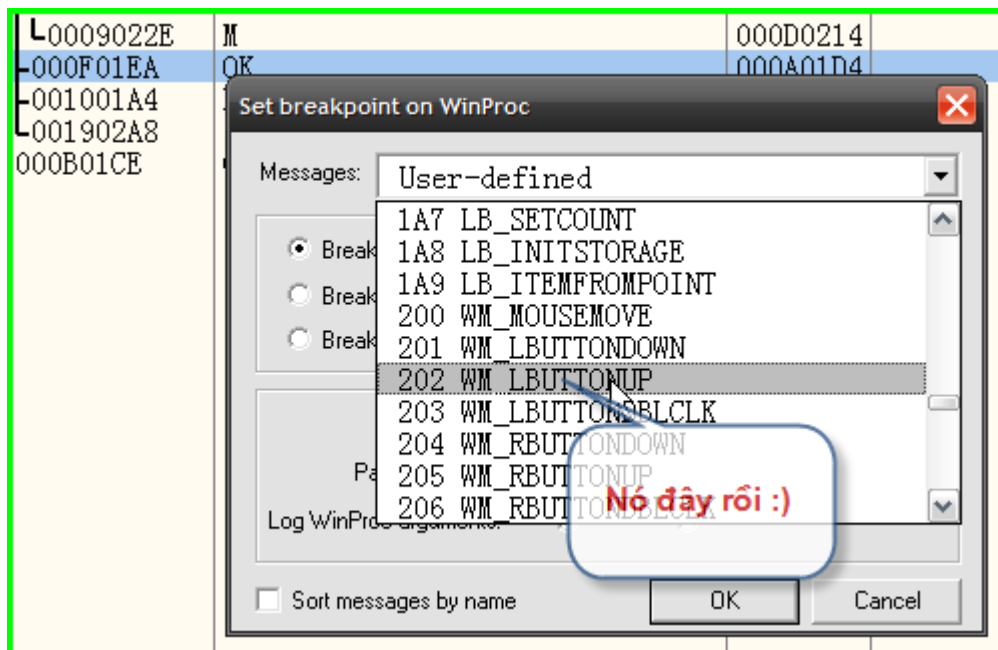




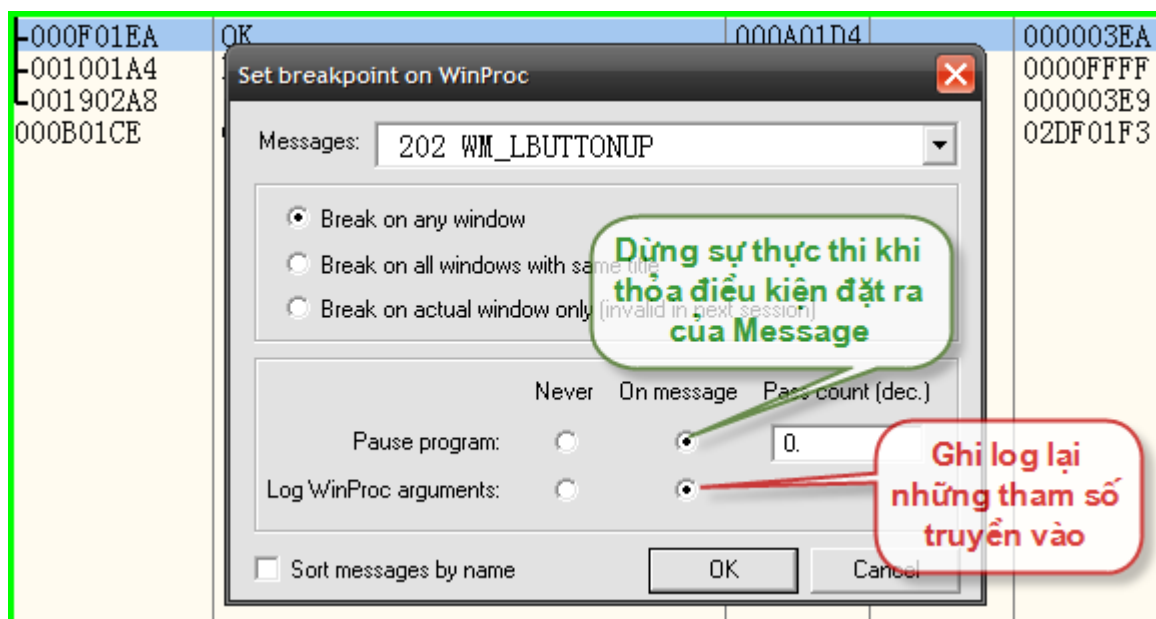
Ở phần Messages là một danh sách liệt kê những dạng Messages mà chúng ta có thể thiết lập BP :



Như các bạn thấy, Olly hỗ trợ đủ loại Messages từ Text, Mouse, Keyboard v...v.. Song bên cạnh đó nó còn hỗ trợ cho ta một loạt danh sách các Messages thông dụng nhất. Quay trở lại ví dụ của chúng ta: ta mong muốn khi ta nhấn chuột vào nút OK thì chương trình sẽ dừng sự thực thi. Vậy ta phân tích một chút, lúc ta bấm chuột mà cụ thể ở đây là chuột trái lên nút OK thì chương trình sẽ gửi một thông điệp là **WM\_LBUTTONDOWN** (**L** ở đây có nghĩa là Left, **BUTTONDOWN** có nghĩa là ta bấm chuột xuống). Lúc ta nhả chuột thì chương trình cũng sẽ gửi một thông điệp là **WM\_LBUTTONUP**. Do vậy trong trường hợp cụ thể của ta, ta sẽ nhờ Olly bắt thông điệp **WM\_LBUTTONUP** khi ta nhả chuột khỏi nút OK ☺. Trong phần Messages ta cuộn chuột xuống và tìm thấy :



Ta chọn nó và cấu hình lại như sau :



Sau khi cấu hình như trên ta nhấn OK :

Handle	Title	Parent	WinProc	ID	Style	ExtStyle	Thread	ClsProc	Class
000A01D4	Register	Topmost			14C800C4	00010101	Main	7E43E53F	#32770
000A0224		000A01D4		000003E3	50010080	00000204	Main	7E43B3B4	Edit
000C024E	Cancel	000A01D4		000003EB	50010000	00000000	Main	7E43AFFE	Button
000D01A2	Serial	000A01D4		0000FFFF	50020000	00000004	Main	7E43E582	Static
000D0214	Default IME	000A01D4			8C000000		Main	7E46C6F6	IME
0009022E	M	000D0214			8C000000		Main	FFFF02D5	MSCTIME UI
000F01EA	OK	000A01D4		000003EA	50010000	00000000	Main	7E43AFFE	Button
001001A4	Name	000A01D4		0000FFFF	50020000	00000004	Main	7E43E582	Static
001902A8		000A01D4		000003E9	50010080	00000204	Main	7E43B3B4	Edit
000B01CE	CrackMe v1.0	Topmost		02DF01F3	1CCF0000	00000100	Main	00401128	No need to disasm the code!

Ta sẽ thấy chỗ *ClsProc* của hai Button Cancel và OK được highlight. Bạn sẽ đặt câu hỏi là tại sao tôi đặt cho nút OK mà lại dính cả vào nút Cancel, đó là vì trong phần cấu hình BP ở trên bạn chọn là **Break on any window**, nếu bạn chọn là **Break on all windows with same title** thì sẽ có kết quả tương tự nhưng lúc đó điều kiện đặt BP sẽ khác, các bạn hãy tự mình khám phá thêm 😊. Bây giờ sau khi đặt BP như trên, ta nhấn nút OK để kiểm tra việc đặt BP :

```

PhantOm - [Professional Edition]
File View D3bug Pluginz Options Window Help Tools BreakPoint->
Paused
7E43AFFE 8BFF mov edi, edi
7E43B000 55 push ebp
7E43B001 8BEC mov ebp, esp
7E43B003 8B4D 08 mov ecx, dword ptr [ebp+8]
7E43B006 56 push esi
7E43B007 E8 C4D4FDFF call 7E4184D0

```

```

0013FCB8 7E418734 CALL to Assumed WinProc from USER32.7E418731
0013FCBC 000F01EA hWnd = 000F01EA ('OK',class='Button',parent=000A01D4)
0013FCC0 00000202 Message = WM_LBUTTONDOWN
0013FCC4 00000000 Keys = 0
0013FCC8 00170033 X = 51. Y = 23.
0013FCCC 7E43AFFE RETURN to USER32.7E43AFFE
0013FCD0 DCBAABCD
0013FCD4 00000000
0013FCD8 0013ED1C

```

Olly đã break ngay lập tức sau khi ta nhấn nút OK, quan sát cửa sổ Stack bạn có được thông tin như trên. Tại cửa sổ Log Window lúc này ta cũng có được các tham số truyền vào :

```

00E31000 Code size in header is 00009000, extending to size of section .text
Debugging information (Microsoft format) available
7E43AFFE CALL to Assumed WinProc from USER32.7E418731
hWnd = 000F01EA ('OK',class='Button',parent=000A01D4)
Message = WM_LBUTTONDOWN
Keys = 0
X = 51. Y = 23.
7E43AFFE Conditional breakpoint at USER32.7E43AFFE
Command
Start:402000,End:401FFF,Value:0

```

Ok vậy là quá trình thực thi của chương trình đã bị dừng lại và quyền điều khiển lúc này là của Olly. Tuy nhiên, như các bạn thấy thông thường khi dừng lại tại các hàm API ta luôn muốn tìm cách trở về đoạn code chính của chương trình, để rồi từ đó lần ra các manh mối tiếp theo. Vậy trong trường hợp này ta làm cách nào để quay về? Rất đơn giản các bạn nhấn **Alt + M** để mở cửa sổ Memory :

003E0000	00004000				Priv	RW	RW
003F0000	00002000				Map	R	R
00400000	00001000	CRACKME	PE header	Image	R	RWE	
00401000	00001000	CRACKME	CODE	code	Image	R	RWE
00402000	00001000	CRACKME	DATA	data	Image	R	RWE
00403000	00001000	CRACKME	.idata	imports	Image	R	RWE
00404000	00001000	CRACKME	.edata	exports	Image	R	RWE
00405000	00001000	CRACKME	.reloc	relocations	Image	R	RWE
00406000	00002000	CRACKME	.rsrc	resources	Image	R	RWE

Các bạn biết rằng chương trình của chúng ta sẽ thực thi code tại section bắt đầu từ 0x00401000, do đó để quay về làm việc với code của chương trình ta đặt một BP tương tự như sau :

003E 0000	00004000				PE header	PF10 Map Imag	R R	R RWE
003F 0000	00002000							
0040 0000	00001000	CRACKME						
00401000	00001000	CRACKME	CODE					
00402000	00001000	CRACKME	DATA		Actualize			
00403000	00001000	CRACKME	.idata		View in Disassembler		Enter	
00404000	00001000	CRACKME	.edata		Dump in CPU			
00405000	00001000	CRACKME	.reloc		Dump			
00406000	00002000	CRACKME	.rsrc		Search		Ctrl+B	
00410000	00008000							
004D 0000	00002000				Set break-on-access		F2	
004E 0000	00103000							
005F 0000	000C1000				Set memory breakpoint on access			
008F 0000	00010000				Set memory breakpoint on write			
00CF 0000	0000E000				Set access			

Sau đó nhấn **F9** để thực thi chương trình và chúng ta sẽ dừng lại tại đây :

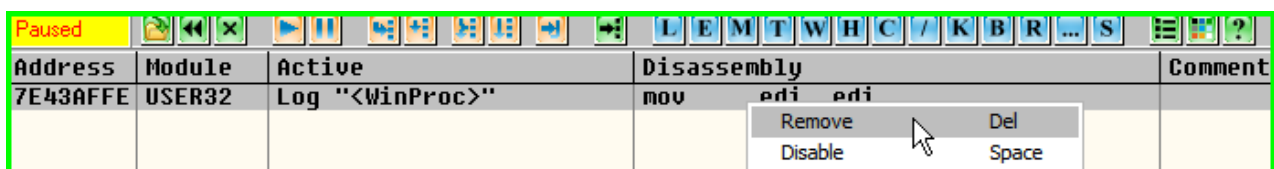
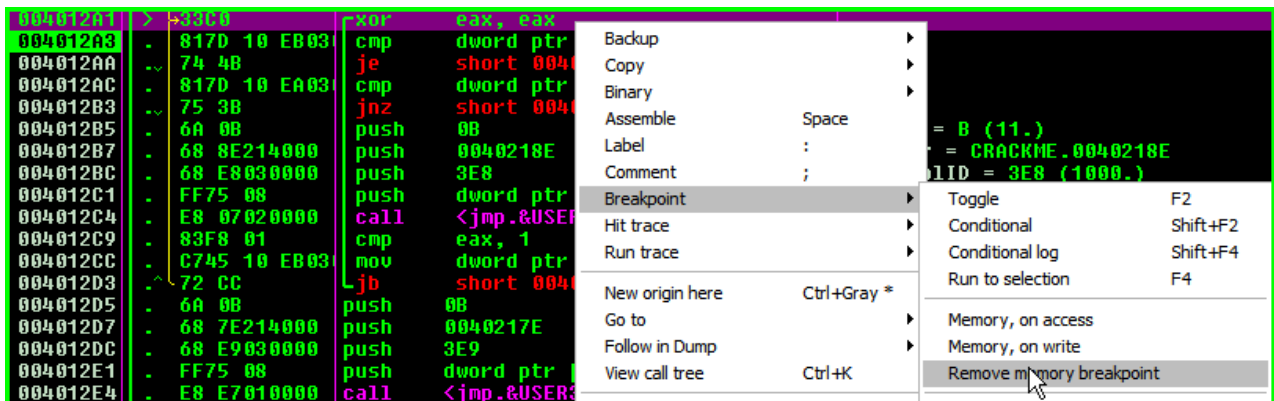
00401253	~> C8 000000	enter	0, 0		
00401257	~> 53	push	ebx		
00401258	~> 56	push	esi		
00401259	~> 57	push	edi		
0040125A	~> 817D 0C 1001	cmp	dword ptr [ebp+C], 110		
00401261	~> 74 34	je	short 00401297		
00401263	~> 817D 0C 1101	cmp	dword ptr [ebp+C], 111		
0040126A	~> 74 35	je	short 004012A1		
0040126C	~> 837D 0C 10	cmp	dword ptr [ebp+C], 10		
00401270	~> 0F84 81000000	je	004012F7		
00401276	~> 817D 0C 0102	cmp	dword ptr [ebp+C], 201		
0040127D	~> 74 0C	je	short 0040128B		
0040127F	~> B8 00000000	mov	eax, 0		
00401284	> 5F	pop	edi		
00401285	~> 5E	pop	esi		
00401286	~> 5B	pop	ebx		
00401287	~> C9	leave			
00401288	~> C2 1000	retn	10		
0040128B	> 6A 01	push	1		Erase = TRUE
0040128D	~> 6A 00	push	0		pRect = NULL
0040128F	~> FF75 08	push	dword ptr [ebp+8]		hWnd
00401292	~> E8 B5010000	call	<jmp.&USER32.InvalidRect>		InvalidRect
00401297	> FF75 08	push	dword ptr [ebp+8]		hWnd
0040129A	~> E8 95010000	call	<jmp.&USER32.SetFocus>		SetFocus
0040129F	~> EB E3	jmp	short 00401284		
004012A1	> 33C0	xor	eax, eax		
004012A3	~> 817D 10 EB03	cmp	dword ptr [ebp+10], 3EB		
004012AA	~> 74 48	je	short 004012F7		
004012AC	~> 817D 10 EA03	cmp	dword ptr [ebp+10], 3EA		
004012B3	~> 75 3B	jnz	short 004012F0		
004012B5	~> 6A 0B	push	0B		Count = B (11.)
004012B7	~> 68 8E214000	push	0040218E		Buffer = CRACKME.0040218E
004012BC	~> 68 E8030000	push	3E8		ControlID = 3E8 (1000.)
004012C1	~> FF75 08	push	dword ptr [ebp+8]		hWnd
004012C4	~> E8 07020000	call	<jmp.&USER32.GetDlgItemTextA>		GetDlgItemTextA

Đừng vội xóa bỏ Memory BP, ta tiếp tục nhấn **F9** để thực thi chương trình. Bạn sẽ thấy lúc này chương trình sẽ lần lượt thực thi từng lệnh một, sau khi thực thi xong lệnh `retn 10` bạn sẽ tới đây:

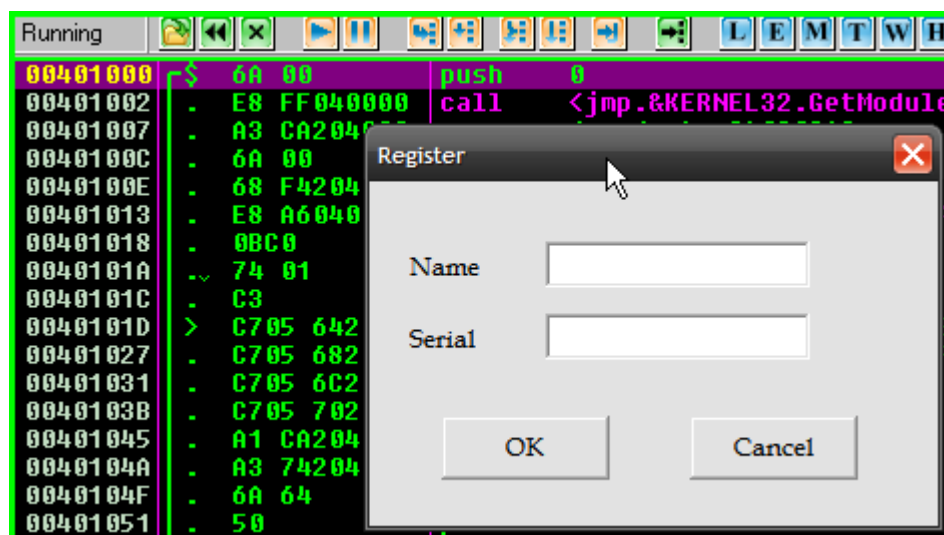
0040129F	~> EB E3	jmp	short 00401284		
004012A1	> 33C0	xor	eax, eax		
004012A3	~> 817D 10 EB03	cmp	dword ptr [ebp+10], 3EB		
004012AA	~> 74 48	je	short 004012F7		
004012AC	~> 817D 10 EA03	cmp	dword ptr [ebp+10], 3EA		
004012B3	~> 75 3B	jnz	short 004012F0		
004012B5	~> 6A 0B	push	0B		Count = B (11.)
004012B7	~> 68 8E214000	push	0040218E		Buffer = CRACKME.0040218E
004012BC	~> 68 E8030000	push	3E8		ControlID = 3E8 (1000.)
004012C1	~> FF75 08	push	dword ptr [ebp+8]		hWnd
004012C4	~> E8 07020000	call	<jmp.&USER32.GetDlgItemTextA>		GetDlgItemTextA
004012C9	~> 83F8 01	cmp	eax, 1		
004012CC	~> C745 10 EB03	mov	dword ptr [ebp+10], 3EB		
004012D3	~> 72 CC	jb	short 004012A1		
004012D5	~> 6A 0B	push	0B		Count = B (11.)
004012D7	~> 68 7E214000	push	0040217E		Buffer = CRACKME.0040217E
004012DC	~> 68 E9030000	push	3E9		ControlID = 3E9 (1001.)
004012E1	~> FF75 08	push	dword ptr [ebp+8]		hWnd
004012E4	~> E8 E7010000	call	<jmp.&USER32.GetDlgItemTextA>		GetDlgItemTextA

Để ý bạn sẽ thấy rằng chúng ta đang ở tại chỗ sắp sửa thực thi hàm API **GetDlgItemTextA**, mà hàm này sẽ lấy thông tin về UserName và Serial ta nhập vào để lưu vào vùng Buffer. Qua đây ta thấy rằng không cần sử dụng đến phương pháp đặt bp tại hàm API ta cũng có thể thông qua Messages BP để lần tới những điểm quan trọng.

Tuy nhiên, giả sử trong một trường hợp nào đó mà chương trình không sử dụng hàm API để lấy text do ta nhập vào thì ta làm thế nào, lúc đó ta cần sử dụng đến Messages BP để tiếp cận mục tiêu. Ta sẽ thực hiện một demo nhỏ, trước tiên xóa Memory BP và Message BP đã đặt trước đó đi :



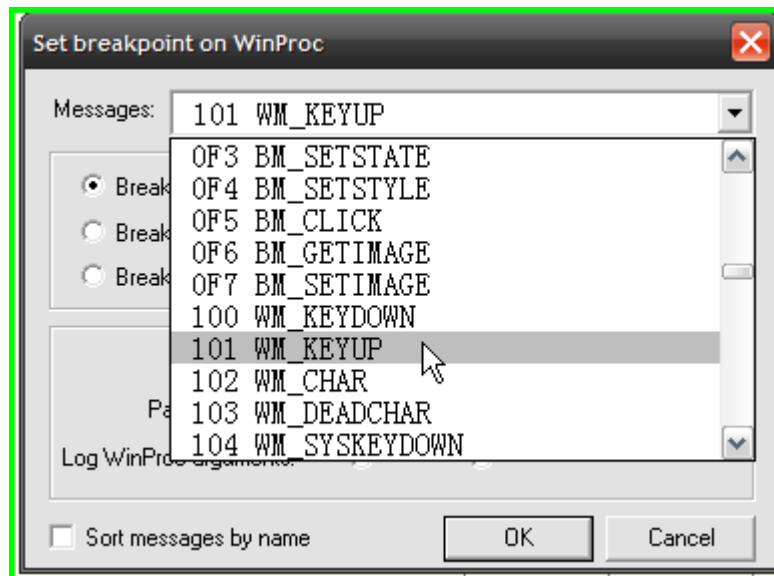
Restart lại Olly và cho thực thi chương trình :



Ở đây tôi chưa nhập thông tin gì vội, chuyển qua cửa sổ Windows và chọn Actualize :

Handle	Title	Parent	WinProc	ID	Style	ExtStyle	Thread	ClsProc	Class
000F01CE	CrackMe v1.0	Topmost		019E01C7	1CCF0000	00000100	Main	00401128	No need to disasm the code!
001101D4	Register	Topmost		000003EB	14C800C4	00010101	Main	7E43E53F	#32770
000C022E	Cancel	001101D4		000003EB	50010000	00000004	Main	7E43AFFE	Button
00110214	Default IME	001101D4			8C000000		Main	7E46C6F6	IME
0000E0224	M	00110214			8C000000		Main	FFFF03E3	MSCTIME UI
0011024E		001101D4		000003E9	50010080	00000204	Main	7E43B3B4	Edit
001201A2	Serial	001101D4		0000FFFF	50020000	00000004	Main	7E43E582	Static
001301EA		001101D4		000003E8	50010080	00000204	Main	7E43B3B4	Edit
001501A4	Name	001101D4		0000FFFF	50020000	00000004	Main	7E43E582	Static
001E02A8	OK	001101D4		000003EA	50010000	00000004	Main	7E43AFFE	Button

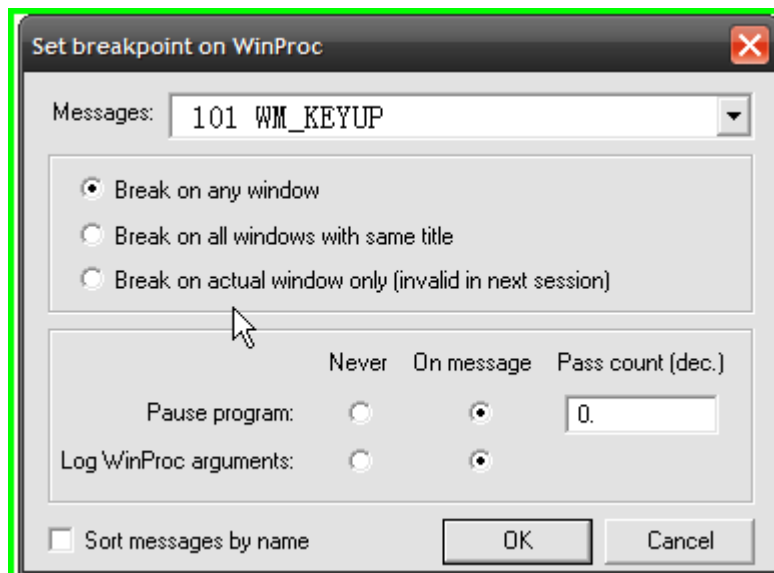
Tại lần minh họa này tôi thực hiện đặt một Message BP như sau :



Các bạn sẽ hỏi tại sao tôi chọn **WM\_KEYUP**, đơn giản là vì khi tôi gõ một kí tự bất kì và nhả phím thì sẽ có một thông điệp **WM\_KEYUP** sinh ra. Tôi muốn Olly bắt lấy thông điệp này và dừng sự thực thi của chương trình lại. Thông tin thêm về **WM\_KEYUP** :

The WM\_KEYUP message is posted to the window with the keyboard focus when a nonsystem key is released. A nonsystem key is a key that is pressed when the ALT key is not pressed, or a keyboard key that is pressed when a window has the keyboard focus.

```
WM_KEYUP
nVirtKey = (int) wParam;    // virtual-key code
lKeyData = lParam;         // key data
```







Address	Module	Active	Disassembly
7E43AFFE	USER32	Log "<WinProc>"	mov edi, edi

Remove Del  
Disable Space  
Edit condition  
Follow in Disassembler Enter  
Disable all

Address	Module	Active	Disassembly
7E43AFFE	USER32	Log "<WinProc>"	mov edi, edi

**Modify conditional log breakpoint at USER32.7E43AFFE**

Condition:  
[ESP+8]==WM\_LBUTTONDOWN

Explanation: <WinProc> Expression: =

Decode value of expression as: Assumed by expression

	Never	On condition	Always	Pass count (dec.)
Pause program:	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	0.
Log value of expression:	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	
Log function arguments:	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	

If program pauses, pass following commands to plugins:

OK Cancel

Ồ.. vậy bản chất của Message BP thực ra là một Conditional Log BP, trong đó điều kiện để dừng sự thực thi của chương trình là  $[ESP+8]==WM\_LBUTTONUP$  (tức là  $[ESP+8]==202$ ). Lúc này ta để ý cửa sổ Stack :

0013FCB8	7E418734	CALL to Assumed WinProc from USER32.7E418731
0013FCBC	00140224	hWnd = 00140224 ('OK',class='Button',parent=001C022E)
0013FCC0	00000202	Message = WM_LBUTTONDOWN
0013FCC4	00000000	Keys = 0
0013FCC8	0010003C	LX = 60. Y = 16.
0013FCCC	7E43AFFE	RETURN to USER32.7E43AFFE

Giá trị tại  $[esp + 8]$  đúng là 202, để cho rõ ràng hơn bạn nhấp đúp chuột tại cột chứa giá trị của ESP sẽ có được như sau :

\$ ==>	7E418734	CALL to Assumed WinProc from USER32.7E418731
\$+4	00140224	hWnd = 00140224 ('OK',class='Button',parent=001C022E)
\$+8	00000202	Message = WM_BUTTONUP
\$+C	00000000	Keys = 0
\$+10	0010003C	X = 60. Y = 16.
\$+14	7E43AFFE	RETURN to USER32.7E43AFFE
\$+18	DCBAABCD	
\$+1C	00000000	

“\$+8” ở đây chính là “ESP + 8”. Hehe qua đó tôi biết chắc một điều rằng, giá trị [esp + 8] sẽ chứa giá trị của các Messages. Vậy để dò các giá trị tôi sửa lại BP như sau :

Address	Module	Active	Disassembly
7E43AFFE	USER32	Log "<WinProc>"	mov edi, edi

Modify conditional log breakpoint at USER32.7E43AFFE

Condition:

Explanation: <WinProc> = Expression: [ESP+8]

Decode value of expression as: Assumed by expression

☐ Never    ☐ On condition    ☐ Always    Pass count (dec.) 0.

Pause program: ☒    ☐    ☐

Log value of expression: ☐    ☐    ☒

Log function arguments: ☐    ☐    ☒

If program pauses, pass following commands to plugins:

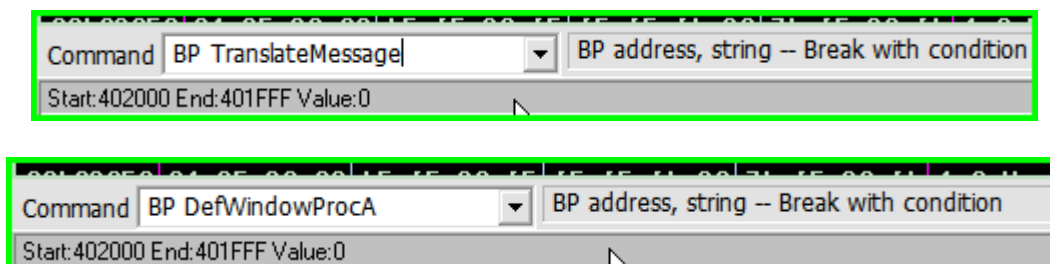
OK Cancel

Ý nghĩa của những thiết lập trong hình trên tôi không cần phải giải thích lại nữa ☺. Sửa lại BP xong tôi cho thực thi chương trình và nhập thông tin vào. Tiếp theo bấm OK và chuyển qua cửa sổ Log để quan sát các giá trị mà tôi thu được :

Address	Message
	hWnd = 001E022E ('OK', class='Button', parent=002501CE) Message = BM_SETSTYLE Style = BS_DEFPUSHBUTTON Redraw = TRUE
7E43AFFE	COND: <WinProc> = 00000201
7E43AFFE	CALL to Assumed WinProc from USER32.7E418731 hWnd = 001E022E ('OK', class='Button', parent=002501CE) Message = WM_LBUTTONDOWN Keys = MK_LBUTTON X = 44, Y = 12.
7E43AFFE	COND: <WinProc> = 00000281
7E43AFFE	CALL to Assumed WinProc from USER32.7E418731 hWnd = 001E022E ('OK', class='Button', parent=002501CE) Message = WM_IME_SETCONTEXT wParam = 1 lParam = C000000F
7E43AFFE	COND: <WinProc> = 00000007
7E43AFFE	CALL to Assumed WinProc from USER32.7E418731 hWnd = 001E022E ('OK', class='Button', parent=002501CE) Message = WM_SETFOCUS hWndLose = 001D024E (class='Edit', parent=002501CE) lParam = 0
7E43AFFE	COND: <WinProc> = 000000F3
7E43AFFE	CALL to Assumed WinProc from USER32.7E418731 hWnd = 001E022E ('OK', class='Button', parent=002501CE) Message = BM_SETSTATE Highlight = TRUE lParam = 0
7E43AFFE	COND: <WinProc> = 00000202
7E43AFFE	CALL to Assumed WinProc from USER32.7E418731 hWnd = 001E022E ('OK', class='Button', parent=002501CE) Message = WM_LBUTTONUP Keys = 0 X = 44, Y = 12.
7E43AFFE	COND: <WinProc> = 000000F3
7E43AFFE	CALL to Assumed WinProc from USER32.7E418731

Khà khà nhiều quá trời, để ý các bạn thấy là chương trình này xử lý hai Message là **WM\_LBUTTONDOWN(201)** và **WM\_LBUTTONUP(202)**, đồng thời ta cũng thấy là nó không hề xử lý các Messages như **WM\_KEYUP** hay **WM\_KEYDOWN** ☺.

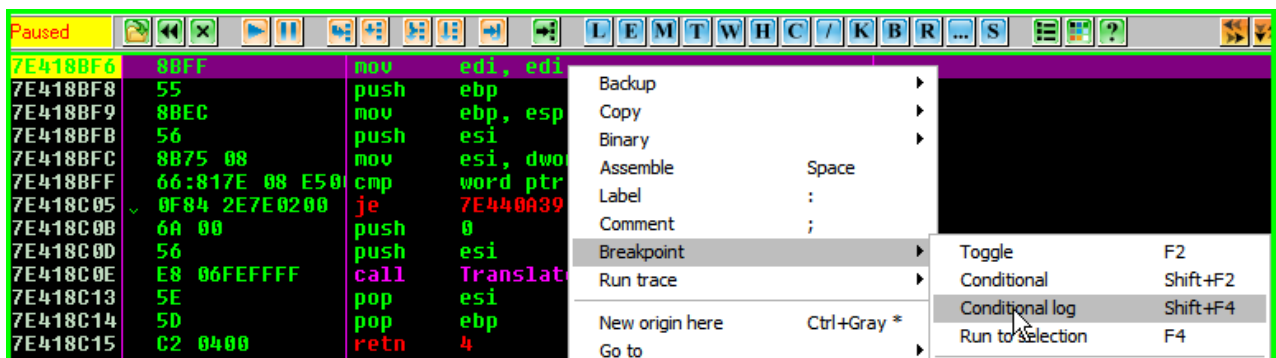
Ngoài ra để kiểm soát toàn bộ các Message cho tất cả các chương trình chúng ta có thể đặt một BP conditional log tại các hàm APIs chuyên kiểm soát các Messages. Hai hàm API đó là **TranslateMessage** và **DefWindowProcA**. Ta thực hiện như sau, tại command bar gõ hai lệnh :



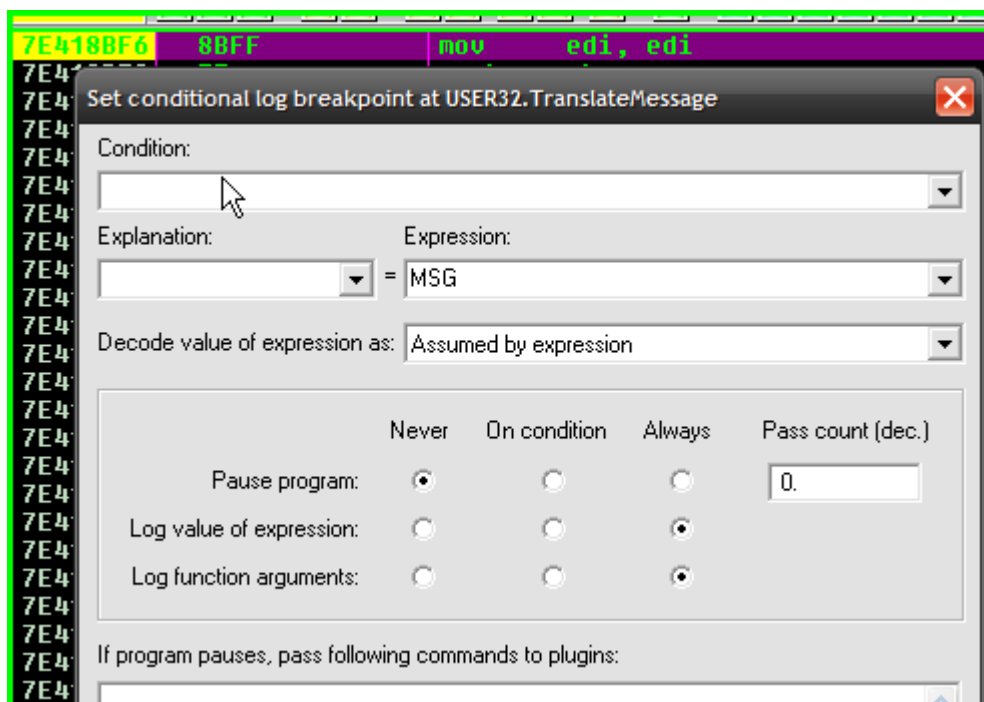
Mở cửa sổ quản lý BP, chọn BP đầu tiên, chuột phải và chọn FOLLOW IN DISSASSEMBLER :

Address	Hex	Assembly
7E418BF6	8BFF	mov edi, edi
7E418BF8	55	push ebp
7E418BF9	8BEC	mov ebp, esp
7E418BFB	56	push esi
7E418BFC	8B75 08	mov esi, dword ptr [ebp+8]
7E418BFF	66:817E 08 E50	cmp word ptr [esi+8], 0E5
7E418C05	0F84 2E7E0200	je 7E440A39
7E418C0B	6A 00	push 0
7E418C0D	56	push esi
7E418C0E	E8 06FEFFFF	call TranslateMessageEx
7E418C13	5E	pop esi

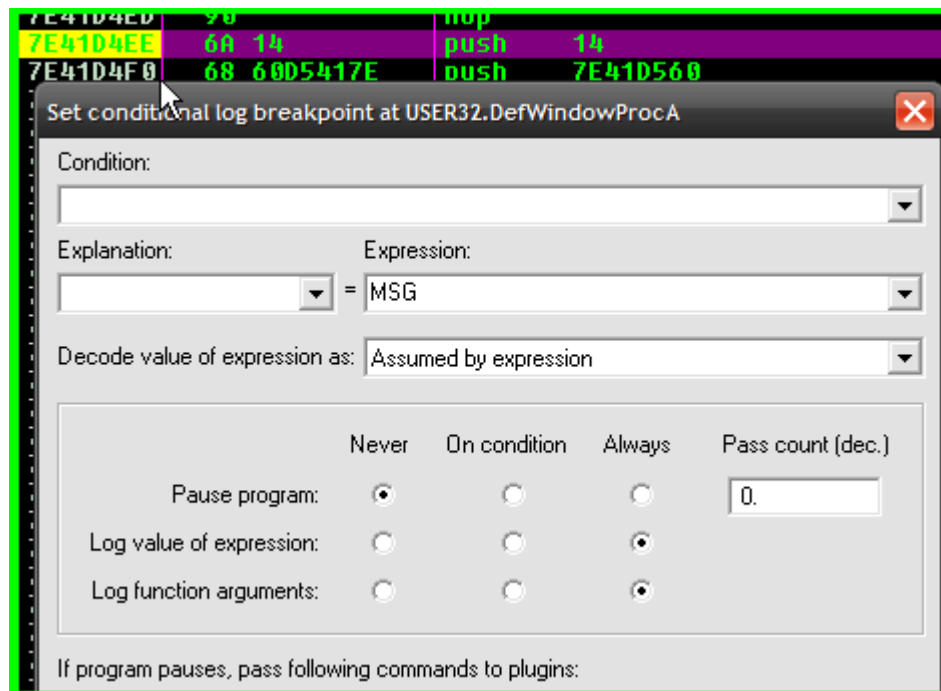
Sau đó lại chuột phải tiếp và chọn :



Đặt một Conditional Log BP như sau :



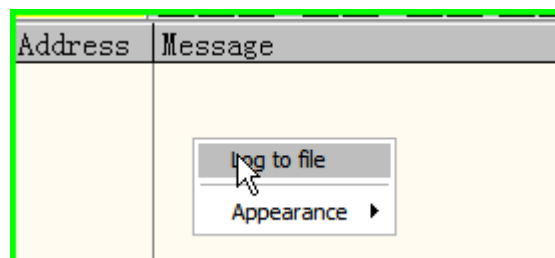
Thực hiện tương tự với BP thứ hai :



Ok như vậy là ta đã chuyển hết hai BP sang dạng Conditional Log BP rồi :

Address	Module	Active	Disassembly
7E418BF6	USER32	Log	mov edi, edi
7E41D4EE	USER32	Log	push 14

Tiếp theo ta sẽ cấu hình cửa sổ Log để lưu toàn bộ thông tin vào một text file để tiện theo dõi. Chuyển qua cửa sổ Log, clear hết Log cũ đi sau đó chuột phải và chọn :



Lưu với tên file bất kỳ mà bạn muốn, cuối cùng ta cho thực thi chương trình :



Running	
Address	Message
7E41D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 004400C0 ('CrackMe v1.0', class='No need to disasm the code!')
	Message = WM_PAINT wParam = 0 lParam = 0
7E41D4EE	COND: 00000086
7E41D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 004400C0 ('CrackMe v1.0', class='No need to disasm the code!')
	Message = WM_NCACTIVATE Active = FALSE lParam = 0
7E41D4EE	COND: 00000006
7E41D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 004400C0 ('CrackMe v1.0', class='No need to disasm the code!')
	Message = WM_ACTIVATE WA_INACTIVE Minimized = 0 hWnd = NULL
7E41D4EE	COND: 0000001C
7E41D4EE	CALL to DefWindowProcA from 755D41B3 hWnd = 004401A4 ('M', class='MSCTIME UI', parent=003A0072)
	Message = WM_ACTIVATEAPP Activate = FALSE ThreadId = 288
7E41D4EE	COND: 0000001C
7E41D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 004400C0 ('CrackMe v1.0', class='No need to disasm the code!')
	Message = WM_ACTIVATEAPP Activate = FALSE ThreadId = 288
7E41D4EE	COND: 00000008
7E41D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 004400C0 ('CrackMe v1.0', class='No need to disasm the code!')
	Message = WM_KILLFOCUS hWndGet = NULL lParam = 0
7E41D4EE	COND: 00000081

Sau đó quan sát cửa sổ Log bạn sẽ thấy có rất nhiều Windows Message được xử lý ☺. Đóng chương trình lại và kiểm tra log file của chúng ta xem ta thu được gì nào :

1	7E41D4EE	COND: 00000081
2	7E41D4EE	CALL to DefWindowProcA from CRACKME.0040118C
3		hWnd = 004400C0 (class='No need to disasm the code!')
4		Message = WM_NCCREATE
5		wParam = 0
6		pCreate = 0013F9A8
7	773D0000	Module C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.2982_x-ww_ac3f9c03\comctl32.dll
8	773D1000	Code size in header is 00090C00, extending to size of section '.text'
9		Debugging information (Microsoft format) available
10	77B40000	Module C:\WINDOWS\system32\apphelp.dll
11	77B41000	Code size in header is 0001C800, extending to size of section '.text'
12		Debugging information (Microsoft format) available
13	7E41D4EE	COND: 00000083
14	7E41D4EE	CALL to DefWindowProcA from CRACKME.0040118C
15		hWnd = 004400C0 ('CrackMe v1.0', class='No need to disasm the code!')
16		Message = WM_NCCALCSIZE
17		CalcFlag = FALSE
18		Data = 0013F9F4
19	7E41D4EE	COND: 00000018

Rất đầy đủ và chi tiết ☺.

Ok vậy là phần 12 của loạt tuts về Ollydbg đến đây là kết thúc, qua bài viết này tôi đã giới thiệu nốt cho bạn loại BP cuối cùng đó là **Message Breakpoints**, việc nắm được ý nghĩa và mục đích của từng loại Windows Message sẽ giúp ta rất nhiều trong quá trình tiếp cận mục tiêu. Giả sử như

trong trường hợp ta làm việc với Nag mà xử lý Message **WM\_CLOSE** thì ta có thể đặt một Message BP liên quan tới **WM\_CLOSE** để lần ra manh mối. Hi vọng qua bài viết này tôi đã truyền tải tới các bạn những kiến thức mà có thể đến bây giờ bạn mới biết khi sử dụng OllyDbg ☺. Trong bài viết tiếp theo của loạt tuts này chắc chắn tôi sẽ cùng bạn hoàn thành nốt quá trình tìm ra Serial cho cái Crackme mà chúng ta đã làm việc từ đầu tới giờ....sẽ có nhiều điều thú vị lắm. 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, QHQCrk, the\_Lighthouse, Merc, Hoadongnoi, Nini ... all REA's members, TQN, HacNho, RongChauA, Deux, tlandn, light.phoenix, dqtl, ARTEAM .... all my friend, and YOU.

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

iamidiot, WhyNotBar, trickyboy, dzungltn, 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 Rogers** for his great site, Reversing.be folks(especially **haggar**), Arteam folks(**Shub-Nigurrath**, **MaDMAn\_H3rCuL3s**) and all folks on crackmes.de, thank to all members of **unpack.cn** (especially **fly** and **linhanshi**). Great thanks to **lena151**(I like your tutorials).Thanx to Orthodox, kanxue, TiGa and finally, thanks to **RICARDO NARVAJA** and all members on **CRACKSLATINOS**.

>>>> If you have any suggestions, comments or corrections email me:

**[kienmanowar\[at\]reaonline.net](mailto:kienmanowar[at]reaonline.net)**