

INTRODUCCIÓN AL CRACKING EN OLLYDBG PARTE 10

BREAKPOINTS EN OLLYDBG

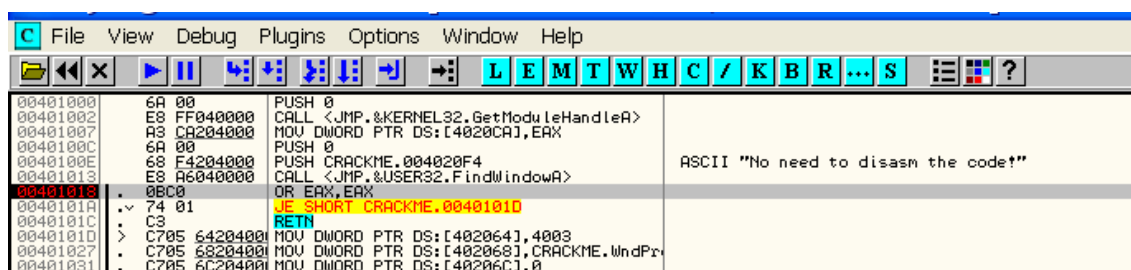
Esta parte la dedicaremos a entender los diferentes tipos de BREAKPOINTS en OLLYDBG, entendiendo por BREAKPOINTS las herramientas que trae OLLY para ayudarnos a detener la ejecución del programa en donde nosotros deseamos, para los cual usamos uno de los diferentes tipos de Breakpoint que estudiaremos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

BREAKPOINT COMUN o BPX

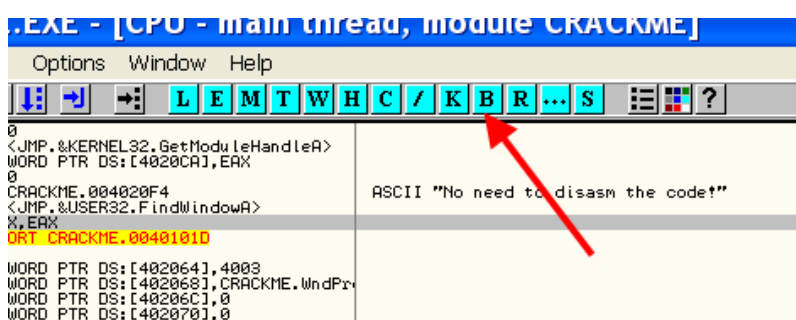
Es el Breakpoint común que hemos utilizado hasta ahora, también llamado BPX por su similitud a la forma de tipearlo en SOFTICE, muchas veces se utiliza la sigla BPX como abreviatura de BREAKPOINT, aunque estrictamente en OLLYDBG se tipea BP.

Normalmente se puede colocar marcando la línea que queremos ponerle el BPX y apretando F2, si apretamos F2 nuevamente se quitara.

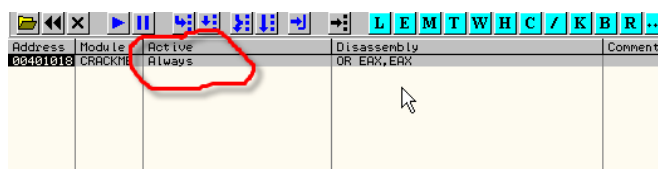
Miremos el CRACKME DE CRUEHEAD en el ENTRY POINT



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.

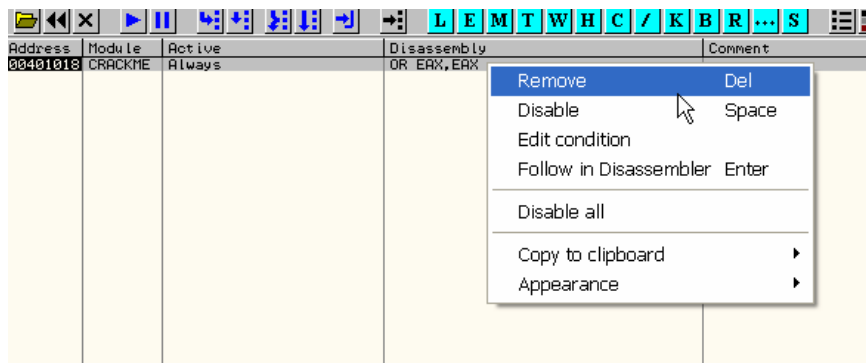


Vemos que nos muestra que esta activo



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

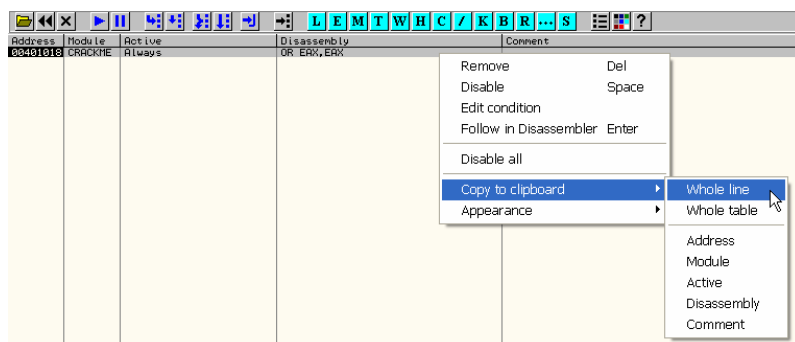
DISABLE: Para que quede en la lista de BREAKPOINTS pero deshabilitado, o sea no parara allí cuando pase por esa dirección.

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.

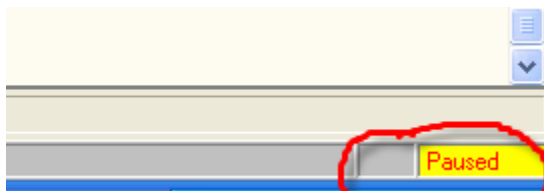
Allí paro

```

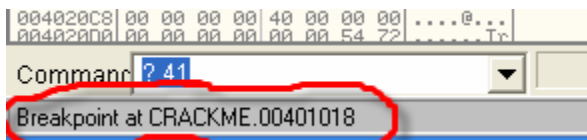
00401000 6A 00      PUSH 0
00401002 E8 FF040000 CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 A3 C8204000 MOV DWORD PTR DS:[4020C8],EAX
0040100C 6A 00      PUSH 0
0040100E 68 F4204000 PUSH CRACKME.004020F4
00401013 E8 A6040000 CALL <JMP.&USER32.FindWindowA>
00401018 0BC0      OR EAX,EAX
0040101A 74 01      JE SHORT CRACKME.0040101D
0040101C C3        RETN
0040101D C705 64204000 MOV DWORD PTR DS:[402064],4003
00401022 C705 68204000 MOV DWORD PTR DS:[402068],CRACKME.00402064

```

Y OLLY nos dice que esta PAUSADO

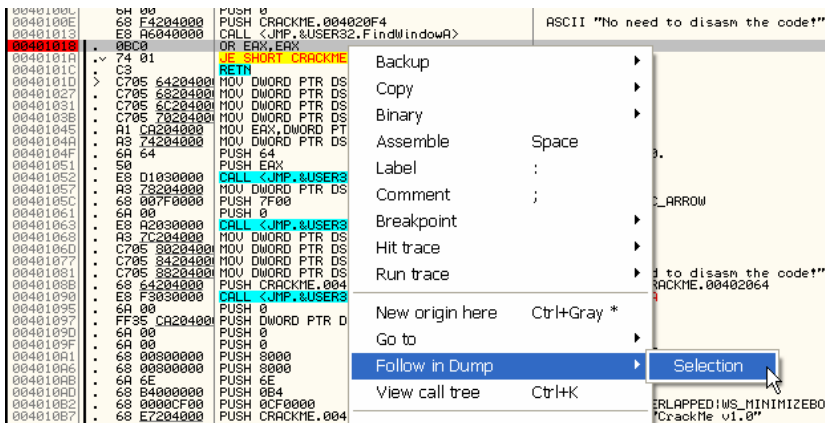


Y a la izquierda nos muestra el motivo porque paro



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

Address	Hex dump	ASCII
00401018	0B C0 74 01 C3 C7 05 64	0t0f&d
00401020	20 40 00 03 40 00 00 C7	@.00..s
00401028	05 68 20 40 00 28 11 40	h @.(40
00401030	00 C7 05 6C 20 40 00 00	.s! @.
00401038	00 00 00 C7 05 70 20 40	...s*p @
00401040	00 00 00 00 00 A1 CA 20[=
00401048	40 00 A3 74 20 40 00 6A	@.ut @.j
00401050	64 50 E8 D1 03 00 00 A3	dPp0..u

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

0040100E	68 F4204000	PUSH CRACKME.004020F4
00401013	E8 05040000	CALL <JMP.&USER32.FindWindowA>
00401018	0BC0	OR EAX,EAX
0040101A	74 01	JE SHORT CRACKME.0040101D
0040101C	C3	RETN

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

00401000	6A 00	PUSH 0	ASCII "No need to disasm the code!"
00401002	E8 FF040000	CALL <JMP.&KERNEL32.GetModuleHandleA>	
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA],EAX	
0040100C	6A 00	PUSH 0	
0040100E	68 F4204000	PUSH CRACKME.004020F4	
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>	
00401018	0BC0	OR EAX,EAX	
0040101A	74 01	JE SHORT CRACKME.0040101D	
0040101C	C3	RETN	
0040101D	C705 64204000	MOV DWORD PTR DS:[402064],4003	

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

00401000	A1 18104000	MOV EAX,DWORD PTR DS:[401018]
00401005	90	NOP
00401006	90	NOP
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA],EAX
0040100C	6A 00	PUSH 0

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

004010CF	FF35 04204000	PUSH DWORD PTR DS:[402004]
004010D5	E8 90030000	CALL <JMP.&USER32.ShowWindow>
DS:[00401018] 0174C00B		
EAX=0174C0CC		

Address	Hex dump	ASCII
00401018	0B C0 74 01	0174C00B
00401019	00 00 00 00	00000000

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.

Registers (FPU)	
EAX	0174C0CC
ECX	0012FFB0
EDX	7C91EB94 ntdll.
EBX	7FFD0000
ESP	0012FFC4
EBP	0012FFF0
ESI	FFFFFFFF
EDI	7C920738 ntdll.
EIP	004010D5 CRACKME
C 0	ES 0023 32bit
P 1	CS 001B 32bit

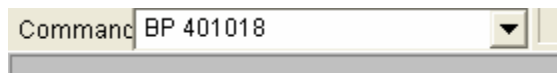
Como no era que en 401018 estaban los bytes 0B C0 74 01 ?, alguien esta mintiendo aquí pues esos bytes al revés son 0174C00B y el valor que movió a EAX es 0174C0CC, o sea que realmente en 401018 no hay un B0 cuando pongo un BREAKPOINT, OLLYDBG lo reemplaza por el valor CC que mas adelante cuando veamos 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.

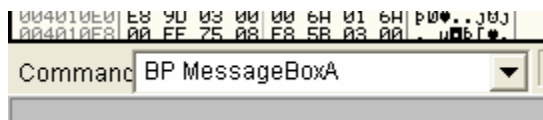
Así que no se confíen, recuerden siempre que si apretamos F2 cambiamos código aunque OLLY mantenga todo igual y el programa continúe, dicho BREAKPOINT puede ser detectado, si en algún programa ven que al colocarle un BREAKPOINT no corre pues quítenlo hay otras posibilidades en OLLYDBG para parar donde queremos.

También un BREAKPOINT se puede tipear en la commandbar

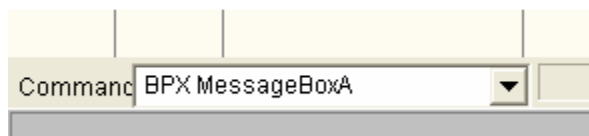
BP 401018



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.

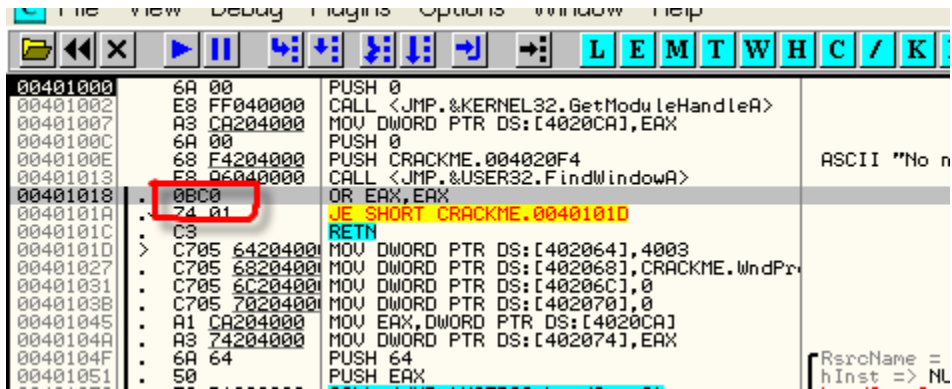
Igual vemos que coloco algunas

Address	Module	Active	Disassembly	Comme
00401018	CRACKME	Always	OR EAX,EAX	
0040135C	CRACKME	Always	CALL <JMP.&USER32.MessageBoxA>	
00401378	CRACKME	Always	CALL <JMP.&USER32.MessageBoxA>	
004013BC	CRACKME	Always	CALL <JMP.&USER32.MessageBoxA>	
77D504EA	USER32	Always	MOV EDI,EDI	

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.

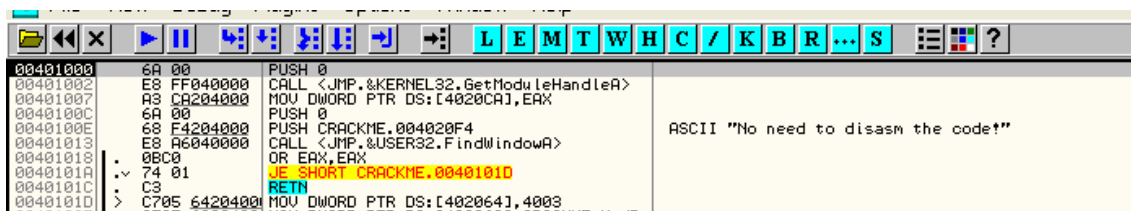


BREAKPOINTS ON MEMORY (MEMORY BREAKPOINTS O BREAKPOINTS EN MEMORIA)

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)

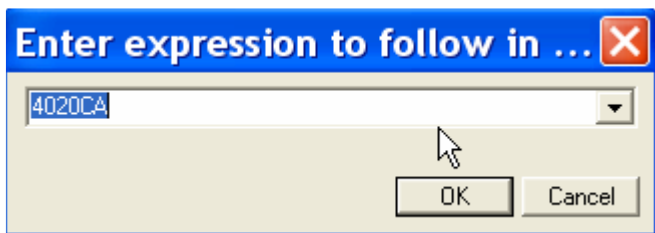
Bueno como funcionan, al colocarlos OLLYDBG lo que hace es cambiarle el permiso de una sección o parte de la misma, según el permiso que le demos, podemos colocar BPM ON ACCESS que detendrá la ejecución, cuando se acceda al sector al cual le colocamos el BPM, o sea parara cuando ejecute esos bytes o 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



Address	Hex dump	ASCII
004020CA	00 00 00 00 00 00 00 00
004020D2	00 00 00 00 54 72 79 20Try
004020DA	74 6F 20 63 72 61 63 6B	to crack
004020E2	20 6D 65 21 00 43 72 61	me!.Cra
004020EA	63 68 4D 65 20 76 31 2E	ckMe v1.
004020F2	30 00 4E 6F 20 6E 65 65	0.No nee
004020FA	64 20 74 6F 20 64 69 73	d to dis
00402102	61 73 6D 20 74 68 65 20	asm the
0040210A	63 6F 64 65 21 00 4D 45	code!.ME
00402112	4E 55 00 44 4C 47 5F 52	NU.DLG_R
0040211A	45 47 49 53 00 44 4C 47	EGIS.DLG
00402122	5F 41 42 4F 55 54 00 47	_ABOUT.G
0040212A	6F 6F 64 20 77 6F 72 6B	ood work
00402132	21 00 47 72 65 61 74 20	!.Great
0040213A	77 6F 72 6B 2C 20 6D 43	work, ma
00402142	74 65 21 0D 4E 6F 77 20	te!.Now
0040214A	74 72 79 20 74 68 65 20	try the
00402152	6E 65 78 74 20 43 72 61	next Cra
0040215A	63 68 4D 65 21 00 4E 6F	ckMe!.No
00402162	20 6C 75 63 6B 21 00 4E	luck!.N
0040216A	6F 20 6C 75 63 6B 20 74	o luck t
00402172	68 65 72 65 2C 20 6D 61	here, ma
0040217A	74 65 21 00 00 00 00 00	te!.....

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.

Address	Hex dump	ASCII
004020CA	00 00 00 00 00 00 00 00
004020D2	00 00 00 00 54 72 79 20Try
004020DA	74 6F 20 63 72 61 63 6B	to crack
004020E2	20 6D 65 21 00 43 72 61	me!.Cra
004020EA	63 68 4D 65 20 76 31 2E	ckMe v1.
004020F2	30 00 4E 6F 20 6E 65 65	0.No nee
004020FA	64 20 74 6F 20 64 69 73	d to dis
00402102	61 73 6D 20 74 68 65 20	asm the

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.

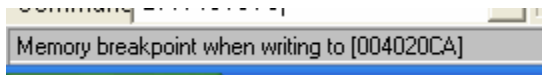
Address	Hex dump	ASCII
004020CA	00 00 00 00 00 00 00 00
004020D2	00 00 00 00 54 72 79 20Try
004020DA	74 6F 20 63 72 61 63 6B	to crack
004020E2	20 6D 65 21 00 43 72 61	me!.Cra
004020EA	63 68 4D 65 20 76 31 2E	ckMe v1.
004020F2	30 00 4E 6F 20 6E 65 65	0.No nee
004020FA	64 20 74 6F 20 64 69 73	d to dis
00402102	61 73 6D 20 74 68 65 20	asm the
0040210A	63 6F 64 65 21 00 4D 45	code!.ME
00402112	4E 55 00 44 4C 47 5F 52	NU.DLG_R
0040211A	45 47 49 53 00 44 4C 47	EGIS.DLG
00402122	5F 41 42 4F 55 54 00 47	_ABOUT.G
0040212A	6F 6F 64 20 77 6F 72 6B	ood work
00402132	21 00 47 72 65 61 74 20	!.Great
0040213A	77 6F 72 6B 2C 20 6D 43	work, ma
00402142	74 65 21 0D 4E 6F 77 20	te!.Now
0040214A	74 72 79 20 74 68 65 20	try the
00402152	6E 65 78 74 20 43 72 61	next Cra
0040215A	63 68 4D 65 21 00 4E 6F	ckMe!.No
00402162	20 6C 75 63 6B 21 00 4E	luck!.N
0040216A	6F 20 6C 75 63 6B 20 74	o luck t
00402172	68 65 72 65 2C 20 6D 61	here, ma
0040217A	74 65 21 00 00 00 00 00	te!.....

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.

Address	Hex dump	Disassembly	Comment
00401000	6A 00	PUSH 0	
00401002	E8 FF400000	CALL <JMP.&KERNEL32.GetModuleHandleA>	
00401007	A3 C0204000	MOV DWORD PTR DS:[4020CA],EAX	CRACKME.00400000
0040100C	6A 00	PUSH 0	
0040100E	68 F4204000	PUSH CRACKME.004020F4	
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>	ASCII "No need to disasm the code!"

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

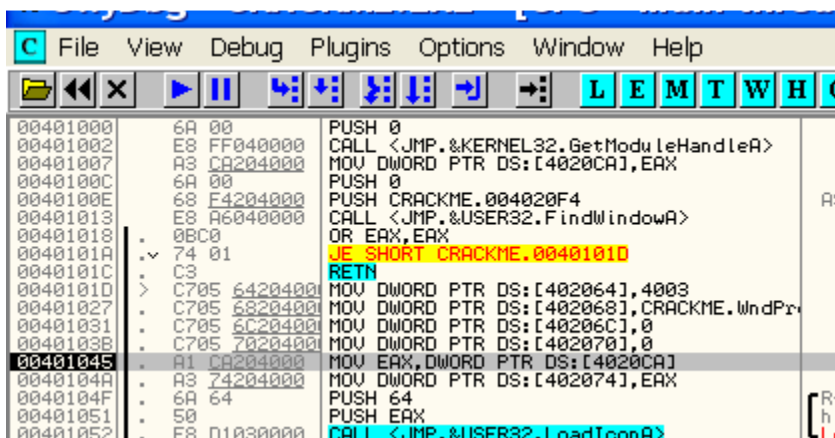


Esta por escribir el valor que esta en EAX al contenido de 4020CA, lo cual provoca que OLLYDBG pare, si recuerdan cuando tratábamos de escribir en la memoria y esta no tenia permiso de escritura, se generaba una excepción, pues eso es lo que básicamente hace OLLYDBG, al colocar un BPM ON ACCESS le quita 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.

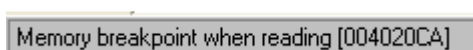
Si apreto F7 ahora si guardara el valor ya que el mecanismo del OLLY es sofisticado y permite que una vez detenido, si se pueda escribir al apretar f7.

Address	Hex dump	ASCII
004020CA	00 00 40 00 00 00 00 00	..@....
004020D2	00 00 00 00 54 72 79 20	...Try
004020DA	74 6F 20 63 72 61 63 6E	to crack
004020E2	20 6D 65 21 00 43 72 61	me!.Cra

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

Registers (FPU)	
EAX	00400000 ASCII "MZP"
ECX	7C92056D ntdll.7C92
EDX	00140608
EBX	7FFDE000
ESP	0012FFC4
EBP	0012FFFF
ESI	FFFFFFFF
EDI	7C920738 ntdll.7C92
EIP	0040104A CRACKME.00
C 0	ES 0023 32bit 0(FF
P 1	CS 001B 32bit 0(FF
A 0	SS 0023 32bit 0(FF
Z 1	DS 0023 32bit 0(FF
S 0	FS 003B 32bit 7FFD
T 0	GS 0000 NULL

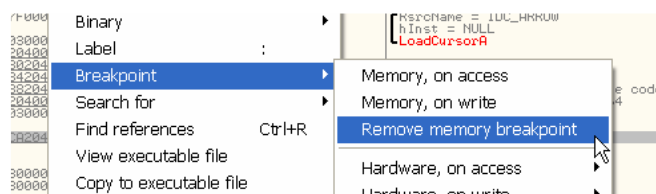
Apreto F9 a ver si alguna vez mas para.

0040108B	. 68 64204000	PUSH CRACKME.00402064
00401090	. E8 F3030000	CALL <JMP.&USER32.RegisterClassA>
00401095	. 6A 00	PUSH 0
00401097	. FF35 0A204000	PUSH DWORD PTR DS:[4020CA]
0040109D	. 6A 00	PUSH 0
0040109F	. 6A 00	PUSH 0
004010A1	. 68 00800000	PUSH 8000
004010A2	. 68 00000000	PUSH 0000

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

Memory breakpoint when reading [004020CA]

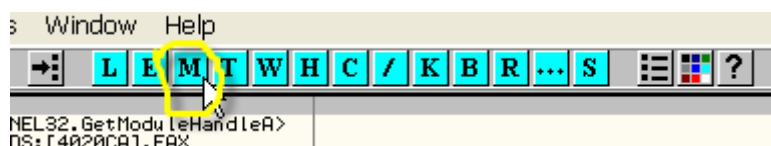
Ahora si quiero quitar el BPM que coloque hago click derecho en el DUMP y elijo BREAKPOINT-REMOVE MEMORY BREAKPOINT y con eso se quita, lo mismo como ya dijimos si colocamos uno nuevo.



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.

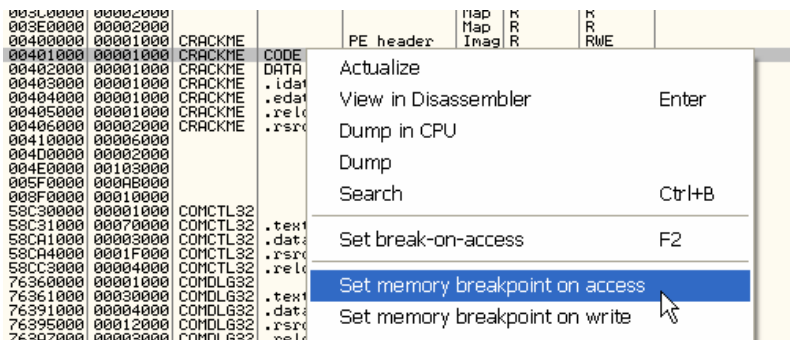
Para ellos vamos a VIEW-MEMORY o a la ventana M que es lo mismo.



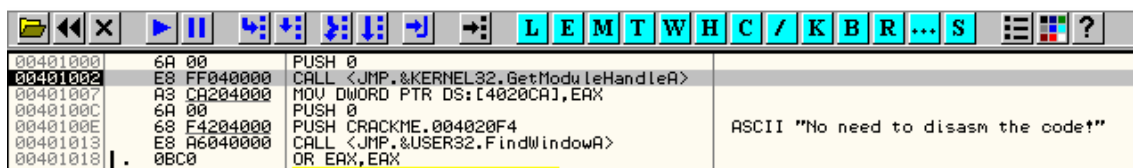
Allí vemos las secciones del crackme y mas abajo las secciones de las dlls y diferentes secciones utilizadas, podemos marcar la que deseemos por ejemplo, la sección que comienza en 401000 allí la vemos.

00390000	00004000				Priv	RW	RW	
003A0000	00004000				Priv	RW	RW	
003B0000	00003000				Map	R	R	
003C0000	00002000				Map	R	R	
003E0000	00002000				Map	R	R	
00400000	00001000	CRACKME	PE header		Imag	R	RWE	
00401000	00001000	CRACKME	CODE	code	Imag	R	RWE	
00402000	00001000	CRACKME	DATA	data	Imag	R	RWE	
00403000	00001000	CRACKME	.idata	imports	Imag	R	RWE	
00404000	00001000	CRACKME	.edata	exports	Imag	R	RWE	
00405000	00001000	CRACKME	.reloc	relocations	Imag	R	RWE	
00406000	00002000	CRACKME	.rsrc	resources	Imag	R	RWE	
00410000	00006000				Map	R E	R E	
004D0000	00002000				Map	R E	R E	
004E0000	00103000				Map	R	R	
005F0000	000AB000				Map	R E	R E	

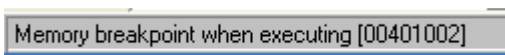
Hago click derecho en dicha sección y elijo SET MEMORY BREAKPOINT ON ACCESS.



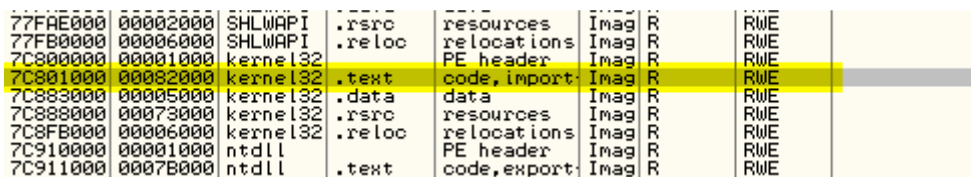
Vemos que justo debajo tenemos la posibilidad de colocar también un BPM ON WRITE pero en este caso lo haremos ON ACCESS doy RUN.



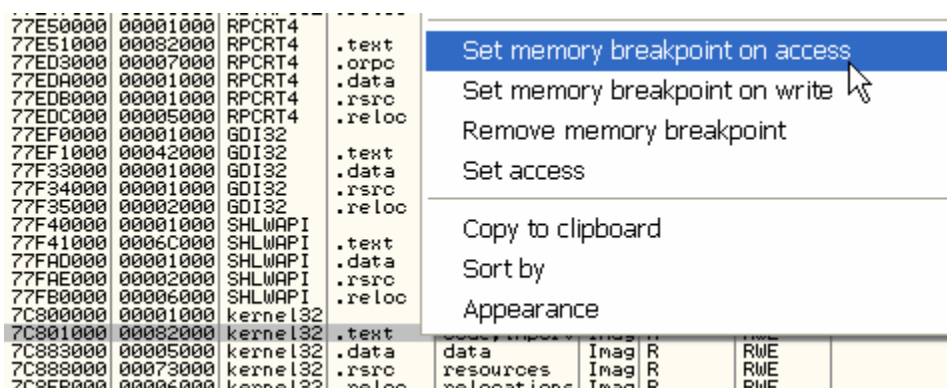
Vemos que para en la siguiente línea veamos porque



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.

Address	Disassembly	Comment
7C80B529	8BFF	MOV EDI,EDI
7C80B52B	55	PUSH EBP
7C80B52C	8BEC	MOV EBP,ESP
7C80B52E	837D 08 00	CMP DWORD PTR SS:[EBP+8],0
7C80B532	74 18	JE SHORT kernel32.7C80B540
7C80B534	FF75 08	PUSH DWORD PTR SS:[EBP+8]
7C80B537	E8 682D0000	CALL kernel32.7C80E2A4
7C80B53C	85C0	TEST EAX,EAX
7C80B53E	74 08	JE SHORT kernel32.7C80B548
7C80B540	FF70 04	PUSH DWORD PTR DS:[EAX+4]
7C80B543	E8 F4300000	CALL kernel32.GetModuleHandleW
7C80B548	5D	POP EBP
7C80B549	C2 0400	RETN 4
7C80B54C	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]
7C80B552	8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]
7C80B555	8B40 08	MOV EAX,DWORD PTR DS:[EAX+8]
7C80B558	EB EE	JMP SHORT kernel32.7C80B548
7C80B55A	90	NOP
7C80B55B	90	NOP
7C80B55C	90	NOP

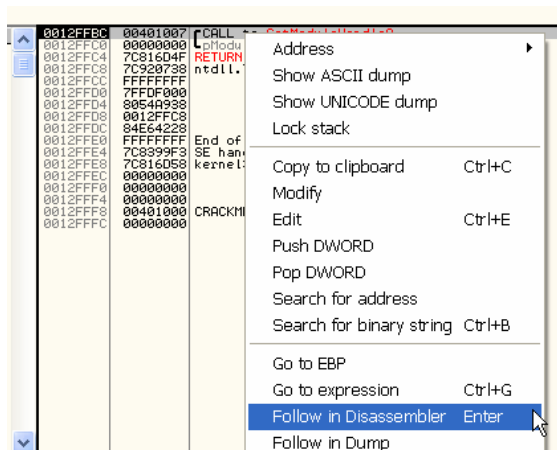
En mi caso para allí y si veo en el stack, veo que paro en una api

Address	Disassembly	Comment
0012FFBC	00401007	CALL to GetModuleHandleA
0012FFC0	00000000	pModule = NULL
0012FFC4	7C816D4F	RETURN to kernel32.7C816D4F
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFDF000	
0012FFD4	8054A938	
0012FFD8	0012FFC8	
0012FFDC	84E64228	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C8399F3	SE handler
0012FFE8	7C816D58	kernel32.7C816D58
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	00401000	CRACKME.<ModuleEntryPoint>
0012FFFC	00000000	

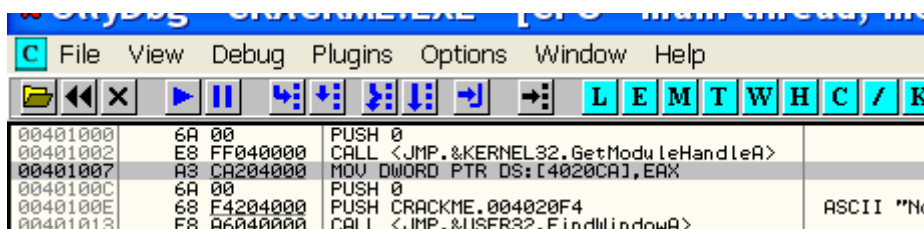
En este caso para al ejecutar una instrucción de la kernel32.dll y me funciona para saber cual es la primer api accedida desde el programa en dicha dll, si vemos la dirección de retorno en la primera línea del stack.

Address	Disassembly	Comment
0012FFBC	00401007	CALL to GetModuleHandleA
0012FFC0	00000000	pModule = NULL
0012FFC4	7C816D4F	RETURN to kernel32.7C816D4F
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFDF000	
0012FFD4	8054A938	
0012FFD8	0012FFC8	
0012FFDC	84E64228	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C8399F3	SE handler
0012FFE8	7C816D58	kernel32.7C816D58
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	00401000	CRACKME.<ModuleEntryPoint>
0012FFFC	00000000	

Si vemos adonde retornara con click derecho- FOLLOW IN DISASSEMBLER



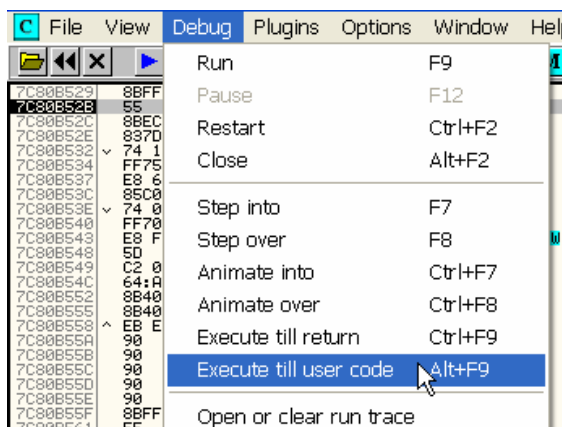
Allí vemos que retornara a 401007 de la llamada a GetModuleHandleA que hay al inicio del programa.



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.

Address	Disassembly	Comment
00401000	6A 00	PUSH 0
00401002	E8 FF040000	CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA1],EAX
0040100C	6A 00	PUSH 0
0040100E	68 F4204000	PUSH CRACKME.004020F4
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>
00401018	0BC0	OR EAX,EAX

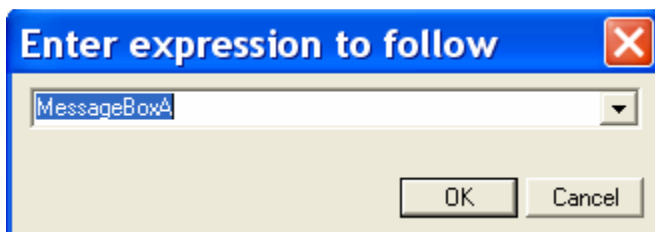
Si pongo un BPM de nuevo en la kernel32.dll parara en la siguiente api que se ejecute de esa dll y así.

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

Address	Disassembly	Comment
00402108	65 20 63 6F 64 65 21 00	e code?.
00402118	4D 4F 4F 5F 00 44 4C 47	MENU.DLG

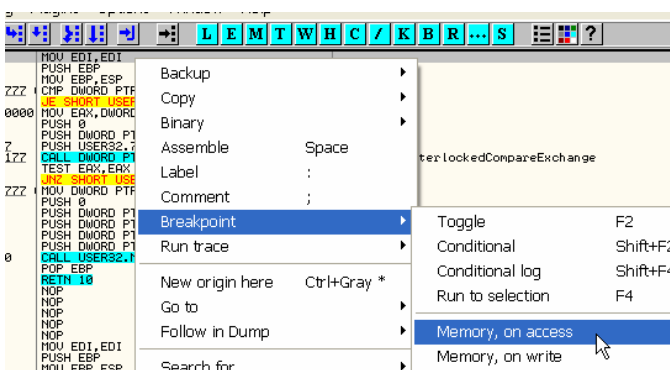
Command: ? MessageBoxA HEX: 77D504EA - DEC: 2010449130 - ASCII: wŃ□ê

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.

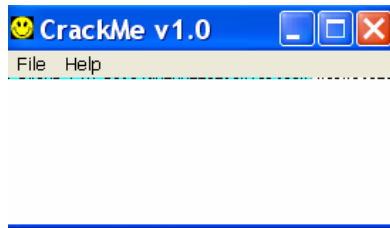


Address	Disassembly	Comment
77D504EA	8BFF	MOV EDI,EDI
77D504EC	55	PUSH EBP
77D504ED	8BEC	MOV EBP,ESP
77D504EF	8330 BC04D777	CMP DWORD PTR DS:[77D704BC],0
77D504F6	74 24	JE SHORT USER32.77D50510
77D504F8	64:01 18000000	MOV EAX,DWORD PTR FS:[18]
77D504FE	6A 00	PUSH 0
77D50500	FF70 24	PUSH DWORD PTR DS:[EAX+24]
77D50503	68 240B0777	PUSH USER32.77D70B24
77D50508	FF15 C812D177	CALL DWORD PTR DS:[<&KERNEL32.Interlock
77D5050E	85C0	TEST EAX,EAX
77D50510	75 0A	JNZ SHORT USER32.77D50510
77D50512	C705 200B0777	MOV DWORD PTR DS:[77D70B20],1
77D5051C	6A 00	PUSH 0
77D5051E	FF75 14	PUSH DWORD PTR SS:[EBP+14]
77D50521	FF75 10	PUSH DWORD PTR SS:[EBP+10]
77D50524	FF75 0C	PUSH DWORD PTR SS:[EBP+0C]
77D50527	FF75 08	PUSH DWORD PTR SS:[EBP+08]
77D5052A	E8 2D000000	CALL USER32.MessageBoxExA
77D5052F	5D	POP EBP
77D50530	C2 1000	RETN 10
77D50533	90	NOP
77D50534	90	NOP

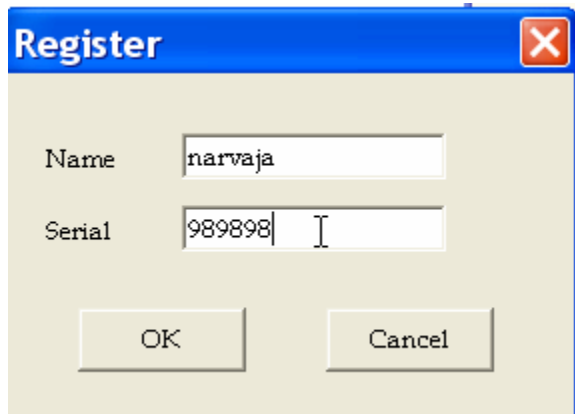
Marco la primera línea



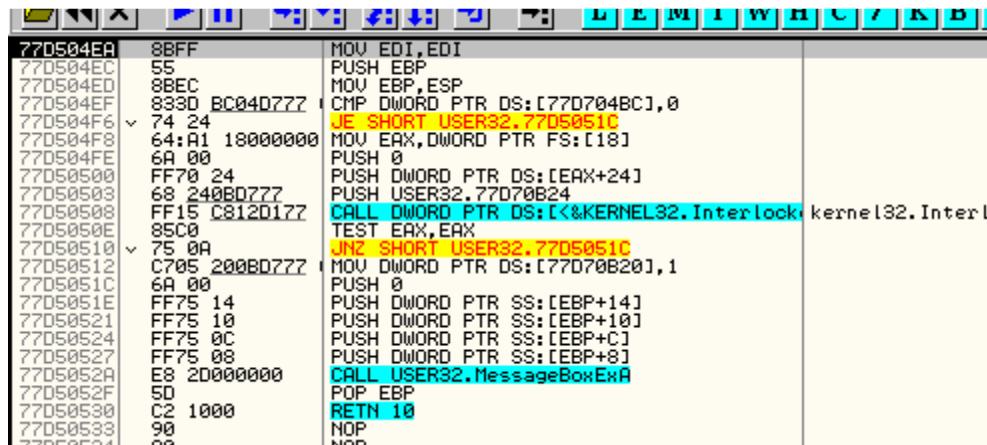
Y veo que también tengo la posibilidad de elegir poner un BPM ON ACCESS o ON WRITE, elijo la primera opción y doy RUN.



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 allí, 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

