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, chúng ta lại gặp nhau trong phần 11 của loạt bài viết về OllyDbg. Lần trước, khi release phần 10 trên hai site : **REA(reaonline.net)** và **HVA(hvaonline.net)** nhận thấy lượng download rất nhiều, chứng tỏ rằng các bài viết của tôi vẫn được các bạn quan tâm và ủng hộ. Đó chính là nguồn động viên lớn cho tôi để tôi tiếp tục viết tiếp các phần tiếp theo.Tiện đây tôi cũng xin lan man một chút, trước đây tôi cũng đã từng có một thời gian tham gia việc training và được quen biết với những đồng nghiệp mà sau này vẫn là những anh em tốt của tôi, lúc đó mặc dù công việc của chúng tôi là giảng dạy kiến thức cho người khác nhưng ***người sếp (người anh cả của chúng tôi)*** đã định hướng cho chúng tôi rằng: *"Chúng ta đứng trên bục giảng không có nghĩa chúng ta tự coi mình là thầy của người khác, không có nghĩa chúng ta là người hiểu biết hơn những người đang ngồi nghe chúng ta nói, mà chúng ta chỉ là những người truyền đạt lại những kiến thức mà chúng ta biết, khơi gợi cho học viên khả năng tự tìm tòi và tự nghiên cứu.Quan hệ giữa học viên và giảng viên là quan hệ hoàn toàn bình đẳng, mọi vấn đề được đem ra trao đổi thẳng thắn theo phương pháp phản biện như thế mới tạo tâm lý thoải mái cho người học, khiến cho buổi học không phải là nơi truyền đạt kiến thức theo kiểu một chiều như những gì chúng ta thấy trên các trường lớp ở Việt Nam"*.
Qua loạt tuts này cũng vậy, kiến thức tôi truyền tải cho các bạn chưa chắc đã đúng 100%, đó chỉ là những gì cá nhân tôi tích lũy được và truyền tải lại cho các bạn, cho nên nếu trong quá trình các bạn đọc thấy có những kiến thức nào tôi viết sai hoặc chưa đúng thì cứ góp ý thẳng thắn, vì như thế mới chứng tỏ các bạn thật sự quan tâm tới bộ tài liệu này. Ở phần trước tôi đã giới thiệu sơ qua về cách thiết lập BP, các thao tác thông qua command bar để đặt các bp với các lệnh BP và BPX, cung cấp thông tin về việc đặt memory bp và cách xử lý để có thể đặt bp nếu như chương trình sử dụng cơ chế anti-bp. Trong phần 11 này các bạn sẽ tìm hiểu thêm về các dạng Break Points khác như **Hardware Breakpoints**, **Conditional** và **Message BreakPoints**, những điều thú vị đang nằm ở phía trước…. N0w….L3t's G0!!!!!!!!!

II. BreakPoints in OllyDbg

1.Hardware Breakpoints :

Hardware Breakpoint là gì nhỉ? Tự nhiên nếu có ai hỏi tôi một câu hỏi như vậy chắc tôi cũng không biết phải giải thích thế nào. Ngay cả trong bài viết của lão làng Ricardo Navarja cũng không giải thích chi tiết về nó. Vậy là phải tự mình tìm hiểu rồi, đọc trong help file của Olly thì chỉ nhận được một chút thông tin như sau :

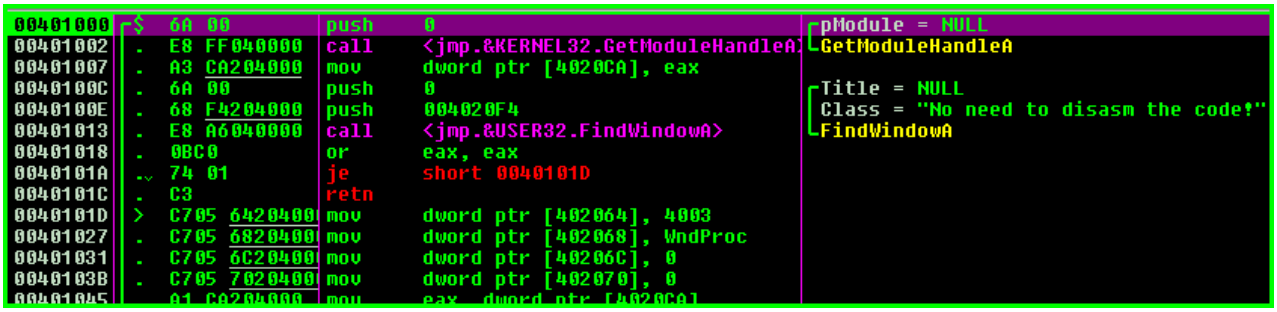
Hardware breakpoint (available only when running Debugger under Windows ME, NT, 2000 or XP). 80x86-compatible processors allow you to set 4 hardware breakpoints. Unlike memory

breakpoint, hardware breakpoints do not slow down the execution speed, but cover only up to 4 bytes. OllyDbg can use hardware breakpoints instead of INT3 when stepping or tracing through the code.

Điều đầu tiên ta thu thập được qua đoạn này là Hardware breakpoint (viết tắt HWBP) không thể sử dụng được nếu như bạn dùng Olly trên môi trường Windows 98. Điều thứ hai là chúng ta được phép thiết lập tới 4 HWBP, so với Memory BP là quá đã rồi vì như ta đã biết tại một thời điểm Olly chỉ cho phép có duy nhất một memory bp. Thêm vào đó HWBP không làm châm quá trình thực thi của chương trình. Điều cuối cùng ta nhận được là HWBP không sử dụng INT3, vậy thì nó sử dụng lệnh gì để dừng sự thực thi của chương trình?? Chúng ta tiếp tục tìm hiểu thêm vậy ☺. Đi lòng vòng một hồi tôi cũng có thêm được một chút thông tin: HWBP được hỗ trợ trực tiếp bởi CPU, sử dụng một số thanh ghi đặc biệt hay còn được gọi là debug registers. Có bốn thanh ghi đó là : **DR0**, **DR1**, **DR2**, **DR3**, bốn thanh ghi này sẽ được sử dụng để lưu giữ những địa chỉ mà ta thiết lập HWBP. Điều kiện của mỗi break points để cho dừng sự thực thi của chương trình lại được lưu trong một thanh ghi đặc biệt khác là CPU register, đó là thanh ghi **DR7**. Khi bất kì một điều kiện nào thỏa mãn (TRUE) thì processor sẽ quăng một exception là **INT1** (khà khà vậy là nó dùng INT1 nhé) và quyền điều khiển lúc này sẽ được trả về cho trình Debug của chúng ta. Có bốn khả năng để dừng sự thực thi của một chương trình :

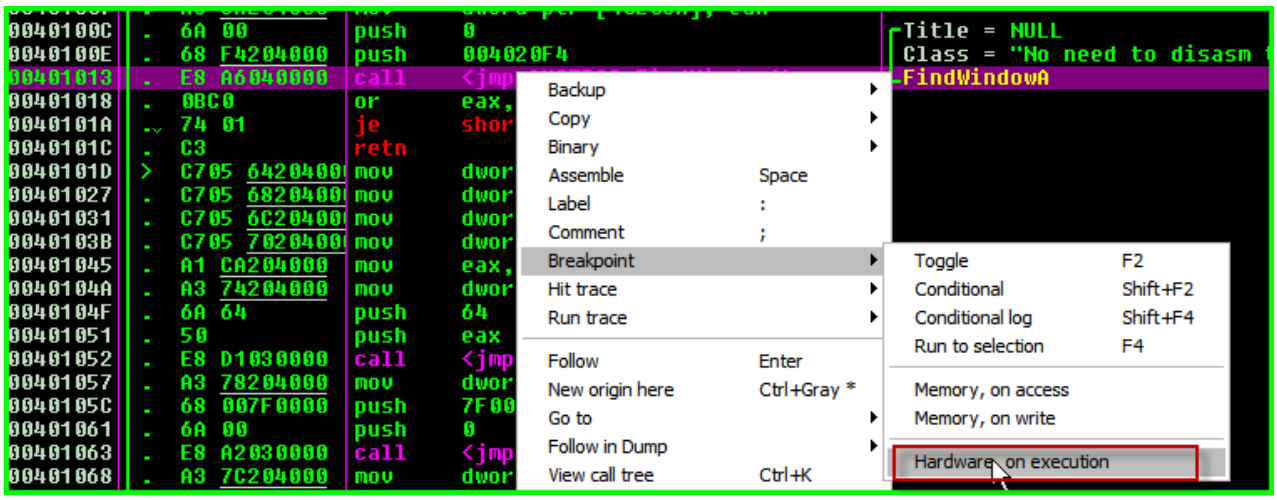
1. Khi một câu lệnh được thực thi.
2. Khi nội dung của memory có thay đổi (modified).
3. Khi một ví trí memory được đọc ra hoặc được cập nhật(updated).
4. Khi một input-output port được tham chiếu tới. (cái này tôi cũng chưa tìm hiểu).

Khả năng của tôi cũng chỉ biết giải thích đến như thế, các bạn muốn tìm hiểu thêm vui lòng tìm đọc các tài liệu khác. Bây giờ là phần thực hành, mở Olly lên và load crackme vào nào :

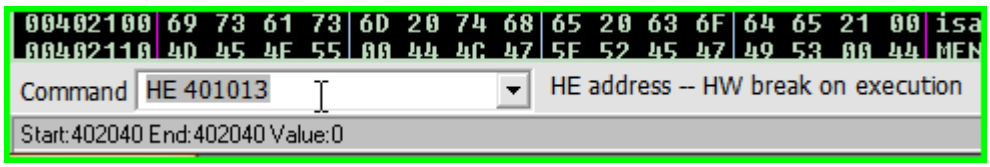


Ta sẽ thực hành lần lượt với các kiểu HWBP là : **HardWare Breakpoint on Execution**, **HWBP on Write** và **HWBP on Access**.

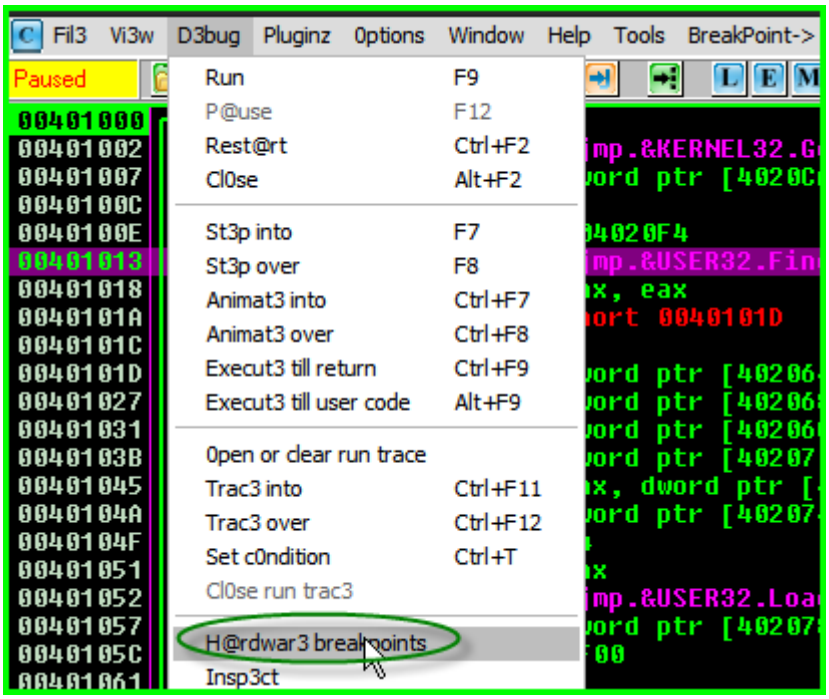
Việc đặt HWBP on Execution cũng gần tương tự như cách ta đặt BP thông thường tại một câu lệnh, chỉ có một điều hơi khác là HWBP không có thay đổi code do đó khiến cho việc phát hiện HWBP trở nên khó khăn hơn. Mặc dù vậy vẫn có những chương trình có khả năng sử dụng một số tricks để xóa HWBP, những kĩ thuật như vậy và cách thức chống lại nó chúng ta sẽ tìm hiểu trong các phần tiếp theo. Bây giờ quay trở lại vấn đề, ta muốn đặt một HWBP tại địa chỉ `0x00401013` thì làm như thế nào? Câu trả lời cực kì đơn giản, ta chọn địa chỉ đó nhấn chuột phải và chọn như sau :



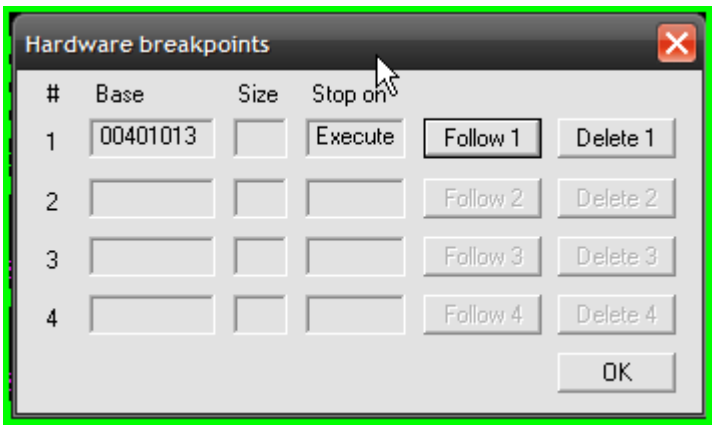
Hoặc một cách khác là thực hiện thông qua command line:



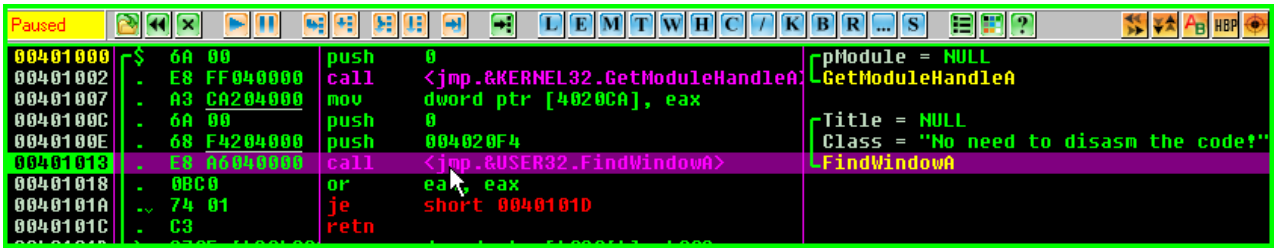
Chà ta đặt HWBP rồi mà chẳng thấy có dấu hiệu gì chứng tỏ là ta đã đặt cả. Có cách gì kiểm chứng không nhỉ? Xin thưa là có ☺, Olly hỗ trợ cho chúng ta một cửa sổ đặc biệt dùng để quản lý các HWBP, để mở cửa sổ này bạn làm như sau :



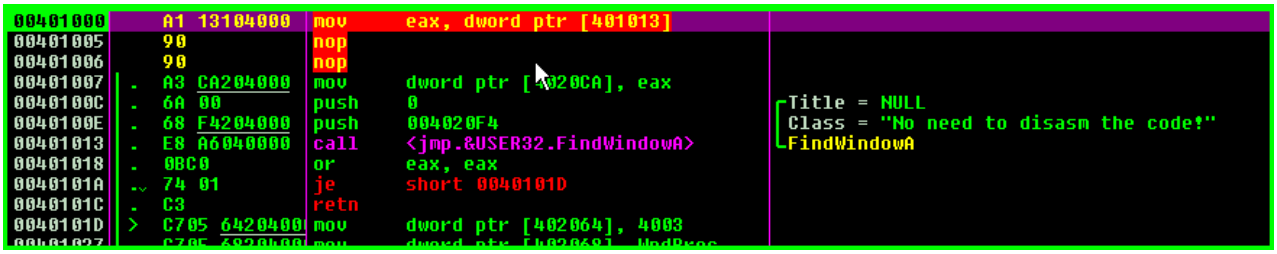
Ta sẽ nhận được một cửa sổ như sau :



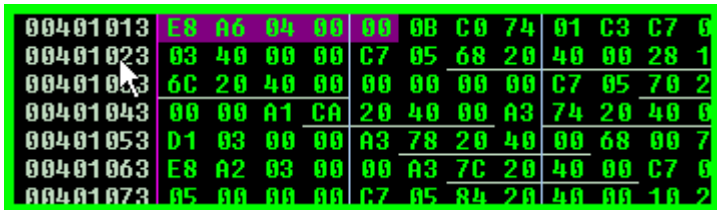
Không cần giải thích nhiều chắc các bạn cũng hiểu được ý nghĩa từng thành phần của cửa sổ HWBP. Giờ để tìm tới chỗ ta vừa đặt HWBP ta chỉ việc nhấn vào nút **Follow** là Olly sẽ di chuyển vệt sáng tới chỗ ta đặt BP. Bây giờ ta nhấn F9 để thực thi chương trình :



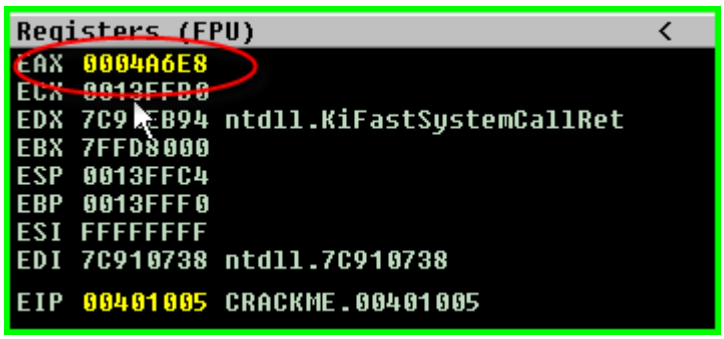
Bùm…Olly đã stop tại chỗ mà ta đặt BP. Ta thấy nó chẳng khác gì với việc ta đặt BP như ở phần 10, giờ ta kiểm chứng xem nó có thay đổi code gì không nhé. Để kiểm chứng ta làm hệt như những gì tôi đã hướng dẫn các bạn ở phần trước :



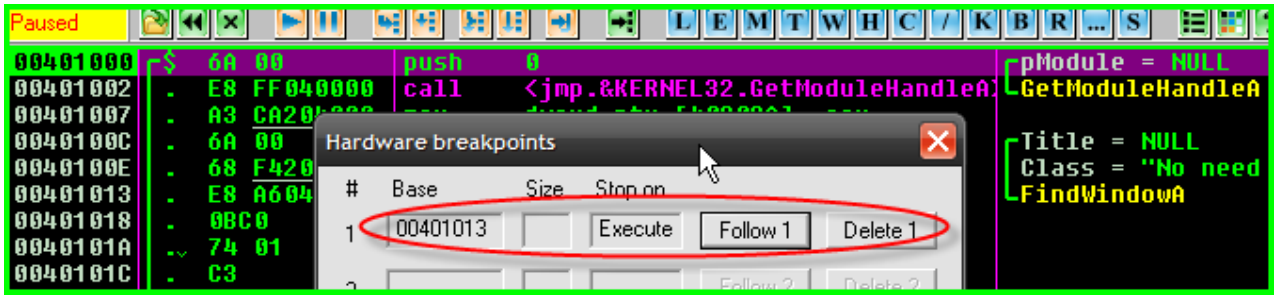
Chúng ta edit lại đoạn code như trên, dòm xuống cửa sổ Tip và cửa sổ Dump ta có được thông tin như sau :



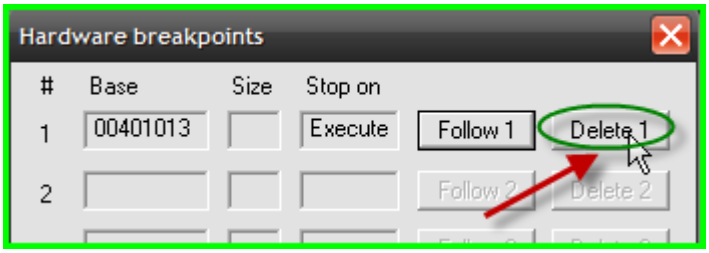
Giờ ta nhấn **F7** để trace và quan sát thanh ghi EAX :

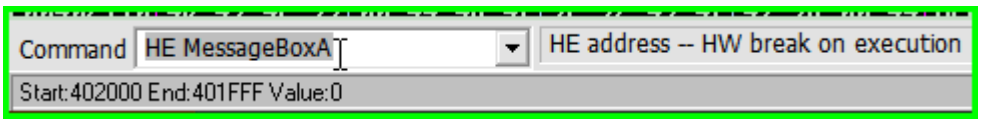


Ta thấy được gì nào, thanh ghi eax vẫn giữ nguyên giá trị được mov vào mà không thấy có thay đổi ☺, vậy chứng tỏ một điều HWBP không có change code. Giờ ta kiểm chứng một điều khác nữa, ở phần trước khi ta restart chương trình thì memory bp cũng theo đó mà mất luôn ☹, vậy HWBP có như thế không? Thử cái biết liền, ta nhấn **Ctrl+F2** để restart chương trình sau đó vào cửa sổ HWBP để kiểm tra xem còn lưu lại BP không :

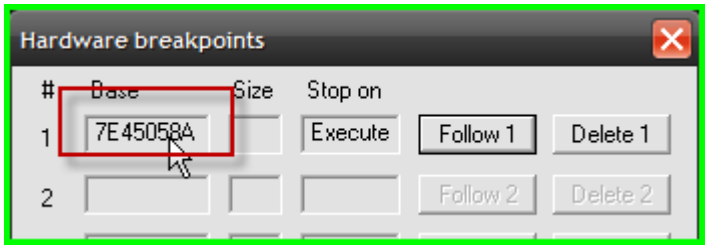


Oh vẫn y nguyên nhé!! Giờ ta xóa HWBP này đi, sau đó đặt một HWBP tại MessageBoxA như ta thực hiện bằng lệnh BPX.





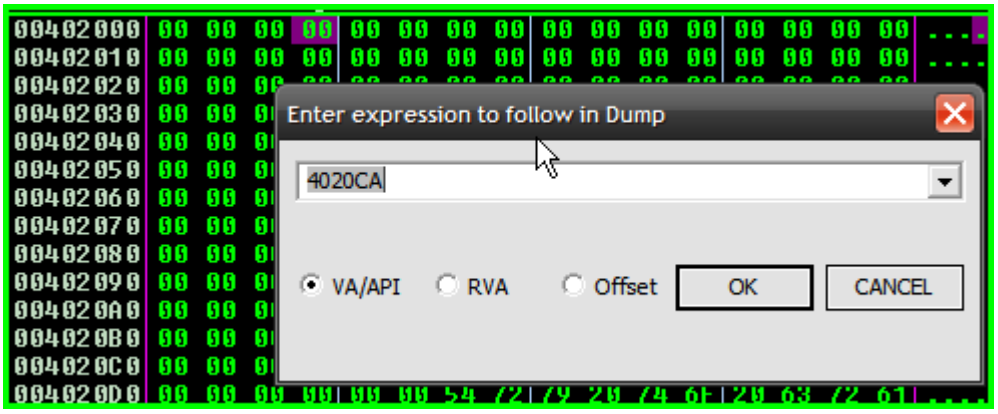
Kiểm tra chắc chắn xem bp đã đặt được chưa :



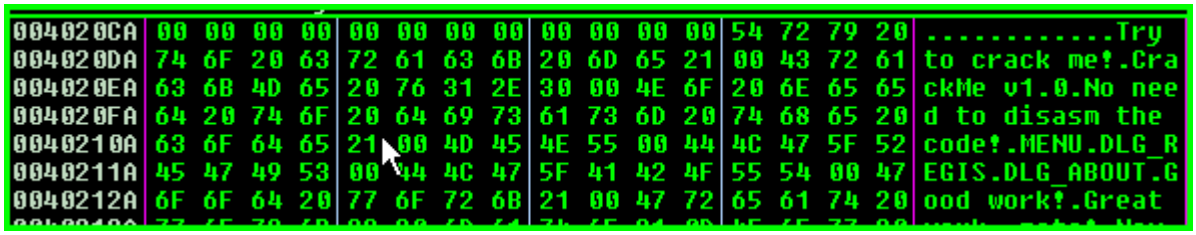
Chắc các bạn cũng đoán ra được ý đồ vì sao tôi lại đặt HWBP tại **MessageBoxA** rồi chứ. Đơn giản, tôi muốn các bạn tự kiểm chứng xem khi ta cho thực thi chương trình, sau đó nhập

UserName và Serial vào và nhấn OK thì Olly sẽ dừng lại tại MessageBoxA, tương tự như những gì ta đã thực hiện ở các phần trước. Các bạn tự thực hành nhé ☺.

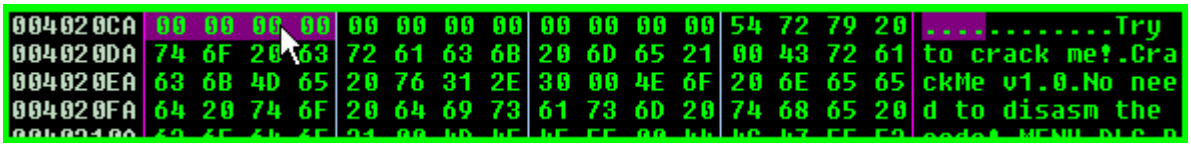
Tiếp theo ta tìm hiểu tiếp về **HWBP on Write** và **HWBP on Access**. Hai kiểu HWBP này cho phép thiết lập với Byte, Word, Dword (tương ứng 1, 2 hoặc 4 bytes). Không khó để thực hiện hai kiểu hwbp này, đơn giản bạn chỉ việc qua cửa sổ Dump, sau đó đánh dấu vùng bytes tương ứng và đặt BP. Ta thực hành luôn nhé, giờ bạn xóa hết các HWBP đã đặt đi, sau đó chúng ta sẽ thử đặt HWBP on Access tại địa chỉ `4020CA` tại cửa sổ Dump. Ta chuyển qua cửa sổ Dump, nhấn **Ctrl+G** và nhập vào :



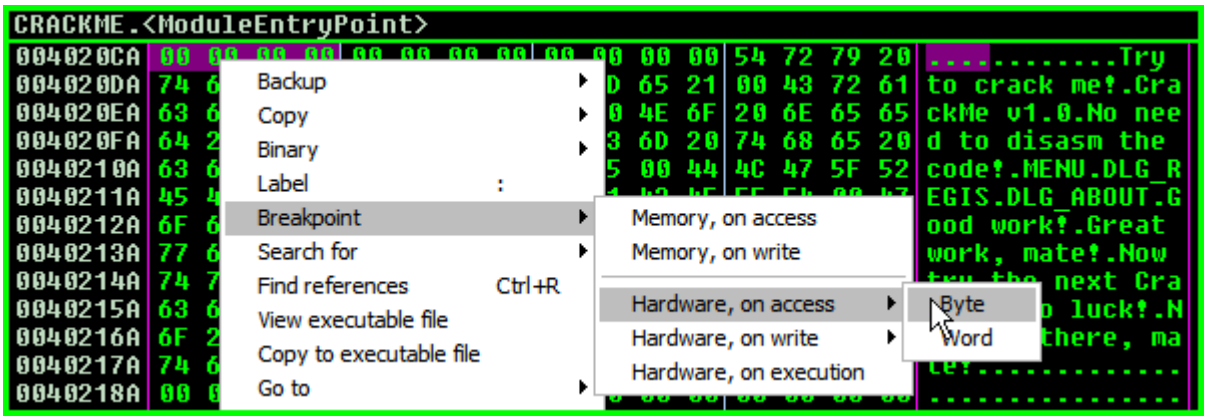
Ta tới đây :



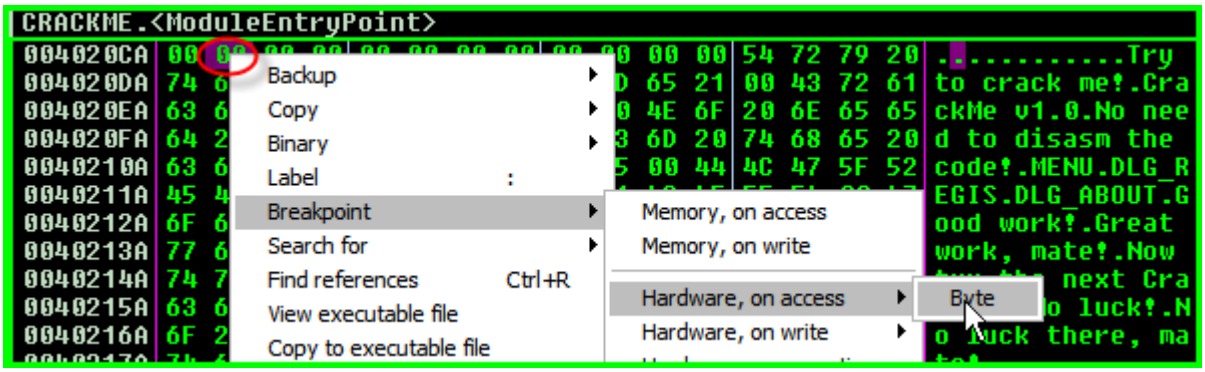
Giờ ta đánh dấu 4 bytes như hình dưới đây :



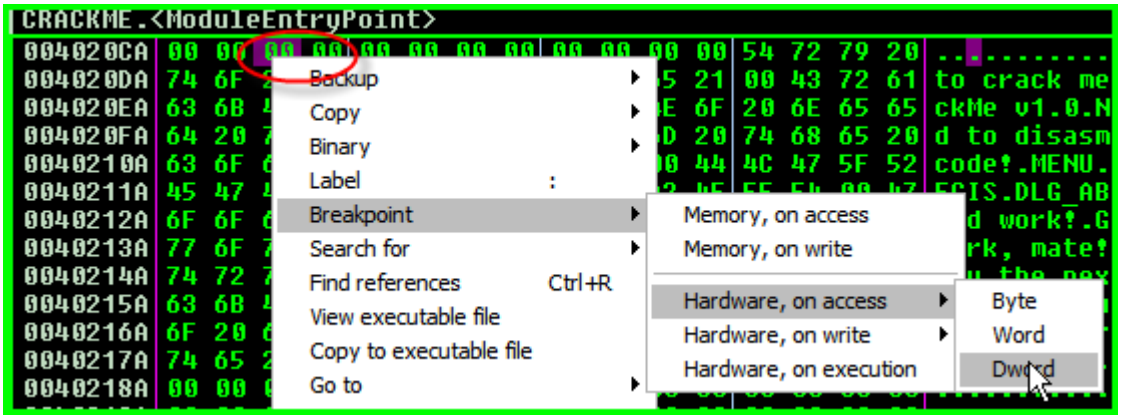
Nhấn chuột phải và chọn như sau :



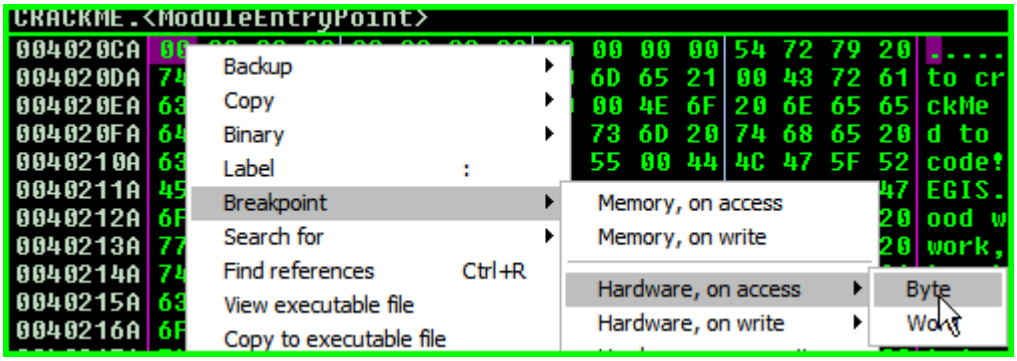
Như các bạn nhìn thấy trên hình, vùng mà chúng ta đánh dấu cho phép chúng ta thiết lập HWBP tại 1 byte hoặc 2 bytes (word). Mặc dù chúng ta đánh dấu chọn 4 bytes nhưng ở đây tùy chọn Dword không xuất hiện trong danh sách của HWBP on Access. Thực hiện tương tự với byte tiếp theo và lần này ta chỉ thấy có tùy chọn Byte xuất hiện :



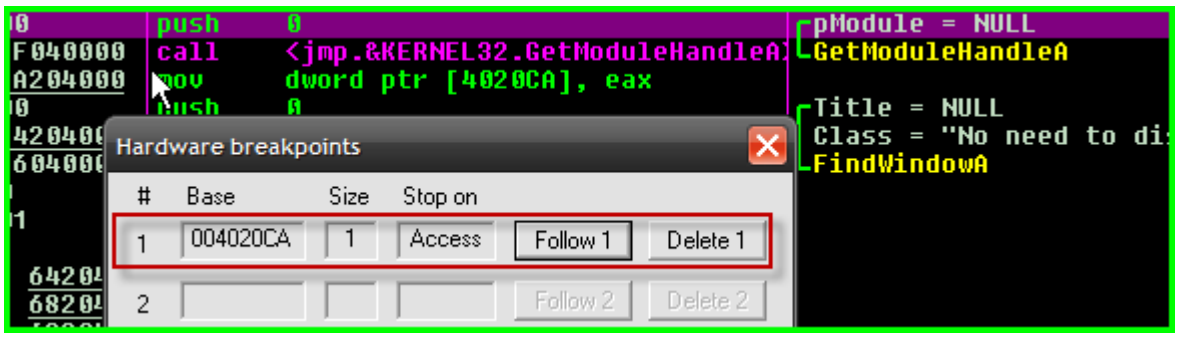
Thử tiếp tục với byte kế tiếp, lần này ta thấy tùy chọn Dword đã xuất hiện :



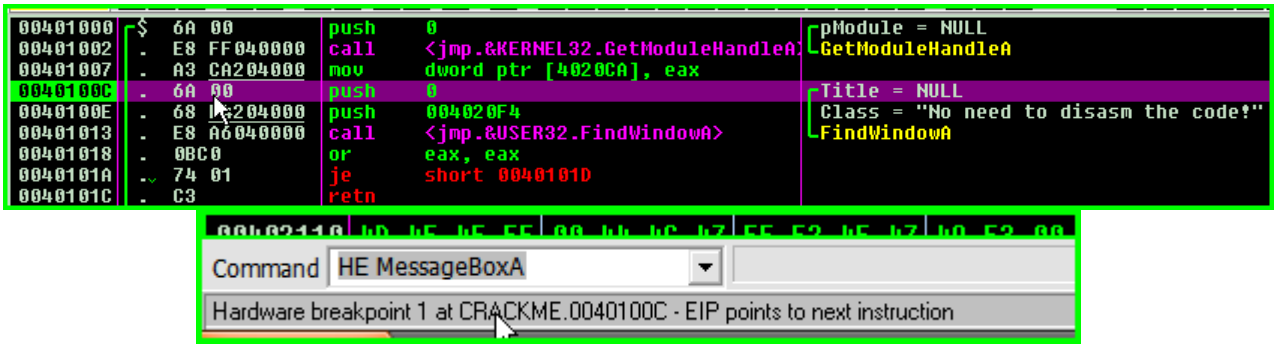
Quay trở lại vấn đề chính, chúng ta đang muốn giám sát vùng nhớ tại `4020CA` khi có bất kì một sự thực thi, đọc hay ghi lên vùng nhớ mà ta đã đặt bp. Để làm điều này ta đặt HWBP như sau :



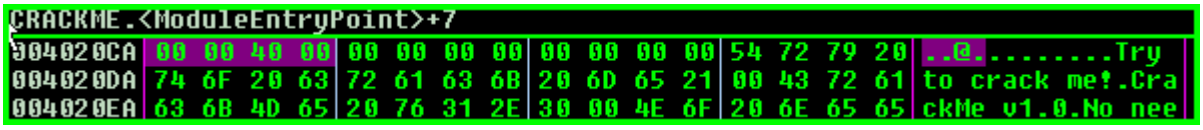
Trong cửa sổ quản lý HWBP ta thấy xuất hiện như sau (size bằng 1 tương ứng với 1 byte) :



Ok, đặt xong hwbp rồi giờ ta run thử chương trình xem thế nào :



Oh, chúng ta đã dừng lại khi có sự tác động vào vùng nhớ 4020CA. Quan sát Olly các bạn thấy dòng *Hardware breakpoint 1 at…*, số **"1"** ở đây tương ứng với số thứ tự của Hwbp được liệt kê trong cửa sổ quản lý Hwbp như các bạn thấy ở hình trên. Ngoài ra các bạn để ý là vệt sáng đang dừng lại tại `0040100C |. 6A 00 push 0 ; /Title = NULL`, chứ không dừng lại tại lệnh tác động vào vùng nhớ [4020CA]: `00401007 |.A3 CA204000 mov dword ptr [4020CA], eax`. Đây là điểm khác biệt giữa Hwbp và Memory Bp, giờ ta quan sát xem nội dung của 4020CA đã thay đổi như thế nào :



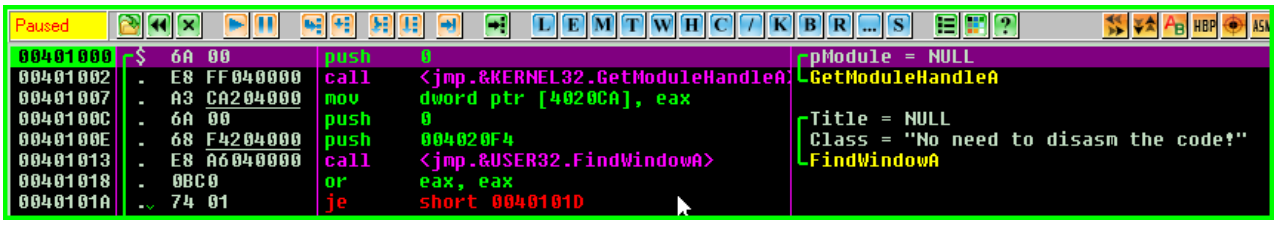
Tương tự, các bạn hãy tự thực hành với cách đặt Hardware Breakpoint on Write. Phần tiếp ta tìm hiểu về **Conditional Breakpoint.**

2.Conditional Breakpoints :

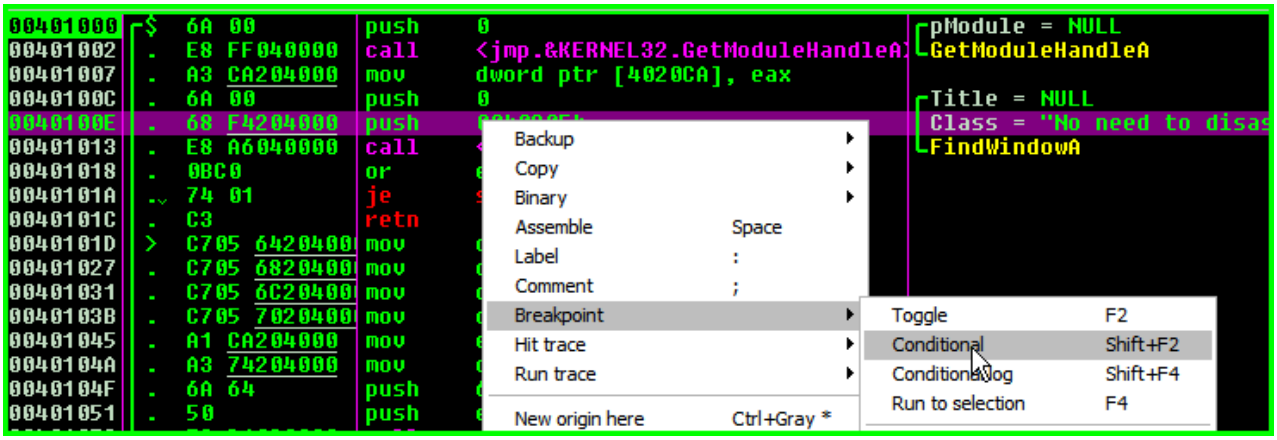
Conditional Breakpoint là gì? Thử tra help của Olly xem có thông tin gì không, tôi có được như sau :

Conditional breakpoint (shortcut Shift+F2) *is an ordinary INT3 breakpoint with associated* ***expression****. Each time Debugger encounters this breakpoint, it estimates expression and, if result is non-zero or expression is invalid, stops debugged program. Of course, the overhead caused by false conditional breakpoint is very high (mostly due to latencies of the operational system). On PII/450 under Windows NT OllyDbg processes up to 2500 false conditional breakpoints per second. An important case of conditional breakpoint is break on Windows message (like WM_PAINT). For this purpose you can use pseudovariable MSG together with proper interpretation of arguments. If window is active, consider message breakpoint described below.*

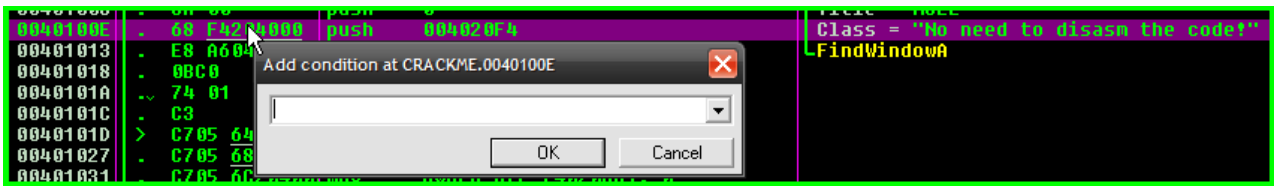
Về mặt khách quan thì nó cũng giống như bp thông thường, tức là nó cũng dùng lệnh Int3 để dừng sự thực thi của chương trình, thêm nữa Condtional bp cũng được quản lý thông qua cửa sổ Breakpoints(**B**). Tuy nhiên, điểm khác biệt nằm ở chỗ điều kiện dừng phải thỏa một điều kiện đã được thiết lập từ trước (Các biểu thức điều kiện như thế nào mời các bạn đọc thêm Help file, phần expression được tôi đánh dấu đỏ ở trên) . Điểm quan trọng khác của conditional bp là nó hay được sử dụng để "tóm" các Windows Message (Ví dụ như: WM_CLOSE, WM_PAINT v..v..). Window Message là gì, chắc tôi không cần giải thích vì nếu các bạn học về lập trình hay muốn nghiên cứu RE thì các bạn phải biết về nó ☺. Bây giờ chúng ta xem xét một ví dụ nhỏ để dễ hiểu và dễ hình dung hơn, chúng ta đang ở tại **Entry point** của Crackme :



Xóa hết các Hwbp đã đặt trước đó đi. Tiếp theo ta sẽ đặt một Conditional BP tại `0x0040100E`, ta làm như sau :



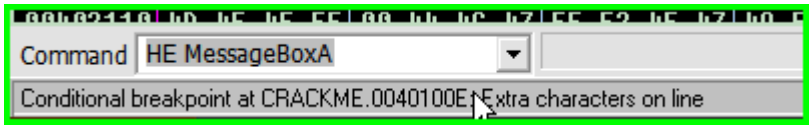
Hoặc đơn giản hơn ta chọn dòng cần đặt Conditional bp và nhấn phím tắt **Shift+F2**, một cửa sổ xuất hiện cho phép ta đặt điều kiện :



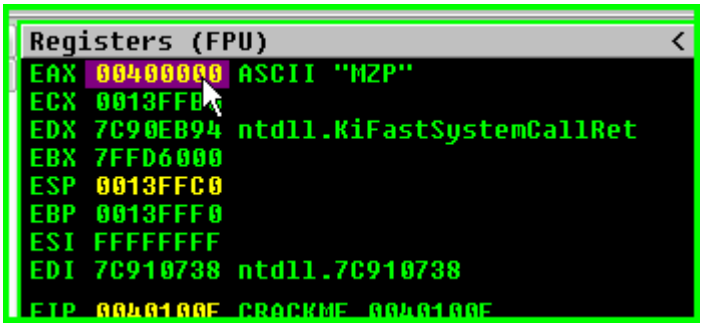
Giả sử, tôi muốn dừng sự thực thi của chương trình khi giá trị của thanh ghi `eax=0x400000`. Tôi nhập điều kiện như sau :

Sau khi nhập điều kiện xong, ngay lập tức ta thấy địa chỉ `0x0040100E` được tô sáng thành màu hồng :

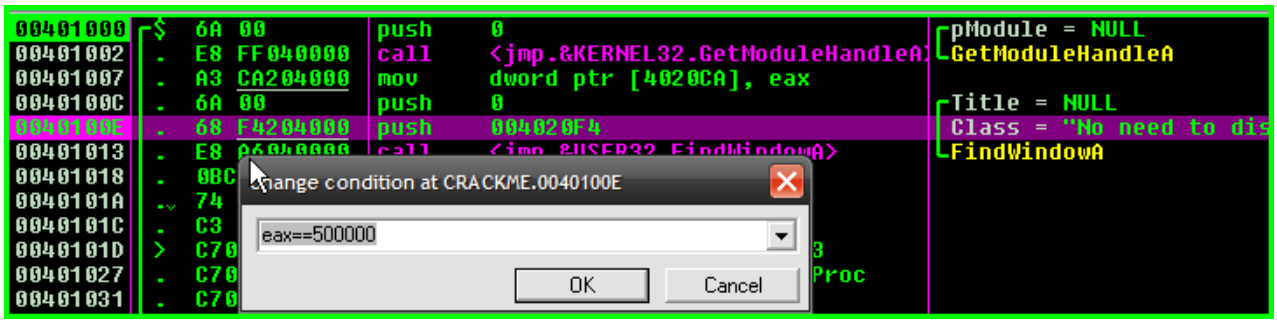
Sau khi đặt xong bp, nhấn **F9** để thực thi chương trình và quan sát :



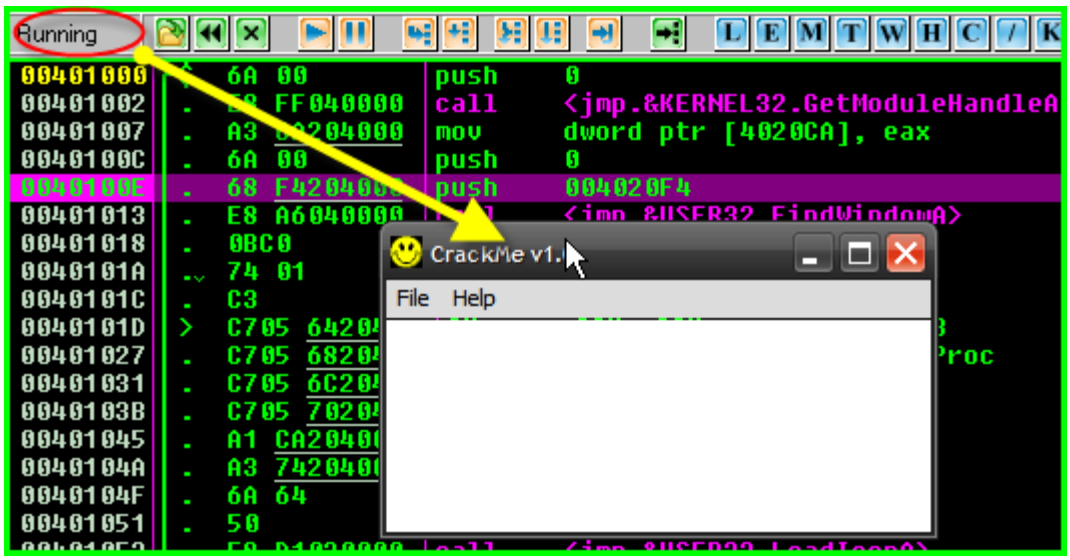
Nhìn sang cửa sổ Registers, ta xem giá trị của EAX lúc này đang là bao nhiêu :



Vậy là ta thấy giá trị của EAX đang là `0x400000`, giá trị này thỏa với điều kiện mà ta đã thiết lập để đặt BP, do đó chương trình dừng lại và quyền điều khiển được trả về cho trình debug. Ta thử đặt lại Conditional bp nhưng với điều kiện `Eax==0x500000` xem thế nào :



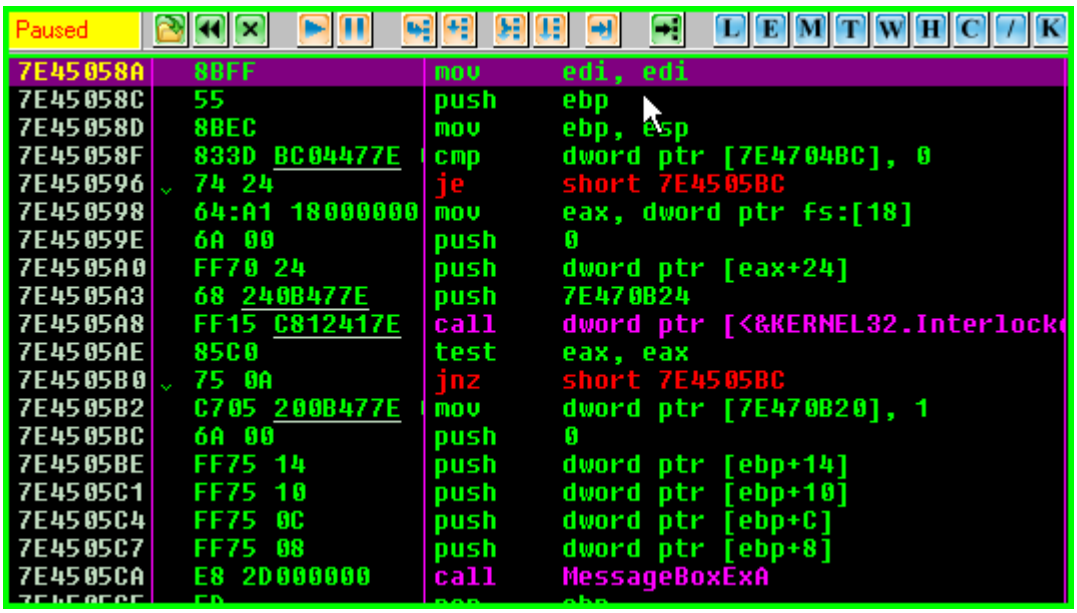
Nhấn **F9** để run chương trình và quan sát, khà khà lần này chẳng thấy nó Olly break gì cả mà lại thấy chương trình chạy vù vù. Điều này chứng tỏ điều kiện ta đặt ra không thỏa ☺.



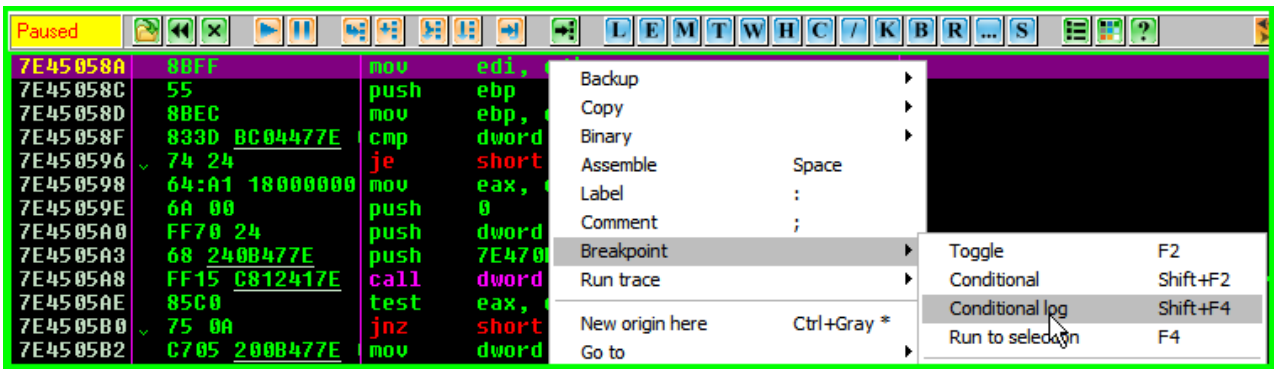
3.Conditional Log Breakpoints :

Conditional logging breakpoint (Shift+F4) *is a conditional breakpoint with the option to log the value of some expression or arguments of known function each time the breakpoint is encountered or when condition is met. For example, you can set logging breakpoint to some window procedure and list all calls to this procedure, or only identifiers of received WM_COMMAND messages, or set it to the call to CreateFile and log names of the files opened for read-only access etc. Logging breakpoints are as fast as conditional, and of course it's much easier to look through several hundred messages in the log window than to press F9 several hundred times. You can select one of several predefined interpretations to your expression.*

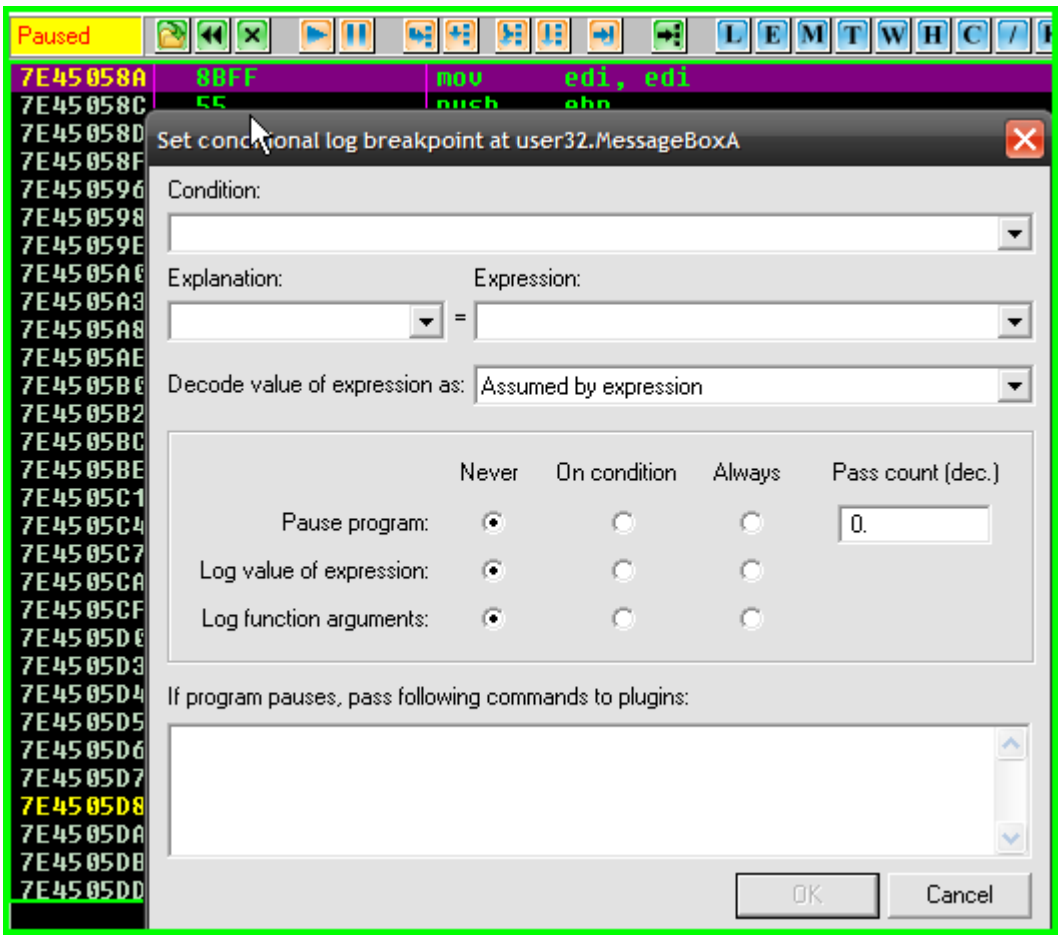
Bp này cũng là một dạng conditional bp tuy nhiên cao cấp hơn một chút. Nó có thêm tùy chọn cho phép ta lưu "vết" giá trị của biểu thức hoặc các tham số của function mỗi khi xảy ra bp hoặc khi thỏa mãn điều kiện mà ta đặt ra. Những thông tin này được lưu lại trong cửa sổ Log(**L**) của Olly. Thực hiện một ví dụ để minh họa, trước tiên chúng ta restart lại Olly cái đã. Sau đó nhấn **Ctrl+G** và tìm tới hàm **MessageBoxA** :



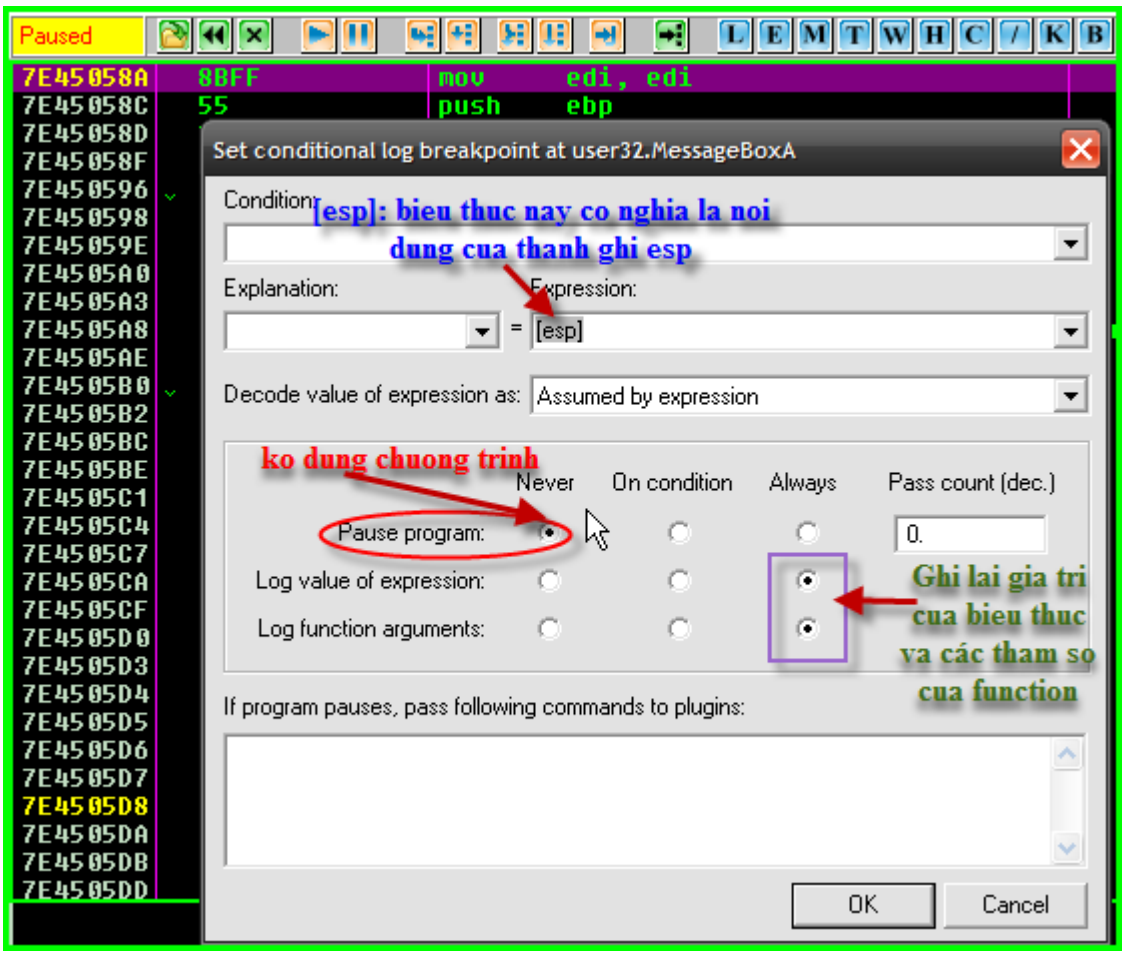
Sau khi tới vị trí tương tự như hình trên, ta tiến hành đặt một Conditional Log BP như sau :



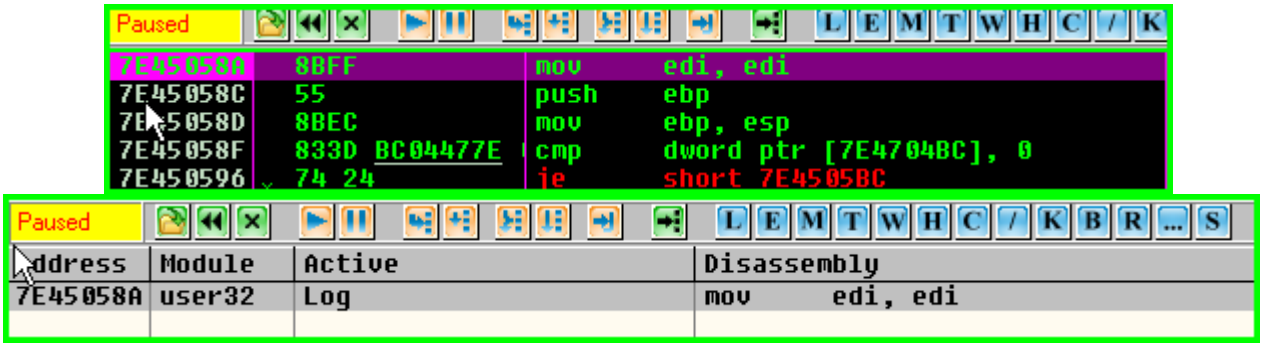
Hoặc đơn giản chỉ việc nhấn phím tắt **Shift+F4**, cửa sổ Conditional Log BP xuất hiện với rất nhiều tùy chọn :



Trong trường hợp này của tôi, tôi không muốn dừng sự thực thi tại hàm **MessageBoxA** mà đơn giản tôi chỉ muốn ghi lại các giá trị mà tôi cần quan tâm, tôi đặt diều kiện như sau :

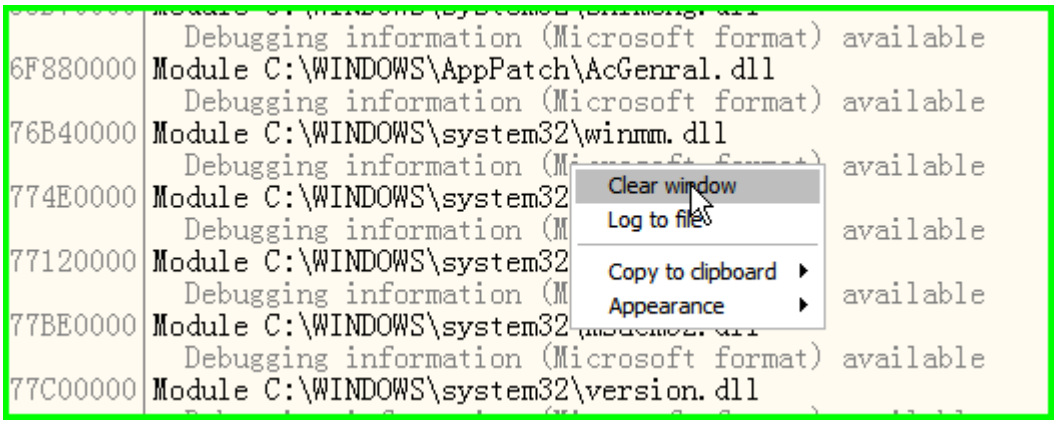


Những gì tôi giải thích trên hình minh họa chắc các bạn cũng hiểu phần nào mục đích của việc đặt BP. Tôi xin nói chi tiết lại như sau, trong phần Expresstion tôi nhập vào là **[esp]** có nghĩa là tôi muốn đọc ra nội dung của thanh ghi Esp. Phần **Pause program** tôi chọn là **Nerver**, có nghĩa là tôi không muốn dừng sự thực thi của chương trình lại. Hai thành phần tiếp theo là ***Log value of expression*** và ***Log function arguments*** tôi chọn là **Always**, có nghĩa là tôi muốn ghi lại nội dung của biểu thức mà cụ thể ở đây là giá trị của thanh ghi esp, đồng thời tôi cũng muốn ghi lạ các tham số truyền vào cho function mà cụ thể ở đây là hàm MessageBoxA. Giải thích vậy các bạn đã hiểu chưa ☺. Sau khi thiết lập như trên xong, nhấn Ok ta có được như sau :

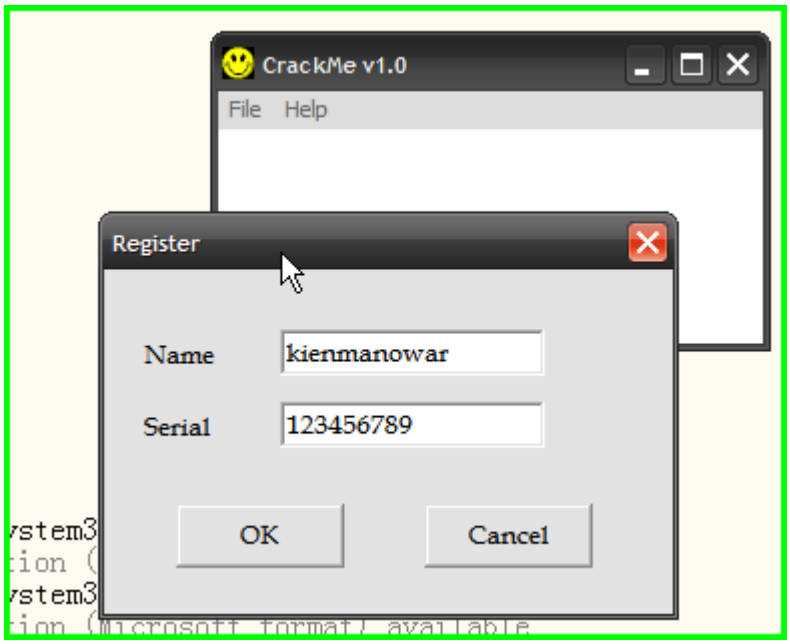


Tiếp theo ta chuyển qua cửa sổ Log :

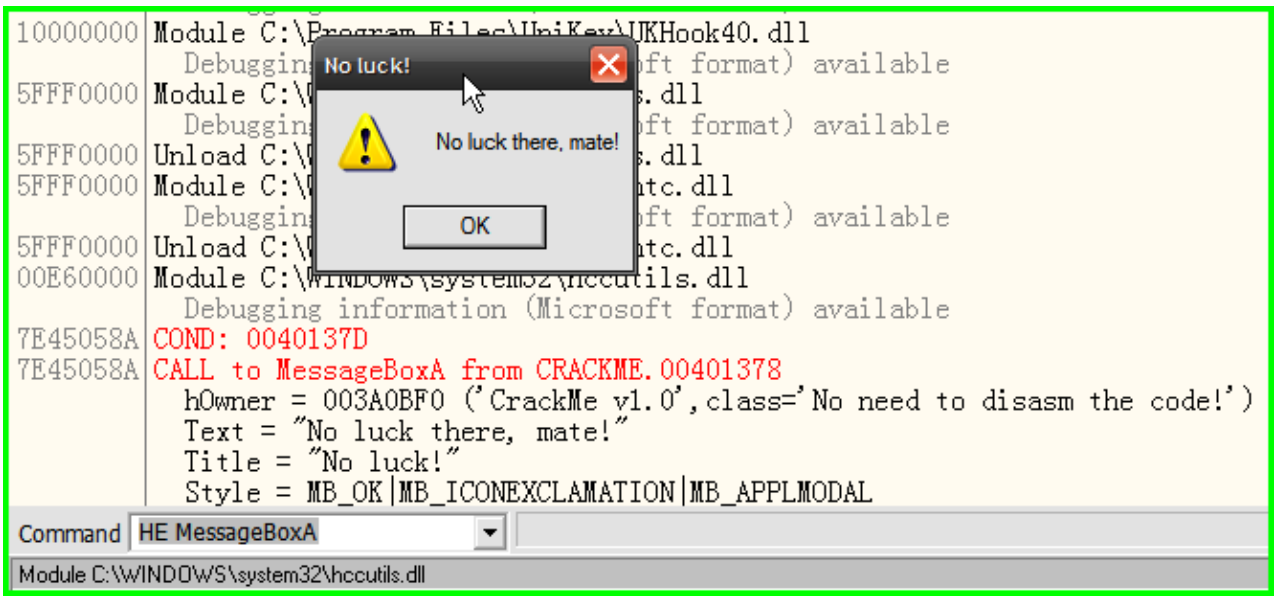
Tại cửa sổ Log ta clear hết các thông tin đi để tiện theo dõi :



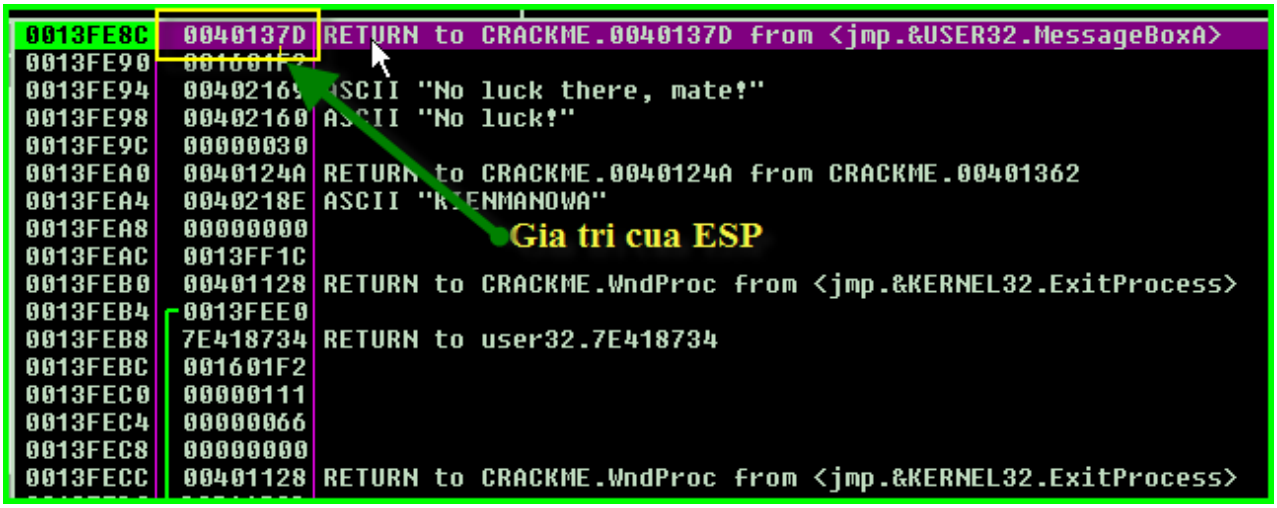
Clear xong ta nhấn **F9** để run chương trình, tiến hành nhập các thông tin mà chương trình yêu cầu :



Sau đó nhấn OK, ta sẽ nhận được Nag thông báo. Lúc này để ý cửa sổ Log ta thấy như sau :

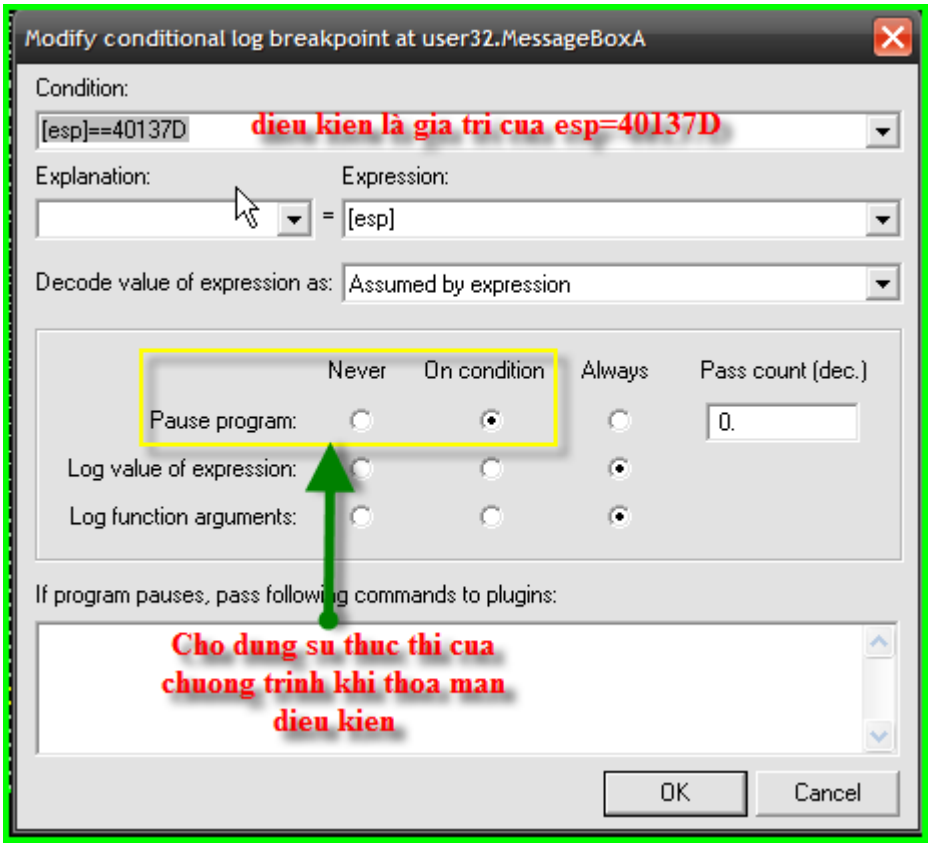


Nhìn vào thông tin trên các bạn thấy dòng màu đỏ đầu tiên COND: 0040137D chắc cũng đoán được đây là giá trị của thanh ghi ESP. Còn các dòng tiếp theo chắc tôi không cần giải thích chắc các bạn cũng đoán ra đó chính là các tham số truyền vào cho hàm MessageBoxA. Để kiểm chứng giá trị của thanh ghi ESP bạn có thể nhìn vào cửa sổ Stack :

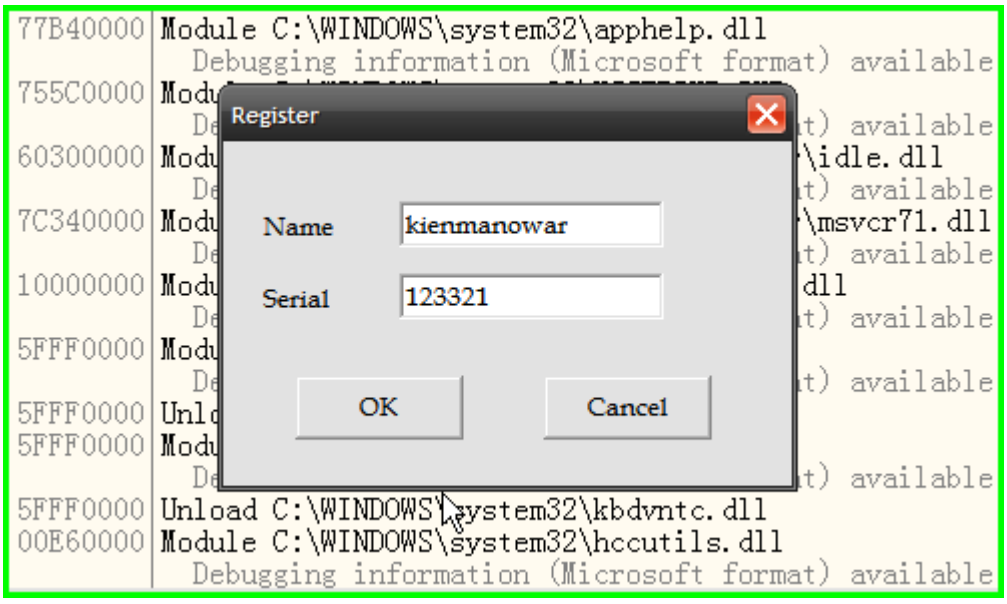


Giá trị `0x0040137D` chính là địa chỉ trở về sau khi thực hiện xong hàm MessageBoxA :

Khà khà mọi thứ đều rõ ràng quá ☺. Tiếp theo chúng ta thử đặt một Conditional Log BP khác xem thế nào, quay trở lại hàm MessageBoxA và nhấn **Shift+F4** để sửa lại như sau :



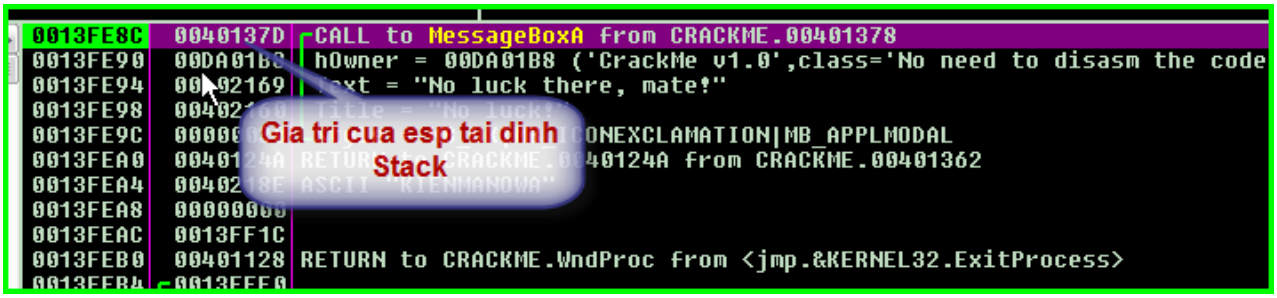
Với Conditional Log bp này tôi thêm điều kiện là `[esp]==0x40137D` và cho dừng sự thực thi của chương trình khi thỏa điều kiện đã đặt ra.Sau khi sửa xong nhấn Ok để chấp nhận những thay đổi.Qua cửa sổ Log và clear hết thông tin cũ. Nhấn **F9** để thực thi chương trình và nhập thông tin theo yêu cầu :



Sau đó nhấn OK, ngay lập tức Olly sẽ break tại chỗ ta đặt BP. Ta chuyển qua cửa sổ Log để quan sát kết quả thu được :

```
7E45058A COND: 0040137D
7E45058A CALL to MessageBoxA from CRACKME.00401378
   hOwner = 00DA01B8 ('CrackMe v1.0',class='No need to disasm the code!')
   Text = "No luck there, mate!"
   Title = "No luck!"
   Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
7E45058A Conditional breakpoint at user32.MessageBoxA
```

Để ý kĩ ta thấy có khác với lần đặt BP trước là sự thực thi của chương trình đã dừng lại và có thêm dòng thông báo : *Conditional breakpoint at user32.MessageBoxA*. Điều đó chứng tỏ là điều kiện đặt bp đã thỏa cho nên chương trình mới dừng lại. Giờ ta kiểm tra cửa sổ Stack xem giá trị của Esp bây giờ có đúng là 0x0040137D hay không ☺.



Ok vậy là phần 11 của loạt tuts về Ollydbg đến đây là kết thúc, qua bài viết này tôi đã giới thiệu cho các bạn biết thêm về hai dạng BP nữa là : **Hardware Breakpoints** và **Conditional Breakpoints** cũng như các cách thiết lập cho hai dạng BP này. Trong phần này tôi vẫn còn nợ các bạn một loại Breakpoint khác đó là **Message Breakpoints**, trong bài viết tiếp theo của loạt tuts này chắc chắn tôi sẽ cùng các bạn khám phá thêm về nó….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, QHQCrker, the_Lighthouse, Merc, Hoadongnoi, Nini ... all REA's members, TQN, HacNho, RongChauA, Deux, tlandn, light.phoenix, dqtln, ARTEAM .... all my friend, and YOU.

--++--==[Thanks To]==--++--
iamidiot, WhyNotBar, trickyboy, dzungltvn, takada, hurt_heart, haule_nth, hytkl, moth, XIANUA, nhc1987, 0xdie, Unregistered!, akira, mranglex v..v.. các bạn đã đóng góp rất nhiều cho REA. Hi vọng các bạn sẽ tiếp tục phát huy ☺

I want to thank **Teddy Roggers** for his great site, Reversing.be folks(especially **haggar**), Arteam folks(**Shub-Nigurrath, MaDMAn_H3rCuL3s**) and all folks on crackmes.de, thank to all members of **unpack.cn** (especially **fly** and **linhanshi**). Great thanks to **lena151**(I like your tutorials).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