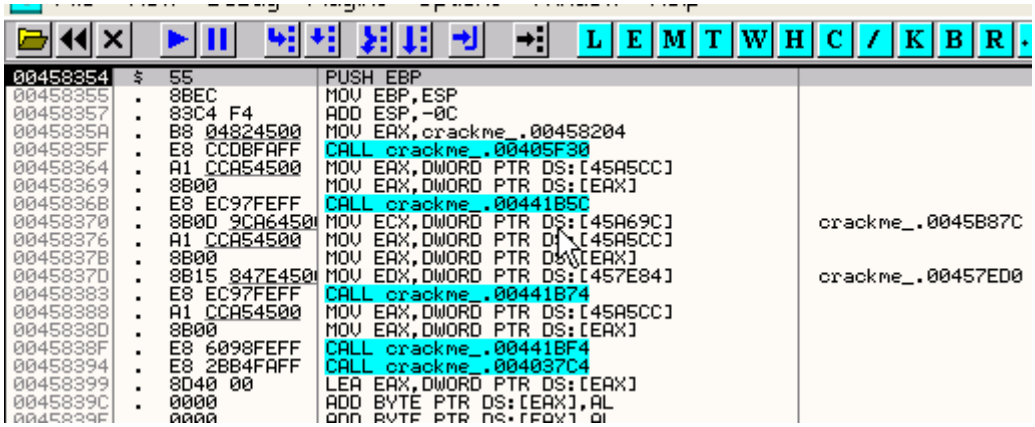
INTRODUCCIÓN AL CRACKING CON OLLYDBG PARTE 18

Bueno, en la parte anterior había comentado que iba aquí a empezar otro tema que seria el tema de la detección del OLLYDBG, por los programas que ejecutamos en el, lo que comúnmente se llama antidebugging, como evitarlo manualmente, y con los plugins disponibles en cada caso y con ejemplos.

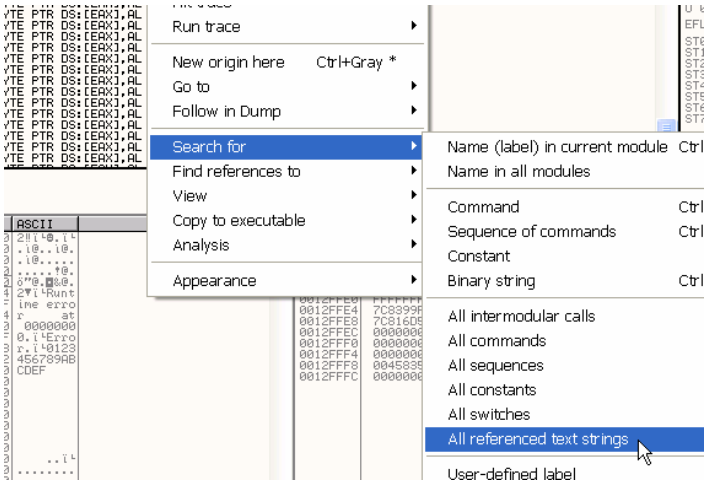
Pero eso con ese tema comenzare en la parte 19, ya que surgió entre varios seguidores del curso que me escribieron al mail, el pedido de que haga un ejemplo mas de crackme sin botones, como el de la parte anterior, y si es posible usando la otra técnica la de WM_KEYUP .

Bueno así que usaremos un crackme que me facilito Stzwei, gracias por el mismo, llamado crackme_4stz.exe y que por supuesto estará adjunto con este tutorial.

Abramos el crackme en OLLYDBG



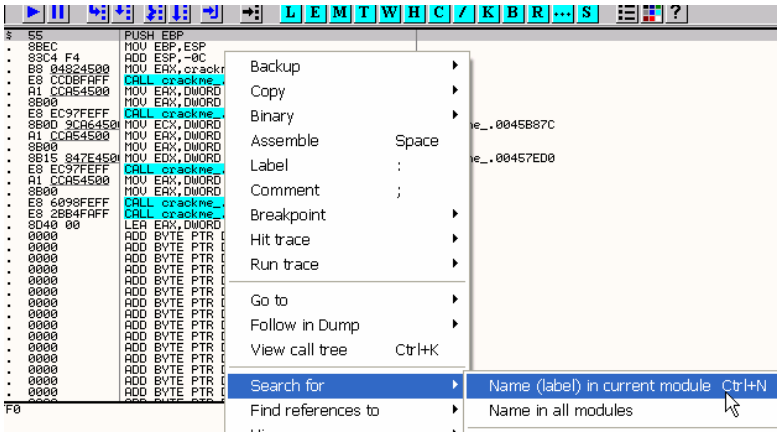
Allí esta parado en el Entry Point, veamos las strings referentes ya que no esta empacado.

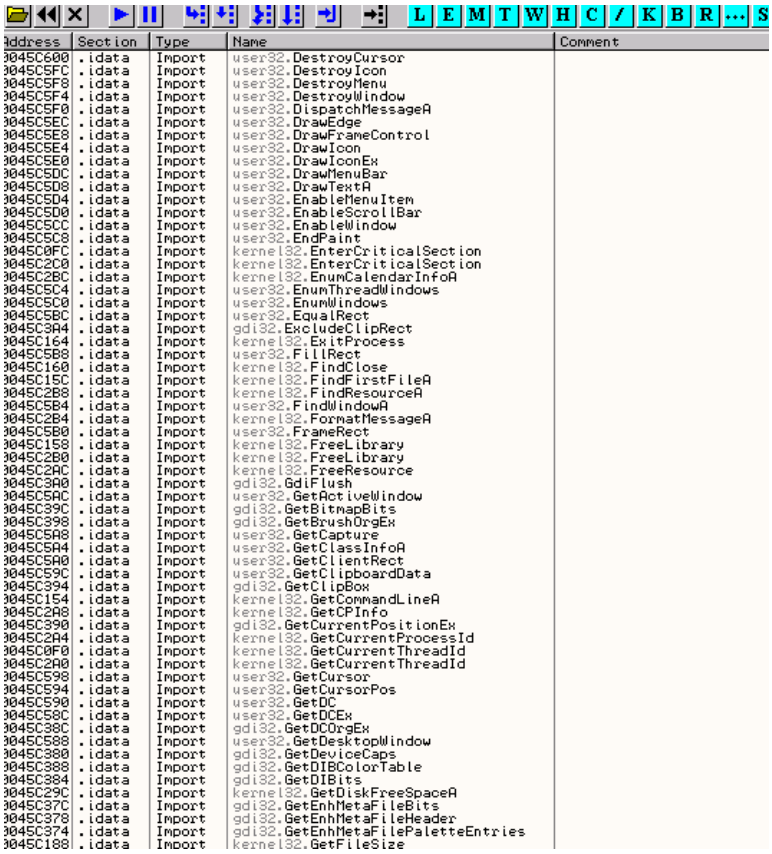


Allí vemos las Strings que usa el programa

```
Address  Disassembly                              Text string
00401006 ASCII "Boolean"
0040101B ASCII "False"
00401021 ASCII "True"
0040102E ASCII "Char"
00401042 ASCII "Integer"
0040105A ASCII "Byte"
0040106E ASCII "Word"
00401082 ASCII "String"
00401088 DD crackme_.004010D4                     ASCII 07,"TObject"
004010A8 DD crackme_.004010D4                     ASCII 07,"TObject"
004010D5 ASCII "TObject"
004010E2 ASCII "TObject"
004010E9 DD crackme_.004010D4                     ASCII 07,"TObject"
004010F4 ASCII "System"
00401102 ASCII "IUnknown"
00401120 ASCII "System"
00401178 DD crackme_.004011C4                     ASCII 11,"TInterfacedObject"
00401198 DD crackme_.004011C4                     ASCII 11,"TInterfacedObject"
004011C5 ASCII "TInterfacedObjec"
004011D5 ASCII "t"
00402B04 PUSH crackme_.00402B84                   ASCII "SOFTWARE\Borland\Delphi\RTL"
00402B38 PUSH crackme_.00402BA0                   ASCII "FPUMaskValue"
00402B84 ASCII "SOFTWARE\Borland"
00402B94 ASCII "\Delphi\RTL",0
00402BA0 ASCII "FPUMaskValue",0
00403747 MOV EBX,crackme_.0045903C                ASCII "  at 00000000"
0040380A MOV EDX,crackme_.0045902C                ASCII "Runtime error     at 00000000"
0040382B PUSH crackme_.0045904C                   ASCII "Error"
00403830 PUSH crackme_.0045902C                   ASCII "Runtime error     at 00000000"
004038B7 ASCII "Portions Copyrig"
004038C7 ASCII "ht (c) 1983,99 B"
004038D7 ASCII "orland",0
00404C99 PUSH crackme_.00404E08                   ASCII "kernel32.dll"
00404CA9 PUSH crackme_.00404E18                   ASCII "GetLongPathNameA"
00404E08 ASCII "kernel32.dll",0
00404E18 ASCII "GetLongPathNameA"
00404E28 ASCII 0
00404E5C PUSH crackme_.00404FE8                   ASCII "Software\Borland\Locales"
00404E7A PUSH crackme_.00405004                   ASCII "Software\Borland\Delphi\Locales"
00404FE8 ASCII "Software\Borland"
00404FF8 ASCII "\Locales",0
00405004 ASCII "Software\Borland"
00405014 ASCII "\Delphi\Locales",0
004051A4 CMP DWORD PTR DS:[EBX+4],10000           UNICODE "ALLUSERSPROFILE=C:\Documents and Settings\All Users"
00405661 MOV EDX,crackme_.00459064                ASCII 20,"                    "
0040587C ASCII " @",0
00405886 ASCII "#@",0
```

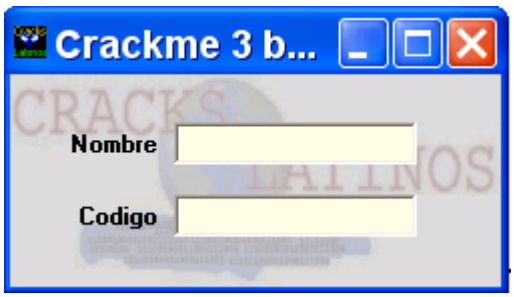
Nada interesante por aquí, veamos las apis



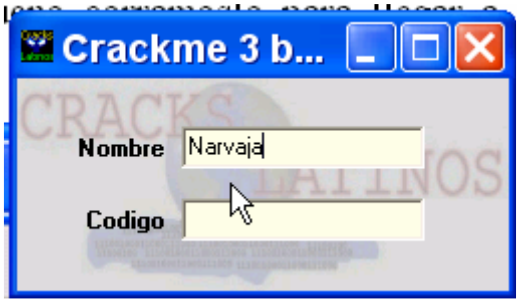


Address	Section	Type	Name	Comment
3045C600	.idata	Import	user32.DestroyCursor	
3045C5FC	.idata	Import	user32.DestroyIcon	
3045C5F8	.idata	Import	user32.DestroyMenu	
3045C5F4	.idata	Import	user32.DestroyWindow	
3045C5F0	.idata	Import	user32.DispatchMessageA	
3045C5EC	.idata	Import	user32.DrawEdge	
3045C5E8	.idata	Import	user32.DrawFrameControl	
3045C5E4	.idata	Import	user32.DrawIcon	
3045C5E0	.idata	Import	user32.DrawIconEx	
3045C5DC	.idata	Import	user32.DrawMenuBar	
3045C5D8	.idata	Import	user32.DrawTextA	
3045C5D4	.idata	Import	user32.EnableMenuItem	
3045C5D0	.idata	Import	user32.EnableScrollBar	
3045C5CC	.idata	Import	user32.EnableWindow	
3045C5C8	.idata	Import	user32.EndPaint	
3045C0FC	.idata	Import	kernel32.EnterCriticalSection	
3045C2C0	.idata	Import	kernel32.EnterCriticalSection	
3045C2BC	.idata	Import	kernel32.EnumCalendarInfoA	
3045C5C4	.idata	Import	user32.EnumThreadWindows	
3045C5C0	.idata	Import	user32.EnumWindows	
3045C5BC	.idata	Import	user32.EqualRect	
3045C3A4	.idata	Import	gdi32.ExcludeClipRect	
3045C164	.idata	Import	kernel32.ExitProcess	
3045C5B8	.idata	Import	user32.FillRect	
3045C160	.idata	Import	kernel32.FindClose	
3045C15C	.idata	Import	kernel32.FindFirstFileA	
3045C2B8	.idata	Import	kernel32.FindResourceA	
3045C5B4	.idata	Import	user32.FindWindowA	
3045C2B4	.idata	Import	kernel32.FormatMessageA	
3045C5B0	.idata	Import	user32.FrameRect	
3045C158	.idata	Import	kernel32.FreeLibrary	
3045C2B0	.idata	Import	kernel32.FreeLibrary	
3045C2AC	.idata	Import	kernel32.FreeResource	
3045C3A0	.idata	Import	gdi32.GdiFlush	
3045C5AC	.idata	Import	user32.GetActiveWindow	
3045C39C	.idata	Import	gdi32.GetBitmapBits	
3045C398	.idata	Import	gdi32.GetBrushOrgEx	
3045C5A8	.idata	Import	user32.GetCapture	
3045C5A4	.idata	Import	user32.GetClassInfoA	
3045C5A0	.idata	Import	user32.GetClientRect	
3045C59C	.idata	Import	user32.GetClipboardData	
3045C394	.idata	Import	gdi32.GetClipBox	
3045C154	.idata	Import	kernel32.GetCommandLineA	
3045C2A8	.idata	Import	kernel32.GetCPInfo	
3045C390	.idata	Import	gdi32.GetCurrentPositionEx	
3045C2A4	.idata	Import	kernel32.GetCurrentProcessId	
3045C0F0	.idata	Import	kernel32.GetCurrentThreadId	
3045C2A0	.idata	Import	kernel32.GetCurrentThreadId	
3045C598	.idata	Import	user32.GetCursor	
3045C594	.idata	Import	user32.GetCursorPos	
3045C590	.idata	Import	user32.GetDC	
3045C58C	.idata	Import	user32.GetDCEx	
3045C38C	.idata	Import	gdi32.GetDCOrgEx	
3045C588	.idata	Import	user32.GetDesktopWindow	
3045C380	.idata	Import	gdi32.GetDeviceCaps	
3045C388	.idata	Import	gdi32.GetDIBColorTable	
3045C384	.idata	Import	gdi32.GetDIBits	
3045C29C	.idata	Import	kernel32.GetDiskFreeSpaceA	
3045C37C	.idata	Import	gdi32.GetEnhMetaFileBits	
3045C378	.idata	Import	gdi32.GetEnhMetaFileHeader	
3045C374	.idata	Import	gdi32.GetEnhMetaFilePaletteEntries	
3045C188	.idata	Import	kernel32.GetFileSize	

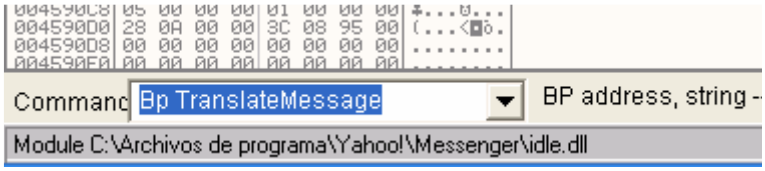
Uff muchísimas para mi gusto, bueno corrámoslo para llegar a la ventana donde se ingresa el serial, apreto F9.



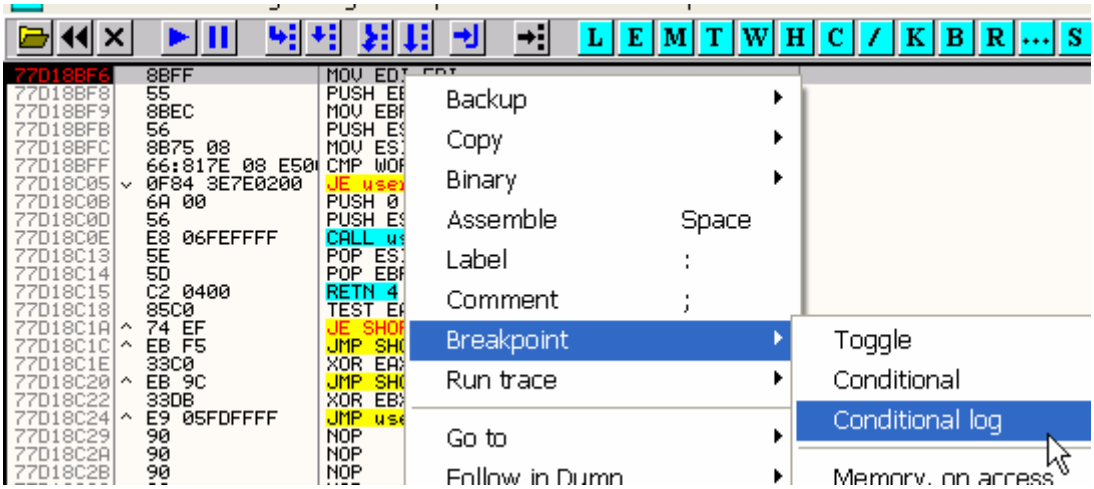
Vemos que no hay botón de registro, así que tipeo un nombre completo



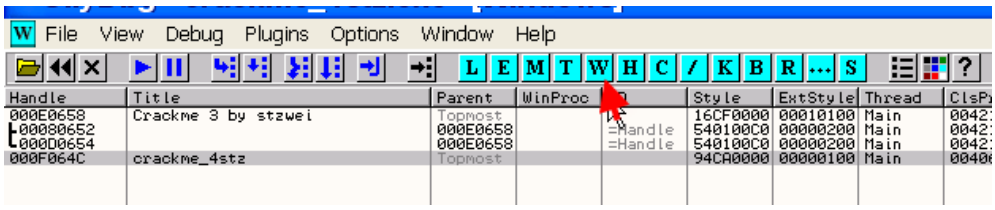
Y pondré un BPX CONDICIONAL LOG en TranslateMessage, primero coloco un BPX común



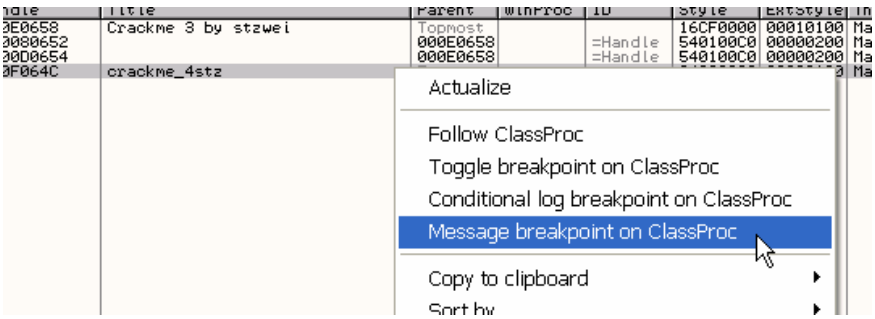
Y cuando voy a acceder al crackme, para solo en el BP allí lo edito.



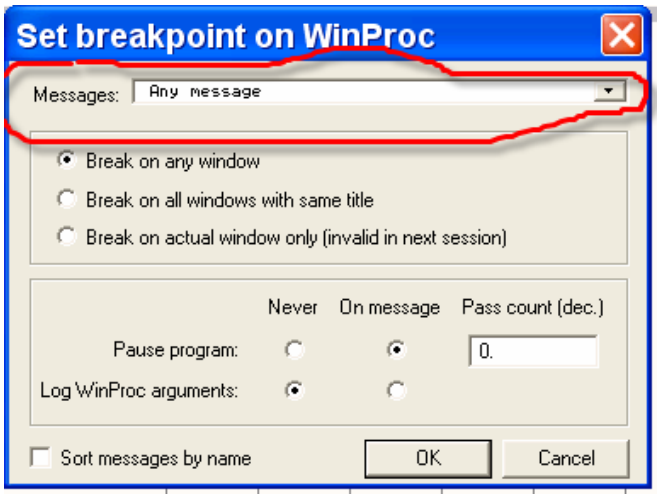
Si alguien no se acuerda el valor correspondiente al WM que vamos a usar, pues vamos a la ventana W, que nos muestra las ventanas del programa.



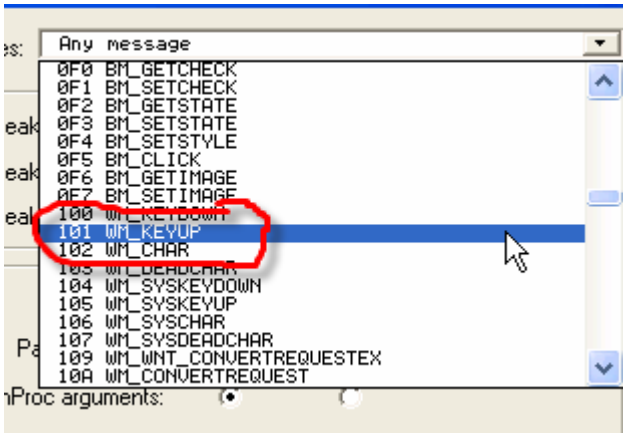
Y nos aparece la lista de ventanas, en cualquiera, total es solo para averiguar el valor numérico, hacemos click derecho



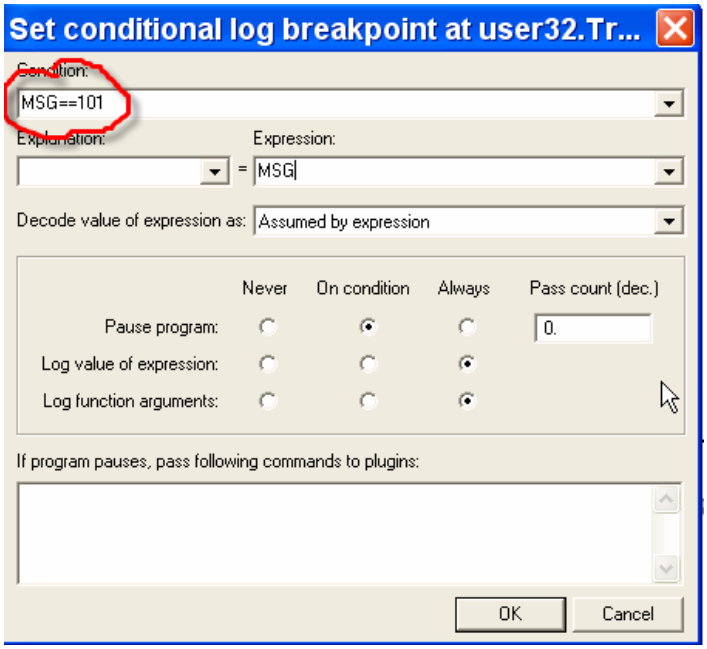
Y nos aparece la ventana de los MESSAGE BREAKPOINTS con el menú desplegable de los WM



Buscamos en la lista WM_KEYUP

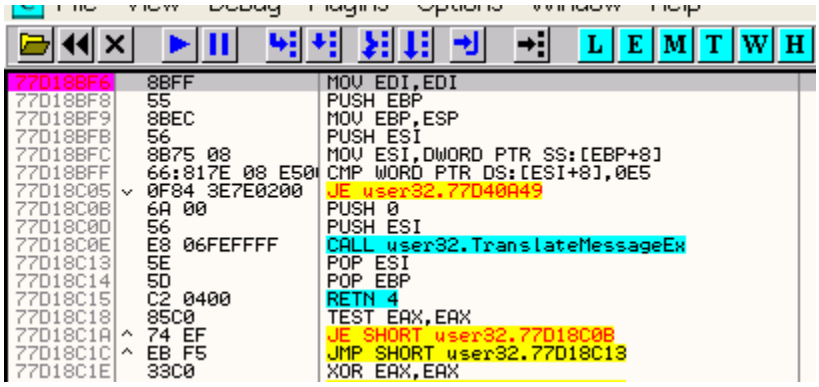


Vemos que es 101, así que cancelamos todo esto que fue solo para averiguar ese valor numérico y volvemos al BREAKPOINT CONDICIONAL LOG en TranslateMessage.

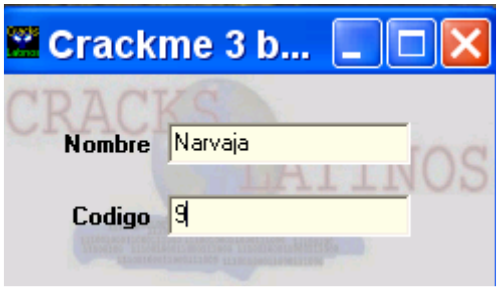


Y allí coloco MSG==101 (recordar el doble signo igual), también podía haber escrito allí si no encuentro el valor, MSG==WM_KEYUP y funcionara, pero a mi me gusta mas usar valores numéricos, cada uno puede hacer como quiera en ese punto.

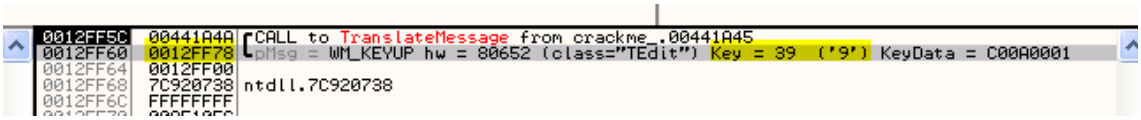
Bueno allí quedo nuestro BPX transformado en CONDICIONAL LOG (color rosa)



Demos RUN y tecleo la primera letra de mi serial falso

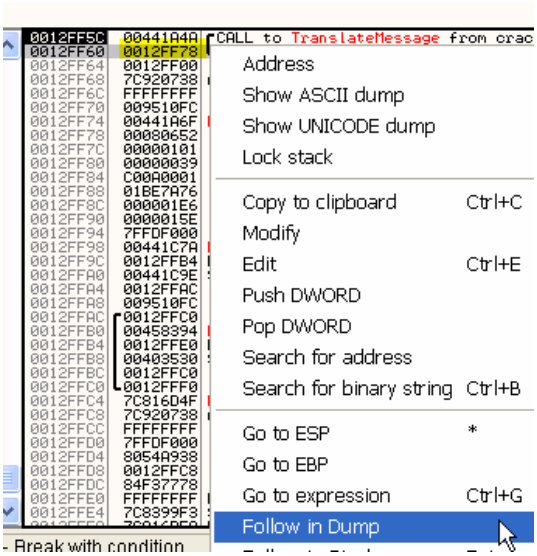


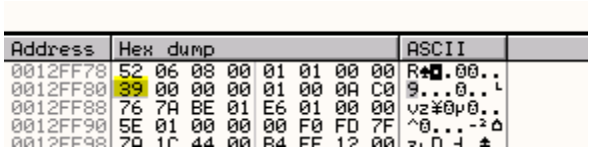
Y allí se detiene el OLLYDBG en el BPX CONDICIONAL.



Vemos allí los parámetros de la api, en 12ff78 hay una estructura que guarda los valores del la tecla que aprete en este caso 39, que corresponde al numero 9.

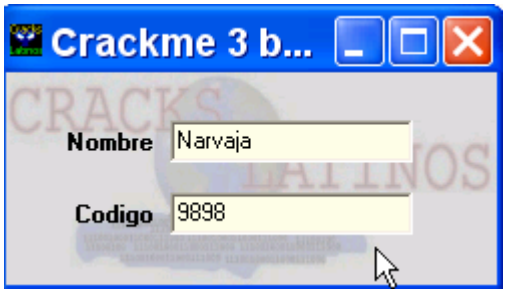
Veamos en el DUMP dicha posición de memoria, haciendo click derecho en dicha dirección y eligiendo FOLLOW IN DUMP.





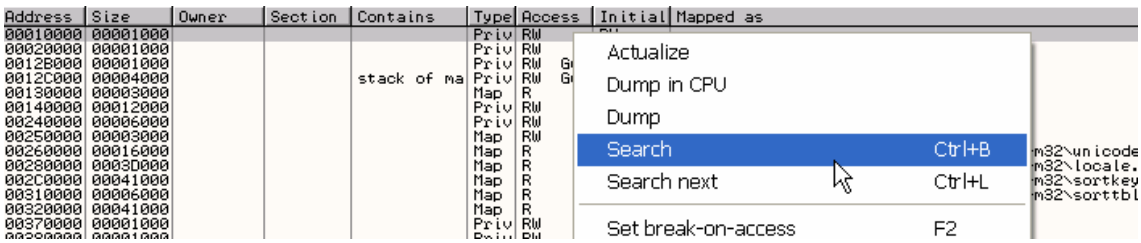
Vemos que conseguimos parar y identificar cuando ingresa el byte el problema es que es un programa en DELPHI lo cual veremos mas adelante y si pongo un BPM ON ACCESS en ese byte, dará mil vueltas antes de llegar a la comparación, por lo cual, el método que vimos la vez anterior se aplica mucho mejor aquí y antes de enloquecer pues lo usaremos, de cualquier forma ya saben como parar el programa cuando ingresa por teclado, y como localizar el byte, seguramente en otro crackme que no sea en DELPHI, se podrá seguir mas fácilmente con un BPM ON ACCESS en el mismo, hasta la comparación.

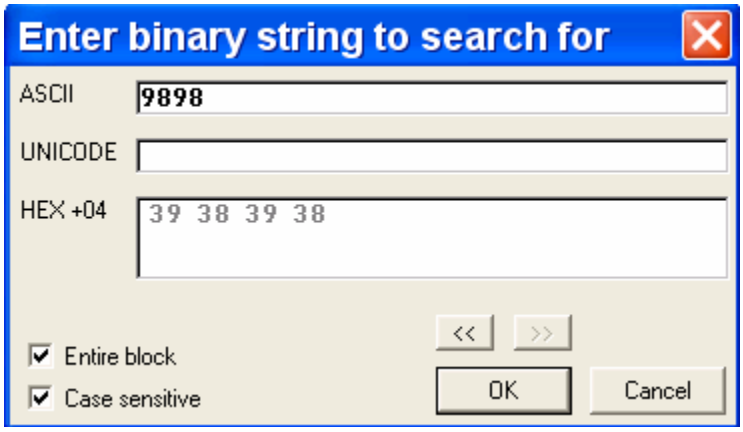
Y doy RUN



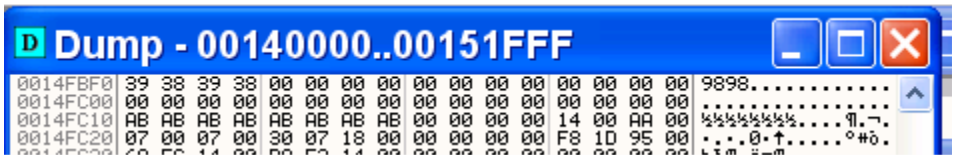
Tipeo 9898, iré a la ventana M y buscare en toda la memoria

Hago click derecho- SEARCH



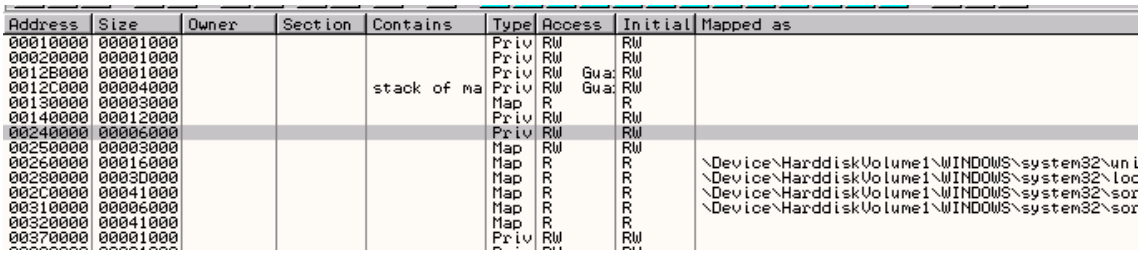


Y busco la string 9898

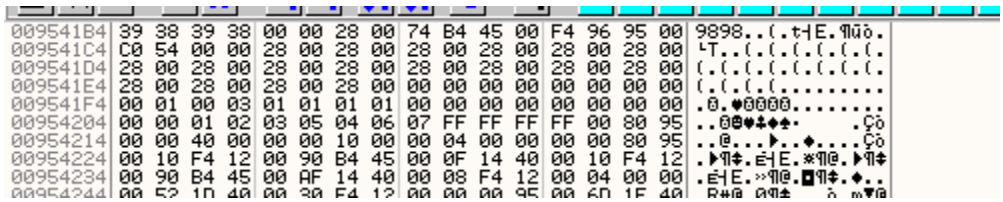


La primera ocurrencia es en el stack apreto CTRL + L varias veces a ver cuantas aparece en el mismo.

Una zona vez ya que nos aparece en amarillo la palabra ITEM NOT FOUND debajo, salgamos de esta sección y volvamos a la ventana M

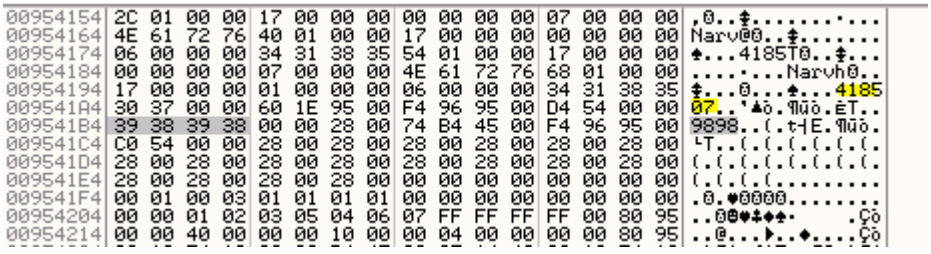


Llego hasta aquí buscando, así que sigamos para abajo con CTRL + L

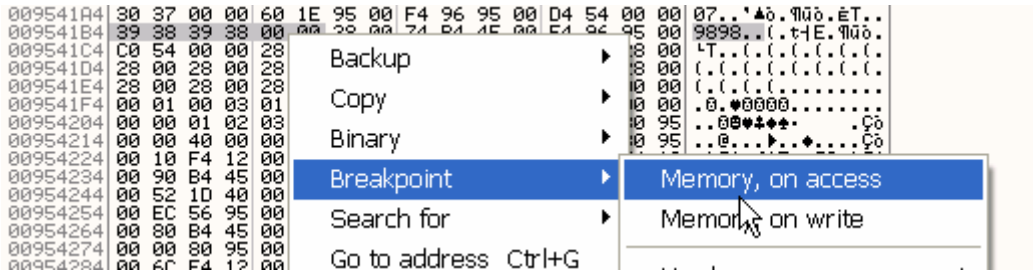


Allí aparece por primera vez en esta sección hagamos CTRL + L para ver si aparece de nuevo

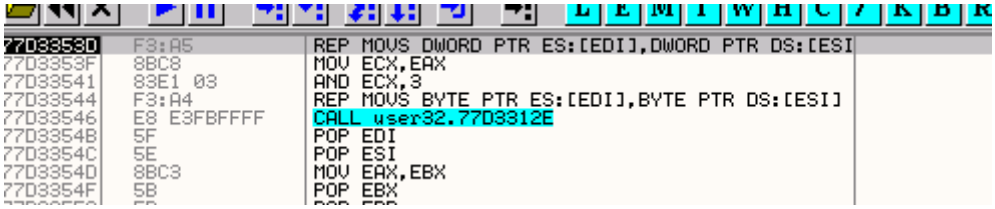
No, no aparece de nuevo si miro en la zona donde esta la ocurrencia



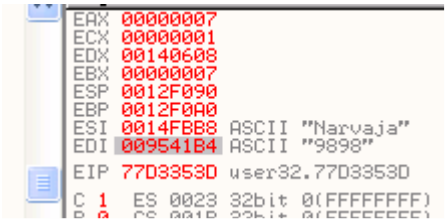
Veo un posible serial 418507, igual le pongo un BPM on ACCESS al lugar de la ocurrencia a ver si es este el que comparara.



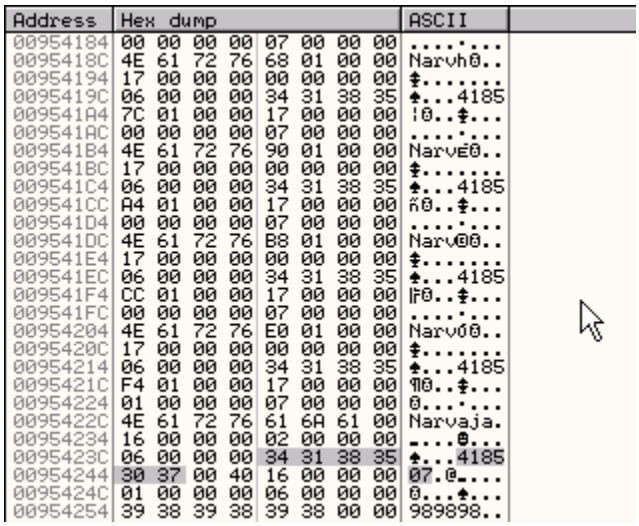
Doy RUN



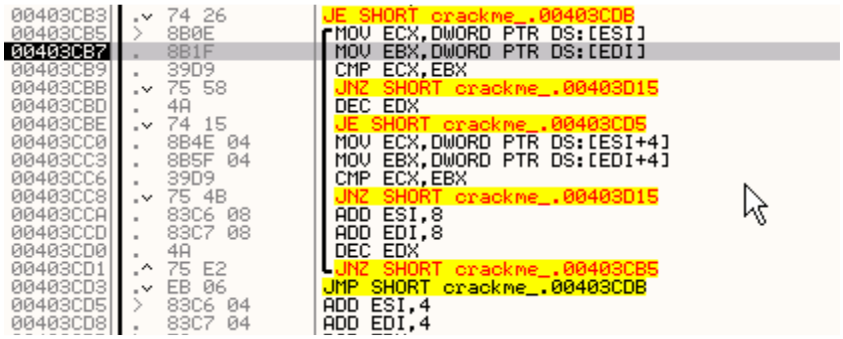
Para alli y vemos que copiara mi nombre



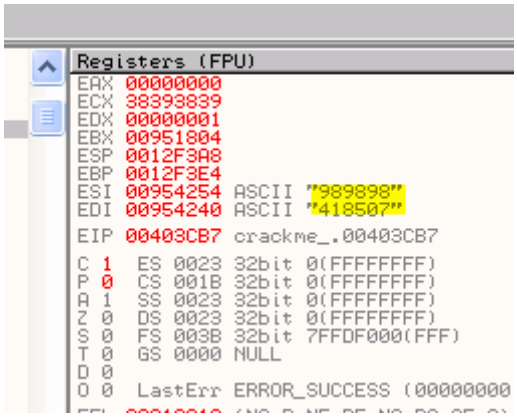
Lo que vemos que cada vez que tipeo va copiando de nuevo mas abajo mi serial falso y cada vez lo va machacando con mi nombre, y con el carácter siguiente vuelve a copiar el serial bueno mas abajo y asi hasta que tipeo los 6 caracteres del serial falso y ahí si cuando están los 6 caracteres a la vista poniendo un BPM en ellos.



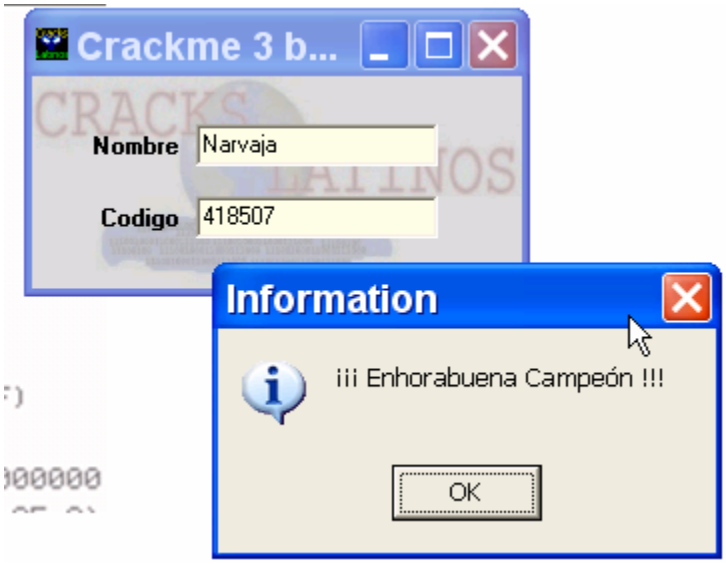
Para justo en la comparación



Allí vemos que ESI y EDI apuntan al serial falso y al verdadero

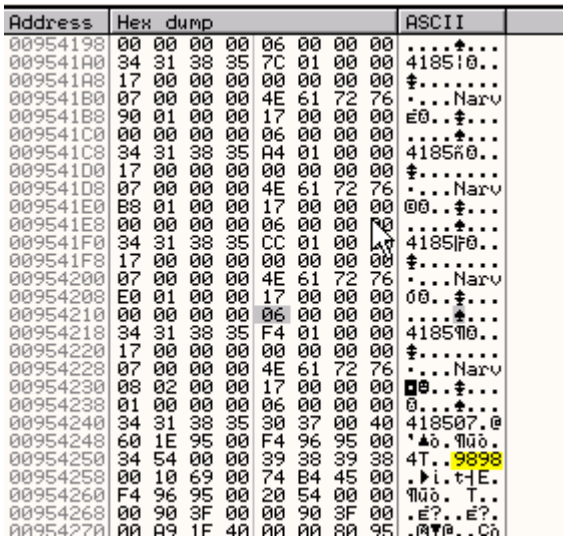


Los cuales son comparados de a 4 bytes en ese loop, probemos si el serial hallado es el correcto

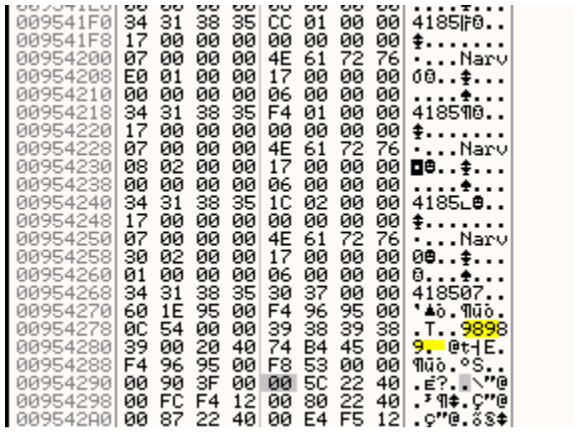


Vemos que a pesar de todo es un crackme difícil ya que es un momento justo, que hay que agarrarlo ya que va variando la posición del serial falso y hay que agarrarlo con el BPM justo cuando recién lo escribe en su nueva posición, para poder parar en la comparación.

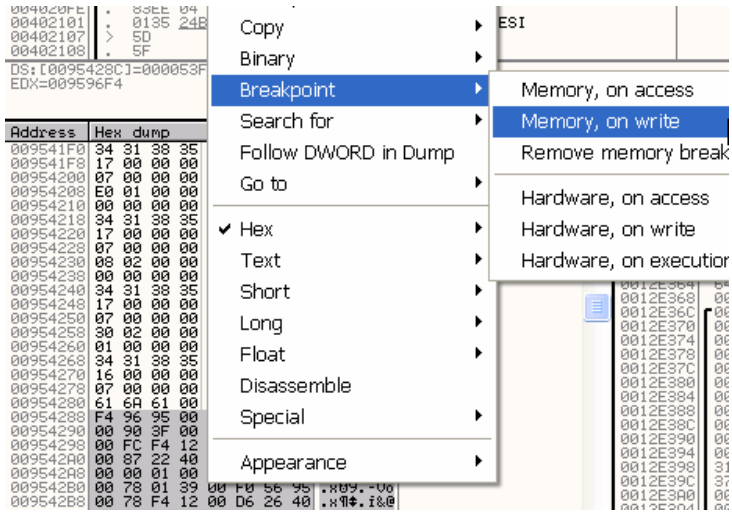
Otra posibilidad es la siguiente veo que con 4 bytes aparece esto



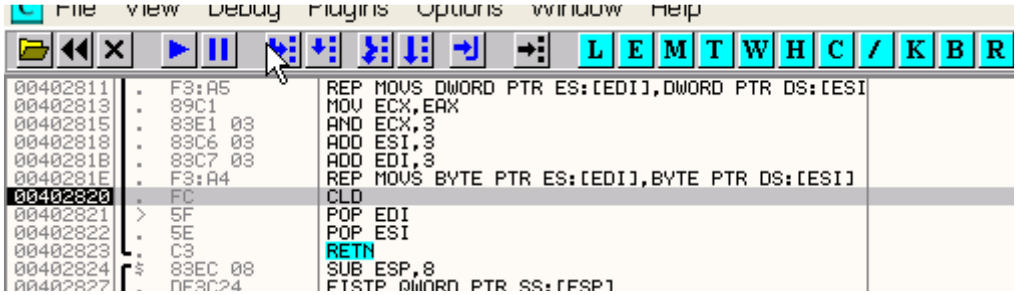
Con 5 Bytes escribe justo abajo



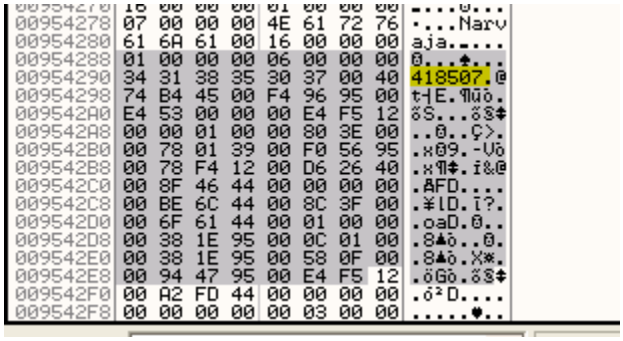
Pues voy poniendo BPM ON WRITE en la zona justo abajo, donde va a escribir cuando ingrese el próximo carácter



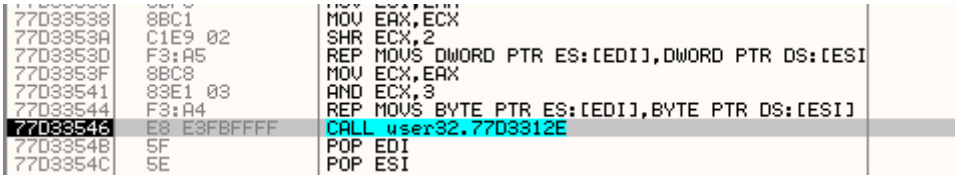
Seguro cuando ingrese el próximo carácter, lo escribirá aquí, así que doy RUN



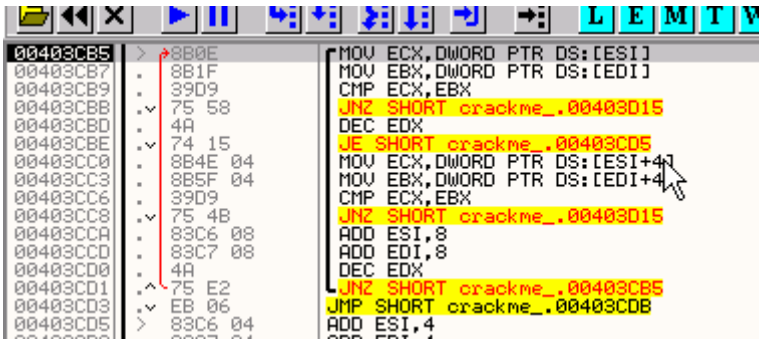
Allí escribio primero el posible serial bueno



Damos RUN



Y luego para cuando copia el serial falso, ahora si que paro apenas se copio, puedo ponerle un BPM on ACCESS al serial falso y al dar RUN para justo en la comparación, nuevamente.



Bueno creo que este fue un caso difícil para cualquier método por lo movedizo de la zona donde guarda el serial.

Bueno les dejare un ejercicio, este será hallar el serial del CRACKME DE CRUEHEAD 2, esto será el fin del primer capitulo de la INTRODUCCIÓN a partir de la parte 19 empezaremos con nuevos temas, y el resolver este sencillo crackme será como la graduación del primer capitulo, recuerden que para abrir el rar de la parte 19 necesitaran del serial verdadero del crackme de cruehead 2 es bien sencillo.

Hasta la parte 19
Ricardo Narvaja
19 de diciembre de 2005