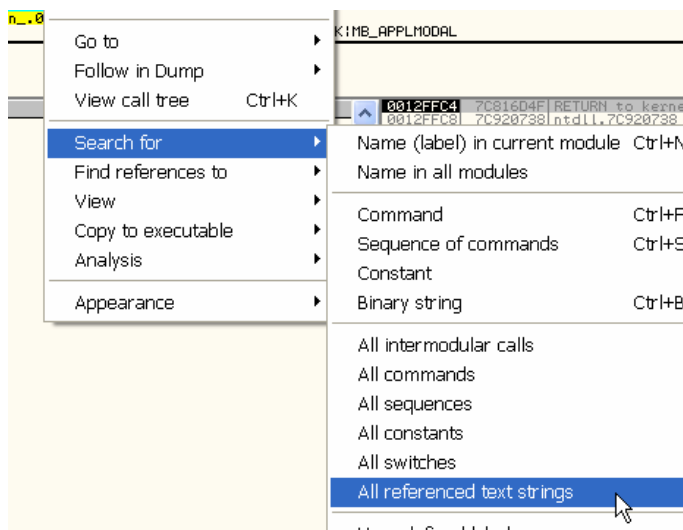


Para hallar las cadenas de texto o strings, hacemos click derecho SEARCH FOR – ALL REFERENCED TEXT STRINGS.



Los resultados son

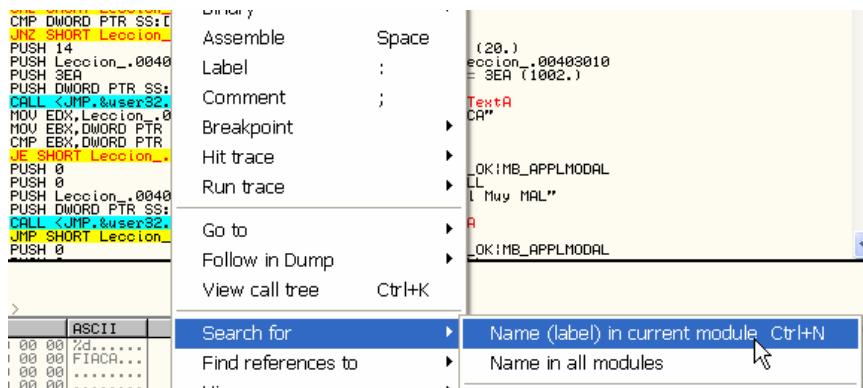
Address	Disassembly	Text string
00401000	PUSH 0	(Initial CPU selection)
00401061	MOV EDX, Leccion_.00403008	ASCII "FIACA"
00401078	PUSH Leccion_.00402035	ASCII "Mal Muy MAL"

Pues allí se ve la palabra FIACA que puede ser la clave, si no tenemos ganas de trabajar podemos probar algunas strings que salen en la lista y por ahí acertamos, pero este método no es aconsejable, por las siguientes causas:

1) En este caso hay 2 strings, pero hay programas que tienen miles de strings, y probar una por una puede llevar a la locura total, así que aunque vemos allí el posible serial correcto, haremos como que hay mil strings y que no sabemos cual es.

2) Hay programas que ponen strings tramposas en la lista que no son la clave y que al tipearlas al ingresar un serial provocan algún perjuicio ya que son trampas para crackers, así que lo mejor es asegurarse y verificar que es el serial correcto llegando hasta la comparación entre lo que tipeas y la string correcta.

Lo primero es mirar las apis que utiliza con click derecho-SEARCH FOR- NAME (LABEL) IN CURRENT MODULE



Aquí salen las apis utilizadas por el programa, veamos si hay alguna conocida

Address	Section	Type	Name	Comment
00402018	.rdata	Import	user32.DialogBoxParamA	
00402020	.rdata	Import	user32.EndDialog	
0040200C	.rdata	Import	kernel32.ExitProcess	
0040201C	.rdata	Import	user32.GetDlgItemTextA	
00402008	.rdata	Import	kernel32.GetModuleHandleA	
00402000	.rdata	Import	comctl32.InitCommonControls	
00402014	.rdata	Import	user32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	

Allí vemos que utiliza GetDlgItemTextA, seguro la usara para ingresar el serial que nosotros tipeamos y MessageBoxA para sacar los mensajes de si es correcto o incorrecto nuestro serial, pongamos un BPX en cada una de dichas apis.

00402018	.rdata	Import	user32.DialogBoxParamA	
00402020	.rdata	Import	user32.EndDialog	
0040200C	.rdata	Import	kernel32.ExitProcess	
0040201C	.rdata	Import	user32.GetDlgItemTextA	
00402008	.rdata	Import	kernel32.GetModuleHandleA	
00402000	.rdata	Import	comctl32.InitCommonControls	
00402014	.rdata	Import	user32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	

Actualize

Follow import in Disassembler

Follow in Dump

Find references to import Enter

View call tree

**Toggle breakpoint on import**

Conditional breakpoint on import

Conditional log breakpoint on import

En ambas click derecho – TOGGLE BREAKPOINT ON IMPORT o en la commandbar

Command

Bp GetDlgItemTextA

Analysing Leccion : 2 heuristical procedures, 8 calls to known functions

Command

Bp MessageBoxA

BP address, string --

Analysing Leccion : 2 heuristical procedures, 8 calls to known functions

Bueno pulsemos F9 para correr el crackme

Serial

Verificar

Salir

Allí sale la ventana para tipear el serial, pongamos un nombre cualquiera por ejemplo narvajita jeje

Serial

narvajita

Verificar

Salir

Apretemos Verificar y vemos que para en el OLLYDBG en uno de los BPX que colocamos

```

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 E9BFBFF 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 FE74CFF CALL user32.GetWindowTextA
77D6AC3E EB 0E     JMP SHORT user32.77D6AC4E
77D6AC40 837D 14 00 CMP DWORD PTR SS:[EBP+14],0
77D6AC44 74 06     JE SHORT user32.77D6AC4C
77D6AC46 8B45 10   MOV EAX,DWORD PTR SS:[EBP+10]
77D6AC49 C600 00   MOV BYTE PTR DS:[EAX],0
77D6AC4C 33C0      XOR EAX,EAX
77D6AC4E 5D        POP EBP
77D6AC4F C2 1000   RETN 10
77D6AC52 90        NOP
77D6AC53 90        NOP
77D6AC54 90        NOP
77D6AC55 90        NOP

```

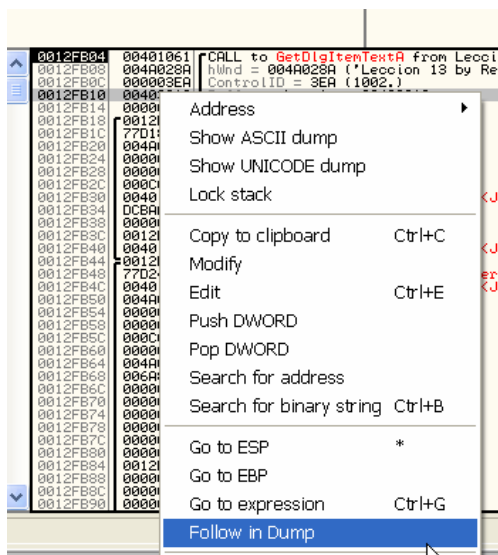
Si miramos el stack, vemos que paro en la api GetDlgItemTextA para ingresar el serial, allí en los parámetros de la api, vemos la dirección del BUFFER donde guardara el serial que ingresa, en este caso es 403010.

```

0012FB04 00401061 CALL to GetDlgItemTextA from Leccion_004
0012FB08 004A028A hWnd = 004A028A ('Leccion 13 by RedH@wK
0012FB0C 000003EA ControlID = 3EA (1002.)
0012FB10 00403010 Buffer = Leccion_00403010
0012FB14 00000014 Count = 14 (20.)
0012FB18 0012FB44
0012FB1C 77D18734 RETURN to user32.77D18734

```

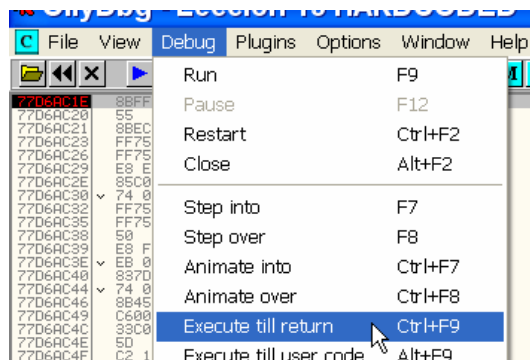
Así que veamos esa dirección en el DUMP haciendo click derecho – FOLLOW IN DUMP en la misma



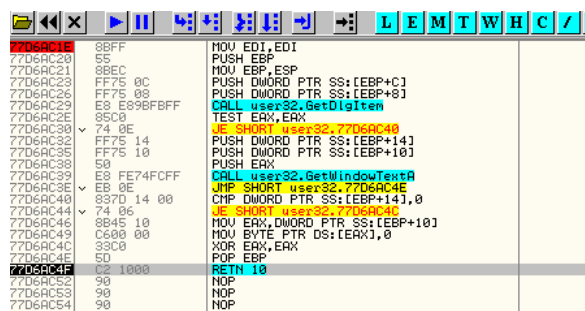
Allí esta visible el BUFFER, y esta vacío pues aun no se ha ejecutado la api

Address	Hex dump	ASCII
00403010	00 00 00 00 00 00 00 00	.....
00403018	00 00 00 00 00 00 00 00	.....
00403020	00 00 00 00 00 00 00 00	.....
00403028	00 00 00 00 00 00 00 00	.....
00403030	00 00 00 00 00 00 00 00	.....
00403038	00 00 00 00 00 00 00 00	.....
00403040	00 00 00 00 00 00 00 00	.....
00403048	00 00 00 00 00 00 00 00	.....
00403050	00 00 00 00 00 00 00 00	.....
00403058	00 00 00 00 00 00 00 00	.....

Hagamos EXECUTE TILL RETURN para que llegue al RET de la misma.



Ahora estamos en el RET

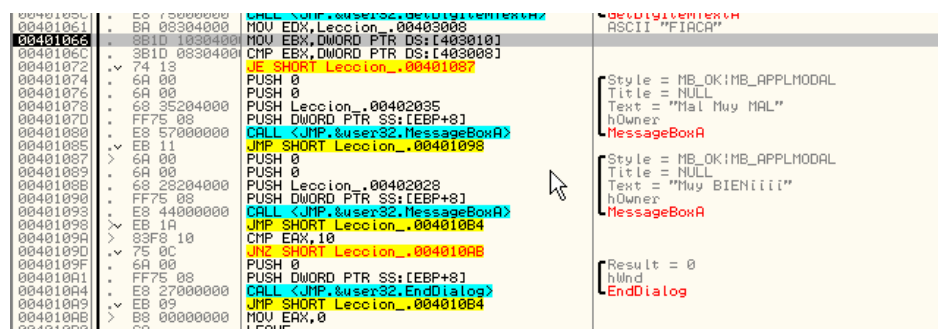


Y la api guardo en el BUFFER el serial que ingrese

Address	Hex dump	ASCII
00403010	6E 61 72 76 61 6A 69 74	narvajit
00403018	61 00 00 00 00 00 00 00	a.....
00403020	00 00 00 00 00 00 00 00	.....
00403028	00 00 00 00 00 00 00 00	.....
00403030	00 00 00 00 00 00 00 00	.....
00403038	00 00 00 00 00 00 00 00	.....

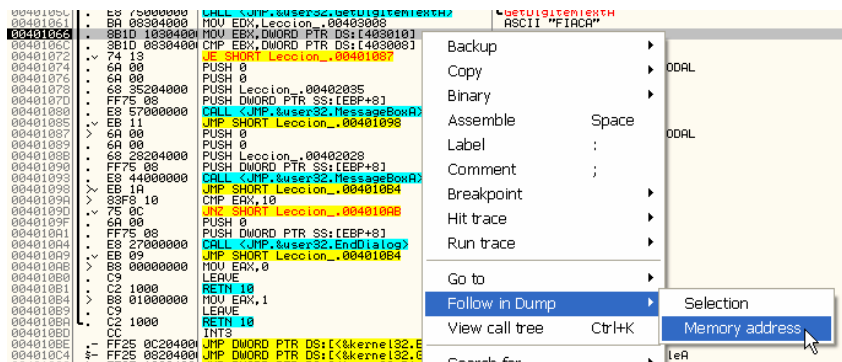
Pues allí lo tengo

Si apreto F7 y vuelvo al programa



Ya vemos viendo que estamos llegando a la comparación y al salto condicional que según salte o no me llevara a los MessageBoxA de “Mal Muy MAL” o “Muy BIEN” obviamente si ese salto condicional lo cambio por un JMP 401087, lo obligo siempre a saltar al cartel de MUY BIEN y programa parchado y resuelto, pero aquí yo quiero hallar el serial, así que veamos que compara.

En 401066 moverá a EBX el contenido de la memoria 403010 como es un DWORD solo moverá los 4 bytes pero es suficiente, veamos que hay en 403010 en el dump.



Marco la línea y hago click derecho FOLLOW IN DUMP-MEMORY ADDRESS y vere que esta leyendo

004010BD	CC	INT3
004010BE	FF25 0C204000	JMP DWORD PTR DS:[&&kerne
004010C4	FF25 08204000	JMP DWORD PTR DS:[&&kerne
004010C4	FF25 18204000	JMP DWORD PTR DS:[&&user32

Address	Hex dump	ASCII
00403010	6E 61 72 76 61 6A 69 74	narvajit
00403018	61 00 00 00 00 00 00 00	a.....
00403020	00 00 00 00 00 00 00 00	.....
00403028	00 00 00 00 00 00 00 00	.....
00403030	00 00 00 00 00 00 00 00	.....
00403038	00 00 00 00 00 00 00 00	.....

En la aclaración veo que esta leyendo 7672616E que son los 4 bytes que se encuentran en el contenido de 403010 leídos al revés como vimos, y son los primeros 4 bytes del serial falso que yo tipee los cuales mueve a EBX.

Registers (FPU)	
EAX	00000009
ECX	77D3219D user32
EDX	00403008 ASCII
EBX	7672616E
ESP	0012FB10
EBP	0012FB10
ESI	0040102E Leccion_
EDI	0012FB80
EIP	0040106C Leccion_
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

Ahora apreto F7 y paso a la siguiente línea

0040105C	EB 75000000	CALL <JMP.&user32.GetDlgItemText>	GetDlgItemText
00401061	BA 08304000	MOV EDX,Leccion_.00403008	ASCII "FIACA"
00401066	8B1D 10304000	MOV EBX,DWORD PTR DS:[403010]	
0040106C	3B1D 08304000	CMP EBX,DWORD PTR DS:[403008]	
00401072	74 13	JE SHORT Leccion_.00401087	
00401074	6A 00	PUSH 0	Style = MB_OF
00401076	6A 00	PUSH 0	Title = NULL

Aquí vemos que compara EBX que como recordaran son los primeros 4 bytes de mi serial falso, con el contenido de la dirección 403008, veamos que hay allí en el dump, de la misma forma que hicimos en la línea anterior.

004010CA	FF25 18204000	JMP DWORD PTR DS:[&&
----------	---------------	----------------------

Address	Hex dump	ASCII
00403008	46 49 41 43 41 00 00 00	FIACA...
00403010	6E 61 72 76 61 6A 69 74	narvajit
00403018	61 00 00 00 00 00 00 00	a.....
00403020	00 00 00 00 00 00 00 00	.....
00403028	00 00 00 00 00 00 00 00	.....
00403030	00 00 00 00 00 00 00 00	.....

Vemos que esta leyendo los 4 primeros bytes de la palabra FIACA, comparándolos con los primeros cuatro bytes que tipee, por lo cual deduzco que si ambos fueran iguales en esta comparación, se activaría el flag Z, al ser la diferencia entre ambos cero, y el JE saltaría, mandándome el cartel de MUY BIEN, por ahora como no son iguales recibiré el escarmiento jeje,

00401061	BA 08304000	MOV EDI,Leccion_.00403008	ASCII "FIACA"
00401066	3B1D 10304000	MOV EBX,DWORD PTR DS:[403010]	
0040106C	3B1D 08304000	CMP EBX,DWORD PTR DS:[403008]	
00401072	74 13	JE SHORT Leccion_.00401087	
00401074	6A 00	PUSH 0	Style = MB_OK!MB_APPLMODAL
00401076	6A 00	PUSH 0	Title = NULL
00401078	68 35204000	PUSH Leccion_.00402035	Text = "Mal Muy MAL"
0040107D	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401080	E8 57000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
00401085	EB 11	JMP SHORT Leccion_.00401098	
00401087	6A 00	PUSH 0	Style = MB_OK!MB_APPLMODAL
00401089	6A 00	PUSH 0	Title = NULL
0040108B	68 28204000	PUSH Leccion_.00402028	Text = "Muy BIENiiii"
00401090	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401093	E8 44000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
00401098	EB 1A	JMP SHORT Leccion_.004010B4	
0040109A	83F8 10	CMP EAX,10	

Como no fueron iguales no salta y va directo al cartel MAL MUY MAL, pues apretemos F9

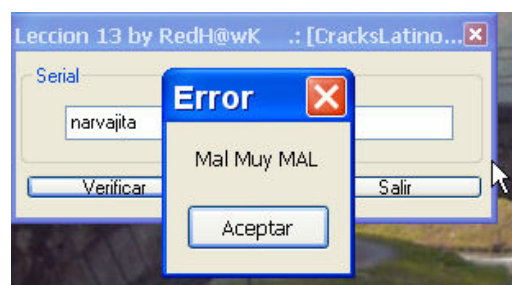
77D504EA	8BFF	MOV EDI,EDI	
77D504EC	55	PUSH EBP	
77D504ED	8BEC	MOV EBP,ESP	
77D504EF	833D BC04D777	CMP DWORD PTR DS:[77D704BC],0	
77D504F6	74 24	JE SHORT user32.77D50510	
77D504F8	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]	
77D504FE	6A 00	PUSH 0	
77D50500	FF70 24	PUSH DWORD PTR DS:[EAX+24]	
77D50503	68 240BD777	PUSH user32.77D70B24	
77D50508	FF15 C812D177	CALL DWORD PTR DS:[<&KERNEL32.Interlock	kernel32
77D5050E	85C0	TEST EAX,EAX	
77D50510	75 0A	JNZ SHORT user32.77D50510	
77D50512	C705 200BD777	MOV DWORD PTR DS:[77D70B20],1	
77D5051C	6A 00	PUSH 0	
77D5051E	FF75 14	PUSH DWORD PTR SS:[EBP+14]	
77D50521	FF75 10	PUSH DWORD PTR SS:[EBP+10]	

Nos para en la otra api MessageBoxA

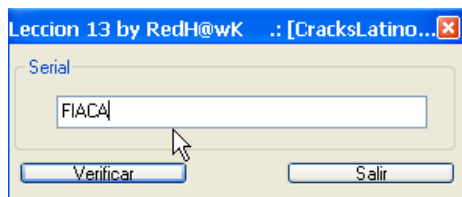
0012FB04	00401085	CALL to MessageBoxA from Leccion_.00401080	
0012FB08	004A028A	hOwner = 004A028A ('Leccion 13 by RedH@wK	:: [...',class='#32770')
0012FB0C	00402035	Text = "Mal Muy MAL"	
0012FB10	00000000	Title = NULL	
0012FB14	00000000	Style = MB_OK!MB_APPLMODAL	
0012FB18	0012FB44		
0012FB1C	77D18734	RETURN to user32.77D18734	
0012FB20	004A028A		

Y ya vemos que nos va a decir MAL MUY MAL en los parámetros de la api.

Demos run o F9



Como vemos salio el cartel malo, aceptémoslo y escribamos la clave correcta FIACA.



Apreto VERIFICAR y repito el proceso anterior hasta llegar a la comparación

```

00401061 | . BA 08304000 MOV EDX,Leccion_.00403008
00401066 | . 8B1D 10304000 MOV EBX,DWORD PTR DS:[403010]
0040106C | . 3B1D 08304000 CMP EBX,DWORD PTR DS:[403008]
00401072 | . 74 13 JE SHORT Leccion_.00401087
00401074 | . 6A 00 PUSH 0
00401076 | . 6A 00 PUSH 0

```

Como antes compara EBX con el contenido de 403008

DS:[00403008]=43414946		EBX=43414946	
Address	Hex dump	ASCII	
00403008	46 49 41 43 41 00 00 00	FIACA...	
00403010	46 49 41 43 41 00 69 74	FIACA.it	
00403018	61 00 00 00 00 00 00 00	a.....	
00403020	00 00 00 00 00 00 00 00	.....	

Ya la aclaración nos muestra que son iguales, por lo cual se restaran y dará cero el resultado el flag Z se activara al ejecutar la comparación con f7

```

EIP 00401072 Leccion_.004010
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (
EFL 00000246 (NO,NB,E,BE,NS,
ST0 empty -8,154345916481978
CT1 empty WINNOM 072A 000000

```

Allí lo vemos y como el JE salta al estar el FLAG Z activo.

00401061   . BA 08304000 MOV EDX,Leccion_.00403008	ASCII "FIACA"
00401066   . 8B1D 10304000 MOV EBX,DWORD PTR DS:[403010]	
0040106C   . 3B1D 08304000 CMP EBX,DWORD PTR DS:[403008]	
00401072   . 74 13 JE SHORT Leccion_.00401087	
00401074   . 6A 00 PUSH 0	[Style = MB_OK!MB_APPLMODAL Title = NULL Text = "Mal Muy MAL" hOwner MessageBoxA
00401076   . 6A 00 PUSH 0	
00401078   . 68 35204000 PUSH Leccion_.00402035	
0040107D   . FF75 08 PUSH DWORD PTR SS:[EBP+8]	[Style = MB_OK!MB_APPLMODAL Title = NULL Text = "Muy BIENiiii" hOwner MessageBoxA
00401080   . E8 57000000 CALL <JMP.&user32.MessageBoxA>	
00401085   . EB 11 JMP SHORT Leccion_.00401098	
00401087   . 6A 00 PUSH 0	
00401089   . 6A 00 PUSH 0	
0040108B   . 68 28204000 PUSH Leccion_.00402028	
00401090   . FF75 08 PUSH DWORD PTR SS:[EBP+8]	
00401093   . E8 44000000 CALL <JMP.&user32.MessageBoxA>	
00401098   . EB 1A JMP SHORT Leccion_.004010B4	
0040109A   . 83F8 10 CMP EAX,10	
0040109D   . 75 0C JNZ SHORT Leccion_.004010AB	

Vemos como en este caso saltara al MessageBoxA de MUY BIEN veamos con F9

0012FB04   00401098   CALL to MessageBoxA from Leccion_.00401093
0012FB08   004A028A   hOwner = 004A028A ('Leccion 13 by RedH@wK .: [...',class='#32770')
0012FB0C   00402028   Text = "Muy BIENiiii"
0012FB10   00000000   Title = NULL
0012FB14   00000000   Style = MB_OK!MB_APPLMODAL
0012FB18   0012FB44   RETURN to user32.77D18734
0012FB1C   77D18734
0012FB20   004A028A
0012FB24   00000111

Allí veo en el stack cuando para en la api, que va a salir el cartel bueno

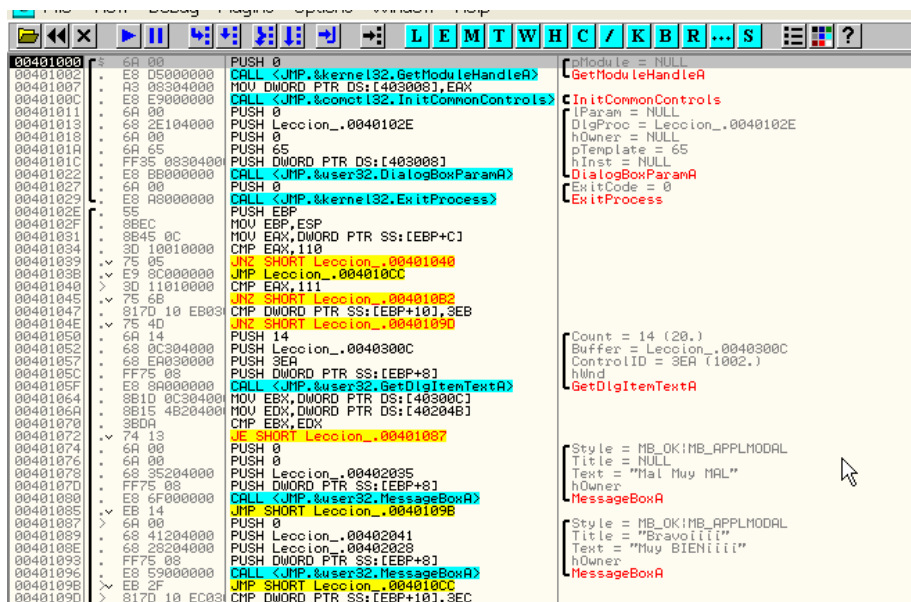




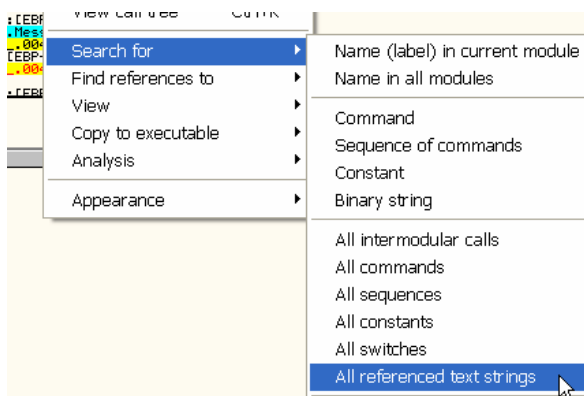
Bueno aunque olvidamos cambiarle el titulo a la ventana pero no importa hallamos el serial correcto de nuestro primer y mas sencillo hardcoded.

El siguiente es el mismo crackme que me facilito mi amigo REDHAWK, este segundo es el original que me hizo, el anterior lo modifique un poco yo jeje (por eso por vagancia no le cambie el titulo del MessageBoxA correcto jejeje)

Abramos el LECCION 13 HARDCODED 2 en el OLLYDBG



Como ven es muy similar pero en las strings no aparece la clave correcta.



Address	Disassembly	Text string
00401000	PUSH 0	(Initial CPU selection)
00401078	PUSH Leccion_.00402035	ASCII "Mal Muy MAL"

No se ve FIACA, ni ninguna otra posible por ahí jeje

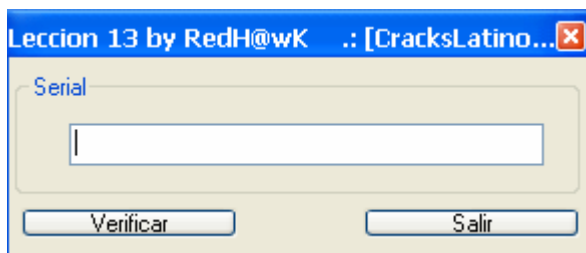
Igual en este la clave es otra, no es FIACA, jeje.

Pues como ya sabemos como es el procedimiento directamente vayamos a la parte de la comparación

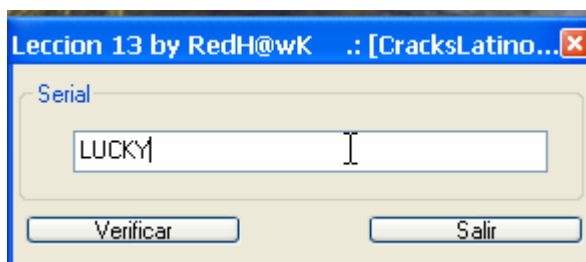
0040105F	E8 8A000000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401064	8B1D 0C304000	MOV EBX,DWORD PTR DS:[40300C]	
0040106A	8B15 4B204000	MOV EDX,DWORD PTR DS:[40204B]	
00401070	3BD8	CMP EBX,EDX	
00401072	74 13	JE SHORT Leccion_.00401087	
00401074	6A 00	PUSH 0	Style = MB_OK;MB_APPLMODAL
00401076	6A 00	PUSH 0	Title = NULL
00401078	68 35204000	PUSH Leccion_.00402035	Text = "Mal Muy MAL"
0040107D	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401080	E8 6F000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
00401085	EB 14	JMP SHORT Leccion_.0040109B	
00401087	6A 00	PUSH 0	Style = MB_OK;MB_APPLMODAL
00401089	68 41204000	PUSH Leccion_.00402041	Title = "Bravo!!!!"
0040108E	68 28204000	PUSH Leccion_.00402028	Text = "Muy BIEN!!!"
00401093	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401096	E8 59000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
0040109B	EB 2F	JMP SHORT Leccion_.004010CC	
0040109D	817D 10 EC03	CMP DWORD PTR SS:[EBP+10],3EC	

Ponemos un BPX allí en 401064 y doy RUN

00401057	68 EA000000	PUSH 3EA	ControlID = 3EA (1002.)
0040105C	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd
0040105F	E8 8A000000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401064	8B1D 0C304000	MOV EBX,DWORD PTR DS:[40300C]	
0040106A	8B15 4B204000	MOV EDX,DWORD PTR DS:[40204B]	
00401070	3BD8	CMP EBX,EDX	
00401072	74 13	JE SHORT Leccion_.00401087	
00401074	6A 00	PUSH 0	Style = MB_OK;MB_APPLMODAL
00401076	6A 00	PUSH 0	Title = NULL
00401078	68 35204000	PUSH Leccion_.00402035	Text = "Mal Muy MAL"
0040107D	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401080	E8 6F000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
00401085	EB 14	JMP SHORT Leccion_.0040109B	
00401087	6A 00	PUSH 0	Style = MB_OK;MB_APPLMODAL
00401089	68 41204000	PUSH Leccion_.00402041	Title = "Bravo!!!!"
0040108E	68 28204000	PUSH Leccion_.00402028	Text = "Muy BIEN!!!"
00401093	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401096	E8 59000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
0040109B	EB 2F	JMP SHORT Leccion_.004010CC	
0040109D	817D 10 EC03	CMP DWORD PTR SS:[EBP+10],3EC	
004010A1	75 2A	JMP SHORT Leccion_.004010B0	



Allí tipeamos un serial falso en este caso LUCKY



Apreto VERIFICAR

0040105C	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	
0040105F	. E8 8A000000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401064	. 8B1D 0C30400	MOV EBX,DWORD PTR DS:[40300C]	
0040106A	. 8B15 4B20400	MOV EDX,DWORD PTR DS:[40204B]	
00401070	. 3BD8	CMP EBX,EDX	
00401072	. 74 13	JE SHORT Leccion_.00401087	
00401074	. 6A 00	PUSH 0	
00401076	. 6A 00	PUSH 0	Style = MB_OK;MB_APPLMODAL
00401078	. 68 35204000	PUSH Leccion_.00402035	Title = NULL
0040107D	. 5275 08	PUSH Leccion_.00402035	Text = "Mal Muy MAL"

Y para en el BPX veremos que compara en este caso, esta pasando a EBX el contenido de la memoria 40300C, veámosla en el DUMP.

004010A8	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	
004010AB	. E8 38000000	CALL <JMP.&user32.EndDialog>	
004010B0	. EB 10	JMP SHORT Leccion_.00401087	
DS:[0040300C]=4B43554C			
EBX=00000000			
Address	Hex dump	ASCII	
0040300C	4C 55 43 4B 59 00 00 00	LUCKY...	
00403014	00 00 00 00 00 00 00 00	.....	
0040301C	00 00 00 00 00 00 00 00	.....	
00403024	00 00 00 00 00 00 00 00	.....	
0040302C	00 00 00 00 00 00 00 00	.....	

Allí vemos que al ejecutar la línea, moverá los 4 bytes a EBX, siempre invirtiéndolos ya vimos que cuando lee de la memoria y pasa el contenido a un registro se invertirán.

Registers (FPU)	
EAX	00000005
ECX	77D32190 user32.77D32190
EDX	7C91EB94 ntdll.KiFastSystemCallRe
EBX	4B43554C
ESP	0012FB18
EBP	0012FB18
ESI	0040102E Leccion_.0040102E
EDI	0012FB80
EIP	0040106A Leccion_.0040106A
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 1	FS 003B 32bit 7FDE000(FFF)

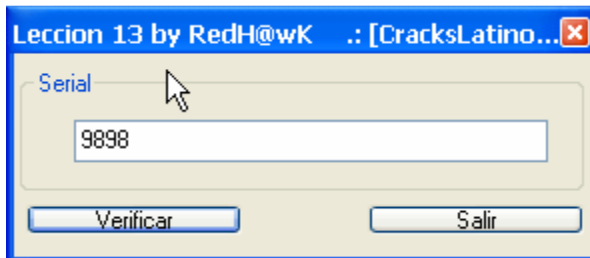
Apreto F7 y allí están en EBX , al ser un **MOV EBX, DWORD PTR DS: [40300C]** se mueven solo cuatro bytes (DWORD)

00401057	. 68 EA030000	PUSH 3EA	
0040105C	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	
0040105F	. E8 8A000000	CALL <JMP.&user32.GetDlgItemTextA>	
00401064	. 8B1D 0C30400	MOV EBX,DWORD PTR DS:[40300C]	
0040106A	. 8B15 4B20400	MOV EDX,DWORD PTR DS:[40204B]	
00401070	. 3BD8	CMP EBX,EDX	
00401072	. 74 13	JE SHORT Leccion_.00401087	
00401074	. 6A 00	PUSH 0	

En este caso vario apenas la cosa, pasara a EDX el contenido de 40204B, veamos que hay allí.

Address	Hex dump	ASCII	
0040204B	39 38 39 38 00 00 00 00	9898....	
00402053	00 AC 20 00 00 00 00 00	.%. ....	
0040205B	00 00 00 00 00 EE 20 00	.....	
00402063	00 08 20 00 00 B8 20 00	.@. .@.	
0040206B	00 00 00 00 00 00 00 00	.....	

Esta pasando los cuatro bytes que corresponden a la string 9898 que en este caso es el serial correcto ya que compara estos 4 bytes con los 4 primeros de LUCKY y si son iguales verificamos que salta a la zona de MUY BIEN, en este caso no saltara, pero si llegamos hasta poder volver a ingresar la clave y tipeamos 9898.



Apreto VERIFICAR

```

0040105C . FF75 08      PUSH DWORD PTR SS:[EBP+
0040105F . E8 8A000000  CALL <JMP.&user32.GetDi
00401064 . 8B1D 0C30400 MOV EBX,DWORD PTR DS:[4
0040106A . 8B15 4B20400 MOV EDX,DWORD PTR DS:[4
00401070 . 3BD8        CMP EBX,EDX
00401072 . 74 13       JE SHORT Leccion_.00401
00401074 . 6A 00       PUSH 0
00401076 . 6A 00       PUSH 0
00401078 . 68 35204000 PUSH Leccion_.00402035
0040107D . FF75 08      PUSH DWORD PTR SS:[EBP+
00401080 . E8 6F000000 CALL <JMP.&user32.Messa
00401085 . EB 14       JMP SHORT Leccion_.0040
00401087 . 6A 00       PUSH 0
00401089 . 68 41204000 PUSH Leccion_.00402041
0040108E . 68 28204000 PUSH Leccion_.00402028
00401093 . FF75 08      PUSH DWORD PTR SS:[EBP+
00401096 . E8 59000000 CALL <JMP.&user32.Messa
0040109B . EB 2F       JMP SHORT Leccion_.0040
0040109D . 817D 10 EC03 CMP DWORD PTR SS:[EBP+1
004010A4 . 75 26       JNZ SHORT Leccion_.0040
004010A6 . 6A 00       PUSH 0
004010A8 . FF75 08      PUSH DWORD PTR SS:[EBP+
004010AB . E8 38000000 CALL <JMP.&user32.EndDi
004010B0 . EB 10       JMP SHORT Leccion_.0040
EDX=38393839
EBX=38393839

```

En la comparación de EBX y EDX ambos son iguales son 38393839 que son los bytes correspondientes a la cadena 9898 que ingresamos y por supuesto saltara al cartel MUY BIEN.

```

0040105C . 68 00000000 PUSH 0
0040105F . FF75 08      PUSH DWORD PTR SS:[EBP+81
00401064 . E8 8A000000 CALL <JMP.&user32.GetDlgItemTextA>
0040106A . 8B1D 0C30400 MOV EBX,DWORD PTR DS:[40300C]
0040106A . 8B15 4B20400 MOV EDX,DWORD PTR DS:[40204B]
00401070 . 3BD8        CMP EBX,EDX
00401072 . 74 13       JE SHORT Leccion_.00401087
00401074 . 6A 00       PUSH 0
00401076 . 6A 00       PUSH 0
00401078 . 68 35204000 PUSH Leccion_.00402035
0040107D . FF75 08      PUSH DWORD PTR SS:[EBP+81
00401080 . E8 6F000000 CALL <JMP.&user32.MessageBoxA>
00401085 . EB 14       JMP SHORT Leccion_.0040109B
00401087 . 6A 00       PUSH 0
00401089 . 68 41204000 PUSH Leccion_.00402041
0040108E . 68 28204000 PUSH Leccion_.00402028
00401093 . FF75 08      PUSH DWORD PTR SS:[EBP+81
00401096 . E8 59000000 CALL <JMP.&user32.MessageBoxA>
0040109B . EB 2F       JMP SHORT Leccion_.004010CC

```

```

GetDlgItemTextA
Style = MB_OK!MB_APPLMODAL
Title = NULL
Text = "Mal Muy MAL"
hOwner
MessageBoxA
Style = MB_OK!MB_APPLMODAL
Title = "Bravo!!!!"
Text = "Muy BIEN!!!!"
hOwner
MessageBoxA

```



Vemos que en este que hizo mi amigo si sale el titulo BRAVO, jeje ya que no estaba vago como yo, jua jua, la cuestión que sacamos el serial correcto también, que en este caso es 9898.

Ahora iremos incrementando la dificultad tenemos dos hardcoded mas difíciles ya no tan directos, el ultimo no apto para cardiacos pero bueno pueden ir viendo en ellos, como se aplican las técnicas de crackeo para casos mas difíciles.

Como ustedes pensaron que yo solo iba a trabajar y ustedes nada, pues no, aquí trabajamos todos, hay un tercer crackme que harán ustedes (mielecrackme1.zip), que queda como tarea para la próxima parte 14, en la cual estará resuelto y explicado por mi, al igual que los otros dos que tenemos pendientes, lo que si les aconsejo hacerlo pues la parte 14 vendrá en un rar con password y el password para abrirlo, será el serial correcto de este crackme, jeje así que deben trabajar si o si (Que malvado jeje)

Como ayuda les digo que la api `strcmpA` que utiliza este crackme, es una api que se utiliza directamente para comparar strings, cuando lleguen a esa api, verán en los parámetros en el stack, las dos strings comparadas por la misma.

En la parte 14 les mostrare bien como funciona y seguiremos con los dos hardcoded, mas difíciles, pero antes deben practicar con algo sencillo, por eso el crackme que agregue aquí y que es obligatorio para seguir adelante jeje. (malo, malo eres jeje)

Hasta la parte 14  
Ricardo Narvaja  
03 de diciembre de 2005