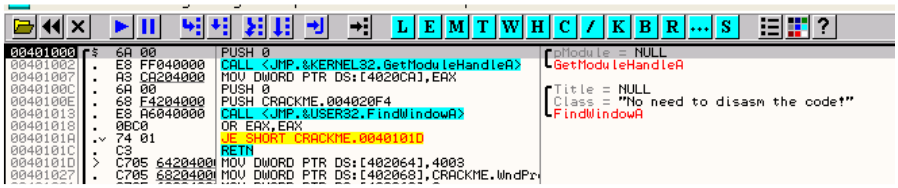
INTRODUCCIÓN AL CRACKING CON OLLYDBG PARTE 16

Comenzaremos en esta parte con los crackmes que a diferencia de los hardcoded como hasta ahora, el serial es variable y depende del nombre que ingresamos, y es calculado a partir de el.

El mecanismo para solucionarlos es muy similar, pero veremos algunos ejemplos para que quede claro.

En la parte anterior los anime a que intenten hallar para su propio nombre, el serial correcto para el CRACKME DE CRUEHEAD y bueno, ese será el primero que haremos, así ven como se soluciona y si estaban en el buen camino, y si lo solucionaron pues mis mas gratas felicitaciones.

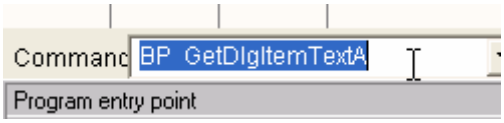
Abramos el crackme de cruehead en OLLYDBG.



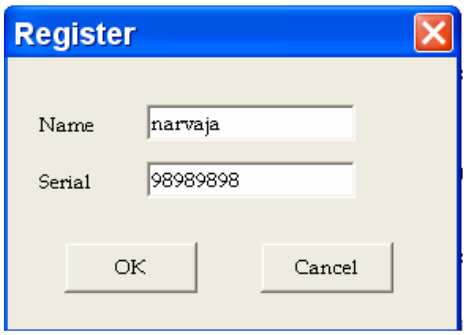
Allí estamos detenidos en el ENTRY POINT

Veamos las apis que utiliza para ingresar el texto del serial que tipeamos

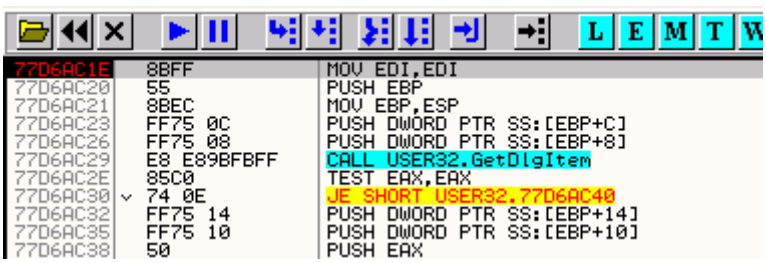
Probamos colocando un BP en dicha api a ver si se detiene cuando ingresa el nombre y serial correcto

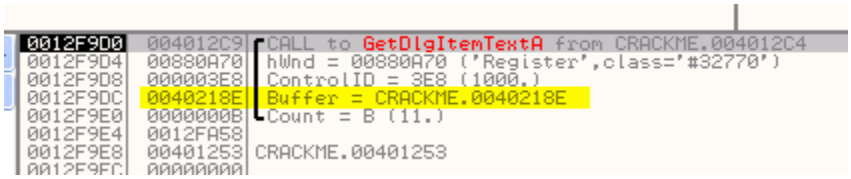


Damos RUN

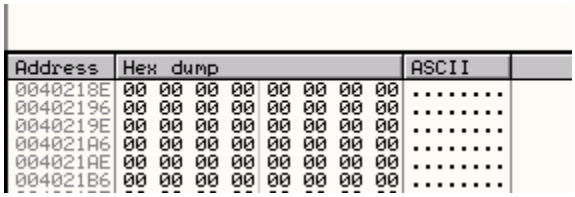


Al apretar OK para en la api y en el stack vemos los parámetros

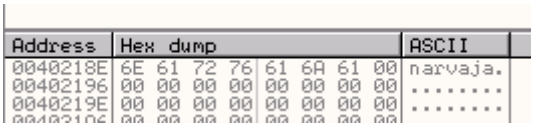




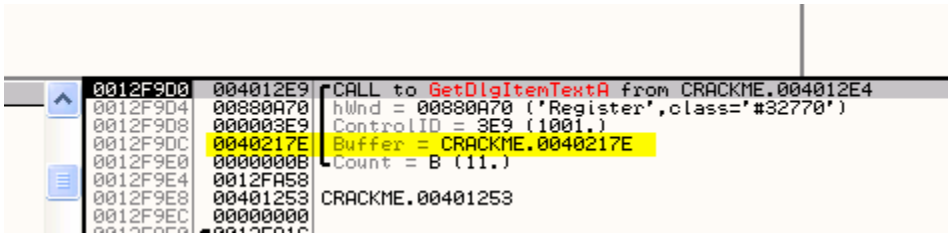
El buffer donde guardara el texto que ingresa esta en 40218E, veamos esa dirección en el dump, hacemos click derecho - FOLLOW IN DUMP en la línea del stack que nos informa del buffer.



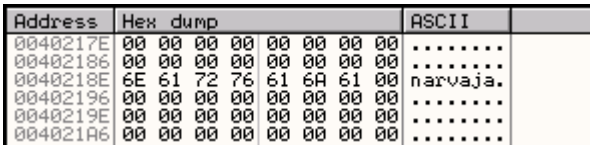
Allí esta, aun vacío porque no se ejecuto la api, así que hagamos DEBUG- EXECUTE TILL RETURN y como estamos en el RET de la api hago F7 para volver al programa.



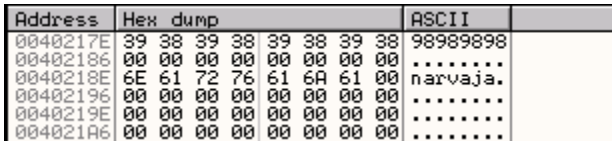
Vemos que en el buffer guardo mi nombre a partir del cual realizara operaciones y creara la clave correcta para el mismo, si quisiéramos hacer un keygen, deberíamos estudiar a partir de aquí las operaciones que realiza el programa para desde un nombre cualquiera, calcular la clave correcta, pero como este no es el caso, dejaremos que el programa genere la clave correcta y trataremos de ver si compara con el serial que tipeamos damos RUN nuevamente.



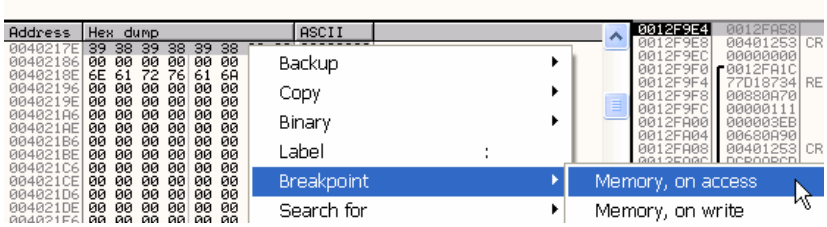
Para por segunda vez en la api, el nuevo BUFFER para la entrada del serial que tipee es 40217E, así que lo busco en el dump.



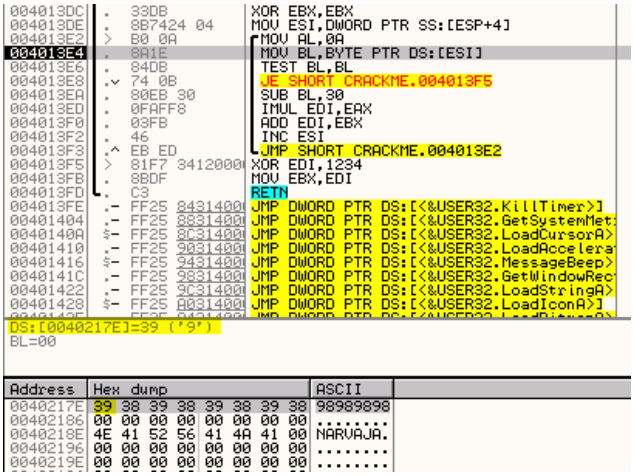
Allí esta justo arriba del otro, ejecuto la api con EXECUTE TILL RETURN y para volver al programa desde el RET apreto F7.



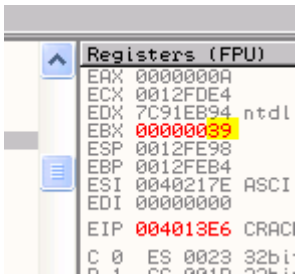
Allí ingreso nuestro serial falso, a partir de este punto el programa comparara el serial falso con el correcto que calculo a partir de mi nombre, así que podemos colocar un BPM ON ACCESS en este serial falso para ver que hace el programa con el.



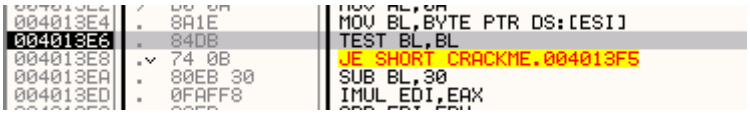
Marco los bytes y hago click derecho – BREAKPOINT –MEMORY ON ACCESS y doy RUN



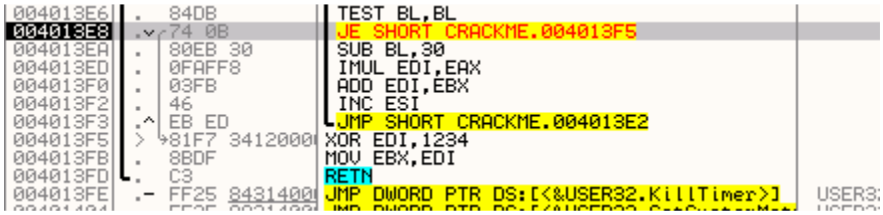
Vemos que para allí cuando el programa lee el primer byte de mi serial falso y lo mueve a BL si apreto F7.



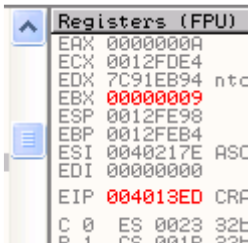
Allí esta en BL el valor 39 de mi primer byte, debemos perseguir lo que el programa realiza con el, y en lo posible ir anotando para no perdernos si realiza operaciones matemáticas con el mismo.



Allí testea si es cero, para salir del loop al encontrar el final de la string, sigamos con f7

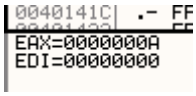


Como no es cero no salta y llega a SUB BL,30

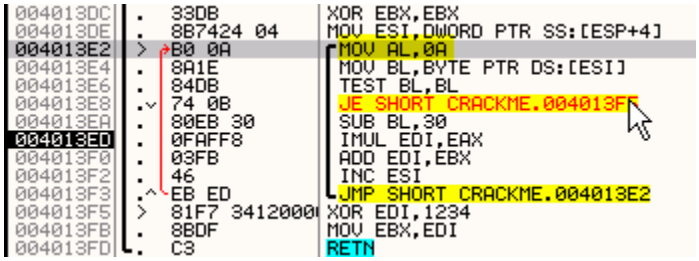


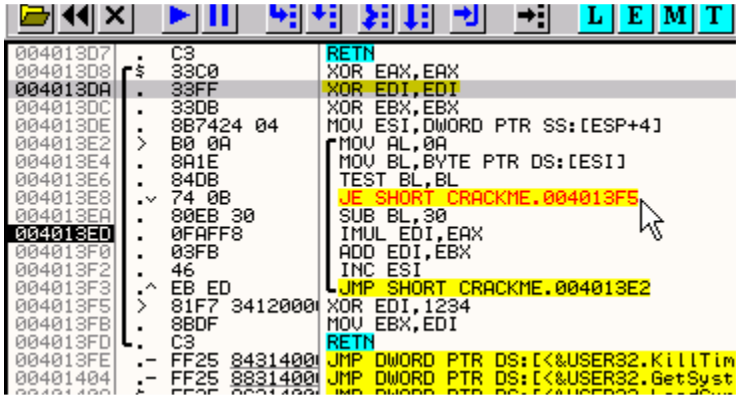
Al restarle 30, queda en BL el valor numérico decimal, del primer carácter de nuestro serial falso que es 9.

En la siguiente línea multiplica EDI con EAX.

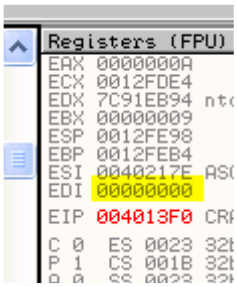


Dichos valores estaban inicializados EAX en 0A justo al inicio de la rutina y EDI antes de entrar al loop se coloco a cero con XOR EDI,EDI





Allí esta de donde surgen ambos valores, al apretar f7 si recordamos la definición de IMUL con dos operandos, en dicho caso multiplica ambos considerando el signo, y guarda el resultado en el primer operando o sea EDI en este caso.



Por eso guarda el resultado en EDI el cual queda a cero.

```
004013EA  .  80EB 30        SUB BL,30
004013ED  .  0FAFF8         IMUL EDI,EAX
004013F0  .  03FB           ADD EDI,EBX
004013F2  .  46             INC ESI
004013F3  .^ EB ED          JMP SHORT CRACKME.004013E2
004013F5  >  81F7 3412000(  XOR EDI,1234
```

Luego le suma a EDI el valor de EBX

```
Registers (FPU)
EAX 0000000A
ECX 0012FDE4
EDX 7C91EB94 ntdll.K
EBX 00000009
ESP 0012FE98
EBP 0012FEB4
ESI 0040217E ASCII "
EDI 00000009
EIP 004013F2 CRACKME
C 0  ES 0023 32bit 0
P 1  CS 001B 32bit 0
```

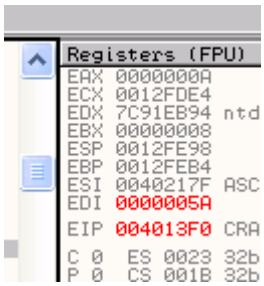
Quedando el EDI el valor 9, a no desesperar que esto parece difícil pero ya verán que no lo es en la próxima línea INC ESI, incrementa ESI para retornar al inicio del LOOP con el JMP y poder leer el siguiente byte de mi serial falso.

```
004013DE  .  8B7424 04      MOV ESI,DWORD PTR SS:[ESP+4]
004013E2  >  B0 0A          MOV AL,0A
004013E4  .  8A1E           MOV BL,BYTE PTR DS:[ESI]
004013E6  .  84DB           TEST BL,BL
004013E8  .v 74 0B          JE SHORT CRACKME.004013F5
004013EA  .  80EB 30        SUB BL,30
004013ED  .  0FAFF8         IMUL EDI,EAX
004013F0  .  03FB           ADD EDI,EBX
004013F2  .  46             INC ESI
004013F3  .^ EB ED          JMP SHORT CRACKME.004013E2
004013F5  >  81F7 3412000(  XOR EDI,1234
004013FB  .  8BDF           MOV EBX,EDI
004013FD  L. C3             RETN
004013FE  .- FF25 8431400(  JMP DWORD PTR DS:[<&USER32.KillTi
00401404  .- FF25 8831400(  JMP DWORD PTR DS:[<&USER32.GetSyst
0040140A  $- FF25 8C31400(  JMP DWORD PTR DS:[<&USER32.LoadCur
00401410  .- FF25 9031400(  JMP DWORD PTR DS:[<&USER32.LoadAcc
00401416  $- FF25 9431400(  JMP DWORD PTR DS:[<&USER32.Message
0040141C  .- FF25 9831400(  JMP DWORD PTR DS:[<&USER32.GetWind
DS:[0040217F]=38 ('8')
BL=09 (TAB)
Address  Hex dump  ASCII
```

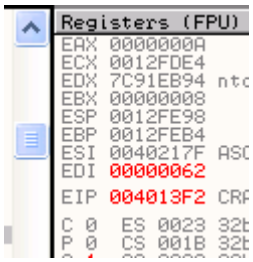
Luego de volver a colocar en AL el valor 0A, ahora lee el segundo byte de mi serial falso, ahí vemos que moverá a BL el valor 38 al igual que antes, testeara si es cero, le restara 30, y llegamos a la IMUL.

```
004013ED  .  0FAFF8         IMUL EDI,EAX
004013F0  .  03FB           ADD EDI,EBX
004013F2  .  46             INC ESI
004013F3  .^ EB ED          JMP SHORT CRACKME.004013E2
004013F5  >  81F7 3412000(  XOR EDI,1234
004013FB  .  8BDF           MOV EBX,EDI
004013FD  L. C3             RETN
004013FE  .- FF25 8431400(  JMP DWORD PTR DS:[<&USER32.Kil
00401404  .- FF25 8831400(  JMP DWORD PTR DS:[<&USER32.Get
0040140A  $- FF25 8C31400(  JMP DWORD PTR DS:[<&USER32.Loa
00401410  .- FF25 9031400(  JMP DWORD PTR DS:[<&USER32.Loa
00401416  $- FF25 9431400(  JMP DWORD PTR DS:[<&USER32.Mes
0040141C  .- FF25 9831400(  JMP DWORD PTR DS:[<&USER32.Get
EAX=0000000A
EDI=00000009
```

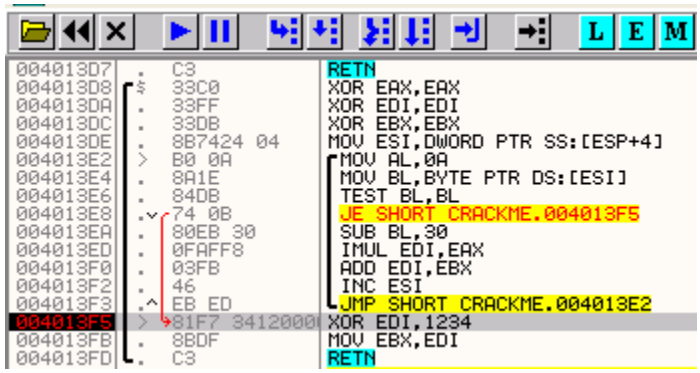
Vemos que multiplica EDI que tiene el valor que arrastra de la pasada anterior con 0A y el resultado lo guardara en EDI veamos.



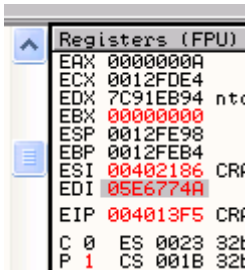
EDI queda en 5A en la siguiente línea le suma EBX



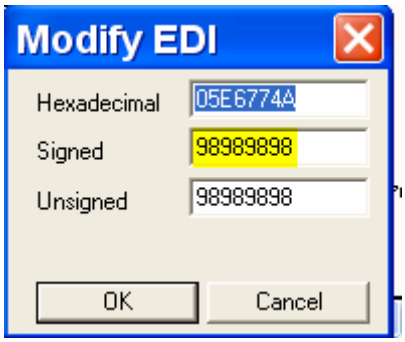
O sea que va realizando operaciones y va juntando el resultado en EDI, para los que son ya asiduos al cracking ya sabrán que hace aquí, lo explicaremos.



Lo mas sencillo es poner un BP en la salida del loop apretemos F9 y cuando se detiene veo que EDI vale



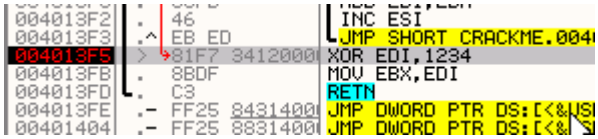
Si hago doble click en EDI



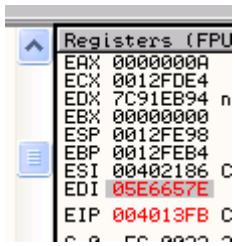
El valor de EDI es un valor hexadecimal, en la segunda línea veo su valor decimal, que corresponde a mi serial falso o sea lo que ha hecho con todo este loop (que es un loop bien conocido para los asiduos al cracking jeje ya lo ven y saben que a la salida estará el valor HEXA del serial falso.

O sea en resumidas cuentas si tipee

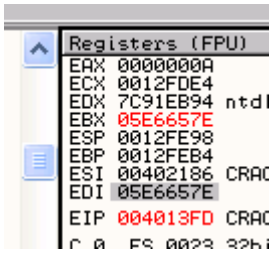
98989898 lo primero transformo esto en el número decimal 98989898 y luego lo transformo a hexa quedando 5E6774A.



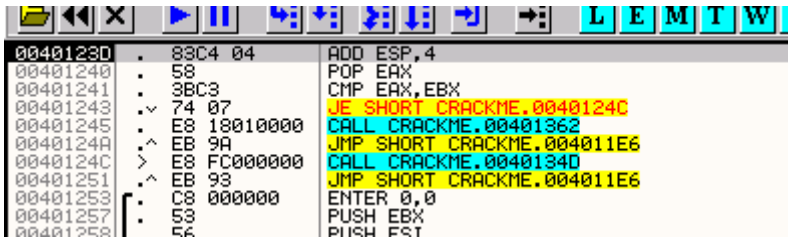
En la siguiente línea se hace EDI XOR con 1234

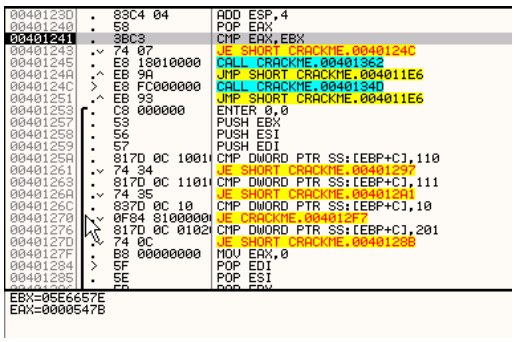


Queda en mi caso 5E6657E sigamos traceando, lo mueve a EBX y llega al RET.



Vemos que al volver del ret llega a la comparación EAX con EBX, lleguemos allí, ese era el salto que decidía si era correcto o no así que estamos bien,





Al llegar vemos que compara el valor que calculó que esta en EBX con EAX que vale 547B

Como EAX es un valor determinado del programa y EBX es calculado a partir de mi serial falso el cual es incorrecto obviamente, aquí no hay igualdad.

Si EBX fuera igual a EAX el programa saltaría a CORRECTO, por lo tanto debemos analizar como llegar a que allí haya una igualdad.

Veamos nuestras anotaciones

EBX= (Valor hexadecimal del serial falso) XOR 1234

Como yo quiero que EBX sea igual que EAX, en dicho caso se verificara la igualdad y estaremos en el caso correcto

Si EAX=EBX

Reemplazo EBX por EAX ya que son iguales

EAX=(Valor hexadecimal del serial correcto) XOR 1234

Y ya no es más serial falso porque el serial que hace que EBX y EAX sean iguales es el serial correcto.

Despejo

EAX XOR 1234= (Valor hexadecimal del serial correcto)

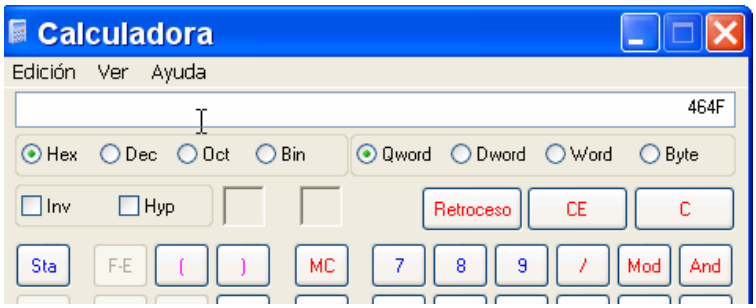
Y EAX es un dato pues lo tengo de la comparación EAX=547B

547B XOR 1234 = (Valor hexadecimal del serial correcto)

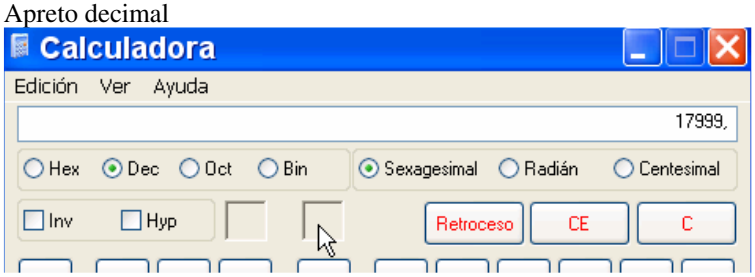
Si soluciono el XOR

464F = (Valor hexadecimal del serial correcto)

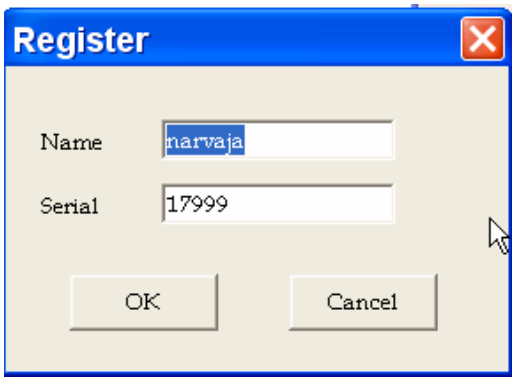
Si 464F es el valor correcto hexadecimal, si lo paso a decimal tendré mi serial para tipear



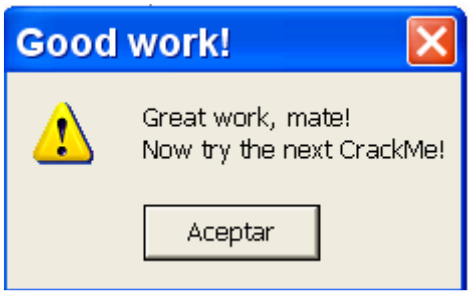
Apreto decimal



Ese es el serial correcto para narvaja quitemos todos los BPM y BPs



Apreto OK



Y obtuve el serial para mi nombre

Esta es una forma de solucionarlo, otra forma de pensarlo que se llega a la misma soluciones es esta

Si a mi serial falso le hace operaciones y lo transforma a esas operaciones las llamamos función F

F (serial falso) =EBX

O sea que a mi serial falso le realiza ciertas operaciones llamadas F y lo convierte en EBX el cual comparará.

El otro miembro de la comparación que es EAX sigue el mismo parámetro

F (serial verdadero)=EAX

Aplicando las mismas operaciones al serial verdadero en este caso llego al valor de EAX, por lo tanto si tengo EAX ya que es un dato de la comparación y tengo las operaciones realizadas.

Para hallar el serial verdadero deberé a EAX realizarle las operaciones opuestas para hallar el serial verdadero.

Serial verdadero= (funciones opuestas de F) EAX

O sea si fuera que a mi serial falso se le sumo un valor para llegar a la comparación, este se lo deberé restar a EAX y así siempre debo realizarle las mismas operaciones pero opuestas.

En este caso la operación opuesta de XOR es la misma XOR ya que es una operación inversible

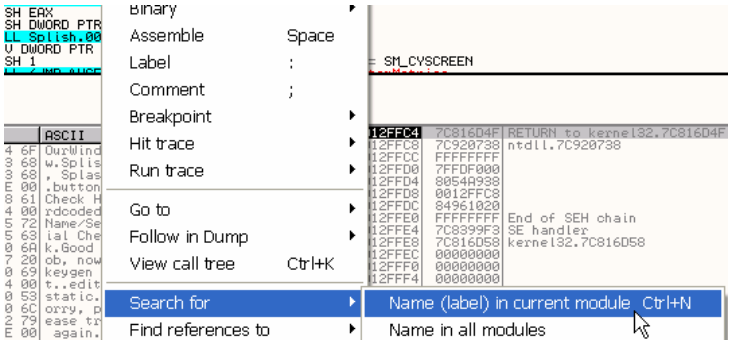
Por lo cual

XOR EAX será realizar la función opuesta de F y me dará el valor hexa de mi serial correcto, a partir del cual, pasándolo a decimal obtendré el serial verdadero.

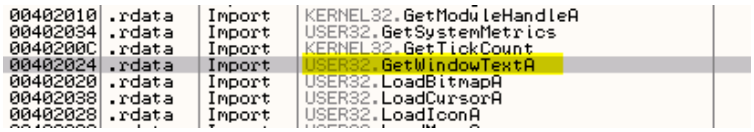
Esto que parece mucho palabrerío no lo es tanto, de ambas formas despejando o realizando las operaciones opuestas normalmente se puede llegar al serial correcto, salvo que se use una operación que no tiene opuesta, ya veremos esos casos.

Veamos otro caso el del ya visto SPLISH pero en el modo USER-NAME ya que la parte HARDCODED ya la hemos hecho.

Arrancamos el SPLISH en OLLYDBG y estamos en el ENTRY POINT



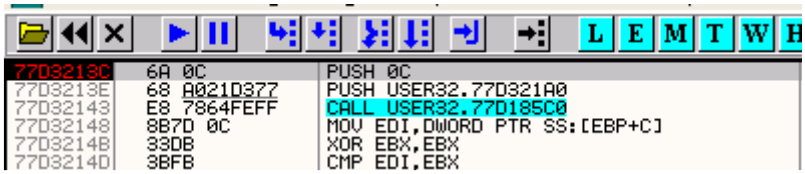
Veamos las apis que utiliza



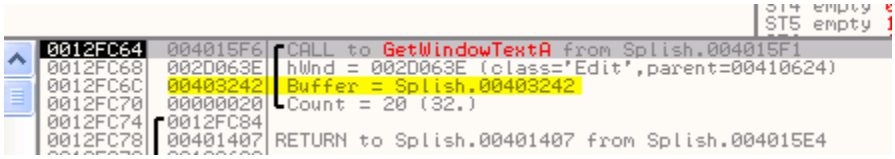
La conocida GetWindowTextA pongamos un BPX en dicha api

Y demos RUN con F9

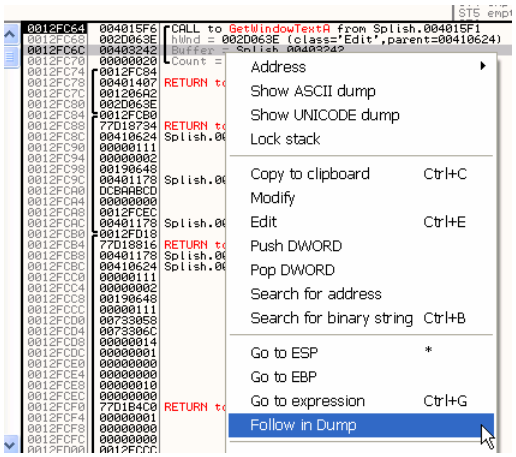
Alli tipeamos el nombre y serial falso y apretamos el boton NAME/SERIAL CHECK

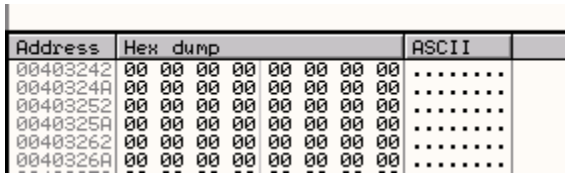


Para en la api veamos el BUFFER

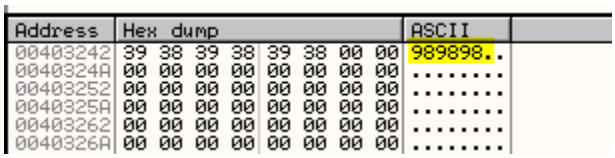


Buscamos el BUFFER en el DUMP

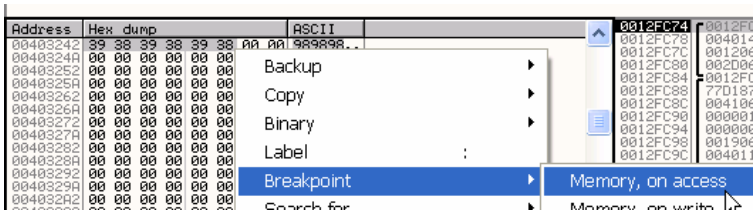




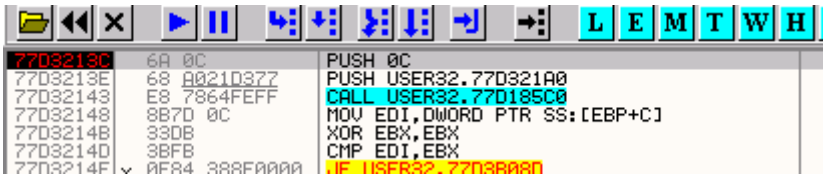
Allí está vacío aun, hagamos EXECUTE TILL RETURN para llegar al RET y luego F7 para volver al programa.



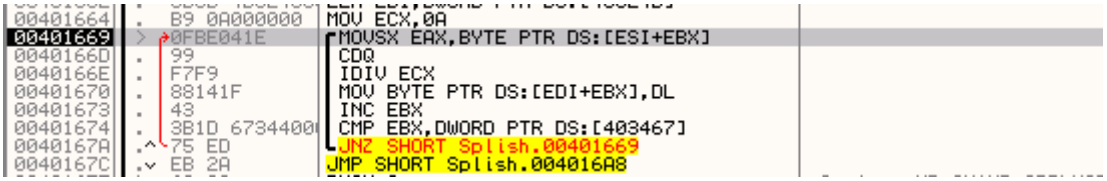
Vemos que en este caso ingreso primero el serial falso, así que marcamos el mismo y ponemos un BPM ON ACCESS.



Damos RUN

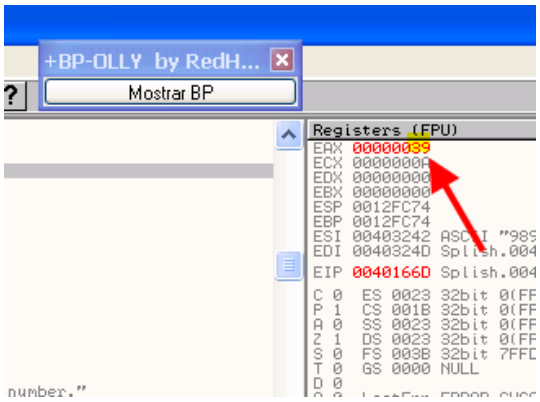


Para de nuevo en la api para ingresar el nombre, como esto por ahora no nos interesa damos RUN nuevamente.



Para en esta rutina

La primera línea mueve nuestro primer byte del serial falso si ejecutamos



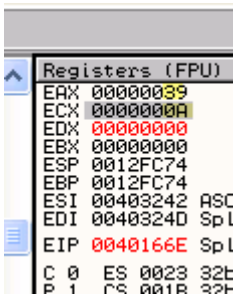
Luego tenemos la siguiente instrucción CDQ, ya que no la explicamos veamos que dice nuestro amigo GOOGLE, el puede ayudar tipeamos en el buscador de GOOGLE CDQ ENSAMBLADOR.

```
cdq
idiv esi
```

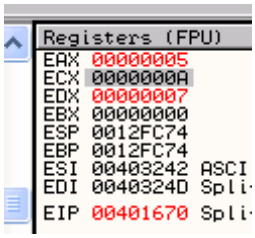
La instrucción `idiv esi` divide el contenido de EDX:EAX por ESI poniendo el resultado de la división en EAX y el resto en EDX. Para poderse efectuar esta división el contenido de EDX debe ser igual a cero, lo que puede conseguirse mediante `cdq` o `xor edx, edx`. En este caso, no sería necesario poner a cero el registro EDX, pues como el resultado de la multiplicación anterior se coloca en EDX:EAX, con los valores que manejamos en este programa se pone EDX a cero.

En este caso igual no es necesario ya que EDX es cero y esta preparado para obtener el resto de la división pero como es un loop por si acaso siempre mejor antes de una IDIV usar el comando CDQ para preparar los registros EDX y EAX para la misma.

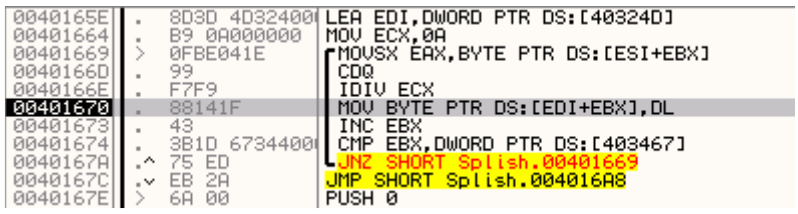
Pues dividirá EDX:EAX por ECX y el resultado va a EAX y el resto a EDX bueno este es todo el truco vemos que el byte nuestro vale 39 y lo dividirá por ECX que es 0A



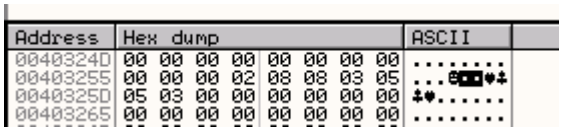
Veamos que ocurre



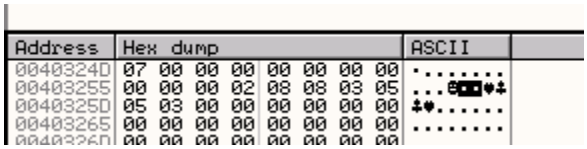
Allí esta el resultado de la división esta en EAX y es 5 y el resto esta en EDX y es 7



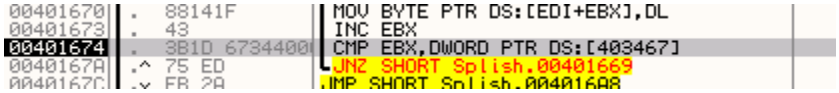
Vemos que en la próxima línea guarda el resto de la división en 40324D veámoslo en el DUMP



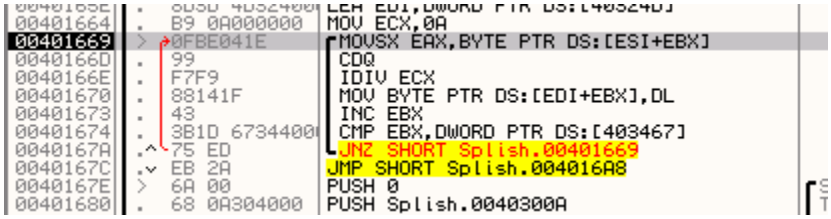
Allí lo guardara ejecuto con f7



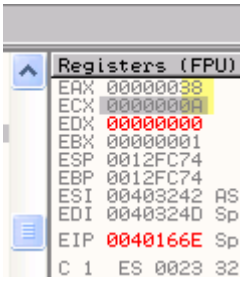
Allí esta el 7 guardado



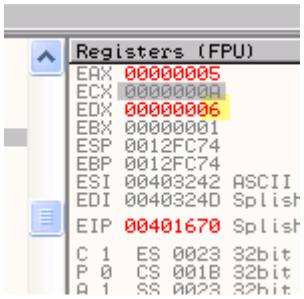
Vemos que EBX que era cero es incrementado y comparado con 6 para salir del loop por ahora no son iguales por lo tanto repite.

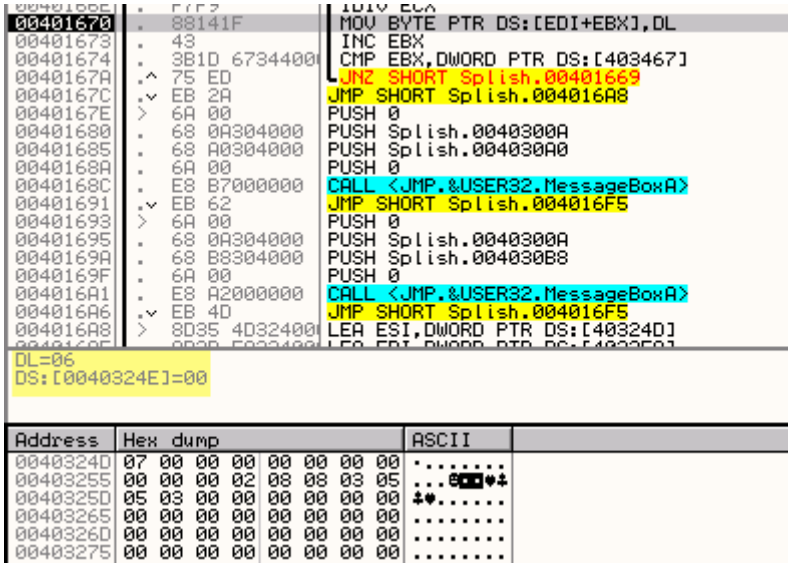


Allí va leer el segundo byte

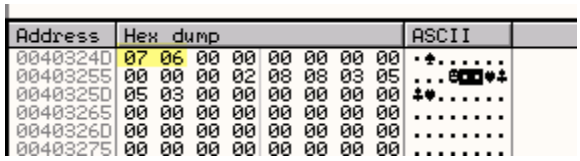


De nuevo lo divide por 0a y guarda el resto

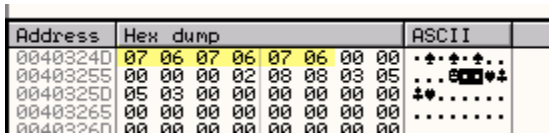




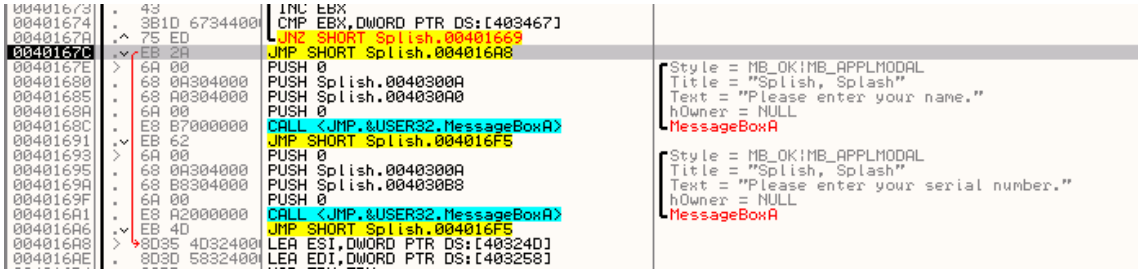
A continuación del otro byte lo guardara



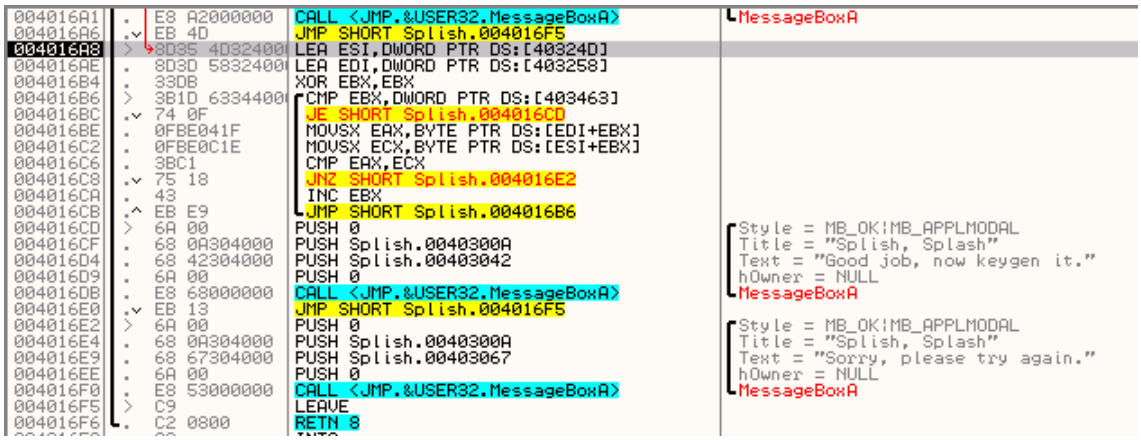
Bueno vemos que no hay variación realiza la misma operación en todos los bytes.



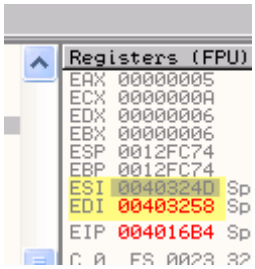
Y sale del loop al no saltar el JNZ que nos retornaba al inicio del mismo.



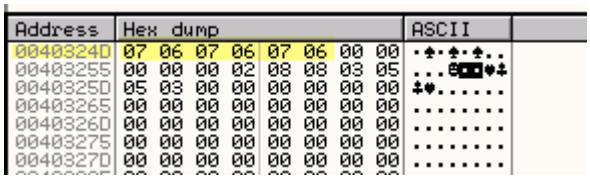
Llegamos allí y vemos los mensajes de correcto o incorrecto por lo cual ya sabemos que estamos cerca.



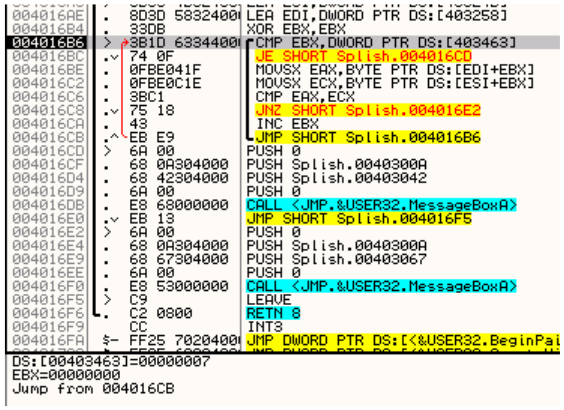
Luego hay dos LEA que mueven direcciones a ESI y EDI



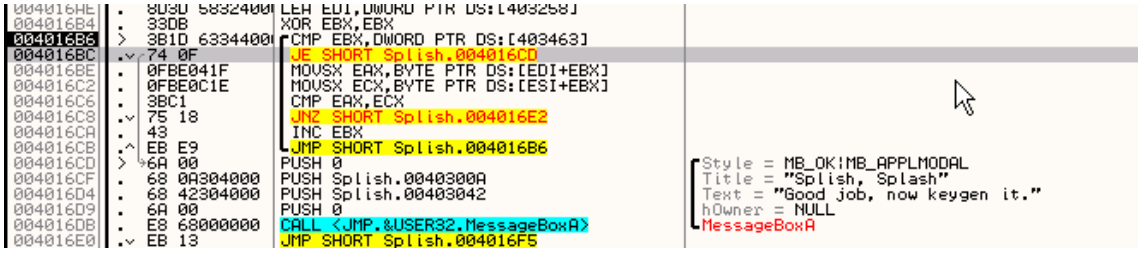
ESI apunta a donde guardo los restos y EDI apunta a algo interesante jeje?



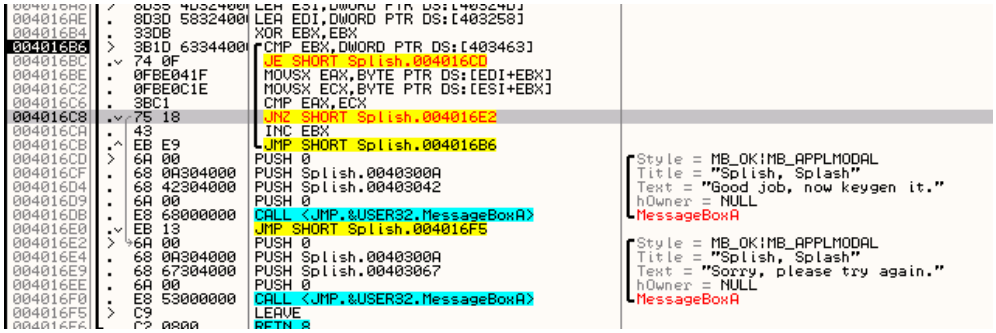
Sigamos



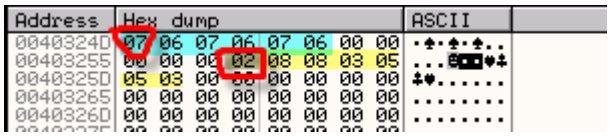
Vemos que EBX vale cero y lo compara con 7 y si es igual



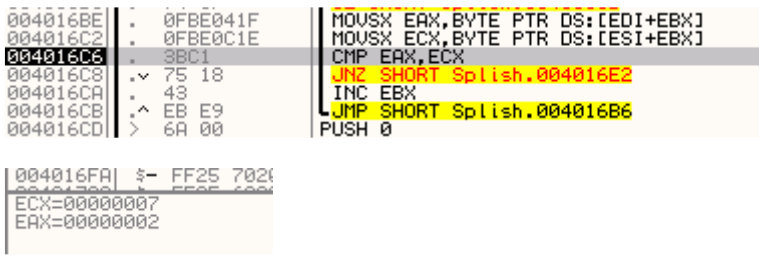
Saltaría al cartel correcto lo que pasa que hay un loop y en el medio otro JNZ de las comparaciones que si una sola falla nos llevara antes al cartel de incorrecto.



Bueno veamos que compara



A EAX moverá el primer byte que apunta EDI que es 02 y a ECX mueve el primer byte de mis restos o sea 7.



Y si vemos que si el resto del primer byte hubiera sido 02 la comparación seria exitosa.

En mi caso para hallar el resto

39= 5 .0A + 7

Ya que el resultado de 39 dividido 0a daba 5 con resto 7, eso quiere decir que al multiplicar 0a por 5 me da 32 y al sumarle 7 me da 39 lo cual se ve reflejado en esa formula

39= 5 x 0A + 7

Como el resto correcto debería ser dos para el caso del serial correcto

BYTE CORRECTO= 5 x 0A + 2

BYTE CORRECTO= 32 +2 = 34

O sea 34 que es el numero 4 en ASCII

También podemos ver que si al realizar

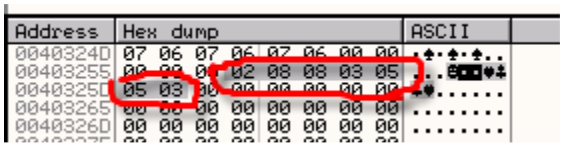
BYTE CORRECTO= 5 x 0A + 2

Al realizar 5 x 0A + 2 se pasa de los números posibles de una cifra (de 30 a 39) en ese caso es porque el resultado de la división no era cinco sino 4 por lo cual disminuimos y intentamos nuevamente con

BYTE CORRECTO= 4 x 0A + RESTO

Bueno ya calculamos nuestro primer byte es 34 o sea **4** en ASCII

Vemos los otros bytes correctos que va a comparar con mis restos.



Con 02 ya calculamos

Sigamos con 08

BYTE CORRECTO= 5 x 0A + 8

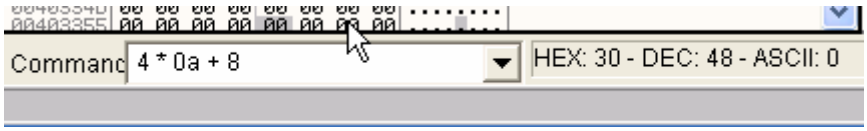
BYTE CORRECTO= 32 + 8

En este caso da 40 se pasa del máximo numero decimal que es 39 (9 en ASCII) así que disminuyamos a

BYTE CORRECTO= 4 x 0A + 8

BYTE CORRECTO= 28 + 8 = 30 que es 0 en ASCII

Si hacemos la prueba vemos que 30 dividido 0A da 4 con resto 8



Pues ya tenemos el segundo byte correcto que es 30 o sea **0** en ASCII

Si realizamos lo mismo byte a byte

El siguiente es 08, así que se repite lo mismo será 30 o sea **0** en ASCII

El siguiente es 03

BYTE CORRECTO= 5 x 0A + 3

BYTE CORRECTO= 32+ 3= 35 o sea el **5** en ASCII

El siguiente es el 05

BYTE CORRECTO= 5 x 0A + 5

BYTE CORRECTO= 32+ 5= 37 o sea el **7** en ASCII

El siguiente es 5 nuevamente, así que repite el 37 o sea **7** en ASCII

El ultimo es 3 que ya sabíamos era 35 o sea 5 en ASCII

Por lo cual el serial correcto para narvaja seria 4005775 quitando todos los BPM y BPX y dando RUN

Apreto el botón NAME/SERIAL CHECK

Otro adentro jeje

El crackme para practicar por ahora será uno de solo serial hardcoded de a poco iremos aumentando por supuesto con el serial correcto se activa la parte 17, así que a buscar que este es muy fácil, el crackme es el MEXCRK1.ZIP y esta adjunto con la lección.

Que tengan suerte así pueden llegar a la parte 17

Ricardo Narvaja
11 de diciembre de 2005