INTRODUCCION AL CRACKING CON OLLYDBG PARTE 13

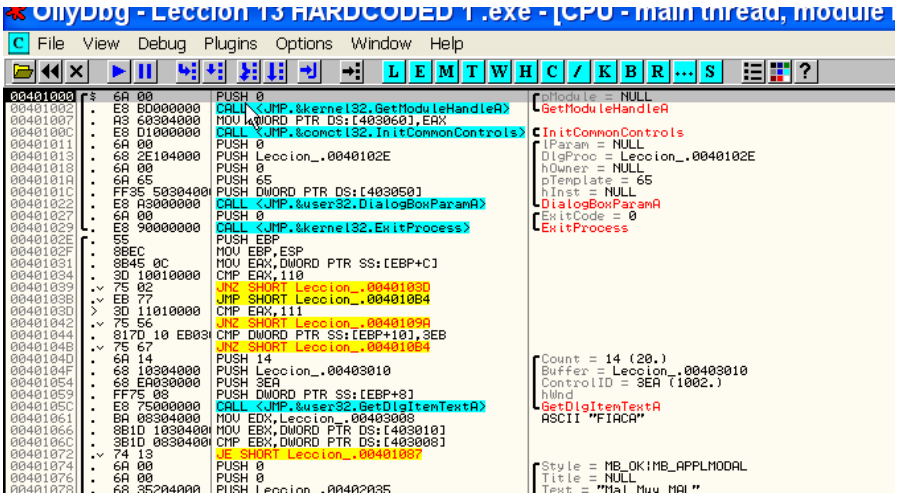
TRABAJANDO PESCANDO Y REVIRTIENDO SERIALES

Creo que una de los temas mas candentes de hoy en día es el tema de hallar seriales, sobre todo luego de que existen fuertes encriptaciones que pueden llegar a complicar la cosa, igual iremos paso a paso desde el caso mas simple al mas complejo, tratando de practicar bastante para poder afianzar los conceptos de trabajar con seriales.

Bueno esta primera parte del trabajo con seriales lo dedicaremos a los HARDCODED, que son los seriales que no son calculados a partir de un nombre, o sea es un serial fijo que puede ser un texto, un numero pero es siempre el mismo y muchas veces son sencillos otras no tanto, se tiene el concepto que son los mas fáciles de hallar jeje, por eso empezamos por este tipo, pero hay de todo fáciles y mas difíciles ya verán los ejemplos en esta parte.

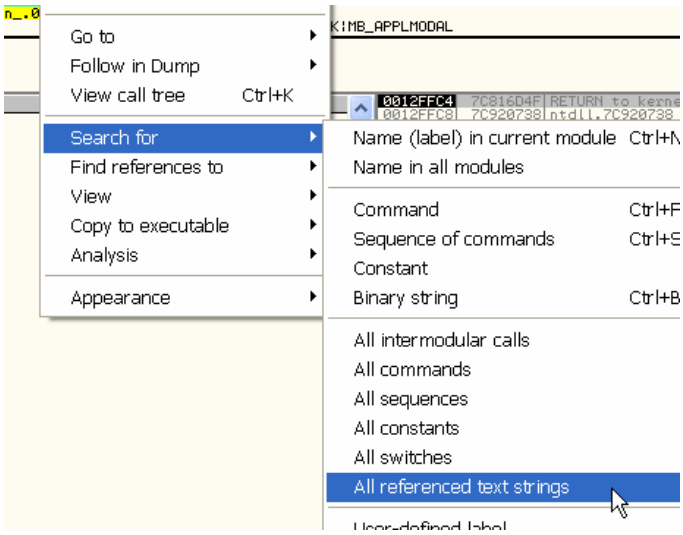
Tenemos cuatro ejemplos para practicar, dos haremos en esta parte 13 y los dos mas difíciles en la parte 14.

El primero de los cuatro que veremos es el llamado LECCION 13 HARDCODED 1, que se encuentra adjunto con la teoría es el caso mas simple, abrámoslo en OLLYDBG.

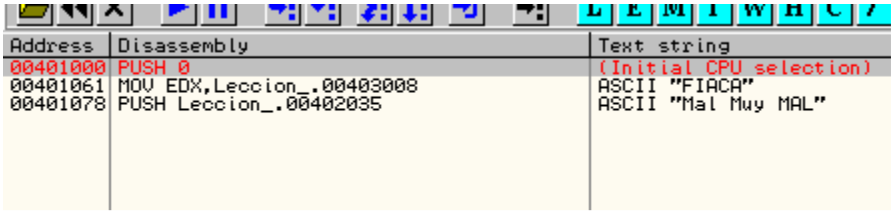


Allí esta en el inicio, el caso mas simple de los HARDCODED es el que aparece la clave correcta en las STRINGS o CADENAS DE TEXTO que utiliza el programa, miremos si aquí aparece.

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



Los resultados son

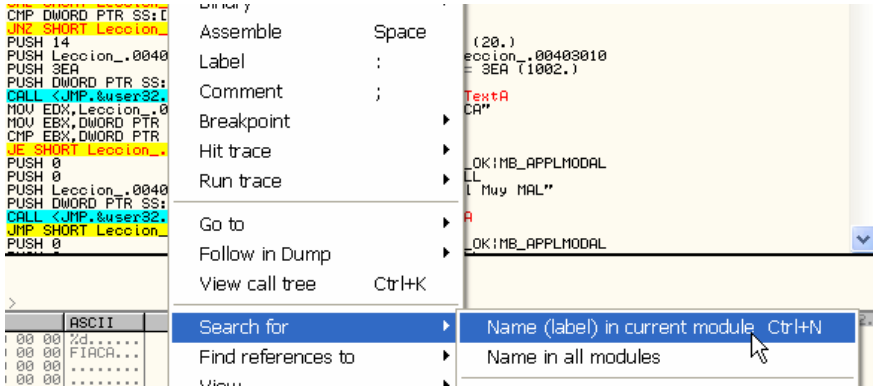


Pues allí se ve la palabra FIACA que puede ser la clave, si no tenemos ganas de trabajar podemos probar algunas strings que salen en la lista y por ahí acertamos, pero este método no es aconsejable, por las siguientes causas:

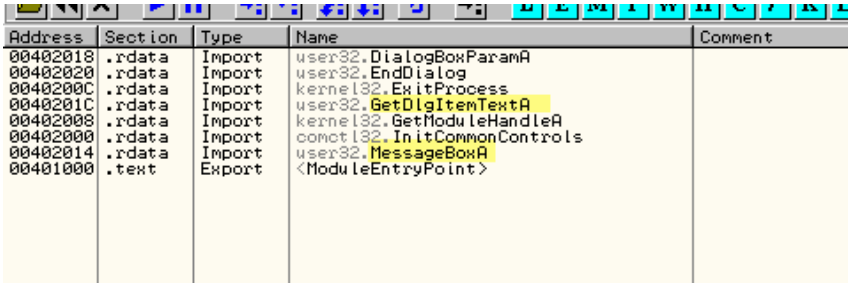
1)En este caso hay 2 strings, pero hay programas que tienen miles de strings, y probar una por una puede llevar a la locura total, así que aunque vemos allí el posible serial correcto, haremos como que hay mil strings y que no sabemos cual es.

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.

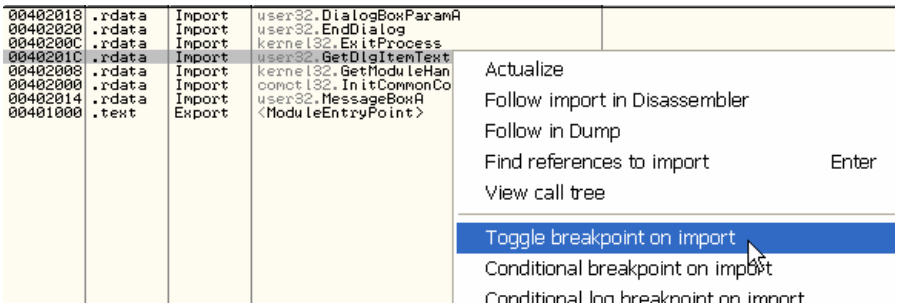
Lo primero es mirar las apis que utiliza con click derecho-SEARCH FOR- NAME (LABEL) IN CURRENT MODULE



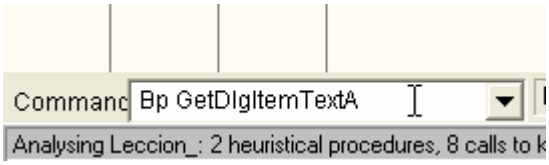
Ahí salen las apis utilizadas por el programa, veamos si hay alguna conocida

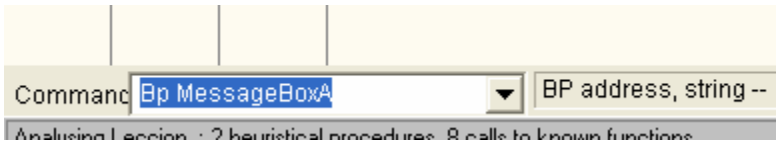


Allí vemos que utiliza GetDlgItemTextA, seguro la usara para ingresar el serial que nosotros tipeamos y MessageBoxA para sacar los mensajes de si es correcto o incorrecto nuestro serial, pongamos un BPX en cada una de dichas apis.

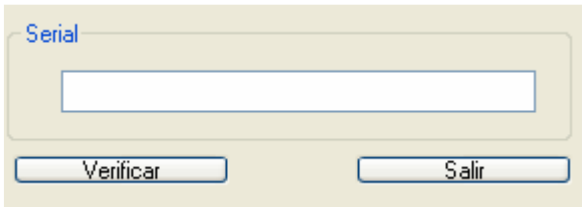


En ambas click derecho – TOGGLE BREAKPOINT ON IMPORT o en la commandbar

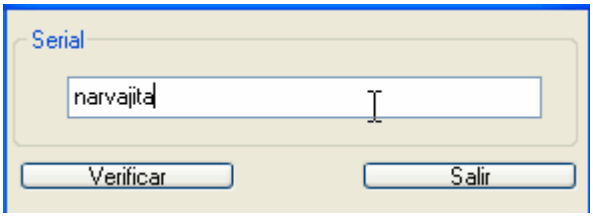




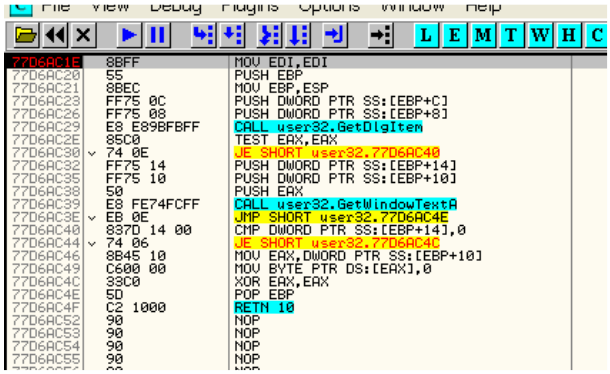
Bueno pulsemos F9 para correr el crackme



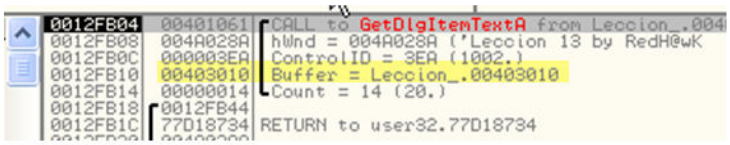
Allí sale la ventana para tipear el serial, pongamos un nombre cualquiera por ejemplo narvajita jeje



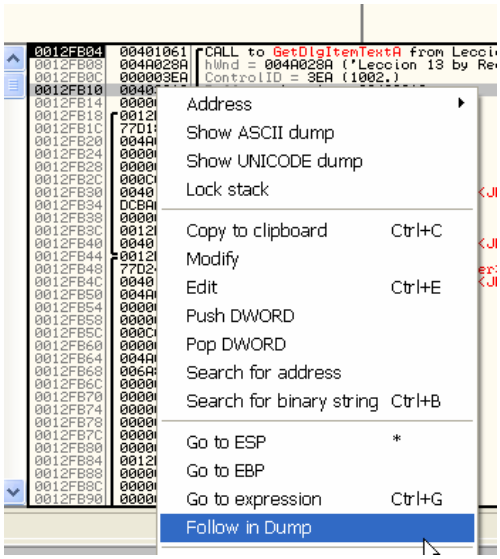
Apretemos Verificar y vemos que para en el OLLYDBG en uno de los BPX que colocamos



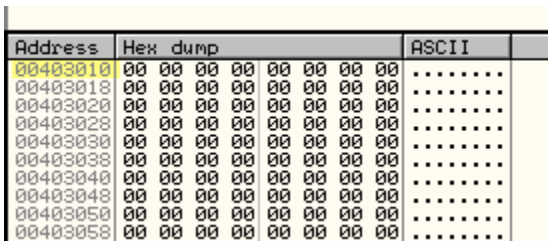
Si miramos el stack, vemos que paro en la api GetDlgItemTextA para ingresar el serial, allí en los parámetros de la api, vemos la dirección del BUFFER donde guardara el serial que ingresa, en este caso es 403010.



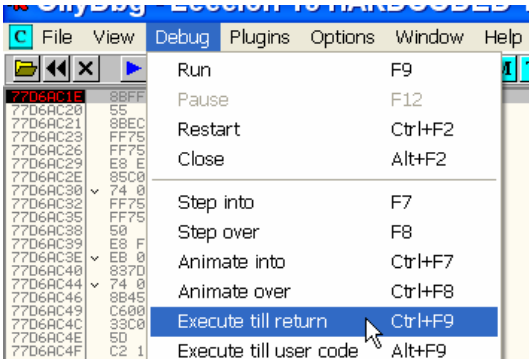
Así que veamos esa dirección en el DUMP haciendo click derecho – FOLLOW IN DUMP en la misma



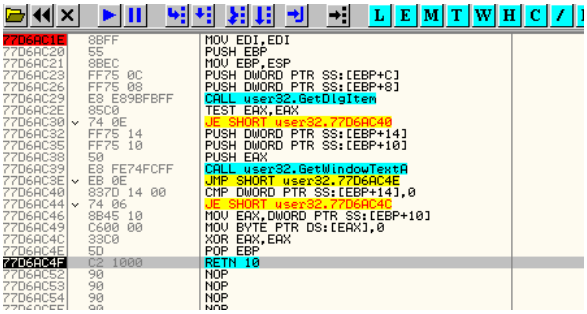
Allí esta visible el BUFFER, y esta vacío pues aun no se ha ejecutado la api



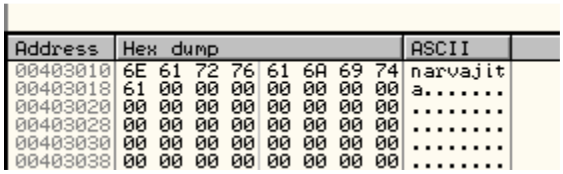
Hagamos EXECUTE TILL RETURN para que llegue al RET de la misma.



Ahora estamos en el RET

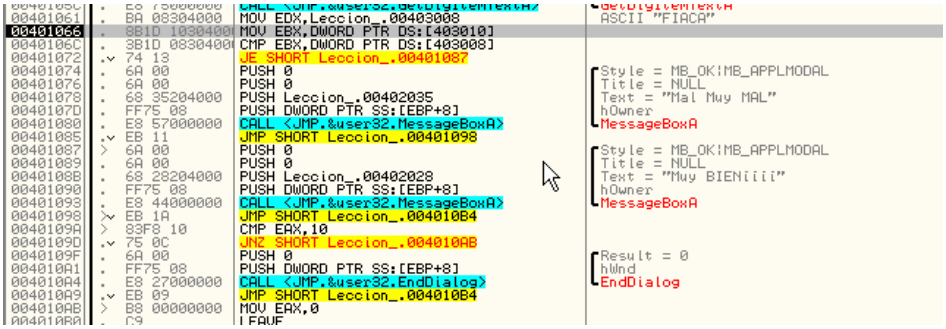


Y la api guardo en el BUFFER el serial que ingrese



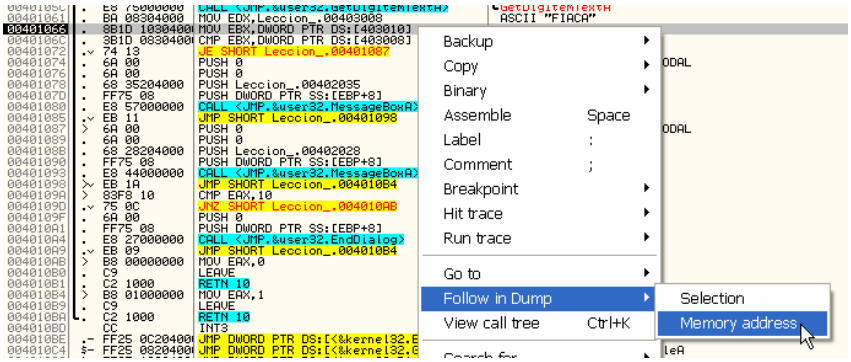
Pues allí lo tengo

Si apreto F7 y vuelvo al programa

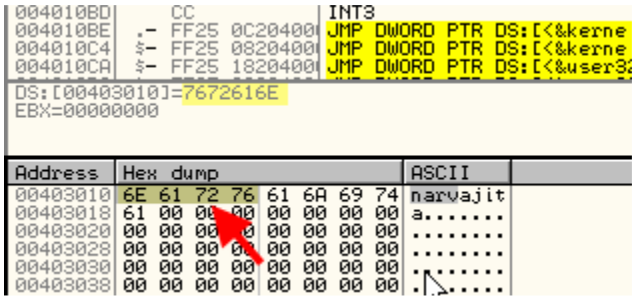


Ya vemos viendo que estamos llegando a la comparación y al salto condicional que según salte o no me llevara a los MessageBoxA de “Mal Muy MAL” o “Muy BIEN” obviamente si ese salto condicional lo cambio por un JMP 401087, lo obligo siempre a saltar al cartel de MUY BIEN y programa parcheado y resuelto,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

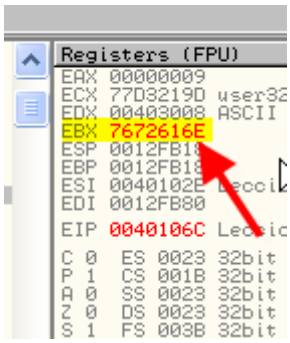
En 401066 moverá a EBX el contenido de la memoria 403010 como es un DWORD solo moverá los 4 bytes pero es suficiente, veamos que hay en 403010 en el dump.



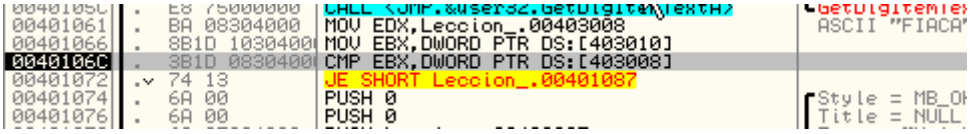
Marco la linea y hago click derecho FOLLOW IN DUMP-MEMORY ADDRESS y vere que esta leyendo



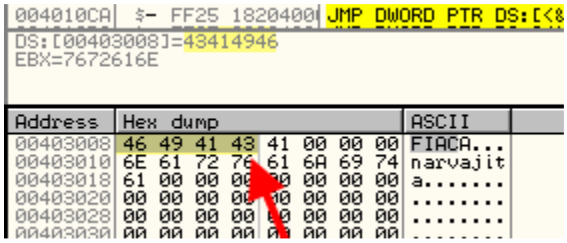
En la aclaración veo que esta leyendo 7672616E que son los 4 bytes que se encuentran en el contenido de 403010 leídos al revés como vimos, y son los primeros 4 bytes del serial falso que yo tipee los cuales mueve a EBX.



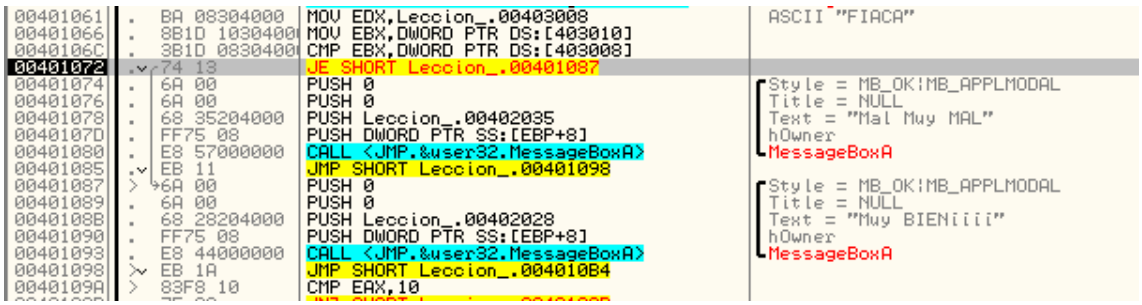
Ahora apreto F7 y paso a la siguiente línea



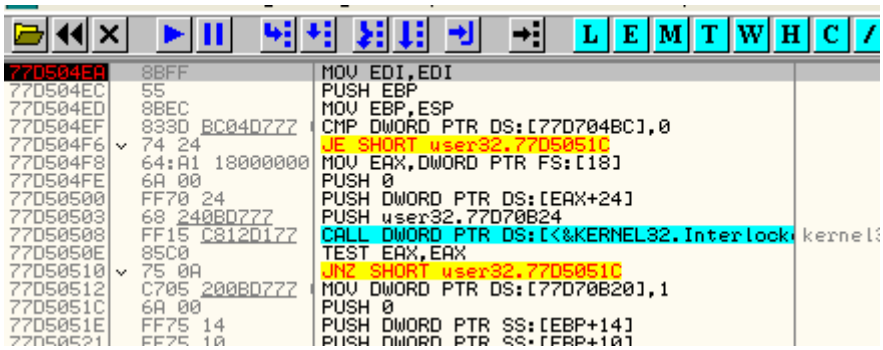
Aquí vemos que compara EBX que como recordaran son los primeros 4 bytes de mi serial falso, con el contenido de la dirección 403008, veamos que hay allí en el dump, de la misma forma que hicimos en la línea anterior.



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,



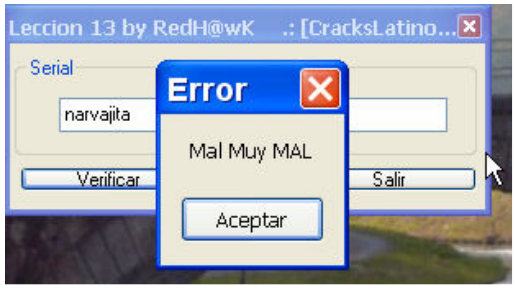
Como no fueron iguales no salta y va directo al cartel MAL MUY MAL, pues apretemos F9



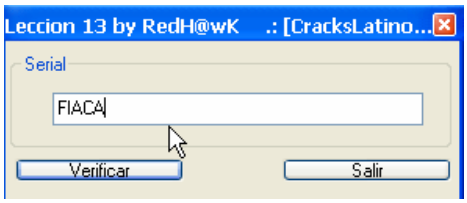
Nos para en la otra api MessageBoxA

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 1030400 MOV EBX,DWORD PTR DS:[403010]
0040106C  .  3B1D 0830400 CMP EBX,DWORD PTR DS:[403008]
00401072  .v 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(FFFFFFF
P 1  CS 001B 32bit 0(FFFFFFF
A 0  SS 0023 32bit 0(FFFFFFF
Z 1  DS 0023 32bit 0(FFFFFFF
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
```

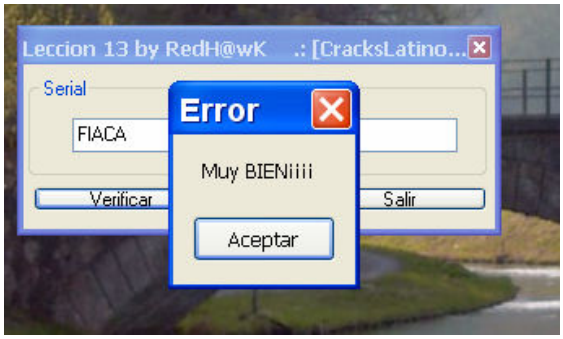
Allí lo vemos y como el JE salta al estar el FLAG Z activo.

```
00401061  .  BA 08304000  MOV EDX,Leccion_.00403008          ASCII "FIACA"
00401066  .  8B1D 1030400 MOV EBX,DWORD PTR DS:[403010]
0040106C  .  3B1D 0830400 CMP EBX,DWORD PTR DS:[403008]
00401072  .v 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  .v 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  >v EB 1A        JMP SHORT Leccion_.004010B4
0040109A  >  83F8 10      CMP EAX,10
0040109D  .v 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
0012FB1C  77D18734  RETURN to user32.77D18734
0012FB20  004A028A
0012FB24  00000111
```

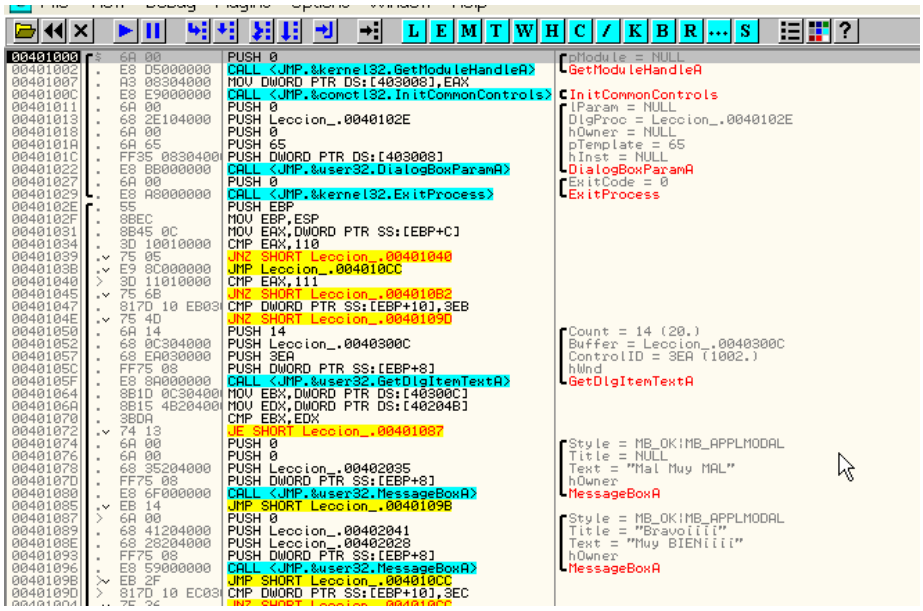
Allí veo en el stack cuando para en la api, que va a salir el cartel bueno



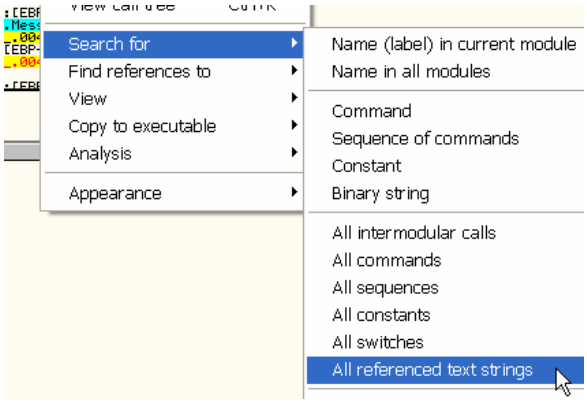
Bueno aunque olvidamos cambiarle el titulo a la ventana pero no importa hallamos el serial correcto de nuestro primer y mas sencillo hardcoded.

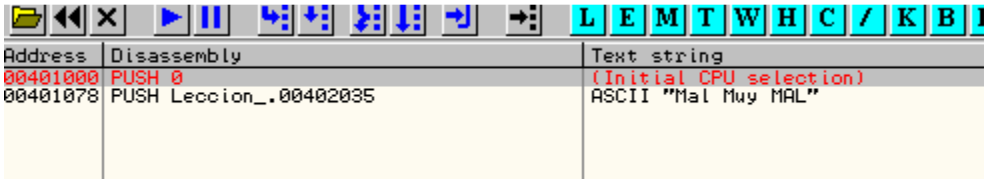
El siguiente es el mismo crackme que me facilito mi amigo REDHAWK, este segundo es el original que me hizo, el anterior lo modifique un poco yo jeje (por eso por vagancia no le cambie el titulo del MessageBoxA correcto jejeje)

Abramos el LECCION 13 HARDCODED 2 en el OLLYDBG



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

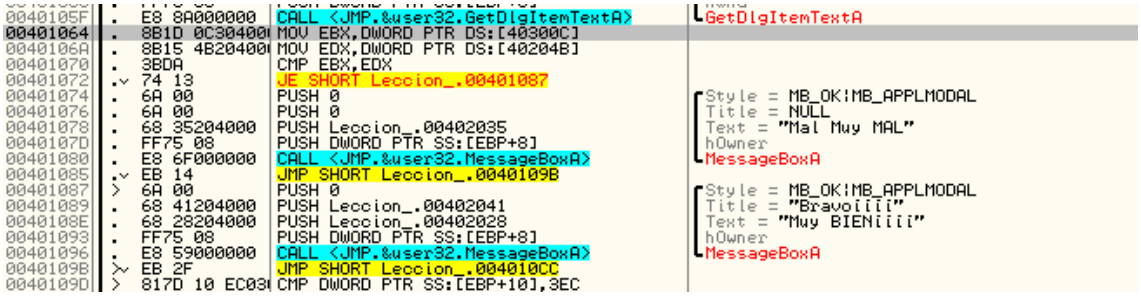




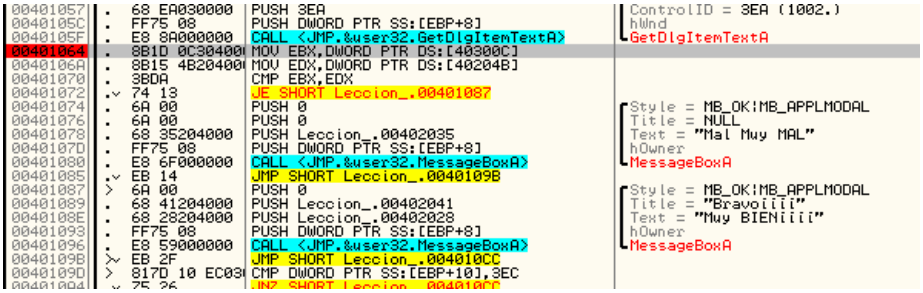
No se ve FIACA, ni ninguna otra posible por ahí jeje

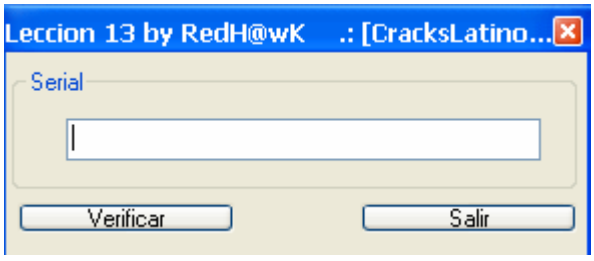
Igual en este la clave es otra, no es FIACA, jeje.

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

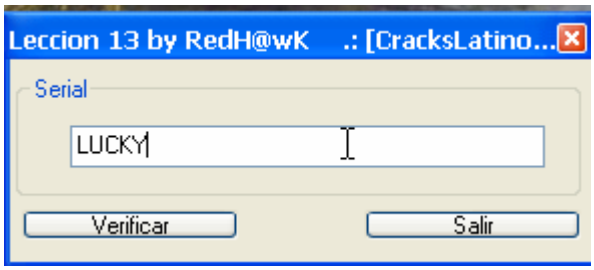


Ponemos un BPX allí en 401064 y doy RUN

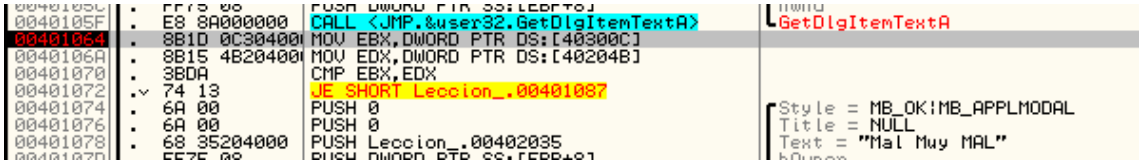




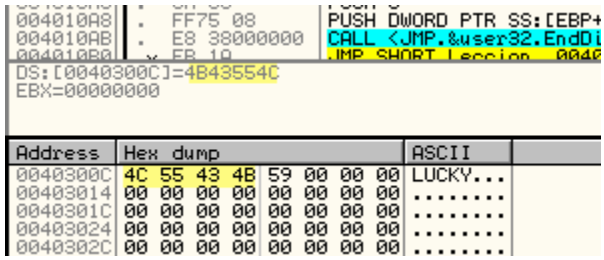
Allí tipeamos un serial falso en este caso LUCKY



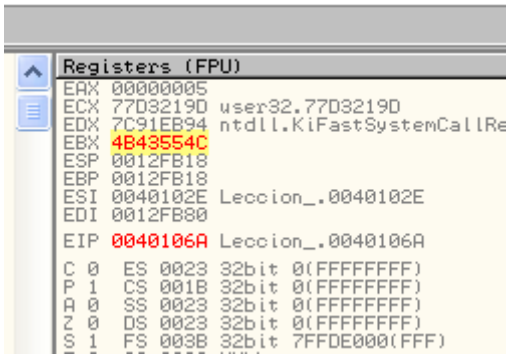
Apreto VERIFICAR



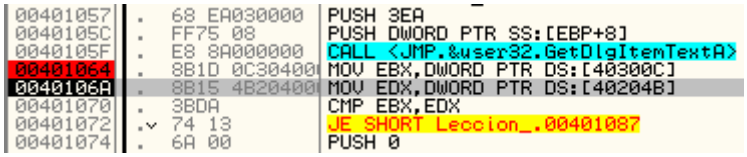
Y para en el BPX veremos que compara en este caso, esta pasando a EBX el contenido de la memoria 40300C, veámosla en el DUMP.



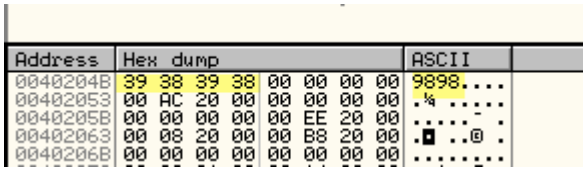
Allí vemos que al ejecutar la línea, moverá los 4 bytes a EBX, siempre invirtiéndolos ya vimos que cuando lee de la memoria y pasa el contenido a un registro se invertirán.



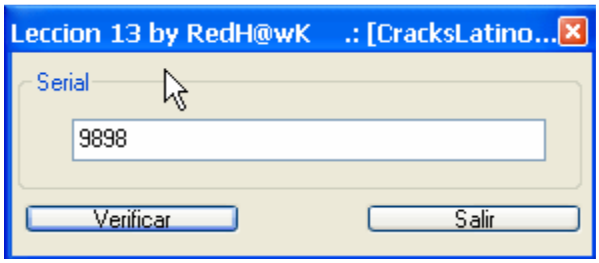
Apreto F7 y allí están en EBX , al ser un **MOV EBX, DWORD PTR DS: [40300C]** se mueven solo cuatro bytes (DWORD)



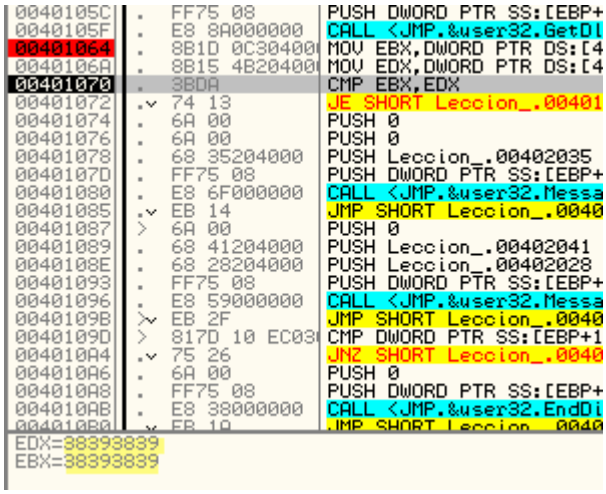
En este caso vario apenas la cosa, pasara a EDX el contenido de 40204B, veamos que hay allí.



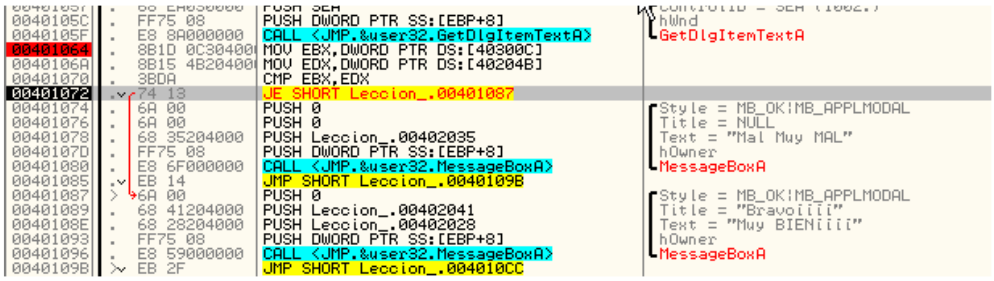
Esta pasando los cuatro bytes que corresponden a la string 9898 que en este caso es el serial correcto ya que compara estos 4 bytes con los 4 primeros de LUCKY y si son iguales verificamos que salta a la zona de MUY BIEN, en este caso no saltara, pero si llegamos hasta poder volver a ingresar la clave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



Apreto VERIFICAR



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.



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 lstrcmpA 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