

INTRODUCCIÓN AL CRACKING CON OLLYDBG parte 12

Veremos la forma práctica de usar los MENSAJES en WINDOWS.

Tengo esta cita que define un poco que son los mensajes en WINDOWS

Los mensajes en Windows son usados para comunicar la mayoría de los sucesos, al menos en los niveles básicos. Si quieres que una ventana o control (el cual es una ventana especializada) haga algo, debes enviarle un mensaje. Si otra ventana quiere que vos hagas algo, entonces te envía un mensaje. Si ocurre un evento, como cuando el usuario mueve el mouse, presiona el teclado, etc... entonces el sistema la envía un mensaje a la ventana afectada. Dicha ventana recibe el mensaje y actúa adecuadamente.

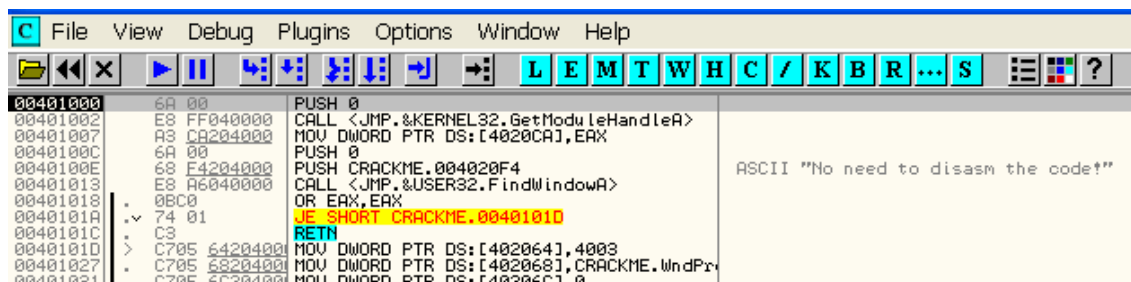
O sea que como ya sabemos, podemos en OLLYDBG trabajar poniendo BPX en las apis que realizan la mayoría de las funciones del programa, pero muchas veces resulta mas directo poner Breakpoints en estos mensajes que el programa envía directo al sistema llamados MESSAGE BREAKPOINTS o BMSG por la sigla que se utilizaba en SOFTICE para poder tipearlos.

Existe un Bucle de mensajes en todo programa compuesto por varias apis, que no profundizaremos aquí, dichas apis se ocupan del tratamiento de los mensajes, el que quiera profundizar el tema, este es un tutorial bastante completo de cómo funcionan, aunque para el cracking no es necesario, profundizar tanto, mas vale saber que dichos mensajes existen y son procesados y en cual colocar BMSG para obtener un buen resultado.

http://winprog.org/tutorial/es/message_loop.html

Veremos como ejemplo el caso siguiente: para hallar seriales queremos que el programa rompa al apretar en el botón que ingresa nuestro nombre y serial, para luego analizar que operaciones realiza con ellos.

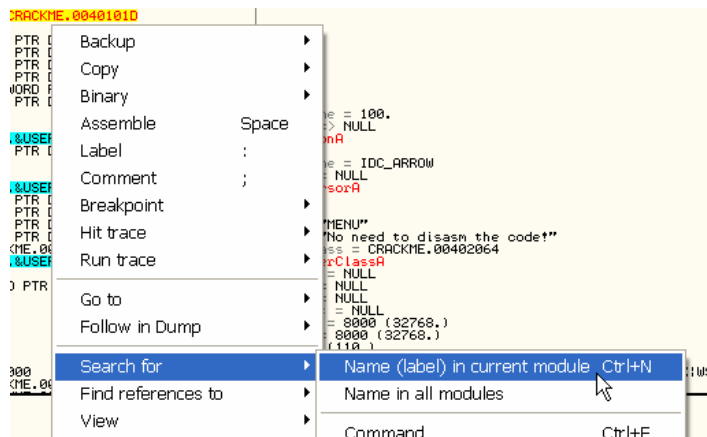
Si abrimos el Crackme de Cruehead en OLLYDBG.



```
File View Debug Plugins Options Window Help
[Icons] [L] [E] [M] [T] [W] [H] [C] [K] [B] [R] [S] [Icons] [?]
00401000 6A 00 PUSH 0
00401002 E8 FF040000 CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 A3 C0204000 MOV DWORD PTR DS:[4020C0],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
00401027 C705 68204000 MOV DWORD PTR DS:[402068],CRACKME.WndProc
00401031 C705 6C204000 MOV DWORD PTR DS:[40206C],0
```

Vamos a ver primero el método con las apis, para ello miramos las apis que importa el programa, a ver cual es la encargada de realizar el trabajo de traer el texto que tipeamos para su procesamiento y futura comparación.

Para ello en el listado hacemos CLICK DERECHO-SEARCH FOR-NAME (LABEL) IN CURRENT MODULE

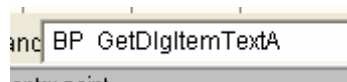


Las apis mas usadas para ingresar texto son GetDlgItemTextA y GetWindowTextA, así que miraremos allí si están

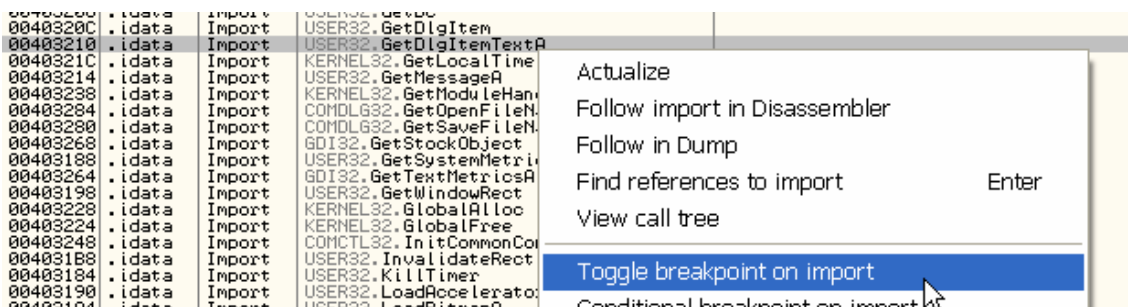
0040326C	.idata	Import	GDI32.EndPage
00403200	.idata	Import	USER32.EndPaint
00403240	.idata	Import	KERNEL32.ExitProcess
00403204	.idata	Import	USER32.FindWindowA
00403208	.idata	Import	USER32.GetDC
0040320C	.idata	Import	USER32.GetDlgItem
00403210	.idata	Import	USER32.GetDlgItemTextA
0040321C	.idata	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

Vemos que usa la api GetDlgItemTextA por lo cual seguro la utilizara para ingresar el texto

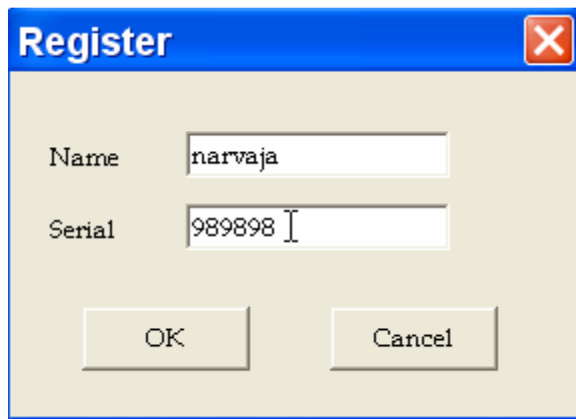
Pongamos un BP en dicha api



O



Bueno una vez colocado el BPX en la api corremos el programa y vamos a ingresar nuestro user y pass



Al apretar OK para en nuestro BREAKPOINT

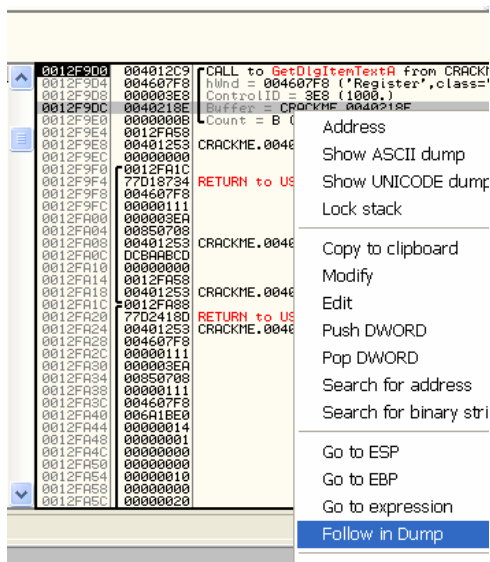
Address	Disassembly	Comment
77D6AC1E	8BFF	MOV EDI,EDI
77D6AC20	55	PUSH EBP
77D6AC21	8BEC	MOV EBP,ESP
77D6AC23	FF75 0C	PUSH DWORD PTR SS:[EBP+C]
77D6AC26	FF75 08	PUSH DWORD PTR SS:[EBP+8]
77D6AC29	E8 E89BFBFF	CALL USER32.GetDlgItem
77D6AC2E	85C0	TEST EAX,EAX
77D6AC30	74 0E	JE SHORT USER32.77D6AC40
77D6AC32	FF75 14	PUSH DWORD PTR SS:[EBP+14]
77D6AC35	FF75 10	PUSH DWORD PTR SS:[EBP+10]
77D6AC38	50	PUSH EAX
77D6AC39	E8 FE74FCFF	CALL USER32.GetWindowTextA
77D6AC3E	EB 0E	JMP SHORT USER32.77D6AC4E
77D6AC40	837D 14 00	CMP DWORD PTR SS:[EBP+14],0
77D6AC44	74 06	JE SHORT USER32.77D6AC4C
77D6AC46	8B45 10	MOV EAX,DWORD PTR SS:[EBP+10]
77D6AC49	C600 00	MOV BYTE PTR DS:[EAX],0
77D6AC4C	33C0	XOR EAX,EAX
77D6AC4E	5D	POP EBP
77D6AC4F	C3	RETN 4

Si miramos el stack

Address	Disassembly	Comment
0012F9D0	004012C9	CALL to GetDlgItemTextA from CRACKME.004012C4
0012F9D4	004607F8	hWnd = 004607F8 ('Register',class='#32770')
0012F9D8	000003E8	ControlID = 3E8 (1000.)
0012F9DC	0040218E	Buffer = CRACKME.0040218E
0012F9E0	00000000	Count = 0
0012F9E4	0012FA58	
0012F9E8	004012C9	CRACKME.004012C9

Vemos que uno de los parámetros de la api es el buffer, allí guardara el texto que estamos ingresando.

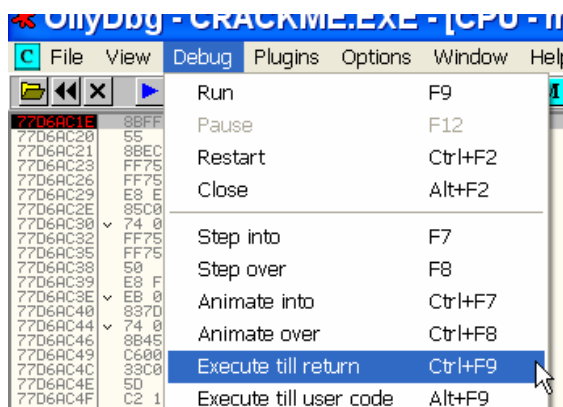
Veamos el BUFFER en el DUMP con FOLLOW IN DUMP o yendo al DUMP y haciendo GOTO EXPRESSION=40218E



Allí lo vemos vacío pues aun no se ejecuto la api

Address	Hex dump	ASCII
0040218E	00 00 00 00 00 00 00 00
00402196	00 00 00 00 00 00 00 00
0040219E	00 00 00 00 00 00 00 00
004021A6	00 00 00 00 00 00 00 00
004021AE	00 00 00 00 00 00 00 00
004021B6	00 00 00 00 00 00 00 00
004021BE	00 00 00 00 00 00 00 00
004021C6	00 00 00 00 00 00 00 00
004021CE	00 00 00 00 00 00 00 00
004021D6	00 00 00 00 00 00 00 00
004021DE	00 00 00 00 00 00 00 00
004021E6	00 00 00 00 00 00 00 00
004021EE	00 00 00 00 00 00 00 00
004021F6	00 00 00 00 00 00 00 00
004021FE	00 00 00 00 00 00 00 00
00402206	00 00 00 00 00 00 00 00

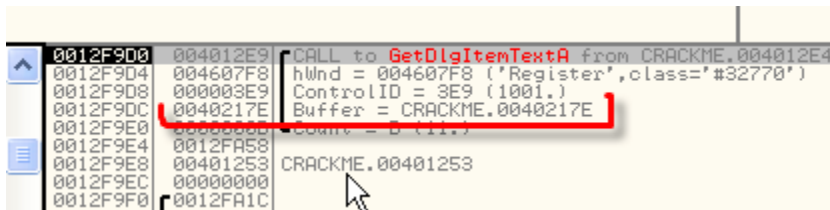
Así que ejecutemos la api hasta el RET con DEBUG- EXECUTE TILL RETURN



Miramos que en el DUMP que la api guardo allí nuestro nombre.

Address	Hex dump	ASCII
0040218E	6E 61 72 76 61 6A 61 00	narvaja.
00402196	00 00 00 00 00 00 00 00
0040219E	00 00 00 00 00 00 00 00
004021A6	00 00 00 00 00 00 00 00
004021AE	00 00 00 00 00 00 00 00
004021B6	00 00 00 00 00 00 00 00
004021BE	00 00 00 00 00 00 00 00
004021C6	00 00 00 00 00 00 00 00

Si doy RUN nuevamente con F9 vuelve a parar en la api



Esta ves el buffer es 40217e veamos en el dump

Address	Hex dump	ASCII
0040217E	00 00 00 00 00 00 00 00
00402186	00 00 00 00 00 00 00 00
0040218E	6E 61 72 76 61 6A 61 00	narvaja.
00402196	00 00 00 00 00 00 00 00
0040219E	00 00 00 00 00 00 00 00
004021A6	00 00 00 00 00 00 00 00

Ahora hago EXECUTE TILL RETURN

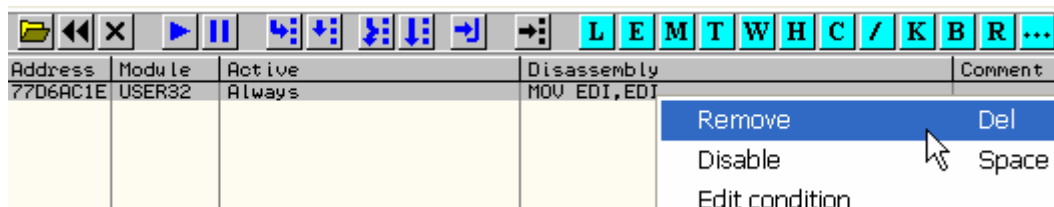
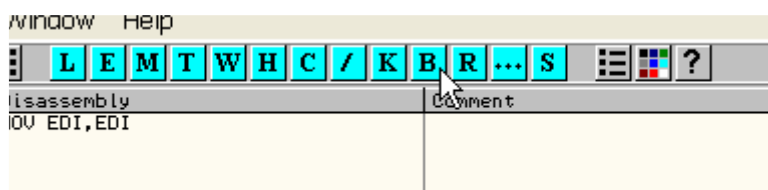
Address	Hex dump	ASCII
0040217E	39 38 39 38 39 38 00 00	989898..
00402186	00 00 00 00 00 00 00 00
0040218E	6E 61 72 76 61 6A 61 00	narvaja.
00402196	00 00 00 00 00 00 00 00
0040219E	00 00 00 00 00 00 00 00

Y ingresa mi serial.

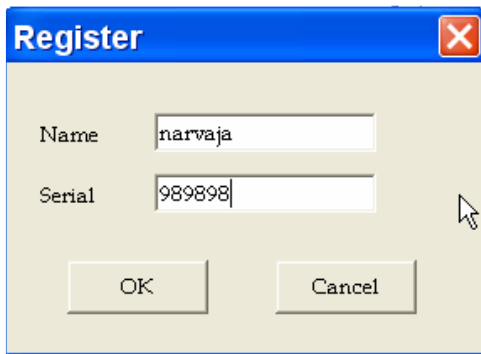
Ya comprendemos que para la búsqueda de seriales correctos, debemos detenernos en donde ingresa nuestro nombre y serial o los datos que se nos pidan, mas adelante veremos como seguir hasta hallar el serial correcto.

Bueno hemos parado en el punto en que ingresa nuestro user y serial con apis, ahora veamos como hacerlo con BMSG, ya que muchos programadores sabedores de que para obtener el serial debemos detenernos aquí, para complicar las cosas, no utilizan apis para ingresar el texto si no que lo hacen mediante mensajes de Windows.

Quitemos los BREAKPOINTS que colocamos yendo a la ventana B y borrando todos con REMOVE

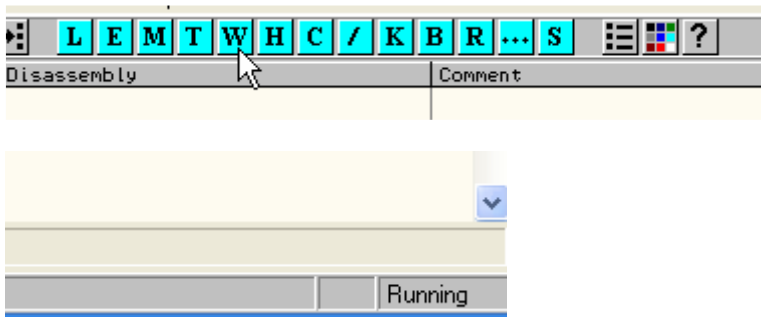


Corramos nuevamente el programa y lleguemos hasta la ventana de ingresar el user y serial



Una de las diferencias entre trabajar con apis y con BMSG es que en las apis podemos poner un BPX al inicio del programa si sabemos que api utiliza, en cambio para poner BMSG debe estar creada la ventana, antes no podemos poner BMSG en ella.

Para colocar el BMSG vamos a la ventana W o WINDOWS sin pausar el programa que continua RUNNING.



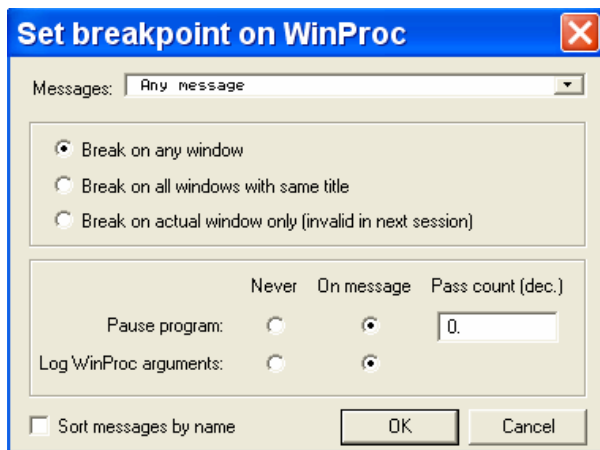
Si la ventana WINDOWS se encuentra vacía, debemos hacer CLICK DERECHO – ACTUALIZE

Handle	Title	Parent	MinProc	ID	Style	ExtStyle	Thread	ClisProc	Class
004907F8	Register	Topmost		14C800C4	00010101		Main	7703E54F	#32770
00410728	Cancel	004907F8		000003EB	50010000	00000004	Main	7703E00E	Button
00000708	Serial	004907F8		0000FFFF	50020000	00000004	Main	7703E592	Static
00F20736		004907F8		000003EB	50010000	00000204	Main	7703E3C4	Edit
011A079C	OK	004907F8		000003EB	50010000	00000004	Main	7703E00E	Button
011E06E2	Name	004907F8		0000FFFF	50020000	00000004	Main	7703E592	Static
000E07CC		004907F8		000003EB	50010000	00000204	Main	7703E3C4	Edit
01840738	CrackMe v1.0	Topmost		00D40718	1CCF0000	00000100	Main	00401128	No need to disasm the code!

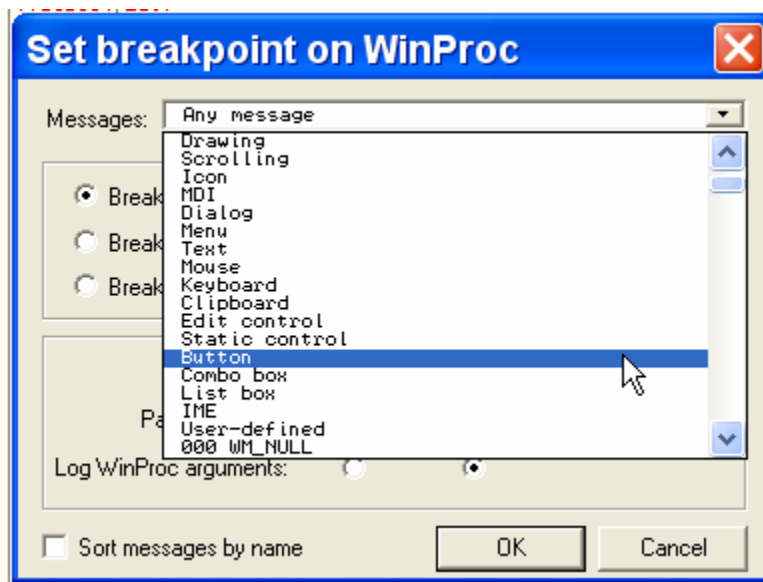
Buscamos el BOTÓN, allí vemos marcada la línea en CLASS dice BUTTON y el titulo es OK como en nuestro BOTON.

Hacemos CLICK DERECHO en esa línea y elegimos MESSAGE BREAKPOINT ON CLASSPROC

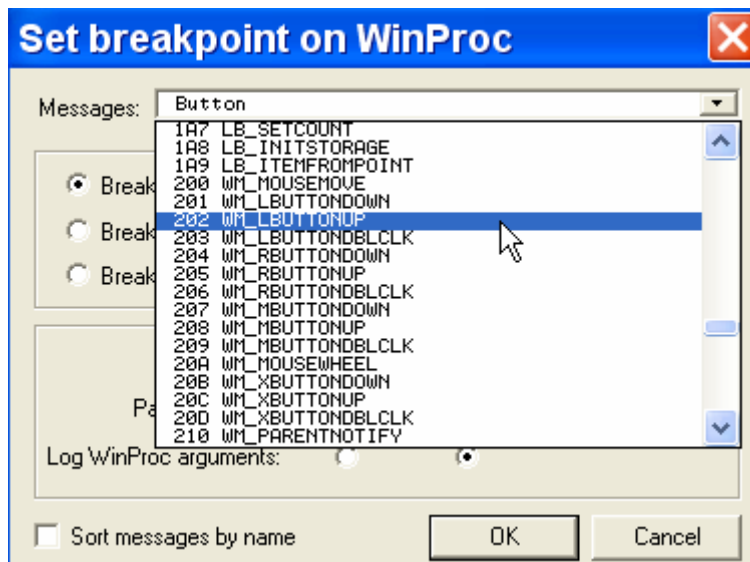
Handle	Title	Parent	MinProc	ID	Style	ExtStyle	Thread	Cl
004907F8	Register	Topmost		14C800C4	00010101		Main	77
00410728	Cancel	004907F8		000003EB	50010000	00000004	Main	77
00000708	Serial	004907F8		0000FFFF	50020000	00000004	Main	77
00F20736		004907F8		000003EB	50010000	00000204	Main	77
011A079C	OK	004907F8		000003EB	50010000	00000004	Main	77
011E06E2	Name	004907F8		0000FFFF	50020000	00000004	Main	77
000E07CC		004907F8		000003EB	50010000	00000204	Main	77
01840738	CrackMe v1.0	Topmost		00D40718	1CCF0000	00000100	Main	00



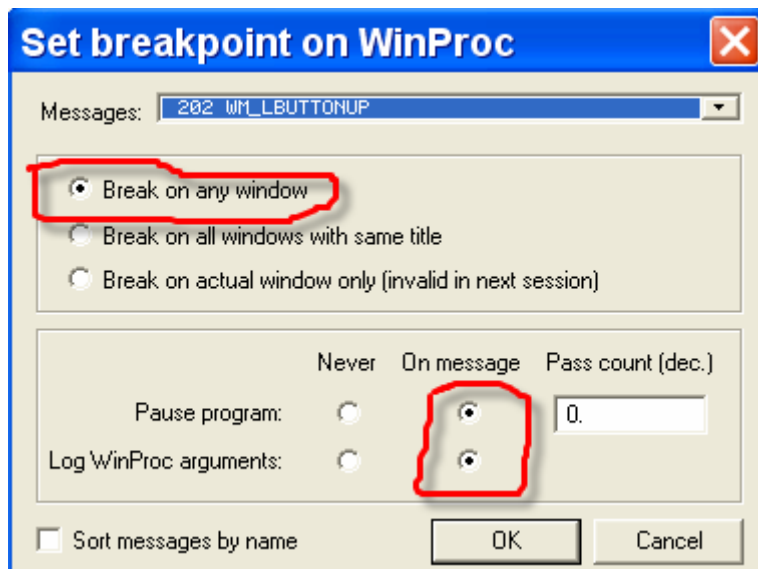
Vemos que nos sale esta ventanita la cual tiene para elegir que tipo de mensaje podemos utilizar si desplegamos el menú MESSAGES



Vemos que hay BMSG genérico para texto, Mouse, portapapeles, botones, cuando uno no sabe bien cual es el mensaje que se utiliza el programa, puede intentar usar las opciones genéricas, pero en mi caso, usare el BMSG exacto, sigo bajando en el menú desplegable, ya que los años me han enseñado que cuando se hace click en un botón, se envía un mensaje WM_LBUTTONDOWN, primero cuando apretamos con el botón izquierdo por eso la L de LEFT y luego WM_LBUTTONUP cuando soltamos el botón izquierdo, así que pondremos el BMSG en este ultimo evento al soltar el botón, que es el 202.



Quedaría así



Elegido el tipo de mensaje 202 WM_LBUTTONDOWN, ponemos la marca en BREAK ON ANY WINDOW, y en PAUSE PROGRAM, que pare ON MESSAGE o sea cuando se ejecute el mensaje que elegimos, y abajo LOG WINPROC ARGUMENTS, ya que nunca viene mal que loguee los argumentos de algo.

Handle	Title	Parent	WinProc	ID	Style	ExtStyle	Thread	ClsProc	Class
004907F8	Register	Topmost			14C800C4	00010101	Main	77D3E54F	#32770
00410728	Cancel	004907F8	000003E8	50010000	50010000	00000004	Main	77D3B000	Button
00000708	Serial	004907F8	0000FFFF	50020000	50020000	00000004	Main	77D3E592	Static
00F20736		004907F8	000003E8	50010000	50010000	00000204	Main	77D3B3C4	Edit
011A079C	OK	004907F8	000003E8	50010000	50010000	00000004	Main	77D3B000	Button
011E06E2	Name	004907F8	0000FFFF	50020000	50020000	00000004	Main	77D3E592	Static
000007CC		004907F8	000003E9	50010000	50010000	00000204	Main	77D3B3C4	Edit
01840738	CrackMe v1.0	Topmost	00D40718	1CCF0000	00000100		Main	00401128	No need to disasm the code!

Vemos que como elegimos en cualquier ventana lo coloco en los dos botones que hay OK y CANCEL, no hay problema, ahora apretemos OK.


```

77D3B00E 8BFF MOV EDI,EDI
77D3B010 55 PUSH EBP
77D3B011 8BEC MOV EBP,ESP
77D3B013 8B4D 08 MOV ECX,DWORD PTR SS:[EBP+8]
77D3B016 56 PUSH ESI
77D3B017 E8 B4D4DFF CALL USER32.77D184D0
77D3B01C 8BF0 MOV ESI,EAX
77D3B01E 85F6 TEST ESI,ESI
77D3B020 74 38 JE SHORT USER32.77D3B05A
77D3B022 8B55 0C MOV EDX,DWORD PTR SS:[EBP+C]
77D3B025 3B15 E800D7Z7 CMP EDX,DWORD PTR DS:[77D700E8]
77D3B02B 77 45 JA SHORT USER32.77D3B072
77D3B02D 33C0 XOR EAX,EAX
77D3B02F 8BCA MOV ECX,EDX
77D3B031 83E1 07 AND ECX,7
77D3B034 40 INC EAX
77D3B035 D3E0 SHL EAX,CL
77D3B037 57 PUSH EDI
77D3B038 8B3D EC00D7Z7 MOV EDI,DWORD PTR DS:[77D700EC]
77D3B03E 8BCA MOV ECX,EDX
77D3B040 C1EB 02 ROR ECX,2

```

```

0012FCB8 77D18734 CALL to Assumed WinProc from USER32.77D18731
0012FCBC 011A079C hWnd = 011A079C ('OK',class='Button',parent=004907F8)
0012FCC0 00000202 Message = WM_LBUTTONDOWN
0012FCC4 00000000 Keys = 0
0012FCC8 00150043 L_X = 67. Y = 21.
0012FCCC 77D3B00E RETURN to USER32.77D3B00E
0012FCD0 DCBAABCD
0012FCD4 00000000
0012FCD8 0012FCD0

```

Aquí es donde se pierden muchos porque dicen que parando con este método, quedan en el medio de la nada ya que la dirección de retorno, no pertenece al programa pero retornar es muy sencillo.

003C0000	00002000				Map	R	R
003D0000	00004000				Priv	RW	RW
003E0000	00002000				Map	R	R
003F0000	00004000				Priv	RW	RW
00400000	00001000	CRACKME	PE header	code	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	00007000				Map	R E	R E

Sabemos que el programa se ejecuta en la sección que comienza en 401000, así que pongo un BPM ON ACCESS en dicha sección para que pare cuando ejecute algo allí.

003D0000	00004000				Priv	RW	RW
003E0000	00002000				Map	R	R
003F0000	00004000				Priv	RW	RW
00400000	00001000	CRACKME	PE header	code	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	00007000				Map	R E	R E
00420000	00001000	SSSensor	PE header	code	Imag	R	RWE
00430000	00001000	SSSensor	.text	code	Imag	R	RWE
00440000	00002000	SSSensor	.rdata	imports,exp	Imag	R	RWE
00450000	00005000	SSSensor	.data	data	Imag	R	RWE
00460000	00001000	SSSensor	.shared	resources	Imag	R	RWE
00470000	00001000	SSSensor	.rsrc	relocations	Imag	R	RWE
00480000	00002000	SSSensor	.reloc	PE header	Imag	R	RWE
58C30000	00001000	COMCTL32			Imag	R	RWE

Al dar RUN paramos allí

Address	Disassembly	Comments
00401253	C8 000000	ENTER 0,0
00401257	53	PUSH EBX
00401258	56	PUSH ESI
00401259	57	PUSH EDI
0040125A	817D 0C 1001	CMP DWORD PTR SS:[EBP+C],110
00401261	74 34	JE SHORT CRACKME.00401297
00401263	817D 0C 1101	CMP DWORD PTR SS:[EBP+C],111
0040126A	74 35	JE SHORT CRACKME.004012A1
0040126C	837D 0C 10	CMP DWORD PTR SS:[EBP+C],10
00401270	0F84 81000000	JE CRACKME.004012F7
00401276	817D 0C 0102	CMP DWORD PTR SS:[EBP+C],201
0040127D	74 0C	JE SHORT CRACKME.0040128B
0040127F	B8 00000000	MOV EAX,0
00401284	5F	POP EDI
00401285	5E	POP ESI
00401286	5B	POP EBX
00401287	C9	LEAVE
00401288	C2 1000	RET 10
0040128B	6A 01	PUSH 1
0040128D	6A 00	PUSH 0
0040128F	FF75 08	PUSH DWORD PTR SS:[EBP+8]
00401292	E8 B5010000	CALL <JMP.&USER32.InvalidRect>
00401297	FF75 08	PUSH DWORD PTR SS:[EBP+8]
0040129A	E8 95010000	CALL <JMP.&USER32.SetFocus>
0040129F	EB E3	JMP SHORT CRACKME.00401284
004012A1	33C0	XOR EAX,EAX
004012A3	817D 10 EB03	CMP DWORD PTR SS:[EBP+10],3EB
004012AA	74 4B	JE SHORT CRACKME.004012F7
004012AC	817D 10 EA03	CMP DWORD PTR SS:[EBP+10],3EA
004012B3	75 3B	JNZ SHORT CRACKME.004012F0
004012B5	6A 0B	PUSH 0B
004012B7	68 8E214000	PUSH CRACKME.0040218E
004012BC	68 E8030000	PUSH 3E8
004012C1	FF75 08	PUSH DWORD PTR SS:[EBP+8]
004012C4	E8 07020000	CALL <JMP.&USER32.GetDlgItemTextA>
004012C9	83F8 01	CMP EAX,1
004012CC	C745 10 EB03	MOV DWORD PTR SS:[EBP+10],3EB
004012D3	72 CC	JB SHORT CRACKME.004012A1

Erase = TRUE
 pRect = NULL
 hWnd
 InvalidateRect
 hWnd
 SetFocus

Count = B (11.)
 Buffer = CRACKME.0040218E
 ControlID = 3E8 (1000.)
 hWnd
 GetDlgItemTextA

SIN QUITAR EL BPM ON ACCESS voy apretando F9 lo cual será como ir ejecutando línea a línea pero solo las instrucciones de esta sección.

33C0	XOR EAX,EAX	
817D 10 EB03	CMP DWORD PTR SS:[EBP+10],3EB	
74 4B	JE SHORT CRACKME.004012F7	
817D 10 EA03	CMP DWORD PTR SS:[EBP+10],3EA	
75 3B	JNZ SHORT CRACKME.004012F0	
6A 0B	PUSH 0B	
68 8E214000	PUSH CRACKME.0040218E	
68 E8030000	PUSH 3E8	
FF75 08	PUSH DWORD PTR SS:[EBP+8]	
E8 07020000	CALL <JMP.&USER32.GetDlgItemTextA>	
83F8 01	CMP EAX,1	
C745 10 EB03	MOV DWORD PTR SS:[EBP+10],3EB	
72 CC	JB SHORT CRACKME.004012A1	
6A 0B	PUSH 0B	
68 7E214000	PUSH CRACKME.0040217E	
68 E9030000	PUSH 3E9	
FF75 08	PUSH DWORD PTR SS:[EBP+8]	
E8 E7010000	CALL <JMP.&USER32.GetDlgItemTextA>	
B8 01000000	MOV EAX,1	
EB 07	JMP SHORT CRACKME.004012F7	
B8 00000000	MOV EAX,0	
EB 80	JMP SHORT CRACKME.00401284	
50	PUSH EAX	

Count = B (11.)
 Buffer = CRACKME.0040218E
 ControlID = 3E8 (1000.)
 hWnd
 GetDlgItemTextA

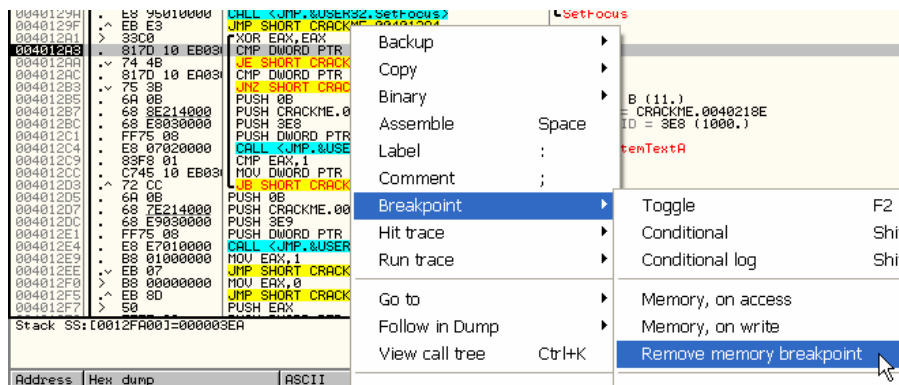
Count = B (11.)
 Buffer = CRACKME.0040217E
 ControlID = 3E9 (1001.)
 hWnd
 GetDlgItemTextA

Result

Vemos que enseguida luego de ejecutar dos veces hasta el RET caemos en las llamadas a GetDlgItemTextA que leen el user y serial que tipeamos, de forma que llegamos al mismo lugar sin haber utilizado BPX en las apis en este caso.

Si el programa no hubiera utilizado apis para entrar el texto, hubiéramos llegado con los BMSG al mismo punto donde ingresara el texto y podremos empezar a trabajar la rutina del serial lo cual veremos en próximas partes.

En el caso que quisiéramos parar al apretar una tecla, podemos intentarlo también borramos el BPM ON ACCESS haciendo CLICK DERECHO-REMOVE MEMORY BREAKPOINT en cualquier parte del listado.



Y vamos a la ventana B y borramos los BMSG anteriores con REMOVE

Address	Module	Active	Disassembly
77D3B00E	USER32	Log "<WinProc>"	MOV EDI,EDI

Ponemos el programa a correr nuevamente y llegamos a la ventana de ingresar el user y serial pero esta vez no ingresamos nada.

Register

Name

Serial

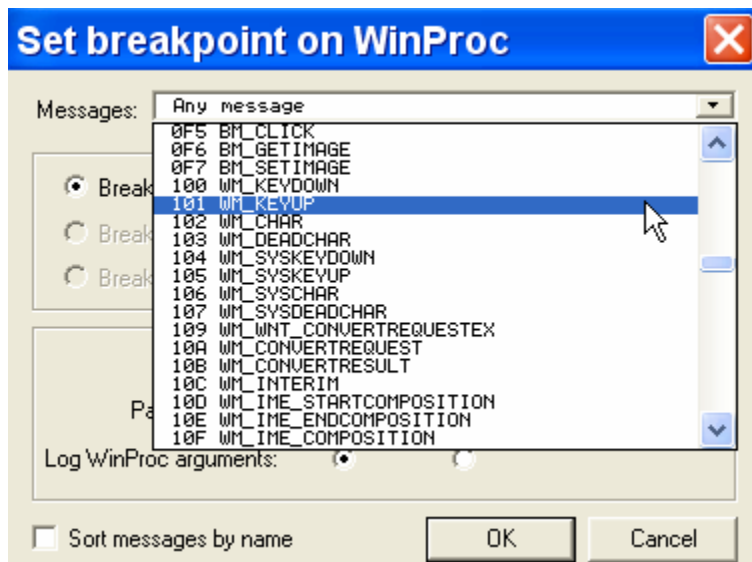
OK

Cancel

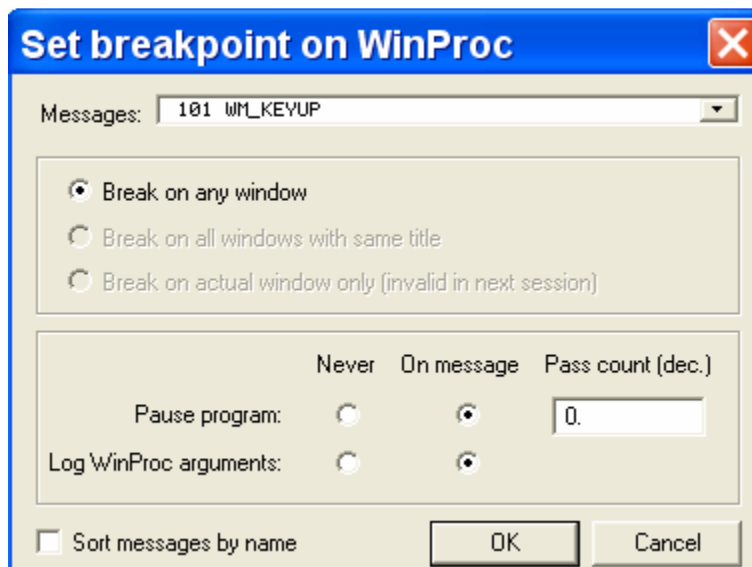
Vamos a la ventana W

Handle	Title	Parent	WinProc	ID	Style	ExtStyle	Thread	ClsProc	Class
004907F8	Register	Topmost			14C800C4	00010101	Main	77D3E54F	#32770
00410728	Cancel	004907F8		000003EB	50010000	00000004	Main	77D3B00E	Button
00000708	Serial	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
00F20736		004907F8		000003EB	50010030	00000204	Main	77D3B3C4	Edit
011A079C	OK	004907F8		000003EA	50010001	00000004	Main	77D3B00E	Button
011E06B2	Name	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
000E07CC		004907F8		000003E9	50010030	00000204	Main	77D3B3C4	Edit
01840738	CrackMe v1.0	Topmost		00D40718	1CCF0000	00000100	Main	00401128	No need to disasm the code*

Y repetimos el procedimiento anterior pero esta vez eligiendo



El tipo de mensaje 101 o sea WM_KEYUP para que pare cuando soltamos la tecla.



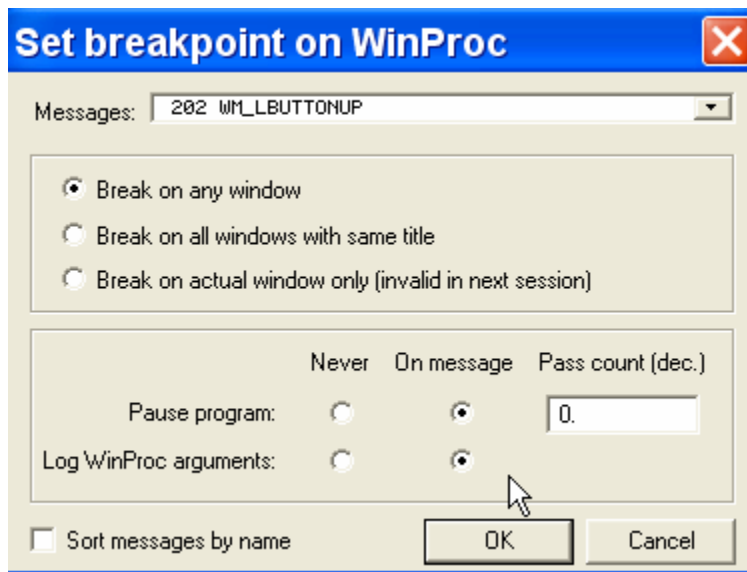
Handle	Title	Parent	WinProc	ID	Style	ExtStyle	Thread	ClsProc	Class
004907F8	Register	Topmost			14C800C4	00010101	Main	77D3E54F	#32770
00410728	Cancel	004907F8		000003EB	50010000	00000004	Main	77D3B00E	Button
00800708	Serial	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
00F20736	OK	004907F8		000003E8	50010000	00000204	Main	77D3B3C4	Edit
011A079C	Name	004907F8		000003EA	50010001	00000004	Main	77D3B00E	Button
011E06B2		004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
000E07CC		004907F8		000003E9	50010000	00000204	Main	77D3B3C4	Edit
01840738	CrackMe v1.0	Topmost		00040718	1CCF0000	00000100	Main	00401128	No need to c

Apretamos una tecla para escribir el nombre en la ventana de user y serial y vemos que en este caso no para, porque el programa no procesa los mensajes de la presión tecla por tecla, pero es bueno saberlo porque hay programas que si lo hacen, y comparan carácter por carácter a medida que tipeamos, y por eso hay que tener en cuenta para esos casos esa opción.

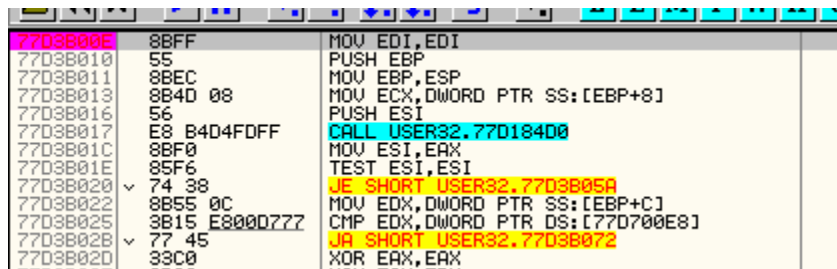
Ahora esto trae otros interrogantes, podemos hacer que OLLY nos liste los mensajes que utiliza el programa sin parar, de forma de luego poder utilizar el BMSG que sabemos que funcionara, pues es utilizado?

Pues es sencillo, lo que tenemos que hacer es utilizar un BMSG que sabemos que funciona, como en este caso el 202 y poner un BMSG en el y que se detenga como hicimos en el primer

caso, quitamos todos los BPX en la ventana B y repetimos hasta que pare al apretar el botón OK por BMSG 202 WM_LBUTTONDOWN

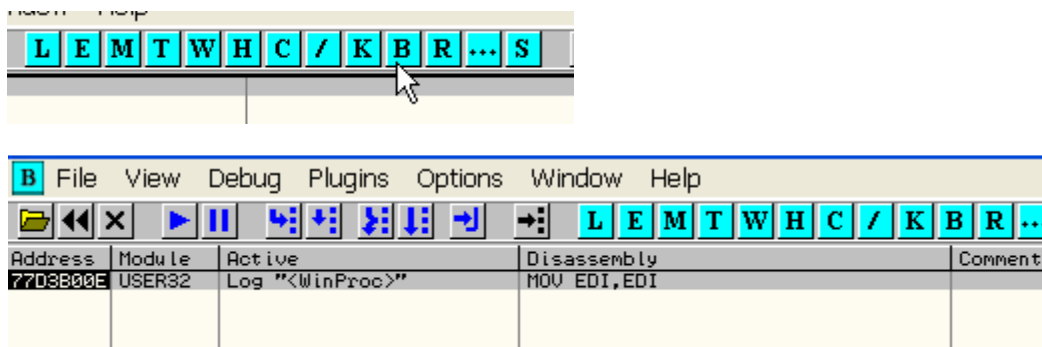


Lo colocamos y apretamos OK en la ventana de ingresar el serial.

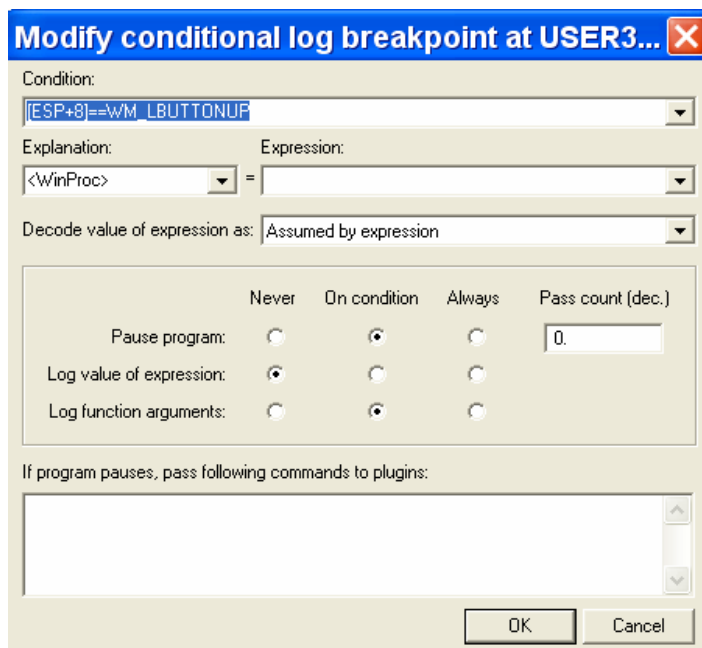
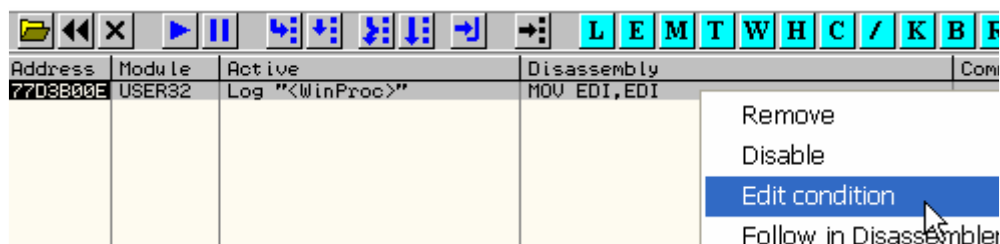


Vemos que allí paro ahora modificaremos las cosas para que loguee todos los mensajes utilizados por los botones.

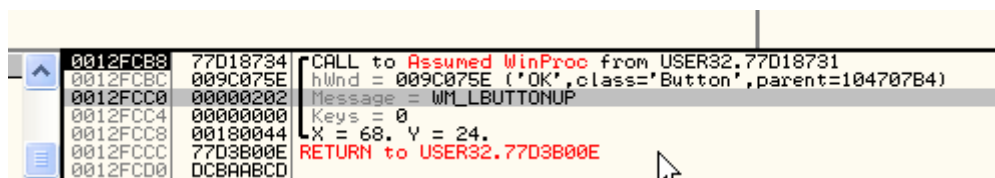
Vamos a la ventana B



Allí esta el BMSG, así que haré CLICK DERECHO-EDIT CONDITION

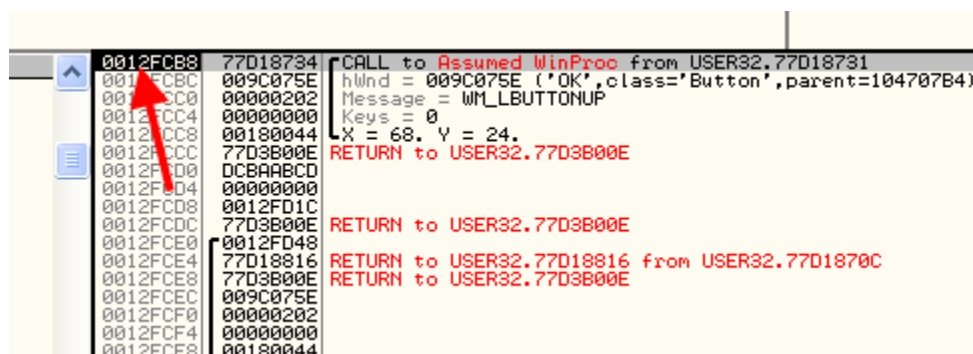


Vemos que el BMSG es un BREAKPOINT CONDICIONAL, cuya condicion es que [esp +8] sea 202 en este caso o sea WM_LBUTTONDOWN, si vemos el stack



Allí esta el valor 202 en [esp +8]

Si tienen dudas que es ESP +8 hacen doble click aquí



En la direccion de la parte superior del stack

```

77D18734 CALL to Assumed WinProc from USER32.77D18731
009C075E hWnd = 009C075E ('OK',class='Button',parent=104707B4)
00000202 Message = WM_LBUTTONDOWN
00000000 Keys = 0
00180044 X = 68, Y = 24.
77D3B00E RETURN to USER32.77D3B00E
DCB8ABCD
00000000
0012FD1C RETURN to USER32.77D3B00E
77D3B00E
0012FD48 RETURN to USER32.77D18816 from USER32.77D1870C
77D18816 RETURN to USER32.77D3B00E
77D3B00E
009C075E

```

Y cambia la numeración relativa a ESP, allí vemos que es "\$ + 8" o "ESP + 8"

Bueno la cuestión es que en [esp+8] esta el numero del BMSG que necesitamos saber así que cambiamos en la ventana del BREAKPOINT.

Modify conditional log breakpoint at USER3...

Condition:

Explanation: Expression:

Decode value of expression as:

	Never	On condition	Always	Pass count (dec.)
Pause program:	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="text" value="0"/>
Log value of expression:	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	
Log function arguments:	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	

If program pauses, pass following commands to plugins:

OK Cancel

Para que loguee el valor que se encuentra en EXPRESSION en este caso [esp + 8], que no pare o sea en PAUSE PROGRAM ponemos NEVER y LOG VALUE OF EXPRESSION y LOG FUNCTION ARGUMENTS pues no esta demás en ambos que los loguee ALWAYS o sea SIEMPRE.

Aceptamos y vamos a colocar nuestro nombre y serial nuevamente al apretar OK miramos el LOG y vemos la lista de mensajes que usa el programa en los botones con sus nombres y números.

```

77D3B00E COND: <WinProc> = 00000087
77D3B00E CALL to Assumed WinProc from USER32.77D18731
hWnd = 00F60728 ('OK',class='Button',parent=01270778)
Message = WM_GETDLGCODE
wParam = 0
lParam = 0
77D3B00E COND: <WinProc> = 00000087
77D3B00E CALL to Assumed WinProc from USER32.77D18731
hWnd = 041507DA ('Cancel',class='Button',parent=01270778)
Message = WM_GETDLGCODE
wParam = 0
lParam = 0
77D3B00E COND: <WinProc> = 000000F4
77D3B00E CALL to Assumed WinProc from USER32.77D18731
hWnd = 00F60728 ('OK',class='Button',parent=01270778)
Message = BM_SETSTYLE
Style = BS_DEFPUSHBUTTON
Redraw = TRUE
77D3B00E COND: <WinProc> = 00000201
77D3B00E CALL to Assumed WinProc from USER32.77D18731
hWnd = 00F60728 ('OK',class='Button',parent=01270778)
Message = WM_LBUTTONDOWN
Keys = MK_LBUTTON
X = 40, Y = 2.
77D3B00E COND: <WinProc> = 00000087
77D3B00E CALL to Assumed WinProc from USER32.77D18731
hWnd = 00F60728 ('OK',class='Button',parent=01270778)
Message = WM_SETFOCUS
hWndLose = 00A9075E (class='Edit',parent=01270778)
lParam = 0
77D3B00E COND: <WinProc> = 000000F3
77D3B00E CALL to Assumed WinProc from USER32.77D18731
hWnd = 00F60728 ('OK',class='Button',parent=01270778)
Message = BM_SETSTATE
Highlight = TRUE
lParam = 0
77D3B00E COND: <WinProc> = 00000200
77D3B00E CALL to Assumed WinProc from USER32.77D18731
hWnd = 00F60728 ('OK',class='Button',parent=01270778)
Message = WM_MOUSEMOVE
Keys = MK_LBUTTON
X = 47, Y = 10.
77D3B00E COND: <WinProc> = 000000F3
77D3B00E CALL to Assumed WinProc from USER32.77D18731
hWnd = 00F60728 ('OK',class='Button',parent=01270778)
Message = BM_SETSTATE
Highlight = TRUE
lParam = 0
77D3B00E COND: <WinProc> = 00000202
77D3B00E CALL to Assumed WinProc from USER32.77D18731
hWnd = 00F60728 ('OK',class='Button',parent=01270778)
Message = WM_LBUTTONUP
Keys = 0
X = 47, Y = 10.
77D3B00E COND: <WinProc> = 000000F3
77D3B00E CALL to Assumed WinProc from USER32.77D18731
hWnd = 00F60728 ('OK',class='Button',parent=01270778)
Message = BM_SETSTATE
Highlight = FALSE
lParam = 0
77D3B00E COND: <WinProc> = 00000008
77D3B00E CALL to Assumed WinProc from USER32.77D18731
hWnd = 00F60728 ('OK',class='Button',parent=01270778)
Message = WM_KILLFOCUS
hWndGet = 01270778 ('Register',class='#32770')
lParam = 0
77D3B00E COND: <WinProc> = 00000002
77D3B00E CALL to Assumed WinProc from USER32.77D18731
hWnd = 00F60728 ('OK',class='Button',parent=01270778)

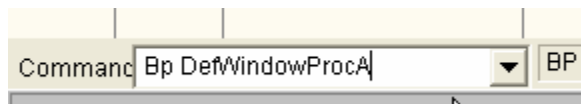
```

Allí vemos como dije que utiliza el 201 WM_LBUTTONDOWN y luego 202 WM_LBUTTONUP, y que no utiliza los de tecla WM_KEYUP o WM_KEYDOWN jeje.

Para hacer un logueador genérico para todo el programa (botones, ventanas etc) se deben hacer condicional breakpoints en las apis que manejan mensajes. Podemos utilizar las apis TranslateMessage y DefWindowProcA que se complementan en su logueo.

Si quieren un reporte completo pueden poner un breakpoint condicional en las dos apis, veamos como se hace eso.

Para hacer rápido tipeo en la commandbar BP TranslateMessage y BP DefWindowProcA.

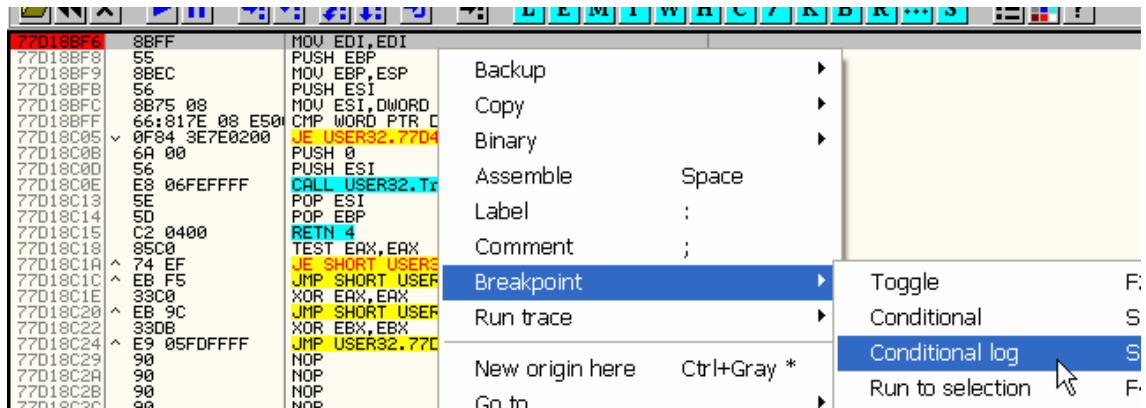


Y

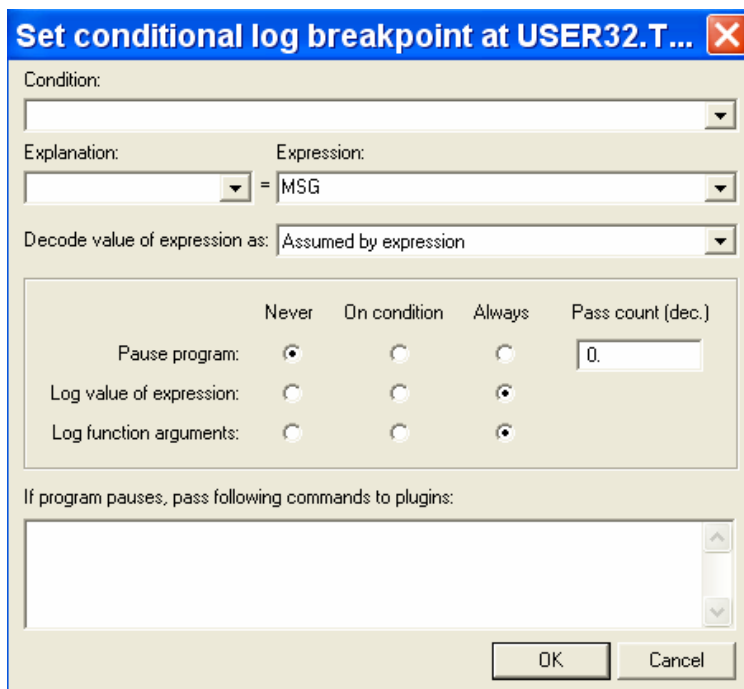


Con eso ponemos dos BPX en las apis ahora los transformaremos en condicionales vamos a la ventana B y hacemos click en el primero FOLLOW IN DISSASSEMBLER.

Address	Module	Active	Disassembly	Comment
77D18BF6	USER32	Always	MOV EDI,EDI	
77D1D4EE	USER32	Always	PUSH 14	

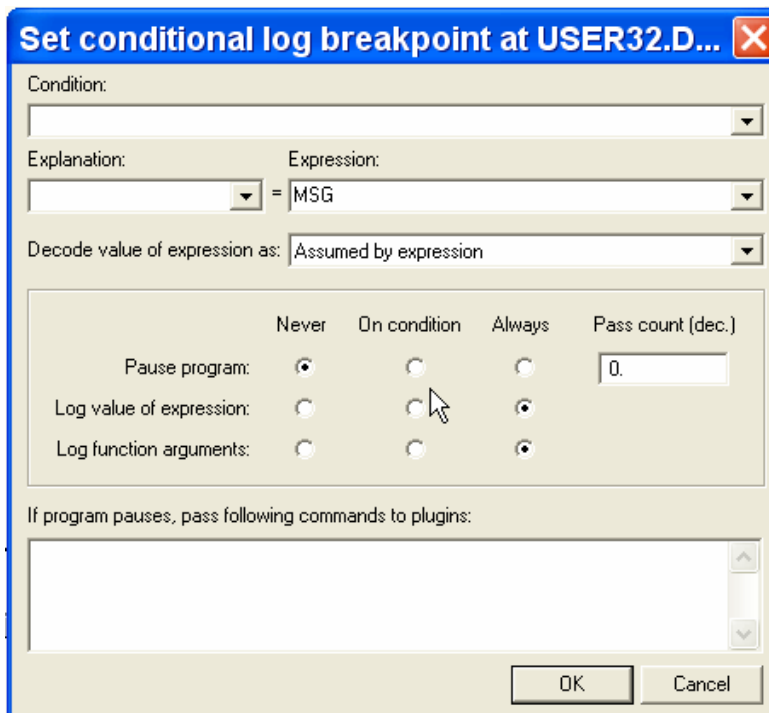


Ahora que estamos en la dirección, colocamos el BREAKPOINT CONDICIONAL LOG.



Poniendo como expresión a loguear MSG que es el valor del tipo de BMSG por ejemplo en WM_LBUTTONDOWN era 202 en ese caso MSG=202

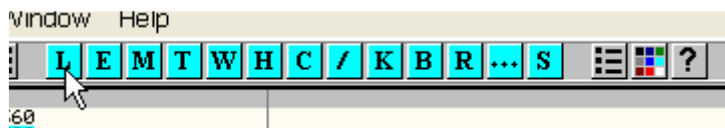
Ponemos que no pare nunca y que loguee siempre y realizamos el mismo trabajo en la otra api.



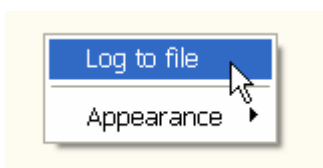
Ya tenemos convertidos los BPX en CONDICIONALES

Address	Module	Active	Disassembly
77D18BF6	USER32	Log	MOV EDI,EDI
77D104EE	USER32	Log	PUSH 14

y si queremos que ya que los resultados serán muchos se loguee a una fila txt en el LOG.



Hacemos click derecho LOG TO FILE



Y elegimos el nombre del archivo de texto que lo guardara, ahora damos RUN.

	Message = WM_ENTERIDLE Source = MSGF_MENU hWnd = 031507F8 (class='#32768',parent=00140742)
77D1D4EE	COND: 00000121
77D1D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
	Message = WM_ENTERIDLE Source = MSGF_MENU hWnd = 031507F8 (class='#32768',parent=00140742)
77D1D4EE	COND: 00000121
77D1D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
	Message = WM_ENTERIDLE Source = MSGF_MENU hWnd = 031507F8 (class='#32768',parent=00140742)
77D1D4EE	COND: 00000085
77D1D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
	Message = WM_NCPAINT Region = E0041935 (region) lParam = 0
77D1D4EE	COND: 00000014
77D1D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
	Message = WM_ERASEBKGD hDC = 9401167D lParam = 0
77D1D4EE	COND: 00000125
77D1D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
	Message = WM_UNINITMENUPOPUP hMenu = 005406F4 ID = 0
77D1D4EE	COND: 0000011F
77D1D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
	Message = WM_MENUSELECT Item = 0, Flags = MF_BYPOSITION MF_SEPARATOR 3 MF_BITMAP MF_OWNERDRAW MF_POPUP MF_ME
77D1D4EE	COND: 00000212
77D1D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
	Message = WM_EXITMENULOOP IsPopUp = FALSE lParam = 0
77D1D4EE	COND: 00000086
77D1D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
	Message = WM_NCACTIVATE Active = FALSE lParam = 0
77D1D4EE	COND: 00000006
77D1D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
	Message = WM_ACTIVATE WA_INACTIVE Minimized = 0 hWnd = NULL
77D1D4EE	COND: 0000001C
77D1D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
	Message = WM_ACTIVATEAPP Activate = FALSE ThreadId = AC0
77D1D4EE	COND: 00000008
77D1D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
	Message = WM_KILLFOCUS hWndGet = NULL lParam = 0
77D18BF6	COND: 0000000F
77D18BF6	CALL to TranslateMessage from CRACKME.0040110C pMsg = WM_PAINT hw = 140742 ("CrackMe v1.0")
77D1D4EE	COND: 0000000F
77D1D4EE	CALL to DefWindowProcA from CRACKME.0040118C hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
	Message = WM_PAINT

Vemos el resultado del LOGUEO completo de todos los mensajes, los cuales se guardan en una fila txt, la cual luego podemos mirar con detenimiento para ver cual BMSG nos conviene mas colocar y en donde ya que el crackme nos muestra cada mensaje en que ventana, boton, etc se aplica.

Sabiendo manejar correctamente los BMSG puede ser una gran ayuda ya lo veremos en casos de NAGS que se utiliza el WM_CLOSE para que pare cuando se cierra la misma y mil usos mas hay mensajes para casi todos los eventos que pueden ocurrir en una ventana.

Espero que lo hayan entendido y les sirva, en la próxima parte terminaremos de hallar el serial valido para el CRACKME DE CRUEHEAD

Hasta la parte 13

Ricardo Narvaja

30 de noviembre de 2005

