



Crackme	BarbecueMe - Xylitol (BBQ)
Misión	Buscar un Name y Serial Hacer un Soft-Keygen
Compilado	Microsoft Visual C++ 5.12
Protección	Check IsDebuggerPresent
Herramientas	OllyDbg 1.10 - RDG v0.7.6 - XN Resource Editor v3.0.1.2 - Resource Hacker v4.2.5 - Editor Hexadecimal HxD v1.7.7.0
Sistema Operativo	Windows Xp SP3
Cracker	QwErTy CLS
Dedicado a	Xylitol - Zelt@ - SoftDat - CracksLatinoS
Descargar Crackme	https://mega.nz/#!Gx8TBAzA!16f-y2W8Y2Cu4gIk6mVsDCi2wh_tW26fH_z6pfqUmFQ

Descomprimimos el archivo (no tiene contraseña),



Menudo pescado...., nada más y nada menos que un escualo con cara de pocos amigos....

ESTUDIANDO LA VÍCTIMA

Lo ejecutamos y nos pide un "Name" y un "Serial" para registrarnos.



Le damos a "About", por si acaso su creador ha querido informarnos de alguna cosa que considere interesante para nuestro menester, y automáticamente se activa una suave y tranquilizante musiquita, un mensaje en cinta descendente/ascendente que únicamente nos da información sobre los datos del autor, y donde también aparece una lista de agradecimientos, que seguro que son sus compañeros de fatigas.....



Una vez efectuada la lectura, salimos de "About", para ello, nos posicionamos con el cursor sobre la ventana que nos muestra el mensaje en cinta, y damos un solo click derecho de ratón, ya que como habrán observado el autor no se ha molestado en insertar ningún "button" para salir de esa ventana.

Tipeamos un "Name" y "Serial" que nos venga en gana para probar suerte, le damos a "Try", y como era de esperar, nos salta el mensaje de Chico malo.

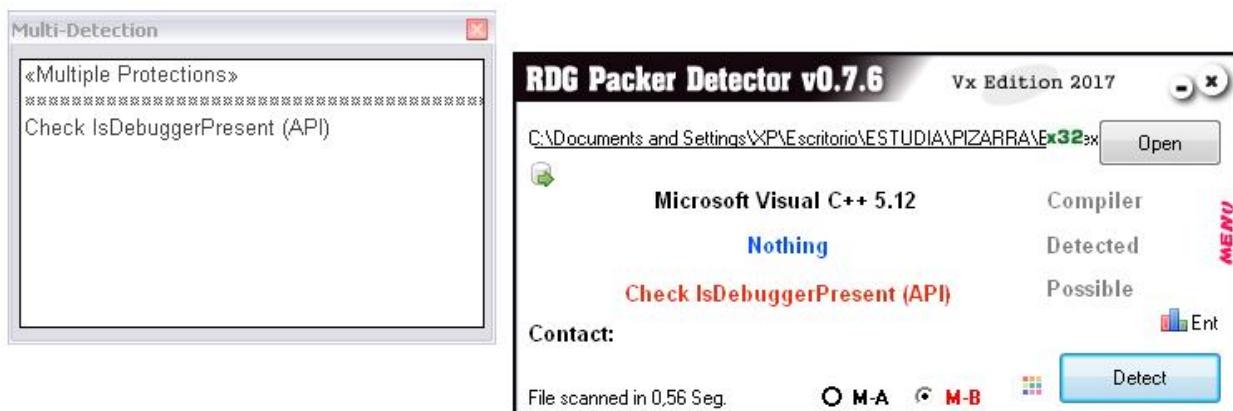


Aceptamos que hemos fallado en nuestro primer intento y salimos del Crackme dándole a "Exit"

CONTINUEMOS ESTUDIANDO LA VÍCTIMA

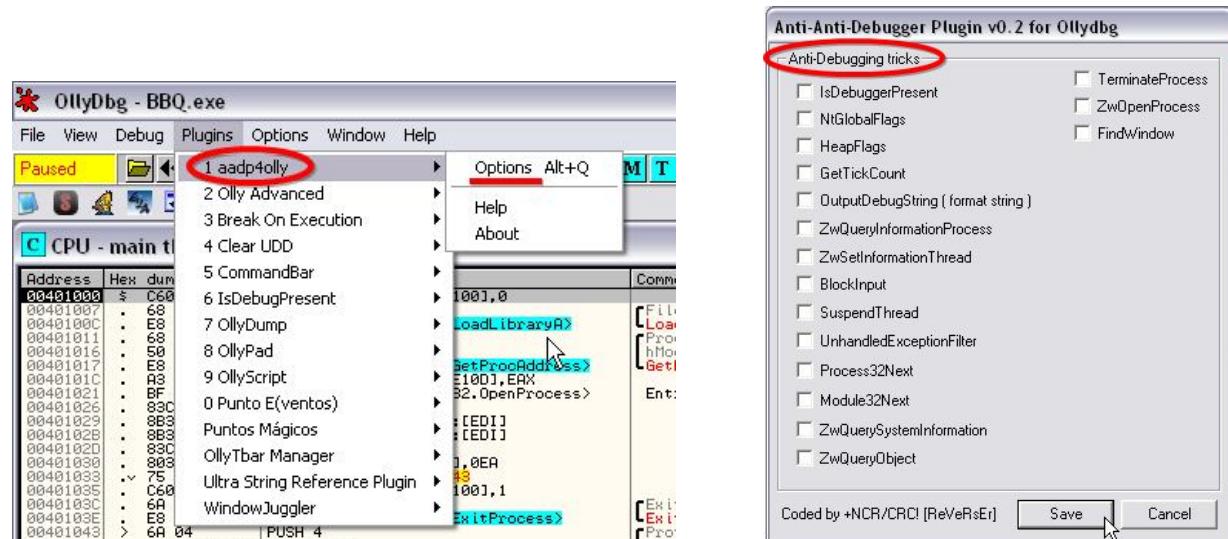
Le pasamos el detector "RDG"

Y en el modo escaneo profundo "**M-B**" nos revela que nos enfrentamos a un compilado en "Microsoft Visual C++ 5.12" y nos avisa de un Posible "**Check IsDebuggerPresent (API)**" como protección.

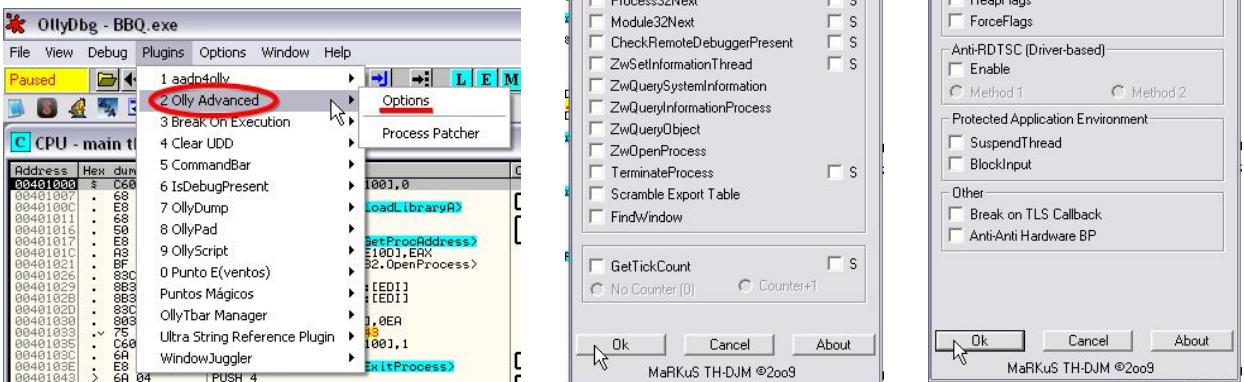


VAMOS A POR ELLA

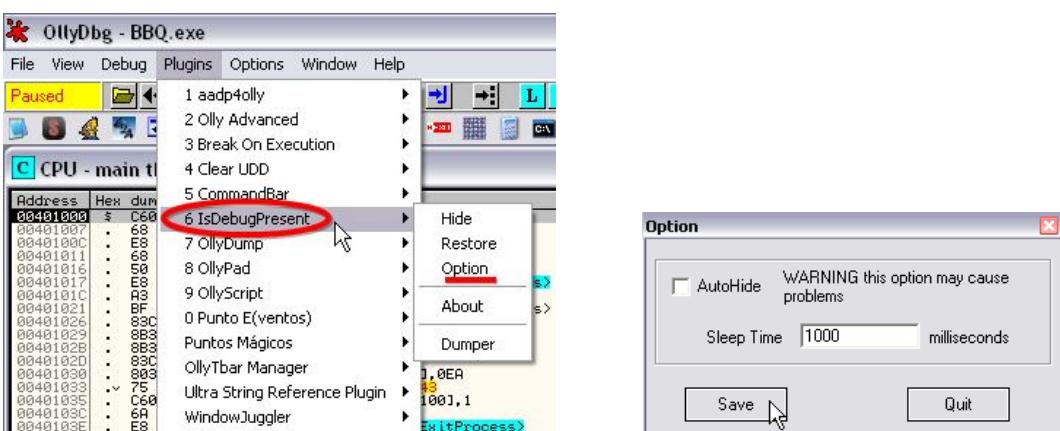
Salimos del "RDG" abrimos el Crackme con Olly, y nos encontramos parados en el OEP address "00401000", verifico que mi armamento de "Plugins antidebugger" esté completamente deshabilitado; (Personalmente me gusta tenerlo a mano, pero por norma solo lo utilizo/habilito cuando realmente lo necesito).



Otro



Y otro



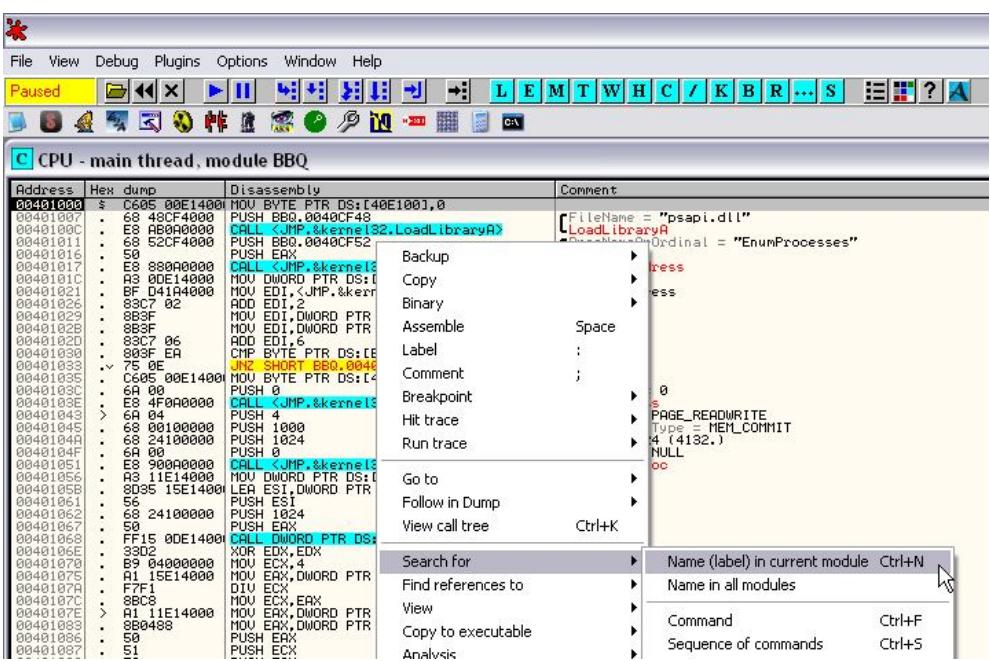
Bien, ahora le damos a "F9" para que corra el Crackme para saber si realmente tiene protección antidebugger y..... efectivamente, algo hay en sus entrañas que detecta que lo estamos debuggeando, ya que descaradamente finaliza el proceso y nos echa fuera sin más explicación.



El detector "RDG", tenía razón, ahora lo primero que debemos hacer es conseguir noquear o saltar esta protección, ya que si no lo logramos no podremos hacer nada de nada.

Pues manos a la obra.

Reiniciamos, y parados de nuevo en el "OEP", en la ventana principal damos click derecho de ratón "Search for - Name (label) in current module",



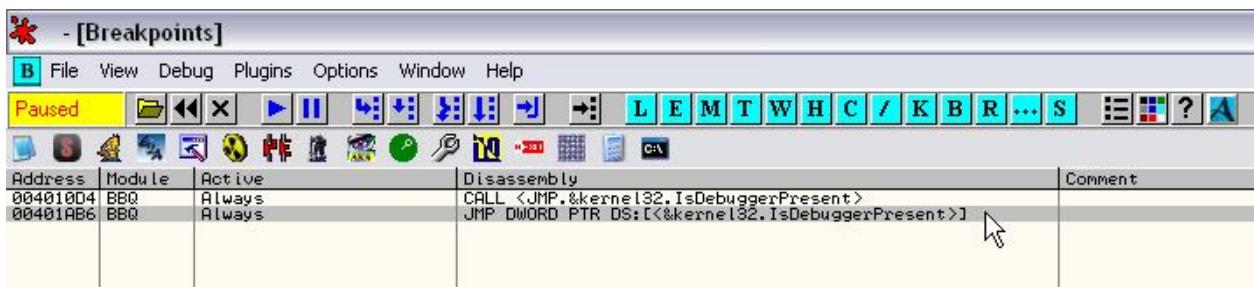
En el listado de funciones APIs que se nos abre y que utiliza este ejecutable, vemos la "IsDebuggerPresent", entre otras interesantes...

Address	Section	Type	Name
004000E4	.rdata	Import	user32.BeginPaint
00400010	.rdata	Import	gdi32.BitBlt
0040003C	.rdata	Import	kernel32.CloseHandle
0040000C	.rdata	Import	gdi32.CreateCompatibleBitmap
00400008	.rdata	Import	gdi32.CreateCompatibleDC
00400014	.rdata	Import	gdi32.CreateFontA
00400018	.rdata	Import	gdi32.CreateSolidBrush
00400040	.rdata	Import	kernel32.CreateThread
0040002C	.rdata	Import	gdi32.DeleteObject
004000E0	.rdata	Import	user32.DialogBoxParamA
004000D0	.rdata	Import	user32.DragTextA
004000D8	.rdata	Import	user32.EndDialog
00400044	.rdata	Import	user32.EndPaint
00400048	.rdata	Import	kernel32.ExitProcess
00400044	.rdata	Import	kernel32.ExitThread
0040004C	.rdata	Import	kernel32.FindResourceA
004000D0	.rdata	Import	user32.GetDC
00400028	.rdata	Import	gdi32.GetDeviceCaps
004000CC	.rdata	Import	user32.GetDlgItemTextA
00400050	.rdata	Import	kernel32.GetModuleHandleA
00400098	.rdata	Import	kernel32.GetProcAddress
004000C8	.rdata	Import	user32.GetWindowRect
00400058	.rdata	Import	kernel32.GlobalAlloc
0040005C	.rdata	Import	kernel32.GlobalFree
00400060	.rdata	Import	comct132.InitCommonControls
00400060	.rdata	Import	kernel32.IsDebuggerPresent
004000C4	.rdata	Import	user32.KillTimer
004000C0	.rdata	Import	user32.LoadIconA
00400064	.rdata	Import	kernel32.LoadLibraryA
00400068	.rdata	Import	kernel32.LoadResource
0040006C	.rdata	Import	kernel32.LockResource
00400084	.rdata	Import	kernel32.lstrcmpA
00400088	.rdata	Import	kernel32.lstrcmpI
0040008C	.rdata	Import	kernel32.lstrcpyA
0040009C	.rdata	Import	kernel32.lstrlenA
0040009C	.rdata	Import	user32.MessageBeep
00401000	text	Export	>ModuleEntryPoint
00400070	.rdata	Import	kernel32.MkDir
00400074	.rdata	Import	kernel32.OpenProcess
00400078	.rdata	Import	kernel32.ReadProcessMemory
00400088	.rdata	Import	user32.ReleaseCapture
00400054	.rdata	Import	kernel32.ResumeThread
0040001C	.rdata	Import	gdi32.SelectObject
00400084	.rdata	Import	user32.SendMessageA
00400020	.rdata	Import	gdi32.SetBkColor
00400024	.rdata	Import	gdi32.SetBkMode
00400034	.rdata	Import	gdi32.SetPixelV
00400030	.rdata	Import	gdi32.SetTextColor
00400090	.rdata	Import	kernel32.SetThreadPriority
00400080	.rdata	Import	user32.SetTimer
004000AC	.rdata	Import	user32.SetWindowTextA
00400088	.rdata	Import	user32.ShowWindow
0040007C	.rdata	Import	kernel32.SizeofResource
00400094	.rdata	Import	kernel32.Sleep
00400090	.rdata	Import	kernel32.SuspendThread
00400080	.rdata	Import	kernel32.VirtualAlloc
004000F0	.rdata	Import	winmm.waveOutClose
004000F4	.rdata	Import	winmm.waveOutGetPosition
004000F8	.rdata	Import	winmm.waveOutOpen
00400100	.rdata	Import	winmm.waveOutPause
00400100	.rdata	Import	winmm.waveOutPrepareHeader
00400104	.rdata	Import	winmm.waveOutReset
00400108	.rdata	Import	winmm.waveOutRestart
0040010C	.rdata	Import	winmm.waveOutUnprepareHeader
00400110	.rdata	Import	winmm.waveOutWrite
00400AE8	.rdata	Import	user32.wsprintfA

Nos posicionamos sobre ella, le damos un click derecho de ratón y le ponemos un "Set Breakpoint on every reference"

00400098	rdata	Import kernel32.GetProcAddress
004000C8	rdata	Import user32.GetWindowRect
00400058	rdata	Import kernel32.GlobalAlloc
0040005C	rdata	Import kernel32.GlobalFree
00400000	rdata	Import comct132.InitCommonControls
00400060	rdata	Import kernel32.IsDebuggerPresent
004000C4	rdata	Import user32.KillTimer
00400064	rdata	Import user32.LoadIconA
00400068	rdata	Import kernel32.LoadLibraryA
0040006C	rdata	Import kernel32.LockResource
00400082	rdata	Import kernel32.LstrCatA
00400083	rdata	Import kernel32.LstrCmpIA
0040008C	rdata	Import kernel32.LstrcpyA
0040009C	rdata	Import kernel32.LstrlenA
004000BC	rdata	Import user32.MessageBoxA
00401000	text	Export <ModuleEntryPoint>
00400070	rdata	Import kernel32.MulDiv
00400074	rdata	Import kernel32.OpenProcess
00400078	rdata	Import kernel32.ReadProcessMemory
004000B8	rdata	Import user32.ReleaseCapture
00400054	rdata	Import kernel32.ResumeThread
004000C1	rdata	Import gdi32.SelectObject
00400084	rdata	Import user32.SendMessageA
00400024	rdata	Import gdi32.SetBKColor
00400034	rdata	Import gdi32.SetBkMode
00400038	rdata	Import gdi32.SetTextIU
00400090	rdata	Import gdi32.SetTextColor
004000B8	rdata	Import kernel32.SetThreadPriority
004000AC	rdata	Import user32.SetTimer
004000A8	rdata	Import user32.ShowWindow
0040007C	rdata	Import kernel32.SizeofResource
00400094	rdata	Import kernel32.Sleep
004000A0	rdata	Import kernel32.SuspendThread
00400080	rdata	Import kernel32.VirtualAlloc

Ahora, nos vamos a la lista de los Breakpoints "B", Olly nos ha colocado dos, una llamada "CALL" y un salto incondicional "JMP".



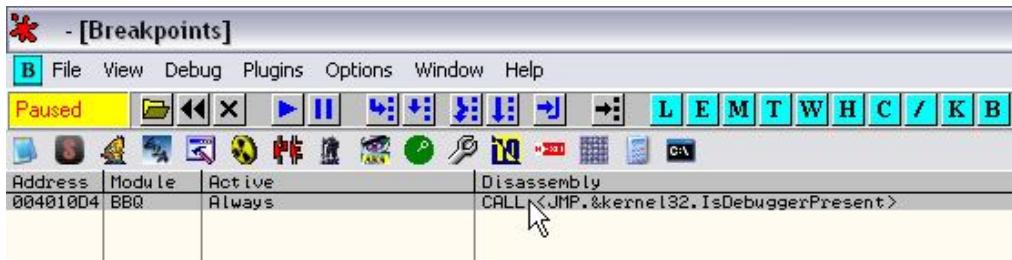
Nos colocamos sobre el "JMP", le damos doble click izquierdo de ratón o si preferimos le damos a "Enter" de nuestro teclado, y aparecemos



en el listado de APIs que utiliza el ejecutable, pues.... gracias por la información.... pero vamos a eliminar este "BP" con "F2" por que no nos interesa para nada parar aquí.

NOTA: Los más avanzados, ya saben que normalmente, por no decir casi siempre, cuando ponemos un " Set Breakpoint on every reference" a una función APIs, y en la lista de "BP" nos marca un punto de parada en una instrucción "JMP", en principio ya lo descartamos de entrada, por que sabemos que nos llevará al listado de APIs, pero en fin, creo que es bueno recordarlo, más que nada para los que se inicien.

Volvemos a nuestro listado de "Breakpoints", nos posicionamos sobre la "CALL", doble click izquierdo de ratón,



y ahora sí que aparecemos en lo que parece la zona que buscamos...je,je,je....

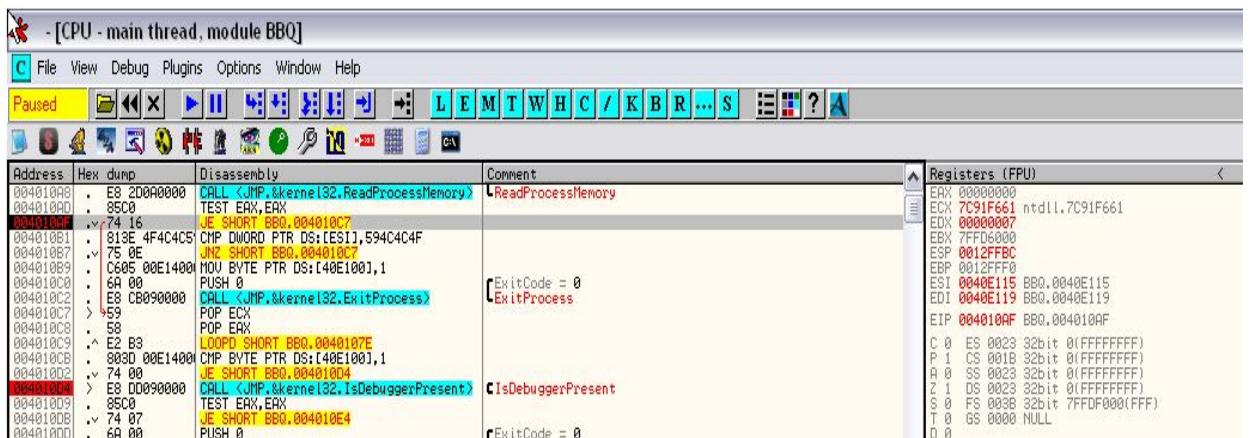
[CPU - main thread, module BBQ]				
Address	Hex dump	Disassembly	Comment	
00401004	> E8 DD090000	CALL <JMP.&kernel32.IsDebuggerPresent>	CIsDebuggerPresent	
00401009	85C0	TEST EAX, EAX		
0040100B	. 74 97	JE SHORT BBQ.004010E4		
0040100D	. 6A 00	PUSH 0		
0040100F	. E8 AE090000	CALL <JMP.&kernel32.ExitProcess>	CExitProcess	
004010E4	> 64:A1 30000000	MOV EAX, DWORD PTR FS:[30]		
004010EA	. 8B40 02	MOV EAX, DWORD PTR DS:[EAX+2]		
004010ED	. 84C0	TEST AL, AL		
004010EF	. 74 09	JE SHORT BBQ.004010FA		
004010F1	. 6A 00	PUSH 0		
004010F3	. E8 9A090000	CALL <JMP.&kernel32.ExitProcess>	CExitProcess	
004010F8	> 64:A1 30000000	JMP SHORT BBQ.004010FA		
004010FA	. EB 00	MOV EAX, DWORD PTR FS:[30]		
00401100	. 8B40 68	MOV EAX, DWORD PTR DS:[EAX+68]		
00401103	. 83E0 70	AND EAX, 70		
00401106	. 74 09	JE SHORT BBQ.00401111		
00401108	. 6A 00	PUSH 0		
0040110A	. E8 83090000	CALL <JMP.&kernel32.ExitProcess>	CExitProcess	
0040110F	. EB 00	JMP SHORT BBQ.00401111		
00401111	> 6A 00	PUSH 0		
00401113	. E8 86090000	CALL <JMP.&kernel32.GetModuleHandleA>	CGetModuleHandleA	
00401118	. A3 27C94000	MOV DWORD PTR DS:[40C927], EAX		
0040111D	. E8 2A0A0000	CALL <JMP.&comctl32.InitCommonControls>	CInitCommonControls	
00401122	. 6A 00	PUSH 0		
00401124	. 68 41114000	PUSH BBQ.00401141		
00401129	. 6A 00	PUSH 0		
0040112B	. 68 E9090000	PUSH 3E9		
00401130	. FF35 27C94000	PUSH DWORD PTR DS:[40C927]		
00401136	. E8 FD080000	CALL <JMP.&user32.DialogBoxParamA>	CDialogBoxParamA	
0040113B	. 50	PUSH EAX	ExitCode	
0040113C	. E8 51090000	CALL <JMP.&kernel32.ExitProcess>	CExitProcess	
00401141	. 55	PUSH EBP		
00401142	. 8BEC	MOV EBP, ESP		
	817D AC 1001	CMP DILMR0 PTR SS:[FRP+C1.110]		

Hacemos un poco de "scrooll" hacia arriba y vemos esto

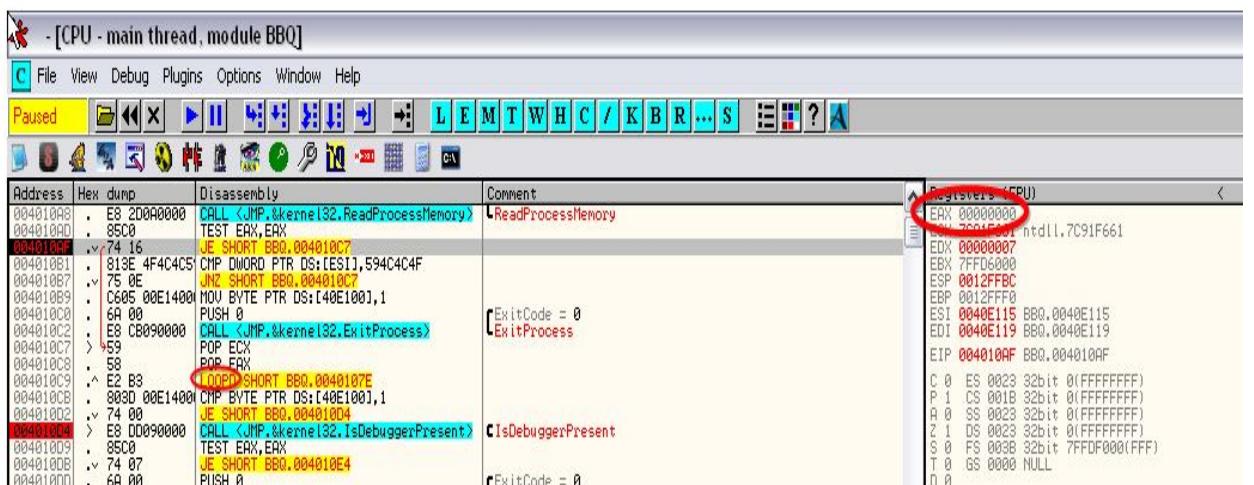
[CPU - main thread, module BBQ]				
Address	Hex dump	Disassembly	Comment	
004010A7	. 50	PUSH EAX		
004010A8	. E8 2D0A0000	CALL <JMP.&kernel32.ReadProcessMemory>	hProcess ReadProcessMemory	
004010AD	. 85C0	TEST EAX, EAX		
004010AF	. 74 16	JE SHORT BBQ.004010C7		
004010B1	. 813E 4F4C4C51	CMP DWORD PTR DS:[ESI1], 594C4C4F		
004010B7	. 75 0E	JNZ SHORT BBQ.004010C7		
004010B9	. C605 00E14000	MOV BYTE PTR DS:[40E100], 1		
004010C0	. 6A 00	PUSH 0		
004010C2	. E8 CB090000	CALL <JMP.&kernel32.ExitProcess>	CExitProcess	
004010C7	. 59	POP ECX		
004010C8	. 58	POP EAX		
004010C9	. E2 B3	LOOPD SHORT BBQ.0040107E		
004010CB	. 803D 00E14000	CMP BYTE PTR DS:[40E100], 1		
004010D2	. 74 00	JE SHORT BBQ.00401004		
004010D4	> E8 DD090000	CALL <JMP.&kernel32.IsDebuggerPresent>	CIsDebuggerPresent	
004010D9	. 85C0	TEST EAX, EAX		
004010DE	. 74 07	JE SHORT BBQ.004010E4		
004010DD	. 6A 00	PUSH 0		
004010DF	. E8 AE090000	CALL <JMP.&kernel32.ExitProcess>	CExitProcess	
004010E4	> 64:A1 30000000	MOV EAX, DWORD PTR FS:[30]		
004010EA	. 8B40 02	MOV EAX, DWORD PTR DS:[EAX+2]		
004010ED	. 84C0	TEST AL, AL		
004010EF	. 74 09	JE SHORT BBQ.004010FA		
004010F1	. 6A 00	PUSH 0		
004010F3	. E8 9A090000	CALL <JMP.&kernel32.ExitProcess>	CExitProcess	
004010F8	. EB 00	JMP SHORT BBQ.004010FA		
004010FA	> 64:A1 30000000	MOV EAX, DWORD PTR FS:[30]		
00401100	. 8B40 68	MOV EAX, DWORD PTR DS:[EAX+68]		
00401103	. 83E0 70	AND EAX, 70		
00401106	. 74 09	JE SHORT BBQ.00401111		
00401108	. 6A 00	PUSH 0		
0040110A	. E8 83090000	CALL <JMP.&kernel32.ExitProcess>	CExitProcess	
0040110F	. EB 00	JMP SHORT BBQ.00401111		
00401111	> 6A 00	PUSH 0		
00401113	. E8 86090000	CALL <JMP.&kernel32.GetModuleHandleA>	CGetModuleHandleA	
00401118	. A3 27C94000	MOV DWORD PTR DS:[40C927], EAX		
0040111D	. E8 2A0A0000	CALL <JMP.&comctl32.InitCommonControls>	CInitCommonControls	
00401122	. 6A 00	PUSH 0		
00401124	. 68 41114000	PUSH BBQ.00401141		
00401129	. 6A 00	PUSH 0		
0040112B	. 68 E9090000	PUSH 3E9		
00401130	. FF35 27C94000	PUSH DWORD PTR DS:[40C927]		
00401136	. E8 FD080000	CALL <JMP.&user32.DialogBoxParamA>	CDialogBoxParamA	
0040113B	. 50	PUSH EAX	ExitCode	
0040113C	. E8 51090000	CALL <JMP.&kernel32.ExitProcess>	CExitProcess	
00401141	. 55	PUSH EBP		
00401142	. 8BEC	MOV EBP, ESP		
	817D AC 1001	CMP DILMR0 PTR SS:[FRP+C1.110]		

Muyyyyyyy interesante, un "**LOOP**", y una serie de saltos condicionales, algunos de los cuales están precedidos de comparaciones y testeos, de los que en función de su resultado nos llevan directos a "**ExitProcces**", que es lo que queremos evitar.

Pues, vamos a poner otro "breakpoint" con "**F2**" en la address "**004010AF**" que es donde está el primer salto condicional:



Damos "run" para que corra el Crackme, y Olly para en nuestro primer punto de parada.



Aquí vemos claramente que estamos después del primer testeo donde verifica que si el valor del registro **EAX** da como resultado 0, el salto condicional "**JE**" nos llevará a la address "**004010C7**", por lo tanto este salto no nos afecta y vamos a dejarlo tal cual ya que iremos a parar después del primer "**ExitProcess**", pero unas líneas más abajo, en la address "**00401009**" tenemos el "**LOOP**" sospechoso que nos lleva más arriba de donde estamos, exactamente a la address "**0040107E**", traceamos hasta él y ahora vemos esto.

- [CPU - main thread, module BBQ]

File View Debug Plugins Options Window Help

Paused L E M T W H C / K B R ... S

Address	Hex dump	Disassembly	Comment
0040106E	33D2	XOR EDX, EDX	
00401070	B9 04000000	MOV ECX, 4	
0040107A	A1 15E14000	MOV EBX, DWORD PTR DS:[40E115]	
0040107C	F7F1	DIV ECX	
0040107E	8B88	MOV ECX, EAX	
00401083	> A1 11E14000	MOV EBX, DWORD PTR DS:[40E111]	
00401085	8B8488	MOV EBX, DWORD PTR DS:[EBX+ECX*4]	
00401086	50	PUSH EBX	
00401087	51	PUSH ECX	
00401088	50	PUSH EBX	
00401089	6A 00	PUSH 0	
0040108B	6A 10	PUSH 10	
0040108D	E8 42000000	CALL QWORD PTR _OpenProcess@4	
00401092	80D5 15E14000	LEA ESI, DWORD PTR DS:[40E115]	
00401093	80D3 19E14000	LEA EDI, DWORD PTR DS:[40E119]	
00401095	57	PUSH EDI	
00401096	6A 04	PUSH 4	
004010A1	56	PUSH ESI	
004010A2	68 4B064B00	PUSH 4B064B	
004010A7	50	PUSH EBX	
004010A8	E8 200A0000	CALL QWORD PTR _ReadProcessMemory@4	
004010AD	85C0	TEST EAX, EAX	
004010B0	> 74 16	JE SHORT BBQ.004010C7	
004010B1	813E 4F4C4C5	CMP DWORD PTR DS:[ESI], 594C4C4F	
004010B7	> 75 0E	JNZ SHORT BBQ.004010C7	
004010B9	C605 00E14000	MOV BYTE PTR DS:[40E100], 1	
004010C0	6A 00	PUSH 0	
004010C2	> 59	POP ECX	
004010C3	58	POP EBX	
004010C9	E2 B3	LOOPD SHORT BBQ.0040107E	
004010CB	80D5 00E14000	CMP BTYE PTR DS:[40E100], 1	
004010D2	> 74 00	JE SHORT BBQ.004010D4	
004010D4	> E8 0D090000	CALL QWORD PTR _IsDebuggerPresent@4	cIsDebuggerPresent
004010D9	85C0	TEST EAX, EAX	
004010DB	> 74 07	JE SHORT BBQ.004010E4	

De lo que deducimos que va a pasar de nuevo por los dos saltos condicionales "JE" y "JNZ" para decidir algo, y en función de los resultados, nos dejará continuar o en otro caso nos llevará a "ExitProcess".

Pues con que nuestra posición actual ya ha rebasado esos dos saltos, y la intención es de no volver atrás por si las moscas.... vamos a cortar por lo sano el "LOOP".

Dos clicks izquierdo de ratón sobre la misma instrucción "LOOP"

- [CPU - main thread, module BBQ]

File View Debug Plugins Options Window Help

Paused L E M T W H C / K B R ... S

Address	Hex dump	Disassembly	Comment
0040106E	33D2	XOR EDX, EDX	
00401070	B9 04000000	MOV ECX, 4	
00401075	A1 15E14000	MOV EBX, DWORD PTR DS:[40E115]	
0040107A	F7F1	DIV ECX	
0040107C	8B88	MOV ECX, EAX	
0040107E	> A1 11E14000	MOV EBX, DWORD PTR DS:[40E111]	
00401083	8B8488	MOV EBX, DWORD PTR DS:[EBX+ECX*4]	
00401086	50	PUSH EBX	
00401087	51	PUSH ECX	
00401088	50	PUSH EBX	
00401089	6A 00	PUSH 0	
0040108B	6A 10	PUSH 10	
0040108D	E8 42000000	CALL QWORD PTR _OpenProcess@4	
00401092	80D5 15E14000	LEA ESI, DWORD PTR DS:[40E115]	
00401093	80D3 19E14000	LEA EDI, DWORD PTR DS:[40E119]	
00401095	57	PUSH EDI	
00401096	6A 04	PUSH 4	
004010A1	56	PUSH ESI	
004010A2	68 4B064B00	PUSH 4B064B	
004010A7	50	PUSH EBX	
004010A8	E8 200A0000	CALL QWORD PTR _ReadProcessMemory@4	
004010AD	85C0	TEST EAX, EAX	
004010B0	> 74 16	JE SHORT BBQ.004010C7	
004010B1	813E 4F4C4C5	CMP DWORD PTR DS:[ESI], 594C4C4F	
004010B7	> 75 0E	JNZ SHORT BBQ.004010C7	
004010B9	C605 00E14000	MOV BYTE PTR DS:[40E100], 1	
004010C0	6A 00	PUSH 0	
004010C2	> 59	POP ECX	
004010C3	58	POP EBX	
004010C9	E2 B3	LOOPD SHORT BBQ.0040107E	
004010CB	80D5 00E14000	CMP BTYE PTR DS:[40E100], 1	
004010D2	> 74 00	JE SHORT BBQ.004010D4	
004010D4	> E8 0D090000	CALL QWORD PTR _IsDebuggerPresent@4	cIsDebuggerPresent
004010D9	85C0	TEST EAX, EAX	

Y la cambiamos por un "Nop"

Assemble at 004010C9

BytesHead => BBB.004010C9
BytesToRead = 4

Address	Hex dump	Disassembly
004010C9	C7 0C E1001,1	Nop
004010D0	004010D0	
004010D1	004010D1	
004010D2	004010D2	
004010D3	004010D3	
004010D4	004010D4	

Le damos a "Assemble" para que nos guarde el cambio

004010A2	.	68 4B064B00	PUSH 4B064B	pBaseAddress = 4B064B
004010A7	.	50	PUSH EAX	hProcess
004010A8	.	E8 200A0000	CALL <JMP.&kernel32.ReadProcessMemory>	ReadProcessMemory
004010A9	.	85C0	TEST EAX,EAX	
004010A9F	.~	74 16	JE SHORT BB0.004010C7	
004010B1	.	813E 4F4C4C5	CMP DWORD PTR DS:[ESI],594C4C4F	
004010B7	.~	75 0E	JNZ SHORT BB0.004010C7	
004010B9	.	C605 00E1400	MOV BYTE PTR DS:[40E100],1	
004010C0	.	6A 00	PUSH 0	
004010C2	.	E8 CB090000	CALL <JMP.&kernel32.ExitProcess>	ExitCode = 0 ExitProcess
004010C7	>	59	POP ECX	
004010C8	.	C0	POP EAX	
004010C9	90	NOP		
004010CA	90	NOP		
004010CB	.	8030 00E1400	CMP BYTE PTR DS:[40E100],1	
004010D2	.~	74 00	JE SHORT BB0.004010D4	
004010D4	>	E8 00090000	CALL <JMP.&kernel32.IsDebuggerPresent>	IsDebuggerPresent
004010D9	.	85C0	TEST EAX,EAX	
004010DB	.~	74 07	JE SHORT BB0.004010E4	
004010DD	.	6A 00	PUSH 0	
004010DF	.	E8 AE090000	CALL <JMP.&kernel32.ExitProcess>	ExitCode = 0 ExitProcess
004010E4	>	64:A1 300000	MOV EAX,DWORD PTR FS:[30]	
004010E9	.	8B40 02	MOV EAX,DWORD PTR DS:[EAX+2]	

Con nuestro "**LOOP**" ahora nopeado, hemos sorteado la primera trampa. Seguimos traceando hasta el siguiente salto condicional "**JE**" address 004010D2, donde vemos que nos antecede un "**CMP**" y nos precede una llamada "**CALL**" donde se encuentra la Apis "**IsDebuggerPresent**"

004010B9	.	C605 00E1400	MOV BYTE PTR DS:[40E100],1	ExitCode = 0 ExitProcess
004010C0	.	6A 00	PUSH 0	
004010C2	.	E8 CB090000	CALL <JMP.&kernel32.ExitProcess>	
004010C7	>	59	POP ECX	
004010C8	.	58	POP EAX	
004010C9	90	NOP		
004010CA	90	NOP		
004010CB	.	8030 00E1400	CMP BYTE PTR DS:[40E100],1	
004010D2	.~	74 00	JE SHORT BB0.004010D4	
004010D4	>	E8 00090000	CALL <JMP.&kernel32.IsDebuggerPresent>	IsDebuggerPresent
004010D9	.	85C0	TEST EAX,EAX	
004010DB	.~	74 07	JE SHORT BB0.004010E4	
004010DD	.	6A 00	PUSH 0	
004010DF	.	E8 AE090000	CALL <JMP.&kernel32.ExitProcess>	ExitCode = 0 ExitProcess
004010E4	>	64:A1 300000	MOV EAX,DWORD PTR FS:[30]	

pero como buenos observadores que somos, deducimos que si entramos dentro de esa "**CALL** address 004010D4", simplemente nos mostrará donde se encuentra la función "**IsDebuggerPresent**" dentro del ejecutable y nada más, por la sencilla razón de que no la antecede ningún "**PUSH 0**" - **ExitCode = 0** - **ExitProcess**", como SÍ sucede en las demás "**CALL**" que debemos evitar.

CPU - main thread, module BBB				
Address	Hex dump	Disassembly	Comment	
0040109E	.	57	PUSH EDI	pBytesRead => BB0.0040E119
0040109F	.	6A 04	PUSH 4	BytesToRead = 4
004010A1	.	56	PUSH ESI	Buffer => BB0.0040E115
004010A2	.	68 4B064B00	PUSH 4B064B	pBaseAddress = 4B064B
004010A7	.	50	PUSH ERX	hProcess
004010A8	.	E8 200A0000	CALL <JMP.&kernel32.ReadProcessMemory>	ReadProcessMemory
004010AD	.	85C0	TEST ERX,ERX	
004010B1	.~	74 16	JE SHORT BB0.004010C7	
004010B7	.	813E 4F4C4C5	CMP DWORD PTR DS:[ESI],594C4C4F	
004010B9	.~	75 0E	JNZ SHORT BB0.004010C7	
004010B9	.	C605 00E1400	MOV BYTE PTR DS:[40E100],1	
004010C0	.	6A 00	PUSH 0	
004010C2	.	E8 CB090000	CALL <JMP.&kernel32.ExitProcess>	ExitCode = 0 ExitProcess
004010C7	>	59	POP ECX	
004010C8	.	58	POP EAX	
004010C9	90	NOP		
004010CA	90	NOP		
004010CB	.	8030 00E1400	CMP BYTE PTR DS:[40E100],1	
004010D2	.~	74 00	JE SHORT BB0.004010D4	
004010D4	>	E8 00090000	CALL <JMP.&kernel32.IsDebuggerPresent>	IsDebuggerPresent
004010D9	.	85C0	TEST EAX,EAX	
004010DB	.~	74 07	JE SHORT BB0.004010E4	
004010DD	.	6A 00	PUSH 0	
004010DF	.	E8 AE090000	CALL <JMP.&kernel32.ExitProcess>	ExitCode = 0 ExitProcess
004010E4	>	64:A1 300000	MOV EAX,DWORD PTR FS:[30]	
004010E9	.	8B40 02	MOV EAX,DWORD PTR DS:[EAX+2]	
004010ED	.	84C0	TEST AL,AL	
004010EF	.~	74 09	JE SHORT BB0.004010FA	
004010F1	.	6A 00	PUSH 0	
004010F3	.	E8 90090000	CALL <JMP.&kernel32.ExitProcess>	ExitCode = 0 ExitProcess
004010F8	.~	EB 00	JMP SHORT BB0.004010FA	
004010FA	.	64:A1 300000	MOV EAX,DWORD PTR FS:[30]	
00401100	.	8B40 68	MOV EAX,DWORD PTR DS:[EAX+68]	
00401103	.	83E9 70	AND EAX,70	
00401106	.~	74 09	JE SHORT BB0.00401111	
00401108	.	6A 00	PUSH 0	
00401109	.	E8 90090000	CALL <JMP.&kernel32.ExitProcess>	ExitCode = 0 ExitProcess
0040110F	.	EB 00	JMP SHORT BB0.00401111	
00401111	.	6A 00	PUSH 0	
00401113	.	E8 86090000	CALL <JMP.&kernel32.GetModuleHandleA>	hModule = NULL
00401118	.	A8 27C94000	MOV DWORD PTR DS:[40C927],ERX	GetModuleHandleA
0040111D	.	E8 20000000	CALL <JMP.&comctl32.InitCommonControls>	InInitCommonControls
00401122	.	6A 00	PUSH 0	IParam = NULL
00401124	.	E8 41114000	PUSH BB0.00401141	DlgProc = BB0.00401141
00401129	.	6A 00	PUSH 0	hOwner = NULL

Vamos a comprobarlo..., un traceo más con "F8" hasta la "**CALL**" address 004010D", entramos con "F7" y efectivamente Olly nos muestra donde se

encuentra la función "**IsDebuggerPresent**" dentro del ejecutable, que si recordamos.... es el mismo punto exacto donde Olly también nos colocó automáticamente el segundo "**BP**", y que nosotros eliminamos por que dijimos que no nos interesaba para nada parar aquí. (**JMP**)

Address	Hex dump	Disassembly	Comment
00401A2B	90	NOP	
00401A2C	FF25 E8A04000	JMP DWORD PTR DS:[<&user32.wsprintfA>]	user32.wsprintfA
00401A32	FF25 E4004000	JMP DWORD PTR DS:[<&user32.BeginPaint>]	user32.BeginPaint
00401A38	FF25 E0A04000	JMP DWORD PTR DS:[<&user32.DialogBoxParamA>]	user32.DialogBoxParamA
00401A3E	FF25 DCA04000	JMP DWORD PTR DS:[<&user32.DrawTextA>]	user32.DrawTextA
00401A44	FF25 D0A04000	JMP DWORD PTR DS:[<&user32.EndDialog>]	user32.EndDialog
00401A48	FF25 D4004000	JMP DWORD PTR DS:[<&user32.EndPaint>]	user32.EndPaint
00401A50	FF25 D0004000	JMP DWORD PTR DS:[<&user32.GetDC>]	user32.GetDC
00401A56	FF25 CCA04000	JMP DWORD PTR DS:[<&user32.GetDlgItemTextA>]	user32.GetDlgItemTextA
00401A5C	FF25 C8004000	JMP DWORD PTR DS:[<&user32.GetWindowRect>]	user32.GetWindowRect
00401A62	FF25 C4004000	JMP DWORD PTR DS:[<&user32.KillTimer>]	user32.KillTimer
00401A68	FF25 C0004000	JMP DWORD PTR DS:[<&user32.LoadIconA>]	user32.LoadIconA
00401A6E	FF25 BCA04000	JMP DWORD PTR DS:[<&user32.MessageBoxA>]	user32.MessageBoxA
00401A74	FF25 B8004000	JMP DWORD PTR DS:[<&user32.ReleaseCapture>]	user32.ReleaseCapture
00401A7A	FF25 B4004000	JMP DWORD PTR DS:[<&user32.SendMessageA>]	user32.SendMessageA
00401A80	FF25 B0004000	JMP DWORD PTR DS:[<&user32.SetTimer>]	user32.SetTimer
00401A86	FF25 ACA04000	JMP DWORD PTR DS:[<&user32.SetWindowTextA>]	user32.SetWindowTextA
00401A8C	FF25 A8004000	JMP DWORD PTR DS:[<&user32.ShowWindowA>]	user32.ShowWindowA
00401A92	FF25 48004000	JMP DWORD PTR DS:[<&kernel32.ExitProcess>]	kernel32.ExitProcess
00401A98	FF25 4C004000	JMP DWORD PTR DS:[<&kernel32.FindResourceA>]	kernel32.FindResourceA
00401A9E	FF25 50004000	JMP DWORD PTR DS:[<&kernel32.GetModuleHandleA>]	kernel32.GetModuleHandleA
00401AA4	FF25 58004000	JMP DWORD PTR DS:[<&kernel32.GetProcAddress>]	kernel32.GetProcAddress
00401AAA	FF25 58004000	JMP DWORD PTR DS:[<&kernel32.GlobalAlloc>]	kernel32.GlobalAlloc
00401AB0	FF25 5CA04000	JMP DWORD PTR DS:[<&kernel32.GlobalFree>]	kernel32.GlobalFree
00401AB6	FF25 60004000	JMP DWORD PTR DS:[<&kernel32.IsDebuggerPresent>]	kernel32.IsDebuggerPresent
00401ABC	FF25 64004000	JMP DWORD PTR DS:[<&kernel32.LoadLibraryA>]	kernel32.LoadLibraryA
00401AC8	FF25 68004000	JMP DWORD PTR DS:[<&kernel32.LoadResource>]	kernel32.LoadResource
00401ACE	FF25 6CA04000	JMP DWORD PTR DS:[<&kernel32.LockResource>]	kernel32.LockResource
00401AC8	FF25 70004000	JMP DWORD PTR DS:[<&kernel32.MulDiv>]	kernel32.MulDiv
00401AD4	FF25 74004000	JMP DWORD PTR DS:[<&kernel32.OpenProcess>]	kernel32.OpenProcess
00401ADH	FF25 78004000	JMP DWORD PTR DS:[<&kernel32.ReadProcessMemory>]	kernel32.ReadProcessMemory
00401AE0	FF25 7CA04000	JMP DWORD PTR DS:[<&kernel32.SizeofResourceA>]	kernel32.SizeofResource
00401AE6	FF25 80004000	JMP DWORD PTR DS:[<&kernel32.VirtualAlloc>]	kernel32.VirtualAlloc
00401AF2	FF25 84004000	JMP DWORD PTR DS:[<&kernel32.IstrcatA>]	kernel32.IstrcatA
00401AF8	FF25 88004000	JMP DWORD PTR DS:[<&kernel32.IstrncpyA>]	kernel32.IstrncpyA
00401AFE	FF25 8CA04000	JMP DWORD PTR DS:[<&kernel32.IstrcpyA>]	kernel32.IstrcpyA
00401B04	FF25 9CA04000	JMP DWORD PTR DS:[<&kernel32.IstrlenA>]	kernel32.IstrlenA
00401B0A	FF25 10004000	JMP DWORD PTR DS:[<&gdi32.BitBlt>]	GDI32.BitBlt
00401B0F	FF25 0CA04000	JMP DWORD PTR DS:[<&gdi32.CreateCompatibleBitmap>]	GDI32.CreateCompatibleBitmap
00401B10	FF25 08004000	JMP DWORD PTR DS:[<&gdi32.CreateCompatibleDC>]	GDI32.CreateCompatibleDC
00401B16	FF25 14004000	JMP DWORD PTR DS:[<&gdi32.CreateFontA>]	GDI32.CreateFontA

Una vez más, le damos gracias a Olly por la información pero continuamos traceando con "F8" hasta llegar al "**RETN**", (indicador que vamos a salir de la "**CALL**").

Address	Hex dump	Disassembly
7C82F6EF	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]
7C82F6F5	8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]
7C82F6F8	0FB640 02	MOVZX EAX,BYTE PTR DS:[EAX+2]
7C82F6FC	C3	RETN
7C82F6FD	90	NOP
7C82F6FE	90	NOP
7C82F6FF	90	NOP
7C82F700	90	NOP

lo sobrepasamos con un solo trazo, y salimos en la address '**004010D9**' donde hace un nuevo testeo del registro "EAX", con lo cual hemos sobreponiendo la "**CALL**" sin que haya pasado nada.

C CPU - main thread, module BBQ

Address	Hex dump	Disassembly	Comment
004010A7	.	PUSH EAX	
004010A8	. E8 2D00A0000	CALL <JMP.&kernel32.ReadProcessMemory>	hProcess
004010A9	. 85C0	TEST EAX, EAX	ReadProcessMemory
004010AF	.~ 74 16	JE SHORT BBQ.004010C7	
004010B1	.~ 813E 4F4C4C5	CMP DWORD PTR DS:[ESI],594C4C4F	
004010B7	.~ 75 0E	JNZ SHORT BBQ.004010C7	
004010B9	. C605 00E1400	MOU BYTE PTR DS:[40E100],1	
004010CB	. 6A 00	PUSH 0	
004010C2	. E8 CB090000	CALL <JMP.&kernel32.ExitProcess>	ExitCode = 0
004010C7	> 59	POP ECX	ExitProcess
004010C8	. 58	POP EAX	
004010C9	. 90	NOP	
004010CA	. 90	NOP	
004010CB	. 8030 00E1400	CMP BYTE PTR DS:[40E100],1	
004010D2	.~ 74 00	JE SHORT BBQ.004010D4	
004010D4	.> E8 DD090000	CALL <JMP.&kernel32.IsDebuggerPresent>	IsDebuggerPresent
004010D9	. 85C0	TEST EAX, EAX	
004010DB	.~ 74 07	JE SHORT BBQ.004010E4	
004010D0	. 6A 00	PUSH 0	ExitCode = 0
004010D1	. E8 AE090000	CALL <JMP.&kernel32.ExitProcess>	ExitProcess
004010E4	> 64:A1 300000	MOV EAX,DWORD PTR FS:[30]	
004010EA	. 8B40 02	MOV EAX,DWORD PTR DS:[EAX+2]	
004010ED	. 84C0	TEST AL,AL	
004010EF	.~ 74 09	JE SHORT BBQ.004010FA	
004010F1	. 6A 00	PUSH 0	ExitCode = 0
004010F3	. E8 9A090000	CALL <JMP.&kernel32.ExitProcess>	ExitProcess
004010F8	.~ EB 00	JMP SHORT BBQ.004010FA	
004010FA	.> 64:A1 300000	MOV EAX,DWORD PTR FS:[30]	
00401100	. 8B40 68	MOV EAX,DWORD PTR DS:[EAX+68]	
00401101	. 83E0 70		
00401106	.~ 74 09		
00401108	. 6A 00		
0040110A	. E8 83090000		
0040110F	.~ EB 00		

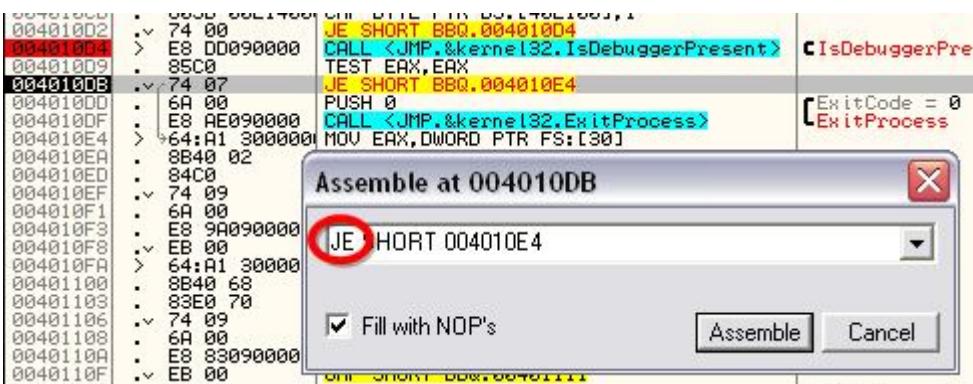
nuestra intuición también era cierta, nos deja continuar.

Seguimos.....

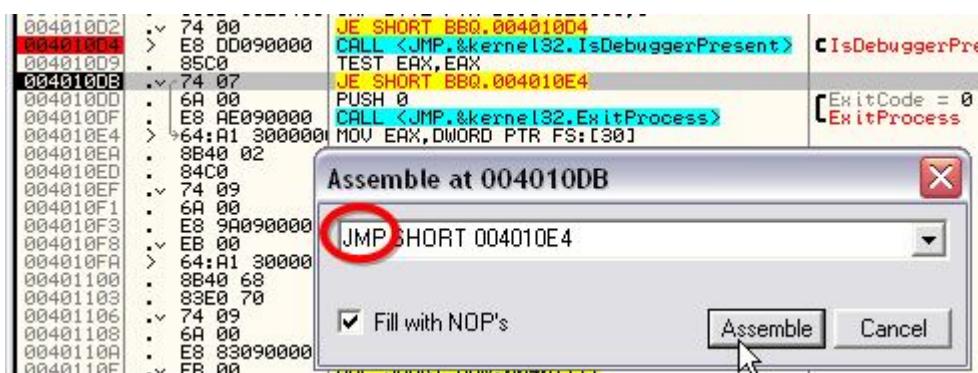
Un traceo más y nos encontramos posicionados en la address "004010DB" (Segunda trampa), donde el salto condicional "JE SHORT BBQ.004010E4" decidirá en función del resultado del testeo de "EAX" si nos saca fuera "004010DD PUSH 0 - ExitCode = 0 - ExitProcess", o en otro caso nos lleva "004010E4", donde nos dejará continuar.

Pues, vamos a substituir el "JE" por un "JMP" para que independientemente del valor de "EAX" después del testeo, salte siempre a la address "004010E4"

Posicionados sobre la address '004010DB' click derecho de ratón y cambiamos



Por

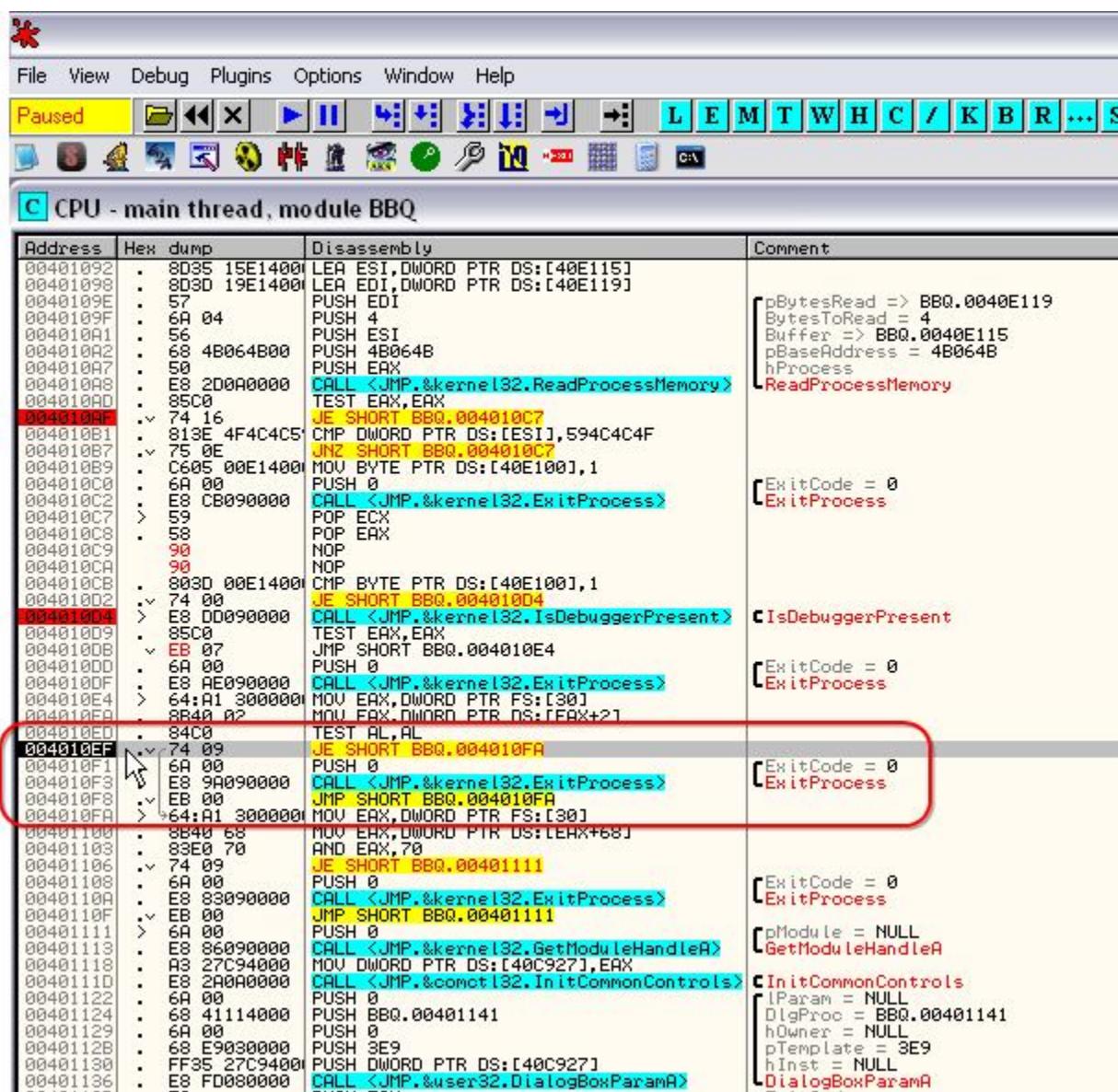


Le damos a "Assemble" y nos queda así:

004010D9	. 85C0	TEST EAX,EAX	
004010DB	✓ EB 07	JMP SHORT BBQ.004010E4	
004010D0	. 6A 00	PUSH 0	
004010D8	. E8 AE090000	CALL <JMP.&kernel32.ExitProcess>	[ExitCode = 0 ExitProcess]
004010E4	> 64:A1 300000	MOV EAX,DWORD PTR FS:[30]	
004010EA	. 8B40 02	MOV EAX,DWORD PTR DS:[EAX+2]	

Y efectivamente, vemos que si seguimos traceando, el salto incondicional "JMP" nos deja pasar burlando así nuestra segunda trampa "ExitProcess".

y continuamos traceando con mucho sigilo hasta la address '004010EF' que es donde se encuentra la próxima decisión "JE".



Viendo la captura superior, la cosa está más que clara, de nuevo nuestra intención ahora es ir directos a la address "004010FA" evitando con ello caer en otra tercera trampa que nos llevaría a "ExitProcess".

Por lo que también substituiremos el salto "JE" por un "JMP"

```

004010ED . 84C0 TEST AL,AL
004010EF .> 74 09 JE SHORT BBQ.004010FA
004010F1 . 6A 00 PUSH 0
004010F3 . E8 9A090000 CALL <JMP.&kernel32.ExitProcess>
004010F8 .> EB 00 JMP SHORT BBQ.004010FA
004010FA > 64:A1 300000 MOV EAX,DWORD PTR FS:[30]
00401100 . 8B40
00401103 . 83E0
00401106 .> 74 09 JE SHORT BBQ.004010FA
00401108 . 6A 00
0040110A . E8 80
0040110B .> EB 00 JMP SHORT BBQ.004010FA
00401111 . 6A 00
00401113 . E8 80
00401118 .> EB 00 JMP SHORT BBQ.004010FA
0040111D . 6A 00
00401122 . 68 20
00401124 . 6A 00
00401129 . 6A 00

```

Assemble at 004010EF

JE SHORT 004010FA

Fill with NOP's

Assemble Cancel

Por

```

004010EF .> 74 09 JE SHORT BBQ.004010FA
004010F1 . 6A 00 PUSH 0
004010F3 . E8 9A090000 CALL <JMP.&kernel32.ExitProcess>
004010F8 .> EB 00 JMP SHORT BBQ.004010FA
004010FA > 64:A1 300000 MOV EAX,DWORD PTR FS:[30]
00401100 . 8B40
00401103 . 83E0
00401106 .> 74 09 JMP SHORT BBQ.004010FA
00401108 . 6A 00
0040110A . E8 80
0040110B .> EB 00 JMP SHORT BBQ.004010FA
00401111 . 6A 00
00401113 . E8 80
00401118 .> EB 00 JMP SHORT BBQ.004010FA
0040111D . 6A 00
00401122 . 68 20
00401124 . 6A 00
00401129 . 6A 00

```

Assemble at 004010EF

JMP SHORT 004010FA

Fill with NOP's

Assemble Cancel

Y nos queda así:

```

004010D0 . 0B00 17E14000 LEA EDI,DWORD PTR DS:140E1174
004010E4 . 57 PUSH EDI
004010F9 . 6A 04 PUSH 4
004010A1 . 56 PUSH ESI
004010A2 . 68 4B064B00 PUSH 4B064B
004010A7 . 50 PUSH EAX
004010A8 . E8 200A0000 CALL <JMP.&kernel32.ReadProcessMemory>
004010AD . 85C0 TEST EAX,EAX
004010AF .> 74 16 JE SHORT BBQ.004010C7
004010B1 . 813E 4F4C4C51 CMP DWORD PTR DS:[ESI],594C4C4F
004010B7 .> 75 0E JNZ SHORT BBQ.004010C7
004010B9 . C605 00E14000 MOV BYTE PTR DS:[40E100],1
004010C0 . 6A 00 PUSH 0
004010C2 . E8 CB090000 CALL <JMP.&kernel32.ExitProcess>
004010C7 .> 59 POP ECX
004010C8 . 58 POP EAX
004010C9 . 90 NOP
004010CA . 90 NOP
004010CB . 803D 00E14001 CMP BYTE PTR DS:[40E100],1
004010D2 .> 74 00 JE SHORT BBQ.00401004
004010D4 .> E8 D0090000 CALL <JMP.&kernel32.IsDebuggerPresent>
004010D9 . 85C0 TEST EAX,EAX
004010DB .> EB 07 JMP SHORT BBQ.004010E4
004010D0 . 6A 00 PUSH 0
004010D0 . E8 A0900000 CALL <JMP.&kernel32.ExitProcess>
004010E4 .> 64:A1 300000 MOV EAX,DWORD PTR FS:[30]
004010EA . 8B40 02 MOV EAX,DWORD PTR DS:[EAX+2]
004010ED . 84C0 TEST AL,AL
004010EF .> EB 09 JMP SHORT BBQ.004010FA
004010F1 . 6A 00 PUSH 0
004010F3 . E8 9A090000 CALL <JMP.&kernel32.ExitProcess>
004010F8 .> EB 00 JMP SHORT BBQ.004010FA
004010FA .> 64:A1 300000 MOV EAX,DWORD PTR FS:[30]
00401100 . 8B40 68 MOV EAX,DWORD PTR DS:[EAX+68]
00401103 . 83E0 70 AND EAX,70
00401106 .> 74 09 JE SHORT BBQ.00401111
00401108 . 6A 00 PUSH 0
0040110A . E8 83090000 CALL <JMP.&kernel32.ExitProcess>
0040110F .> EB 00 JMP SHORT BBQ.00401111
00401111 . 6A 00 PUSH 0
00401113 . E8 86090000 CALL <JMP.&kernel32.GetModuleHandleA>

```

Parece que la cosa va bien, seguimos traceando sorteando así la tercera trampa hasta el próximo "JE" address 00401106"

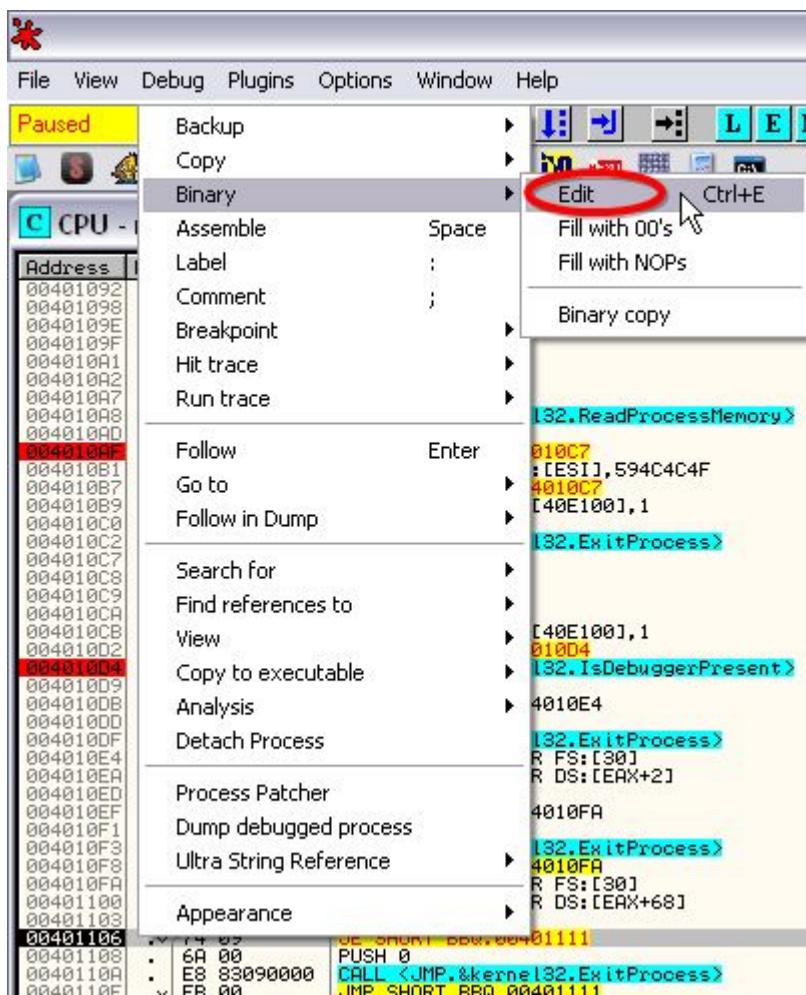
```

00401100 . 8B40 68 MOV EAX,DWORD PTR DS:[EAX+68]
00401103 . 83E0 70 AND EAX,70
00401106 .> 74 09 JE SHORT BBQ.00401111
00401108 . 6A 00 PUSH 0
0040110A . E8 83090000 CALL <JMP.&kernel32.ExitProcess>
0040110F .> EB 00 JMP SHORT BBQ.00401111
00401111 . 6A 00 PUSH 0
00401113 . E8 86090000 CALL <JMP.&kernel32.GetModuleHandleA>

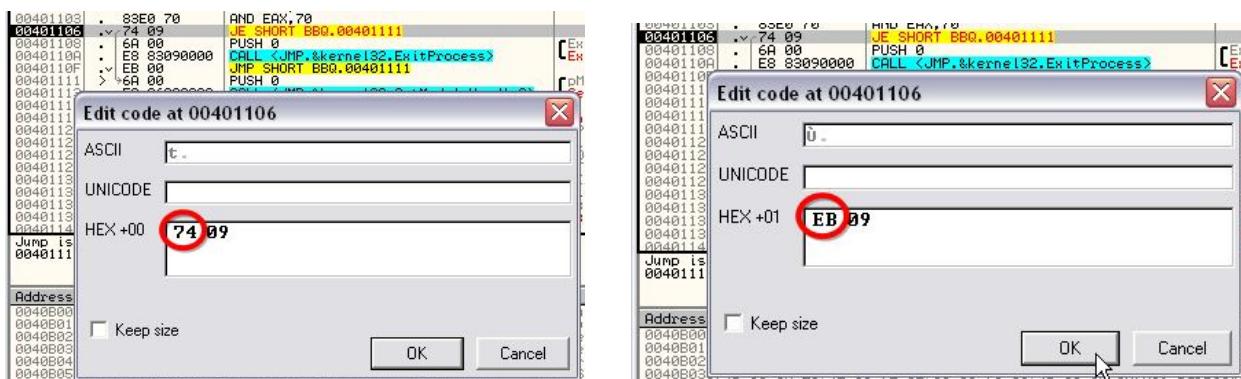
```

Y lo mismo de lo mismo, ahora cambiaremos la instrucción "JE" por un "JMP" directamente de la siguiente forma,

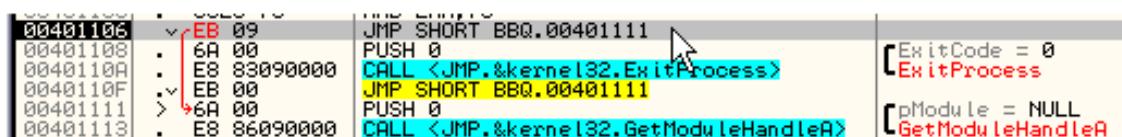
Posicionados en la address '00401106' click derecho de ratón y "Binary" - "Edit"



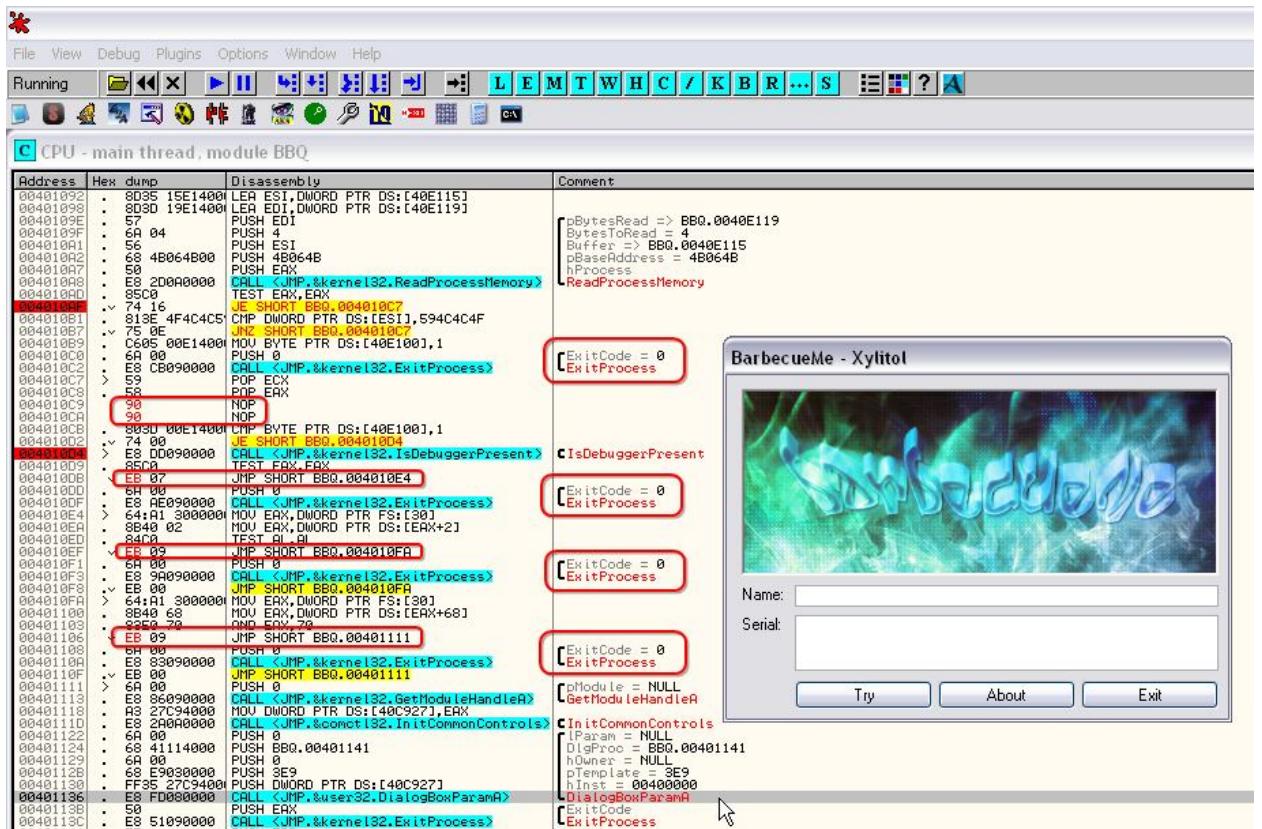
Y cambiamos el binario "74" = **JE** por un "EB" = **JMP** directamente



Y nos queda así



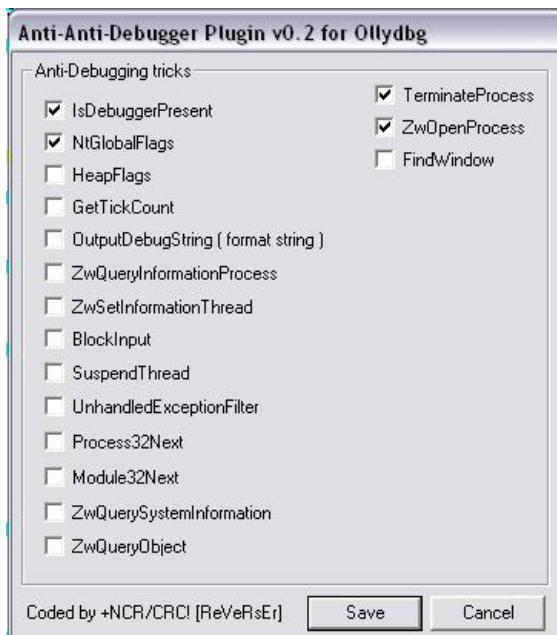
Seguimos traceando con "F8", burlando así la cuarta trampa, y al llegar a la address "00401136", con grata satisfacción, observamos que por fin se despliega el "BBQ.exe", con lo que podemos decir que hemos vencido la protección "IsDebuggerPresent" - **ExitProcess** de este Crackme de forma manual.



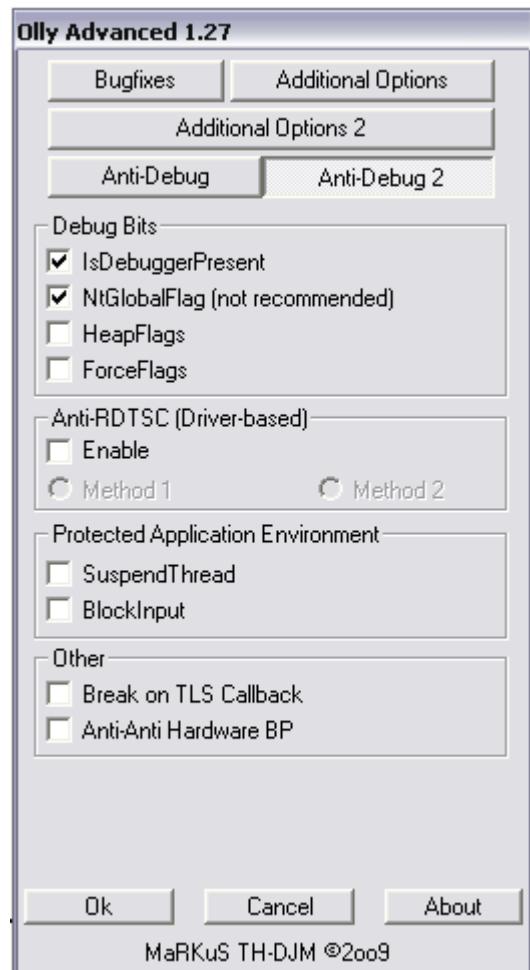
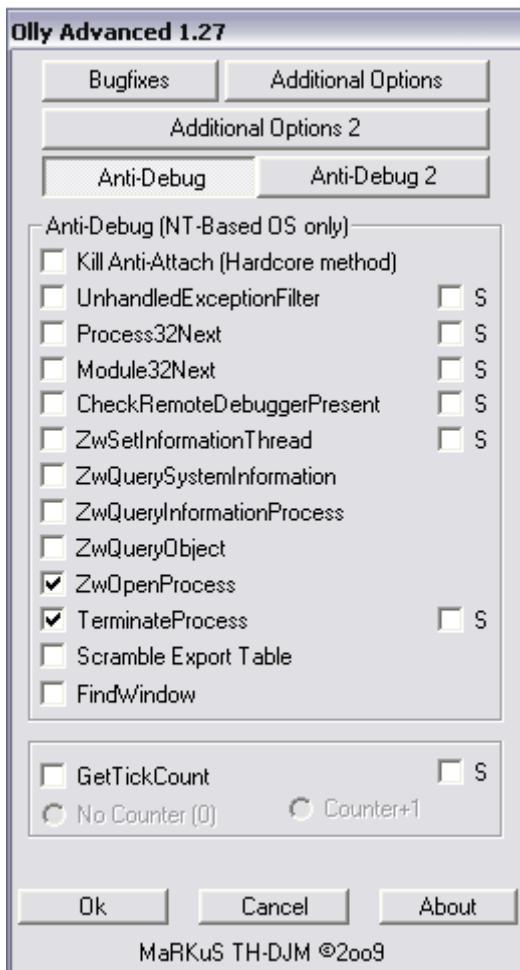
Ahora vamos a Registrarnos

Pero antes y sólo a título de información para los más vagos, les voy a confesar que los cuatro "plugins antidebugger" que deberíamos tener habilitados para saltar la protección de este Crackme de forma automática son los siguientes:

En el caso del plugin "Anti-Anti-Debugger v0.2"



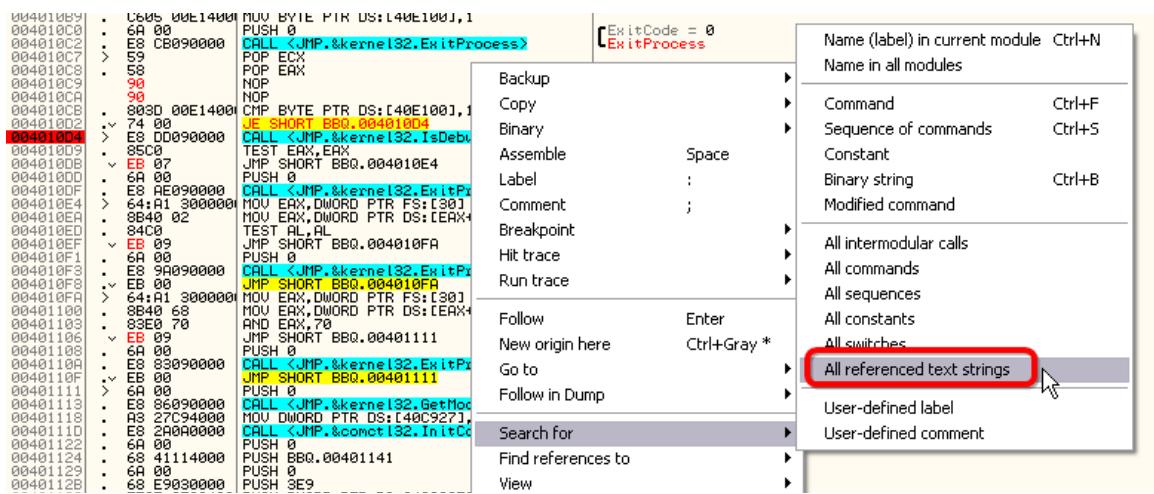
O bien, en el caso del plugin "Olly Advanced 1.27"



Pero ya lo probarán después, nosotros seguimos con los plugins totalmente deshabilitados.

- Buscando el "Serial" para nuestro "Name"

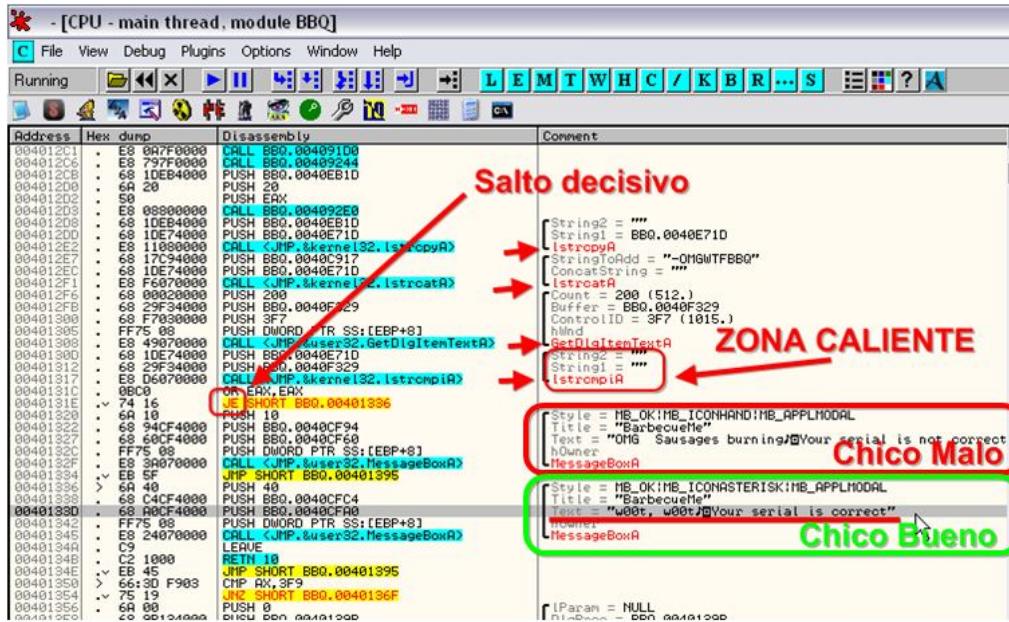
De momento no tipeamos nada, y en la "ventana principal del ensamblador damos un Click derecho de ratón y nos vamos a "Search for - All referenced text strings" para que nos muestre las referencias,



Y vemos estas:

R Text strings referenced in BBQ:.text			
Address	Disassembly	Text string	
00401007	PUSH BBB.0040CF48	ASCII "psapi.dll"	
00401011	PUSH BBB.0040CF52	ASCII "EnumProcesses"	
F 00401136	CALL <JMP.&user32.DialogBoxParamA>	(Initial CPU selection)	
0040118C	PUSH BBB.0040CF4	ASCII "BarbecueMe - Xylitol"	
0040122C	PUSH BBB.0040CF09	ASCII "...X"	
00401278	PUSH BBB.0040CF90	ASCII "KPCR tapz"	
9 004012E7	PUSH BBB.0040CF17	ASCII "-OMGWTFBBO"	
00401322	PUSH BBB.0040CF94	ASCII "BarbecueMe"	
00401327	PUSH BBB.0040CF60	ASCII "OMG Sausages burning@Your serial is not correct"	
00401338	PUSH BBB.0040CF04	ASCII "Proudly presents"	
0040133B	PUSH BBB.0040CFA0	ASCII "w00t, w00t! Your serial is correct"	
B 004014F4	MOU EDI,BBB.0040B000	ASCII "Xylitol@Proudly presents My First Crypto-KeygenMe: BarbecueMe@GFX: xsptd3r	
0040151C	MOU EDI,BBB.0040B000	ASCII "Xylitol@Proudly presents My First Crypto-KeygenMe: BarbecueMe@GFX: xsptd3r	
9 004018FD	PUSH BBB.0040B15D	ASCII "Lucida Console"	
0040196D	MOU EDI,BBB.0040B000	ASCII "Xylitol@Proudly presents My First Crypto-KeygenMe: BarbecueMe@GFX: xsptd3r	
00401984	PUSH BBB.0040B000	ASCII "Xylitol@Proudly presents My First Crypto-KeygenMe: BarbecueMe@GFX: xsptd3r	
004032A3	MOV DWORD PTR DS:[EAX+58],10000	UNICODE "ALLUSERSPROFILE=C:\Documents and Settings\All User"	

Que biénnnn, nos posicionamos sobre la que parece el mensaje de "Chico Bueno", le damos dos clicks izquierdos de ratón y aparecemos en una zona, donde ya vemos todo el entramado, je,je,je...



el Salto decisivo, la zona caliente, las APIs interesantes que también vimos al principio del tute, y el Chico Malo y Chico Bueno.

Hoy no tengo muchas ganas de estrujarme el tarro, y voy a poner un "BP" directamente en la APIs "Istrcmp" que se encuentra en la address "00401317", y que como sabemos su función es la de comparar strings.

Nos posicionamos sobre ella, y con "F2" le ponemos un punto de parada "BP"

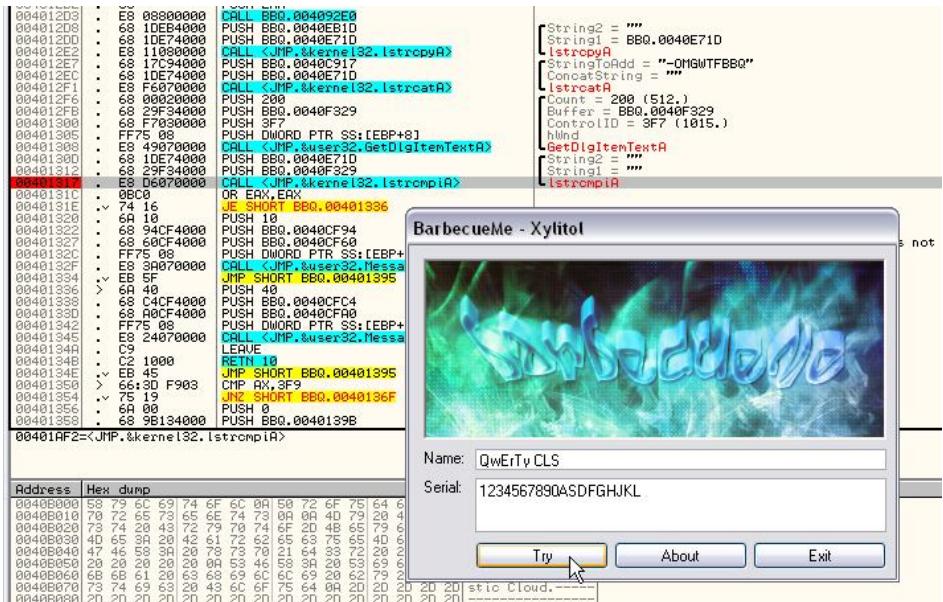
004012D3	. E8 00800000	CALL BBB.004092E0	
004012D8	. 68 1DEB4000	PUSH BBB.0040EB1D	
004012D9	. 68 1DE74000	PUSH BBB.0040E71D	
004012E2	. E8 11080000	CALL <JMP.&kernel32.lstrcpyA>	
004012E7	. 68 17C94000	PUSH BBB.0040CF17	
004012EC	. 68 1DE74000	PUSH BBB.0040E71D	
004012F1	. E8 F6070000	CALL <JMP.&kernel32.lstrcmpA>	
004012F6	. 68 00020000	PUSH 200	
004012FB	. 68 29F34000	PUSH BBB.0040F329	
00401300	. FF75 08	PUSH 3F7	
00401305	. E8 49070000	CALL <JMP.&user32.GetItemTextA>	
00401308	. 68 1DE74000	PUSH BBB.0040E71D	
0040130D	. 68 29F34000	PUSH BBB.0040F329	
00401312	. E8 D6070000	CALL <JMP.&kernel32.IstrcmpA>	
0040131C	. 0BC0	OR EAX,EAX	
00401320	. v 74 16	JE SHORT BBB.00401336	
00401322	. 6A 10	PUSH 10	
00401327	. 68 94CF4000	PUSH BBB.0040CF94	
0040132C	. 68 80CF4000	PUSH BBB.0040CF60	
0040132F	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	
00401334	. v EB 5F	CALL <JMP.&user32.MessageBoxA>	
00401336	. 6A 40	PUSH 40	
00401338	. 68 C4CF4000	PUSH BBB.0040CFC4	
00401342	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	
00401345	. E8 24070000	CALL <JMP.&user32.MessageBoxA>	
00401349	. C9	LEAVE	
0040134B	. v EB 45	RETN 10	
0040134E		JMP SHORT BBB.00401395	

```

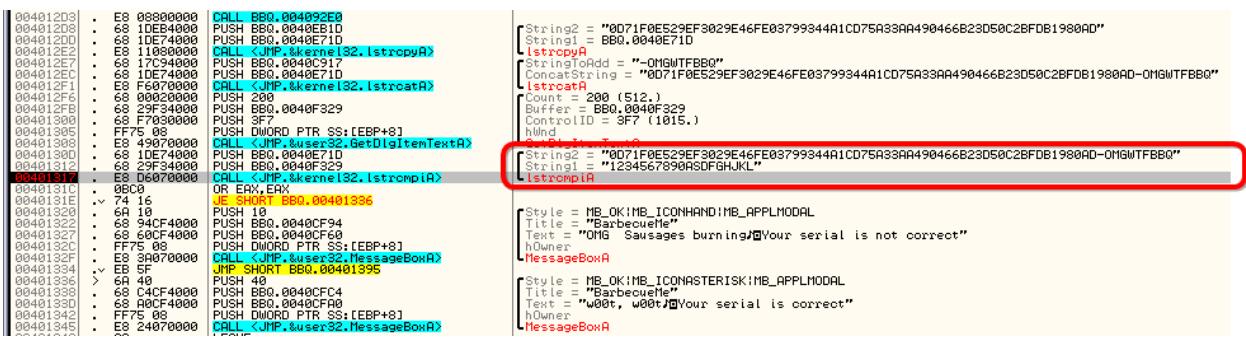
String2 = ""
String1 = BBB.0040E71D
IstrcpyA
StringToAdd = "-OMGWTFBBO"
ConcatString = ""
IstrcmpA
Count = 200 (512.)
Buffer = BBB.0040F329
ControlID = 3F7 (1015.)
hInd
GetDlgItemTextA
String2 = ""
String1 = ""
IstrcmpA
Style = MB_OK|MB_ICONHAND|MB_APPLMODAL
Title = "BarbecueMe"
Text = "OMG Sausages burning@Your serial is not correct"
hOwner
MessageBoxA
Style = MB_OK|MB_ICONASTERISK|MB_APPLMODAL
Title = "BarbecueMe"
Text = "w00t, w00t! Your serial is correct"
hOwner
MessageBoxA
Style = MB_OK|MB_ICONHAND|MB_APPLMODAL
Title = "BarbecueMe"
Text = "OMG Sausages burning@Your serial is not correct"
hOwner
MessageBoxA
Style = MB_OK|MB_ICONASTERISK|MB_APPLMODAL
Title = "BarbecueMe"
Text = "w00t, w00t! Your serial is correct"
hOwner
MessageBoxA

```

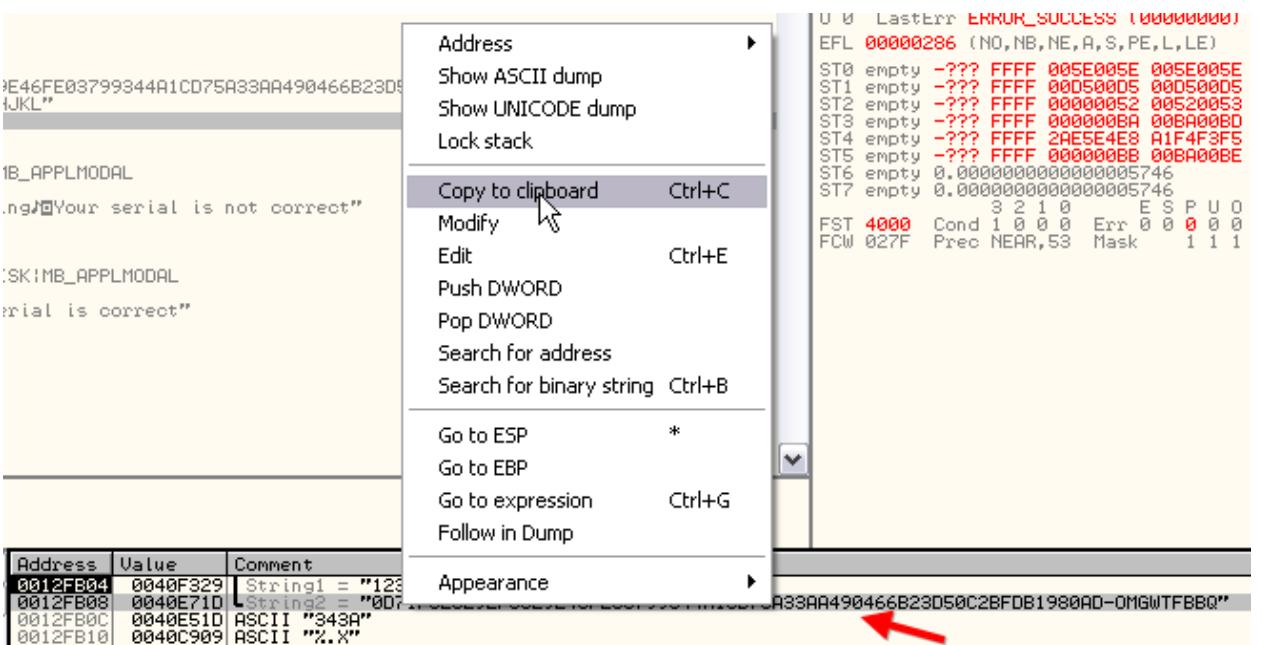
Acto seguido tipeamos un "Name" y un "Serial" que nos venga en gana



Le damos a "Try" para que el Crackme siga corriendo, y Olly hace parada en este último "BP" (ya que los anteriores que tenemos puestos ya los habíamos rebasado), mostrándonos una larguísima cadena de texto que parece un "Serial" generado especialmente para nuestro "Name".

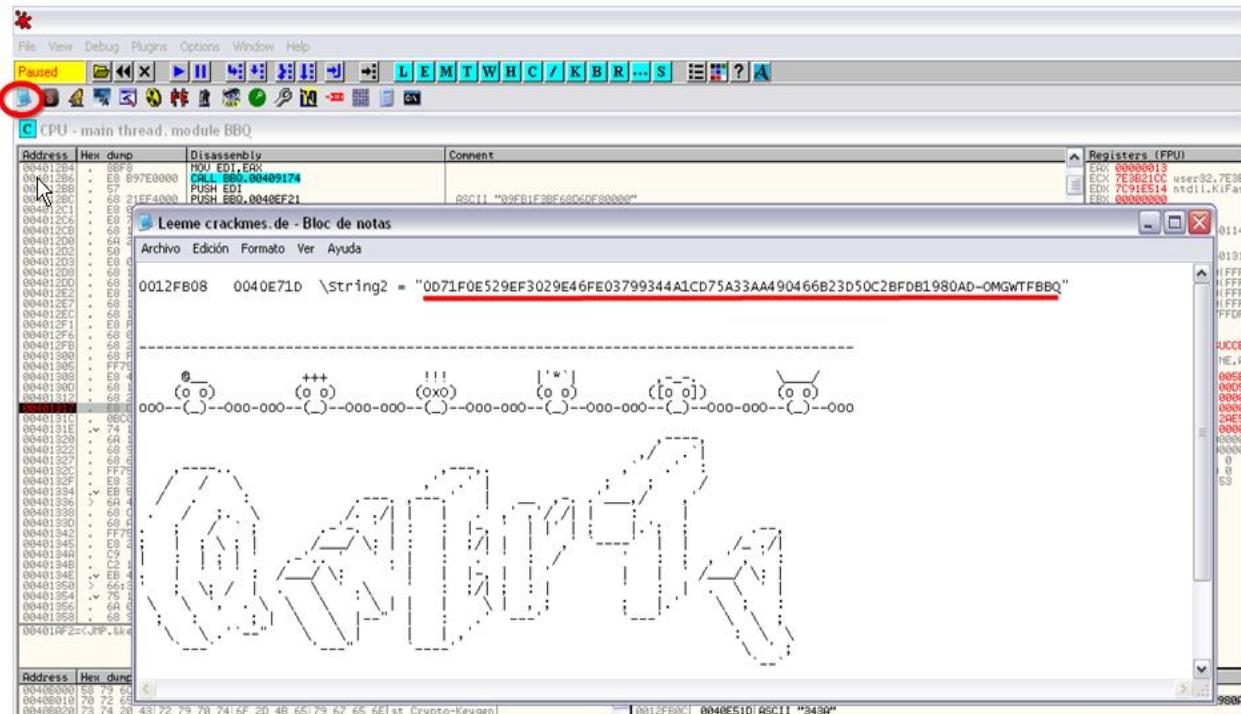


Nos vamos a la ventana "Stack", nos posicionamos sobre la cadena de texto, Click derecho de ratón y "Copy to clipboard"

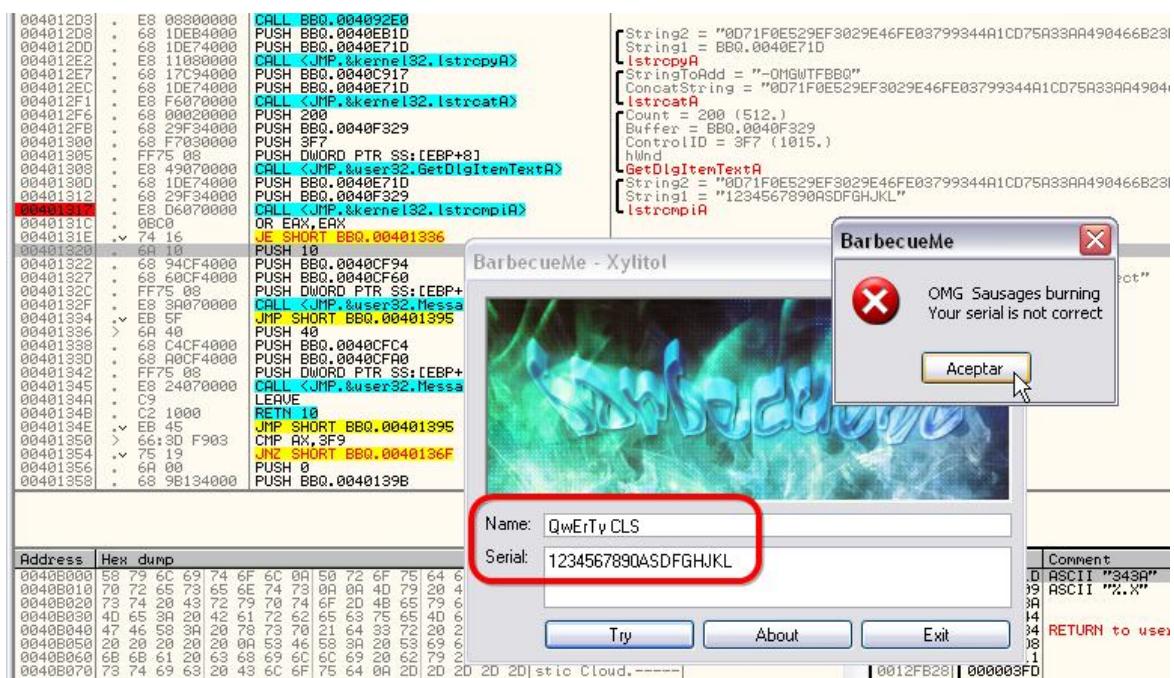


Lo pegamos en un bloc de notas,

Y nos queda así:



Damos "Run" para que salga el mensaje de que hemos fallado,



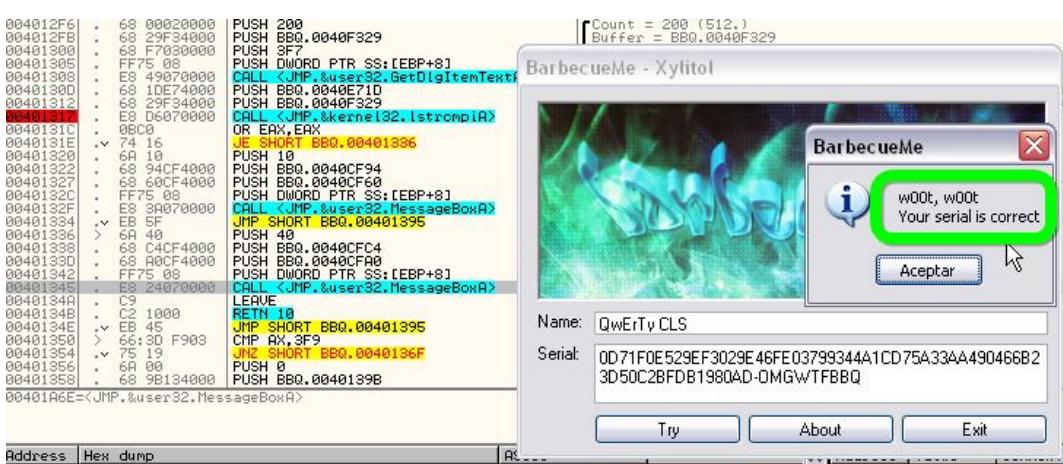
Le damos a "Aceptar", y ahora introducimos el "Serial" Obtenido para nuestro "Name" con un copiar/pegar desde el bloc de notas para no equivocarnos, ya que el muy puñetero es larguísimo.

Le damos a "Try" para que vuelva a correr el Crackme..... y parados de nuevo en el "BP".

Le damos a "Try" para que vuelva a correr el Crackme..... y parados de nuevo en el "BP".

Ya vemos que la comparación de las dos Strings será exitosa por que tienen el mismo valor, vamos traceando con "F8" hasta llegar al Salto condicional decisivo "JE" address "0040131E", cogemos un pañuelo, nos secamos el sudor de la frente, pegamos un buen trago de "Woll-Damm doble malta", y con enorme satisfacción ya vemos que nos llevará directos a la zona de "Chico Bueno".

Continuamos traceando y.....



Bingooooooo, hemos dado en la diana, je,je,je , hemos hallado el "Serial" correcto para nuestro "Name" tipeado "QwErTy CLS"

"OD71FOE529EF3029E46FE03799344A1CD75A33AA490466B23D50C2BFDB1980AD-OMGWTFBQQ"

Si quisieramos hacer un Patch para que aceptara cualquier Serial (menos el bueno) independientemente del "Name" que introduzcamos, simplemente cambiaríamos el "JE" del salto decisivo por un "JNE" en la address "0040131C" y eso sería todo.

Bueno, la verdad es que en esta ocasión ha sido muy fácil encontrar como registrarnos, y el tiburón no era tan fiero como parecía.

Todo el truco de este Crackme estaba únicamente en saltar la protección antidebugger, y tampoco ha sido tan difícil sortearla, pero vamos a esforzarnos un poco más y intentaremos hacer un Self-Keygen para que sea el mismo Crackme el que autogeneré y nos muestre directamente el "Serial" correcto para cualquier "Name" que tipeemos.

Manos a la obra.....

- Creando un Self-Keygen

Utilizaremos la APIs "User32.SetDlgItemTextA" que como ya sabemos, esta función envía un mensaje "WM_SETTEXT" al control especificado, y también establece el título o texto de un control en un cuadro de diálogo.

Para que esta APIs funcione, necesitamos encontrar el "handle de la ventana", el "ID de la caja de texto" y la "String a mostrar".

Pues vamos a buscar estos ingredientes.

Recordemos que No hemos guardado ningún cambio en el ejecutable, ni pensamos hacerlo.....todavía....., le damos a "Aceptar" al mensaje de felicitación



Damos "run"

Y ahora ingresamos un Serial trucho,



Le damos a "Try" para validar, y Olly vuelve a parar en nuestro "BP" , que está en la address "00401317" donde se encuentra la APIs "IstrcmpiA"

Y de un vistazo, vemos todos los datos que nos hacen falta.....je,je,je...

Address Hex dump Disassembly Comment

```

004012B0: E8 21EF4000 PUSH BB0.0040EF21 ASCII "09FB1F3BF68D6DF80000"
004012C1: E8 007F0000 CALL BB0.004091D0
004012C6: E8 797F0000 CALL BB0.00409244
004012C9: E8 10EB4000 PUSH BB0.0040EB10
004012D0: E8 20 PUSH EAX
004012D2: E8 50
004012D3: E8 08800000 CALL BB0.004092E0
004012D8: E8 10EB4000 PUSH BB0.0040EB10
004012D9: E8 10DE74000 PUSH BB0.0040E710
004012E2: E8 11080000 CALL <JMP.&kernel32.IstrcpyA>
004012E7: E8 17C94000 PUSH BB0.0040C917
004012EC: E8 10EB74000 PUSH BB0.0040E710
004012F1: E8 F6070000 CALL <JMP.&kernel32.IstrcatA>
004012F6: E8 00020000 PUSH 200
004012F9: E8 29F34000 PUSH BB0.0040F329
00401300: E8 F7030000 PUSH 3F7
00401305: FF75 08 PUSH DWORD PTR SS:[EBP+8]
00401308: E8 00800000 CALL <JMP.&user32.GetDlgItemTextA>
0040130D: E8 10EB4000 PUSH BB0.0040E710
00401312: E8 29F34000 PUSH BB0.0040F329
00401317: E8 D6070000 CALL <JMP.&kernel32.IstrcpyA>
0040131C: E8C0 OR EAX,EAX
0040131E: ✓ 74 16 JE SHORT BB0.00401336
00401320: ✓ 6A 10 PUSH 10
00401322: ✓ 68 94CF4000 PUSH BB0.0040CF94
00401327: ✓ 68 60CF4000 PUSH BB0.0040CF60
0040132C: FF75 08 PUSH DWORD PTR SS:[EBP+8]
0040132F: E8 3A070000 CALL <JMP.&user32.MessageBoxA>
00401334: > EB 5F JNP SHORT BB0.00401395
00401336: > 6A 40 PUSH 40
00401338: > 68 C4CF4000 PUSH BB0.0040CF4
0040133D: > 68 A0CF4000 PUSH BB0.0040CF00
00401342: > FF75 08 PUSH BB0.0040F329
00401345: > E8 24070000 CALL <JMP.&user32.MessageBoxA>
0040134A: > C2 1000 LEAVE
0040134B: > EB 45 RETN 10
0040134E: > EB 45 JNP SHORT BB0.00401395
00401350: > 66:3D F903 CMP AX,3F9
00401354: > 75 19 JNZ SHORT BB0.0040136F

```

String a mostrar -----> PUSH 0040E71D

ID de la caja de texto -----> PUSH 3F7

Handle de la ventana -----> PUSH DWORD PTR SS:[EBP+8]

(ojo, en algunas PC estos datos pueden cambiar, en ese caso el Self-Keygen que vamos a crear solo funcionará en nuestra máquina)

Acto seguido NOPeamos la parte de código que no nos interesa para nada, y que va desde la address "0040131C" hasta la "00401345"

EIP 00401317 BB0.00401317

C 0	E5 0023	32bit 0(FFFF)
P 1	C5 001B	32bit 0(FFFF)
A 0	SS 0023	32bit 0(FFFF)
T 2	D5 0023	32bit 0(FFFF)
S 1	F5 0098	32bit 7FFDE0
T 0	G5 0000	NULL
D 0		
O 0		LastErr ERROR_SUCCESS
EFL 00000286		(NO,NB,NE,A,
ST0 empty -???	FFFF 0050	ST1 empty -???
ST1 empty -???	FFFF 0050	

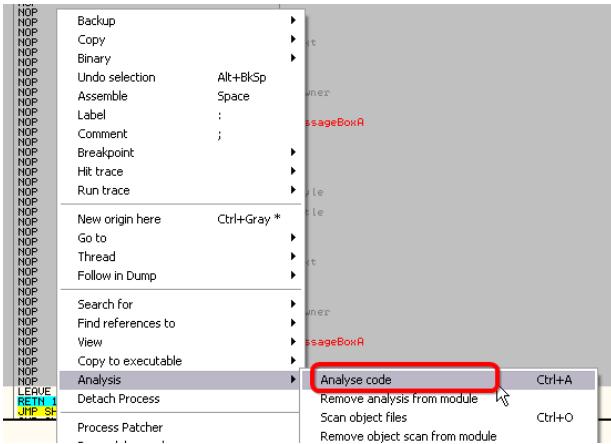
Backup
Copy
Binary
Assemble Space
Label :
Comment ;
Breakpoint
Hit trace
Run trace

Fill with 0's
Fill with NOPs
Binary copy
Binary paste

Y nos queda así:

Address	Hex dump	Disassembly	Comment
004012F1	.	E8 F6970000 CALL <JMP.&kernel32.IstrcmplA>	IstrcmplA Count = 200 (512.) Buffer = BB0.0040F329 ControlID = 3F7 (1015.) hWnd GetDlgItemTextA String2 = "0071F0E529EF3029f" String1 = "989898" IstrcmplA
004012F5	.	68 00020000 PUSH 200	NOP
004012FB	.	68 29F34000 PUSH BB0.0040F329	NOP
00401300	.	68 F7030000 PUSH 3F7	NOP
00401303	.	68 00000000 PUSH DWORD PTR SS:[EBP+8]	NOP
00401308	.	E8 49070000 CALL <JMP.&user32.GetDlgItemTextA>	NOP
0040130D	.	68 1DE74000 PUSH BB0.0040E71D	NOP
00401312	.	68 29F34000 PUSH BB0.0040F329	NOP
00401317	.	E8 D6870000 CALL <JMP.&kernel32.IstrcmplA>	NOP
0040131C	.	90 NOP	NOP
0040131D	.	90 NOP	NOP
0040131E	.	90 NOP	NOP
0040131F	.	90 NOP	NOP
00401320	.	90 NOP	NOP
00401321	.	90 NOP	NOP
00401322	.	90 NOP	NOP
00401323	.	90 NOP	NOP
00401324	.	90 NOP	NOP
00401325	.	90 NOP	NOP
00401326	.	90 NOP	NOP
00401327	.	90 NOP	NOP
00401328	.	90 NOP	NOP
00401329	.	90 NOP	NOP
0040132A	.	90 NOP	NOP
0040132B	.	90 NOP	NOP
0040132C	.	90 NOP	NOP
0040132D	.	90 NOP	NOP
0040132E	.	90 NOP	NOP
0040132F	.	90 NOP	NOP
00401330	.	90 NOP	NOP
00401331	.	90 NOP	NOP
00401332	.	90 NOP	NOP
00401333	.	90 NOP	NOP
00401334	.	90 NOP	NOP
00401335	.	90 NOP	NOP
00401336	.	90 NOP	NOP
00401337	.	90 NOP	NOP
00401338	.	90 NOP	NOP
00401339	.	90 NOP	NOP
0040133A	.	90 NOP	NOP
0040133B	.	90 NOP	NOP
0040133C	.	90 NOP	NOP
0040133D	.	90 NOP	NOP
0040133E	.	90 NOP	NOP
0040133F	.	90 NOP	NOP
00401340	.	90 NOP	NOP
00401341	.	90 NOP	NOP
00401342	.	90 NOP	NOP
00401343	.	90 NOP	NOP
00401344	.	90 NOP	NOP
00401345	.	90 NOP	NOP
00401346	.	90 NOP	NOP
00401347	.	90 NOP	NOP
00401348	.	90 NOP	NOP
00401349	.	90 NOP	NOP
0040134E	.	EB 10 EB 45 LEAVE RETN 10 JMP SHORT BB0.00401395	LEAVE RETN 10 JMP SHORT BB0.00401395

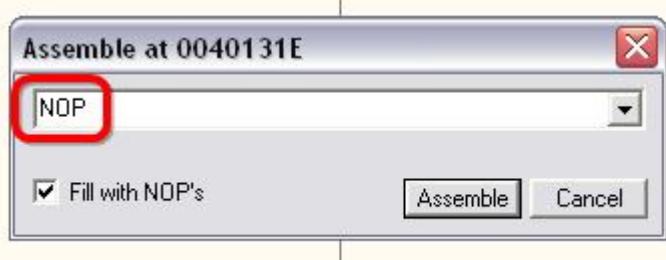
Analizamos código para que se reorganice todo:



Nos posicionamos en cualquier parte del código NOPeado, (yo lo hago en la tercera línea),

Dos clicks izquierdos de ratón

004012F6	.	68 00020000 PUSH 200	Count = 200 (512.)
004012FB	.	68 29F34000 PUSH BB0.0040F329	Buffer = BB0.0040F329
00401300	.	68 F7030000 PUSH 3F7	ControlID = 3F7 (1015.)
00401303	.	68 00000000 PUSH DWORD PTR SS:[EBP+8]	hWnd
00401308	.	E8 49070000 CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
0040130D	.	68 1DE74000 PUSH BB0.0040E71D	String2 = "0071F0E529EF3029f"
00401312	.	68 29F34000 PUSH BB0.0040F329	String1 = "989898"
00401317	.	E8 D6870000 CALL <JMP.&kernel32.IstrcmplA>	IstrcmplA
0040131C	.	90 NOP	
0040131D	.	90 NOP	
0040131E	.	90 NOP	
0040131F	.	90 NOP	
00401320	.	90 NOP	
00401321	.	90 NOP	
00401322	.	90 NOP	
00401323	.	90 NOP	
00401324	.	90 NOP	
00401325	.	90 NOP	
00401326	.	90 NOP	
00401327	.	90 NOP	
00401328	.	90 NOP	
00401329	.	90 NOP	
0040132A	.	90 NOP	
0040132B	.	90 NOP	
0040132C	.	90 NOP	
0040132D	.	90 NOP	
0040132E	.	90 NOP	
0040132F	.	90 NOP	



Y primero cambiamos la instrucción "NOP" por un "PUSH 0040E71D", que es la dirección donde el ejecutable guarda la String en memoria correspondiente al "Serial" correcto.

```

300 CALL <JMP.&kernel32.lstrcmpiA>
300 PUSH 200
300 PUSH BBQ.0040F329
300 PUSH 3F7
300 PUSH DWORD PTR SS:[EBP+8]
300 CALL <JMP.&user32.GetDlgItemTextA>
300 PUSH BBQ.0040E71D
300 PUSH BBQ.0040F329
300 CALL <JMP.&kernel32.lstrcmpiA>
NOP

```

Registers window (top right):

- lstrcmpA: Count = 200 (512.), Buffer = BBQ.0040F329, ControlID = 3F7 (1015.), hWnd
- GetDlgItemTextA: String2 = "0D71F0E529EF3029E4", String1 = "989898"
- lstrcmpiA

Assemble at 0040131E dialog (center):

PUSH 0040E71D

Fill with NOP's

Assemble Cancel

Y le damos a "Assemble",

Segundo, cambiamos el siguiente "NOP" por la instrucción "PUSH 3F7", correspondiente al "ID de la caja de texto"

Address	Hex dump	Disassembly	Comment
004012F1	• E8 F6070000	CALL <JMP.&kernel32.lstrcmpA>	
004012F6	• 68 00020000	PUSH 200	
004012FB	• 68 29F34000	PUSH BBQ.0040F329	
00401300	• 68 F7030000	PUSH 3F7	
00401305	• FF75 08	PUSH DWORD PTR SS:[EBP+8]	
00401308	• E8 49070000	CALL <JMP.&user32.GetDlgItemTextA>	
0040130D	• 68 1DE74000	PUSH BBQ.0040E71D	
00401312	• 68 29F34000	PUSH BBQ.0040F329	
00401317	• E8 D6070000	CALL <JMP.&kernel32.lstrcmpiA>	
0040131C	• 90	NOP	
0040131D	• 90	NOP	
0040131E	• 68 1DE74000	PUSH BBQ.0040E71D	ASCII "0D71F0E529EF3029E46
00401323	• 90	NOP	
00401324	• 90	NOP	
00401325	• 90	NOP	
00401326	• 90	NOP	
00401327	• 90	NOP	
00401328	• 90	NOP	
00401329	• 90	NOP	
0040132A	• 90	NOP	
0040132B	• 90	NOP	
0040132C	• 90	NOP	
0040132D	• 90	NOP	
0040132E	• 90	NOP	
0040132F	• 90	NOP	
00401330	• 90	NOP	
00401331	• 90	NOP	
00401332	• 90	NOP	

Assemble at 00401323 dialog (bottom right):

PUSH 3F7

Fill with NOP's

Assemble Cancel

Tercero, cambiamos el siguiente "NOP" por la instrucción "PUSH DWORD PTR SS:[EBP+8]" que corresponde al "Handle de la ventana"

Address Hex dump Disassembly Comment

004012F1	.	E8 F6070000 CALL <JMP.&kernel32.lstrcmpA>	[lstrcmpA]
004012F6	.	68 00020000 PUSH 200	Count = 200 (512.)
004012FB	.	68 29F34000 PUSH BB0.0040F329	Buffer = BB0.0040F329
00401300	.	68 F7030000 PUSH 3F7	ControlID = 3F7 (1015.)
00401305	.	FF75 08 PUSH DWORD PTR SS:[EBP+8]	hWnd
00401308	.	E8 49070000 CALL <JMP.&user32.GetDlgItemTextA>	[GetDlgItemTextA]
0040130D	.	68 1DE74000 PUSH BB0.0040E71D	String2 = "0D71F0E529EF3029E46FE03799344A1CD75A3AA490466B23D50C2BFDB1980AD-0MGWTFBBO"
00401312	.	68 29F34000 PUSH BB0.0040F329	String1 = "989898"
00401317	.	E8 D6070000 CALL <JMP.&kernel32.lstrcmpA>	[lstrcmpA]
0040131C	.	90 NOP	ASCII "0D71F0E529EF3029E46FE03799344A1CD75A3AA490466B23D50C2BFDB1980AD-0MGWTFBBO"
0040131D	.	90 NOP	
0040131E	.	68 1DE74000 PUSH BB0.0040E71D	
00401323	.	68 F7030000 PUSH 3F7	
00401328	.	90 NOP	
00401329	.	90 NOP	
0040132B	.	90 NOP	
0040132C	.	90 NOP	
0040132D	.	90 NOP	
0040132E	.	90 NOP	
0040132F	.	90 NOP	
00401330	.	90 NOP	
00401331	.	90 NOP	
00401332	.	90 NOP	
00401333	.	90 NOP	
00401334	.	90 NOP	
00401335	.	90 NOP	
00401336	.	90 NOP	
00401337	.	90 NOP	
00401338	.	90 NOP	

Y cuarto y último, insertamos la instrucción "CALL SetDlgItemTextA" que se encargará de mostrarnos el Serial válido para nuestro Name en la caja de texto que le hemos indicado.

Address Hex dump Disassembly Comment

004012F1	.	E8 F6070000 CALL <JMP.&kernel32.lstrcmpA>	[lstrcmpA]
004012F6	.	68 00020000 PUSH 200	Count = 200 (512.)
004012FB	.	68 29F34000 PUSH BB0.0040F329	Buffer = BB0.0040F329
00401300	.	68 F7030000 PUSH 3F7	ControlID = 3F7 (1015.)
00401305	.	FF75 08 PUSH DWORD PTR SS:[EBP+8]	hWnd
00401308	.	E8 49070000 CALL <JMP.&user32.GetDlgItemTextA>	[GetDlgItemTextA]
0040130D	.	68 1DE74000 PUSH BB0.0040E71D	String2 = "0D71F0E529EF3029E46FE03799344A1CD75A3AA490466B23D50C2BFDB1980AD-0MGWTFBBO"
00401312	.	68 29F34000 PUSH BB0.0040F329	String1 = "989898"
00401317	.	E8 D6070000 CALL <JMP.&kernel32.lstrcmpA>	[lstrcmpA]
0040131C	.	90 NOP	ASCII "0D71F0E529EF3029E46FE03799344A1CD75A3AA490466B23D50C2BFDB1980AD-0MGWTFBBO"
0040131D	.	90 NOP	
0040131E	.	68 1DE74000 PUSH BB0.0040E71D	
00401323	.	68 F7030000 PUSH 3F7	
00401328	.	90 NOP	
00401329	.	90 NOP	
0040132B	.	90 NOP	
0040132C	.	90 NOP	
0040132D	.	90 NOP	
0040132E	.	90 NOP	
0040132F	.	90 NOP	
00401330	.	90 NOP	
00401331	.	90 NOP	
00401332	.	90 NOP	
00401333	.	90 NOP	
00401334	.	90 NOP	
00401335	.	90 NOP	
00401336	.	90 NOP	
00401337	.	90 NOP	
00401338	.	90 NOP	

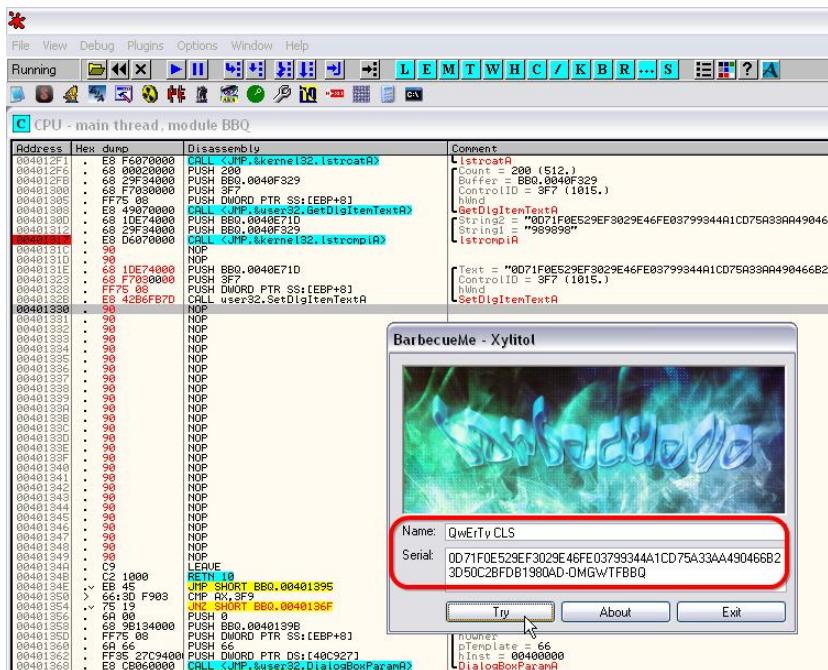
Volvemos a analizar código para que se reorganice todo, y ahora se ve de maravilla....., que bonito, incluso me atrevería a decir que tiene muy buena pinta

Address Hex dump Disassembly Comment

004012F1	.	E8 F6070000 CALL <JMP.&kernel32.lstrcmpA>	[lstrcmpA]
004012F6	.	68 00020000 PUSH 200	Count = 200 (512.)
004012FB	.	68 29F34000 PUSH BB0.0040F329	Buffer = BB0.0040F329
00401300	.	68 F7030000 PUSH 3F7	ControlID = 3F7 (1015.)
00401305	.	FF75 08 PUSH DWORD PTR SS:[EBP+8]	hWnd
00401308	.	E8 49070000 CALL <JMP.&user32.GetDlgItemTextA>	[GetDlgItemTextA]
0040130D	.	68 1DE74000 PUSH BB0.0040E71D	String2 = "0D71F0E529EF3029E46FE03799344A1CD75A3AA490466B23D50C2BFDB1980AD-0MGWTFBBO"
00401312	.	68 29F34000 PUSH BB0.0040F329	String1 = "989898"
00401317	.	E8 D6070000 CALL <JMP.&kernel32.lstrcmpA>	[lstrcmpA]
0040131C	.	90 NOP	Text = "0D71F0E529EF3029E46FE03799344A1CD75A3AA490466B23D50C2BFDB1980AD-0MGWTFBBO"
0040131D	.	90 NOP	ControlID = 3F7 (1015.)
0040131E	.	68 1DE74000 PUSH BB0.0040E71D	
00401323	.	68 F7030000 PUSH 3F7	
00401328	.	90 NOP	
00401329	.	90 NOP	
0040132B	.	68 42B6FB70 CALL user32.SetDlgItemTextA	[SetDlgItemTextA]
00401330	.	90 NOP	
00401331	.	90 NOP	
00401332	.	90 NOP	
00401333	.	90 NOP	
00401334	.	90 NOP	
00401335	.	90 NOP	
00401336	.	90 NOP	
00401337	.	90 NOP	
00401338	.	90 NOP	

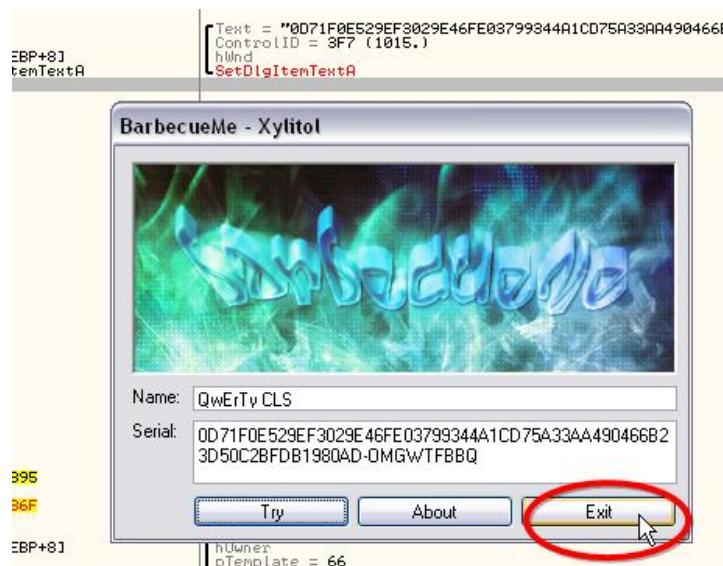
(Una vez más recordemos que todavía estamos en ejecución, y que habíamos tipeado el serial trucho "989898").

Damos "run" para que siga corriendo el Crackme.....y.....



Efectivamente, podemos comprobar que el Crackme ha substituido el serial trucho que tipeamos, y en su lugar nos muestra el Serial correcto de forma automática...je, je, je...

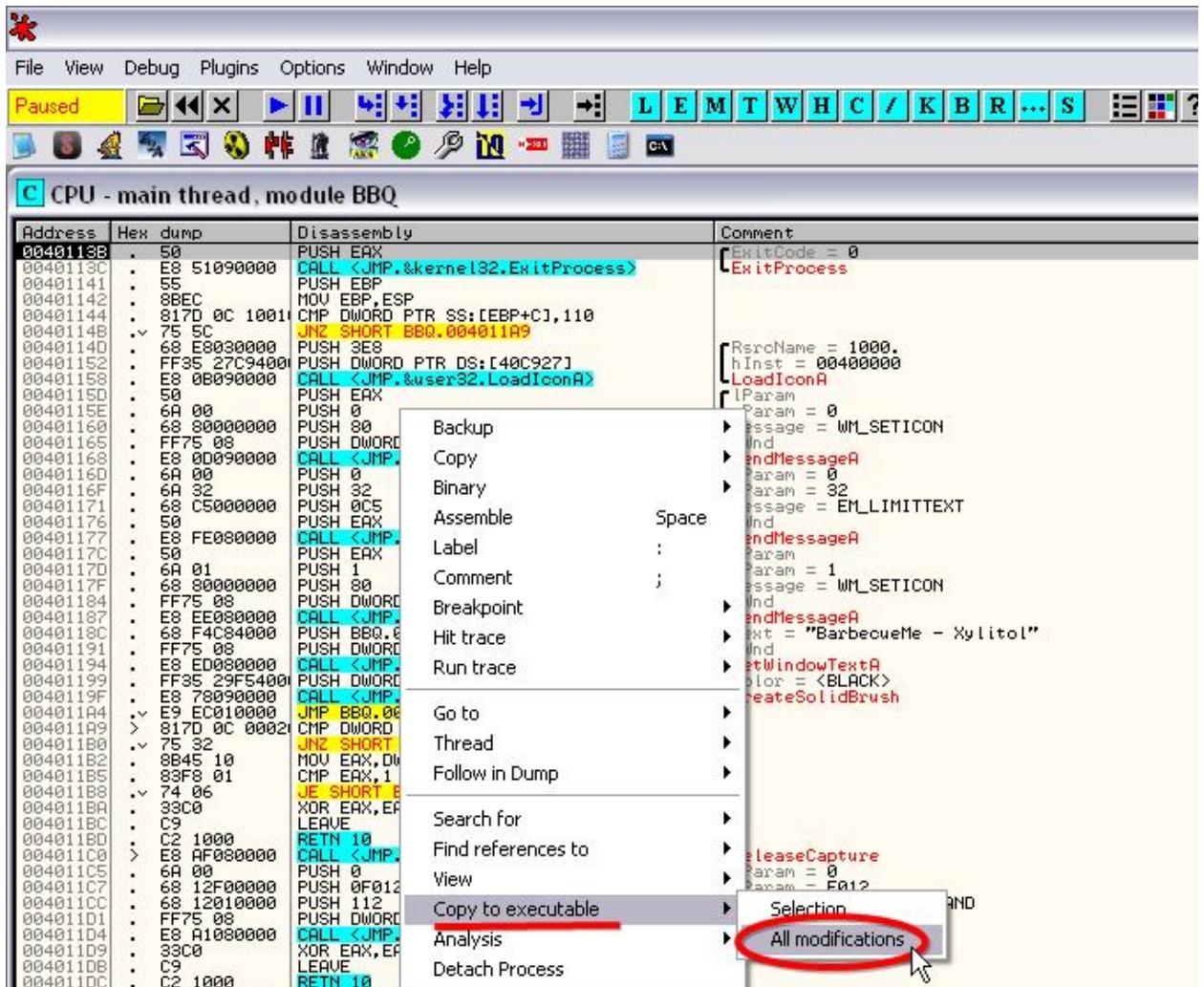
Le damos a "Exit" para Salir del Crackme



Borramos todos los "BP" que tenemos puestos

Op	Disassembly	Comment
50	PUSH EAX	
E803	CALL <JMP.&kernel32.ExitProcess>	[ExitCode = 0 ExitProcess
7D 00		
5C		
E803	Address Module Active Disassembly Comment	
35 21	004010AF BBQ Always JE SHORT BBQ.004010C7	
0B09	004010D4 BBQ Always CALL <JMP.&kernel32.IsDebuggerPresent>	
	00401317 BBQ Always CALL <JMP.&kernel32.IstronpiA>	
1 00		
8000		
75 08		
BBQ		
		Remove Del
		Disable Space
		Edit condition

Ahora SI, guardamos cambios,



Renombramos nuestro Crackme modificado, yo le he llamado, **Self-Keygen.exe** y le damos a "Guardar"

Nombre:	<input type="text" value="Self-Keygen.exe"/>	<input type="button" value="Guardar"/>
Tipo:	<input type="text" value="Executable file (*.exe)"/>	<input type="button" value="Cancelar"/>

Ya podemos salir de Olly, nos vamos a la ruta donde lo hemos guardado, y comprobamos que funcione.

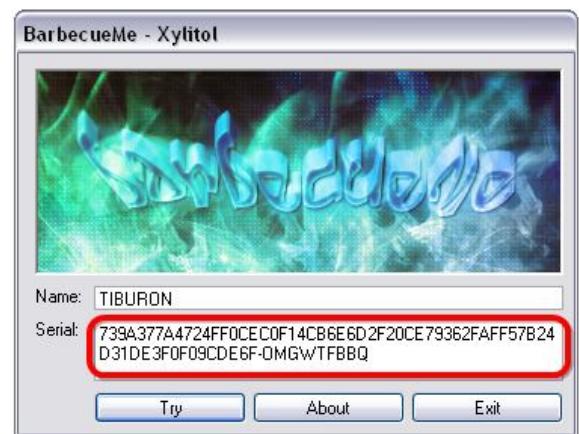
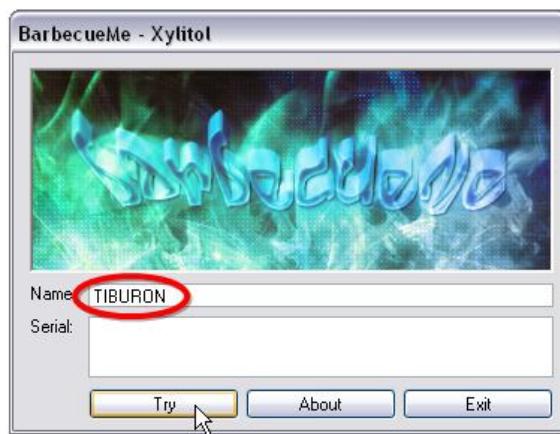


Ejecutamos el "Self-Keygen", en "Name" tipo "QwErTy 2018" le damos a "Try"



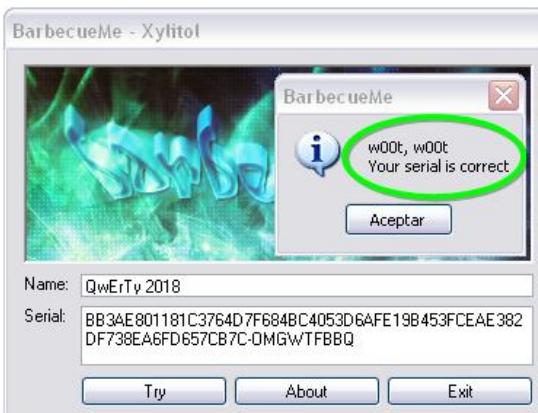
Yparece que funciona,

Ahora pruebo con otro "Name" , Tipeo "TIBURON" le damos a "Try"



Compruebo los resultados obtenidos





Yambos seriales son correctos.

Pero no hemos terminado, vamos a mejorar su aspecto exterior para conseguir que quede perfecto, tal y como se merece un generador de llaves.

Salimos de nuestro "Selft-Keygen.exe" y vamos a maquearlo.

Primero extraigo la imagen "BRBCM22 Imagen de bits", (la encontrareis dentro de la carpeta "Source Code") , aunque podríamos utilizar cualquier otra Imagen de bits, pero ya que tenemos ésta y con las mediadas exactas pues la aprovechamos y punto.



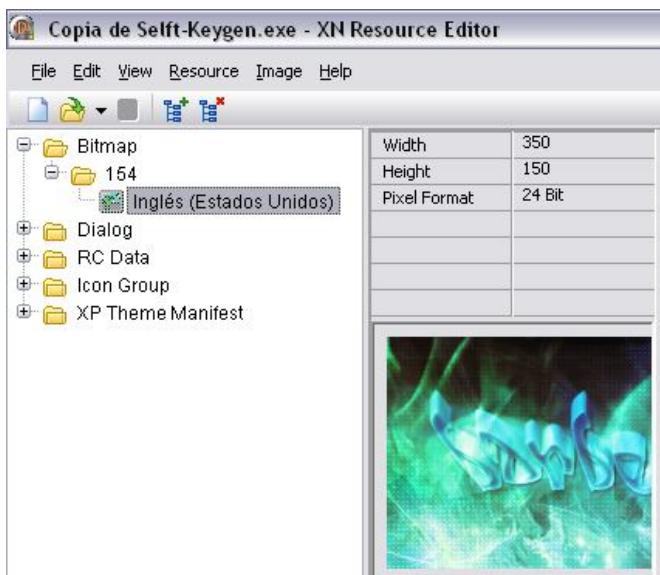
Y con cualquier editor de fotos la ponemos a nuestro gusto



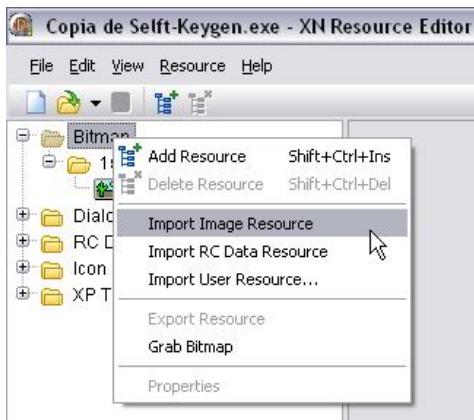
Segundo, hacemos una copia de nuestro "Selft-Keygen" con la cual trabajaremos (si la cosa va mal siempre tendremos el original)



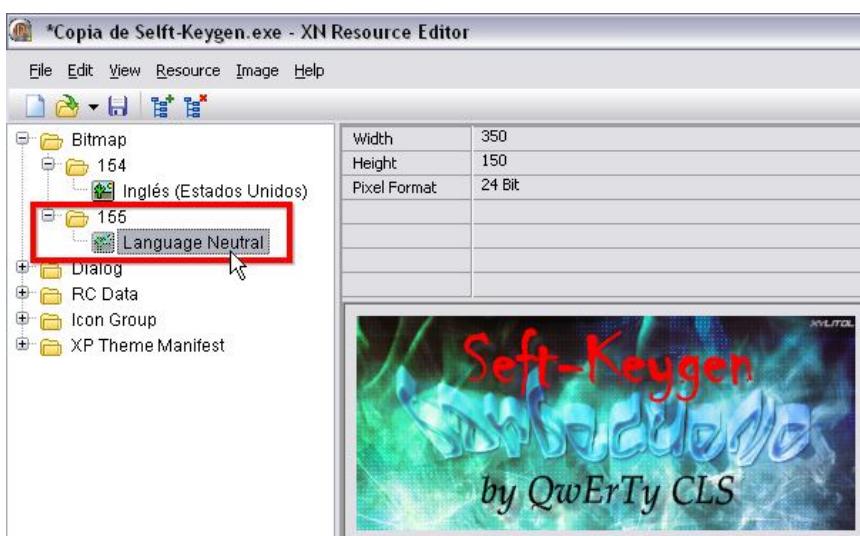
Tercero, abrimos nuestra "**Copia Seft-Keygen**" con la Tool "**XN Resource Editor**", desplegamos "**Bitmap**" (formato mapa de bits), y vemos que en la carpeta **154** se encuentra el recurso original del exe.,



Ahora nos posicionamos sobre "**Resource**" del menú principal, o si preferimos sobre la misma carpeta "**Bitmap**", click derecho de ratón, le damos a "**Import Image Resource**", buscamos la ruta donde tenemos guardado nuestra "**BRBCM22 (Imagen de bits)**" maqueada para la ocasión



La cargamos, y el nuevo recurso importado vemos que lo tenemos en la carpeta **155** que se nos ha creado.

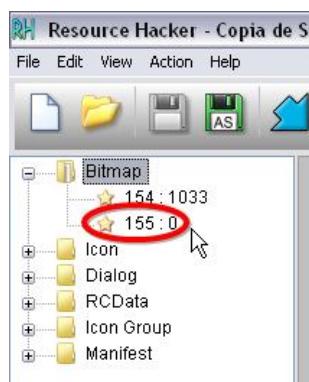


Y lo guardamos.

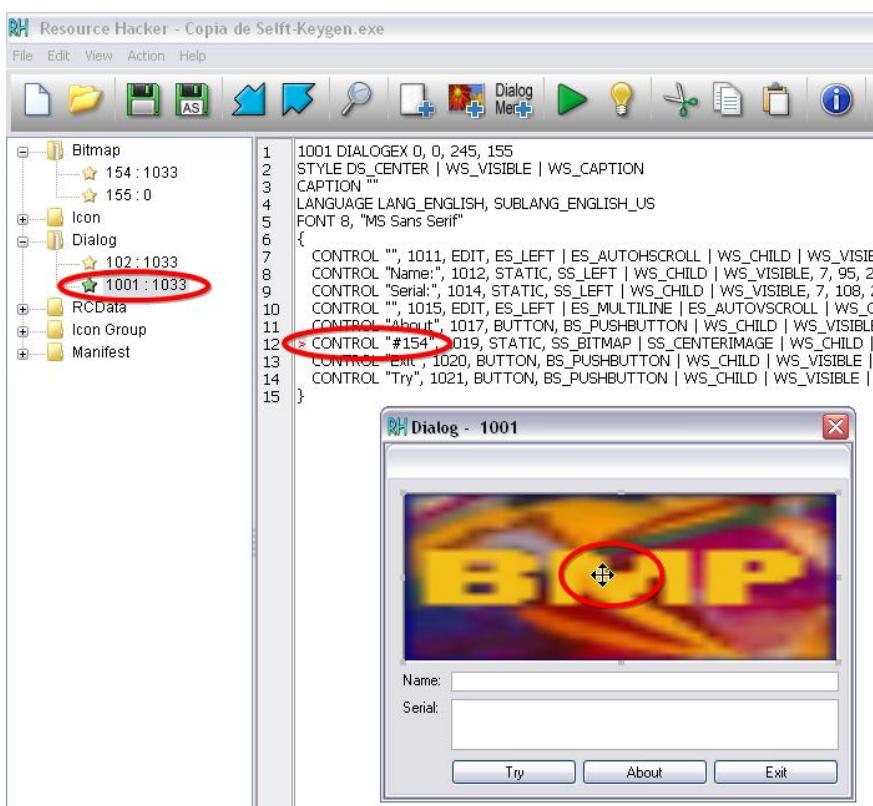


Ya podemos salir de esta Tool.

Cuarto: Ahora abrimos nuestra "*Copia Selft-Keygen.exe*" con el "**ResourceHacker**", desplegamos la carpeta "Bitmap" y vemos que tenemos cargado el nuevo recurso. "155:0"



Desplegamos la carpeta "Dialog", y en "1001 : 1033" se nos aparece los recursos de nuestro Crackme en la ventana central, nos posicionamos con el cursor sobre la imagen "BMP"



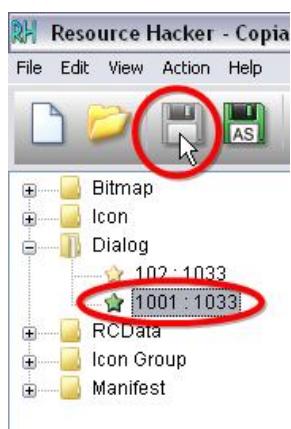
Y en "> CONTROL "#154", (imagen original) lo substituimos directamente por "#155", (imagen maqueada)

```
    CONTROL "", 1011, EDIT, ES_LEFT |  
    CONTROL "Name:", 1012, STATIC, S:  
    CONTROL "Serial:", 1014, STATIC, S:  
    CONTROL "", 1015, EDIT, ES_LEFT |  
    CONTROL "About", 1017, BUTTON,  
    > CONTROL "#155" 1019, STATIC, SS:  
    CONTROL "Exit", 1020, BUTTON, BS  
    CONTROL "Try", 1021, BUTTON, BS,
```

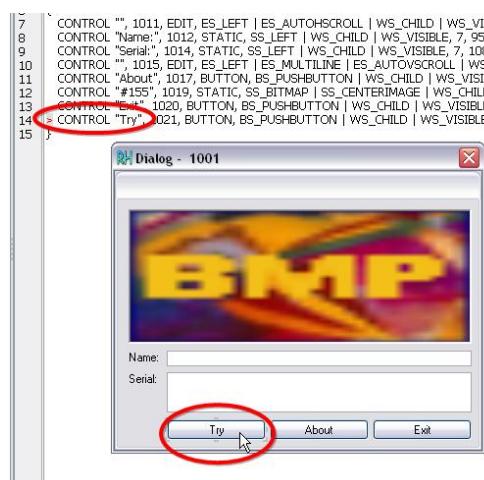
Ahora le damos a Compilar



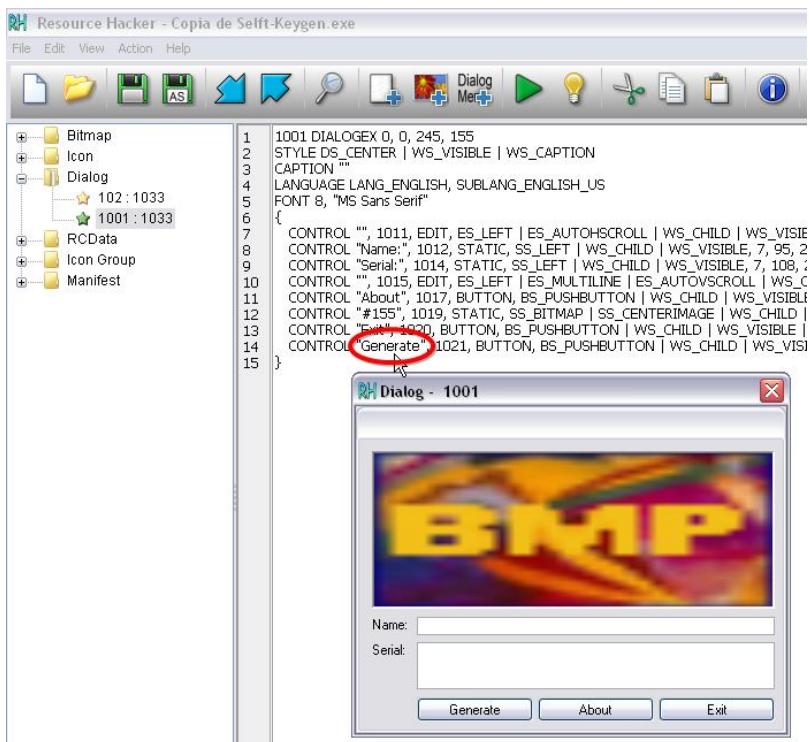
Y se nos queda la estrellita en rojo, por último guardamos los cambios



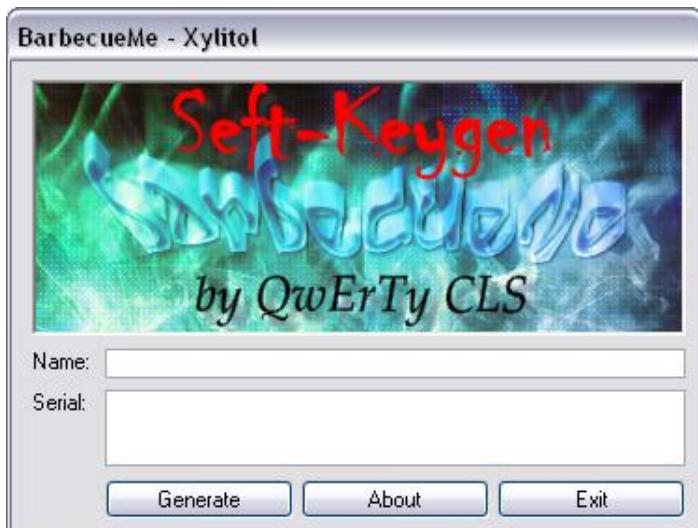
La estrellita se nos pone verde, indicativo de que se han guardado correctamente los cambios.



Quinto: cambiamos el "Caption" del control >CONTROL "Try" por "Generate", volvemos a "Compilar" y a "Guardar" los cambios y nos queda así:



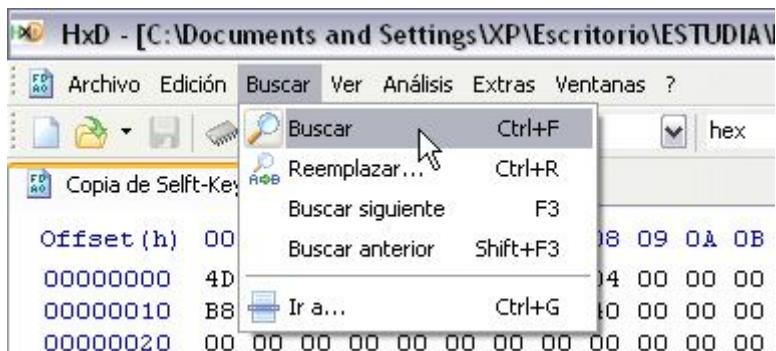
Ahora salimos del "Resource Hacker", ejecutamos el "Copia Seft-Keygen.exe" Y observamos el menudo cambio de imagen que le hemos dado.



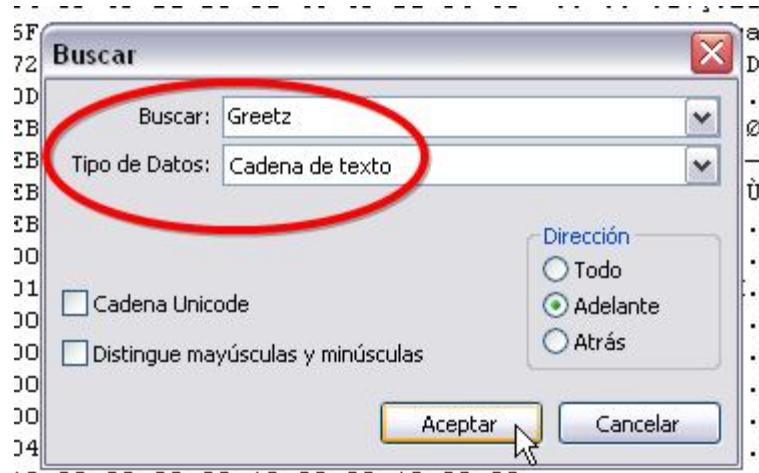
Para finalizar nos queda otro detalle por solucionar, y con que nos gustan las cosas bien hechas, le damos a "About" y apuntamos cualquier palabra que sale en la cinta, yo por ejemplo apunto "Greetz", click derecho de ratón para salir de "About" y salimos también del "Copia Seft-Keygen.exe"



Sexto: Lo abrimos con un editor Hexadecimal que nos venga en gana, yo utilizaré el "HxD"



Buscamos la palabra "**Greetz**",



Le damos a "Aceptar", y apareceremos aquí,

OA 50 72 6F 75 64 6C 79 20 Xylitol.Proudly
73 OA OA 4D 79 20 46 69 72 presents..My Fir
74 6F 2D 4B 65 79 67 65 6E st Crypto-Keygen
62 65 63 75 65 4D 65 OA OA Me: BarbecueMe..
70 21 64 33 72 20 20 20 20 GFX: xsp!d3r
46 58 3A 20 53 69 6D 70 75 .SFX: Simpu
6C 6C 69 20 62 79 20 4D 79 kka chilli by My
6F 75 64 OA 2D 2D 2D 2D 2D stic Cloud.----
2D 2D 2D 2D 2D 2D 2D 2D -----
OA 47 72 65 65 74 7A 20 32 ----- .Greetz 2
6C 61 79 65 52 OA 78 73 70 ..BytePlayeR.xsp
63 72 79 70 74 6F OA 48 79 !d3r.Encrypto.Hy
OA 48 61 63 6B 5F 54 68 45 perlisk.Hack_The
53 65 OA 4D 69 53 53 69 4E _PaRaDiSe.MiSSiN
54 45 53 OA 76 30 30 64 30 G iN ByTES.v00d0
5A 75 67 6F OA 43 6F 73 74 Ochile.Zugo.Cost
6F OA 62 6C 61 63 6B 70 69 y.Rizer0.blackpi
2D 2D 2D 2D 2D 2D 2D 2D rate.-----
2D 2D 2D 2D 2D 2D 2D 2D -----
2F 2F 78 79 6C 69 74 6F 6C -.http://xylitol
72 OA OA 4E 2D 6A 30 79 20 .free.fr..N-jOy
68 20 21 OA OA 00 4C 75 63 Blowfish !...Luc
73 6F 6C 65 00 00 00 00 00 ida Console.....
00 00 00 00 00 00 00 00 00

Lo cambio por:

```

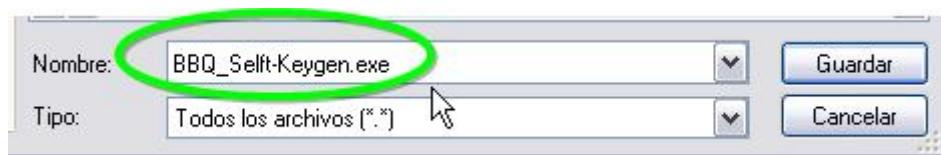
00 00 00 00 00 . .
20 20 20 20 20
20 20 53 65 6C Sel
79 20 51 77 45 ft-Keygen by QwE
20 20 20 20 20 rTy CLS
20 20 20 20 20
20 20 20 20 20
20 20 20 20 20
2D 2D 2D 2D 2D .-----
2D 2D 2D 2D 2D -----
65 74 7A 20 20 -----Greetz
61 74 69 6E 6F .. CracksLatino
20 20 0A 20 20 s .
20 20 20 20 20
20 20 20 20 20
20 20 20 20 20
20 20 20 20 20
20 20 20 20 20 Salu2
20 20 20 20 20
20 20 20 20 20
20 20 20 20 20
20 20 20 20 20
00 00 00 00 00 .....
00 00 00 00 00 .....
00 00 00 00 00 .....

```

Guardo los cambios, ("Guardar como..." por si las moscas). Debo confesarles que hacer este cambio me llevó su tiempo para que se viera el texto perfectamente centrado en el mensaje en cinta, pero es cuestión de práctica.



Le cambiamos el nombre, y le ponemos "BBQ_Selft-Keygen.exe"



y salimos del Editor Hexadecimal.

nos vamos a la ruta donde tenemos nuestro flamante "BBQ_Selft-Keygen.exe" final,



lo miramos fijamente durante unos segundos, reconocemos que estamos nerviosos, sacamos de nuevo nuestro pañuelo y volvemos a secar el sudor de nuestra frente.....(más que nada para que con el goteo no se moje el teclado, que a estas alturas del tute nos podría crear un desafortunado cortocircuito y irse todo nuestro trabajo a hacer puñetas), observamos también que ya no nos queda ni un sorbo de cerveza, y armamos de valor....., ejecutamos el "BBQ_Seft-Keygen.exe", le damos a "About" y afortunadamente comprobamos que el cambio que hemos hecho también a surtido efecto.... y aquí lo tenemos.....



Espectacular, espléndido, magnífico..... con su movimiento de cinta de subida y bajada ininterrumpido, y con su musiquita original.

Probamos, y nos muestra los seriales autogenerados:



Comprobamos los Seriales y.....

BarbecueMe - Xylitol



BarbecueMe - Xylitol



Van a la perfección....

*Hemos transformado el peligroso tiburón en una inocente sardina, je,je,je...
Que contento estoy....y que bien me lo he pasado.*

Hasta el próximo tute.

.....MISIÓN CUMPLIDA.....



Mis agradecimientos infinitos a

CracksLatinoS

Quien no añade nada a sus conocimientos, los disminuye.

El Talmud

Salu2

QwErTy CLS

23 de Abril de 2018