INTRODUCCIÓN AL CRACKING CON OLLYDBG PARTE 11

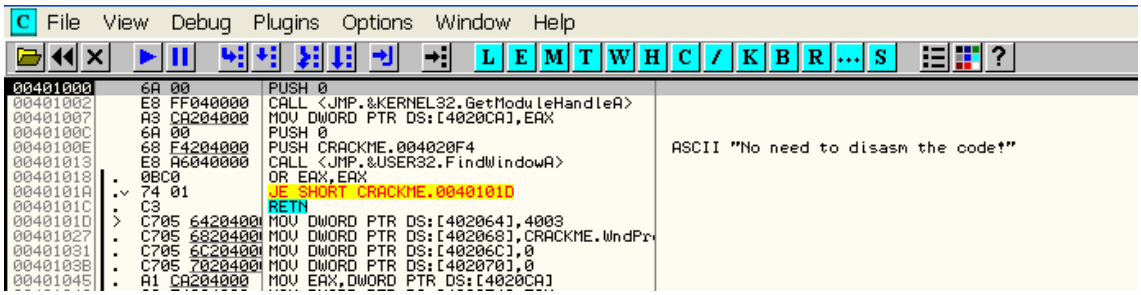
Bueno, ya nos queda ver los hardware breakpoints, los condicionales y los message breakpoints, para terminar con los tipos de breakpoints.

HARDWARE BREAKPOINTS

Los HARDWARE BREAKPOINTS o (HBP) son una propiedad física del procesador, realmente yo no conozco la arquitectura de los mismos para explicarla, pero es como si tuviera varias entradas en las cuales vos le podes poner las direcciones en las cuales querés que el programa se detenga y el al ir ejecutando compara y al hallar la dirección parara.

Tenemos la posibilidad de poner cuatro HBP, a la vez al poner el quinto OLLYDBG nos pedirá que elijamos cual de los 4 existentes queremos borrar.

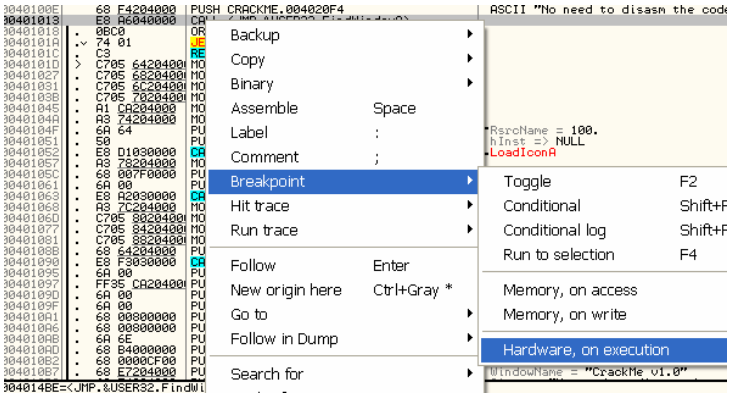
Como siempre usemos el CRACKME DE CRUEHEAD para practicar



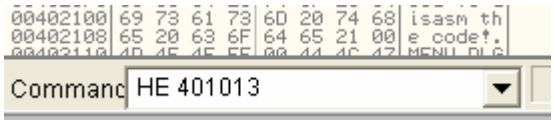
Existen HBP ON EXECUTION, ON WRITE y ON ACCESS son las tres posibilidades existentes.

Poner un HB ON EXECUTION cumple la misma función que poner un BPX en una dirección, salvo que el HBP no cambia código y es mas difícil que sea detectado, eso si hay programas que tienen trucos que nos borran los HBP, esos trucos y la forma de contrarrestarlos los veremos en próximas entregas.

Si quiero poner un HBP EXECUTION en 401013, marco la línea, hago click derecho y elijo BREAKPOINT-HARDWARE ON EXECUTION

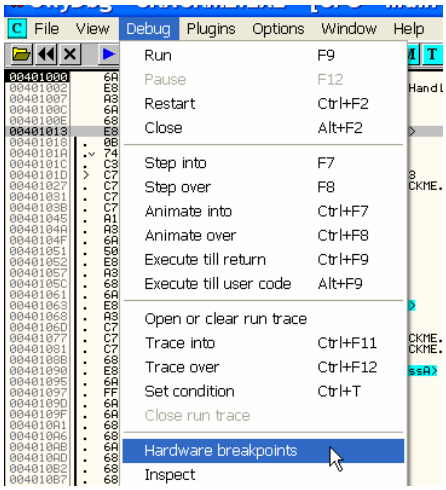


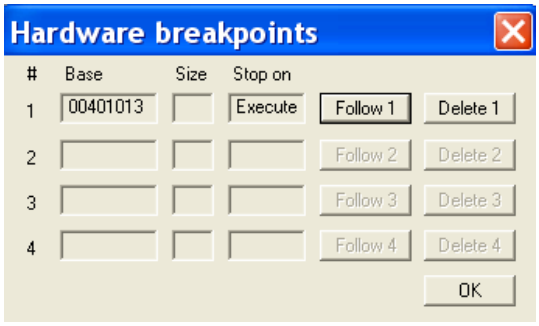
También puedo tipear en la commandbar



Con lo cual el HBP estará puesto.

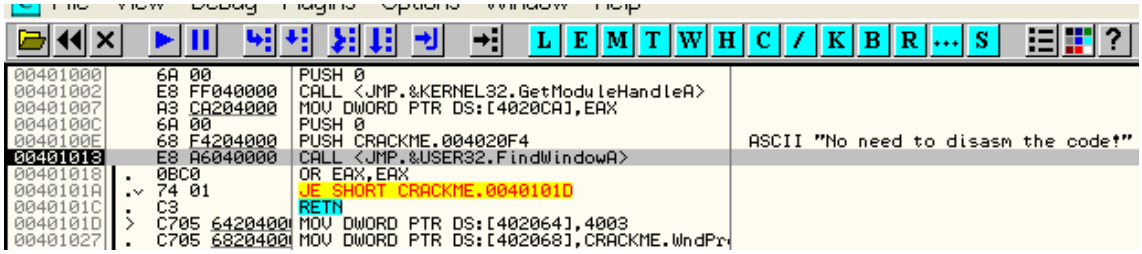
OLLYDBG tiene una ventana especial para que podamos ver y manejar los HBP, para ello vamos a DEBUG-HARDWARE BREAKPOINTS





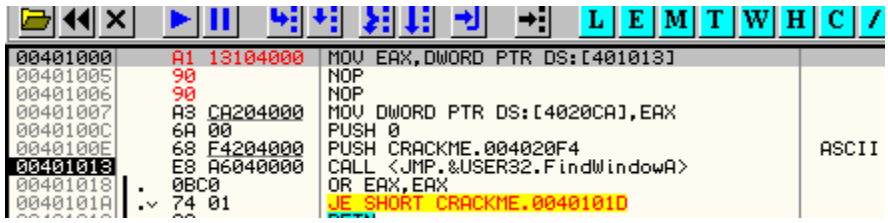
Allí tenemos la ventana de los HBP, si apretamos FOLLOW nos mostrara en el listado donde esta colocado el mismo, y con DELETE podremos borrarlo.

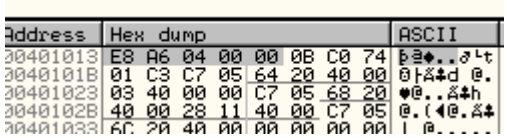
Ahora apreto F9 para correr el programa



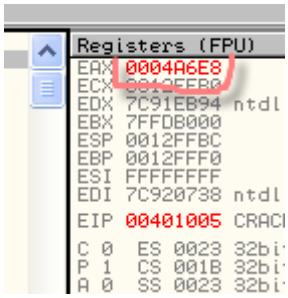
Y para en 401013

Como vemos se comporta como un BPX común, si realizamos la misma prueba que hicimos con el BPX de escribir MOV EAX, DWORD PTR DS:[401013] y lo ejecutamos vemos que el código no ha sido cambiado.



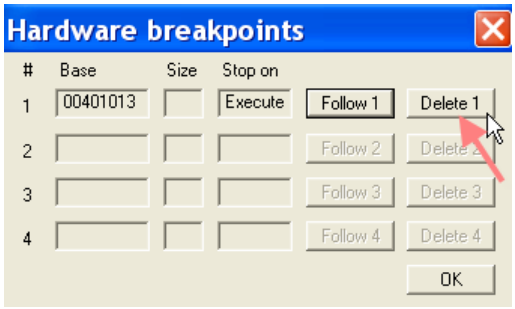


Cambio EIP a 401000 con NEW ORIGIN HERE y apreto F7

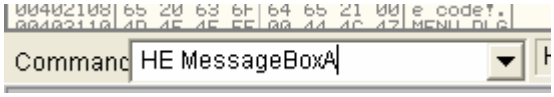


Vemos que los bytes son E8 A6 04 00, los lee al revés y nos muestra 00004A6E8, sin cambios, así que verificamos que no hay cambios de código.

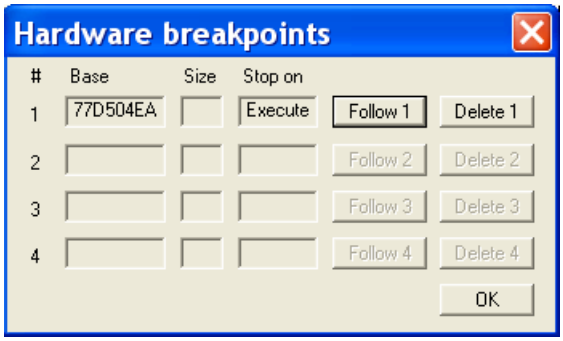
A su vez podemos reiniciar el OLLYDBG veremos que el HBP se mantiene colocado



Lo borramos y colocamos uno en MessageBoxA como hicimos con el BPX directo en la api



Ahora lo vemos en la lista de HBP

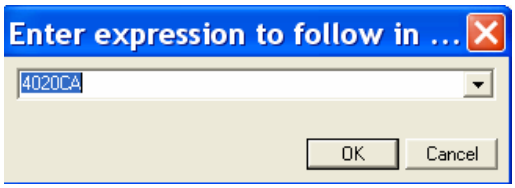


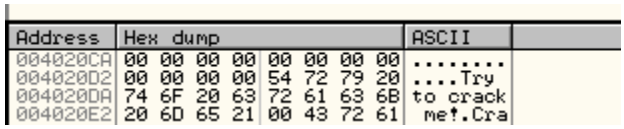
No repetiremos lo mismo pero sabemos que si corremos el programa y le colocamos un user y pass, y lo aceptamos, parara en la api MessageBoxA de la misma forma que lo hemos hecho antes con BPX comunes.

Los HBP ON ACCESS y ON WRITE solo pueden abarcar 1, 2 o 4 bytes de largo, por mas que marquemos zonas en el dump mas largas, solo tomara como máximo los primeros 4 bytes de la misma veamos el ejemplo.

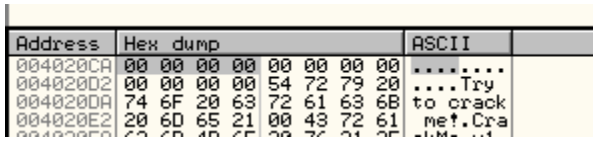
Reinicio y borro todos los HBP que quedaron, y tratare de colocar un HBP ON ACCESS en la dirección 4020CA

Voy a ver dicha dirección en el DUMP

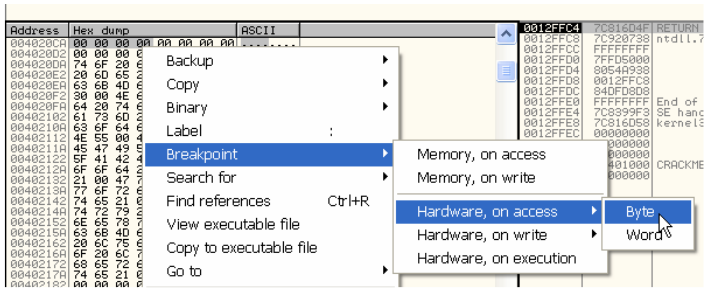




Allí esta ahora marco los primeros 4 bytes

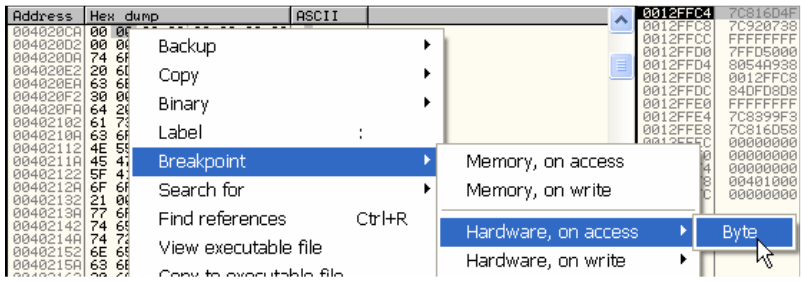


Y hago click derecho

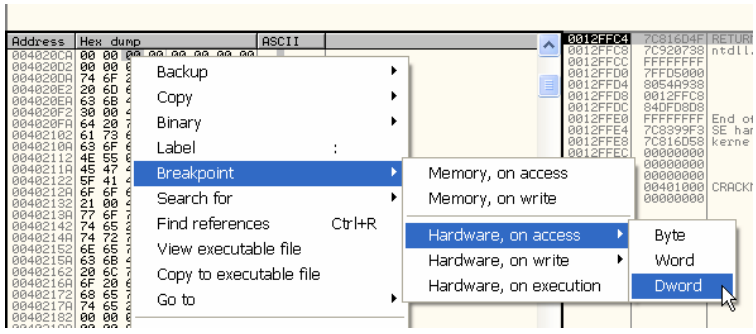


Veo que haya marcado la zona que haya marcado, me aparece solo en este caso la opción de colocarlo en un solo byte a partir de 4020CA o dos bytes (WORD), la opción de 4 bytes va de cuatro en cuatro así que si uno de casualidad elije una dirección intermedia no aparecerá, como en este caso,

Vemos que si hago click derecho en el byte siguiente y elijo HARDWARE ON ACCESS, solo aparece la opción BYTE

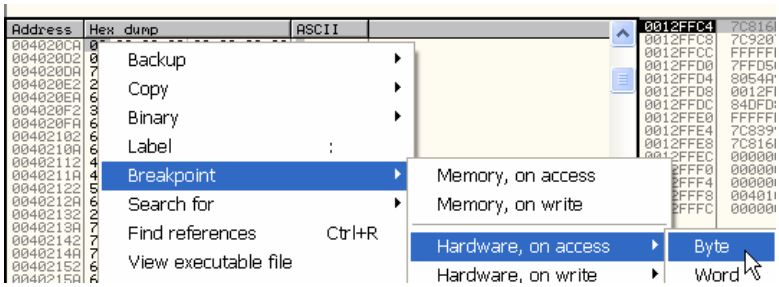


Y en el siguiente ya me aparece nuevamente la opción DWORD o sea que abarque cuatro bytes.

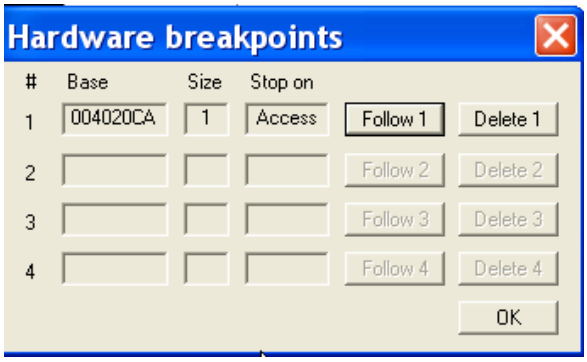


El tema es que si yo quiero que pare cuando guarda o lee en 4020CA, con que haya un HBP en un solo byte es suficiente pues al intentar guardar o leer en la dirección parara igual.

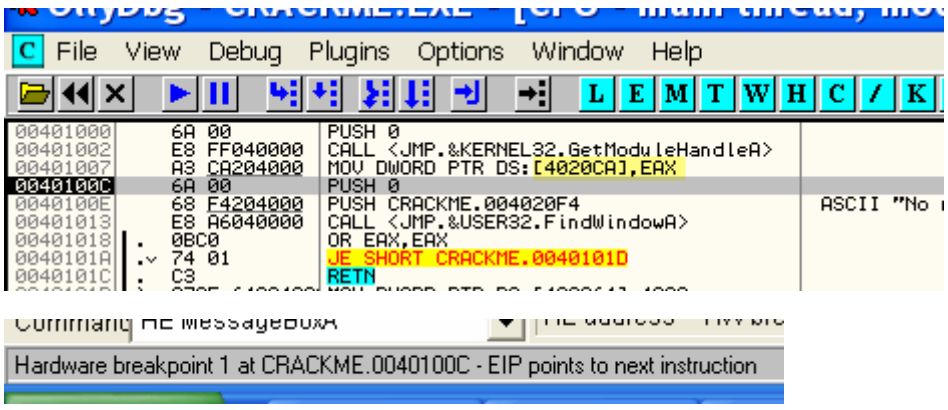
Vuelvo a 4020CA y coloco un HBP ON ACCESS- BYTE



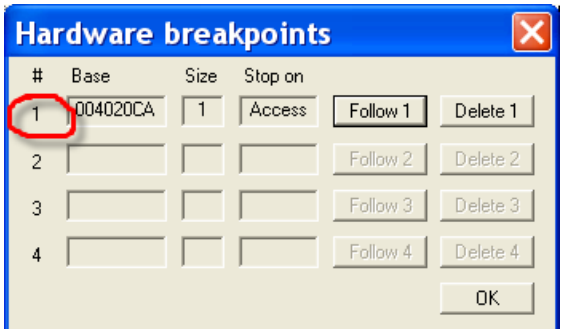
Vemos que en la lista nos marca el tamaño 1 o sea un BYTE



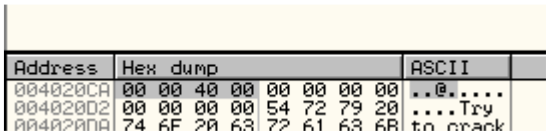
Doy RUN



Allí nos dice OLLY que paro por el HBP 1 o sea el primero de la lista de los HBP.



Veo que para en la línea siguiente a donde guarda o lee, eso ocurre con los HBP ON ACCESS y ON WRITE siempre debemos recordar que se pasan una instrucción y paran en la instrucción siguiente.



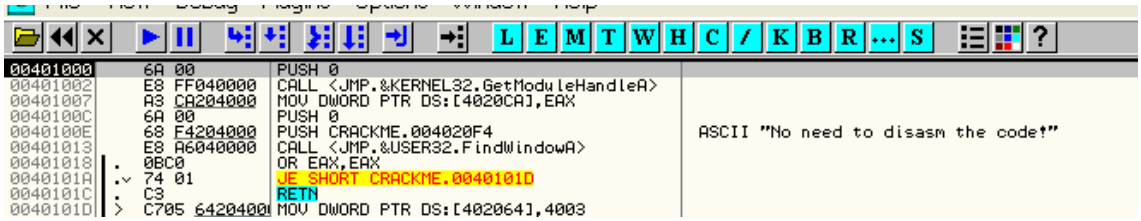
Y para justo después de guardar, a diferencia del MEMORY BREAKPOINT que hubiera parado en 401007, justo en la instrucción que lo activa.

Bueno lo mismo se puede realizar con los HBP ON WRITE, en ese caso solo parara cuando guarda no cuando lea, y en la instrucción siguiente a la que lo activa.

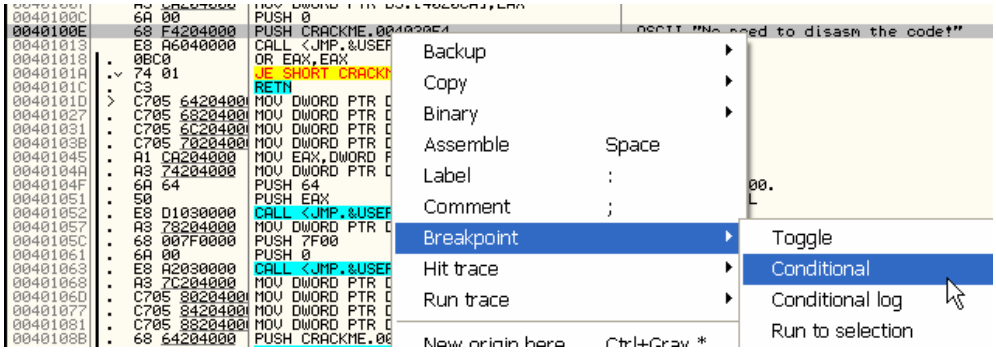
BREAKPOINT CONDICIONALES

Son en realidad una variante de los BPX comunes, solo que al activarse OLLYDBG chequera si alguna condición que le colocamos se cumple y en ese caso parara, si no se cumple seguirá corriendo como si no existiera.

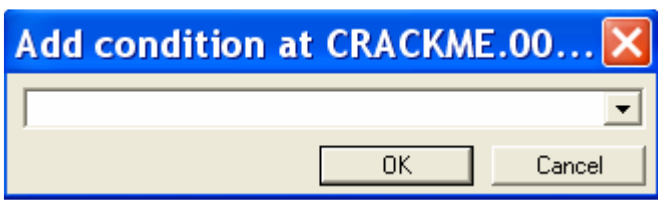
Veamos un ejemplo



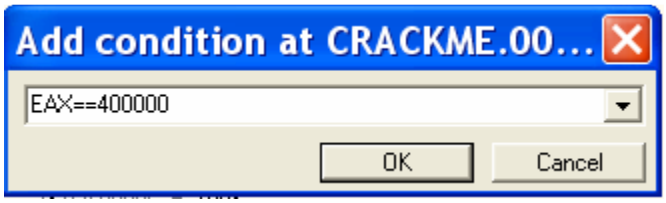
Reiniciamos el OLLY y borramos todos los HBP si quedo alguno del ejemplo anterior y colocare un BPX CONDICIONAL o BREAKPOINT CONDICIONAL en 40100E, para ello marco la línea, hago click derecho y elijo BREAKPOINT CONDICIONAL.



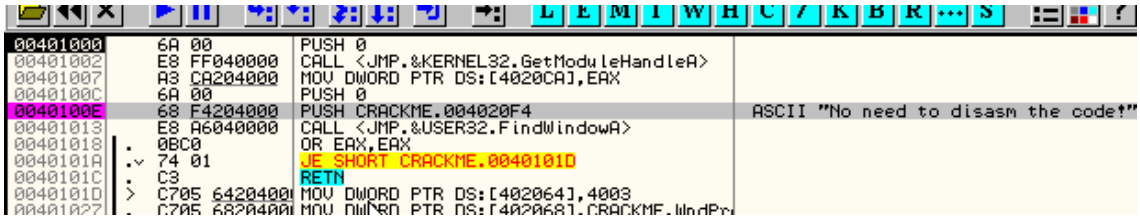
Me sale la ventanita para que escriba la condición



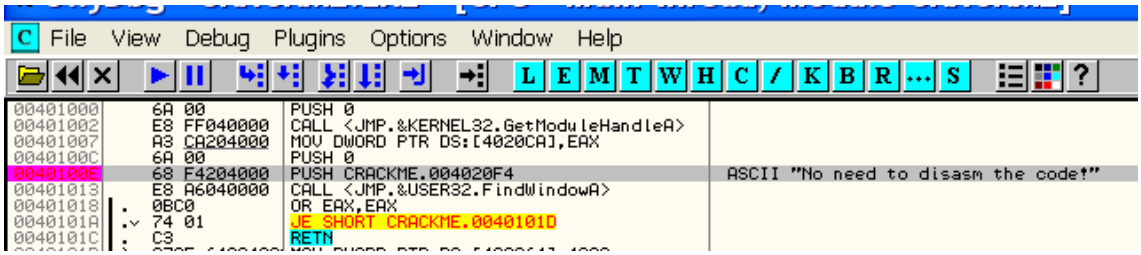
Pondré que pare si EAX vale 400000, o sea la condición será EAX==400000



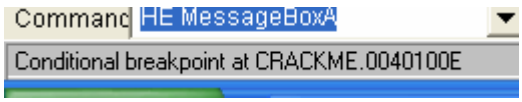
En la ayuda del OLLYDBG nos muestra los símbolos que podemos utilizar, en este caso el igual, se representa por dos símbolos igual consecutivos.



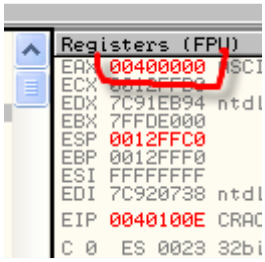
Vemos que en el caso de un BPX CONDICIONAL se cambia a color rosado, apreto F9



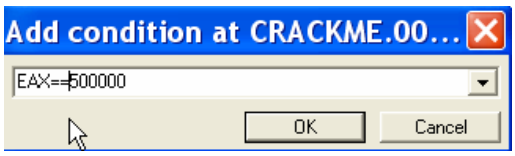
Vemos que paro y OLLYDBG nos dice



Si vemos EAX vemos que vale 400000, lo cual fue la condición para que se detuviese



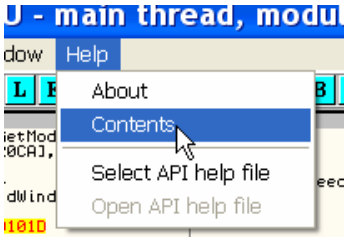
Si reinicio y borro el BPX CONDICIONAL y ahora coloco otro pero que pare si EAX==500000 por ejemplo

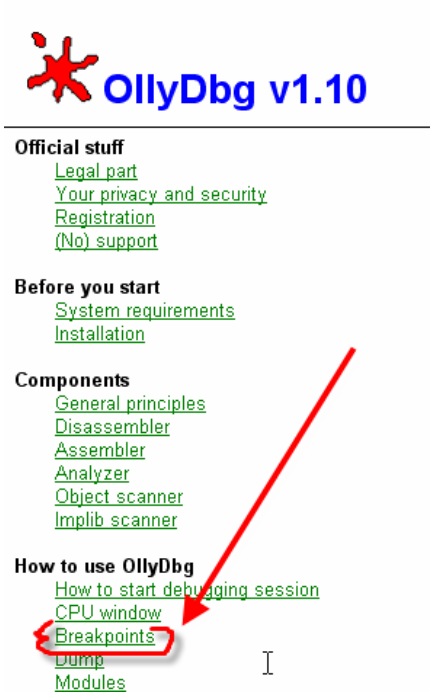


Y doy RUN

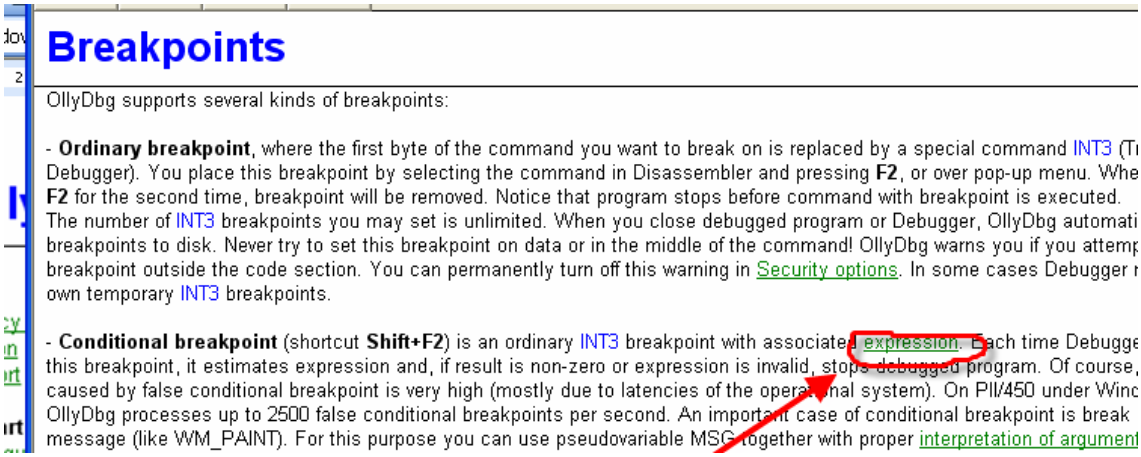
Vemos que el crackme se ejecuta y no para en el BREAKPOINT ya que siempre al pasar por allí, vale EAX=400000 y la condición no es cumplida, por lo tanto no se detiene.

Si vamos a HELP-CONTENTS





Allí a BREAKPOINTS y luego a expression



OLLYDBG nos mostrara ejemplos de expresiones que pueden ser usadas en las condiciones

Evaluation of expressions

OllyDbg supports very complex expressions. Formal grammar of expressions is described at the end of this topic, but hone not interested in it, are you? So I'll begin with examples:

10 - constant 0x10 (unsigned). All integer constants are assumed hexadecimal unless followed by a decimal point;

10. - decimal constant 10 (signed);

'A' - character constant 0x41;

EAX - contents of register EAX, interpreted as unsigned number;

EAX. - contents of register EAX, interpreted as signed number;

[123456] - contents of unsigned doubleword at address 123456. By default, OllyDbg assumes doubleword operands;

DWORD PTR [123456] - same as above. Keyword PTR is optional;

[SIGNED BYTE 123456] - contents of signed byte at address 123456. OllyDbg allows both MASM- and IDEAL-like mei expressions;

STRING [123456] - ASCII zero-terminated string that begins at address 123456. Square brackets are necessary becau the contents of memory;

[[123456]] - doubleword at address that is stored in doubleword at address 123456;

2+3*4 - evaluates to 14. OllyDbg assigns standard C priorities to arithmetical operations;

(2+3)*4 - evaluates to 20. Use parentheses to change the order of operations;

EAX.<0. - 0 if EAX is in range 0..0x7FFFFFFF and 1 otherwise. Notice that constant 0 is also signed. When comparin unsigned, OllyDbg always converts signed operand to unsigned.

EAX<0 - always 0 (false), because unsigned numbers are always positive.

MSG==111 - true if message is WM_COMMAND. 0x0111 is the code for WM_COMMAND. Use of MSG makes sense conditional or conditional logging breakpoint set on call to or entry of known function that processes messages.

[STRING 123456]=="Brown fox" - true if memory starting from address 0x00123456 contains ASCII string "Brown fox FOX JUMPS", "brown fox???" or similar. The comparison is case-insensitive and limited in length to the length of text c

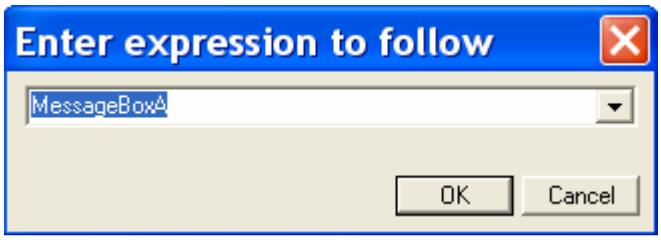
EAX=="Brown fox" - same as above, EAX is treated as a pointer.

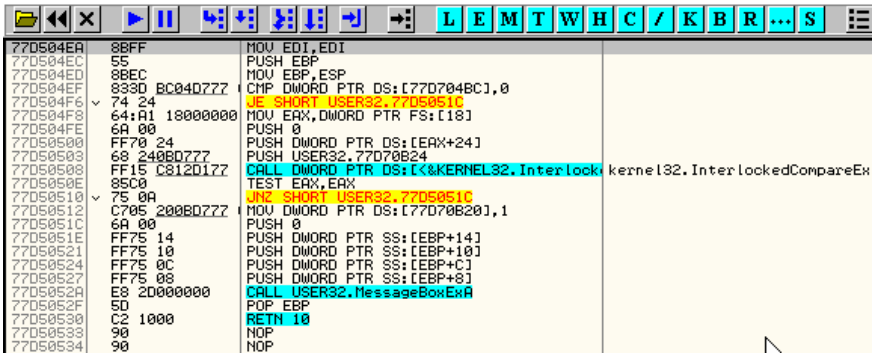
UNICODE [EAX]=="Brown fox" - OllyDbg treats EAX as a pointer to UNICODE string, converts it to ASCII and compa

BREAKPOINT CONDICIONAL LOG

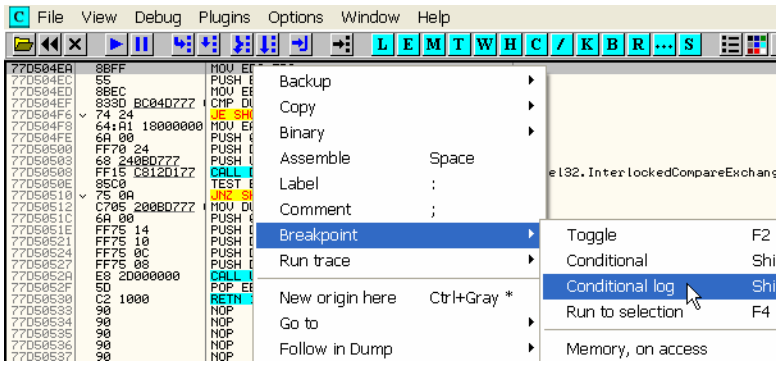
Es un BPX condicional igual que el anterior, solo que en este caso podemos activar que nos loguee o nos guarde ciertos valores al pasar por el BPX CONDICIONAL, en este caso las opciones son muchísimas, probaremos en el caso de una API que tiene varios accesos y queremos que loguee o nos guarde ciertos datos al pasar por ella.
Usamos el ejemplo de la api pero puede ser colocado un BPX CONDICIONAL LOG en cualquier dirección sea api o no, reiniciemos el OLLY.

Vamos a la dirección de la api, ya sabemos como CLICK DERECHO-GOTO EXPRESSION

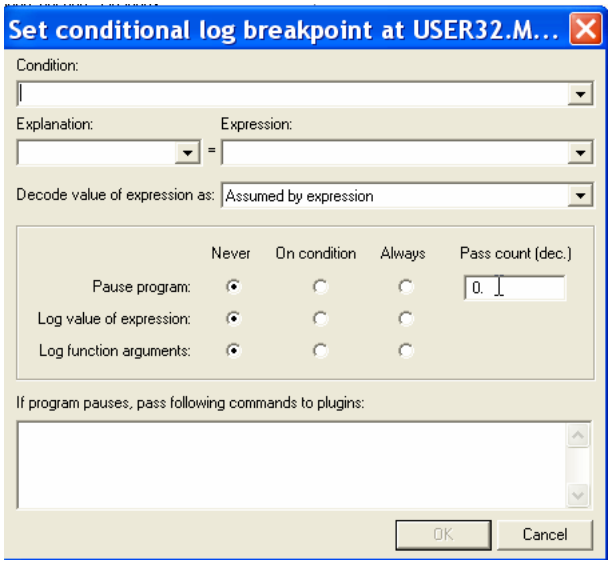




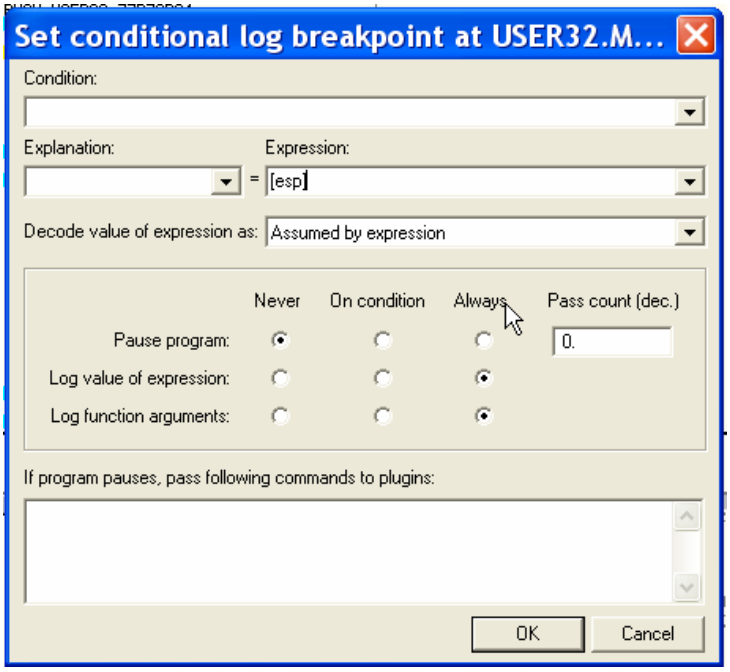
Allí estamos ahora hacemos CLICK DERECHO- BREAKPOINT- CONDICIONAL LOG



Vemos que nos aparece una ventana con más opciones



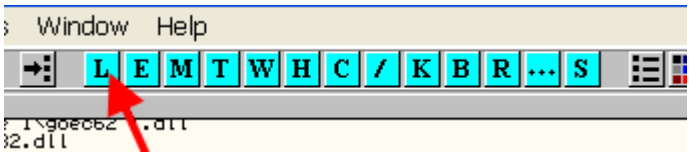
En este caso no haremos que pare en la api, solo que nos loguee ciertos datos de interés.



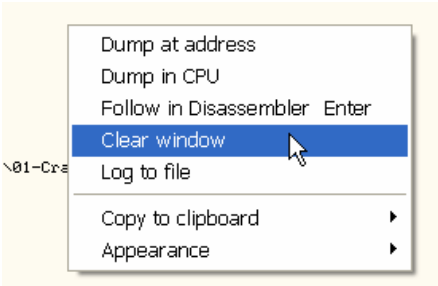
Como no queremos que pare no colocamos ninguna CONDICIÓN y en la opción PAUSE PROGRAM elegimos que no pare nunca o sea ponemos la marca en NEVER.

Luego en LOG VALUE OF EXPRESIÓN ponemos que nos guarde el valor de [esp] que escribimos en el renglón EXPRESSION, que como sabemos será la dirección de retorno al programa desde la api, ya que es el valor superior del stack, por lo tanto en la segunda opción LOG VALUE OF EXPRESIÓN elegimos ALWAYS para que la LOGUEE siempre, y en el tercer renglón nos da la opción si queremos LOGUEAR los argumentos o parámetros de la función o api, pues no estará de mas, ponemos ALWAYS tambien.

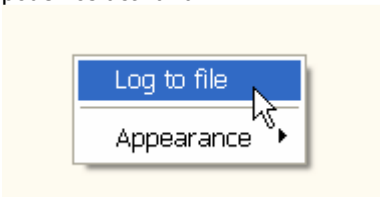
Vamos a la ventana LOG o L



Y limpiamos la misma para ver claro lo que guarde el programa a partir de aquí con CLICK DERECHO-CLEAR WINDOW.



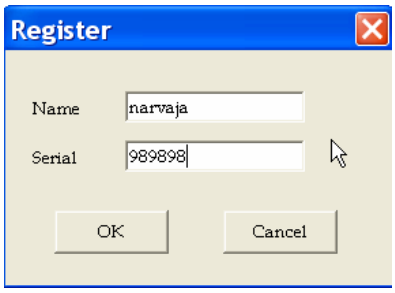
Vemos que también allí esta la opción de guardar lo logueado, en una fila de texto, si lo necesitamos lo podemos activar allí.



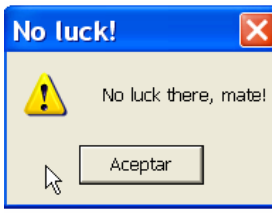
Ahora doy RUN con f9

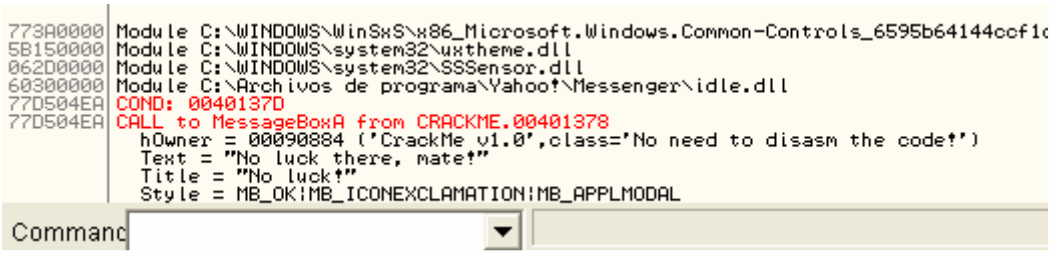
Llego a la ventana del crackme y aun no hay en el LOG nada sobre nuestra api pues aun no paso por la misma.

Vamos a REGISTER y pongamos un user y pass cualquiera.



Al apretar OK



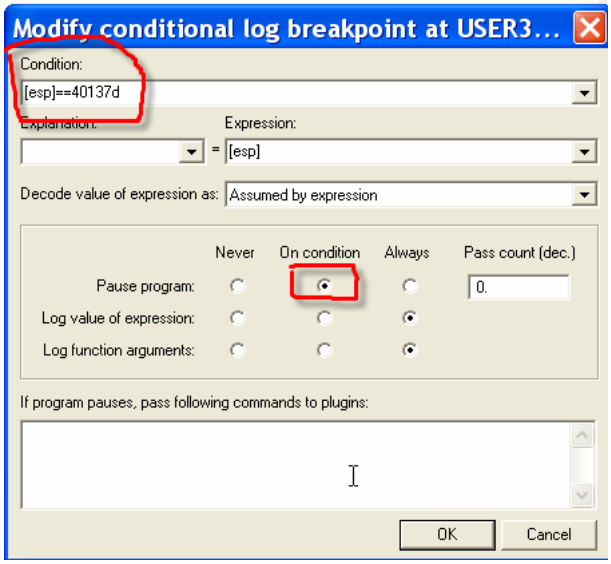


Vemos allí que nos muestra aun sin haberse detenido el OLLY, la info de la MessageBoxA, primero el valor superior del stack que es 40137D o sea la dirección de retorno de la api, y luego los parámetros de la misma, en este caso que es un call desde 401378, el texto, el titulo etc.

En este caso nos muestra solo una información pero que tal si una api se ejecuta 100 veces por ejemplo, podemos loguear todas a una fila de texto, pero si queremos que pare en una sola determinada de toas esas veces en ese caso, debemos usar la CONDICION.

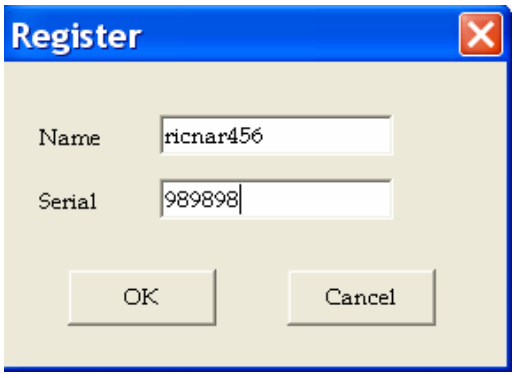
Si tuviera aquí una lista con la info de 100 veces diferentes que paro el programa en esa api, puedo colocarle algún dato que me da la info en la condición, por ejemplo la dirección de retorno en este caso, la puedo usar como condición y de esa forma solo parara de las 100, en la que regrese a 40137d, las que son llamadas de otras direcciones del programa no parara.

Para ello reinicio el OLLY y voy nuevamente a la api y ahora pongo un BPX CONDICIONAL LOG así.

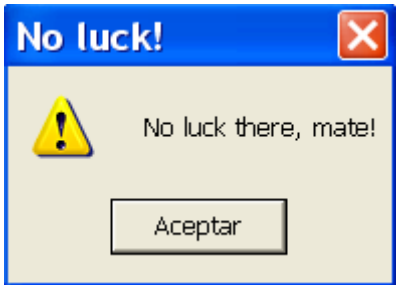


Cambio la tilde para que solo pare cuando se cumpla la condición (ON CONDITION) y escribo la condición o sea que solo pare cuando [esp] sea igual a 40137D.
En esta forma solo parara cuando pase por la api y sea el valor superior del stack 40137d el cual es la dirección de retorno de la api y me asegura que solo parara en esa.

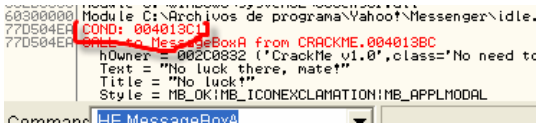
Corramos el programa pero ahora cuando vayamos a REGISTER coloquemos un nombre con números como ricnar456 y password 989898



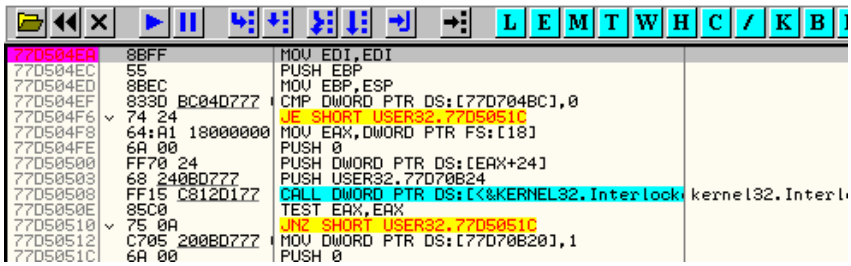
Recordamos que cuando poníamos un nombre que tenia números, paraba 2 veces en la api messageboxa, apretemos OK



Vemos que salio la primera MessageBoxA pero no paro OLLY en ella, vemos en el log la dirección de retorno es 4013C1 y no se activa la condición por ello no paro.

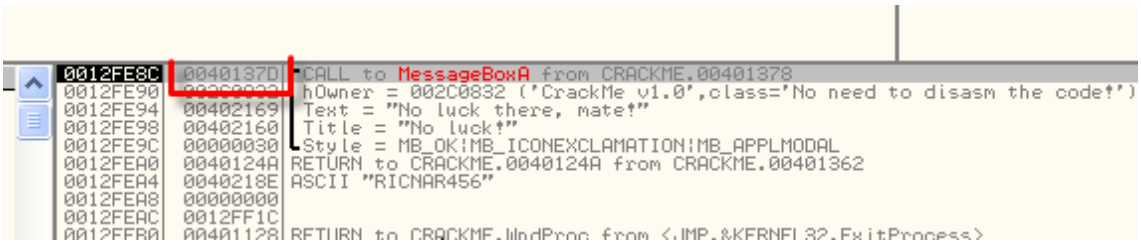


Al apretar aceptar

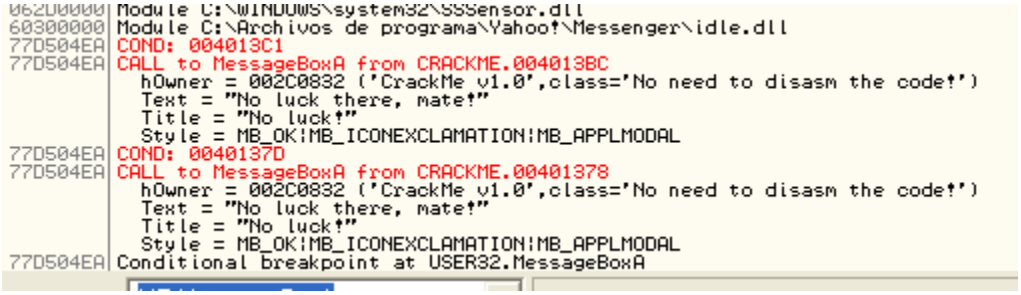


Ahí si en la segunda vez que va a la api, se cumple la condición y para.

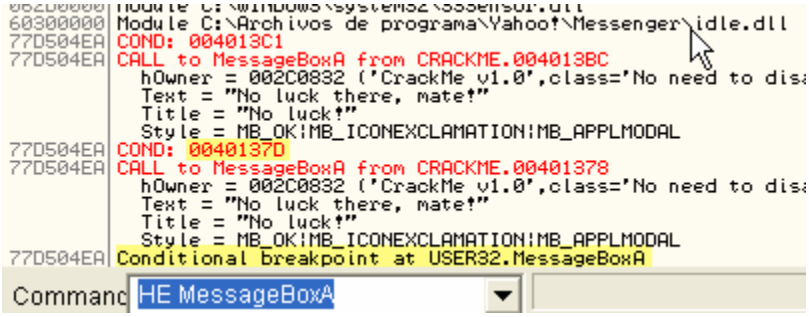
Vemos que en la parte superior del stack esta el valor 40137d que fue el que activo el BPX CONDICIONAL LOG.



De cualquier forma OLLYDBG guardo el LOG de ambas ocasiones en que se detuvo



Ahí se ve claro que la segunda vez la dirección de retorno fue 40137D y nos muestra en el mismo LOG que la segunda vez se detuvo.



Bueno creo que con esto tienen para practicar un rato, aun quedan los MESSAGE BREAKPOINTS pero no quiero matarlos, lo dejamos para la parte 12 y luego de los MESSAGE BPX le prometo seguir crackeando un poco para no aburrirlos, pero esto deben saberlo antes de continuar.

Hasta la parte 12
Ricardo Narvaja
27 de noviembre de 2005