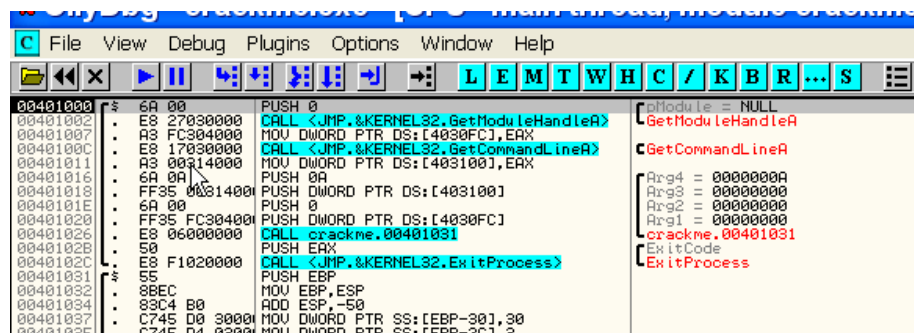


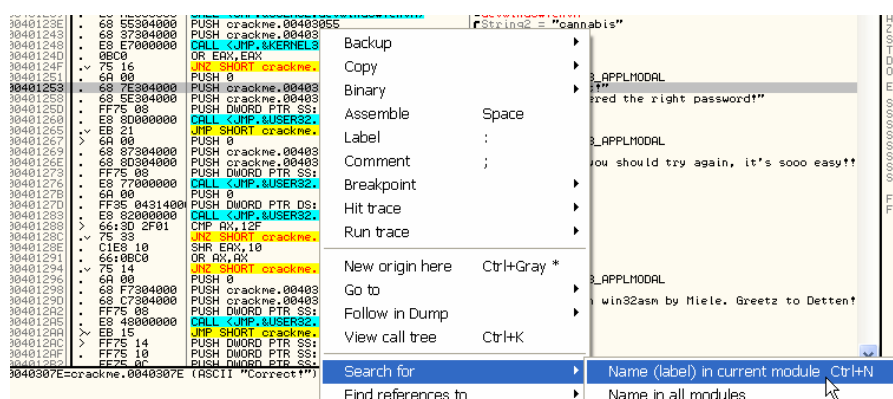
## INTRODUCCIÓN AL CRACKING CON OLLYDBG PARTE 14

Bueno antes que nada vamos a explicar como se soluciona el crackme que deje como tarea en la parte 13.



```
00401000 6A 00 PUSH 0
00401002 E8 27030000 CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 A3 FC040000 MOV DWORD PTR DS:[4030FC],EAX
0040100C E8 17030000 CALL <JMP.&KERNEL32.GetCommandLineA>
00401011 A3 00314000 MOV DWORD PTR DS:[403100],EAX
00401016 6A 00 PUSH 0
00401018 FF35 00314000 PUSH DWORD PTR DS:[403100]
0040101E 6A 00 PUSH 0
00401020 FF35 FC040000 PUSH DWORD PTR DS:[4030FC]
00401026 E8 06000000 CALL crackme.00401031
0040102B 50 PUSH EAX
0040102C F1020000 CALL <JMP.&KERNEL32.ExitProcess>
00401031 55 PUSH ESP
00401032 8BEC MOV EBP,ESP
00401034 83C4 B0 ADD ESP,-50
00401037 C745 D0 3000 MOV DWORD PTR SS:[EBP-30],30
0040103F C745 D4 0300 MOV DWORD PTR SS:[EBP-2C],3
```

Allí esta el mielecrackme abierto en OLLYDBG, y detenido en el ENTRY POINT, veamos las apis que utiliza con SEARCH FOR-NAME (LABEL) IN CURRENT MODULE.

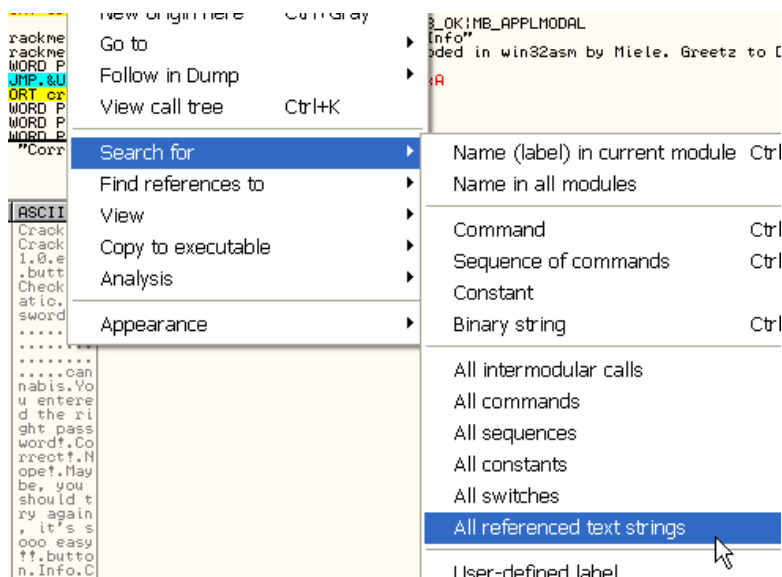


Aquí están las apis utilizadas

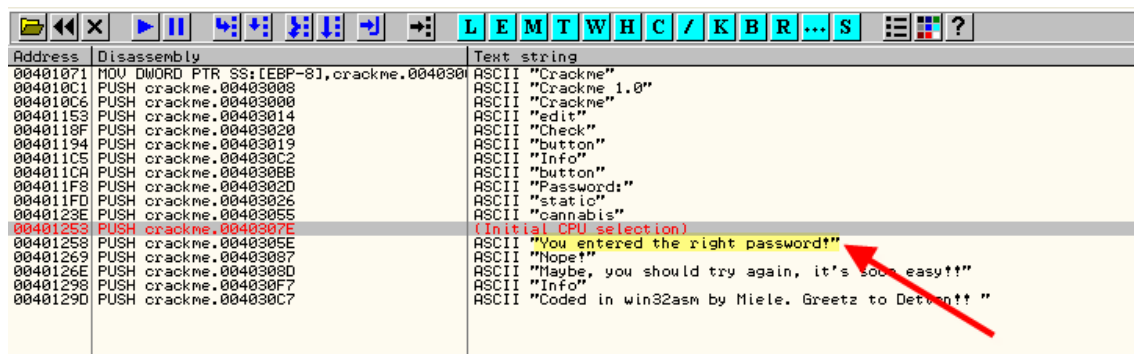
Address	Section	Type	Name	Comment
00402048	.rdata	Import	USER32.CreateWindowExA	
00402028	.rdata	Import	USER32.DefWindowProcA	
00402040	.rdata	Import	USER32.DispatchMessageA	
00402008	.rdata	Import	KERNEL32.ExitProcess	
0040200C	.rdata	Import	KERNEL32.GetModuleHandleA	
0040203C	.rdata	Import	USER32.GetMessageA	
00402004	.rdata	Import	KERNEL32.GetModuleHandleA	
00402024	.rdata	Import	USER32.GetWindowTextA	
00402020	.rdata	Import	USER32.LoadCursorA	
00402014	.rdata	Import	USER32.LoadIconA	
00402000	.rdata	Import	KERNEL32.lstrcmpA	
00402018	.rdata	Import	USER32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	
0040201C	.rdata	Import	USER32.PostQuitMessage	
00402044	.rdata	Import	USER32.RegisterClassExA	
0040204C	.rdata	Import	USER32.SetFocus	
0040202C	.rdata	Import	USER32.SetWindowTextA	
00402030	.rdata	Import	USER32.ShowWindow	
00402034	.rdata	Import	USER32.TranslateMessage	
00402038	.rdata	Import	USER32.UpdateWindow	

Allí están las apis que son importantes, GetWindowTextA para ingresar el serial que tipeamos, lstrcmpA como les anticipe en la parte anterior para comparar strings y MessageBoxA para mostrar el mensaje de si colocamos el serial correcto o no.

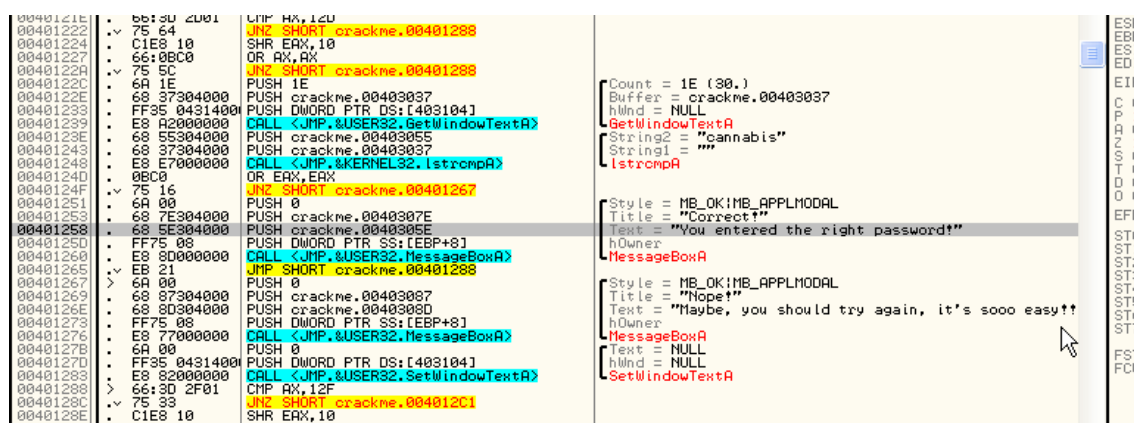
Podemos poner un BPX en esas apis, para parar cuando ingresa nuestro serial falso, pero en este caso que es bien sencillo, podemos hacer mas rápido si miramos las STRINGS que utiliza el programa.



Haciendo SEARCH FOR – ALL REFERENCED TEXT STRINGS sale la lista de STRINGS o CADENAS DE TEXTO usadas por el programa veamos.



Allí vemos las strings de que acertamos y las que muestra cuando fallamos, si hacemos doble click en alguna de ellas, nos llevara a la zona de los MessageBoxA, probemos haciendo doble click en YOU ENTERED THE RIGHT PASSWORD (has ingresado el password correcto)



Allí vemos la zona caliente.

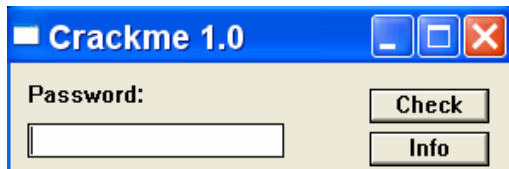
Primero GetWindowTextA como dijimos para ingresar el serial que tipeamos, luego lstrcmpA para comparar con el serial correcto, y luego según si son iguales continua al MessageBoxA, con la leyenda YOU ENTERED THE RIGHT PASSWORD, y si no son iguales salta al otro MessageBoxA, MAYBE, YOU SHOULD TRY AGAIN, IT'S SO EASY, que intentemos de nuevo que es fácil.

Así que pondremos un BPX, allí en el CALL a la api lstrcmpA, para ver que compara.

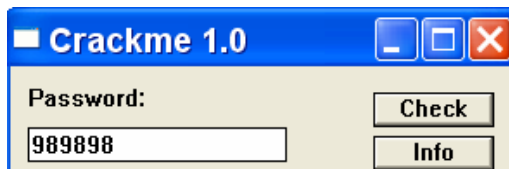
00401224	. C1E8 10	SHR EAX,10	
00401227	. 66:0BC0	OR AX,AX	
0040122A	. 75 5C	JNZ SHORT crackme.00401288	
0040122C	. 6A 1E	PUSH 1E	
0040122E	. 68 37304000	PUSH crackme.00403037	
00401233	. FF35 04310400	PUSH DWORD PTR DS:[403104]	
00401239	. E8 A2000000	CALL <JMP.&USER32.GetWindowTextA>	
0040123E	. 68 55304000	PUSH crackme.00403055	
00401243	. 68 37304000	PUSH crackme.00403037	
00401248	. E8 E7000000	CALL <JMP.&KERNEL32.lstrcmpA>	
0040124D	. 0BC0	OR EAX,EAX	
0040124F	. 75 16	JNZ SHORT crackme.00401267	
00401251	. 6A 00	PUSH 0	

Count = 1E (30.)  
Buffer = crackme.00403037  
hwnd = NULL  
GetWindowTextA  
String2 = "cannabis"  
String1 = ""  
lstrcmpA  
Style = MB\_OK|MB\_APPLMODAL

Ahora si corro el programa con F9



Sale la ventana para ingresar el serial y ponemos uno cualquiera por ejemplo 989898.



Al apretar CHECK para en el BPX que colocamos.

00401239	. E8 A2000000	CALL <JMP.&USER32.GetWindowTextA>	
0040123E	. 68 55304000	PUSH crackme.00403055	
00401243	. 68 37304000	PUSH crackme.00403037	
00401248	. E8 E7000000	CALL <JMP.&KERNEL32.lstrcmpA>	
0040124D	. 0BC0	OR EAX,EAX	

GetWindowTextA  
String2 = "cannabis"  
String1 = "989898"  
lstrcmpA

Vemos que OLLY nos aclara los parámetros, los cuales son las dos STRINGS que comparara, en este caso vemos que compara la que tipee "989898", con la palabra "cannabis".

Al apretar F8 para ejecutar el CALL de la api

Registers (FPU)	
EAX	FFFFFFFF
ECX	00005771
EDX	0000000C
EBX	00000000
ESP	0012FC84
EBP	0012FC84
ESI	00401115 cra
EDI	0012FCEC
EIP	0040124D cra
C 0	ES 0023 32b
P 1	CS 001B 32b
A 0	SS 0023 32b
Z 0	DS 0023 32b
S 1	FS 003B 32b
T 0	SS 0000 NULL
D 0	
O 0	lastErr ERROR
EFL	00000286 (NO,N
ST0	empty -??? FFF
ST1	empty -??? FFF

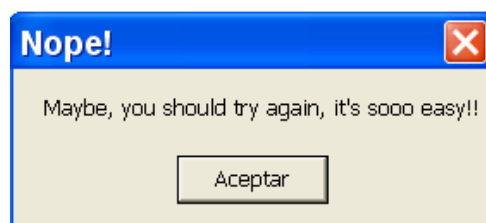
El resultado dicha api lo muestra en EAX, si es FFFFFFFF o sea -1, es que las strings no son iguales.

EIP	0040124F crack
C 0	ES 0023 32bit
P 1	CS 001B 32bit
A 0	SS 0023 32bit
Z 0	DS 0023 32bit
S 1	FS 003B 32bit
T 0	SS 0000 NULL
D 0	
O 0	lastErr ERROR
EFL	00000286 (NO,N
ST0	empty -??? FFF
ST1	empty -??? FFF

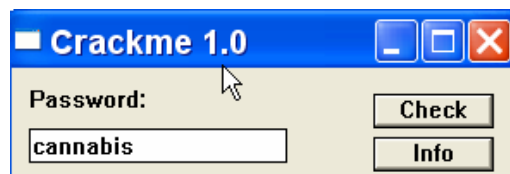
Como el resultado de la comparación no es cero, no se activa el FLAG Z, y el JNZ salta ya que el FLAG Z es cero. (Recordar que JNZ salta si el FLAG Z es cero porque es el inverso de JZ que salta cuando el FLAG Z esta activo o 1)

00401243	68 37304000	PUSH crackme.00403037	String1 = "cannabis"
00401244	E8 E7000000	CALL <JMP.&KERNEL32.lstrcmpA>	String1 = "989898"
00401245	0BC0	OR EAX,EAX	
0040124F	75 16	JNZ SHORT crackme.00401267	
00401251	6A 00	PUSH 0	Style = MB_OK MB_APPLMODAL
00401253	68 7E304000	PUSH crackme.0040307E	Title = "Correct!"
00401258	68 5E304000	PUSH crackme.0040305E	Text = "You entered the right passwo
0040125D	FF 75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401260	E8 80000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
00401265	EB 21	JMP SHORT crackme.00401288	
00401267	6A 00	PUSH 0	Style = MB_OK MB_APPLMODAL
00401269	68 87304000	PUSH crackme.00403087	Title = "Nope!"
0040126E	68 8D304000	PUSH crackme.0040308D	Text = "Maybe, you should try again,
00401273	FF 75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401276	E8 77000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
0040127B	6A 00	PUSH 0	Style = MB_OK MB_APPLMODAL
0040127D	FF 35 04314000	PUSH DWORD PTR DS:[403104]	hWnd = 02240882 (class='Edit',parent
00401283	E8 82000000	CALL <JMP.&USER32.SetWindowTextA>	SetWindowTextA
00401288	66 3D 2F01	CMPSB	

Pues allí salta y va al cartel de error, así que ya sabemos que compara con la palabra cannabis, o sea que este es el serial correcto doy RUN nuevamente, acepto el cartel malo.



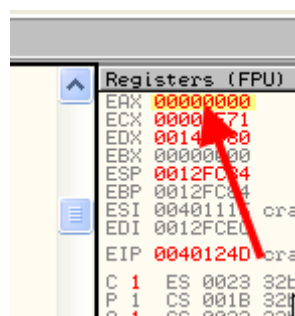
Y vuelvo a la ventana para ingresar el serial esta vez tipeo el serial correcto, cannabis.



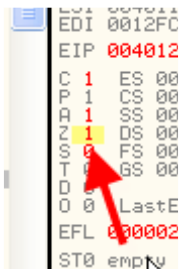
Apreto el botón CHECK y para en el BPX.

00401239	E8 A2000000	CALL <JMP.&USER32.GetWindowTextA>	GetWindowTextA
0040123E	68 55304000	PUSH crackme.00403055	String2 = "cannabis"
00401243	68 37304000	PUSH crackme.00403037	String1 = "cannabis"
00401248	E8 E7000000	CALL <JMP.&KERNEL32.lstrcmpA>	lstrcmpA
0040124D	0BC0	OR EAX,EAX	
0040124F	75 16	JNZ SHORT crackme.00401267	

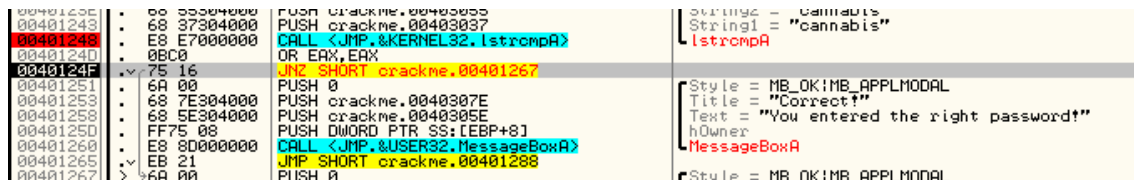
Veo que va a comparar ambas strings y estas son iguales, apreto F8 para ejecutar el CALL de la api.



Ahora al ser ambas strings iguales el resultado de la api que guarda en EAX es cero, por lo cual se activa el FLAG Z.



Y el JNZ en este caso no salta al estar el FLAG Z activo.



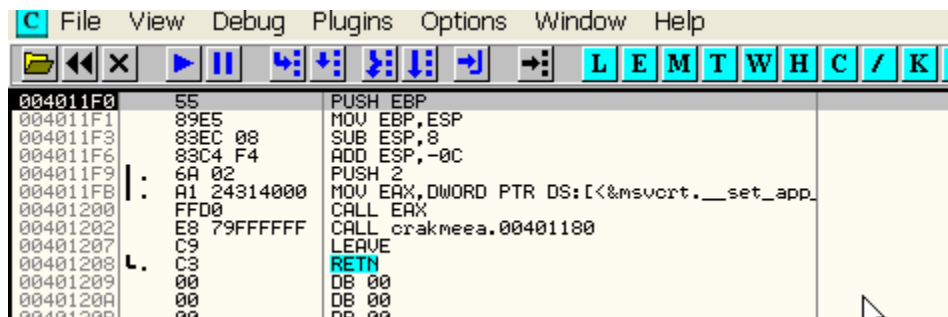
Allí no va a saltar y va a ir al cartel de que acertamos demos RUN.



Pues esta es la solución del crackme que deje como tarea de la parte 13, jeje, el serial es la palabra "cannabis"

Sigamos ahora avanzando con HARDCODED SERIALS mas difíciles nos quedan para ver dos.

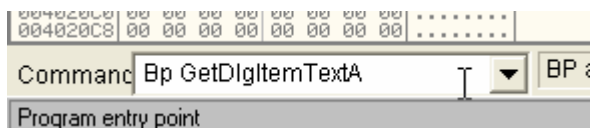
El caso siguiente es un paso mas, en este ya no compara directamente el serial que tipeamos sino que realiza algunas operaciones antes de comparar veamos el ejemplo, este es el crackmeeasy, abrámoslo en OLLYDBG.



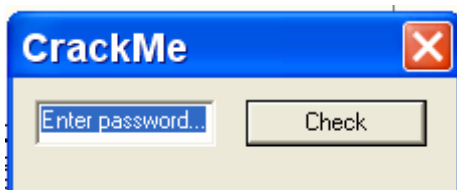
Ya sabemos como ver las apis que utiliza, así que en la lista vemos GetDlgItemTextA, pongámosle un BPX en dicha api.

Address	Section	Type	Name	Comment
00403110	.idata	Import	msvcrt.atexit	
004030F4	.idata	Import	msvcrt.cexit	
00403130	.idata	Import	USER32.DialogBoxParamA	
00403134	.idata	Import	USER32.EndDialog	
004030D4	.idata	Import	KERNEL32.ExitProcess	
004030F8	.idata	Import	msvcrt._fileno	
004030FC	.idata	Import	msvcrt._fnmode	
00403100	.idata	Import	msvcrt._fpreset	
004030D8	.idata	Import	KERNEL32.GetCommandLineA	
00403138	.idata	Import	USER32.GetDlgItem	
0040313C	.idata	Import	USER32.GetDlgItemTextA	
0040310C	.idata	Import	msvcrt._getmainargs	
004030DC	.idata	Import	KERNEL32.GetModuleHandleA	
004030E0	.idata	Import	KERNEL32.GetStartupInfoA	
00403140	.idata	Import	USER32.GetWindowTextLengthA	
004030E4	.idata	Import	KERNEL32.GlobalAlloc	
00403104	.idata	Import	msvcrt._iob	
00403118	.idata	Import	msvcrt._memset	
00403144	.idata	Import	USER32.MessageBoxA	
004011F0	.text	Export	<ModuleEntryPoint>	
00403114	.idata	Import	msvcrt._p__environ	
00403148	.idata	Import	USER32.SetDlgItemTextA	
00403108	.idata	Import	msvcrt._setmode	
004030E8	.idata	Import	KERNEL32.SetUnhandledExceptionFilter	
00403124	.idata	Import	msvcrt._set_app_type	
0040311C	.idata	Import	msvcrt._signal	
00403120	.idata	Import	msvcrt._strlen	

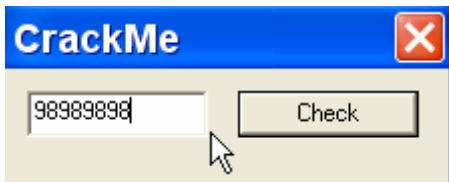
En la commandbar tipo



Ahora doy RUN con F9 y aparece la ventana para ingresar el serial



Allí tipo mi serial falso



Y apreto el botón Check y para en la api

77D6AC1E	8BFF	MOV EDI,EDI
77D6AC20	55	PUSH EBP
77D6AC21	8BEC	MOV EBP,ESP
77D6AC23	FF75 0C	PUSH DWORD PTR SS:[EBP+C]
77D6AC26	FF75 08	PUSH DWORD PTR SS:[EBP+8]
77D6AC29	E8 E396FBFF	CALL USER32.GetDlgItem
77D6AC2E	85C0	TEST EAX,EAX
77D6AC30	74 0E	JE SHORT USER32.77D6AC40
77D6AC32	FF75 14	PUSH DWORD PTR SS:[EBP+14]
77D6AC35	FF75 10	PUSH DWORD PTR SS:[EBP+10]
77D6AC38	50	PUSH EAX
77D6AC39	E8 FE74FCFF	CALL USER32.GetWindowTextA
77D6AC3E	EB 0E	JMP SHORT USER32.77D6AC4E
77D6AC40	837D 14 00	CMPI DWORD PTR SS:[EBP+14],0
77D6AC44	74 06	JE SHORT USER32.77D6AC48
77D6AC46	8B45 10	MOV EAX,DWORD PTR SS:[EBP+10]
77D6AC49	C600 00	MOV BYTE PTR DS:[EAX],0
77D6AC4C	33C0	XOR EAX,EAX
77D6AC4E	5D	POP EBP
77D6AC4F	C2 1000	RETN 10
77D6AC52	90	NOP

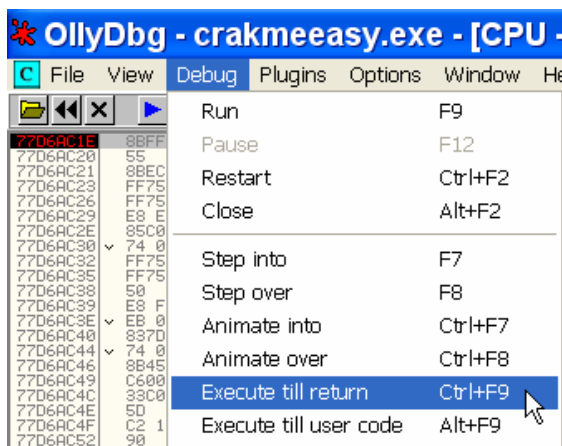
En el stack miro los parámetros

0240F998	00401303	CALL to GetDlgItemTextA from crakmea.004012FE
0240F99C	00770A36	hWnd = 00770A36 ('CrackMe',class='#32770')
0240F9A0	00000191	ControlID = 191 (401.)
0240F9A4	00044BB8	Buffer = 00044BB8
0240F9A8	00000009	Count = 9
0240F9AC	05011BD1	
0240F9B0	0240FA3C	

Allí vemos el BUFFER donde guardara el serial falso, marco esa línea y hago CLICK DERECHO-FOLLOW IN DUMP

Address	Hex dump	ASCII
00044BB8	00 00 00 00 00 00 00 00	.....
00044BC0	00 AB AB AB AB AB AB AB	.....
00044BC8	AB FE EE FE EE FE EE FE	.....
00044BD0	00 00 00 00 00 00 00 00	.....
00044BD8	56 00 05 00 EE 04 EE 00	U..-♦.
00044BE0	28 04 04 00 28 04 04 00	(♦♦.(♦♦.
00044BE8	EE FE EE FE EE FE EE FE	.....
00044BF0	EE FE EE FE EE FE EE FE	.....

Allí esta el buffer vacío, porque aun no se ejecuto la api, así que hago DEBUG-EXECUTE TILL RETURN



Con lo cual ejecuto la api y llego al RET de la misma.

77D6AC46	8B45 10	MOV EAX,DWORD PTR SS:[EBP+1
77D6AC49	C600 00	MOV BYTE PTR DS:[EAX],0
77D6AC4C	33C0	XOR EAX,EAX
77D6AC4E	5D	POP EBP
77D6AC4F	C2 1000	RETN 10
77D6AC52	90	NOP
77D6AC53	90	NOP
77D6AC54	90	NOP
77D6AC55	90	NOP

Apreto F7 para volver al programa

Address	Hex dump	ASCII
00044BB8	39 38 39 38 39 38 39 38	98989898
00044BC0	00 AB AB AB AB AB AB AB	.....
00044BC8	AB FE EE FE EE FE EE FE	.....
00044BD0	00 00 00 00 00 00 00 00	.....
00044BD8	56 00 05 00 EE 04 EE 00	U..-♦.
00044BE0	28 04 04 00 28 04 04 00	(♦♦.(♦♦.
00044BE8	EE FE EE FE EE FE EE FE	.....
00044BF0	EE FE EE FE EE FE EE FE	.....

Y veo que en el BUFFER esta ahora el serial falso que ingrese.

004012FA	. 8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	
004012FD	. 50	PUSH EAX	
004012FE	. E8 4D030000	CALL <JMP.&USER32.GetDlgItemTextA>	hWnd GetDlgItemTextA
00401303	. C745 F0 0000	MOV DWORD PTR SS:[EBP-10],0	
00401308	. B8 22124000	MOV EAX,crakmeea.00401222	ASCII "10445678951"
0040130F	. 8B10	MOV EDX,DWORD PTR DS:[EAX]	
00401311	. 8955 D0	MOV DWORD PTR SS:[EBP-30],EDX	
00401314	. 8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]	
00401317	. 8955 D4	MOV DWORD PTR SS:[EBP-2C],EDX	
0040131A	. 8B40 08	MOV EAX,DWORD PTR DS:[EAX+8]	
0040131D	. 8945 D8	MOV DWORD PTR SS:[EBP-28],EAX	
00401320	. 8D45 DC	LEA EAX,DWORD PTR SS:[EBP-24]	
00401323	. 83C4 FC	ADD ESP,-4	
00401326	. 6A 08	PUSH 8	
00401328	. 6A 00	PUSH 0	
0040132A	. 50	PUSH EAX	
0040132B	. E8 F0020000	CALL <JMP.&msvcrt.memset>	n = 8 s = 00 memset
00401330	. 83C4 10	ADD ESP,10	
00401333	. C745 CC 0000	MOV DWORD PTR SS:[EBP-34],0	
0040133A	. 8DB6 00000000	LEA ESI,DWORD PTR DS:[ESI]	
00401340	. 83C4 F4	ADD ESP,-0C	
00401343	. 8D45 D0	LEA EAX,DWORD PTR SS:[EBP-30]	
00401346	. 50	PUSH EAX	
00401347	. E8 DC020000	CALL <JMP.&msvcrt.strlen>	strlen
0040134C	. 83C4 10	ADD ESP,10	
0040134F	. 89C0	MOV EAX,EAX	
00401351	. 8D50 FF	LEA EDX,DWORD PTR DS:[EAX-1]	
00401354	. 3955 F0	CMP DWORD PTR SS:[EBP-10],EDX	
00401357	. 72 07	JBE SHORT crakmeea.00401360	
00401359	. EB 35	JMP SHORT crakmeea.00401360	
0040135B	. 90	NOP	
0040135C	. 8D7426 00	LEA ESI,DWORD PTR DS:[ESI]	
00401360	. 8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
00401363	. 8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]	
00401366	. 01D0	ADD EAX,EDX	
00401368	. 0FBF10	MOVSX EDX,BYTE PTR DS:[EAX]	
0040136A	. 8D42 FC	LEA EAX,DWORD PTR DS:[EDX-14]	
0040136E	. 8D55 D0	LEA EDX,DWORD PTR SS:[EBP-30]	
00401371	. 8B40 F0	MOV ECX,DWORD PTR SS:[EBP-10]	
00401374	. 0FBF1411	MOVSX EDX,BYTE PTR DS:[ECX+EDX]	
00401378	. 39D0	CMP EAX,EDX	
0040137A	. 75 00	JNZ SHORT crakmeea.00401389	
0040137C	. 8D45 D0	LEA EAX,DWORD PTR SS:[EBP-30]	
0040137F	. 8B55 F4	MOV EDX,DWORD PTR SS:[EBP-10]	
Stack SS:[0240FA04]=0240000F			

Allí veo una larga rutina con un numero constante, que si alguno tuvo la idea de probar si es el serial correcto ya sabrá que no lo es jeje.

004012FE	. E8 4D030000	CALL <JMP.&USER32.GetDlgItemTextA>	GetDlgItemTextA
00401303	. C745 F0 0000	MOV DWORD PTR SS:[EBP-10],0	
00401308	. B8 22124000	MOV EAX,crakmeea.00401222	ASCII "10445678951"
0040130F	. 8B10	MOV EDX,DWORD PTR DS:[EAX]	
00401311	. 8955 D0	MOV DWORD PTR SS:[EBP-30],EDX	
00401314	. 8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]	

Allí mueve 401222 a EAX y en el registro EAX vemos que dicha dirección apunta a la string del numero constante.

Registers (FPU)	
EAX	00401222 ASCII "10445678951"
ECX	77D3219D USER32.77D3219D
EDX	00040608
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 crakmeea.00401240
EDI	0240FA7C
EIP	0040130F crakmeea.0040130F
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
7 0	DS 0023 32bit 0(FFFFFFFF)

Address	Hex dump	ASCII
00401222	31 30 34 34 35 36 37 38	10445678
0040122A	39 35 31 00 43 6F 72 72	951.Corr
00401232	65 63 74 21 00 49 6E 76	ect!.Inv
0040123A	61 6C 69 64 21 00 55 89	alid!.Ue
00401242	E5 83 EC 68 8B 45 0C 83	ôâÿhiE.ä
0040124A	F8 10 0F 84 C3 01 00 00	°¸*ä}0..
00401252	83 F8 10 77 0E 83 F8 02	ä°¸wä°0
0040125A	0F 84 B5 01 00 00 E9 C3	*äA0..ü†
00401262	01 00 00 3D 10 01 00 00	0..=00..
0040126A	74 0C 3D 11 01 00 00 74	t.=00..t
00401272	00 00 00 00 00 00 00 00	.....

En la siguiente línea como EAX vale 401222 ,

MOV EDX,DWORD PTR DS:[EAX]

En realidad es similar a

MOV EDX,DWORD PTR DS:[401222]



0040130A	. B8 22124000	MOV EAX,crakmea.00401222	ASC
0040130F	. 8B10	MOV EDX,DWORD PTR DS:[EAX]	
00401311	. 8955 D0	MOV DWORD PTR SS:[EBP-30],EDX	
00401314	. 8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]	
00401317	. 8955 D4	MOV DWORD PTR SS:[EBP-2C],EDX	
0040131A	. 8B40 08	MOV EAX,DWORD PTR DS:[EAX+8]	
0040131D	. 8945 D8	MOV DWORD PTR SS:[EBP-28],EAX	
00401320	. 8D45 DC	LEA EAX,DWORD PTR SS:[EBP-24]	
00401323	. 8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]	

O sea que mueve el contenido de 401222 a EDX.

En la aclaración del OLLY se ve bien que son los 4 primeros bytes del numero 10445678951

0040137C	. 8D45 D0	LEA EAX,DWORD PTR SS:
0040137E	. 8B55 F0	MOV EDX,DWORD PTR SS:
DS:[00401222]=34343031		
EDX=00040608		
Address	Hex dump	ASCII
00401222	31 30 34 34 35 36 37 38	10445678
0040122A	39 35 31 00 43 6F 72 72	951.Corr
00401232	2E 23 74 31 0A 40 2E 72	...* ...

Al ejecutar la línea con F7 se mueven a EDX (siempre se moverán al revés al mover de la memoria a un registro)

Registers (FPU)	
EAX	00401222 ASCII
ECX	77D32190 USER32
EDX	34343031
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 crakme
EDI	0240FA7C
EIP	00401311 crakme
C 0	ES 0023 32bit

La siguiente línea esos bytes que están en EDX los mueve a [EBP-30]

0040130F	. 8B10	MOV EDX,DWORD PTR DS:[EAX]
00401311	. 8955 D0	MOV DWORD PTR SS:[EBP-30],EDX
00401314	. 8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]
00401317	. 8955 D4	MOV DWORD PTR SS:[EBP-2C],EDX

En la aclaración del OLLY vemos que [EBP-30] es en mi maquina 240f9e4, lo busco en el DUMP

0040137C	. 8D45 D0	LEA EAX
0040137E	. 8B55 F0	MOV EDX
EDX=34343031		
Stack SS:[0240F9E4]=0004443C		

Address	Hex dump	ASCII
0240F9E4	3C 44 04 00 00 02 00 00	<D...@..
0240F9EC	03 00 00 00 03 00 00 00	...@...
0240F9F4	57 00 00 00 14 00 00 00	W...@...
0240F9FC	11 00 00 00 3C 44 04 00	!...<D...
0240FA04	00 00 00 00 B8 4B 04 00	...@K...

Al ejecutar con F7 se copiaran allí los bytes que estaban en EDX

Address	Hex dump	ASCII
0240F9E4	31 30 34 34 00 02 00 00	1044.@..
0240F9EC	03 00 00 00 03 00 00 00	...@...
0240F9F4	57 00 00 00 14 00 00 00	W...@...

Luego

00401311	. 8955 D0	MOV DWORD PTR SS:[EBP-30],EDX
00401314	. 8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]
00401317	. 8955 D4	MOV DWORD PTR SS:[EBP-2C],EDX
0040131A	. 8B40 08	MOV EAX,DWORD PTR DS:[EAX+8]
0040131D	. 8945 D8	MOV DWORD PTR SS:[EBP-28],EAX
00401320	. 8D45 DC	LEA EAX,DWORD PTR SS:[EBP-24]
00401323	. 83C4 FC	ADD ESP,-4
00401326	. 6A 08	PUSH 8

Mueve a EDX los siguientes cuatro bytes del número constante

0040132E	. 8B55 F0	MOV EDX,DWORD PTR SS:[00401226]=38373635
		EDX=34343031

Address	Hex dump	ASCII
00401222	31 30 34 34 35 36 37 38	10445678
0040122A	39 35 31 00 43 6F 72 72	951.Corr
00401232	65 63 74 21 00 49 6E 76	ectf.Inv
0040123A	61 6C 69 64 21 00 55 89	alid*.ll

La aclaración del OLLY lo muestra, en este caso [eax+4] es en este caso el contenido de 401226 y al ejecutar con F7 mueve los 4 bytes siguientes a EDX.

Registers (FPU)	
EAX	00401222 ASC
ECX	77D3219D USE
EDX	38373635
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 crs
EDI	0240FA7C
EIP	00401317 crs

Y los copia a continuación de donde copio los anteriores

Address	Hex dump	ASCII
0240F9E0	00 00 00 00 31 30 34 34	....1044
0240F9E8	35 36 37 38 03 00 00 00	5678...
0240F9F0	03 00 00 00 57 00 00 00	...w...
0240F9F8	14 00 00 00 11 00 00 00	...l...

En realidad lo que esta haciendo es copiar ese numero de a 4 bytes a otra parte de la memoria

00401314	. 8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]
00401317	. 8955 D4	MOV DWORD PTR SS:[EBP-2C],EDX
0040131A	. 8B40 08	MOV EAX,DWORD PTR DS:[EAX+8]
0040131D	. 8945 D8	MOV DWORD PTR SS:[EBP-28],EAX

Y aquí copia los últimos 4 bytes

Address	Hex dump	ASCII
0240F9E0	00 00 00 00 31 30 34 34	....1044
0240F9E8	35 36 37 38 39 35 31 00	5678951.
0240F9F0	03 00 00 00 57 00 00 00	...w...

Quedando el número completo copiado allí.

Allí vemos que un poco mas abajo se acerca a una llamada a la api memset, allí vemos los parámetros en OLLYDBG

00401323	. 83C4 FC	ADD ESP,-4	
00401326	. 6A 08	PUSH 8	n = 8
00401328	. 6A 00	PUSH 0	c = 00
0040132A	. 50	PUSH EAX	s
0040132B	. E8 F0020000	CALL <JMP.&msvcrt.memset>	memset
00401330	. 83C4 10	ADD ESP,10	

Tiene tres valores (n, c y s)

s es la dirección de inicio

n que es la cantidad de bytes que van a llenar

c es el valor con el cual se va a llenar esa zona

0240F99C	0240F9F0	s = 0240F9F0
0240F9A0	00000000	c = 00
0240F9A4	00000008	n = 8
0240F9A8	00000009	
0240F9AC	05011BD1	

En el stack se ven mejor en este caso los parámetros, o sea que llenara con ceros (c), 8 bytes (n) a partir de s (240f9f0).

Address	Hex dump	ASCII
0240F9E0	00 00 00 00 31 30 34 34	...1044
0240F9E8	35 36 37 38 39 35 31 00	5678951.
0240F9F0	00 00 00 00 00 00 00 00	.....
0240F9F8	14 00 00 00 11 00 00 00	1...4...
0240FA00	3C 44 04 00 00 00 00 00	<D...>

Al ejecutar la api con F8, allí vemos que lleno de ceros, los 8 bytes a partir de 240f9f0 que es la dirección de inicio s.

Mas abajo vemos una llamada a strlen, que es una api que calcula el largo de una string lleguemos hasta el call a la misma.

00401343	. 8D45 D0	LEA EAX,DWORD PTR SS:[EBP-30]	
00401346	. 50	PUSH EAX	
00401347	. E8 DC020000	CALL <JMP.&msvcrt.strlen>	strlen
0040134C	. 83C4 10	ADD ESP,10	

En el stack vemos los parámetros

0240F99C	0240F9E4	s = "10445678951"
0240F9A0	00000000	

O sea nos dará el largo de la string que comienza en 240f9E4, que es el numero constante famoso jeje.

Al pasar con F8 el call a dicha api en EAX nos devuelve el largo de la string

Registers (FPU)	
EAX	0000000B
ECX	0240F9E4 ASCII "10445678951"
EDX	7F303438
EBX	00000000
ESP	0240F99C
EBP	0240FA14
ESI	00401240 crakmees.00401240
EDI	0240FA7C
EIP	0040134C crakmees.0040134C

Vemos que el largo es 0B que es 11 decimal, que es el largo del número constante.

0040134F	. 89C0 --	MOV EAX,EAX
00401351	. 8D50 FF	LEA EDX,DWORD PTR DS:[EAX-1]
00401354	. 3955 F0	CMP DWORD PTR SS:[EBP-10],EDX
00401357	. 72 07	JB SHORT crakmees.00401360
00401359	. EB 35	JMP SHORT crakmees.00401390

Allí le resta uno a EAX o sea 0B-1 y como es LEA mueve ese valor directo a EDX o sea que EDX vale 0A.

En la siguiente línea compara EDX que vale 0A con el contenido de [EBX-10] que es cero.

0040134F	. 89C0 --	MOV EAX,EAX
00401351	. 8D50 FF	LEA EDX,DWORD PTR DS:[EAX-1]
00401354	. 3955 F0	CMP DWORD PTR SS:[EBP-10],EDX
00401357	. 72 07	JB SHORT crakmees.00401360
00401359	. EB 35	JMP SHORT crakmees.00401390
0040135B	. 90	NOP
0040135C	. 8D7426 00	LEA ESI,DWORD PTR DS:[ESI]
00401360	. 8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
00401363	. 8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]

Pues como cero es mas bajo que 0A en la comparación, salta y va a 40135C.

0040135C	8D7426 00	LEA ESI,DWORD PTR DS:[ESI]
00401360	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
00401363	8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]
00401366	01D0	ADD EAX,EDX

En la próxima línea mueve a EAX el puntero a nuestro serial falso, allí vemos que luego de ejecutar con F7, EAX apunta a nuestro serial falso 98989898

Registers (FPU)	
EAX	00044BB8 ASCII "98989898"
ECX	0240F9E4 ASCII "10445678951"
EDX	00000000
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 crakmeea.00401240
EDI	0240FA7C
EIP	00401363 crakmeea.00401363
C 1	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)

En la siguiente línea mueve a EDX cero

0040135C	8D7426 00	LEA ESI,DWORD PTR DS:[ESI]
00401360	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
00401363	8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]
00401366	01D0	ADD EAX,EDX
00401369	0FBE10	MOVSX EDX,BYTE PTR DS:[EAX]

Registers (FPU)	
EAX	00044BB8 ASCII "
ECX	0240F9E4 ASCII "
EDX	00000000
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 crakmea
EDI	0240FA7C
EIP	00401366 crakmea
C 1	ES 0023 32bit 0

En la siguiente línea

00401363	8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]
00401366	01D0	ADD EAX,EDX
00401368	0FBE10	MOVSX EDX,BYTE PTR DS:[EAX]
0040136B	8D42 EC	LEA EAX,DWORD PTR DS:[EAX-14]

Como EAX apunta al inicio de nuestro serial falso, le suma en este caso EDX que vale cero posiblemente para hacer un LOOP que se ira actualizando incrementando EDX a 1, 2 etc y recorrer de esa forma todos los bytes de nuestro serial falso uno a uno en dicho LOOP.

00401363	8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]
00401366	01D0	ADD EAX,EDX
00401368	0FBE10	MOVSX EDX,BYTE PTR DS:[EAX]
0040136B	8D42 EC	LEA EAX,DWORD PTR DS:[EAX-14]
0040136E	8D55 D0	LEA EDX,DWORD PTR SS:[EBP-30]
00401371	8B4D F0	MOV ECX,DWORD PTR SS:[EBP-10]
00401374	0FBE1411	MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378	39D0	CMP EAX,EDX
0040137A	75 00	JNZ SHORT crakmeea.00401389
0040137C	8D45 D0	LEA EAX,DWORD PTR SS:[EBP-30]
0040137F	8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]
00401382	C60402 73	MOV BYTE PTR DS:[EDX+EAX],73
00401386	FF45 CC	INC DWORD PTR SS:[EBP-34]
00401389	FF45 F0	INC DWORD PTR SS:[EBP-10]
0040138C	EB B2	JMP SHORT crakmeea.00401340
0040138E	89F6	MOV ESI,ESI
00401390	B8 2E124000	MOV EAX,crakmeea.0040122E
00401395	3B10	MOV EDX,DWORD PTR DS:[EAX]
00401397	8955 B0	MOV DWORD PTR SS:[EBP-50],EDX
0040139A	8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]
0040139D	8955 B4	MOV DWORD PTR SS:[EBP-4C],EDX

DS:[00044BB8]=39 ('9')

EDX=00000000

Como ya sabemos que MOVSX en el caso de números positivos mueve el byte a EDX y si es positivo llenara el resto con ceros, y si es negativo con efes jeje.

En este caso no hay problema mueve el primer byte de mi serial falso el 39 a EDX al ejecutar la línea vemos que EDX vale 39.

Registers (FPU)	
EAX	00044BB8 ASCII "
ECX	0240F9E4 ASCII "
EDX	00000039
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 crakmee
EDI	0240FA7C
EIP	0040136B crakmee
C 0	ES 0023 32bit 0
P 1	CS 001B 32bit 0
A 0	SS 0023 32bit 0
Z 0	DS 0023 32bit 0
S 0	FS 003B 32bit 7

La siguiente línea es un LEA

00401366	. 01D0	ADD EAX,EDX
00401368	. 0FBE10	MOVSX EDX,BYTE PTR DS:[EAX]
0040136B	. 8D42 EC	LEA EAX,DWORD PTR DS:[EDX-14]
0040136E	. 8D55 D0	LEA EDX,DWORD PTR SS:[EBP-30]
00401371	. 8B4D F0	MOV ECX,DWORD PTR SS:[EBP-10]
00401374	. 0FBE1411	MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378	. 39D0	CMP EAX,EDX

Y como EDX vale 39, le resta 14 y como es un LEA mueve el resultado directo a EAX.

Registers (FPU)	
EAX	00000025
ECX	0240F9E4 ASCII "
EDX	00000039
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 cr.
EDI	0240FA7C
EIP	0040136E cr.
C 0	ES 0023 32
P 1	CS 001B 32
A 0	SS 0023 32

O sea que la operación que realizo es tomar 39 que es el valor hexa del primer carácter de mi serial, y le resto 14 y quedo por ahora en 25, el cual esta en EAX.

00401368	. 0FBE10	MOVSX EDX,BYTE PTR DS:[EAX]
0040136B	. 8D42 EC	LEA EAX,DWORD PTR DS:[EDX-14]
0040136E	. 8D55 D0	LEA EDX,DWORD PTR SS:[EBP-30]
00401371	. 8B4D F0	MOV ECX,DWORD PTR SS:[EBP-10]
00401374	. 0FBE1411	MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378	. 39D0	CMP EAX,EDX

La siguiente línea mueve el valor EBP-30 que en mi maquina es 240f9E4 que apunta al inicio del numero constante guardado a EDX.

0040139A	. 8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]
0040139D	. 8B55 B4	MOV EDI,DWORD PTR SS:[EBP-4C]
Stack address=0240F9E4, (ASCII "10445678951")		
EDX=00000039		

Al apretar F7

Registers (FPU)	
EAX	00000025
ECX	0240F9E4 ASCII "10445678951"
EDX	0240F9E4 ASCII "10445678951"
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 crakmeea.00401240
EDI	0240FA7C
EIP	00401371 crakmeea.00401371
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 0	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 002B 32bit 7E9F0000(EEF)

Queda EDX apuntando al inicio del número constante.

0040136E	. 8D55 D0	LEA EDX,DWORD PTR SS:[EBP-30]
00401371	. 8B4D F0	MOV ECX,DWORD PTR SS:[EBP-10]
00401374	. 0FBE1411	MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378	. 39D0	CMP EAX,EDX
0040137A	. 75 0D	JNZ SHORT crakmeea.00401389

Aquí vemos que mueve a ECX el valor cero y en cada pasada que haga en el LOOP se incrementará, permitiendo a ECX+EDX apuntar a los distintos bytes del numero constante.

0040136E	. 8D55 D0	LEA EDX,DWORD PTR SS:[EBP-30]
00401371	. 8B4D F0	MOV ECX,DWORD PTR SS:[EBP-10]
00401374	. 0FBE1411	MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378	. 39D0	CMP EAX,EDX
0040137A	. 75 0D	JNZ SHORT crakmeea.00401389

Ahí vemos como ECX por ahora vale cero y EDX apunta al inicio del numero, moverá a EDX en este primer paso por esta línea, el primer byte del numero vemos en la aclaración del OLLYDBG.

00401374	. 0FBE1411	MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378	. 39D0	CMP EAX,EDX
0040137A	. 75 0D	JNZ SHORT crakmeea.00401389
Stack DS:[0240F9E4]=31 ('1')		
EDX=0240F9E4, (ASCII "10445678951")		

Que mueve el byte 31 que corresponde al 1 en ASCII, es la primera cifra del número constante.

00401371	. 8B4D F0	MOV ECX,DWORD PTR SS:[EBP-10]
00401374	. 0FBE1411	MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378	. 39D0	CMP EAX,EDX
0040137A	. 75 0D	JNZ SHORT crakmeea.00401389
0040137C	. 8D45 D0	LEA EAX,DWORD PTR SS:[EBP-30]
0040137E	. 0F5C F0	MOVSX EAX,BYTE PTR DS:[ECX+EDX]

Allí llegamos a una comparación como recordamos

00401374	. 0FBE1411	MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378	. 39D0	CMP EAX,EDX
0040137A	. 75 0D	JNZ SHORT crakmeea.00401389
0040137C	. 8D45 D0	LEA EAX,DWORD PTR SS:[EBP-30]
0040137E	. 0F5C F0	MOVSX EAX,BYTE PTR DS:[ECX+EDX]
EDX=00000031		
EAX=00000025		

En EAX esta el valor del primer byte de mi serial falso 39 al cual le resto 14, quedando en EAX el valor 25, y en EDX el primer byte del numero constante o sea 31

Por lo tanto vemos que

CMP EAX,EDX

En realidad es

CMP (PRIMER BYTE DE MI SERIAL FALSO – 14), PRIMER BYTE DEL NUMERO CONSTANTE

CMP 25,31

Y al ser la diferencia entre ambos miembros diferente de cero, no se activa el FLAG Z y salta el JNZ.

Como estoy utilizando el serial falso, la comparación no es igual, solo serán iguales ambos miembros cuando usemos el serial correcto ya que es el valor que hace que la comparación llegue a ser una igualdad.

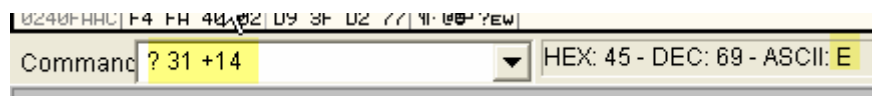
$\text{CMP (PRIMER BYTE DE MI SERIAL CORRECTO - 14), 31}$

Ya que la condición es que ambos miembros sean iguales

$\text{PRIMER BYTE DEL SERIAL CORRECTO} - 14 = \text{PRIMER BYTE DEL NUMERO CONSTANTE}$

Por lo tanto

$\text{PRIMER BYTE DEL SERIAL CORRECTO} = \text{PRIMER BYTE DEL NUMERO CONSTANTE} + 14$



$\text{PRIMER BYTE DEL SERIAL CORRECTO} = 31 + 14$

$\text{PRIMER BYTE DEL SERIAL CORRECTO} = 45$  que corresponde a E en ASCII.

O sea que la primera letra del serial es E.

Esta afirmación si repetimos el loop vemos que se cumple byte a byte

$\text{PRIMER BYTE DEL SERIAL CORRECTO} = \text{PRIMER BYTE DEL NUMERO CONSTANTE} + 14$

$\text{SEGUNDO BYTE DEL SERIAL CORRECTO} = \text{SEGUNDO BYTE DEL NUMERO CONSTANTE} + 14$

$\text{TERCER BYTE DEL SERIAL CORRECTO} = \text{TERCER BYTE DEL NUMERO CONSTANTE} + 14$

Y así sucesivamente

Siguiendo la misma lógica, a cada byte del numero constante, le sumamos 14 y obtenemos el valor del byte correspondiente a nuestro serial correcto.

31 30 34 34 35 36 37 38 10445678  
39 35 31 00 00 00 00 00 951.....

$31 + 14 = 45$  es la letra E en ASCII

$30 + 14 = 44$  es la letra D en ASCII

$34 + 14 = 48$  es la letra H en ASCII

$34 + 14 = 48$  es la letra H en ASCII

$35 + 14 = 49$  es la letra I en ASCII

$36 + 14 = 50$  es la letra J en ASCII

$37 + 14 = 51$  es la letra K en ASCII

$38 + 14 = 52$  es la letra L en ASCII

$39 + 14 = 53$  es la letra M en ASCII

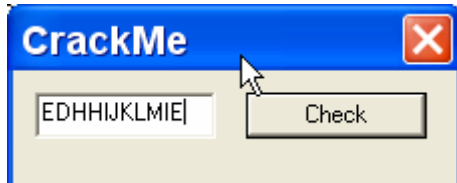
$35 + 14 = 48$  es la letra **I** en ASCII

$31 + 14 = 45$  es la letra **E** en ASCII

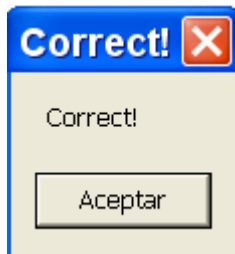
Por lo cual el serial correcto es

EDHHIJKLMIE

Pongámoslo en la ventana del serial borrando todos los BPX.



Apretemos Check



Les dejo como tarea recorrer el loop completo, ver como va actualizando los contadores y como va incrementando los bytes del numero constante y del serial falso, y como llega al cartel de correct o de incorrect.

No es tan easy el crackme jeje ya que recién están empezando, pero creo que si lo practican pueden sacarle buen jugo.

Bueno aquí tienen la tarea sencilla para la parte 15 es otro crackme llamado SPLISH, solo tienen que hallar en la parte de HARDCODED el serial correcto y con el serial abrirán la parte 15 jeje sigo malísimo, pero les digo que este crackme no se parece en nada a este anterior que hice, es bien sencillo, ya iremos practicando mas difíciles de a poco.

Ricardo Narvaja

05 de diciembre de 2005



