INTRODUCCIÓN AL CRACKING EN OLLYDBG PARTE 10

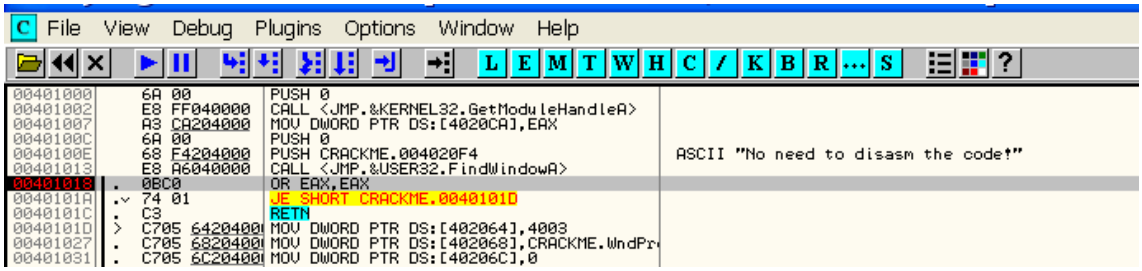
BREAKPOINTS EN OLLYDBG

Esta parte la dedicaremos a entender los diferentes tipos de BREAKPOINTS en OLLYDBG, entendiendo por BREAKPOINTS las herramientas que trae OLLY para ayudarnos a detener la ejecución del programa en donde nosotros deseamos, para los cual usamos uno de los diferentes tipos de Breakpoint que estudiaremos a continuación, para lo cual usaremos como victima el inefable Crackme de Cruehead.

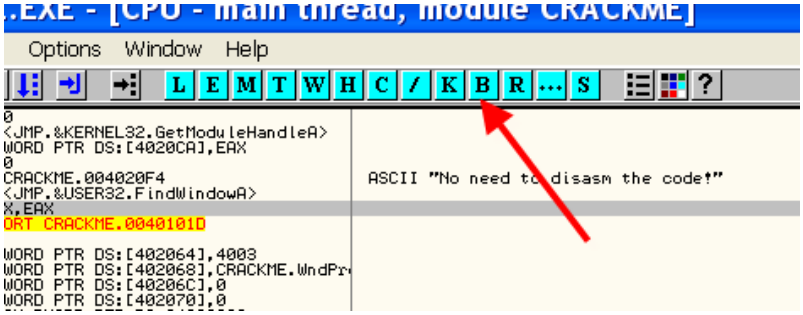
BREAKPOINT COMUN o BPX

Es el Breakpoint común que hemos utilizado hasta ahora, también llamado BPX por su similitud a la forma de tipearlo en SOFTICE, muchas veces se utiliza la sigla BPX como abreviatura de BREAKPOINT, aunque estrictamente en OLLYDBG se tipea BP.
Normalmente se puede colocar marcando la línea que queremos ponerle el BPX y apretando F2, si apretamos F2 nuevamente se quitara.

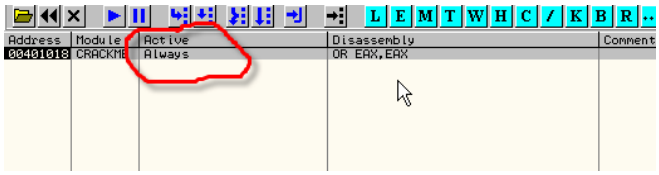
Miremos el CRACKME DE CRUEHEAD en el ENTRY POINT



Si marco por ejemplo 401018 y apreto F2, pues se marca en rojo la dirección para señalarnos que hay un BPX y además en la ventana B que es la lista de BPX que hemos colocado, se nos mostrara.

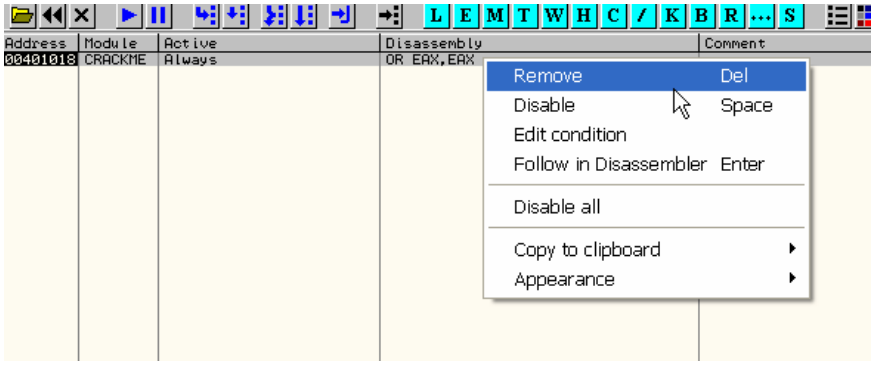


Vemos que nos muestra que esta activo



En la columna ACTIVE nos dice que ALWAYS que significa que parara SIEMPRE que pase por allí o sea que esta activo.

Si hacemos Click derecho sobre el BPX podemos ver las opciones que tenemos para manejar el BREAKPOINT



REMOVE: para quitarlo de la lista totalmente

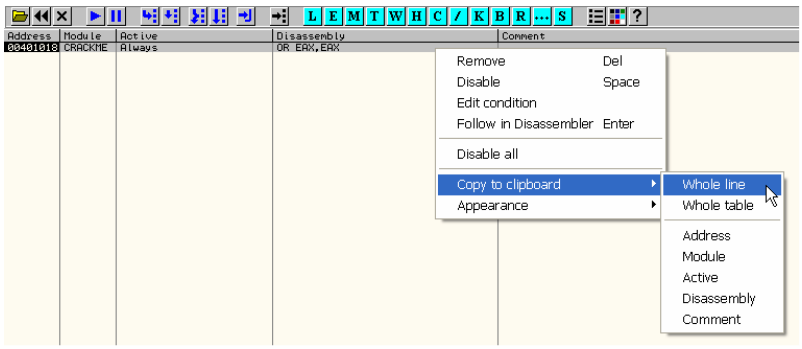
DISABLE: Para que quede en la lista de BREAKPOINTS pero deshabilitado, o sea no parara allí cuando pase por esa dirección.

EDIT CONDITION: Para transformarlo en un BREAKPOINT CONDICIONAL que mas adelante ya veremos que es.

FOLLOW IN DISASSEMBLER: Para buscar la dirección en el listado del breakpoint que marcamos.

DISABLE ALL o ENABLE ALL: Deshabilitar o habilitar todos, en este caso habilitar todos no aparece porque el único existente esta habilitado.

COPY TO CLIPBOARD: para copiar al portapapeles los datos sobre el BPX marcado, si elegimos esta opción y pegamos aquí por ejemplo.



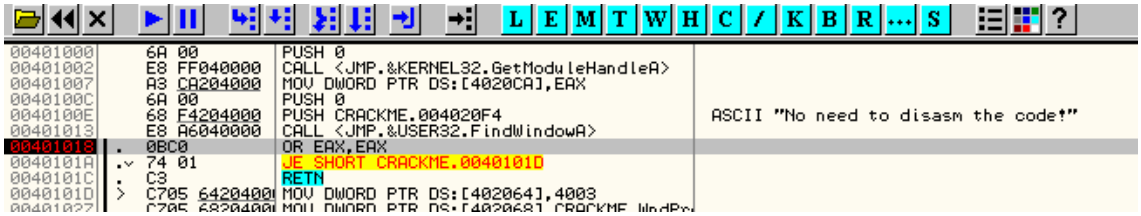
Elijo WHOLE LINE o sea que copie toda la linea, WHOLE TABLE copiara toda la lista de BREAKPOINTS.

Breakpoints, item 0
 Address=00401018
 Module=CRACKME
 Active=Always
 Disassembly=OR EAX,EAX

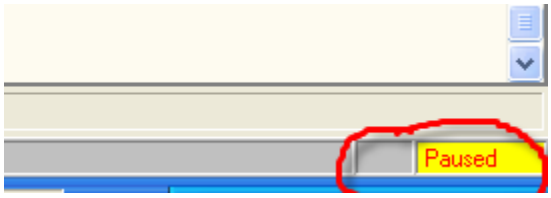
Lo copiado y pegado aquí muestra los datos de ese BREAKPOINT como la instrucción donde fue colocado, su dirección, si esta activo etc.

Ya vimos que si doy RUN con F9, parara en la dirección del BPX si se ejecuta, en este caso para.

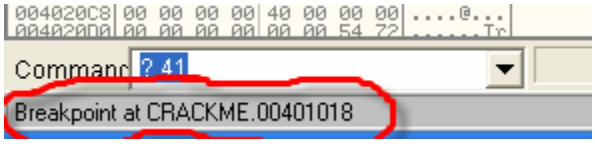
Allí paro



Y OLLY nos dice que esta PAUSADO

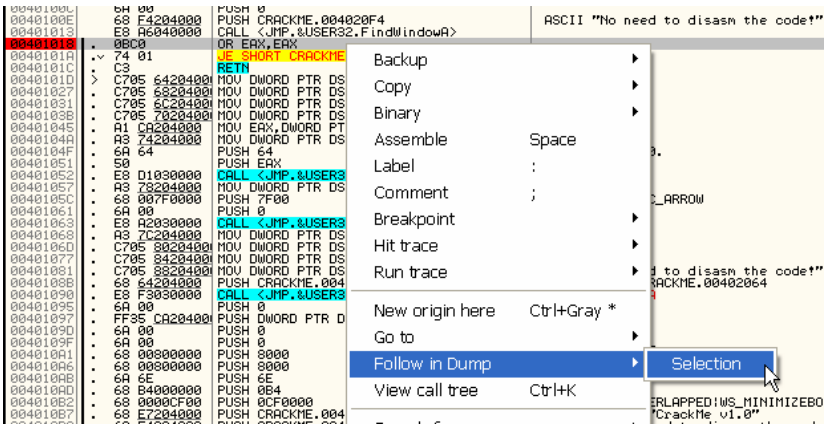


Y a la izquierda nos muestra el motivo porque paro

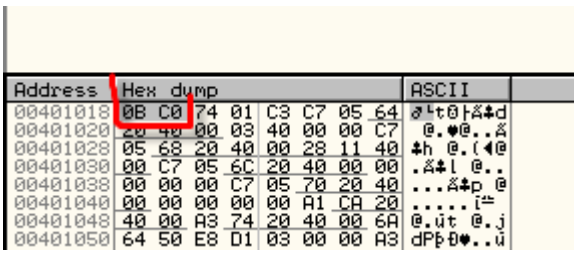


Ahora estrictamente el BPX que es ?, veamos si realiza OLLYDBG algun cambio en el codigo al activarlo.

Si hago click derecho FOLLOW IN DUMP-SELECTION



Podremos ver la direccion 401018 y su contenido en el DUMP



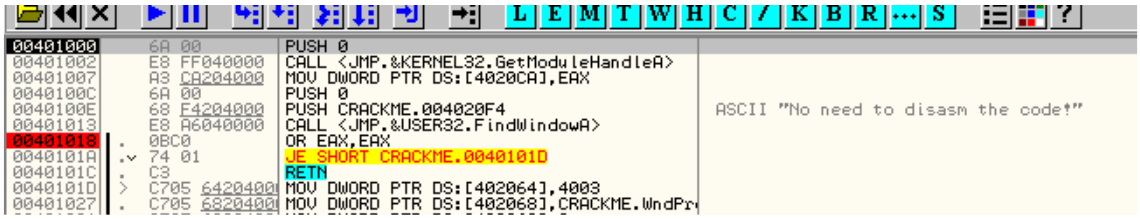
Aparentemente no hay cambios con respecto a los que vemos en el listado

Allí vemos tanto en el DUMP como en el listado los bytes 0B C0 correspondientes a la instrucción

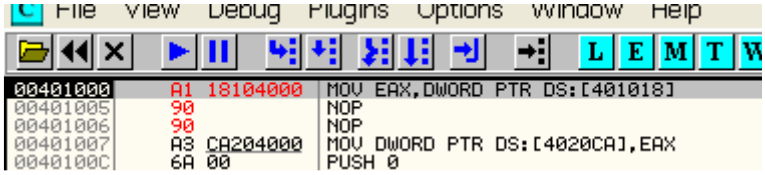
OR EAX,EAX

asi que no hay cambios parece pero es así realmente?

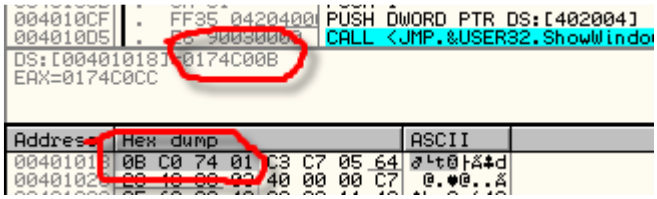
Reiniciemos el crackme y veamos que el BPX continúe puesto en 401018



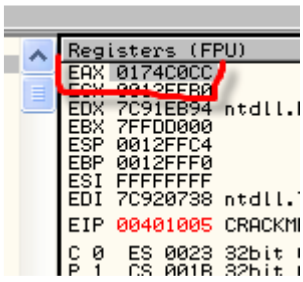
Voy a escribir una línea que leerá de la memoria realmente el valor que hay allí.



Eso leerá el contenido de 401018 y lo moverá a EAX, veamos que dice la aclaración de OLLY



Tanto en el DUMP como en la aclaración nos muestra 0B C0 los bytes originales, pero aun desconfío, apreto F7 para ver que mueve a EAX.



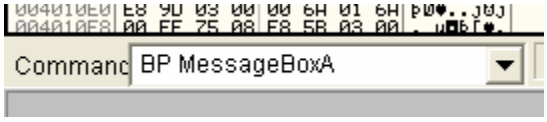
Como no era que en 401018 estaban los bytes 0B C0 74 01 ?, alguien esta mintiendo aquí pues esos bytes al revés son 0174C00B y el valor que movió a EAX es 0174C0CC, o sea que realmente en 401018 no hay un B0 cuando pongo un BREAKPOINT, OLLYDBG lo reemplaza por el valor CC que mas adelante cuando veamos el estudio de las excepciones, veremos bien que significa, pero OJO a pesar de que OLLYDBG para no ensuciar el listado original, no lo cambia ni en el DUMP, ni en las aclaraciones, cada vez que ponemos un BP estamos colocando el byte 0CC en la dirección del BPX y eso como ven puede ser fácilmente detectado por un programa

que verifique si en esa dirección hay un CC en vez del código original, y de esa forma detecta que hay un debugger y puede evitar que corra el programa saltando si encuentra un CC a la salida del mismo.
Así que no se confíen, recuerden siempre que si apretamos F2 cambiamos código aunque OLLY mantenga todo igual y el programa continúe, dicho BREAKPOINT puede ser detectado, si en algún programa ven que al colocarle un BREAKPOINT no corre pues quítenlo hay otras posibilidades en OLLYDBG para parar donde queremos.

También un BREAKPOINT se puede tipear en la commandbar

BP 401018

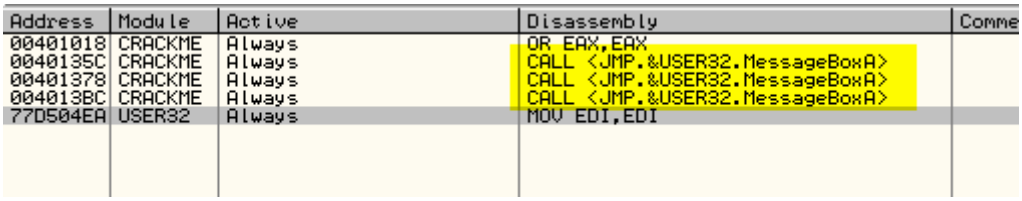
En NT, 2000, XP y 2003 no hay problemas para colocar BP en las apis como vimos en partes anteriores podemos poner un BP en la api MessageBoxA, tipeando



Lo escribimos respetando las mayúsculas y minúsculas del nombre de la api, en Windows 98 en cambio ya que no se puede colocar breakpoint en las apis se tipea.

Lo cual nos pone BREAKPOINTS en las referencias o llamadas que el programa haga a la api y que OLLYDBG pueda detectar, lo cual no es muy bueno, pero en 98 no hay otra posibilidad.
Por supuesto el comando BPX existe también en XP aunque no es usado prácticamente, porque siempre es mas poderoso colocar un BP en la misma api que en las llamadas a la misma que OLLYDBG pueda detectar.

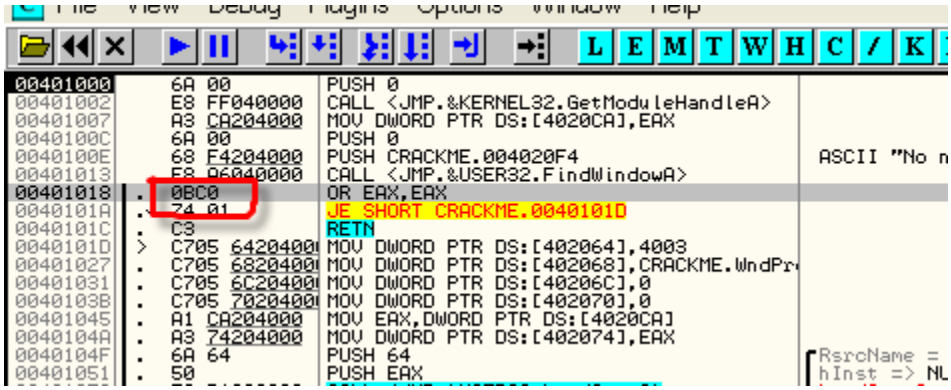
Igual vemos que coloco algunas



Allí en la lista de BREAKPOINTS al apretar BPX MessageBoxA encontró tres llamadas a dicha api y le puso un BP a cada una.

De cualquier forma aclaro que este comando no lo usare casi nunca al no trabajar en WINDOWS 98, y siempre que me refiera a un BREAKPOINT o BPX me refiero a tipear BP en la comandbar.

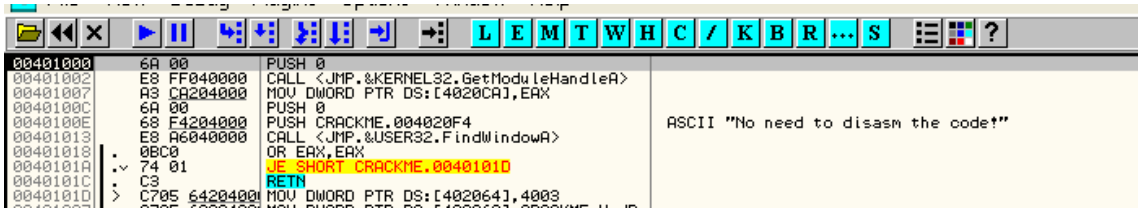
La ultima forma de colocar un BP es con el Mouse haciendo doble click en la línea que queremos colocarlo, en la columna donde están los bytes de la instrucción, al hacer doble click nuevamente se quita.



BREAKPOINTS ON MEMORY (MEMORY BREAKPOINTS O BREAKPOINTS EN MEMORIA)

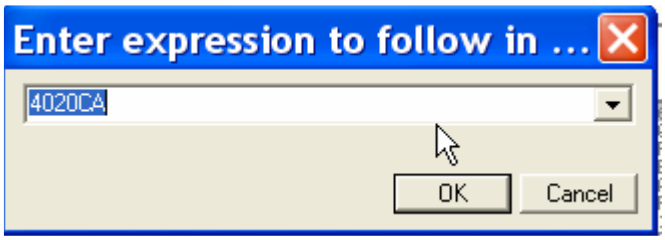
Bueno pasemos a los MEMORY BREAKPOINTS o también llamados BPM por BREAKPOINT ON MEMORY (algunos viejos memoriosos no confundir con los BPM del SOFTICE que son otra cosa)

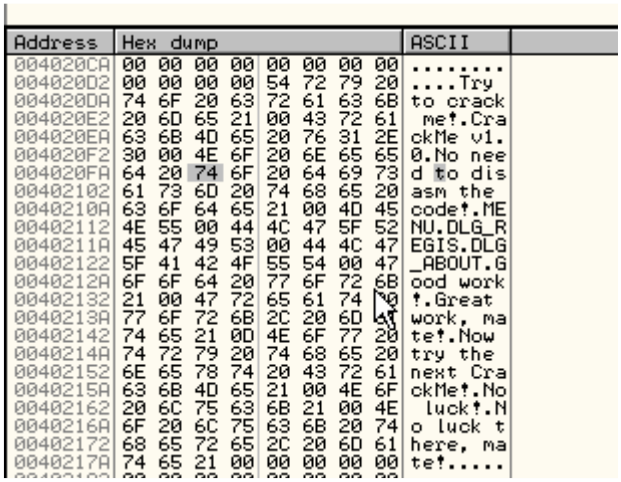
Bueno como funcionan,al colocarlos OLLYDBG lo que hace es cambiarle el permiso de una sección o parte de la misma, según el permiso que le demos, podemos colocar BPM ON ACCESS que detendrá la ejecución, cuando se acceda al sector al cual le colocamos el BPM, o sea parara cuando ejecute esos bytes o cuando lea o escriba en los mismos.
También existe el BPM ON WRITE que para solo cuando escribe en la zona que colocamos el BPM, veremos algunos ejemplos prácticos.



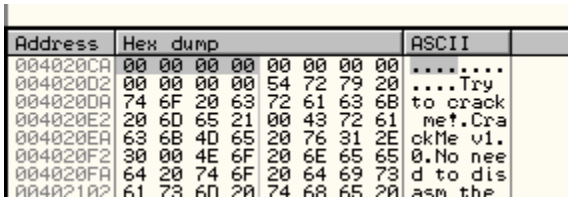
Allí estoy en el ENTRY POINT del Crackme de Cruehead, y pondré distintos BPM, para ver la utilidad de los mismos.

En el DUMP voy a ver la dirección 4020CA con GO TO EXPRESIÓN : 4020CA

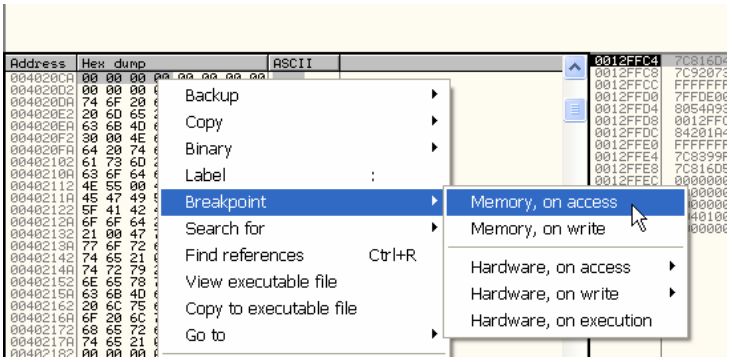




Allí la veo, puedo colocar un BPM ON ACCESS en los 4 bytes del contenido de la dirección 4020CA, y dar RUN, supuestamente parara o bien cuando lea de 4020CA, o cuando escriba en 4020CA o si ejecuta alguna instrucción en 4020CA, para colocar el BPM marco los bytes que quiero que abarque en este caso marco cuatro.

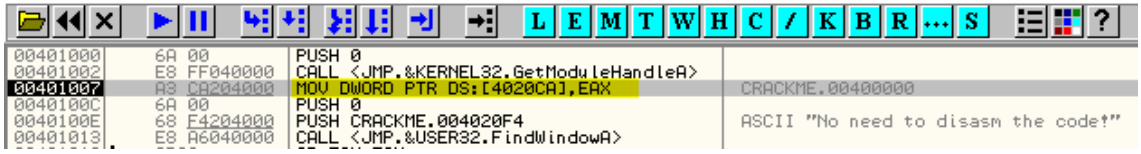


Ahí están marcados, hago click derecho BREAKPOINT-MEMORY ON ACCESS que es poner un BPM ON ACCESS en los bytes que marque, que en este caso son 4 pero podrían ser mas o menos según lo que nos convenga, o sea podemos marcar zonas mas grandes, y colocarles BPM en la misma forma.

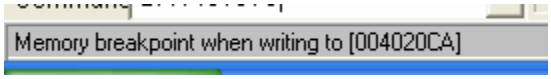


Lo único molesto que tienen los BPM es que no figuran en la ventana B de breakpoints ni en ninguna parte en OLLY, por lo cual debemos recordar donde lo colocamos.
A su vez OLLY permite solo un BPM a la vez por lo cual si colocamos un segundo, automáticamente borra el anterior.

Bueno demos RUN y para en 401007 donde el programa trata de escribir en nuestra zona del BPM o sea en 4020CA.

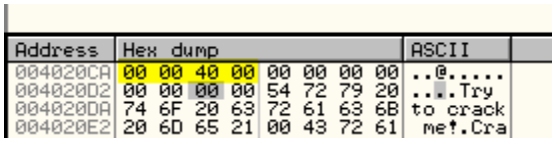


Si vemos el motivo por cual paro, abajo en OLLYDBG nos dice

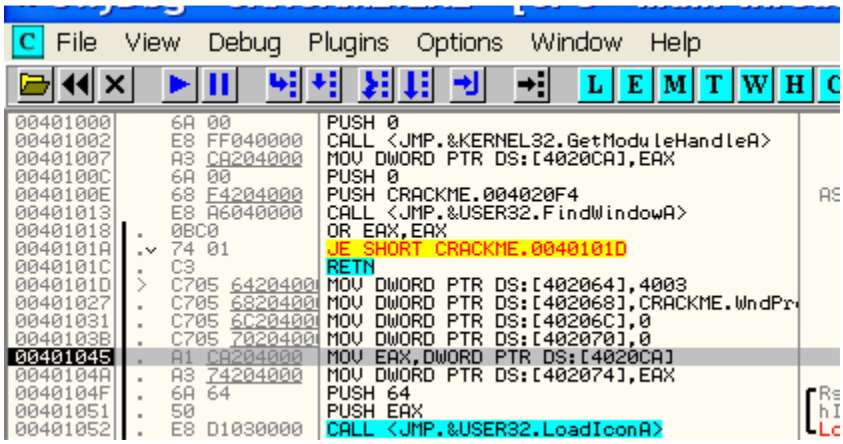


Esta por escribir el valor que esta en EAX al contenido de 4020CA, lo cual provoca que OLLYDBG pare, si recuerdan cuando tratábamos de escribir en la memoria y esta no tenia permiso de escritura, se generaba una excepción, pues eso es lo que básicamente hace OLLYDBG, al colocar un BPM ON ACCESS le quita el permiso de lectura y de escritura a esa zona, y al tratar de escribir o leer de allí, genera una excepción que detiene el programa, ya lo veremos mas claramente cuando veamos el capitulo de excepciones, pero es bueno que tengan una idea.

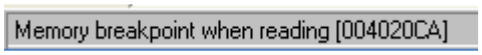
Si apreto F7 ahora si guardara el valor ya que el mecanismo del OLLY es sofisticado y permite que una vez detenido, si se pueda escribir al apretar f7.



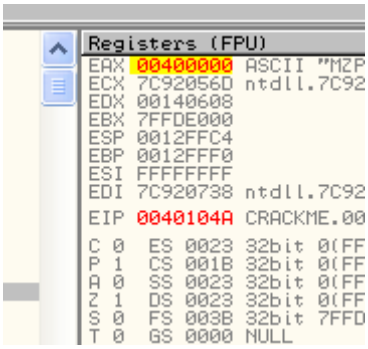
Allí esta guardo 400000 si doy RUN nuevamente como el BPM esta aun activo, parara si el programa vuelve a escribir o intenta leer este valor.



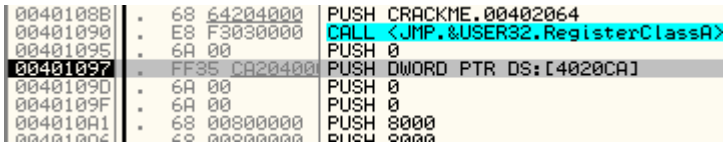
Vemos que ahora para allí, al tratar de leer de 4020CA en este caso nos dice MEMORY BREAKPOINT ON READING ya que paro al leer de allí.



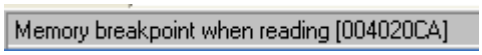
Si apreto F7 lee el valor guardado y lo pasa a EAX



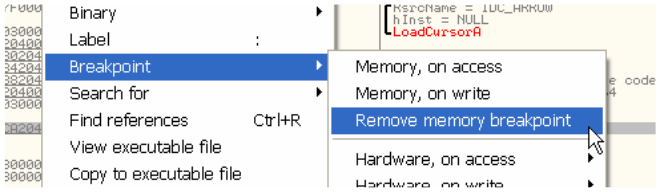
Apreto F9 a ver si alguna vez mas para.



Nuevamente para ON READING ya que lee el valor que hay en 4020CA y lo manda al stack con PUSH.



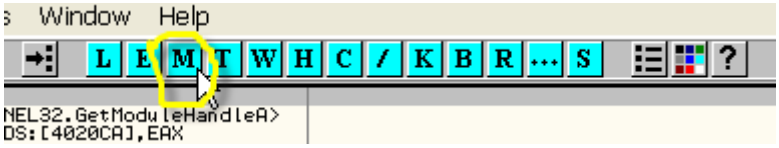
Ahora si quiero quitar el BPM que coloque hago click derecho en el DUMP y elijo BREAKPOINT-REMOVE MEMORY BREAKPOINT y con eso se quita, lo mismo como ya dijimos si colocamos uno nuevo.



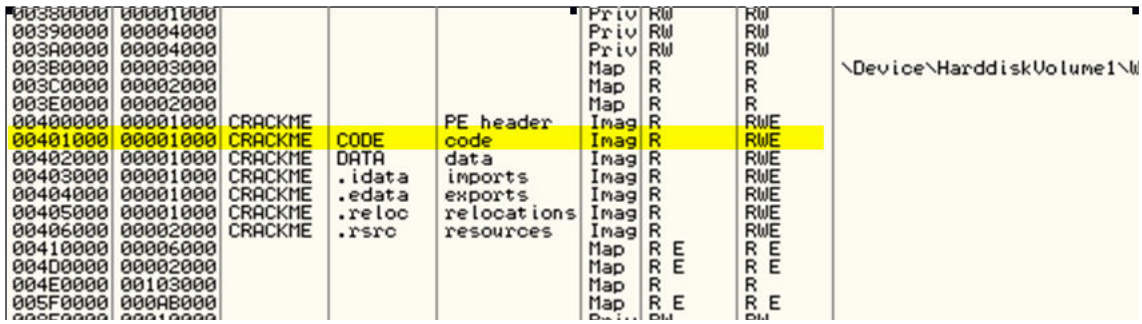
En el caso anterior si al colocar el BREAKPOINT, hubiéramos elegido que sea ON WRITE o sea en escritura, OLLYDBG parara solo cuando escriba en la zona del BPM y no cuando lee.

Otra opción que nos da OLLYDBG es colocar un BPM en una sección completa.

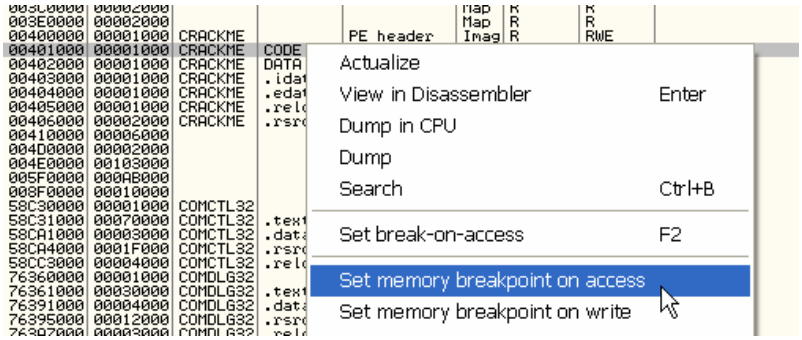
Para ellos vamos a VIEW-MEMORY o a la ventana M que es lo mismo.



Allí vemos las secciones del crackme y mas abajo las secciones de las dlls y diferentes secciones utilizadas, podemos marcar la que deseemos por ejemplo, la sección que comienza en 401000 allí la vemos.

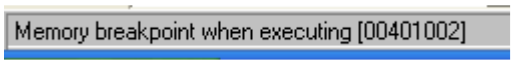


Hago click derecho en dicha sección y elijo SET MEMORY BREAKPOINT ON ACCESS.

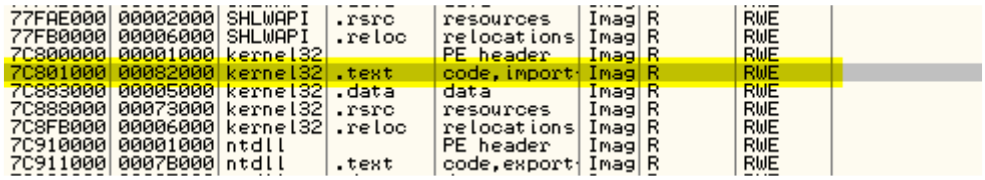


Vemos que justo debajo tenemos la posibilidad de colocar también un BPM ON WRITE pero en este caso lo haremos ON ACCESS doy RUN.

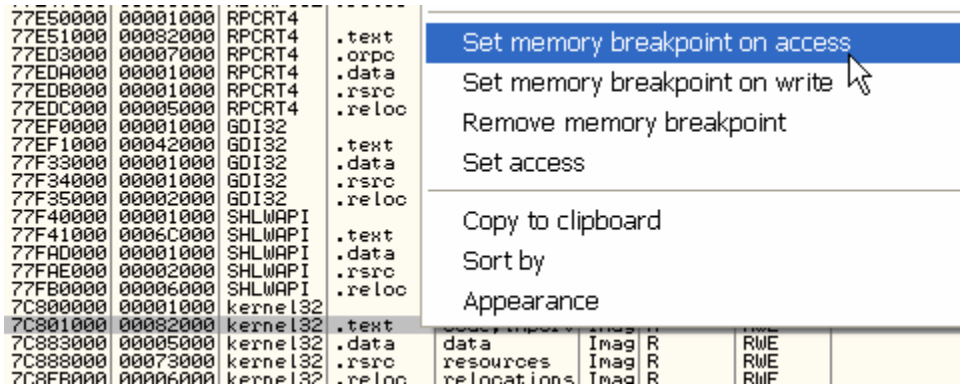
Vemos que para en la siguiente línea veamos porque



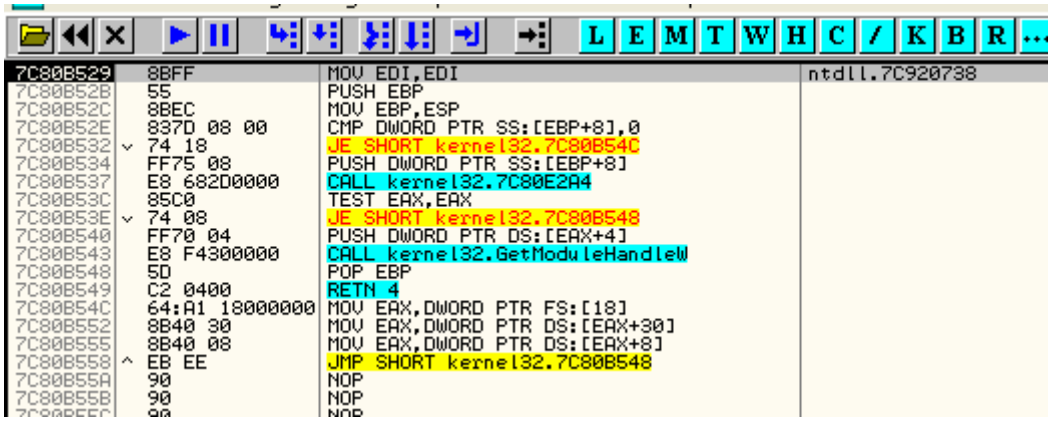
Claro paro ON EXECUTION o sea el ejecutar, ya que pusimos el BPM en la sección que se ejecuta, pues al tratar de ejecutar alguna instrucción en dicha sección, para en este caso ON EXECUTION.



Ahora cambiaremos el BPM a la sección de KERNEL32 que esta a continuación del header, en mi caso es esta.



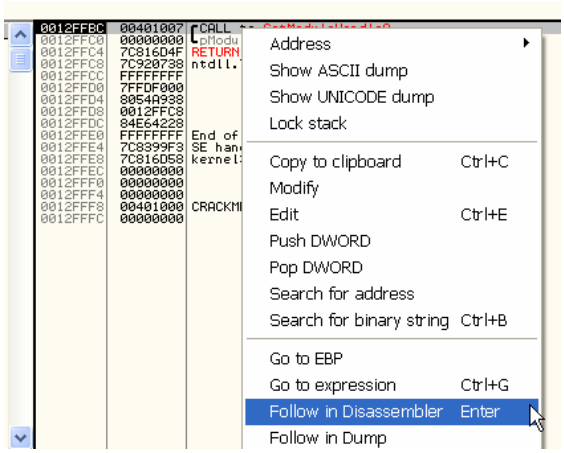
Le coloco un BPM ON ACCESS allí, así que parara cuando lea, escriba allí o cuando ejecute alguna instrucción en esa sección de KERNEL32, jeje doy RUN.



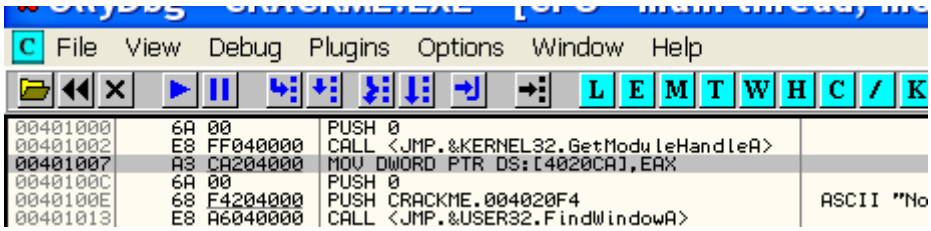
En mi caso para allí y si veo en el stack, veo que paro en una api

En este caso para al ejecutar una instrucción de la kernel32.dll y me funciono para saber cual es la primer api accedida desde el programa en dicha dll, si vemos la dirección de retorno en la primera línea del stack.

Si vemos adonde retornara con click derecho- FOLLOW IN DISASSEMBLER

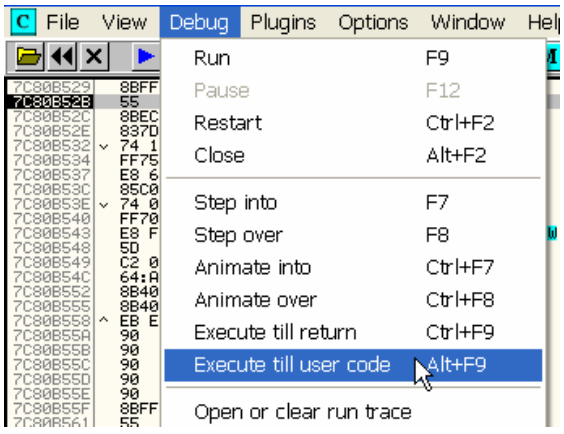


Allí vemos que retornara a 401007 de la llamada a GetModuleHandleA que hay al inicio del programa.



Veo que un poco mas abajo hay una llamada a FindWindowA pero esa no parara pues corresponde a otra dll, en este caso a User32.dll.
Si doy run nuevamente parara en la siguiente línea de la api ya que el BREAKPOINT ON ACCESS me hace parar en cada línea de la misma que se esta ejecutando, pues entonces lo quitare con REMOVE MEMORY BREAKPOINT como antes.

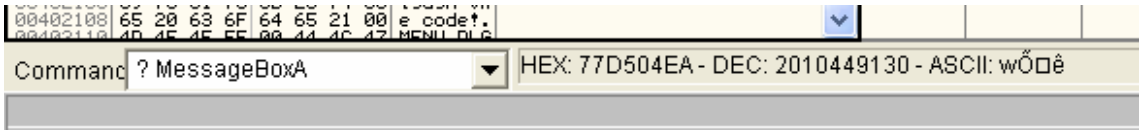
Si quiero volver al programa, hago click derecho EXECUTE TILL USER CODE, o si no funciona en algún caso, puedo usar allí mismo EXECUTE TILL RETURN lo que me lleva hasta el RET y luego apreto F7 y vuelvo al programa.



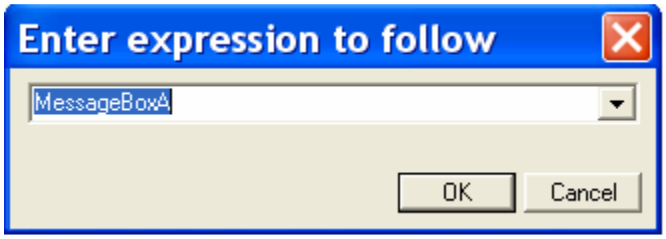
En este caso EXECUTE TILL USER CODE funciona perfectamente ya veremos en que casos no funciona.

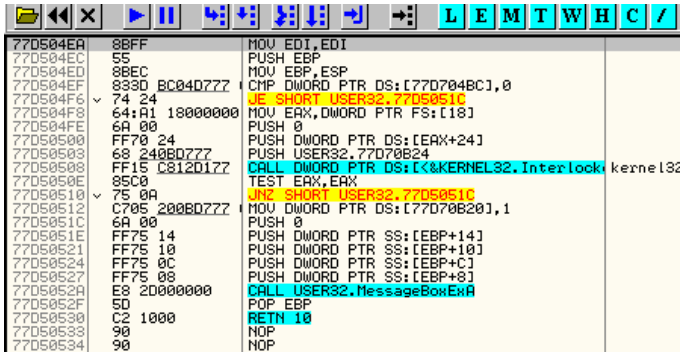
Si pongo un BPM de nuevo en la kernel32.dll parara en la siguiente api que se ejecute de esa dll y asi.

También si no puedo colocar porque es detectado un BP MessageBoxA porque el programa detecta el CC como vimos antes podemos poner un BPM allí también y cumplirá la misma función, veamos el ejemplo

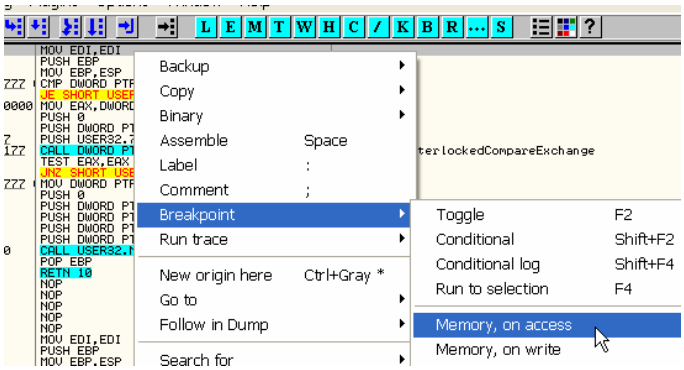


En mi maquina es 77D504EA así que voy allí en el listado con GOTO EXPRESIÓN, puedo tipear la dirección o el nombre siempre respetando mayúsculas y minúsculas.

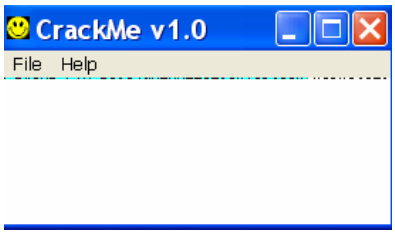




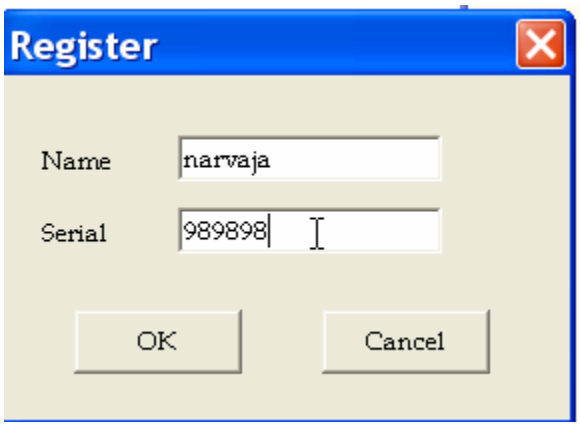
Marco la primera línea



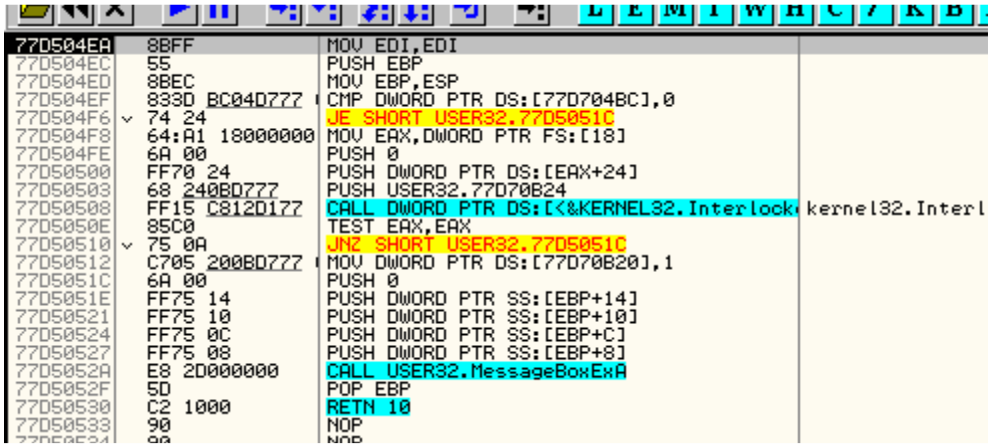
Y veo que también tengo la posibilidad de elegir poner un BPM ON ACCESS o ON WRITE, elijo la primera opción y doy RUN.



Voy a HELP –REGISTER y allí pongo un user y serial cualquiera



Al apretar OK



Para en la misma forma en la api que si hubiéramos puesto un BP allí y si los BP son detectados es otro metodo para parar alli, que aunque puede ser detectado por el programa, mirando el permiso de la zona si cambio, igual es menos probable que ello ocurra, al menos es otra alternativa y hay que conocerlas todas.

Bueno para la parte siguiente nos quedan los HARDWARE BREAKPOINT y los MESSAGE BREAKPOINTS y el caso de los BPX CONDICIONALES que requiere más explicación, creo que por ahora tienen para practicar.

Hasta la parte 11
Ricardo Narvaja
26 de noviembre de 2005