**INTRODUCCION AL CRACKING CON OLLYDBG PARTE 9**

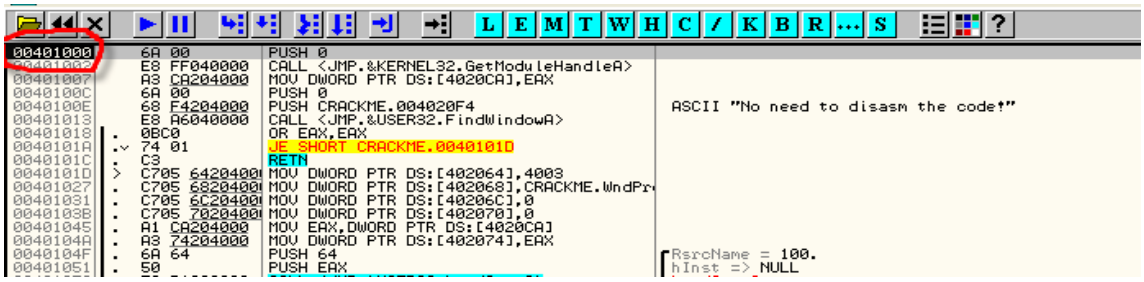
Bueno ya hemos terminado de explicar las instrucciones assembler y ahora se nos presenta una disyuntiva, por donde seguir, hay muchísimo que aprender y practicar, por lo cual trataremos de ir como siempre pasito a pasito, sin apuros usando lo que aprendimos, y agregando mas cosas que faltan aun.

Por supuesto nuestra primera victima será el famoso CRACKME DE CRUEHEAD pero no nos limitaremos solo a ir viendo las distintas formas de crackearlo, si no también que nos iremos ubicando con conceptos que luego nos serán básicos a la hora de profundizar en el arte.

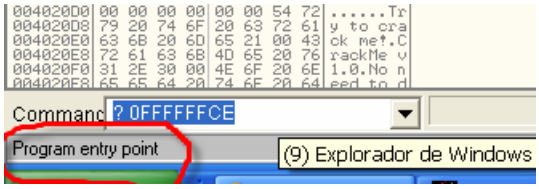
Abramos el famoso crackme en OLLYDBG y orientémonos un poco antes de empezar.

Veremos en el mismo crackme algunas definiciones que son útiles para cualquier programa.

**ENTRY POINT**: Es la primera línea que se ejecuta del programa normalmente, no confundir con OEP (Original Entry Point) que es otra cosa que definiremos en partes posteriores de esta introducción, o sea si abrimos un programa en OLLYDBG, este para y lo analiza, allí donde termina de analizar y queda detenido es el ENTRY POINT del programa.

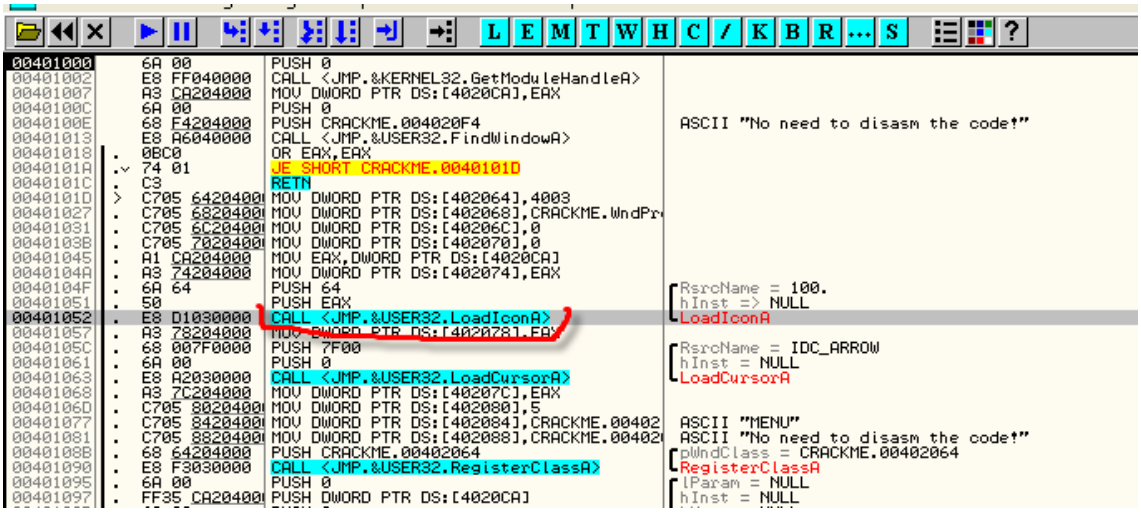



En nuestro caso del Crackme de Cruehead, el ENTRY POINT será 401000 y OLLYDBG nos muestra en el margen inferior que normalmente usa para avisarnos porque esta detenido un programa, que en este caso estamos detenidos en el ENTRY POINT.




Casi todos los programas (el 99 %), cuando arrancan en OLLYDBG se detienen en el ENTRY POINT, los que no lo hacen es porque tienen alguna modificación especial realizada para evitar que pare en el mismo, ese tipo de trucos veremos mas adelante, pero la idea es saber esto.

Otro concepto que necesitamos y que usaremos en el de DLLs y sus APIS




Vemos que en ciertos puntos del programa el mismo nos muestra una CALL o JMP que salta en vez de a una dirección como normalmente vimos por ejemplo CALL 401020 o JMP 421367, en la imagen anterior vemos que el call por ejemplo es

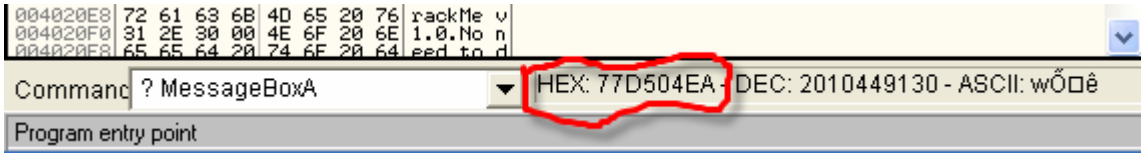
CALL LoadIconA

Y a la derecha nos muestra cierta información, pero que es en este caso LoadIconA?

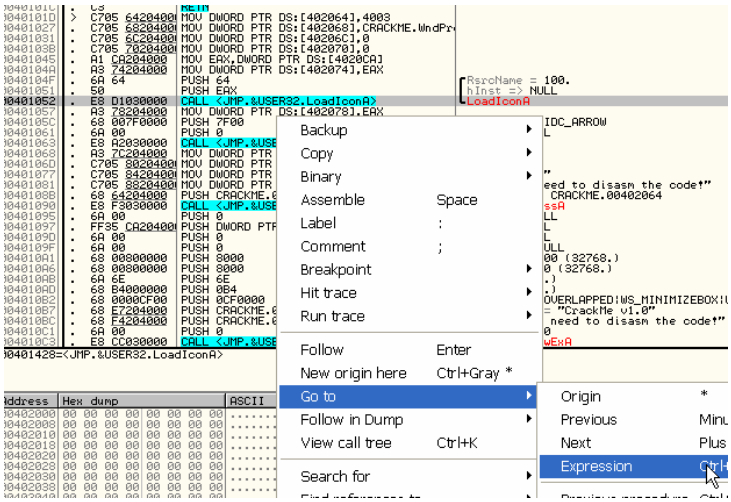
El sistema WINDOWS utiliza para evitar que los programadores repitan las mismas rutinas que en casi todos los programas son usadas, un sistema de archivos de extensión DLL que son archivos ejecutables, pero además tienen la propiedad de tener FUNCIONES DE EXPORTACION O APIS que no son mas que funciones que pueden ser utilizadas por cualquier programa, para tener que evitar repetir lo mismo en todos los programas.
Así en este caso llama a la DLL user32.dll y ella tiene una función denominada LoadIconA, que realizara cierto trabajo para facilitarme la programación.

El caso mas sencillo de entender es la api MessageBoxA

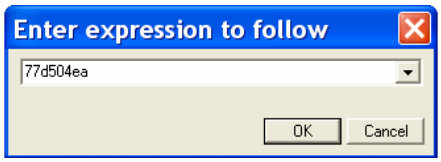
Si en la commandbar del OLLYBD tipeo



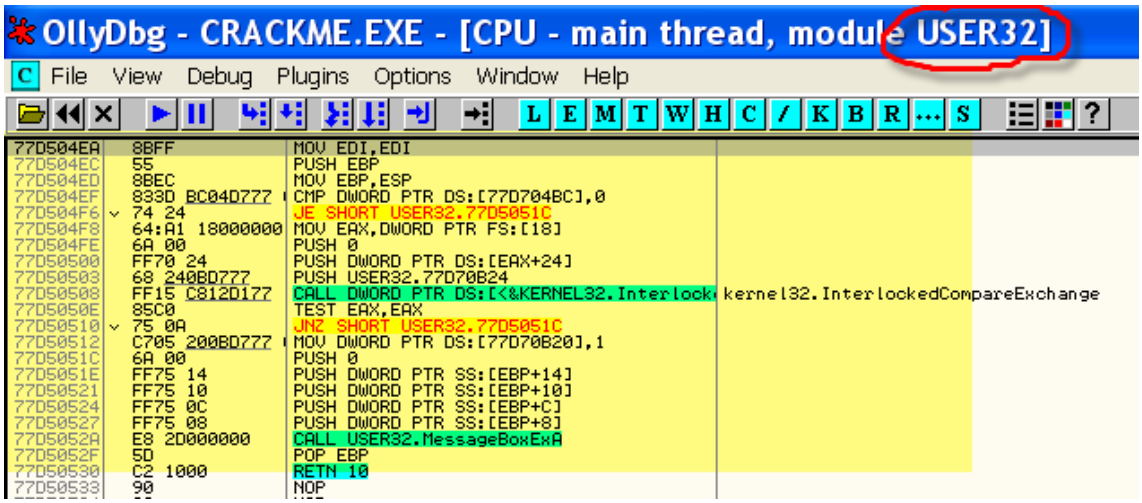

Nos muestra la dirección de dicha api, si voy a mirar a dicha dirección en el listado haciendo CLICK DERECHO y copiando la dirección que salio en sus maquinas ya que puede variar en cada una.




Allí tipeo la dirección que en mi maquina apareció

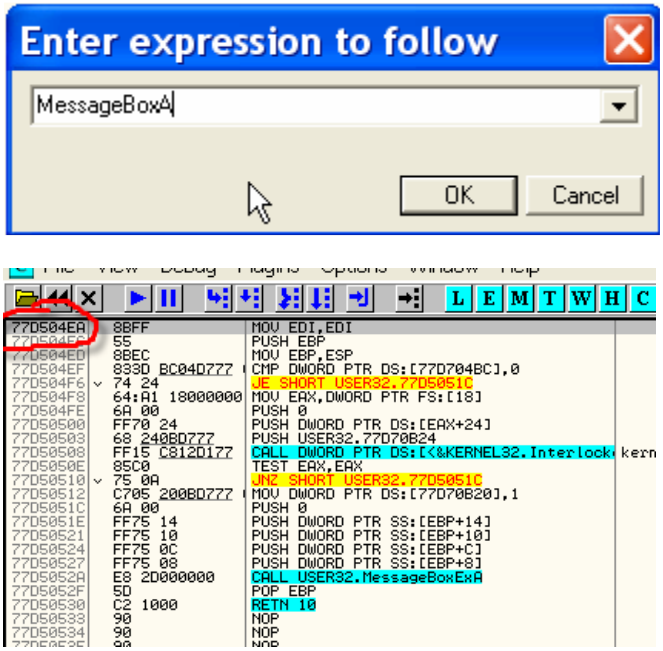



USUARIOS DE WINDOWS 98 no hace esto ya verán porque




Allí vemos que pertenece a la dll llamada USER32.dll y que no es mas que una rutina que termina en un RET, lo único que nos salva esto es de tener que agregar toda esta rutina en nuestro programa, así se hacen programas mas pequeños y se nos facilita la vida a los crackers jeje.

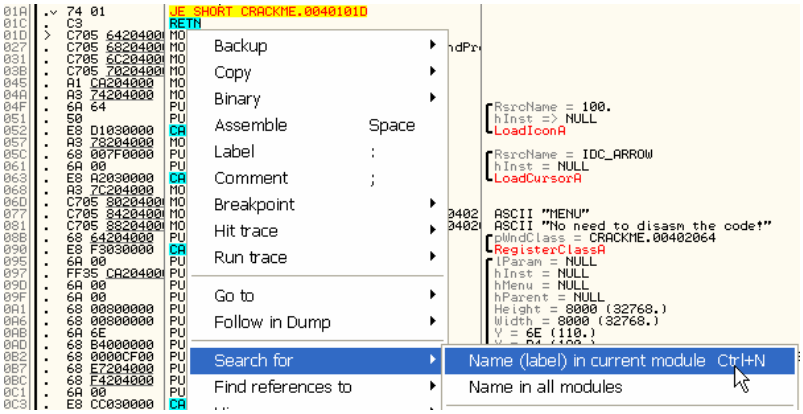
Si apreto la tecla MENOS volveré al punto donde estaba anteriormente y puedo verificar también que si hago GOTO EXPRESIÓN y directamente tipeo MessageBoxA en dicha ventana nos lleva al mismo lugar,



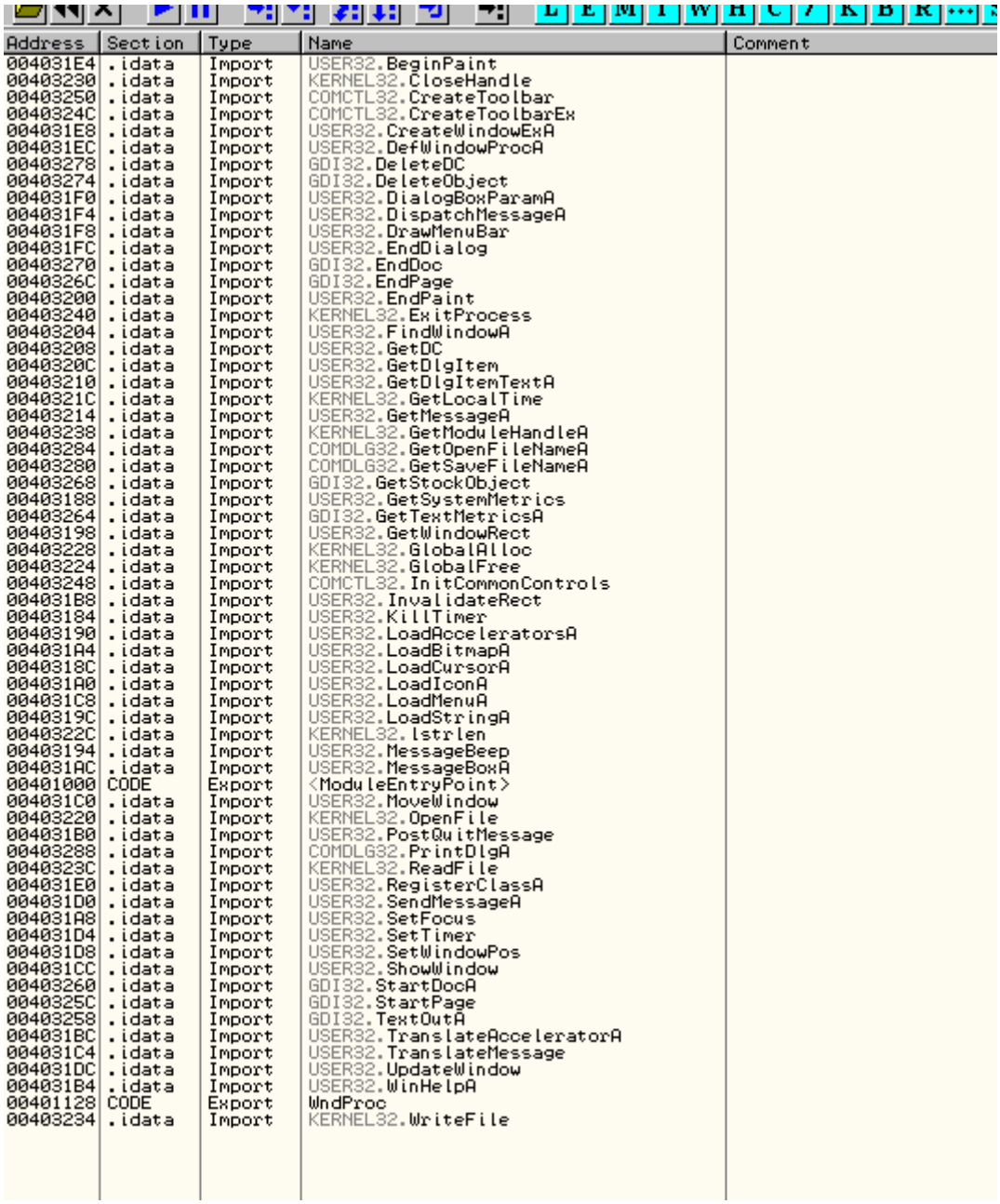

Allí vemos la rutina de la api MessageBoxA correspondiente a USER32.dll

Como verán tuve que tipear MessageBoxA y no messageboxa ya que el nombre de la api es reconocido por OLLYDBG si tipeamos en forma correcta sus mayúsculas y minúsculas, y la pregunta siguiente es pensar como se como se escribe una api cuales son mayúsculas y minúsculas en su nombre, pues es sencillo.

Volvamos con MENOS al ENTRY POINT

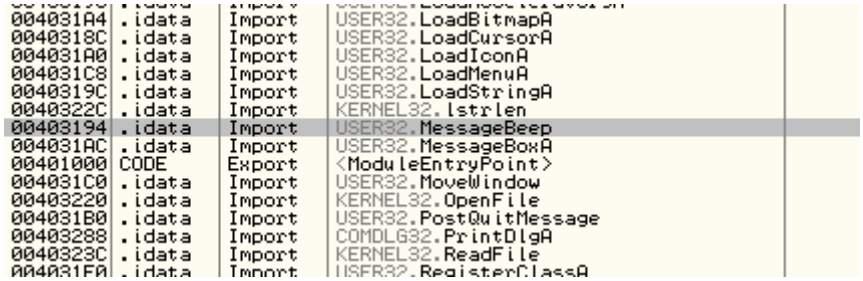



Haciendo en el listado CLICK DERECHO-SEARCH FOR NAME (label) in CURRENT MODULE nos dará la lista de Apis o NAMES utilizadas por este CRACKME DE CRUEHEAD.



| 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 | COMDLG32.GetOpenFileNameA | |
| 00403280 | .idata | Import | COMDLG32.GetSaveFileNameA | |
| 00403268 | .idata | Import | GDI32.GetStockObject | |
| 00403188 | .idata | Import | USER32.GetSystemMetrics | |
| 00403264 | .idata | Import | GDI32.GetTextMetricsA | |
| 00403198 | .idata | Import | USER32.GetWindowRect | |
| 00403228 | .idata | Import | KERNEL32.GlobalAlloc | |
| 00403224 | .idata | Import | KERNEL32.GlobalFree | |
| 00403248 | .idata | Import | COMCTL32.InitCommonControls | |
| 004031B8 | .idata | Import | USER32.InvalidateRect | |
| 00403184 | .idata | Import | USER32.KillTimer | |
| 00403190 | .idata | Import | USER32.LoadAcceleratorsA | |
| 004031A4 | .idata | Import | USER32.LoadBitmapA | |
| 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.PostQuitMessage | |
| 00403288 | .idata | Import | COMDLG32.PrintDlgA | |
| 0040323C | .idata | Import | KERNEL32.ReadFile | |
| 004031E0 | .idata | Import | USER32.RegisterClassA | |
| 004031D0 | .idata | Import | USER32.SendMessageA | |
| 004031A8 | .idata | Import | USER32.SetFocus | |
| 004031D4 | .idata | Import | USER32.SetTimer | |
| 004031D8 | .idata | Import | USER32.SetWindowPos | |
| 004031CC | .idata | Import | USER32.ShowWindow | |
| 00403260 | .idata | Import | GDI32.StartDocA | |
| 0040325C | .idata | Import | GDI32.StartPage | |
| 00403258 | .idata | Import | GDI32.TextOutA | |
| 004031BC | .idata | Import | USER32.TranslateAcceleratorA | |
| 004031C4 | .idata | Import | USER32.TranslateMessage | |
| 004031DC | .idata | Import | USER32.UpdateWindow | |
| 004031B4 | .idata | Import | USER32.WinHelpA | |
| 00401128 | CODE | Export | WndProc | |
| 00403234 | .idata | Import | KERNEL32.WriteFile | |

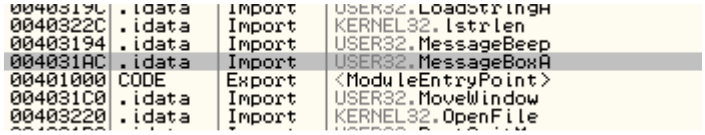
Para encontrar la que quiero no necesito bajar y buscarla a mano solo en esta ventana apretando la M



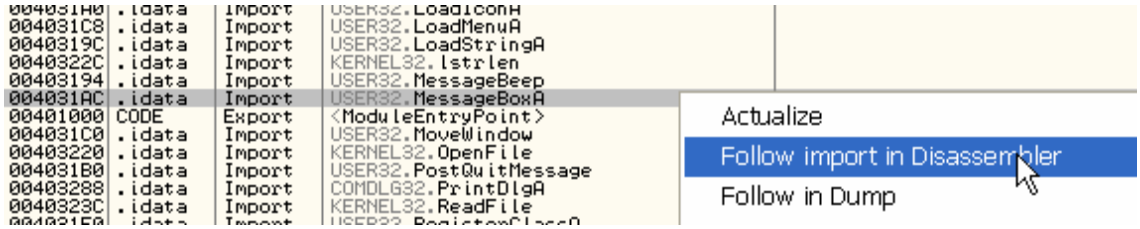
| | | | |
|---|---|---|---|
| 004031A4 | .idata | Import | USER32.LoadBitmapA |
| 0040318C | .idata | Import | USER32.LoadCursorA |
| 004031A0 | .idata | Import | USER32.LoadIconA |
| 004031C8 | .idata | Import | USER32.LoadMenuA |
| 0040319C | .idata | Import | USER32.LoadStringA |
| 0040322C | .idata | 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.PostQuitMessage |
| 00403288 | .idata | Import | COMDLG32.PrintDlgA |
| 0040323C | .idata | Import | KERNEL32.ReadFile |
| 004031E0 | .idata | Import | USER32.RegisterClassA |

Vemos que el cursor se acomoda en la primera api que empieza con M si a continuación sigo tipeando letras del nombre de la api,

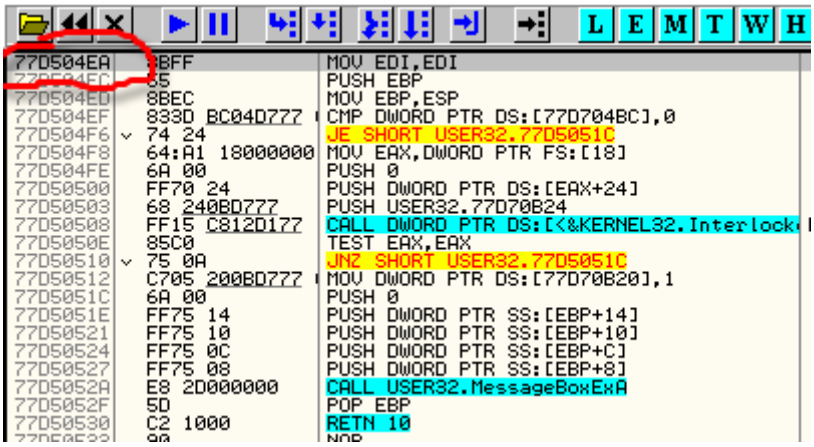

Arriba nos va marcando las letras que tipeamos y el cursor halla la api



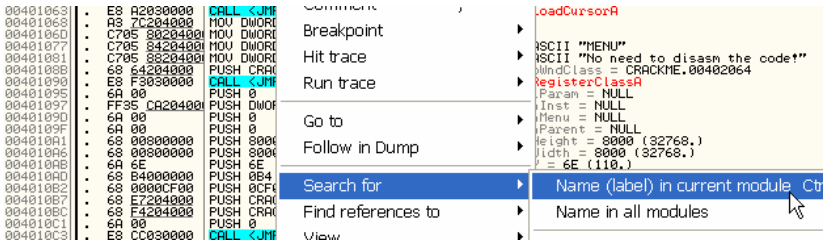

Si hago click derecho encima del nombre de la api tengo diferentes opciones




Si elijo FOLLOW IMPORT IN DISSASSEMBLER nos llevara a la dirección de la api, este es otro método para llegar a la dirección, si no tenemos ganas de tipear en la comandbar.

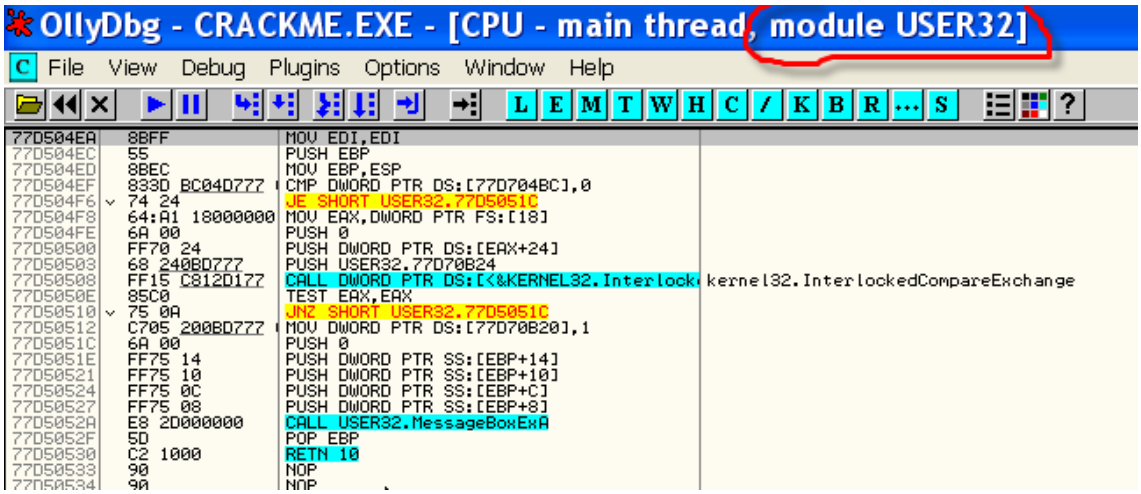



Aquí vemos un común error que cometen muchos que recién se inician, si allí en la api hago SEARCH FOR NAME (LABEL) IN CURRENT MODULE




OLLYDBG buscara en este caso las APIS o NAMES correspondientes a USER32.dll ya que allí lo especifica busca en el MODULO que esta visible en el listado, CURRENT MODULE y el

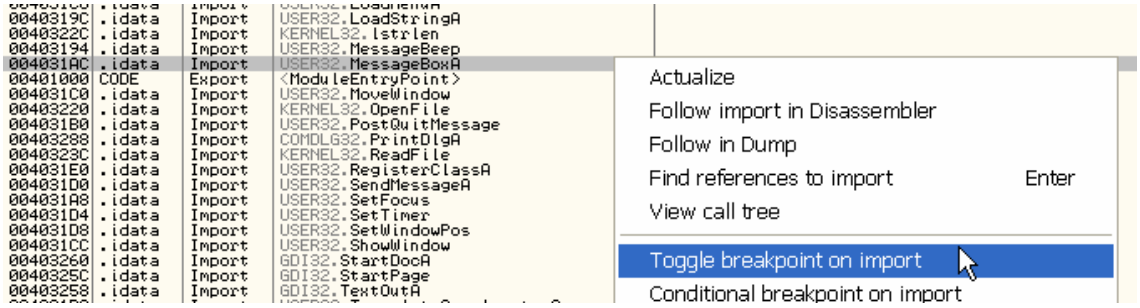
que esta visible cuando estamos en la api es USER32.dll lo podemos ver en el mismo OLLYDBG arriba




Aun cuando nosotros no estemos ejecutando en este momento la api y solo mirando al buscar NAMES saldran las de USER32.dll que no son las que en este momento nos interesan, por lo tanto si queremos volver a ver las apis del Crackme de Cruehead debemos apretar MENOS hasta volver a ver el listado del CRACKME por ejemplo en el ENTRY POINT y allí si, si hacemos SEARCH FOR NAMES nos mostrara las apis del mismo.

## PARA WINDOWS NT/2000, XP o 2003

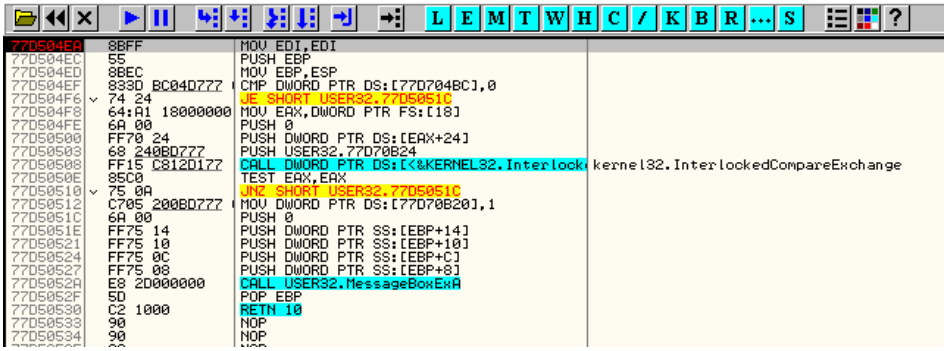
A PARTIR DE AQUÍ EL TUTORIAL CONTINUA PARA SISTEMAS NT/2000 y XP, les recomiendo a los que tienen WINDOWS 95 o 98 pasarse a cualquiera de esos sistemas, que allí es donde OLLYDBG es mas potente, pero si no pueden hacerlo salteen esta parte y vayan a donde dice APENDICE PARA WINDOWS 98



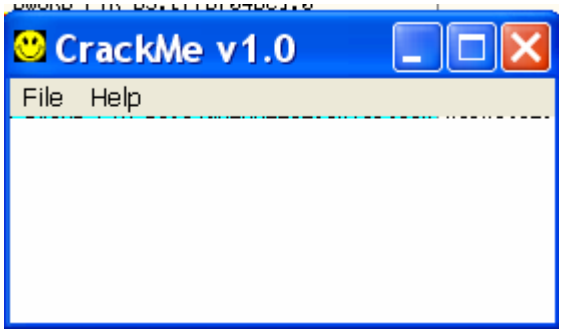

Vemos que cuando estoy en la lista de apis del crackme otra opción es poner un BREAKPOINT en dicha api, asi cuando el programa llama a la misma parara, hagámoslo con CLICK DERECHO-TOGGLE BREAKPOINT ON IMPORT.

También podríamos hacerlo directamente en la commandbar tipeando

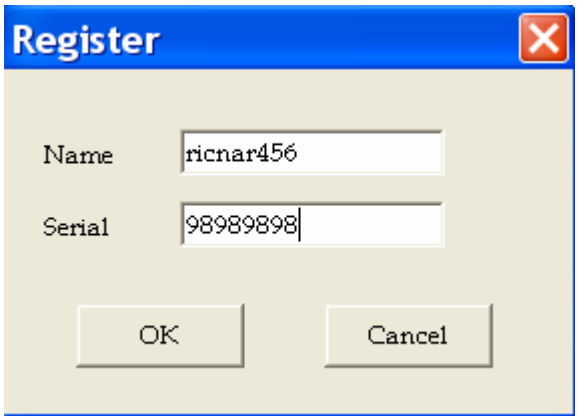
Bp MessageBoxA




Si ahora vemos la direccion de la api apreciamos que se ha puesto un BREAKPOINT en la dirección de inicio de la misma, de esta forma si la api es usada parara OLLYDBG en ella, apretemos F9 para correr el CRACKME DE CRUEHEAD y ver si para alli.



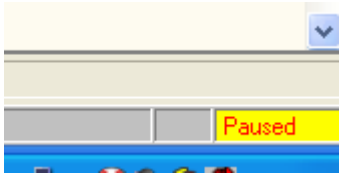

Vemos que aparece la ventanita y aun no paro vayamos a HELP-REGISTER



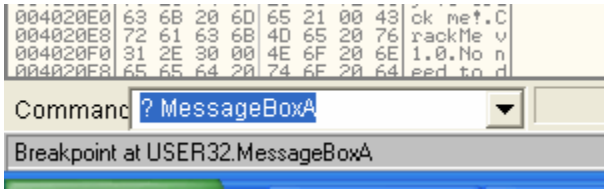

En la ventana tipeemos algún nombre y serial falso y apretemos OK

Vemos que el OLLYDBG paro veamos porque

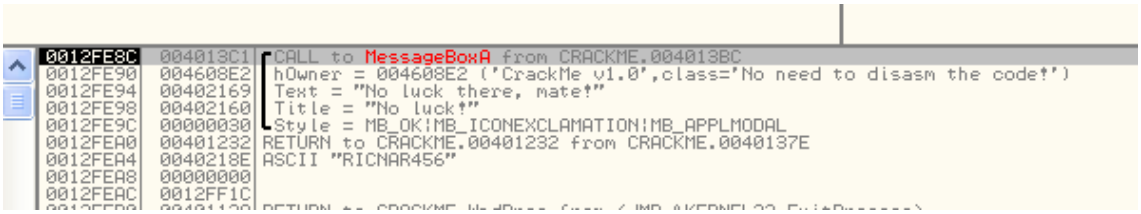
En la esquina inferior derecha vemos que esta PAUSADO



Y en la esquina inferior  izquierda siempre nos muestra el motivo porque paro.

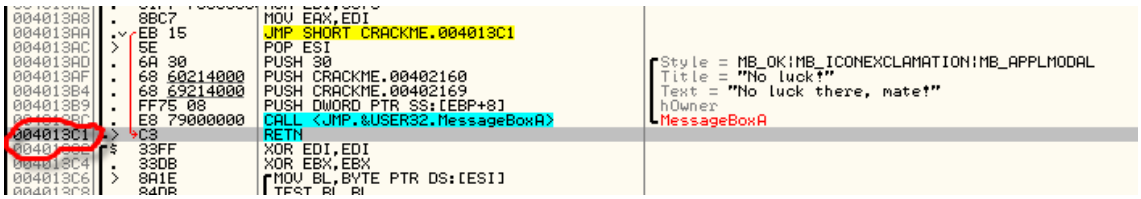



Allí dice BREAKPOINT en USER32.MessageBoxA o sea que paro en nuestro BREAKPOINT en la API.




Vemos que OLLYDBG nos muestra información ya que cada api se llama con determinados parámetros que se pasan al stack antes de llamarla, y en este caso vemos

En la primera línea la dirección de retorno del CALL que nos hizo llegar aquí en este caso 4013C1

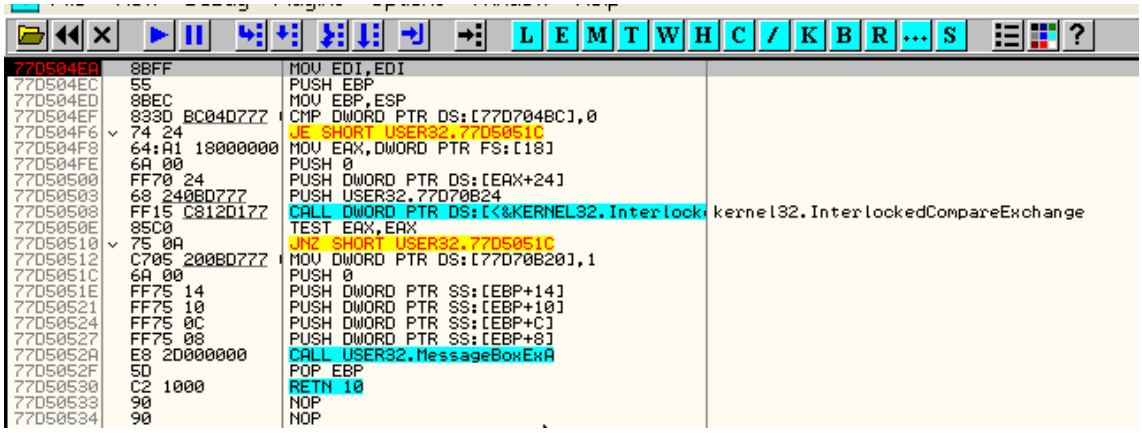



Como vimos cuando explicamos CALL Y RET siempre se pasa al stack la dirección de retorno del mismo, y allí esta cuando llegue al RET de la api volverá a 4013c1.

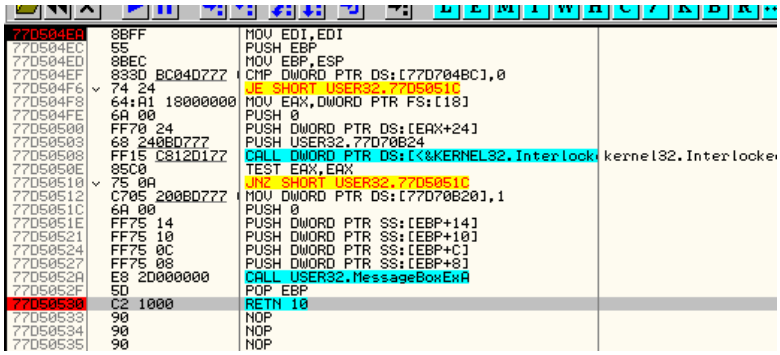
Luego abajo los parámetros de la misma nos muestran entre otras cosas ya que la api MessageboxA es la encargada de mostrarnos los típicos cartelitos de mensajes de Windows, el titulo del mensaje, el texto, el estilo etc.

Ya vemos que el TEXTO es NO LUCK THERE, MATE que es el cartel que coloca el Crackme de Cruehead cuando el serial que introdujiste no es correcto.

Pues allí estamos y el crackme esta a punto de mostrarnos el fatídico cartelito.



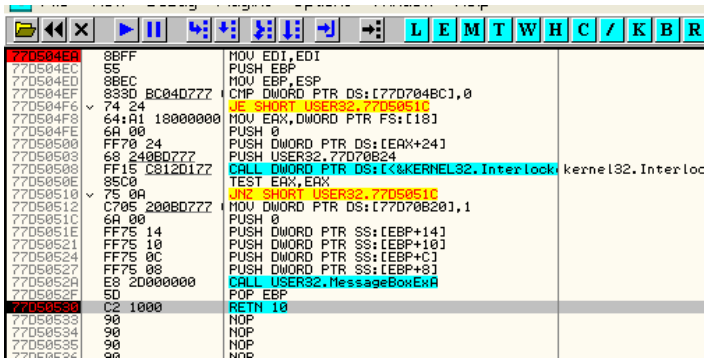

Para que vean que no le miento y que la api es la encargada de hacerlo, pongamos un BREAKPOINT en el RET 10 que es el final de la api, allí lo vemos debajo, en su maquina puede cambiar pero siempre es el primer RET a partir de la dirección de inicio de la api que vemos al ir bajando.




Allí tenemos la api cercada apretemos F9 o RUN.

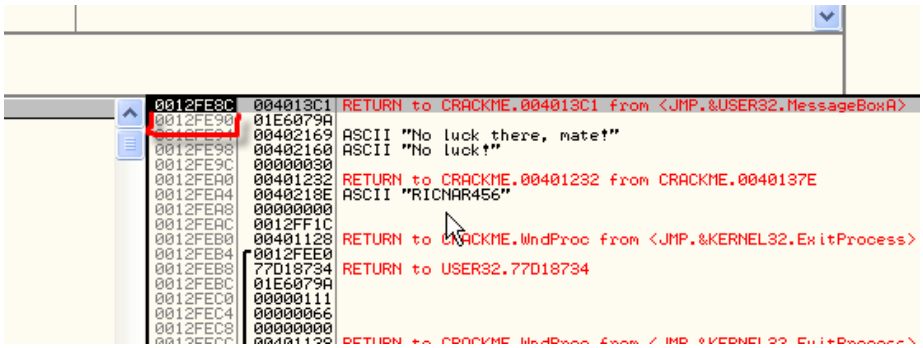

Allí vemos el típico cartelito de la MessageboxA y como nos aviso OLLYDBG el titulo es NO LUCK! Y el texto NO LUCK THERE MATE! o sea que no tuvimos suerte con nuestro serial.

Por supuesto al aceptar para en el RET de la api



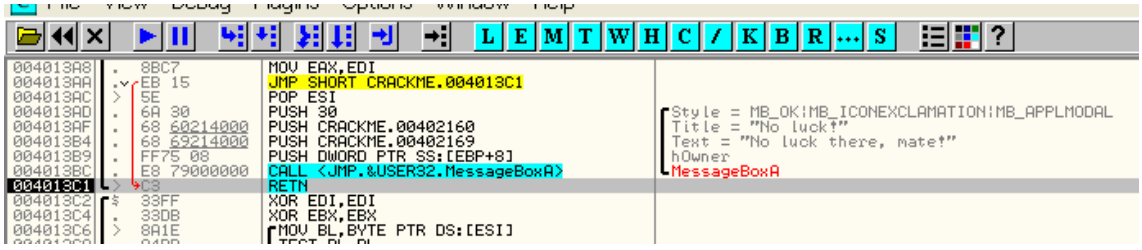

Como vimos el proceso de aparición del cartel, ocurrió entre el inicio y final de la api, allí estamos en el RETN10.

Ya que no aclaramos la diferencia entre en RETN 10 y el RET común lo haremos aquí en este caso si el RET fuera común, al ejecutarlo, volvería a la dirección de retorno 4013C1

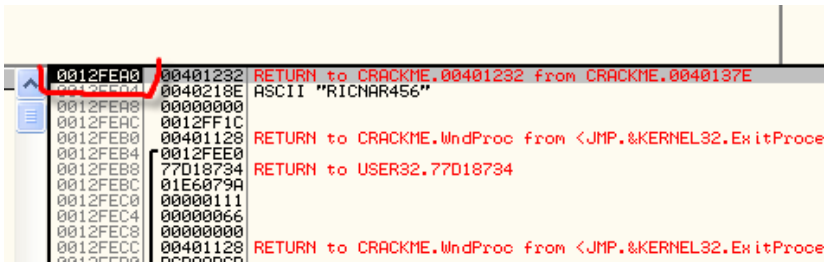



Y al quitar el valor de la dirección de retorno de arriba del stack, este quedaría en mi caso en 12Fe90, en el caso del RETN10 vuelve a la misma dirección 4013C1, pero a ESP se le suma 10 con lo cual el stack debería quedar en 12fe90 mas 10 seria esp=12fea0 veamos apretemos F7.

RETORNO DEL APÉNDICE PARA WINDOWS 98 desde aquí sigue para todos los SO.

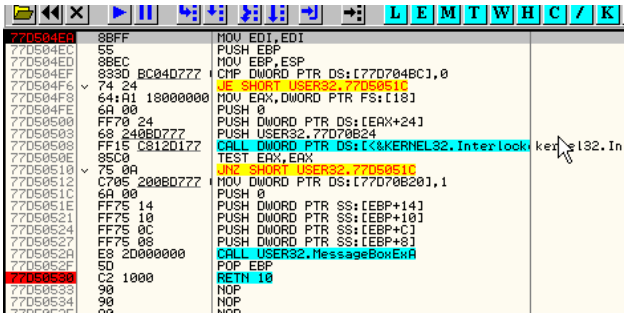


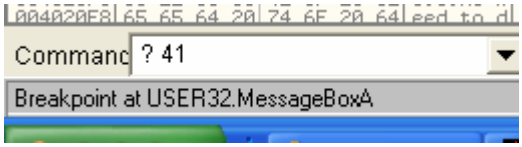

Allí retornamos de la api al crackme y vemos que en el stack se cumplió lo que mostramos el RETN 10 le suma a ESP 10 mas de lo que valdría si retorna como RET solo.



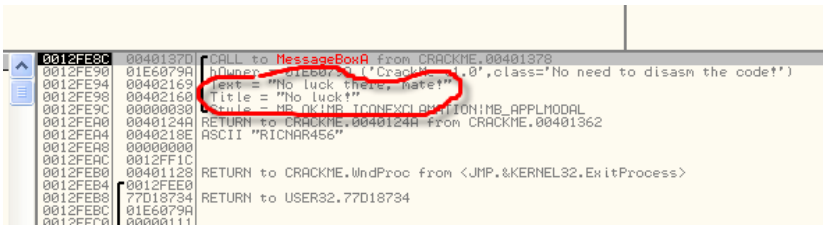
La cuestión es que la decisión ya fue tomada y nosotros estamos en el horno, ya nos dijo mala suerte amigo, el serial que tipeaste no sirve.

Apretemos F9 nuevamente

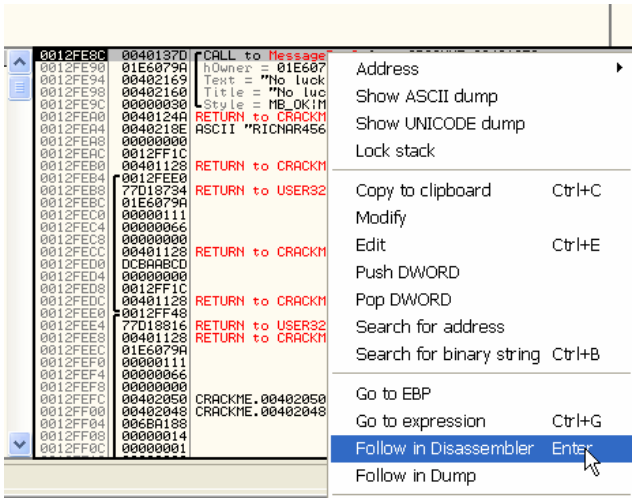


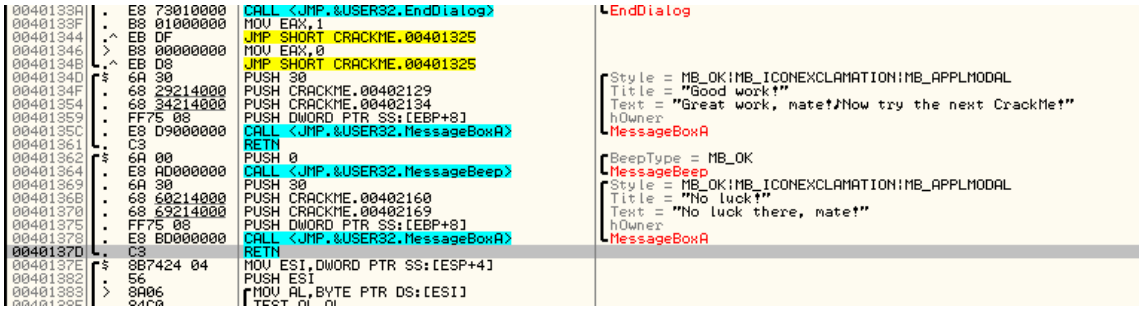

Vemos que para nuevamente en la api, y que por si no me di cuenta me va a decir que no tuve suerte jeje




Vemos que en  este caso la dilección de retorno es 40137d veamos que hay allí con GOTO EXPRESIÓN 40137D en el listado o bien en la primera línea del stack CLICK DERECHO-FOLLOW IN DISASSEMBLER



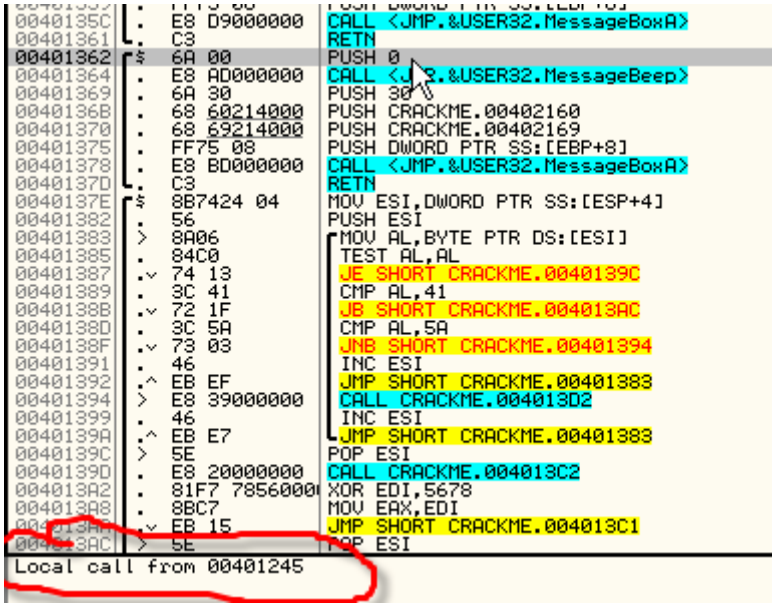

Alli vemos que volveria a 40137D y que estamos dentro del call que llama a MessageboxA que esta justo arriba en 401378.



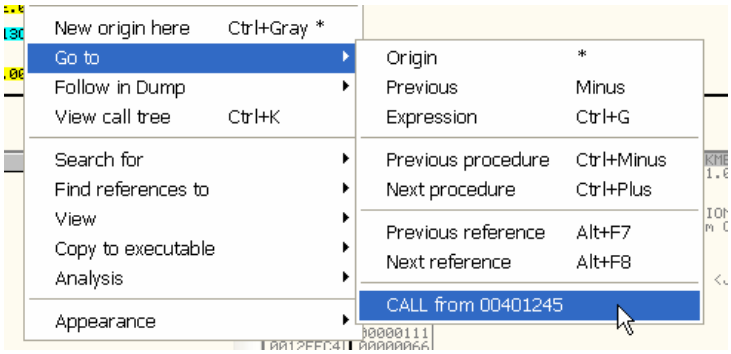

Vemos arriba que hay otro MessageBoxA pero con el mensaje de felicitación de que acertamos GREAT WORK, jeje si pudiéramos llegar allí en vez de al cartel de que no tuvimos suerte seria un primer gran paso.

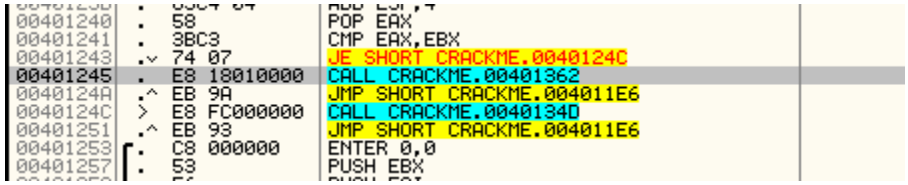

Vemos que OLLYDBG en su análisis inicial nos muestra unos corchetes, que significa que eso es una rutina que empieza y termina allí, vemos que hay dos, una para el cartel de NO LUCK que empieza en 401362 y otra para el GREAT WORK que empieza en 40134D.

Si vamos a 401362 que es el inicio de la rutina donde estamos (aun dentro del MessageBoxA) y hago click allí el OLLYDBG me muestra en las aclaraciones.

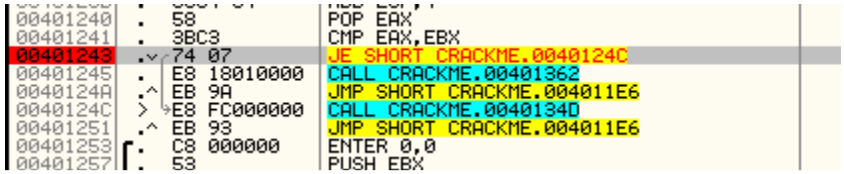



Quiere decir que OLLYDBG sabe que esa rutina ya que es DIRECTA es llamada desde 401245 veamos allí que hay CLICK DERECHO – GOTO CALL FROM 401245.

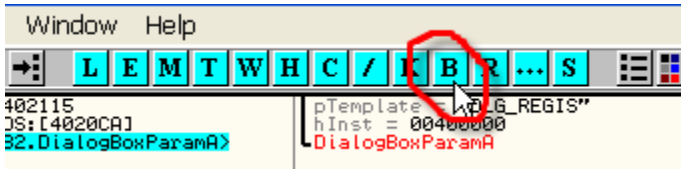


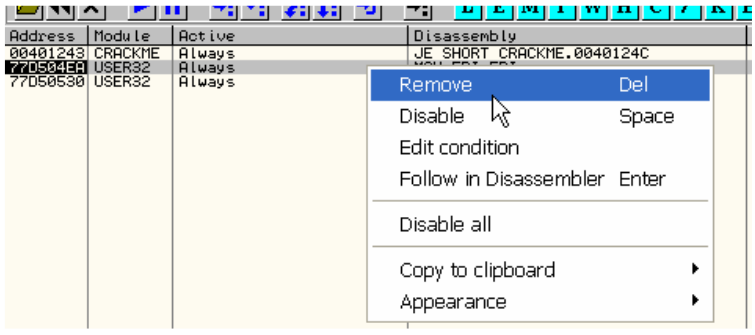




Hmm esta zona parece muy sospechosa hay una comparación y un salto y según el resultado de ese salto va al call de 401362 que es el que muestra NO LUCK y si no va al CALL 40134D que muestra GREAT WORK, esto no podemos perderlo pongamos un BREAKPOINT en dicho salto condicional.



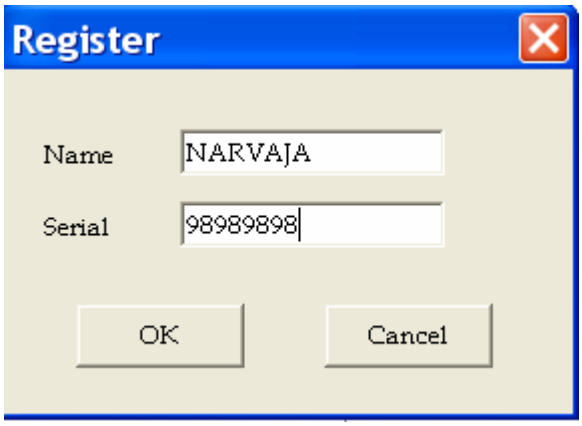

Y quitemos los breakpoints en la api MessageBoxA por ahora, eso puede hacerse en la ventana B de breakpoints



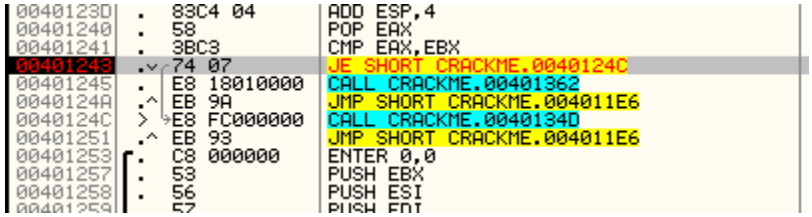




Quito CON CLICK DERECHO-REMOVE los dos BREAKPOINTS y dejo solo el de 401243 que es el salto condicional.

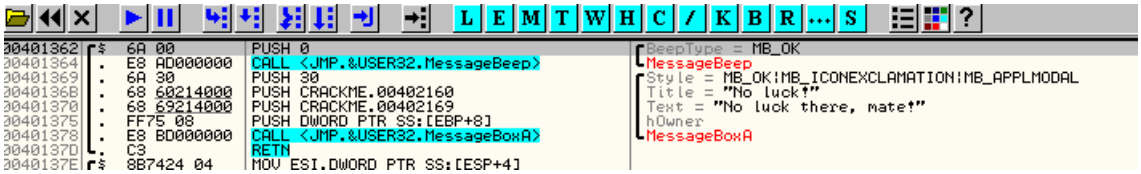
Ahora doy RUN con F9 acepto el NO LUCK que estábamos antes y vuelvo a ingresar a poner el nombre y serial en este caso pondré, usen el mismo que yo, ya verán porque



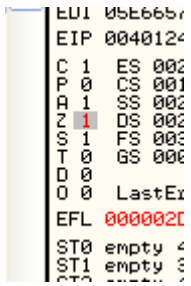

Apreto OK



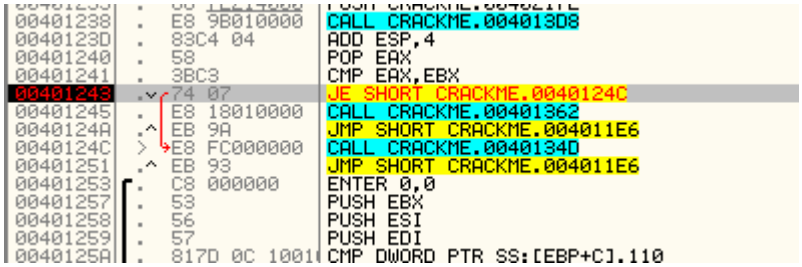

Vemos que de la comparación NO SALTARA al no ser EAX y EBX iguales y seguirá ejecutando
en 401245 que nos llevara a CALL 401362 que sabemos que allí esta el cartel malo, si no recuerdan, hagan click en 401245 y con CLICK DERECHO –FOLLOW pueden ver adonde iria
.



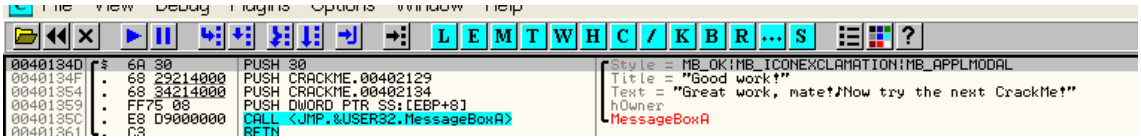

Allí vemos si ese salto condicional no salta pues, ira al cartel de que el serial es malo, que pasa si cambio el salto condicional, haciendo doble click en el flag Z



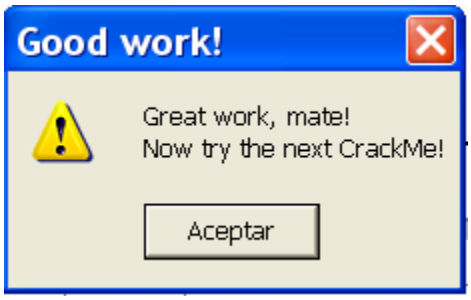

Allí cambie el FLAG Z a 1 que seria como si EAX y ECX en la comparación hubieran sido iguales y la resta de la comparación hubiera sido cero y hubiera activado así el flag Z, el salto JE salta si el FLAG Z es uno, así que ahora saltara veamos.



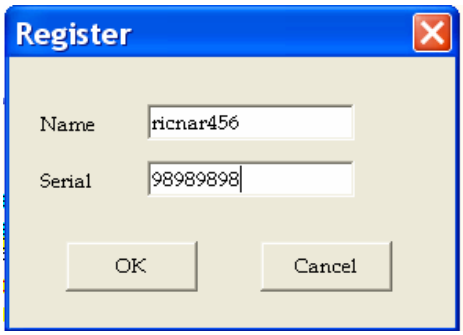

Allí lo cambiamos y si vemos en 40124c con FOLLOW vemos que ira a



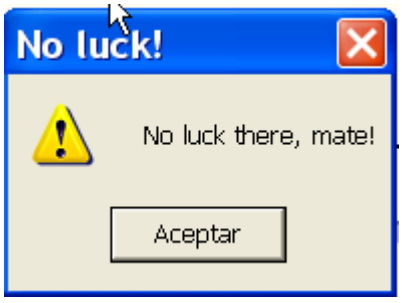

Jeje apretemos RUN o F9



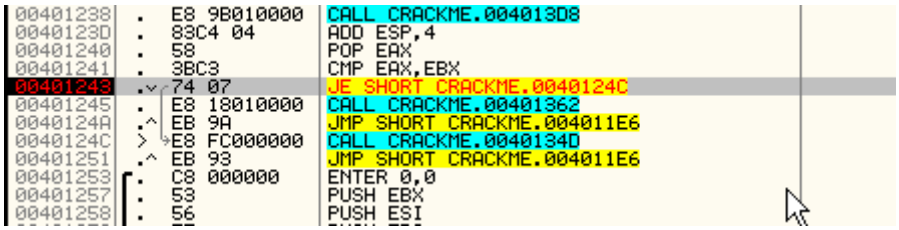

O sea que esa comparación y ese salto condicional que invertimos es el punto de inflexión de la registración o validación del serial en este crackme, según si salta o no, pues sale el cartel bueno o malo, pero antes habíamos visto que había 2 carteles malos porque es eso, aquí no salio el primero, y eso es porque el crackme detecta el uso de números en el nombre (antes había puesto como nombre ricnar456) si es así te saca un primer cartel de NO LUCK, prueben nuevamente.



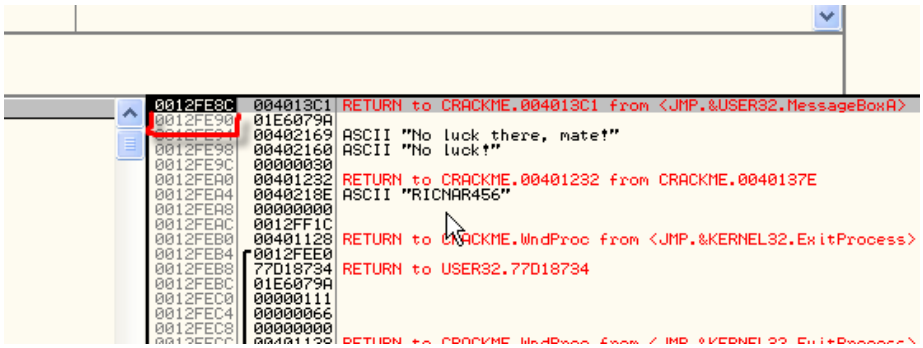

Al aceptar



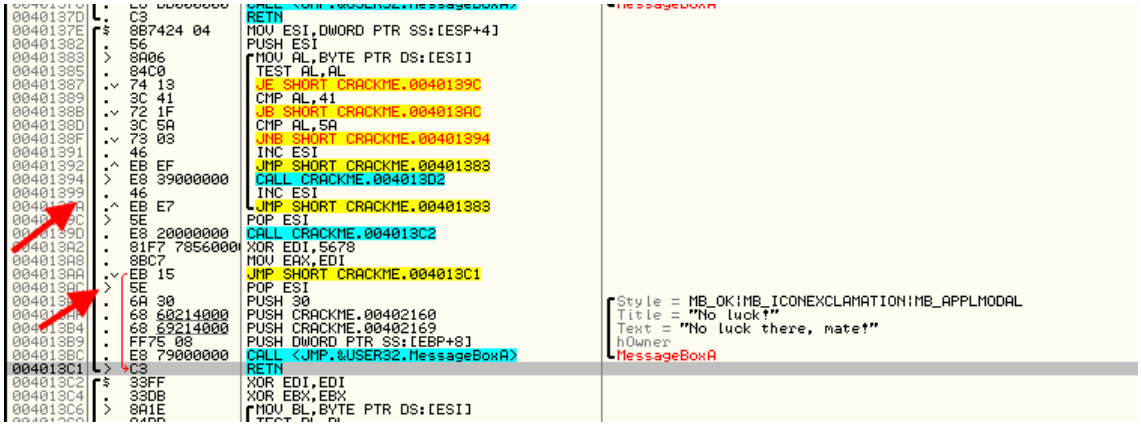
Y recién al aceptar este primero, llega al salto condicional



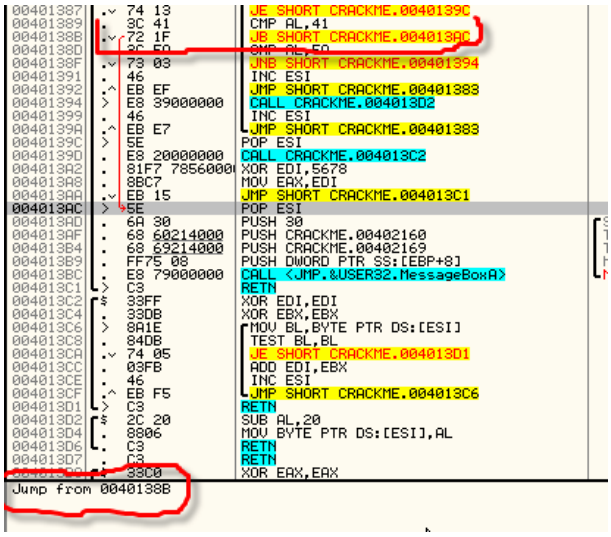

Que muestra el mensaje definitivo.
Como podemos recordar la primera vez que paramos en la api MessageBoxA en este tute fue por el primer cartel ese



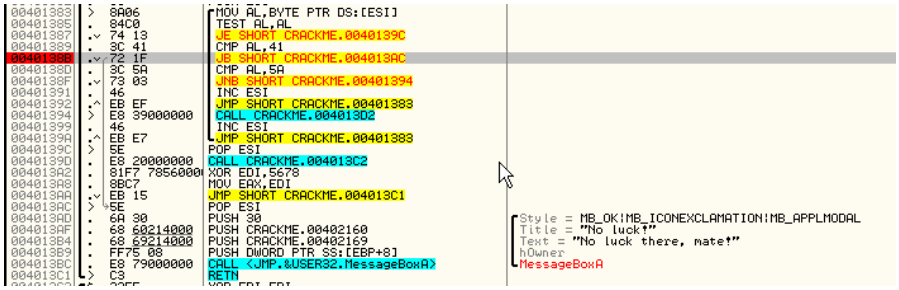

Y la dirección de retorno era 4013C1



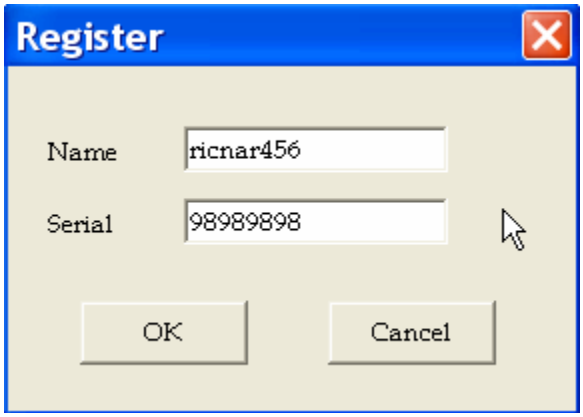

Allí vemos que el OLLYDBG en su análisis me muestra una rutina que comienza en 40137e y termina en 4013C1 justo donde retorna de la api.
Vemos también otra ayuda de OLLYDBG en 4013AC hay un > que significa que hay un salto que apunta hacia esa dirección si hacemos click allí nos aclarara mas.



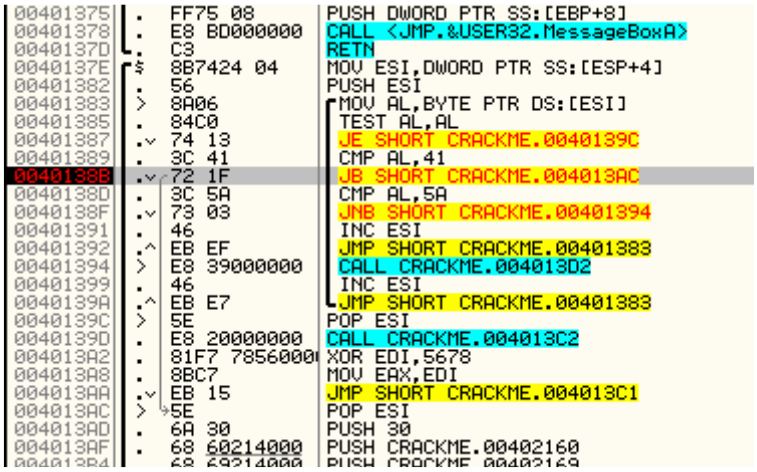

Allí vemos ora comparación y un salto condicional que nos llevan al cartel maldito pongamos otro BREAKPOINT allí.




Y demos RUN acepto el cartel malo anterior y voy a poner de nuevo

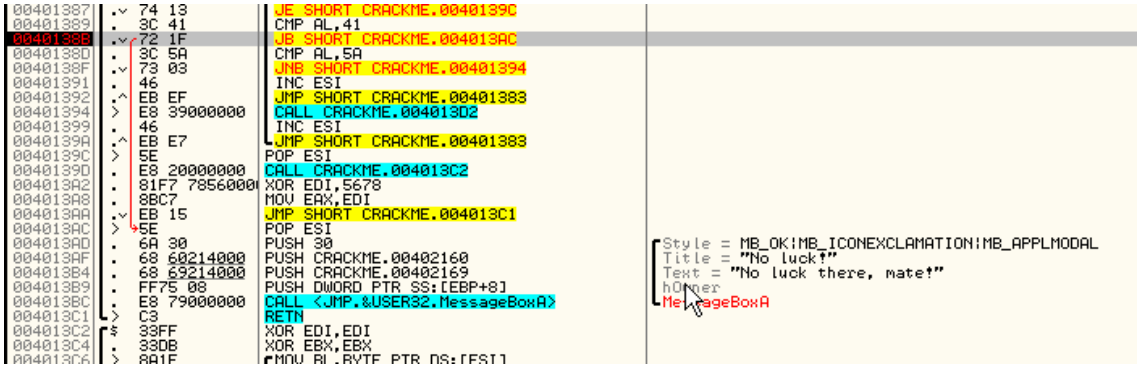


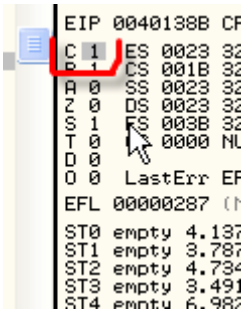

Al aceptar



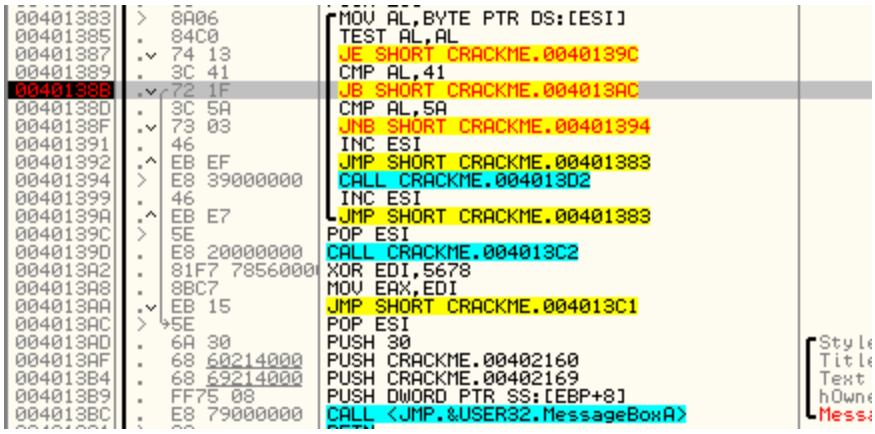

Vemos que no saltara la primera vez que para hacia el cartel malo, aunque si compara cada letra que tipee a ver si son números parara una vez por cada letra, F9 nuevamente

A la 7ma vez que apreto recordar que en ricnar456 el 4 es la séptima letra me quiere mostrar el cartel malo saltando

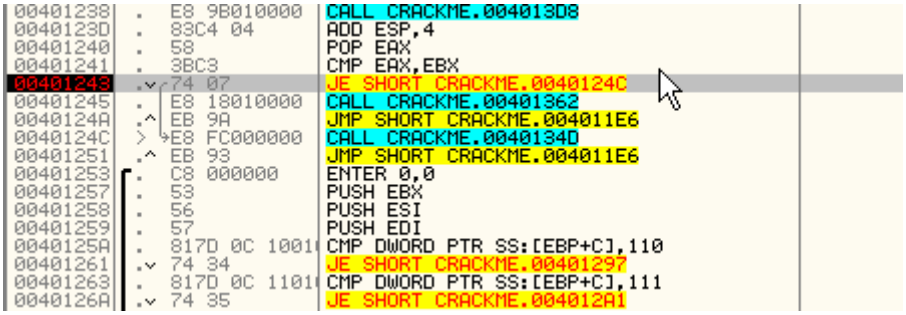


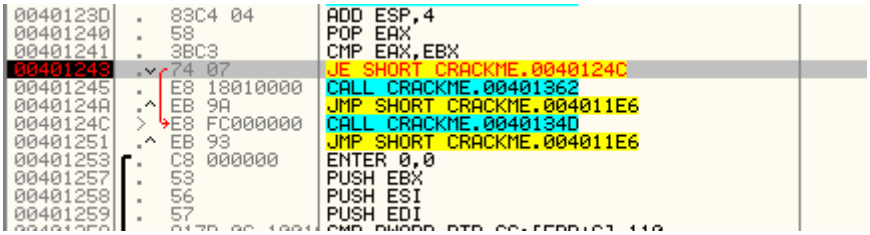

Habíamos visto que los JB saltan si es mas bajo y se activa el FLAG C, hago doble click en el FLAG C




Ahora ya no salta, apreto F9 y repito el mismo procedimiento para los otros dos números que puse en mi nombre fuerzo que nunca salte.



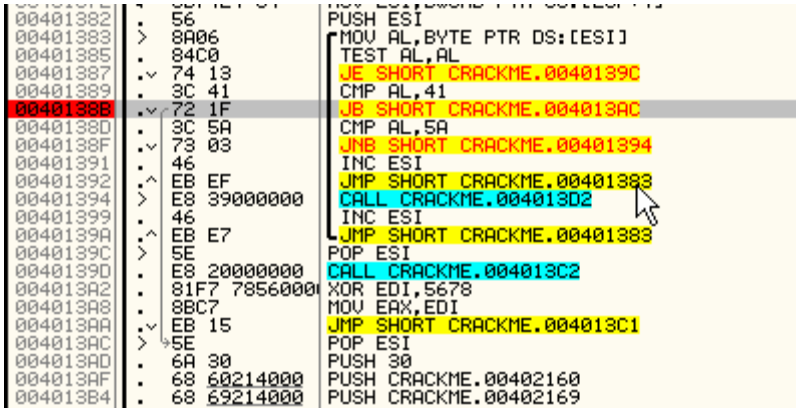

Luego de eso si llego a la comparación final y la cual debe saltar hago doble click en Z



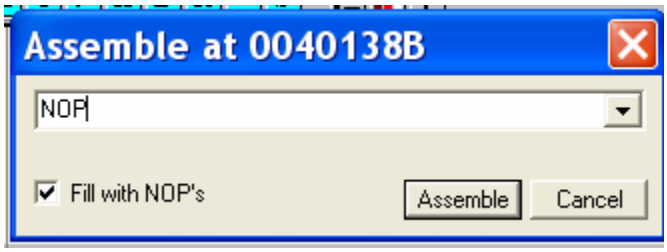

Y al apretar F9

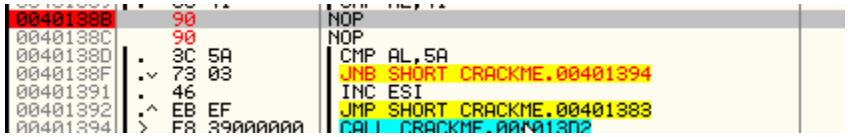

Ahora por supuesto todo esto lo hemos hecho en memoria cambiando FLAGS, como hacemos para guardar definitivos estos cambios y que el crackme acepte cualquier user y serial sin OLLYDBG.

Vayamos al primer salto

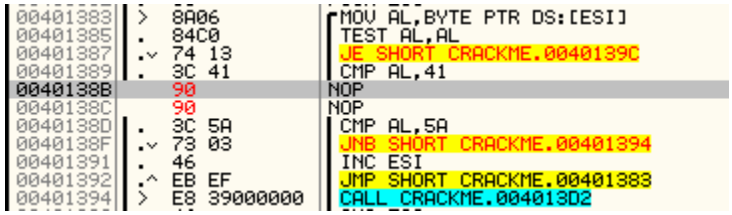



Lo que nosotros hemos hecho en este salto es forzarlo mediante los flags a que no salte nunca a pesar de los valores de la comparación, eso es similar a NOPEAR el salto condicional, si hago click allí, apreto la barra espaciadora y escribo NOP.

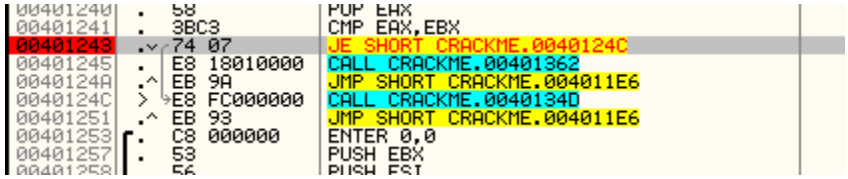


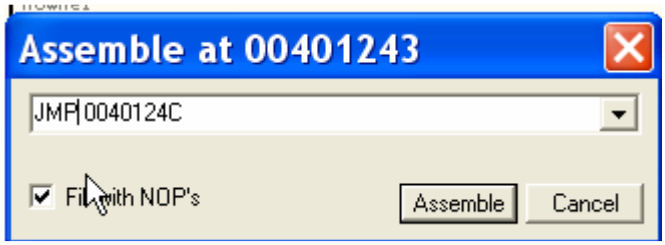

Vemos que nunca saltara quito ese BREAKPOINT apretando F2

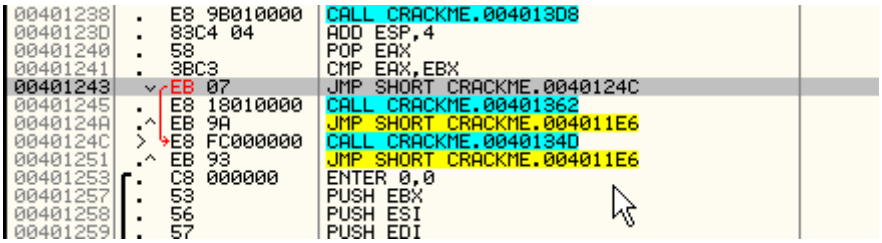



Ahora veamos el otro salto

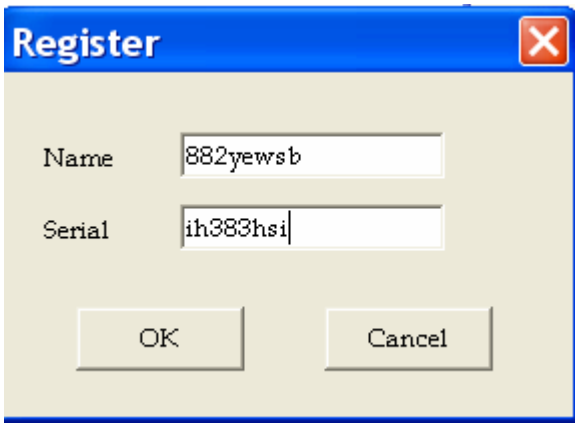



En esta caso fue al revés lo forzamos a saltar siempre mediante la manipulación del FLAG Z, lo cual seria equivalente a cambiar el salto condicional por un JMP.

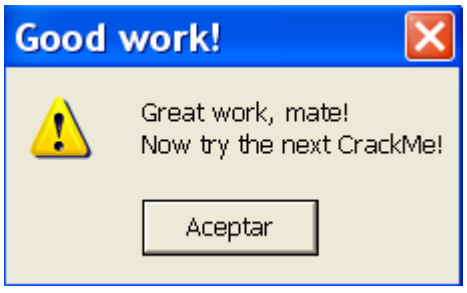


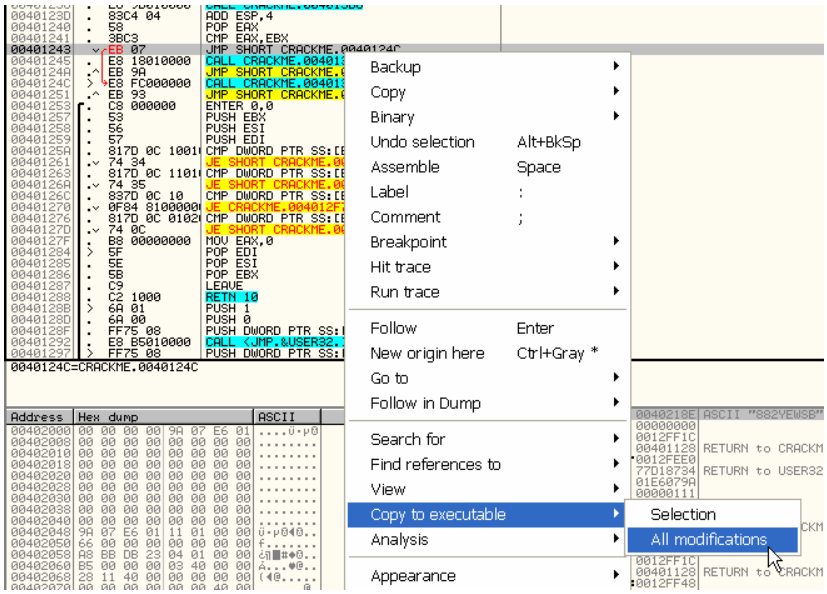

Allí siempre saltara, quitamos también el breakpoint y sin salir de OLLYDBG probamos si quedo bien, apreto F9



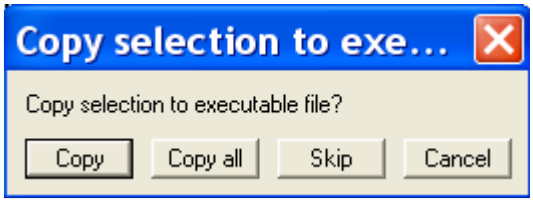

Al apretar OK



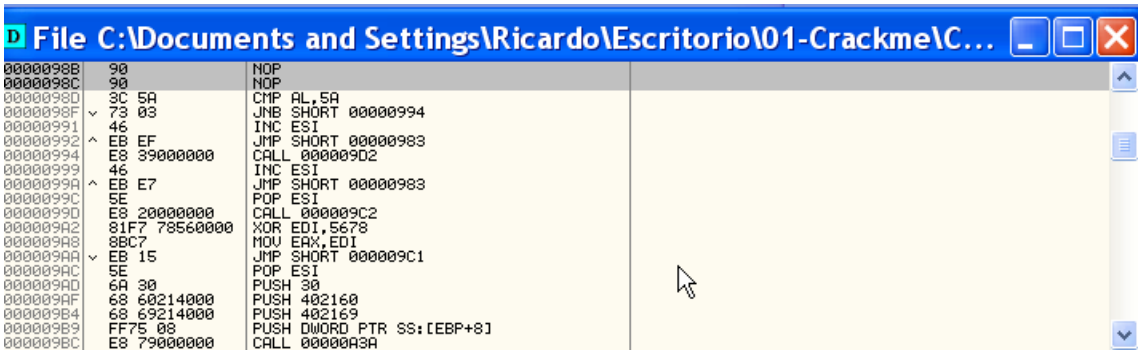

Igual recordamos que cuando escribimos en OLLY con la barra espaciadora o ASSEMBLE los cambios desaparecían al reiniciar, tenemos que hallar la forma de que los guarde de la memoria al archivo definitivo, eso se hace de la siguiente manera



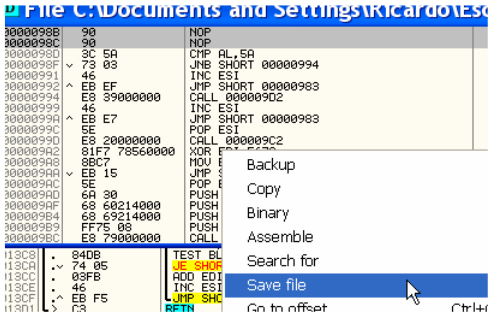

CLICK DERECHO en cualquier parte del listado COPY TO EXECUTABLE-ALL MODIFICATIONS allí se nos abre esto

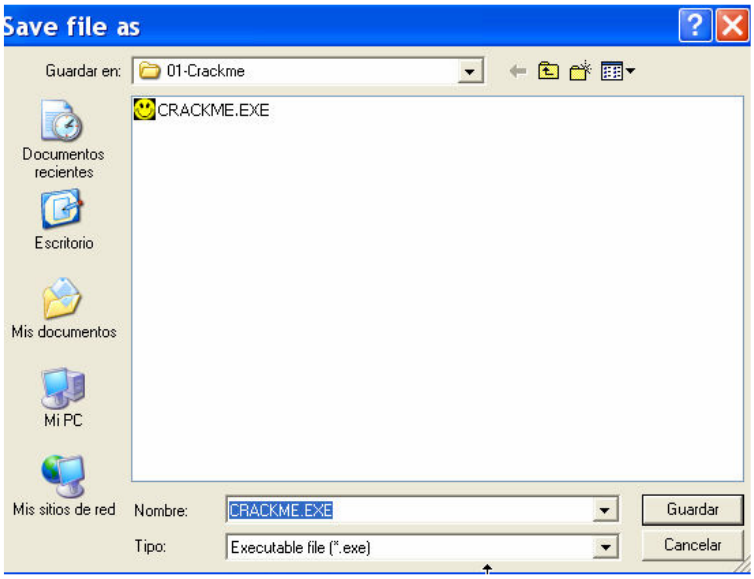



Elegimos COPY ALL para que copie los dos cambios que hicimos

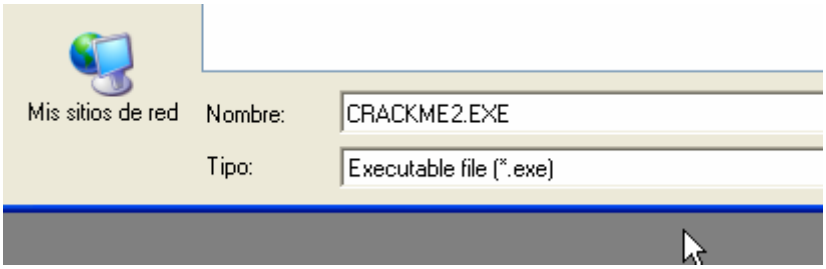



Se nos abre otra ventana, allí hacemos nuevamente CLICK DERECHO-SAVE FILE



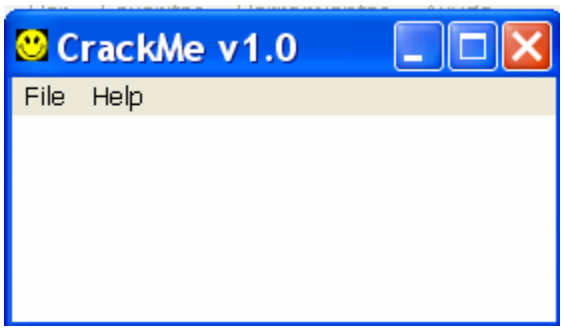



Lo guardamos con OTRO NOMBRE para tener el original para seguir practicando le pondré CRACKME 2

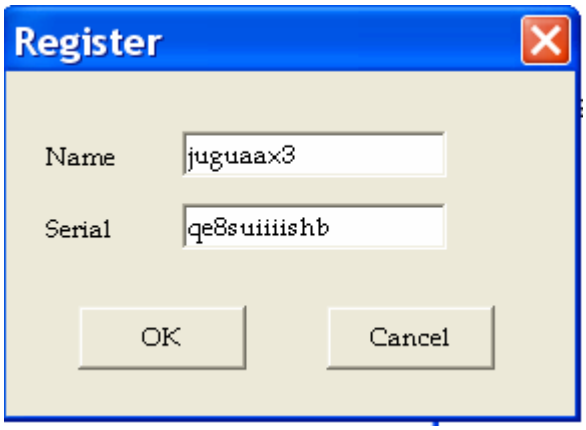



Cierro el OLLYDBG y veo que al lado de donde tenia el crackme esta el crackme2

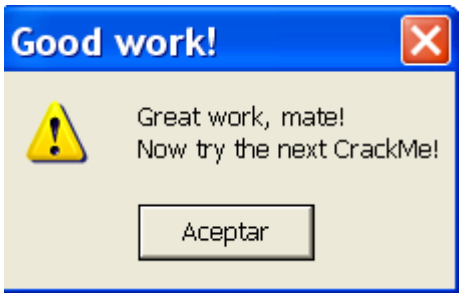

Hago doble click en el crackme2 a ver si quedo bien modificado lo corro sin OLLYDBG.




Voy REGISTER



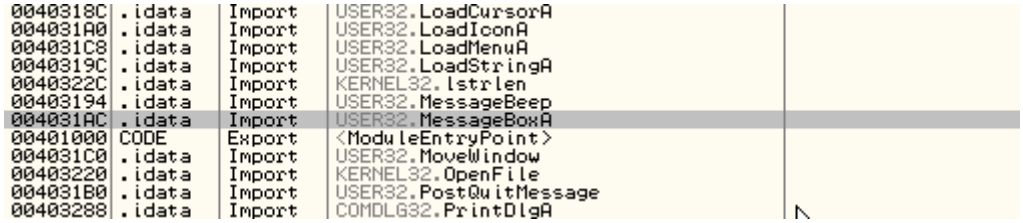

Apreto OK



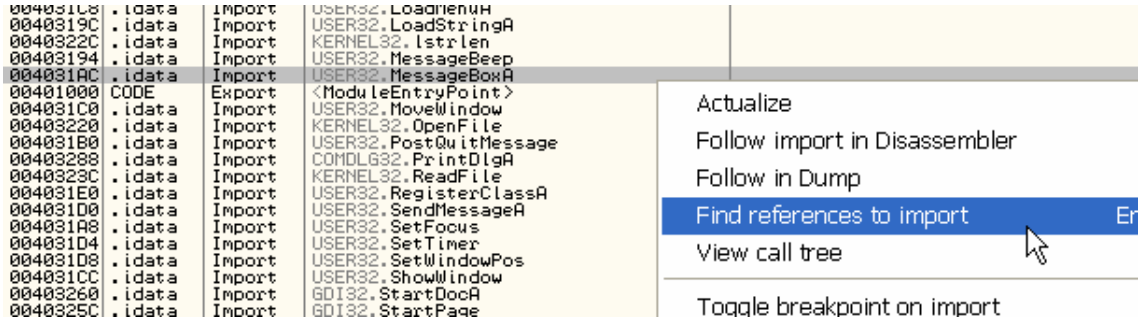

Jeje ahí tengo el crackme parcheado pero aun no me conformare con esto, mas adelante veremos como trabaja con los seriales y como compara si en el nombre hay números, exhaustivamente y hallaremos seriales correctos para nuestro nombre que los acepte sin parchear pero para llegar a eso aun faltan algunos conocimientos y practicas anteriores.

## APÉNDICE PARA WINDOWS 98

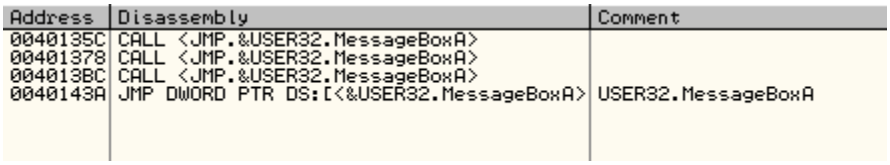
Bueno usar OLLYDBG en w98 como ya les dije es bastante limitado, el mismo sistema no nos deja poner BREAKPOINTS en las apis directamente como en w98, por lo tanto el método será similar con la única limitación que ustedes no podrán poner BREAKPOINT en la api, si hacen CLICK DERECHO SEARCH FOR NAME (labels) in this module como dice allí la primera parte de la explicación para XP




Les saldrá la misma lista y de la misma forma apretando el nombre de la api llegaran hasta el nombre de la misma, pero en 98 no pueden hacer CLICK DERECHO-TOGGLE BREAKPOINT ON IMPORT ya que no permite poner BREAKPOINTS en las apis, si no que deberán hacer.

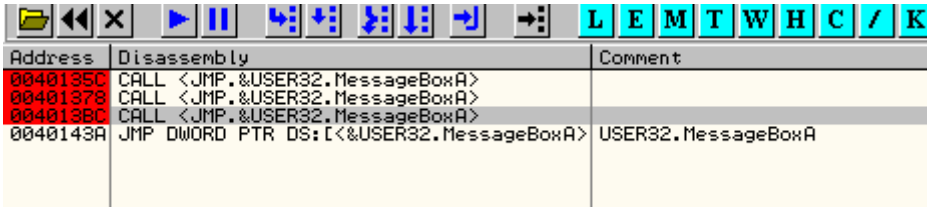



Creo que el menú es similar si no siempre tendrá la opción para BUSCAR LAS REFERENCIAS o sea los llamados a la api en el crackme.

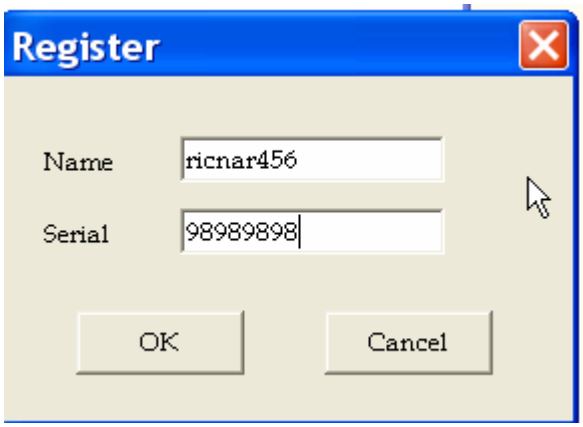



Esto es limitadísimo comparado con la posibilidad de poner un BREAKPOINT en la API, porque si la API es llamada, si hay un BP en la API parara, en cambio en 98, la API puede ser llamada en alguna forma que engañe el análisis del OLLYDBG (y hay muchas formas créanlo) y no saldrá dicha llamada entre las referencias y no tendrán forma de saber de donde fue llamada la api ni parara.

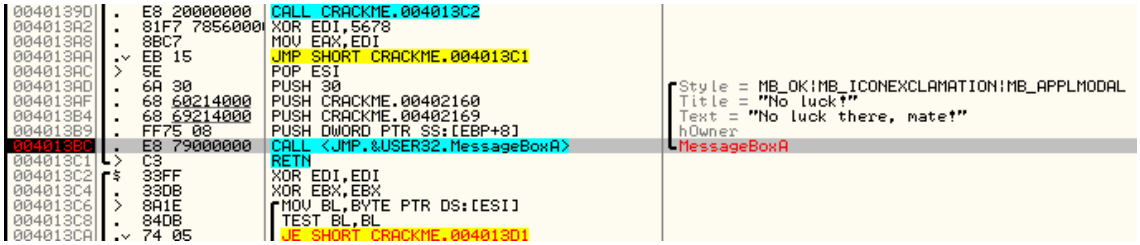
Igual pueden seguir este tute de la misma forma sabiendo que hay tres llamadas a la api MessageboxA y en vez de poner UN BREAKPOINT en la api, pongo un BP en cada referencia a la api, allí en el cuadro de referencias apreto F2 en cada una,



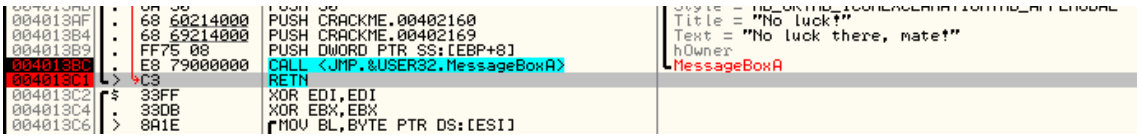

Al dar Run voy a HELP-REGISTER tipeo estos datos




Y al apretar OK parara




Luego en XP ponemos un BP en el RET de la api o donde termina la misma, eso equivale en 98 a poner un BP en el RET que esta justo debajo ya que no podemos entrar a la api.




Pues ya saben cuando el el tute dice que esta en el inicio de la api y ve a donde retorna ustedes saben que retorna en este caso a 4013C1 al ret que esta justo abajo y si es en otra referencia sera el RET que esta debajo correspondiente.

Sabiendo esto ya pueden retornar a

RETORNO DEL APÉNDICE PARA WINDOWS 98 desde aquí sigue para todos los SO.

Y continuar el tute desde allí siempre sabiendo la diferencia principal que es que en W98 no podemos poner BPX o BREAKPOINTS en las apis si no en las referencias.

Hasta la parte 10
Ricardo Narvaja
22 de noviembre de 2005