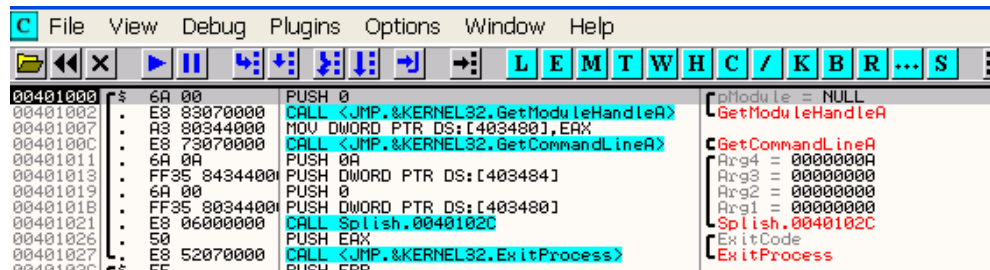


INTRODUCCIÓN AL CRACKING CON OLLYDBG PARTE 15

Bueno antes de seguir con el ultimo hardcoded, mostraremos la solución del que quedo pendiente como tarea, el SPLISH que es bien sencillo.

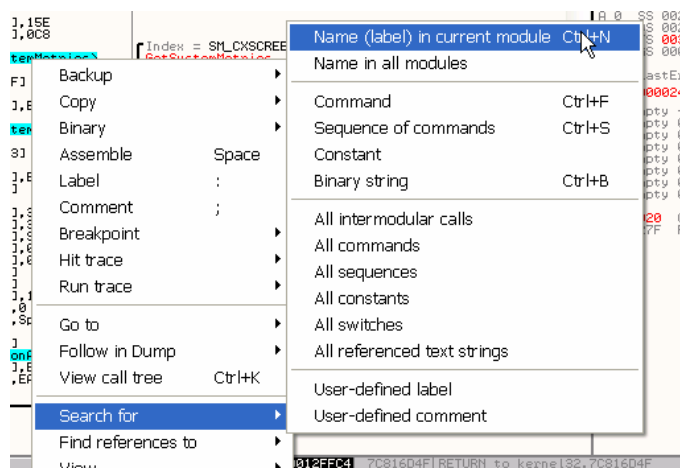
Abrámoslo en OLLYDBG.



The screenshot shows the OLLYDBG interface. The main window displays assembly code starting at address 00401000. The code includes instructions like PUSH 0, CALL <JMP.<KERNEL32.GetModuleHandleA>, MOV DWORD PTR DS:[403480],EAX, CALL <JMP.<KERNEL32.GetCommandLineA>, PUSH 0A, PUSH DWORD PTR DS:[403484], PUSH 0, PUSH DWORD PTR DS:[403480], CALL Splish.0040102C, PUSH EAX, CALL <JMP.<KERNEL32.ExitProcess>, and PUSH EBP. On the right, a call stack is visible, showing the current function as GetModuleHandleA, followed by GetCommandLineA, and then Splish.0040102C.

Allí arranco en OLLYDBG y se detuvo en el ENTRY POINT.

Si vemos las apis que utiliza con click derecho SEARCH FOR-NAME (LABEL) IN CURRENT MODULE



Vemos la lista de apis que utiliza

Address	Section	Type	Name	Comment
00402070	.rdata	Import	USER32.BeginPaint	
00402004	.rdata	Import	GDI32.CreatePatternBrush	
00402060	.rdata	Import	USER32.CreateWindowExA	
00402068	.rdata	Import	USER32.DefWindowProcA	
00402064	.rdata	Import	USER32.DispatchMessageA	
0040203C	.rdata	Import	USER32.EndPaint	
00402018	.rdata	Import	KERNEL32.ExitProcess	
0040205C	.rdata	Import	USER32.GetClientRect	
00402014	.rdata	Import	KERNEL32.GetCommandLineA	
00402058	.rdata	Import	USER32.GetMessageA	
00402010	.rdata	Import	KERNEL32.GetModuleHandleA	
00402034	.rdata	Import	USER32.GetSystemMetrics	
0040200C	.rdata	Import	KERNEL32.GetTickCount	
00402024	.rdata	Import	USER32.GetWindowTextA	
00402020	.rdata	Import	USER32.LoadBitmapA	
00402038	.rdata	Import	USER32.LoadCursorA	
00402028	.rdata	Import	USER32.LoadIconA	
0040202C	.rdata	Import	USER32.LoadMenuA	
00402030	.rdata	Import	USER32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	
0040206C	.rdata	Import	USER32.PostQuitMessage	
00402074	.rdata	Import	USER32.RegisterClassExA	
00402040	.rdata	Import	USER32.SendMessageA	
00402000	.rdata	Import	GDI32.SetBkMode	
00402044	.rdata	Import	USER32.SetFocus	
00402048	.rdata	Import	USER32.SetMenu	
0040204C	.rdata	Import	USER32.ShowWindow	
00402050	.rdata	Import	USER32.TranslateMessage	
00402054	.rdata	Import	USER32.UpdateWindow	

The screenshot shows the 'Search for' menu in a debugger. The menu is open, displaying various search criteria. The 'All referenced text strings' option is highlighted in blue. A mouse cursor is pointing at this option. The menu also includes options like 'Name in all modules', 'Command', 'Sequence of commands', 'Constant', 'Binary string', 'All intermodular calls', 'All commands', 'All sequences', 'All constants', 'All switches', 'User-defined label', and 'User-defined comment'. The background shows a list of memory addresses and their corresponding values.

Address	Value
00000000	00000000
00000001	00000000
00000002	00000000
00000003	00000000
00000004	00000000
00000005	00000000
00000006	00000000
00000007	00000000
00000008	00000000
00000009	00000000
0000000A	00000000
0000000B	00000000
0000000C	00000000
0000000D	00000000
0000000E	00000000
0000000F	00000000
00000010	00000000
00000011	00000000
00000012	00000000
00000013	00000000
00000014	00000000
00000015	00000000
00000016	00000000
00000017	00000000
00000018	00000000
00000019	00000000
0000001A	00000000
0000001B	00000000
0000001C	00000000
0000001D	00000000
0000001E	00000000
0000001F	00000000
00000020	00000000
00000021	00000000
00000022	00000000
00000023	00000000
00000024	00000000
00000025	00000000
00000026	00000000
00000027	00000000
00000028	00000000
00000029	00000000
0000002A	00000000
0000002B	00000000
0000002C	00000000
0000002D	00000000
0000002E	00000000
0000002F	00000000

Address	Disassembly	Text string
00401080	PUSH 0	Initial CPU selection)
00401085	MOV DWORD PTR SS:[EBP-8],Splish.00403060	ASCII "OurWindow"
00401108	PUSH Splish.00403060	ASCII "Splish, Splash"
00401110	PUSH Splish.00403060	ASCII "OurWindow"
00401181	PUSH Splish.00403060	ASCII "Hard Coded:"
00401186	PUSH Splish.00403060	ASCII "static"
004011D0	PUSH Splish.00403052	ASCII "Name"
004011F7	PUSH Splish.00403060	ASCII "static"
00401207	PUSH Splish.00403098	ASCII "Serial:"
0040120C	PUSH Splish.00403060	ASCII "static"
00401237	PUSH Splish.0040305B	ASCII "edit"
0040126A	PUSH Splish.0040305B	ASCII "edit"
00401290	PUSH Splish.0040305B	ASCII "edit"
004012D4	PUSH Splish.00403028	ASCII "Check Hardcoded"
004012DE	PUSH Splish.00403019	ASCII "button"
0040130F	PUSH Splish.00403030	ASCII "Name/Serial Check"
00401314	PUSH Splish.00403019	ASCII "button"
00401363	ASCII "Hardcoded",0	
0040138E	ASCII "Congratulations,"	
0040139C	ASCII "you got the har"	
004013A0	ASCII "d coded serial",0	
004013BF	PUSH Splish.00403060	ASCII "Splish, Splash"
004013C4	PUSH Splish.004013BE	ASCII "Congratulations, you got the hard coded serial"
004013D4	PUSH Splish.00403060	ASCII "Splish, Splash"
004013D9	PUSH Splish.00403067	ASCII "Sorry, please try again."
00401433	PUSH Splish.00403060	ASCII "Splish, Splash"
00401438	PUSH Splish.004030C0	ASCII "Your mission is to disable the Splash Screen, find the hardcod
00401477	ASCII "Splash_Class",0	
0040148C	PUSH Splish.00403080	ASCII "MyBmp"
004014D0	MOV DWORD PTR SS:[EBP-8],Splish.00401477	ASCII "Splash_Class"
00401520	PUSH Splish.00403080	ASCII "Splish, Splash"
0040152D	PUSH Splish.00401477	ASCII "Splash_Class"
00401698	PUSH Splish.00403060	ASCII "Splish, Splash"
004016A3	PUSH Splish.00403060	ASCII "Please enter your name."
00401695	PUSH Splish.00403060	ASCII "Splish, Splash"
00401696	PUSH Splish.00403088	ASCII "Please enter your serial number."
004016C1	PUSH Splish.00403060	ASCII "Splish, Splash"
004016D4	PUSH Splish.00403042	ASCII "Good job, now keygen it."
004016E4	PUSH Splish.00403060	ASCII "Splish, Splash"
004016E5	PUSH Splish.00403067	ASCII "Sorry, please try again."

Allí vemos la zona donde trabaja con el serial que introducimos

```

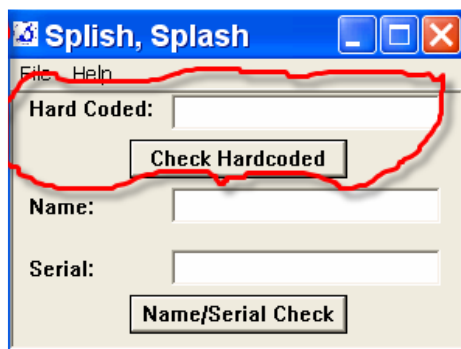
00401350 JMP SHORT SpIsh,00401350
00401351 ASCI1 "HardCoded",0
00401352 MOV ESI,ESI
00401353 MOV EDI,EDI
00401354 MOV EAX,EAX
00401355 MOV ECX,ECX
00401356 MOV EDX,EDX
00401357 MOV EBP,EBP
00401358 MOV ESP,ESP
00401359 MOV EIP,EIP
0040135A JMP SHORT SpIsh,0040135A
0040135B MOV ESI,ESI
0040135C MOV EDI,EDI
0040135D MOV EAX,EAX
0040135E MOV ECX,ECX
0040135F MOV EDX,EDX
00401360 MOV EBP,EBP
00401361 MOV ESP,ESP
00401362 MOV EIP,EIP
00401363 JMP SHORT SpIsh,00401363
00401364 MOV ESI,ESI
00401365 MOV EDI,EDI
00401366 MOV EAX,EAX
00401367 MOV ECX,ECX
00401368 MOV EDX,EDX
00401369 MOV EBP,EBP
0040136A MOV ESP,ESP
0040136B MOV EIP,EIP
0040136C JMP SHORT SpIsh,0040136C
0040136D MOV ESI,ESI
0040136E MOV EDI,EDI
0040136F MOV EAX,EAX
00401370 MOV ECX,ECX
00401371 MOV EDX,EDX
00401372 MOV EBP,EBP
00401373 MOV ESP,ESP
00401374 MOV EIP,EIP
00401375 JMP SHORT SpIsh,00401375
00401376 MOV ESI,ESI
00401377 MOV EDI,EDI
00401378 MOV EAX,EAX
00401379 MOV ECX,ECX
0040137A MOV EDX,EDX
0040137B MOV EBP,EBP
0040137C MOV ESP,ESP
0040137D MOV EIP,EIP
0040137E JMP SHORT SpIsh,0040137E
0040137F MOV ESI,ESI
00401380 MOV EDI,EDI
00401381 MOV EAX,EAX
00401382 MOV ECX,ECX
00401383 MOV EDX,EDX
00401384 MOV EBP,EBP
00401385 MOV ESP,ESP
00401386 MOV EIP,EIP
00401387 JMP SHORT SpIsh,00401387
00401388 MOV ESI,ESI
00401389 MOV EDI,EDI
0040138A MOV EAX,EAX
0040138B MOV ECX,ECX
0040138C MOV EDX,EDX
0040138D MOV EBP,EBP
0040138E MOV ESP,ESP
0040138F MOV EIP,EIP
00401390 JMP SHORT SpIsh,00401390
00401391 MOV ESI,ESI
00401392 MOV EDI,EDI
00401393 MOV EAX,EAX
00401394 MOV ECX,ECX
00401395 MOV EDX,EDX
00401396 MOV EBP,EBP
00401397 MOV ESP,ESP
00401398 MOV EIP,EIP
00401399 JMP SHORT SpIsh,00401399
0040139A MOV ESI,ESI
0040139B MOV EDI,EDI
0040139C MOV EAX,EAX
0040139D MOV ECX,ECX
0040139E MOV EDX,EDX
0040139F MOV EBP,EBP
004013A0 MOV ESP,ESP
004013A1 MOV EIP,EIP
004013A2 JMP SHORT SpIsh,004013A2
004013A3 MOV ESI,ESI
004013A4 MOV EDI,EDI
004013A5 MOV EAX,EAX
004013A6 MOV ECX,ECX
004013A7 MOV EDX,EDX
004013A8 MOV EBP,EBP
004013A9 MOV ESP,ESP
004013AA MOV EIP,EIP
004013AB JMP SHORT SpIsh,004013AB
004013AC MOV ESI,ESI
004013AD MOV EDI,EDI
004013AE MOV EAX,EAX
004013AF MOV ECX,ECX
004013B0 MOV EDX,EDX
004013B1 MOV EBP,EBP
004013B2 MOV ESP,ESP
004013B3 MOV EIP,EIP
004013B4 JMP SHORT SpIsh,004013B4
004013B5 MOV ESI,ESI
004013B6 MOV EDI,EDI
004013B7 MOV EAX,EAX
004013B8 MOV ECX,ECX
004013B9 MOV EDX,EDX
004013BA MOV EBP,EBP
004013BB MOV ESP,ESP
004013BC MOV EIP,EIP
004013BD JMP SHORT SpIsh,004013BD
004013BE MOV ESI,ESI
004013BF MOV EDI,EDI
004013C0 MOV EAX,EAX
004013C1 MOV ECX,ECX
004013C2 MOV EDX,EDX
004013C3 MOV EBP,EBP
004013C4 MOV ESP,ESP
004013C5 MOV EIP,EIP
004013C6 JMP SHORT SpIsh,004013C6
004013C7 MOV ESI,ESI
004013C8 MOV EDI,EDI
004013C9 MOV EAX,EAX
004013CA MOV ECX,ECX
004013CB MOV EDX,EDX
004013CC MOV EBP,EBP
004013CD MOV ESP,ESP
004013CE MOV EIP,EIP
004013CF JMP SHORT SpIsh,004013CF
004013D0 MOV ESI,ESI
004013D1 MOV EDI,EDI
004013D2 MOV EAX,EAX
004013D3 MOV ECX,ECX
004013D4 MOV EDX,EDX
004013D5 MOV EBP,EBP
004013D6 MOV ESP,ESP
004013D7 MOV EIP,EIP
004013D8 JMP SHORT SpIsh,004013D8
004013D9 MOV ESI,ESI
004013DA MOV EDI,EDI
004013DB MOV EAX,EAX
004013DC MOV ECX,ECX
004013DD MOV EDX,EDX
004013DE MOV EBP,EBP
004013DF MOV ESP,ESP
004013E0 MOV EIP,EIP
004013E1 JMP SHORT SpIsh,004013E1
004013E2 MOV ESI,ESI
004013E3 MOV EDI,EDI
004013E4 MOV EAX,EAX
004013E5 MOV ECX,ECX
004013E6 MOV EDX,EDX
004013E7 MOV EBP,EBP
004013E8 MOV ESP,ESP
004013E9 MOV EIP,EIP
004013EA JMP SHORT SpIsh,004013EA
004013EB MOV ESI,ESI
004013EC MOV EDI,EDI
004013ED MOV EAX,EAX
004013EE MOV ECX,ECX
004013EF MOV EDX,EDX
004013F0 MOV EBP,EBP
004013F1 MOV ESP,ESP
004013F2 MOV EIP,EIP
004013F3 JMP SHORT SpIsh,004013F3
004013F4 MOV ESI,ESI
004013F5 MOV EDI,EDI
004013F6 MOV EAX,EAX
004013F7 MOV ECX,ECX
004013F8 MOV EDX,EDX
004013F9 MOV EBP,EBP
004013FA MOV ESP,ESP
004013FB MOV EIP,EIP
004013FC JMP SHORT SpIsh,004013FC
004013FD MOV ESI,ESI
004013FE MOV EDI,EDI
004013FF MOV EAX,EAX
00401400 MOV ECX,ECX
00401401 MOV EDX,EDX
00401402 MOV EBP,EBP
00401403 MOV ESP,ESP
00401404 MOV EIP,EIP
00401405 JMP SHORT SpIsh,00401405
00401406 MOV ESI,ESI
00401407 MOV EDI,EDI
00401408 MOV EAX,EAX
00401409 MOV ECX,ECX
0040140A MOV EDX,EDX
0040140B MOV EBP,EBP
0040140C MOV ESP,ESP
0040140D MOV EIP,EIP
0040140E JMP SHORT SpIsh,0040140E
0040140F MOV ESI,ESI
00401410 MOV EDI,EDI
00401411 MOV EAX,EAX
00401412 MOV ECX,ECX
00401413 MOV EDX,EDX
00401414 MOV EBP,EBP
00401415 MOV ESP,ESP
00401416 MOV EIP,EIP
00401417 JMP SHORT SpIsh,00401417
00401418 MOV ESI,ESI
00401419 MOV EDI,EDI
0040141A MOV EAX,EAX
0040141B MOV ECX,ECX
0040141C MOV EDX,EDX
0040141D MOV EBP,EBP
0040141E MOV ESP,ESP
0040141F MOV EIP,EIP
00401420 JMP SHORT SpIsh,00401420
00401421 MOV ESI,ESI
00401422 MOV EDI,EDI
00401423 MOV EAX,EAX
00401424 MOV ECX,ECX
00401425 MOV EDX,EDX
00401426 MOV EBP,EBP
00401427 MOV ESP,ESP
00401428 MOV EIP,EIP
00401429 JMP SHORT SpIsh,00401429
0040142A MOV ESI,ESI
0040142B MOV EDI,EDI
0040142C MOV EAX,EAX
0040142D MOV ECX,ECX
0040142E MOV EDX,EDX
0040142F MOV EBP,EBP
00401430 MOV ESP,ESP
00401431 MOV EIP,EIP
00401432 JMP SHORT SpIsh,00401432
00401433 MOV ESI,ESI
00401434 MOV EDI,EDI
00401435 MOV EAX,EAX
00401436 MOV ECX,ECX
00401437 MOV EDX,EDX
00401438 MOV EBP,EBP
00401439 MOV ESP,ESP
0040143A MOV EIP,EIP
0040143B JMP SHORT SpIsh,0040143B
0040143C MOV ESI,ESI
0040143D MOV EDI,EDI
0040143E MOV EAX,EAX
0040143F MOV ECX,ECX
00401440 MOV EDX,EDX
00401441 MOV EBP,EBP
00401442 MOV ESP,ESP
00401443 MOV EIP,EIP
00401444 JMP SHORT SpIsh,00401444
00401445 MOV ESI,ESI
00401446 MOV EDI,EDI
00401447 MOV EAX,EAX
00401448 MOV ECX,ECX
00401449 MOV EDX,EDX
0040144A MOV EBP,EBP
0040144B MOV ESP,ESP
0040144C MOV EIP,EIP
0040144D JMP SHORT SpIsh,0040144D
0040144E MOV ESI,ESI
0040144F MOV EDI,EDI
00401450 MOV EAX,EAX
00401451 MOV ECX,ECX
00401452 MOV EDX,EDX
00401453 MOV EBP,EBP
00401454 MOV ESP,ESP
00401455 MOV EIP,EIP
00401456 JMP SHORT SpIsh,00401456
00401457 MOV ESI,ESI
00401458 MOV EDI,EDI
00401459 MOV EAX,EAX
0040145A MOV ECX,ECX
0040
```

Vemos la entrada de texto con la api `GetWindowTextA` y los `MessageBoxA` con sus respectivos textos de si acertamos o no.

Pues pongamos un BPX allí, en el call a la api GetWindowTextA para comenzar desde donde se introduce el serial que tipeamos.

<pre> 00401351 . EB 0A JMP SHORT Splish.0040135D 00401353 . 48 61 72 64 ASCII "HardCoded",0 0040135D . 6A 20 PUSH 20 0040135F . 68 15324000 PUSH Splish.00403215 00401364 . FF35 90344000 PUSH DWORD PTR DS:[4034901] 0040136A . E8 BB030000 CALL <JMP.>USER32.GetWindowTextA 0040136F . 8D05 53134000 LEA EAX, DWORD PTR DS:[401353] 00401375 . 8D1D 15324000 LEA EBX, DWORD PTR DS:[403215] 0040137B . 8A08 CMP BYTE PTR DS:[EAX],0 0040137E . 74 0C JE SHORT Splish.00401380 00401380 . 8A08 MOV CL, BYTE PTR DS:[EAX] 00401382 . 8A13 MOV DL, BYTE PTR DS:[EAX] 00401384 . 38D1 CMP CL, DL 00401386 . 75 4A JNE SHORT Splish.004013D2 00401388 . 40 INC EAX 00401389 . 43 INC EBX 0040138A . EB EF JMP SHORT Splish.0040137B 0040138C . EB 2F JMP SHORT Splish.0040136D 0040138E . 43 6F 6E 67 ASCII "Congratulations," 0040139E . 20 79 6F 75 ASCII " you got the har" 004013AE . 64 20 63 6F ASCII "d coded serial",0 004013B0 . 6A 00 PUSH 0 004013B2 . 68 0A304000 PUSH Splish.0040300A 004013C4 . 68 8E134000 PUSH Splish.0040138E 004013C9 . 6A 00 PUSH 0 004013CB . E8 79030000 CALL <JMP.>USER32.MessageBoxA 004013D0 . EB 13 JMP SHORT Splish.004013E5 004013D2 . 6A 00 PUSH 0 004013D4 . 68 0A304000 PUSH Splish.0040300A 004013D9 . 68 67304000 PUSH Splish.00403067 004013DE . 6A 00 PUSH 0 004013E0 . E8 63030000 CALL <JMP.>USER32.MessageBoxA 004013E5 . 61 POP EDI 004013E6 . EB 73 JMP SHORT Splish.0040145E </pre>	<pre> [Count = 20 (32.) Buffer = Splish.00403215 hWnd = NULL GetWindowTextA Style = MB_OK MB_APPLMODAL Title = "Splish, Splash" Text = "Congratulations, you got the hard coded serial" hOwner = NULL MessageBoxA Style = MB_OK MB_APPLMODAL Title = "Splish, Splash" Text = "Sorry, please try again." hOwner = NULL MessageBoxA </pre>
--	--

Corro el crackme con F9 o RUN



Como solo pedimos por ahora solucionar la parte de HARDCODED, tipearemos un serial falso, allí en la parte superior y apretamos el botón CHECK HARDCODED.



Para en el BPX que habíamos colocado

<pre> 00401353 . 48 61 72 64 ASCII "HardCoded",0 0040135D . 6A 20 PUSH 20 0040135F . 68 15324000 PUSH Splish.00403215 00401364 . FF35 90344000 PUSH DWORD PTR DS:[4034901] 0040136A . E8 BB030000 CALL <JMP.>USER32.GetWindowTextA 0040136F . 8D05 53134000 LEA EAX, DWORD PTR DS:[401353] 00401375 . 8D1D 15324000 LEA EBX, DWORD PTR DS:[403215] 0040137B . 8A08 CMP BYTE PTR DS:[EAX],0 </pre>	<pre> [Count = 20 (32.) Buffer = Splish.00403215 hWnd = 000B0680 (class='Edit',parent=00150674) GetWindowTextA </pre>
---	---

Allí vemos tanto en el stack como en las ayudas que nos muestra el OLLY que el BUFFER donde guardara el serial que tipeamos, esta en 403215.

0012FC58	000B0680	hWnd = 000B0680 (class='Edit',parent=00150674)
0012FC5C	00403215	Buffer = Splish.00403215
0012FC60	00000020	Count = 20 (32.)
0012FC64	0012FCEC	
0012FC68	00401178	Splish.00401178

Así que vayamos al dump a ver el buffer.

0012FC58	000B0680	hWnd = 000B0680 (class='Edit',parent=00150674)
0012FC5C	00403215	Buffer = Splish.00403215
0012FC60	00000020	Count = 20 (32.)
0012FC64	0012FCEC	
0012FC68	00401178	Splish.00401178

Address	Hex dump	ASCII
00403215	00 00 00 00 00 00 00 00
0040321D	00 00 00 00 00 00 00 00
00403225	00 00 00 00 00 00 00 00
0040322D	00 00 00 00 00 00 00 00
00403235	00 00 00 00 00 00 00 00
0040323D	00 00 00 00 00 00 00 00
00403245	00 00 00 00 00 00 00 00

Allí esta el buffer donde guardara el serial que tipeamos, apretemos F8 para ejecutar el CALL a la api.

Address	Hex dump	ASCII
00403215	39 38 39 38 39 38 39 38	98989898
0040321D	00 00 00 00 00 00 00 00
00403225	00 00 00 00 00 00 00 00
0040322D	00 00 00 00 00 00 00 00

Como al ejecutar con F8, ejecuta todo el call con la api incluida, pues ya tenemos el serial falso guardado en el BUFFER.

0040136F	8005 5313400	CALL <JMP.>USER32.GetWindowText
00401375	801D 1532400	LEA EAX,DWORD PTR DS:[401353]
0040137B	8038 00	LEA EBX,DWORD PTR DS:[403215]
0040137E	74 0C	CMP BYTE PTR DS:[EAX],0
00401380	8A08	JE SHORT Splish.0040138C
00401382	8A13	MOV CL,BYTE PTR DS:[EAX]
00401384	8A13	MOV CL,BYTE PTR DS:[EBX]

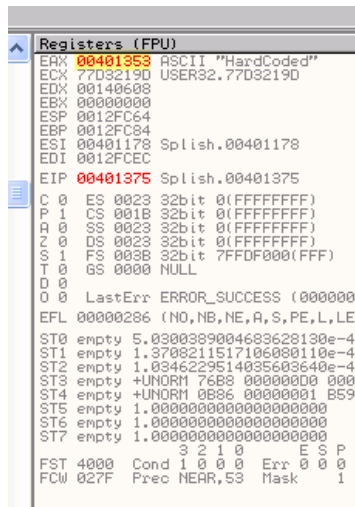
Bueno la próxima línea moverá 401353 a EAX, (recordar que los LEA no mueven el contenido de la dirección de memoria, si no solo el resultado de lo que hay entre corchetes, en este caso moverá 401353) Veamos en el DUMP dicha dirección, en la línea hacemos click derecho FOLLOW IN DUMP-MEMORY ADDRESS lo que nos muestra en el DUMP la dirección de memoria 401353.

Go to	MODAL
Follow in Dump	Selection
View call tree	Ctrl+K
Search for	Memory address

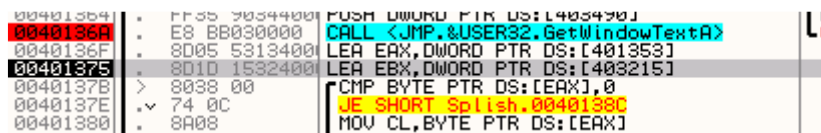
Pues 401353 apunta a la string "HARDCODED", apretemos F7 para que se ejecute el LEA.

00401410	75 1B	JNZ SHORT Splish.0040141C
00401412	6A 00	PUSH 0
Address=00401353, (ASCII "HardCoded")		
EAX=00000008		
Address	Hex dump	ASCII
00401353	48 61 72 64 43 6F 64 65	HardCode
0040135B	64 00 6A 20 68 15 32 40	d.j h320
00401363	00 FF 35 90 34 40 00 E8	. 5e40.0
0040136B	BB 03 00 00 8D 05 53 13	77.14S!!
00401373	40 00 8D 1D 15 32 40 00	@.1#320.
0040137B	80 38 00 74 0C 8A 08 8A	C8.t.e0e
00401383	13 38 D1 75 4A 40 43 EB	!!80uJ0C0

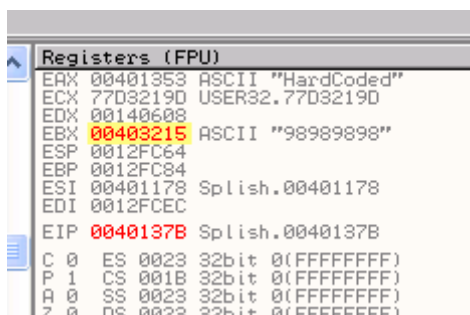
Al ejecutar queda EAX con el valor 401353 y al lado nos aparece la aclaración de que dicha dirección apunta a una string en este caso “HARDCODED”



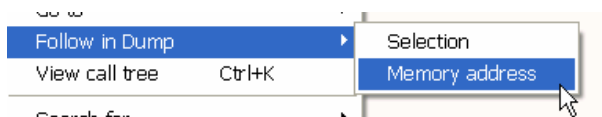
La siguiente línea mueve otra dirección a EBX en este caso 403215.



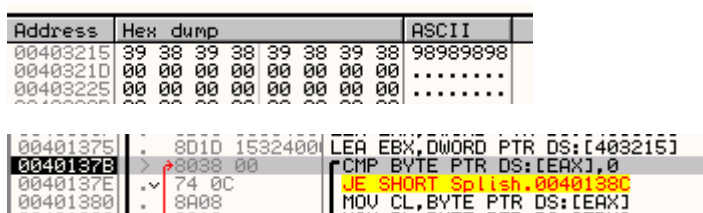
Al apretar F7, EBX quedara valiend 403215



Y OLLYDBG nos muestra a la derecha, que esa dirección apunta a la string “98989898”, que es mi serial falso y que se ve en el dump si hago como en la instrucción anterior.



Allí esta 403215 apunta a nuestro serial falso



La siguiente línea compara si el primer BYTE del contenido de EAX es cero, como EAX vale 401353

Registers (FPU)		
EAX	00401353	ASCII "HardCoded"
ECX	77D3219D	USER32.77D3219D
EDX	00140608	
EBX	00403215	ASCII "98989898"
ESP	0012FC64	
EBP	0012FC84	
ESI	00401178	Splish.00401178
EDI	0012FCEC	
EIP	0040137B	Splish.0040137B

Sabemos porque lo vimos en el DUMP que el contenido son los bytes de la cadena HARDCODED.

00401412 | 60 00 | PUSH 0
DS:[00401353]=48 ('H')
Jump from 0040138A

Address	Hex	Disasm	ASCII
00401353	48 61 72 64 43 6F 64 65	HardCode	
0040135B	64 00 6A 20 6A 15 32 40	d.i h&20	

Allí esta el primer Byte es 48, en la aclaración OLLYDBG también nos dice que esta leyendo el 48 que corresponde a la H en ASCII de la primera letra de la palabra HARDCODED, ve si es cero, como no es cero continuara.

EIP	0040137
C 0	ES 002
P 1	CS 001
A 0	SS 002
Z 0	DS 002
S 0	ES 003
T 0	CS 000
O 0	LastEr
O 0	
EFL	0000020

Como la comparación no activo el FLAG Z al no ser ambos miembros iguales, pues el JE siguiente no saltara, recordar que JE solo salta al estar activado el FLAG Z.

Pues continuamos con F7

0040137B	74 0C	JE SHORT Splish.0040138C
00401380	8A 08	MOV CL, BYTE PTR DS:[EAX]
00401382	8B 13	MOV DL, BYTE PTR DS:[EBX]
00401384	3B 01	CMPL DL, CL
00401386	75 4A	JNZ SHORT Splish.004013D2
00401388	40	INC EAX
00401389	43	INC EBX
0040138A	EB EF	JMP SHORT Splish.0040137B
0040138C	EB 2F	JMP SHORT Splish.004013BD
0040138E	43 6F 6E 67	ASCII "Congratulations,"
0040139E	20 79 6F 75	ASCII "you got the har"
004013AE	64 20 63 6F	ASCII "d coded serial",0
004013BD	6A 00	PUSH 0
004013BF	68 0A304000	PUSH Splish.0040300A
004013C4	68 8E134000	PUSH Splish.0040138E
004013C9	6A 00	PUSH 0
004013CB	E8 78030000	CALL <JMP.&USER32.MessageBoxA>
004013D0	EB 13	JMP SHORT Splish.004013E5
004013D2	6A 00	PUSH 0
004013D4	68 0A304000	PUSH Splish.0040300A
004013D9	68 67304000	PUSH Splish.00403067
004013DE	6A 00	PUSH 0
004013E0	E8 63030000	CALL <JMP.&USER32.MessageBoxA>
004013E5	61	POPAD

Aquí vemos con una visión mas ampliada, que ahora moverá el primer byte de [EAX] que es el 48 que corresponde a la palabra HARDCODED y en la siguiente línea moverá [EBX] que corresponde al primer byte de mi serial falso y los comparara si no son iguales salta a 4013D2 que es el cartel de SORRY, PLEASE TRY AGAIN.

Verifiquemos que esto es asi

Apreto F7 y mueve a CL el valor 48

Registers (FPU)				
EAX	00401353	ASCII "Har"		
ECX	77D32148	USER32.77D		
EDX	00140608			
EBX	00403215	ASCII "989"		
ESP	0012FC64			
EBP	0012FC84			
ESI	00401178	Splish.004		
EDI	0012FCEC			
EIP	00401382	Splish.004		
C 0	ES 0023	32bit 0(FF)		
P 1	CS 001B	32bit 0(FF)		
A 0	SS 0023	32bit 0(FF)		
Z 0	DS 0023	32bit 0(FF)		
S 0	FS 003B	32bit 7FFD		

CL como recordamos es el registro que corresponde a los dos ultimas cifras de ECX, alli movio el 48, lo vemos en la imagen resaltado.

0040137E	. 74 0C	JE SHORT Splish.0040138C
00401380	. 8A08	MOV CL,BYTE PTR DS:[EAX]
00401382	. 8A13	MOV DL,BYTE PTR DS:[EBX]
00401384	. 3BD1	CMP CL,DL
00401386	. 75 4A	JNZ SHORT Splish.004013D2

En la siguiente mueve a DL el primer byte de mi serial falso

00401412	. 60 00	
DS:[00403215]=39 ('9')		
DL=08 (Backspace)		

Al ejecutar con F7 vemos el 39 en DL

Registers (FPU)				
EAX	00401353	AS		
ECX	77D32148	US		
EDX	00140639			
EBX	00403215	AS		
ESP	0012FC64			
EBP	0012FC84			
ESI	00401178	Sp		
EDI	0012FCEC			
EIP	00401384	Sp		
C 0	ES 0023	32		
P 1	CS 001B	32		

Ahora compara CL con DL

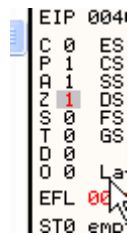
00401380	. 8A08	MOV CL,BYTE PTR DS:[EAX]
00401382	. 8A13	MOV DL,BYTE PTR DS:[EBX]
00401384	. 3BD1	CMP CL,DL
00401386	. 75 4A	JNZ SHORT Splish.004013D2
00401388	. 4A	INC EBX

La aclaración del OLLYDBG no deja dudas, compara el 39 que es el 9 de la primera cifra de mi serial falso con H que es la primera cifra de HARDCODED.

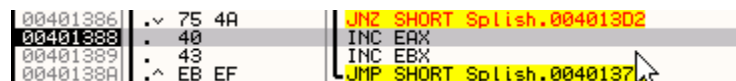
DL=39 ('9')	
CL=48 ('H')	

00401380	. 8A08	MOV CL,BYTE PTR DS:[EAX]	
00401382	. 8A13	MOV DL,BYTE PTR DS:[EBX]	
00401384	. 3BD1	CMP CL,DL	
00401386	. 75 4A	JNZ SHORT Splish.004013D2	
00401388	. 40	INC EAX	
00401389	. 43	INC EBX	
0040138A	. EB EF	JMP SHORT Splish.0040137B	
0040138C	. EB 2F	JMP SHORT Splish.004013B0	
0040138E	. 43 6F 6E 67	ASCII "Congratulations,"	
0040139E	. 20 79 6F 75	ASCII "you got the har"	
004013AE	. 64 20 63 6F	ASCII "d coded serial",0	
004013B0	. 6A 00	PUSH 0	
004013BF	. 68 0A304000	PUSH Splish.0040300A	
004013C4	. 68 8E134000	PUSH Splish.0040138E	
004013C9	. 6A 00	PUSH 0	
004013CB	. E8 70030000	CALL <JMP.&USER32.MessageBoxA>	Style = MB_OK MB_APPLMODAL
004013D0	. EB 13	JMP SHORT Splish.004013E5	Title = "Splish, Splash"
004013D2	. 6A 00	PUSH 0	Text = "Congratulations, you got th
004013D4	. 68 0A304000	PUSH Splish.0040300A	hOwner = NULL
004013D9	. 68 67304000	PUSH Splish.00403067	MessageBoxA
004013DE	. 6A 00	PUSH 0	
004013E0	. E8 63030000	CALL <JMP.&USER32.MessageBoxA>	Style = MB_OK MB_APPLMODAL
004013F0	. 61	POP EAX	Title = "Splish, Splash"

Vemos que al no ser iguales salta al cartel de error, si fueran iguales y no saltara lo cual podemos hacer fácil cambiando el FLAG Z con doble click.



Ahora Z vale 1 como si en la comparación ambos bytes fueron iguales y la resta entre ambos dio cero y activo el FLAG Z o cero.

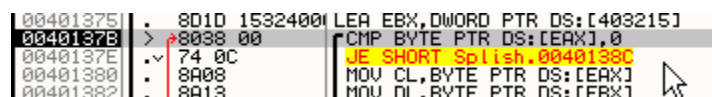


Vemos que en este caso INCREMENTA EAX y EBX y salta con un JMP al inicio del LOOP

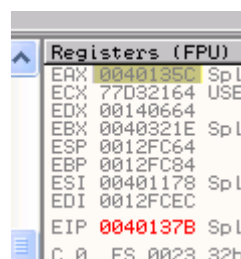


Ahora EAX apunta al segundo byte de HARDCODED y vemos que el loop se repetirá comparando byte a byte, hasta que sea cero el contenido de EAX o sea cuando se terminen los bytes de la palabra HARDCODED ya que en una string, luego de su ultimo carácter, hay un cero.

Pues entonces al incrementar EAX y EBX ira leyendo y comparando los segundos bytes, si son iguales, repetira el loop y hará lo mismo con los terceros y asi, una vez que se completo la palabra HARDCODED si todos los bytes en la comparación CL con DL han sido iguales, no saltará a mostrar el mensaje de SORRY y luego aquí



Como EAX ya termino de apuntar a todos los bytes de la palabra HARDCODED pues llego al cero del final de la string veamos.



Allí EAX apunta a 40135C que en el DUMP vemos que es el cero al final de la string entonces al ser la comparación ambos miembros iguales

Address	Hex dump	ASCII
00401354	61 72 64 43 6F 64 65 64	ardCoded
0040135C	00 6A 20 68 15 32 40 00	.j h320.
00401364	FF 35 90 34 40 00 E8 0B	5E40.00

El JE salta y nos saca del LOOP


```

00401375 | 8D1D 1532400 | LEA EBX,DWORD PTR DS:[403215]
0040137B | 8038 00 | CMP BYTE PTR DS:[EAX],0
0040137E | 74 0C | JE SHORT Splish.0040138C
00401380 | 8A08 | MOV CL,BYTE PTR DS:[EAX]
00401382 | 8A13 | MOV DL,BYTE PTR DS:[EBX]
00401384 | 38D1 | CMP CL,DL
00401386 | 75 4A | JNZ SHORT Splish.004013D2
00401388 | 40 | INC EAX
00401389 | 43 | INC EBX
0040138A | EB EF | JMP SHORT Splish.0040137B
0040138C | EB 2F | JMP SHORT Splish.00401380

```

Y nos lleva al cartel de CORRECTO (porque cambiamos los flags Z de las comparaciones byte a byte jeje)

```

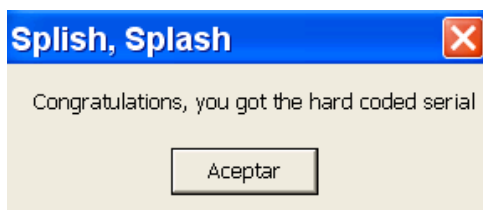
00401389 | 43 | INC EBX
0040138A | EB EF | JMP SHORT Splish.0040137B
0040138C | EB 2F | JMP SHORT Splish.00401380
0040138E | 6F 67 | ASCII "Congratulations,"
00401390 | 20 79 6F 75 | ASCII " you got the har,"
00401392 | 64 20 63 6F | ASCII "d coded serial",0
00401394 | 6A 00 | PUSH 0
00401396 | 68 0A304000 | PUSH Splish.0040300A
00401398 | 68 8E134000 | PUSH Splish.0040138E
0040139A | 6A 00 | PUSH 0
0040139C | 78 030000 | CALL JMP, &USER32.MessageBoxA
0040139E | EB 13 | JMP SHORT Splish.004013E5

```

Por supuesto yo lo hice invirtiendo en cada comparación de CL con DL el flag Z, para que el programa crea que son iguales si no saltará al cartel malo y no llegaré aquí, pero de todas formas ya sabemos entonces que el serial correcto es la palabra HardCoded respetando mayúsculas y minúsculas porque tienen diferente valor ASCII.



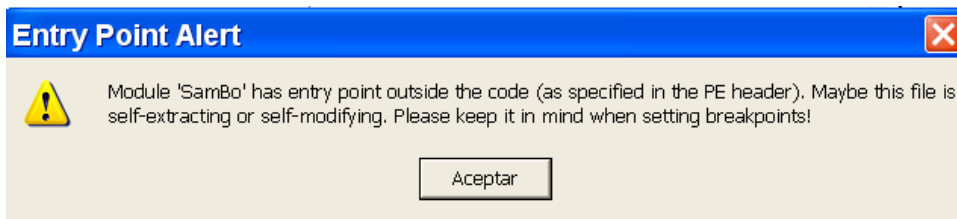
Quito todos los BPX y tipo el serial que halle, y al apretar CHECK HARDCODED



Otro vencido, si ustedes están leyendo esto pues, felicitaciones también para ustedes pues lo han vencido también, si no no podrían leer esta lección y si el password del rar se los paso un amigo, mejor releer todo y practicar que haciendo trampa no se aprende, jeje.

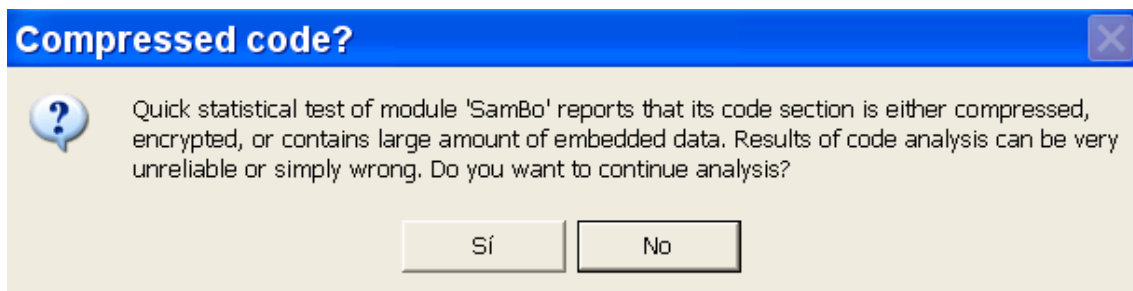
Bueno pues vamos con el ultimo HARDCODED y terminamos ya con esto, para pasar el siguiente tema en la parte 16.

Bueno el siguiente crackme es algo diferente a los anteriores es al llamado SAMBO y lo arrancamos en OLLYDBG.

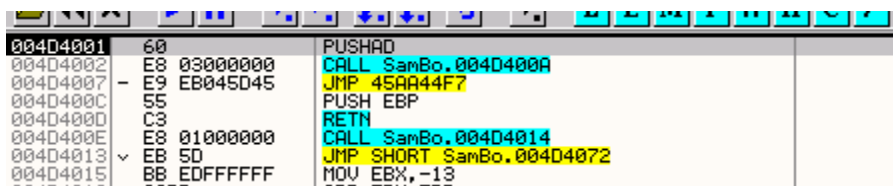


Nos sale este cartelito de OLLYDBG que nos avisa que algo pasa, allí dice que la fila se auto-descomprime o se auto modifica, lo que en la jerga del cracking se llama una fila empacada, comprimida o empaquetada, lo cual estudiaremos en profundidad mas adelante, pero igual podemos a pesar de que este empacada, correrla en OLLYDBG y tratar de hallar el hardcoded serial.

Aceptamos el aviso de OLLYDBG y llegamos al EP.

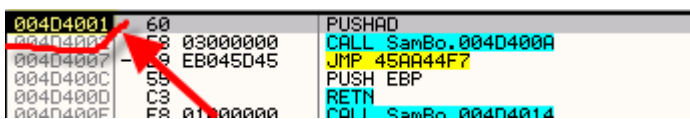


Otro avisito en este caso antes de analizar OLLYDBG nos avisa que es un programa empacado, comprimido, encriptado y que analizar eso es mas inútil que cenicerero de moto, jeje, pues el programa se ira desempacando mientras corre, por lo cual, elegimos NO, para que no lo analice por ahora.

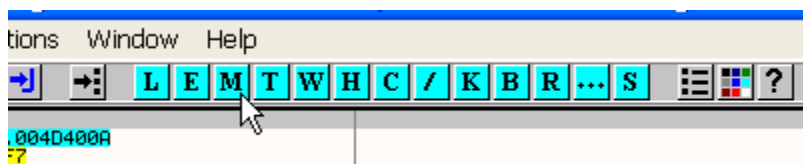


Pues si allí estamos en las primeras líneas del crackme, vemos como curiosidad que el crackme no esta ejecutando como los desempacados la primera sección después del header que comienza en 401000

Como la dirección del Entry Point es 4d4001



Vamos a ver que sección esta ejecutando, así que voy a VIEW-MEMORY o apreto el botón M.



003A0000	00004000				Priv	RW			
003B0000	00003000				Map	R			
00400000	00001000	SamBo		PE header	Imag	R		RWE	
00401000	00007C000	SamBo	.text	code	Imag	R		RWE	
0047D000	0000F000	SamBo	.data	data	Imag	R		RWE	
0048C000	00001000	SamBo	.tls		Imag	R		RWE	
0048D000	00001000	SamBo	.rdata		Imag	R		RWE	
0048E000	00003000	SamBo	.idata		Imag	R		RWE	
00491000	00003000	SamBo	.edata	exports	Imag	R		RWE	
00494000	000037000	SamBo	.rsrc	resources	Imag	R		RWE	
004CB000	00009000	SamBo	.reloc		Imag	R		RWE	
004D4000	00002000	SamBo	.aspack	SFX, imports	Imag	R		RWE	
004D6000	00001000	SamBo	.adata		Imag	R		RWE	
004E0000	00008000				Map	R	E	R	E
00500000	00000000				Map	R	E	R	E

Vemos que el programa se esta ejecutando en la sección que comienza en 4D4000 y su largo es 2000, pues 4d4001, es una dirección de memoria perteneciente a esa seccion.

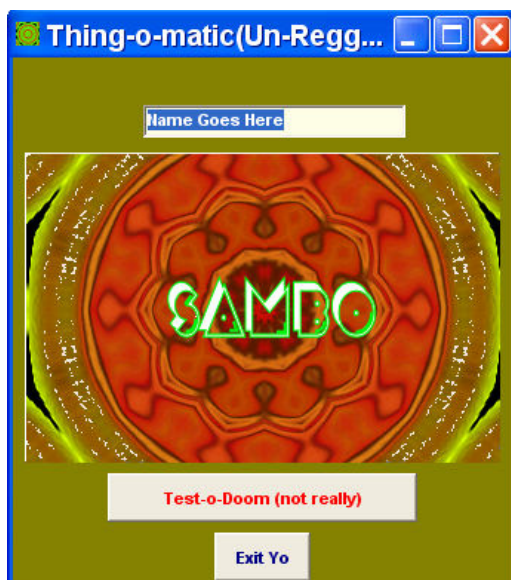
Pues eso era lo que avisaba OLLYDBG que el ENTRY POINT esta fuera de la sección code

003B0000	00003000	SamBo		PE header	Map	R		R	
00400000	00001000	SamBo		code	Imag	R		RWE	
00401000	00007C000	SamBo	.text		Imag	R		RWE	
0047D000	0000F000	SamBo	.data	data	Imag	R		RWE	
0048C000	00001000	SamBo	.tls		Imag	R		RWE	
0048D000	00001000	SamBo	.rdata		Imag	R		RWE	
0048E000	00003000	SamBo	.idata		Imag	R		RWE	
00491000	00003000	SamBo	.edata	exports	Imag	R		RWE	
00494000	000037000	SamBo	.rsrc	resources	Imag	R		RWE	
004CB000	00009000	SamBo	.reloc		Imag	R		RWE	
004D4000	00002000	SamBo	.aspack	SFX, imports	Imag	R		RWE	
004D6000	00001000	SamBo	.adata		Imag	R		RWE	
004E0000	00008000				Map	R	E	R	E
00500000	00000000				Map	R	E	R	E

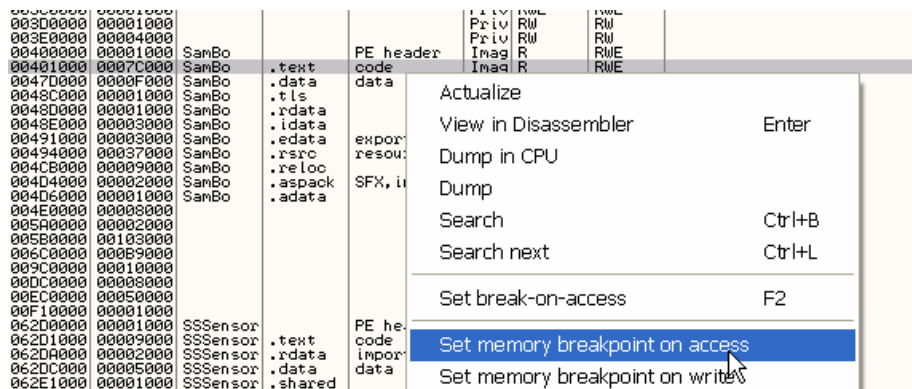
La sección CODE empieza en 401000, allí vemos resaltada la palabra CODE y nosotros estamos ejecutando la dirección 4D4001 que pertenece a otra sección, OLLYDBG nos aviso de esto que es una característica de muchos programas empackados.

La sección en la cual esta el ENTRY POINT corresponde al desempacador y una vez que se ejecuta y realiza su trabajo, salta a la sección code donde se ejecuta el programa en si.

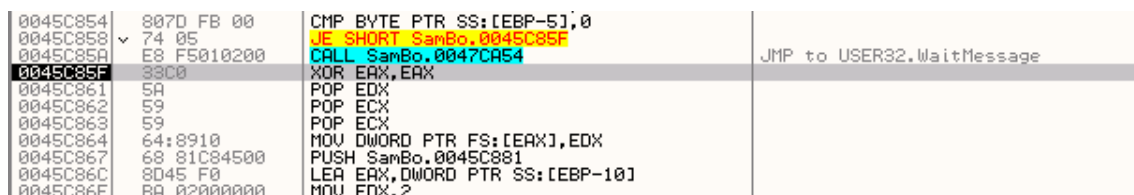
Pues entonces hagamos F9 o RUN



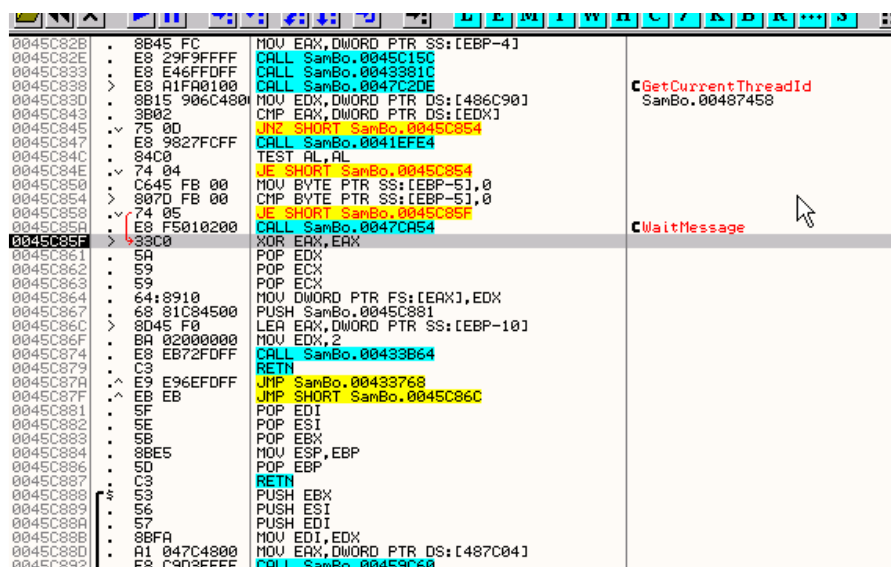
Al aparecer la ventana del crackme, para ingresar el serial, sabemos que el programa ya esta desempacado en memoria y esta ejecutándose ahora en la sección code, por lo cual podemos poner un BPM ON ACCESS en dicha sección, para que pare allí cuando ejecute alguna línea de la misma.



Al tratar de volver a la ventana del crackme, para OLLYDBG en la sección code o primera sección después del header.

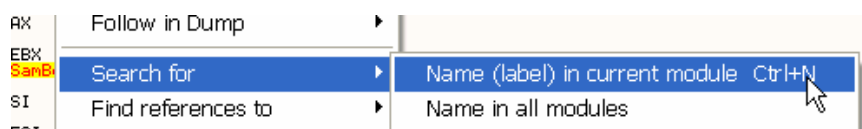


Veo que ahora que esta descomprimido el programa en memoria, puedo analizarlo entonces, en el listado hago click derecho



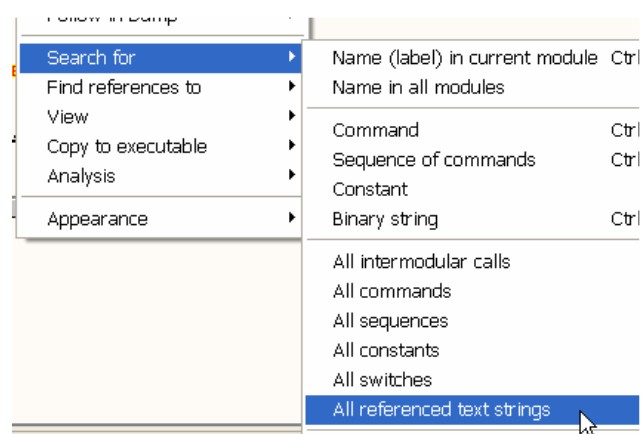
Vemos que OLLYDBG nos muestra el código analizado ahora perfectamente.

Ahora que estamos en la sección code, podemos ver que apis usa el programa, esto no lo podíamos hacer en el inicio, pues en ese momento solo nos mostraría las apis que usa el desempacador, y no las que usa el programa, ahora no hay problema.



Address	Section	Type	Name	Comment
004050BA	.aspack	Import	oleaut32.#35	
0040306C	.text	Export	@\$xp\$26Shdocvw_t lb@TCppWebBrowser	
00410960	.text	Export	@\$xp\$26Shdocvw_t lb@TCppWebBrowser	
00402274	.text	Export	@\$xp\$28Shdocvw_t lb@TCppShellWindows	
0040FC68	.text	Export	@\$xp\$28Shdocvw_t lb@TCppShellWindows	
00402180	.text	Export	@\$xp\$29Shdocvw_t lb@TCppShellUIHelper	
0040FBCC	.text	Export	@\$xp\$29Shdocvw_t lb@TCppShellUIHelper	
0040300C	.text	Export	@\$xp\$29Shdocvw_t lb@TCppWebBrowser_U1	
0041138C	.text	Export	@\$xp\$29Shdocvw_t lb@TCppWebBrowser_U1	
00401EE0	.text	Export	@\$xp\$32Shdocvw_t lb@TCppScriptErrorList	
0040F508	.text	Export	@\$xp\$32Shdocvw_t lb@TCppScriptErrorList	
00402A14	.text	Export	@\$xp\$32Shdocvw_t lb@TCppInternetExplorer	
00410360	.text	Export	@\$xp\$32Shdocvw_t lb@TCppInternetExplorer	
00401D98	.text	Export	@\$xp\$38Shdocvw_t lb@TCppSearchAssistantOC	
0040FE8E	.text	Export	@\$xp\$38Shdocvw_t lb@TCppSearchAssistantOC	
004023BC	.text	Export	@\$xp\$34Shdocvw_t lb@TCppShellBrowserWindow	
0040FD5C	.text	Export	@\$xp\$34Shdocvw_t lb@TCppShellBrowserWindow	
00401FD4	.text	Export	@\$xp\$36Shdocvw_t lb@TShellFavoritesNameSpace	
0040FA78	.text	Export	@\$xp\$36Shdocvw_t lb@TShellFavoritesNameSpace	
00411B94	.text	Export	@\$Shdocvw_ock@Finalize	
00411B7C	.text	Export	@\$Shdocvw_ock@Initialize	
00406790	.text	Export	@Unit1@Finalize	
00406770	.text	Export	@Unit1@Initialize	
004050AA	.aspack	Import	user32.ActivateKeyboardLayout	
004050A2	.aspack	Import	gdi32.BitBlt	
004050B2	.aspack	Import	ole32.CoCreateInstance	
0047D098	.data	Export	_CPPdebugHook	
00486E9C	.data	Export	_Form1	
004013C5	.text	Export	__GetExceptDLLInfo	
00404F60	.aspack	Import	kernel32.GetModuleHandleA	
00404F5C	.aspack	Import	kernel32.GetProcAddress	
0040509A	.aspack	Import	comctl32.ImageList_Add	
00404F64	.aspack	Import	kernel32.LoadLibraryA	
00404001	.aspack	Export	<ModuleEntryPoint>	
00405092	.aspack	Import	advapi32.RegCloseKey	
0040F37C	.text	Export	@Shdocvw_ock@Register\$qqrv	
0047D6F8	.data	Export	@Shdocvw_t lb@TCppScriptErrorList@	
00481E44	.data	Export	@Shdocvw_t lb@TCppScriptErrorList@	
00401F68	.text	Export	@Shdocvw_t lb@TCppScriptErrorList@\$bctr\$qqrp18Classes@TComponent	
0040EA08	.text	Export	@Shdocvw_t lb@TCppScriptErrorList@\$bctr\$qqrp18Classes@TComponent	
0040EA88	.text	Export	@Shdocvw_t lb@TCppScriptErrorList@BeforeDestruction\$qqrv	
0040E7D4	.text	Export	@Shdocvw_t lb@TCppScriptErrorList@Connect\$qqrv	
0040EACC	.text	Export	@Shdocvw_t lb@TCppScriptErrorList@ConnectTo\$qqrp91%TConInterface\$28Shd 1 argument	
0040E3D0	.text	Export	@Shdocvw_t lb@TCppScriptErrorList@Disconnect\$qqrv	
0040E5D0	.text	Export	@Shdocvw_t lb@TCppScriptErrorList@GetDefaultInterface\$qv	
0040E5F4	.text	Export	@Shdocvw_t lb@TCppScriptErrorList@GetDunk\$qqrv	
0040EBEC	.text	Export	@Shdocvw_t lb@TCppScriptErrorList@InitServerData\$qqrv	
0047D83C	.data	Export	@Shdocvw_t lb@TCppInternetExplorer@	
00482188	.data	Export	@Shdocvw_t lb@TCppInternetExplorer@	
00403000	.text	Export	@Shdocvw_t lb@TCppInternetExplorer@\$bctr\$qqrp18Classes@TComponent	
004108F0	.text	Export	@Shdocvw_t lb@TCppInternetExplorer@\$bctr\$qqrp18Classes@TComponent	
00408838	.text	Export	@Shdocvw_t lb@TCppInternetExplorer@BeforeDestruction\$qqrv	
004084EC	.text	Export	@Shdocvw_t lb@TCppInternetExplorer@Connect\$qqrv	
0040884C	.text	Export	@Shdocvw_t lb@TCppInternetExplorer@ConnectTo\$qqrp83%TConInterface\$24Shd 1 argument	
00408750	.text	Export	@Shdocvw_t lb@TCppInternetExplorer@Disconnect\$qqrv	
00408298	.text	Export	@Shdocvw_t lb@TCppInternetExplorer@GetDefaultInterface\$qv	
0040822C	.text	Export	@Shdocvw_t lb@TCppInternetExplorer@GetDunk\$qqrv	

Ughh que mal trago una terrible lista y ninguna conocida, veamos la lista de strings, aclaramos que esto tampoco se podía hacer en el inicio pues estábamos en otra sección y además las strings estaban empaçadas junto con el programa así que no veríamos nada.



En una terrible lista vemos

Address	Disassembly	Text string
0040101F	ASCII " o\$G",0	
00401043	ASCII "00 G",0	
00401049	ASCII "0LZG",0	
00401068	ASCII "pwG"	
0040106E	DD SamBo.00477CA4	ASCII "hLRH"
0040107A	DD SamBo.00478018	ASCII "hDRH"
0040161C	MOV EDX,SamBo.0047D0C1	ASCII "tSamBo!"
00401744	ASCII "Sysutils::Except"	
00401754	ASCII "ion",0	
004017F8	ASCII "Exception &",0	
00401834	ASCII "System::AnsiStri"	
00401844	ASCII "ng",0	
00401912	DD SamBo.00400000	ASCII "MZP"
00401934	ASCII "System::TObject",0	
0040195C	ASCII "Exception *",0	
00401A24	ASCII "TForm1 *",0	
00401A36	MOV ECX,SamBo.0047D1C5	ASCII "Thnx to Crackmes.de"
00401A48	MOV EDX,SamBo.0047D1AD	ASCII "SamBoS's First Crackme"
00401B08	MOV ECX,SamBo.0047D1FD	ASCII "You did it!"
00401B0D	MOV EDX,SamBo.0047D1D9	ASCII "I'd be proud, if it weren't so easy"
00401B1F	MOV EDX,SamBo.0047D209	ASCII "Thing-o-matic(Registered)"
00401B54	MOV ECX,SamBo.0047D233	ASCII "YUP"
00401B59	MOV EDX,SamBo.0047D223	ASCII "YAY YOU GOT IT!"
00401B6C	MOV ECX,SamBo.0047D253	ASCII "JUST JOKING!"
00401B71	MOV EDX,SamBo.0047D237	ASCII "nope, actually it was wrong"
00401B83	MOV EDX,SamBo.0047D260	ASCII "Thing-o-matic(Shareware)"
00401BFC	ASCII "TForm *",0	
00401C10	ASCII "AnsiString *",0	
00401C2E	DD SamBo.00400000	ASCII "MZP"
00401C50	ASCII "Forms::TForm",0	
00401CA8	ASCII "TForm1",0	
00401CCF	ASCII "TForm1",0	

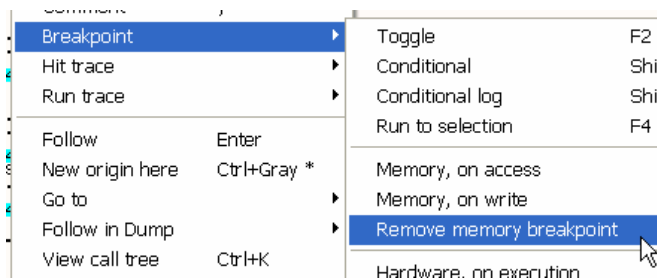
YOU DID IT, una de las strings del cartel de felicitación

30401AF7	E8 54A40700	CALL SamBo.0047BF50	CALL SamBo.0047BF50
30401AFC	59	POP ECX	
30401AFD	84C9	TEST CL,CL	
30401AFF	74 4C	JE SHORT SamBo.00401B40	
30401B01	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
30401B06	6A 40	PUSH 40	
30401B08	B9 FDD14700	MOV ECX,SamBo.0047D1FD	ASCII "You did it!"
30401B0D	BA D9D14700	MOV EDX,SamBo.0047D1D9	ASCII "I'd be proud, if it weren't so easy"
30401B12	8B00	MOV EAX,DWORD PTR DS:[EAX]	
30401B14	E8 6FA30700	CALL SamBo.0047BE88	
30401B19	66:C745 DC 1	MOV WORD PTR SS:[EBP-24],14	
30401B1F	BA 09D24700	MOV EDX,SamBo.0047D209	ASCII "Thing-o-matic(Registered)"
30401B24	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
30401B27	E8 6CA30700	CALL SamBo.0047BE98	
30401B2C	FF45 E8	INC DWORD PTR SS:[EBP-18]	
30401B2F	8B10	MOV EDX,DWORD PTR DS:[EAX]	
30401B31	A1 9C6E4800	MOV EAX,DWORD PTR DS:[_Form1]	
30401B36	E8 35180600	CALL SamBo.00463370	
30401B3B	FF4D E8	DEC DWORD PTR SS:[EBP-18]	
30401B3E	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
30401B41	BA 02000000	MOV EDX,2	
30401B46	E8 05A40700	CALL SamBo.0047BF50	
30401B4B	EB 62	JMP SHORT SamBo.00401BAF	
30401B4D	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
30401B52	6A 40	PUSH 40	
30401B54	B9 33D24700	MOV ECX,SamBo.0047D233	ASCII "YUP"
30401B59	BA 23D24700	MOV EDX,SamBo.0047D223	ASCII "YAY YOU GOT IT!"
30401B5E	8B00	MOV EAX,DWORD PTR DS:[EAX]	
30401B60	E8 23A30700	CALL SamBo.0047BE88	
30401B65	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
30401B6A	6A 10	PUSH 10	
30401B6C	B9 53D24700	MOV ECX,SamBo.0047D253	ASCII "JUST JOKING!"
30401B71	BA 37D24700	MOV EDX,SamBo.0047D237	ASCII "nope, actually it was wrong"
30401B76	8B00	MOV EAX,DWORD PTR DS:[EAX]	
30401B78	E8 0BA30700	CALL SamBo.0047BE88	
30401B7D	66:C745 DC 2	MOV WORD PTR SS:[EBP-24],20	
30401B83	BA 60D24700	MOV EDX,SamBo.0047D260	ASCII "Thing-o-matic(Shareware)"
30401B88	8D45 F0	LEA EAX,DWORD PTR SS:[EBP-10]	

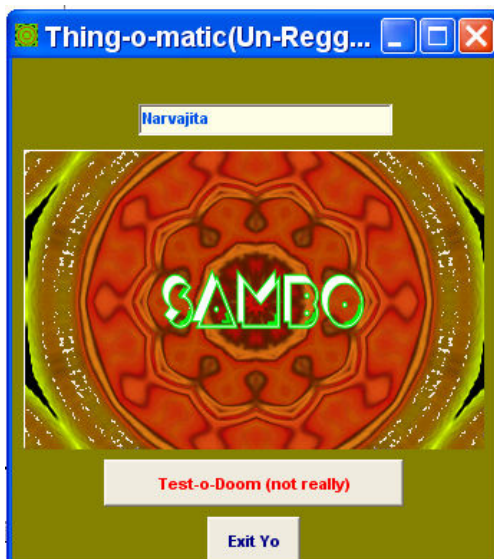
Bueno hay una comparación y un salto vemos strings de felicitación y de error aunque no hay MessageBoxA por aquí.

00401AF7	59	POP ECX	
00401AFC	84C9	TEST CL,CL	
00401AFF	74 4C	JE SHORT SamBo.00401B40	
00401B01	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
00401B06	6A 40	PUSH 40	
00401B08	B9 FDD14700	MOV ECX,SamBo.0047D1FD	ASCII "You did it!"
00401B0D	BA D9D14700	MOV EDX,SamBo.0047D1D9	ASCII "I'd be proud, if it weren't so easy"
00401B12	8B00	MOV EAX,DWORD PTR DS:[EAX]	
00401B14	E8 6FA30700	CALL SamBo.0047BE88	
00401B19	66:C745 DC 1	MOV WORD PTR SS:[EBP-24],14	
00401B1F	BA 09D24700	MOV EDX,SamBo.0047D209	ASCII "Thing-o-matic(Registered)"
00401B24	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
00401B27	E8 6CA30700	CALL SamBo.0047BE98	
00401B2C	FF45 E8	INC DWORD PTR SS:[EBP-18]	
00401B2F	8B10	MOV EDX,DWORD PTR DS:[EAX]	
00401B31	A1 9C6E4800	MOV EAX,DWORD PTR DS:[_Form1]	
00401B36	E8 35180600	CALL SamBo.00463370	
00401B3B	FF4D E8	DEC DWORD PTR SS:[EBP-18]	
00401B3E	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
00401B41	BA 02000000	MOV EDX,2	
00401B46	E8 05A40700	CALL SamBo.0047BF50	
00401B4B	EB 62	JMP SHORT SamBo.00401BAF	
00401B4D	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
00401B52	6A 40	PUSH 40	

Pongamos un BPX en el salto condicional para verificar si es el salto decisivo, quitemos el BPM ON ACCESS con click derecho

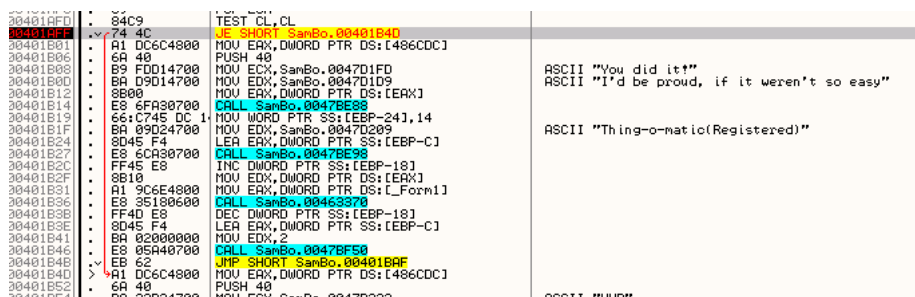


Y demos RUN

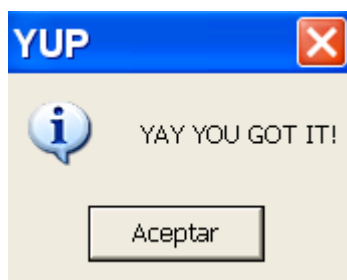


En la ventana del crackme tipeamos Narvajita o el serial falso que quieran.

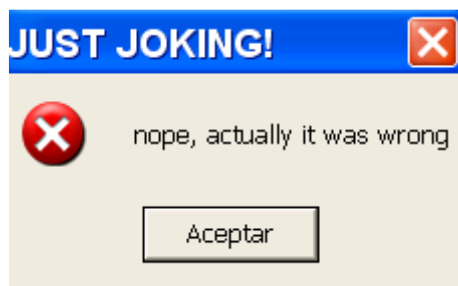
Apreto el botón Test-o-Doom



Vemos que salta apretamos F9



Pues sale el cartel de que lo lograste y si aceptamos



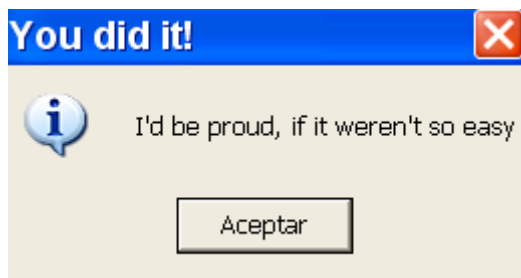
Te dice que era un chiste y que no acertaste nada jeje, demos RUN nuevamente lleguemos a la ventana y lleguemos de nuevo al salto apretando el botón Test-o-Doom.

00401AF7	E8 54A40700	CALL SanBo.0047BF50	SanBo.0047BF50
00401AFC	59	POP ECX	
00401AFD	84C9	TEST CL,CL	
00401AFF	74 4C	JE SHORT SanBo.00401B4D	
00401B01	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
00401B06	6A 40	PUSH 40	
00401B08	B9 FDD14700	MOV ECX,SanBo.0047D1FD	ASCII "You did it!"
00401B0D	BA D9D14700	MOV EDX,SanBo.0047D1D9	ASCII "I'd be proud, if it weren't so easy"
00401B12	8B00	MOV EAX,DWORD PTR DS:[EAX]	
00401B14	E8 6FA30700	CALL SanBo.0047BE88	
00401B19	66:C745 DC 1	MOV WORD PTR SS:[EBP-24],14	
00401B1F	BA 09D24700	MOV EDX,SanBo.0047D209	ASCII "Thing-o-matic(Registered)"
00401B24	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
00401B27	E8 6CA30700	CALL SanBo.0047BE98	
00401B2C	FF45 E8	INC DWORD PTR SS:[EBP-18]	
00401B2F	8B10	MOV EDX,DWORD PTR DS:[EAX]	
00401B31	A1 9C6E4800	MOV EAX,DWORD PTR DS:[_Form1]	
00401B36	E8 35180600	CALL SanBo.00463370	
00401B3B	FF4D E8	DEC DWORD PTR SS:[EBP-18]	
00401B3E	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
00401B41	BA 02000000	MOV EDI,2	
00401B46	E8 05A40700	CALL SanBo.0047BF50	
00401B48	EB 62	JMP SHORT SanBo.00401BAF	
00401B4D	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
00401B52	6A 40	PUSH 40	
00401B54	B9 33D24700	MOV ECX,SanBo.0047D233	ASCII "YUP"
00401B5C	BA 09D24700	MOV EDX,SanBo.0047D209	ASCII "YOU GOT IT!"

Invirtamos el salto a ver si sale el cartel de felicitaciones, para ver si es esto verdaderamente lo que decide.

00401AFD	84C9	TEST CL,CL	
00401AFF	74 4C	JE SHORT SanBo.00401B4D	
00401B01	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
00401B06	6A 40	PUSH 40	
00401B08	B9 FDD14700	MOV ECX,SanBo.0047D1FD	ASCII "You did it!"
00401B0D	BA D9D14700	MOV EDX,SanBo.0047D1D9	ASCII "I'd be proud, if it weren't so easy"
00401B12	8B00	MOV EAX,DWORD PTR DS:[EAX]	
00401B14	E8 6FA30700	CALL SanBo.0047BE88	
00401B19	66:C745 DC 1	MOV WORD PTR SS:[EBP-24],14	
00401B1F	BA 09D24700	MOV EDX,SanBo.0047D209	ASCII "Thing-o-matic(Registered)"
00401B24	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
00401B27	E8 6CA30700	CALL SanBo.0047BE98	
00401B2C	FF45 E8	INC DWORD PTR SS:[EBP-18]	
00401B2F	8B10	MOV EDX,DWORD PTR DS:[EAX]	
00401B31	A1 9C6E4800	MOV EAX,DWORD PTR DS:[_Form1]	
00401B36	E8 35180600	CALL SanBo.00463370	
00401B3B	FF4D E8	DEC DWORD PTR SS:[EBP-18]	
00401B3E	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
00401B41	BA 02000000	MOV EDI,2	
00401B46	E8 05A40700	CALL SanBo.0047BF50	
00401B48	EB 62	JMP SHORT SanBo.00401BAF	
00401B4D	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	
00401B5C	6A 40	PUSH 40	

Al hacer doble click en el flag Z no saltara doy RUN



Y sale el cartel de felicitación, así que por aquí es donde decide las cosas, veamos la comparación antes del salto.

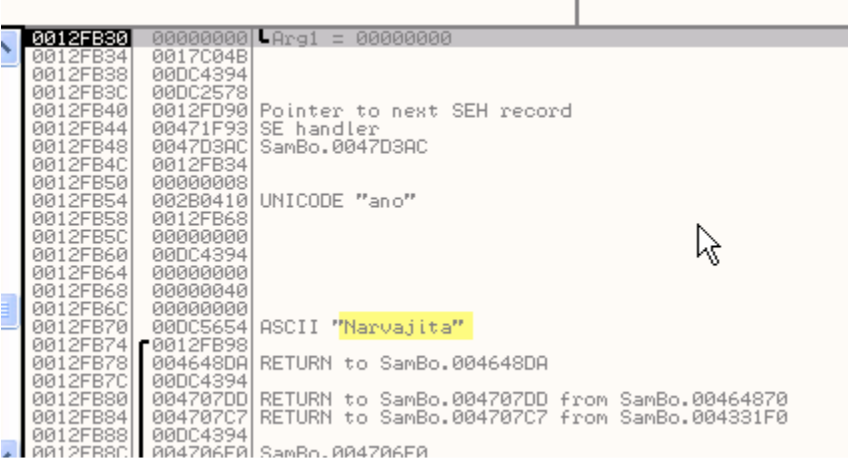
00401AFD	59	POP ECX	
00401AFF	84C9	TEST CL,CL	
00401B01	74 4C	JE SHORT SanBo.00401B4D	
00401B06	A1 DC6C4800	MOV EAX,DWORD PTR DS:[486CDC]	

Es un TEST CL,CL que solo testea si CL es cero o uno, así que la comparación se realiza antes, posiblemente en el CALL que esta justo antes pongamos un BPX allí.

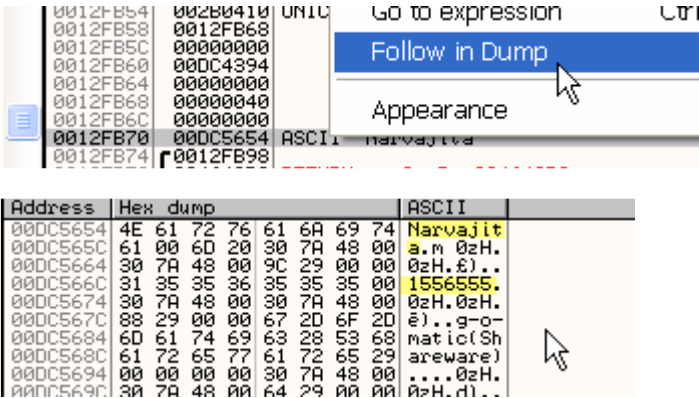
```
00401AF2 | . BA 02000000 MOV EDX,2
00401AF7 | . E8 54A40700 CALL SamBo.0047BF50
00401AFC | . 59 POP ECX
00401AFD | . 84C9 TEST CL,CL
00401AFF | . 74 4C JE SHORT SamBo.00401B4D
00401B01 | . A1 DC6C4800 MOV EAX,DWORD PTR DS:[486CDC]
```

Demos RUN y volvamos a la ventana y apretemos nuevamente el botón Test para que pare ahora en el CALL.

Una vez que paro como soy curioso miro un poco alrededor jeje



Si miro en el stack veo mi serial falso allí, veamos en el DUMP haciendo FOLLOW IN DUMP en el mismo.



Mi ojo de cracker ve por ahí una string con forma de posible serial correcto, podría probar a ver si es pero por ahora seguiré con el método de buscarlo a fondo sin intentar.

Si alguna vez comparara o operara con mi serial falso dentro del CALL, pues poniendo un BPX ON ACCESS en el mismo, parara cuando el programa quiera hacer algo con el, o sea al acceder al mismo para cualquiera operación o comparación que desee realizar.

Address	Hex dump	ASCII
00DC5654	4E 61 72 76 61 6A 69 74	Narvajit
00DC565C	61 00 6D 20 30 7A 48 00	a.m 0zH.
00DC5664	30 7A 48 00 9C 29 00 00	0zH.é)..
00DC566C	31 35 35 36 35 35 35 00	1556555.
00DC5674	30 7A 48 00 30 7A 48 00	0zH.0zH.
00DC567C	88 29 00 00 67 2D 6F 2D	é)..g-o-
00DC5684	6D 61 74 69 63 28 53 68	matic(Sh
00DC568C	61 72 65 77 61 72 65 29	areware)
00DC5694	00 00 00 00 30 7A 48 00	...0zH.
00DC569C	30 7A 48 00 64 29 00 00	0zH.dl..

Así que marco mi serial falso y hago click derecho

address	Hex dump	ASCII	
00C5654	4E 61 72 76 61 6A 69 74	Narvajit	
00C565C	61 00 6D 20 30 7A 48 00	a.m	
00C5664	30 7A 48 00 9C 29 00 00	0zH	Backup
00C566C	31 35 35 36 35 35 35 00	155	Copy
00C5674	30 7A 48 00 30 7A 48 00	0zH	Binary
00C567C	88 29 00 00 67 2D 6F 2D	e).	
00C5684	6D 61 74 69 63 28 53 68	mat	
00C568C	61 72 65 77 61 72 65 29	are	
00C5694	00 00 00 00 30 7A 48 00	...	Breakpoint
00C569C	30 7A 48 00 64 29 00 00	0zH	Memory, on access
00C56A4	15 4F 41 00 00 00 00 00	30F	Search for
00C56AC	00 00 00 00 00 00 00 00	...	Memory, on write

Con lo cual si doy RUN y dentro del call accede a mi serial falso, parara OLLYDBG.

Apreto F9

Vemos que no para, quizás la comparación sea antes, así que o bien podemos ir poniendo BPX en los calles que están mas arriba de este y repetir el método o bien empleamos el método de parar justo cuando el programa ingresa el serial, aquí no utiliza la api GetWindowTextA pero tenemos nuestros Mensajes de Windows los cuales seguro nos servirán.

Quito el BPM ON ACCESS

Comment	;	
Breakpoint		Toggle F2
Hit trace		Conditional SI
Run trace		Conditional log SI
Go to		Memory, on access
Follow in Dump		Memory, on write
View call tree	Ctrl+K	Remove memory breakpoint

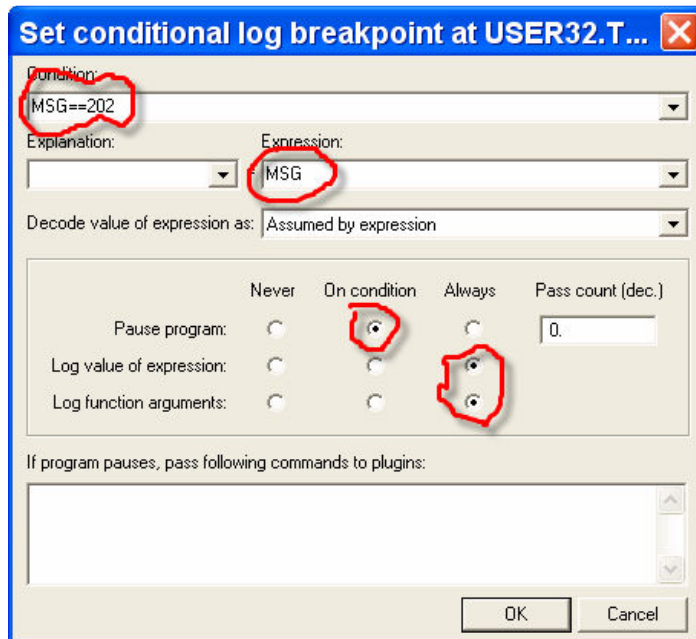
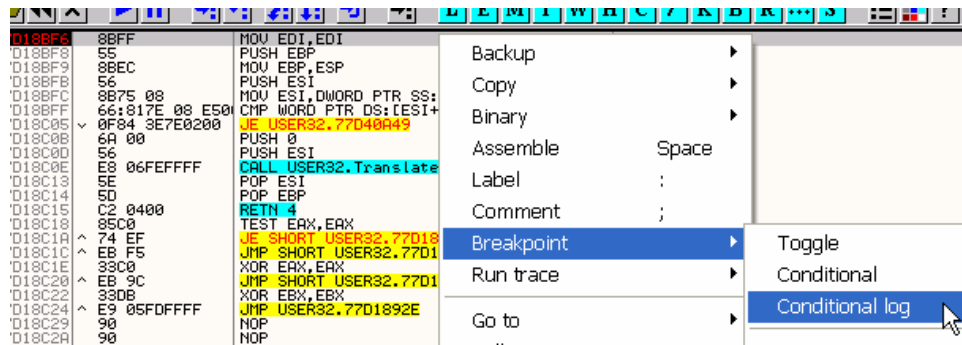
Así que para hacer rápido pongo un BP TranslateMessage

000C572C	00 00 00 00 00 00 00 00
000C5734	00 00 00 00 41 01 00 00	...
Command	BP TranslateMessage	

Y doy RUN

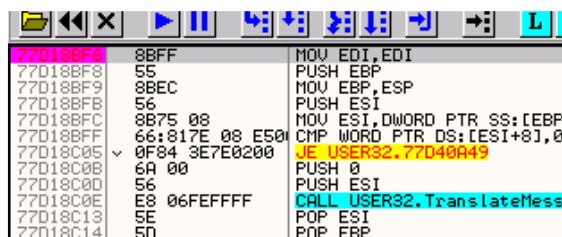
77D18BF6	8BFF	MOV EDI,EDI
77D18BF8	55	PUSH EBP
77D18BF9	8BEC	MOV EBP,ESP
77D18BF8	56	PUSH ESI
77D18BFC	8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]
77D18BFF	66:817E 08 E50	CMP WORD PTR DS:[ESI+8],0E5
77D18C05	0F84 3E7E0200	JE USER32.77D40A49
77D18C0B	6A 00	PUSH 0
77D18C0D	56	PUSH ESI
77D18C0E	E8 06FEFFFF	CALL USER32.TranslateMessageEx

Allí para en la api, así que hago click derecho



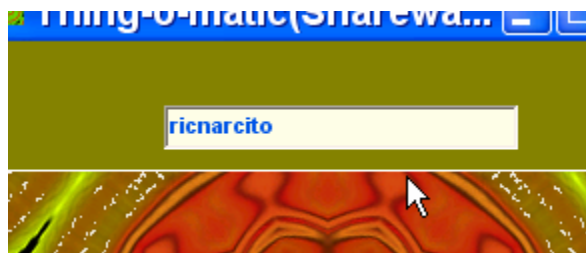
Y en la ventana tipeamos la condición MSG==202 que era el valor de WM_LBUTTONDOWN

Y ponemos las tildes para que pare ON CONDITION y que loguee siempre todo.



Allí esta en rosado el BPX CONDICIONAL doy RUN

Llego a la ventana del crackme y para evitar confusiones cambio el serial por si quedo el anterior en la memoria.

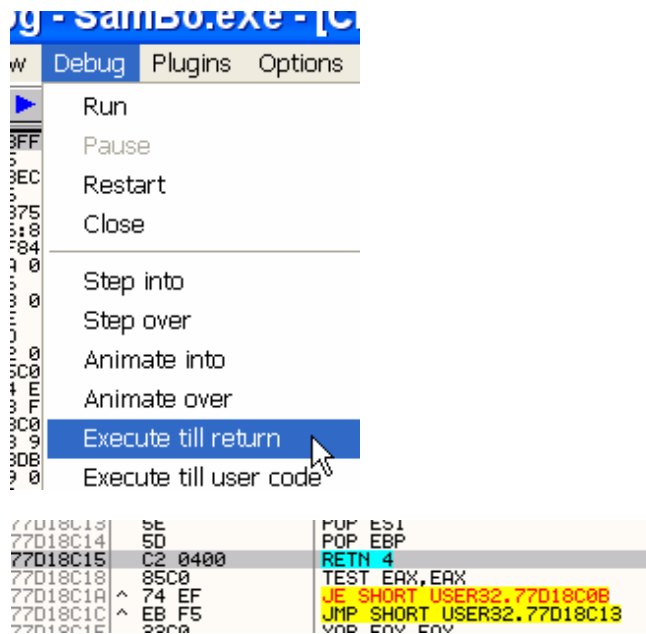


y al apretar el botón para en mi BPX CONDICIONAL

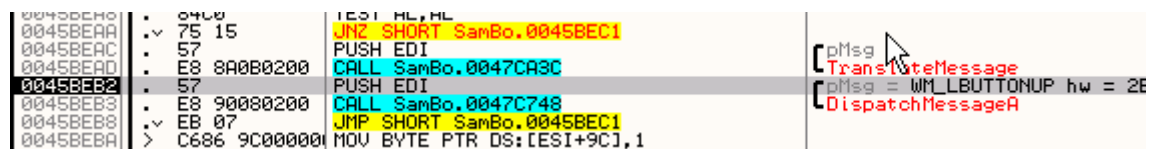


Allí esta paro en WM_LBUTTONDOWN

Volvamos al programa con EXECUTE TILL RETURN

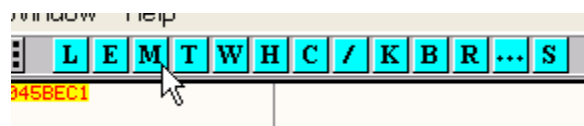


Apreto F7 y vuelvo al programa

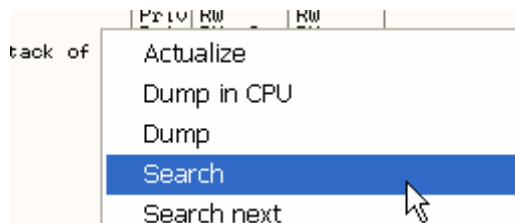


Aquí se me abren dos posibilidades poner un BPM ON ACCESS en la primera sección para con f9 ir saltando y ver cuando llega a la rutina de comparación del serial, lo cual me parece un poco complejo porque aquí ya vi que la rutina del serial es bastante larga y me pasare saltando jeje.

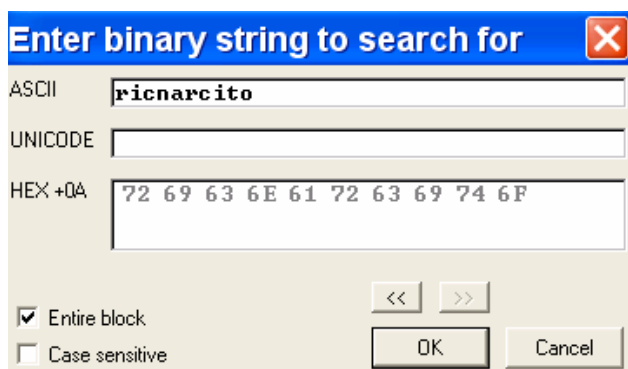
La otra posibilidad es ver si el serial que tipee ya ingreso en la memoria, como es un serial nuevo pues si esta en la memoria, no es porque quedo de antes si no que ya ingreso, así que voy a la ventana M.



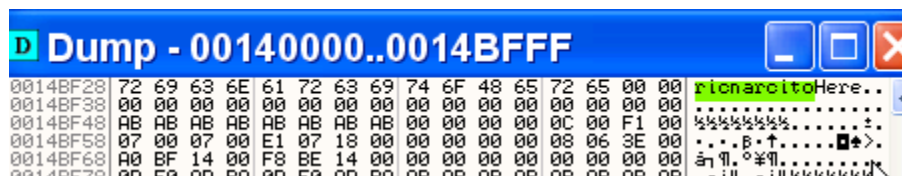
Que además de permitirme ver las secciones me permite buscar una cadena en toda la memoria, así que hago click derecho SEARCH



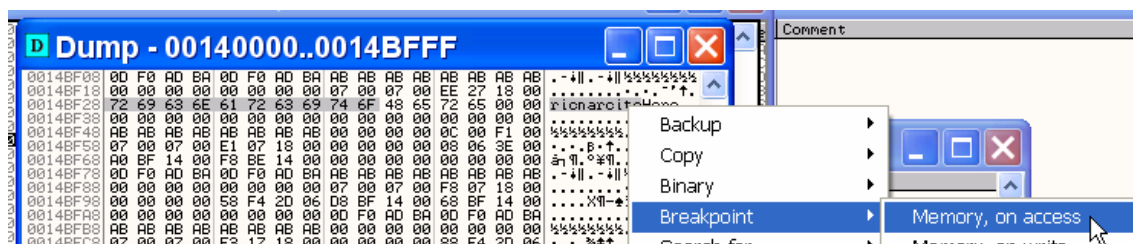
Y en la ventana que aparece, en la parte ASCII tipeo mi serial falso.



Apreto OK para que busque en toda la memoria.

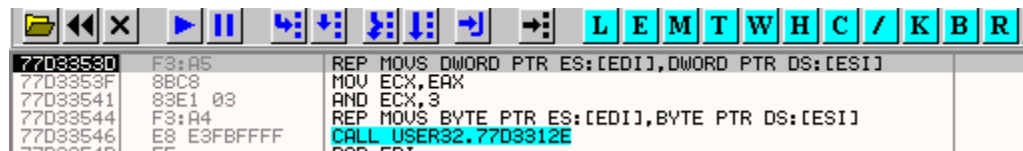


Pues la primera ocurrencia es aquí puedo repetir con CTRL+L y no hallara mas ocurrencias, ni en la sección que se abrió ni en el resto de la memoria.

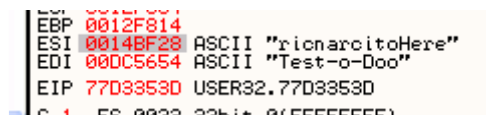


Pues pongo un BPM ON ACCESS en mi serial falso pues en algún momento accederá a el para leer, operar o comparar.

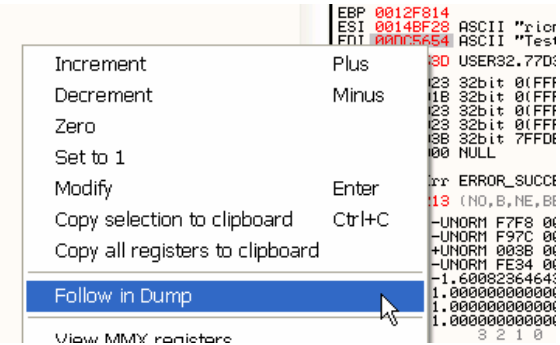
Y apreto F9



Para aquí donde lo copiara a su ubicación la que habíamos visto en el ultimo CALL



Sabemos que el REP MOVSB copiara el contenido de ESI a EDI, así que veamos EDI en el DUMP, haciendo click derecho en EDI-FOLLOW IN DUMP.



Address	Hex dump	ASCII
00DC5654	54 65 73 74 2D 6F 2D 44	Test-o-D
00DC565C	6F 6F 00 00 30 7A 48 00	o..0zH.
00DC5664	30 7A 48 00 9C 29 00 00	0zH.é)..
00DC566C	00 00 73 74 30 7A 48 00	..st0zH.

Allí lo copiara el REP MOVSB apretemos F8 para que lo haga

Address	Hex dump	ASCII
00DC5654	72 69 63 6E 61 72 63 69	ricnarcio
00DC565C	74 6F 00 00 30 7A 48 00	to..0zH.
00DC5664	30 7A 48 00 9C 29 00 00	0zH.é)..
00DC566C	00 00 73 74 30 7A 48 00	..st0zH.

Al llegar con F8 al RET lo habrá copiado entero, y pongo ahora el BPM ON ACCESS en mi serial falso aquí

Address	Hex dump	ASCII
00DC5654	72 69 63 6E 61 72 63 69	ricnarcio
00DC565C	74 6F 00 00 30 7A 48 00	to..0zH.
00DC5664	30 7A 48 00 9C 29 00 00	0zH.é)..
00DC566C	00 00 73 74 30 7A 48 00	..st0zH.

Backup	▶
Copy	▶
Binary	▶
Breakpoint	▶ Memory, on access
Search for	▶ Memory, on write

Al dar RUN para aquí

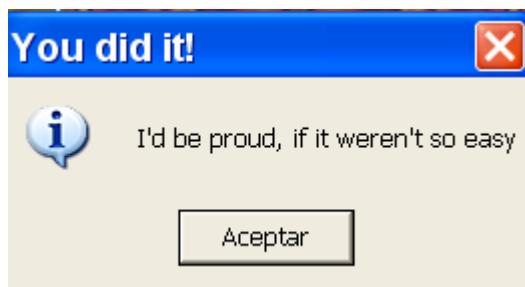
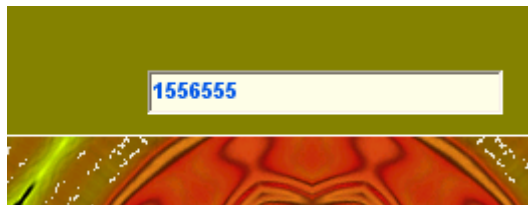
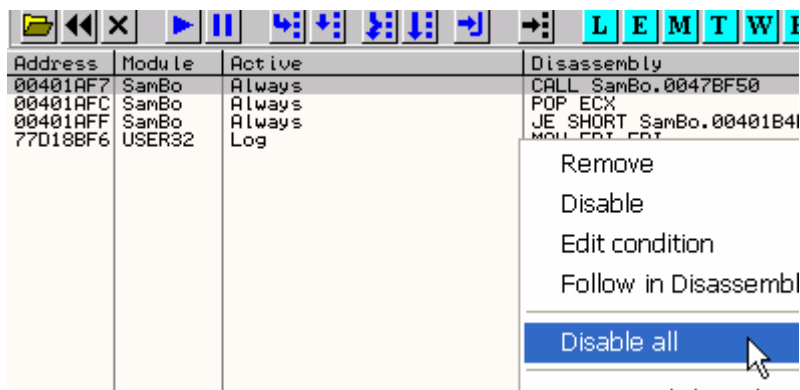
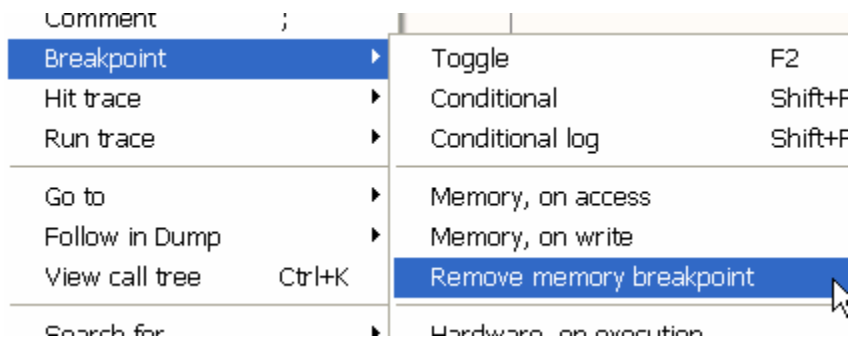
00433F65	> 8B0E	MOV ECX,DWORD PTR DS:[ESI]
00433F67	. 8B1F	MOV EBX,DWORD PTR DS:[EDI]
00433F69	. 39D9	CMP ECX,EBX
00433F6B	75 58	JNZ SHORT SamBo.00433FC5
00433F6D	. 4A	DEC EDX
00433F6E	74 15	JE SHORT SamBo.00433F85
00433F70	0B4E 04	MOV ECX,DWORD PTR DS:[ESI+4]

Justo en la comparación jeje

Registers (FPU)	
EAX	00000003
ECX	0012F56C ASCII "1556555"
EDX	00000001
EBX	00DC4394
ESP	0012FB14
EBP	0012FB2C
ESI	00DC5654 ASCII "ricnarcito"
EDI	00DC566C ASCII "1556555"
EIP	00433F65 SamBo.00433F65

ESI apunta a mi serial falso y EDI apunta al correcto que es 1556555

Si quitamos todos los BPX



Pues allí esta el cartel de felicitación

Ahora para intentar pues no hemos visto aun el tema, por lo cual la siguiente lección no estará con password, si no para intentar y divertirse, el que quiere intentar el crackme de cruehead hallar un serial para su nombre pues puede hacerlo, o sea en user pone su nombre y halla el serial correcto, en la próxima parte empezaremos con crackmes USER y SERIAL y el primero que resolveremos será ese, así que a no frustrarse si no pueden, que es un tema no visto aun, así que tómenlo como diversión si pueden con el , pues felicitaciones, si no pueden, lo aprenden en la próxima parte.

Hasta la parte 16

Ricardo Narvaja

08 de diciembre de 2005