INTRODUCCION AL CRACKING EN OLLYDBG PARTE 26

CRACKEANDO VISUAL BASIC EN OLLYDBG

En las subsiguientes partes aprenderemos a crackear programas hechos en VISUAL BASIC en OLLYDBG, ya se me dirán algunos que para que existe una herramienta tan buena como SMARTCHECK, pero aquí este curso trata sobre cracking en OLLYDBG y trataremos siempre de utilizarlo al mismo antes de otras herramientas, en caso de que no podamos de ninguna forma con OLLYDBG o que se nos complique mucho, solo en ese caso, recurriremos a otras TOOLS.

Antes que nada les haré un regalito es este OLLYDBG ya conocido en la lista crackslatinos y es especial.

http://www.ricnar456.dyndns.org/HERRAMIENTAS/L-M-N-Ñ-O-P/OLLY%20PARCHEADO%20PARA%20BUSCAR%20OEPs.rar

Se llama OLLYDBG PARCHEADO PARA BUSCAR OEPs, que como vemos se utiliza mas que nada para desempacar, pero en Crackslatinos yo lo dije cuando lo envié, también es muy útil, para los programas en VISUAL BASIC en NATIVE code, no así para los hechos en P-CODE.

Pues cuales la diferencia entre ambos, ya veremos también capítulos de P-CODE pero el VISUAL BASIC NATIVE CODE es el visual común que ejecuta instrucciones en la sección code del programa, mientras que el P-CODE no ejecuta la sección code, solo se ejecuta la dll de visual y lee bytes del la sección code del programa como indicación de que comando querés ejecutar.

O sea si un programa de VisualBasic nunca vuelve a ejecutar líneas de código en la sección code, pues es casi seguro que es P-CODE del cual estudiaremos su forma de trabajar mas adelante.

Pues que particularidad tiene el OLLY este que parchee?, que si colocas un BPM ON ACCESS no para ON READ y ON EXECUTE, sino que para solo ON EXECUTE y esto es muy útil para hallar OEPs y para Visual Basic ya que si no el programa para miles de veces en la dll de Visual, y el tema es aquí siempre volver al código del programa, no empantanarnos traceando en las dll de VisualBasic lo cual te lleva a error seguro y a trabajar de mas.

También para trabajar en Visual Basic es necesario conocer las apis de VB, las cuales son diferentes a las apis comunes y no se encuentran en el WINAPIS32.

Por lo demás hay muchísimos tutes con Puntos especiales para VB, y técnicas geniales que no usaremos y el que quiere consultarlas, puede ver al el NUEVO CURSO de Crackslatinos los tutes de COCO y otros integrantes de la lista que pueden hacer mas fácil la tarea, aquí lo haremos como si no conocemos ninguna de esas técnicas y lo tenemos que crackear solo con este OLLY especial.

El mismo se coloca en la misma carpeta del OLLYDBG común, y lógicamente elegiremos usar este OLLY solo cuando crackeemos VB o cuando busquemos OEPs o alguna tarea que requiera que los BPM ON ACCESS funcionen realmente como BPM ON EXECUTION solamente.

Por supuesto si estamos crackeando con este OLLY, si necesitamos un BPM ON ACCESS real que pare ON READ y ON EXECUTE no podremos usarlo, o debemos arreglarnos con los HARDWARE BPX ON ACCESS que pueden a veces cumplir la función.

El problema máximo que hay con las apis de Visual Basic es que no es una información que Microsoft suministro ni suministra, por lo cual no hay un winapis32 con apis de VisualBasic ni nada de eso, lo mas que podes hacer es buscar el nombre de la api en GOOGLE y ver si tenes suerte de encontrar alguna pagina donde se use y explique como y para que sirve, antes de comenzar a crackear recopilare en este tute lo poco que encontré sobre apis y info. en general sobre visual Basic, cosa de que también puedan consultar este tute para saber algunas apis que significan.

SIGNIFICADO DE LAS PARTES DEL NOMBRE DE UNA API DE VISUAL

En las apis encontraremos estas abreviaturas como parte del nombre de las mismas.

bool Boolean
str String
i2 Integer (2 bytes)
ui2 Unsigned integer (2 bytes unsigned integer)
i4 Long (4 bytes integer)
r4 Single (4 bytes real)
r8 Double (8 bytes real)
cy Currency
var Variant or variable
fp Floating point
cmp compare
comp compare

Aquí en esta la tabla se ven las definiciones de los tipos de variables, pues combinando los tipos de variables se entiende mucho sobre que hace cada api.

Tipo de variable	Valor	Ocupa	Rango
Integer	Valor Entero	2 Bytes	-32768 a 32767
Long	Valor Entero Largo	4 Bytes	-2147483648 a 2147483647
Single	Valor Real	4 Bytes	-3,402823E38 a -1,401298E-45 (valores negativos) - 1,401298E-45 a 3,402823E38 (valores positivos)
Double	Valor Real Doble	8 Bytes	-1,79769313486232 E308 a -4,94065645841247 E-324 (valores negativos) 4,94065645841247E-324 a 1,79769313486232E308 (valores positivos)
String	Carácter (texto)	1 Byte por carácter	Desde 1 a 65000
Byte	Byte	1 Byte	0 a 255
Boolean	Valor Booleano (1/0)	2 Bytes	True o False (1 ó 0)
Currency (real)	Monedas y Punto Fijo	8 Bytes	-922337203685477,5808 a 922337203685477,5807
Date	Fecha	8 Bytes	01/01/100 a 31/12/9999
Object	Referencias a objetos	4 Bytes	[Objeto]
Variant	Cualquiera	16-22 Bytes	Números: 16 Bytes hasta el intervalo Double Caracteres: 22 Bytes + longitud ed la cadena

Por ejemplo:

__vbaI2Str quiere decir que convierte una string en un entero, por ejemplo.

Aquí hay más definiciones:

1) Ejemplos de apis de conversión de datos:

i) __vbaI2Str Convierte una String a Integer
ii) __vbaI4Str Convierte una String a Long
iii) __vbar4Str Convierte una String a Single
iv) __vbar8Str Convierte una String a Double
v) VarCyFromStr Convierte String a Currency
vi) VarBstrFromI2 Convierte Integer a String

Vemos que con las abreviaturas que vimos al inicio nos damos cuenta por donde vienen los tiros, aquí hay mas apis.

2) Moviendo datos

i) __vbaStrCopy - Copia una String a memoria
ii) __vbaVarCopy - Copia una Variable a memoria
iii) __vbaVarMove – Mueve una Variable en la memoria

3) Mathematical

i) __vbavaradd – Suma de dos Variables
ii) __vbavarsub – Resta dos Variables
iii) __vbavarmul – Multiplica dos Variables
iv) __vbavaridiv - Divide dos variables dando como resultado un Integer
v) __vbavarxor – function XOR

4) Miscelaneas:

i) __vbavarfornext - Loop
ii) __vbafreestr – Libera una String que no se utiliza
iii) __vbafreeobj – Libera un Objeto que no se utiliza
iv) __vbastrvarval – Toma el valor de una ubicación especifica en una String
v) multibytetowidechar – Cambia una String a formato ancho
vi) rtcMsgBox – Muestra un message box – similar a Windows API messagebox/a/exa
vii) __vbavarcat – Une dos variables
viii) __vbafreevar Libera una Variable que no se utiliza
ix) __vbaobjset Activa un Objeto
x) __vbaLenBstr – Obtiene el largo de una string
xi) rtcInputBox – Muestra una Input Box de Visual Basic – similar a Windows API getwindowtext/a, GetDlgItemtext/a
xii) __vbaNew – Muestra una caja de dialogo – Similar a Windows API Dialogbox
xiii) __vbaNew2 - Muestra una caja de dialogo Similar a Windows API Dialogboxparam/a
xiv) rtcTrimBstr – Recorta una String
xv) __vbaEnd – Finalizacion del Programa
xvi) __vbaLenVar - Obtiene el largo de una variable
xvii) rtcMidCharVar –Toma un determinado carácter de una string para trabajar con el.
xviii) _rtcDir – Busca si existe una fila.
xix) __vbaFileOpen – Abre una fila.

5) Comparaciones:

i) __vbastrcomp – Compara dos Strins - Similar a Windows API lstrcmp
ii) __vbastrcmp - Compara dos strings - Similar a Windows API lstrcmp
iii) __vbavartsteq - Compara dos variables si son iguales
iv) __vbaFpCmpCy - Compara Floating point con Currency
v) __vbavartstNe - Compara dos variables si no son iguales

Estas definiciones son para dar una idea de que esta haciendo el programa en ese momento, ya que no tenemos mucha ayuda, quizás no sean perfectas, pero nos ayudaran, asimismo hay muchas mas combinando lo que vimos nos daremos cuenta que hace la api.

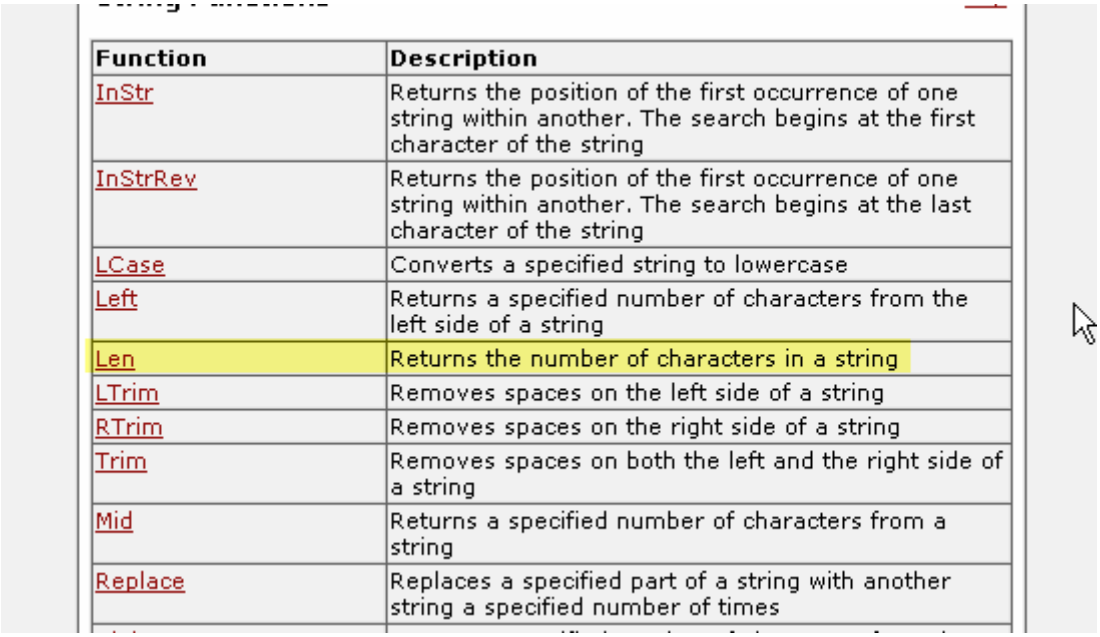
Asimismo aunque por ahora no veremos cracking en SMART CHECK en esta pagina tiene lo que quieren decir los comandos extraños para el cracker que vemos en el.

http://www.w3schools.com/vbscript/vbscript_ref_functions.asp

Por ejemplo el Smartcheck nos muestra la función ASC, o MID que no corresponde a lo que vemos en OLLYDBG si no al lenguaje de Visual Basic de programación, pero en esa página esta detallado que

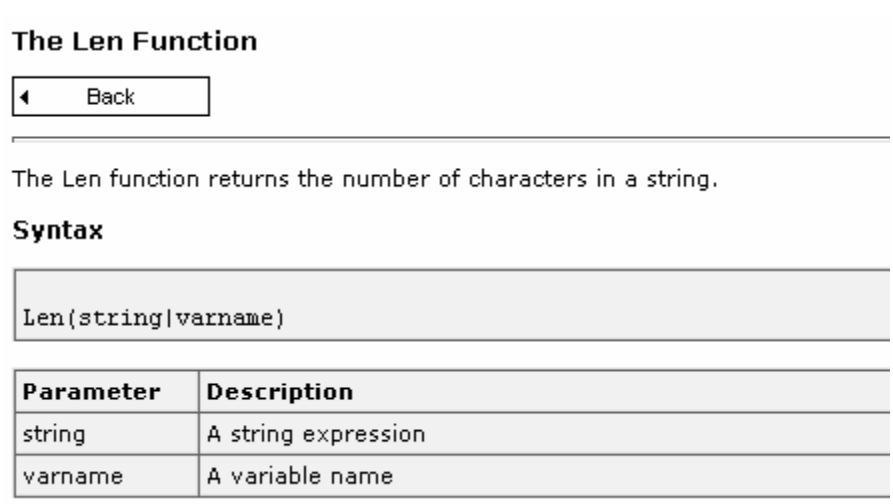
significa cada posible comando y sus parámetros por lo cual el cracking en SMART CHECK se facilita mucho.

Por ejemplo en SMART CHECK vemos el comando LEN que corresponde a la api __vbaLenVar,



Function	Description
InStr	Returns the position of the first occurrence of one string within another. The search begins at the first character of the string
InStrRev	Returns the position of the first occurrence of one string within another. The search begins at the last character of the string
LCase	Converts a specified string to lowercase
Left	Returns a specified number of characters from the left side of a string
Len	Returns the number of characters in a string
LTrim	Removes spaces on the left side of a string
RTrim	Removes spaces on the right side of a string
Trim	Removes spaces on both the left and the right side of a string
Mid	Returns a specified number of characters from a string
Replace	Replaces a specified part of a string with another string a specified number of times

En la pagina vemos que esta la definición de LEN.



The Len Function

Back

The Len function returns the number of characters in a string.

Syntax

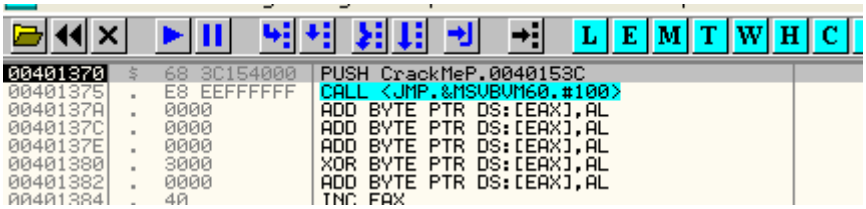
```
Len(string|varname)
```

Parameter	Description
string	A string expression
varname	A variable name

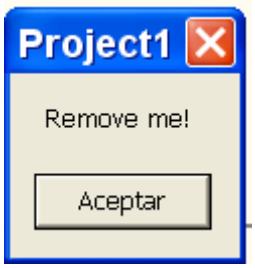
Pues para los que tienen problemas para usar el SMART CHECK eso los ayudara, aunque en este tute no veremos su uso, creo que ningún tute de SMART CHECK da una ayuda sobre ciertas expresiones usadas por el mismo, creo que eso servirá, por eso lo agregue aquí.

Bueno ya sabido todo lo anterior encaremos el cracking del CrackMePls

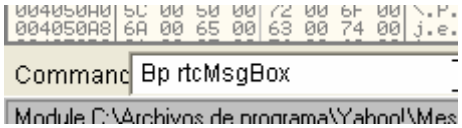
Abramos el mismo en el OLLYDBG parcheado por mi, que les di el link al inicio.



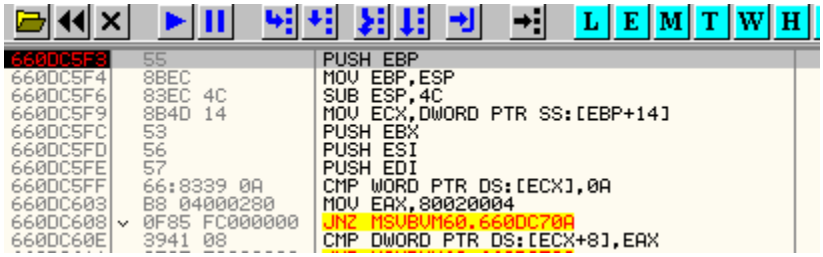
Este crackme es muy sencillo tiene una nag que eliminar y el serial que hallar, veamos, demos RUN.



Sale la nag, la cual varia de texto cada vez que la corremos, es tipo MessageBoxA, así que en VisualBasic eso corresponde a la api rtcMsgBox, si pongo un BP en dicha api.

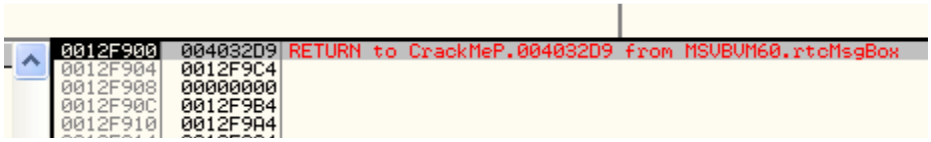


Y reinicio a ver si para antes de que salga la nag

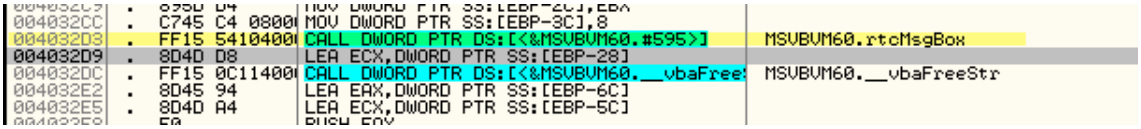


Allí paro, no tenemos ayuda de parámetros ni nada, pero no importa.

Veamos de donde fue llamada esta api como siempre aunque no diga RETURN TO el primer valor del stack es la dirección de retorno de la api, si no, lo analizamos aquí nuevamente y aparece la info.



Vayamos a ver de donde fue llamada esta api.

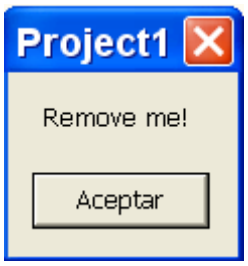


Allí esta el call a la api y abajo lógico el punto de retorno que es 4032d9.

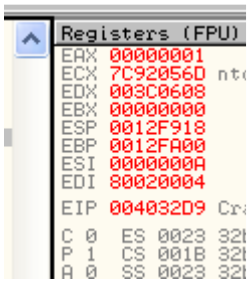
```
004032C6  .  8975 B4      MOV DWORD PTR SS:[EBP-4C],ESI
004032C9  .  895D D4      MOV DWORD PTR SS:[EBP-2C],EBX
004032CC  .  C745 C4 0800 MOV DWORD PTR SS:[EBP-3C],8
004032D3  .  FF15 5410400 CALL DWORD PTR DS:[<&MSVBVM60.#595>]     MSVBVM60.rtcMsgBox
004032D9  .  8D4D D8      LEA ECX,DWORD PTR SS:[EBP-28]
004032DC  .  FF15 0C11400 CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeStr
004032E2  .  8D45 94      LEA EAX,DWORD PTR SS:[EBP-6C]
004032E5  .  8D4D A4      LEA ECX,DWORD PTR SS:[EBP-5C]
004032E8  .  50           PUSH EAX
004032E9  .  8D55 B4      LEA EDX,DWORD PTR SS:[EBP-4C]
004032EC  .  51           PUSH ECX
```

```
004032C9  .  8955 D4      MOV DWORD PTR SS:[EBP-2C],EBX
004032CC  .  C745 C4 0800 MOV DWORD PTR SS:[EBP-3C],8
004032D3  .  FF15 5410400 CALL DWORD PTR DS:[<&MSVBVM60.#595>]     MSVBVM60.rtcMsgBox
004032D9  .  8D4D D8      LEA ECX,DWORD PTR SS:[EBP-28]
004032DC  .  FF15 0C11400 CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeStr
004032E2  .  8D45 94      LEA EAX,DWORD PTR SS:[EBP-6C]
004032E5  .  8D4D A4      LEA ECX,DWORD PTR SS:[EBP-5C]
004032E8  .  50           PUSH EAX
```

Pongo un BP en el punto de retorno de la api, y doy RUN

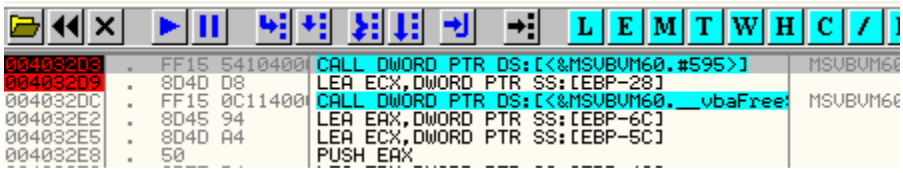


Acepto y para en el retorno.

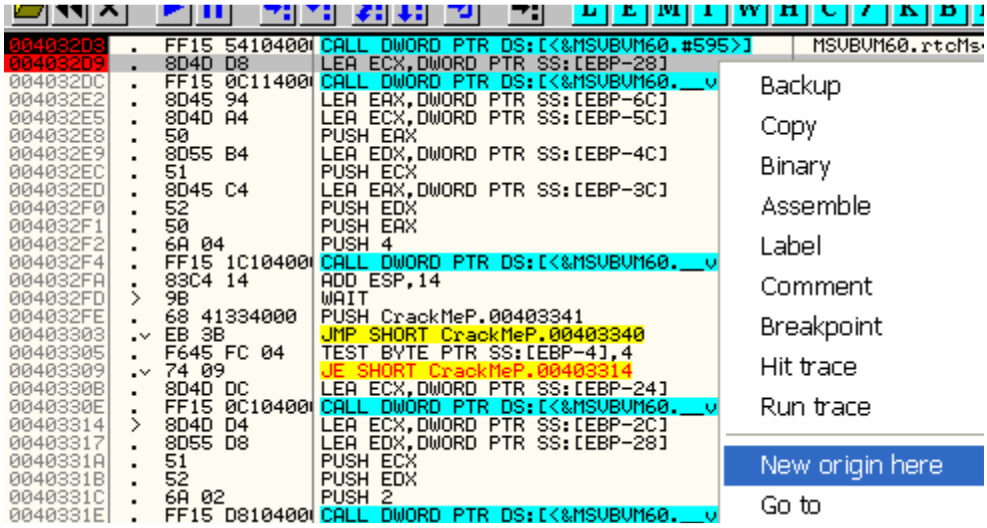


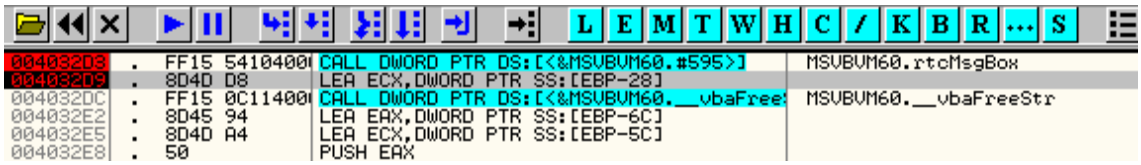
EAX =1 significa que la api se ejecuto con éxito, ahora reiniciemos, intentaremos nopear la nag de alguna forma.

Reiniciamos y para en el call

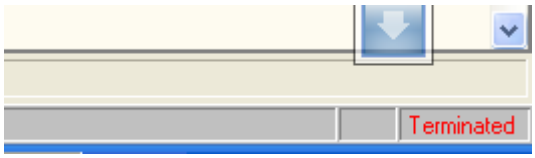


Si hago en la línea de retorno CLICK DERECHO -NEW ORIGIN HERE continuara ejecutándose desde alli, sin salir el cartel, es como una simulación de que ocurriría si nopeo el call.

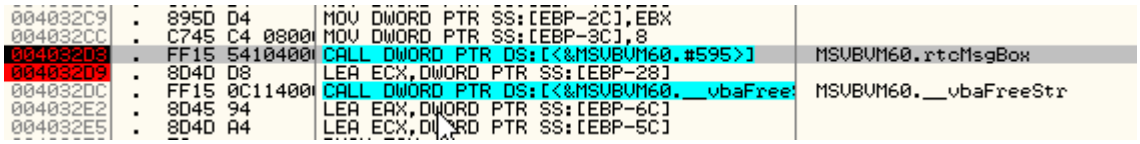




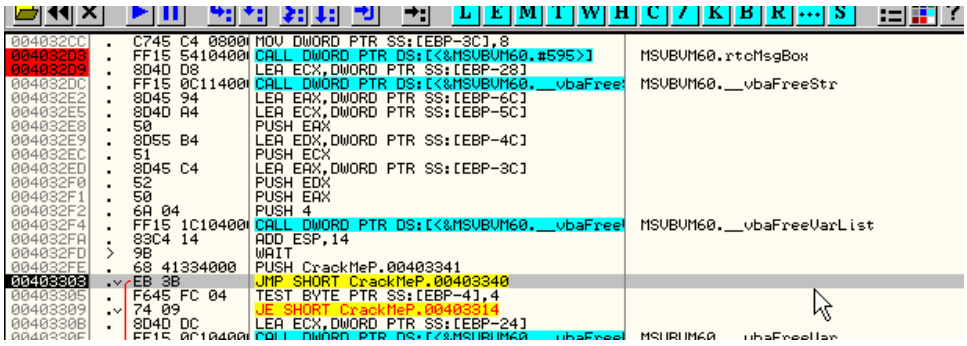
Allí esta demos Run a ver que pasa



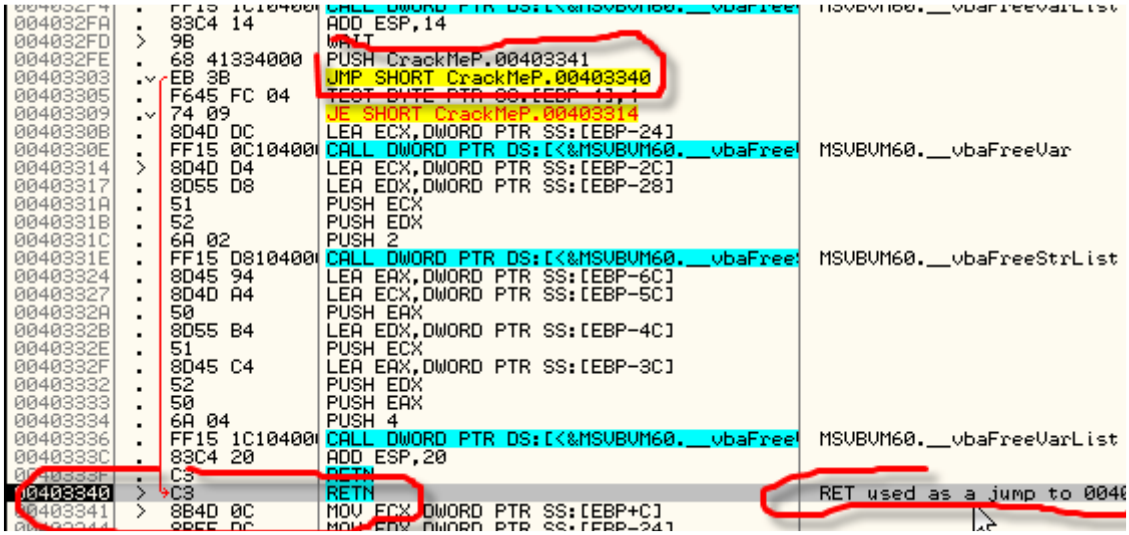
Evidentemente la solución de nopear el call a la api no funciona, probemos otra, reiniciemos.



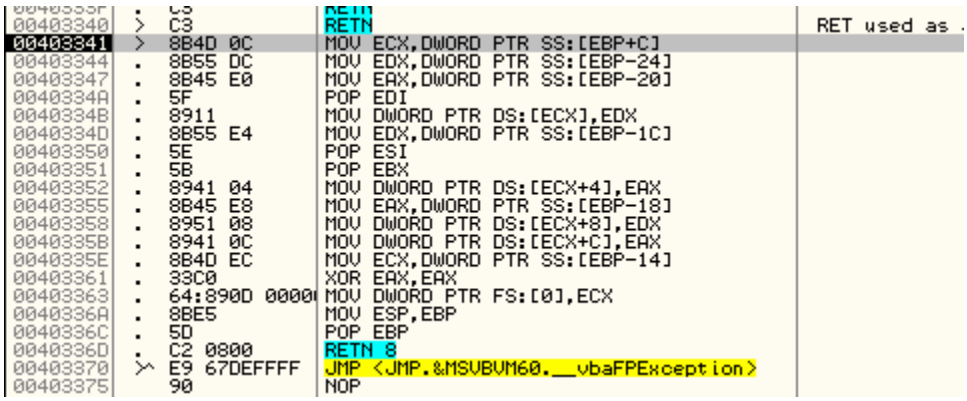
Lleguemos hasta 4032d9, y sigamos traceando con f8



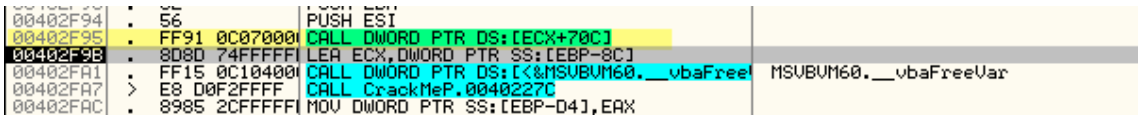
Llego al JMP y sigo traceando con F8



Antes del JMP hizo un PUSH 403341 y salta a un RET quiere decir que esta usando el RET como un salto a 403341, allí ollydbg lo aclara RET USED AS A JUMP TO 403341, por eso no aparece en el stack, jeje este truquito es para evitar que nos fijemos en el stack hacia abajo los retornos de los calls y lleguemos a la zona de 403341 mirando solamente el stack, este truco previene eso, sigamos traceando.

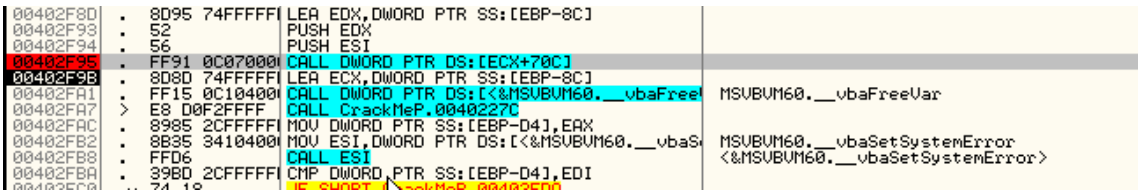


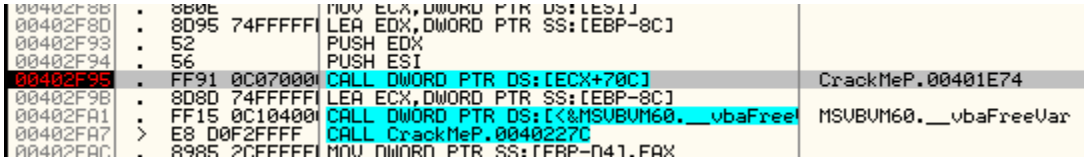
Llegamos hasta el RETN 8



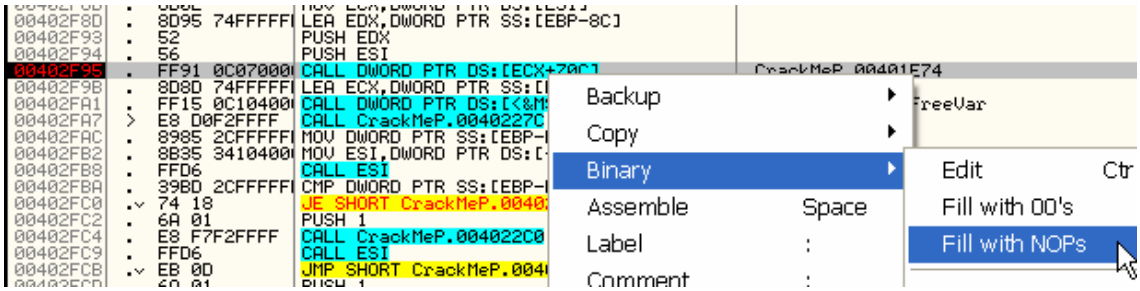
Allí esta acabamos de salir del call que esta en 402f95 pero eso no estaba en el stack gracias al truquito que uso para disimular este CALL y no mostrar la dirección de retorno.

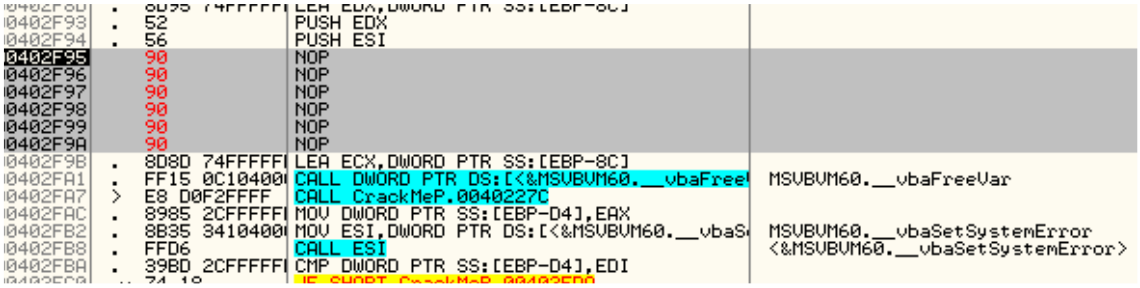
Pongamos un BP en este call y reiniciemos





Nopeemos el call a ver si corre sin la nag



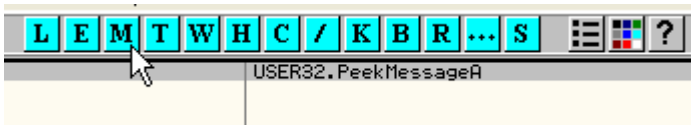


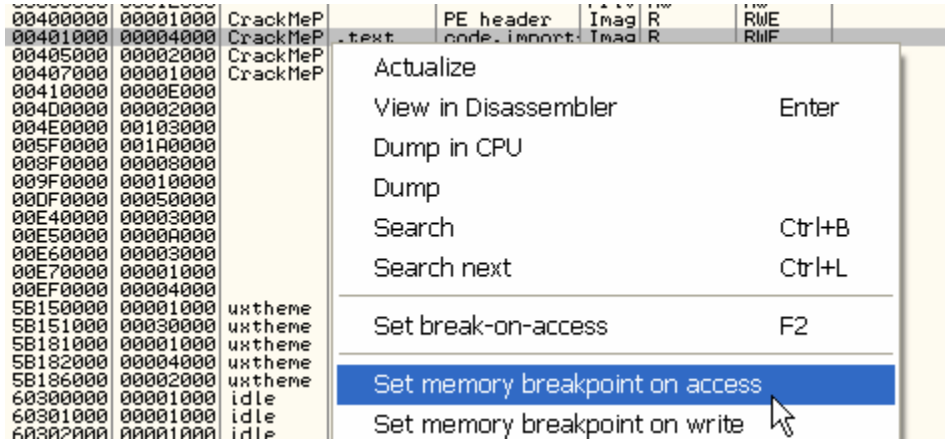
Demos RUN

Bye bye nag, jeje ya veremos mas adelante muchos ejemplos de remover nags, pero bueno este es sencillo salvo ese pequeño truco, la cuestión es ir saliendo desde la nag hacia los retornos de los calls, para ver si se puede nopear alguno de los calls anteriores, para evitar la aparición de la misma.

Allí tenemos el crackme tipeemos el serial bueno y dejémoslo RUNNING.

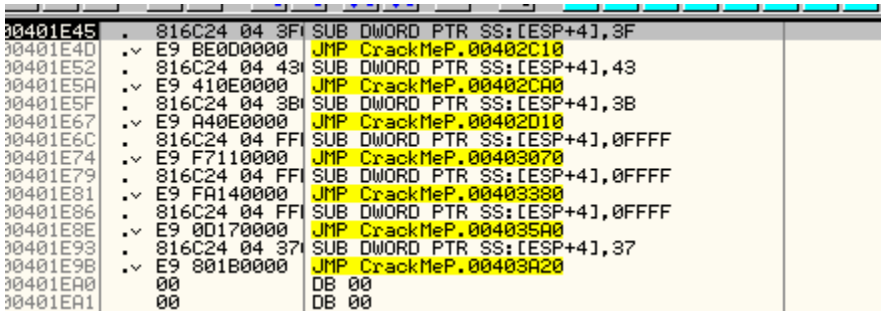
Ahora veremos el uso que le damos al OLLY este parcheado, lo mejor es intentar poner un BPM ON ACCESS (execute) en la sección code, vayamos a M.



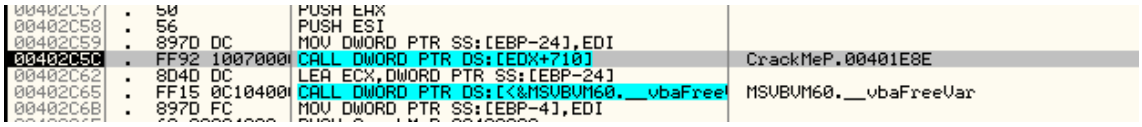


Recordemos que en este OLLY eso es un BPM ON EXECUTION, no para cuando lee y escribe en la sección ejecutando la dll de visual y eso es lo que necesitamos, volver rápido al programa y evitar las miles de veces que para ON READ antes de volver.

Apreto OK



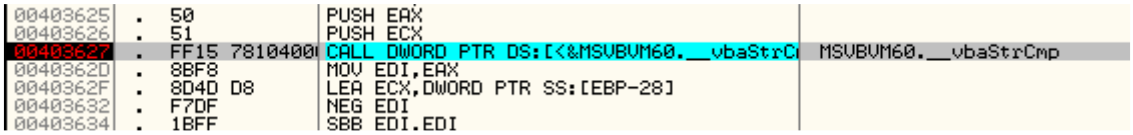
Paramos aquí traceemos un poco

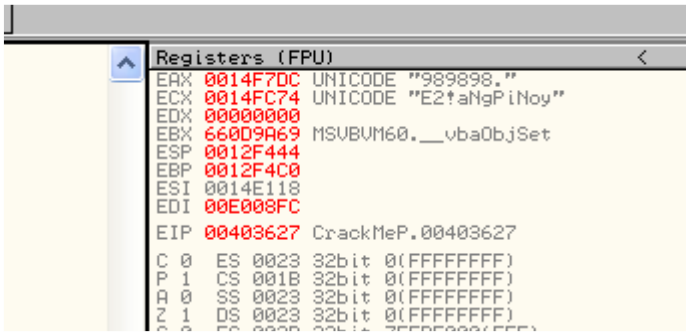


Llegamos a ese call recordamos pasar por encima los calls a apis, y entrar en los calls internos del programa.

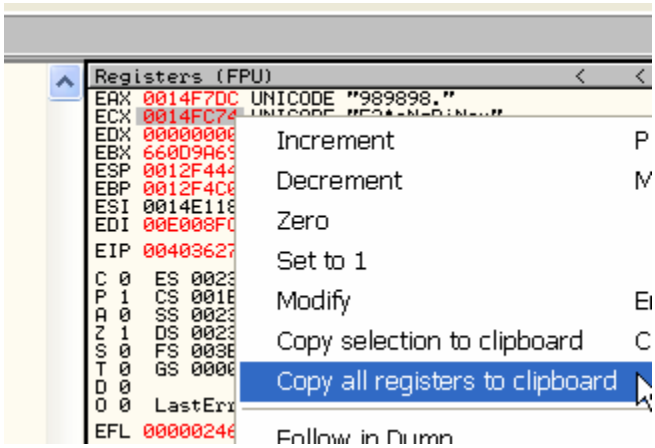
Entonces entramos con F7

Llegamos a la posible parte caliente abajo vemos __vbaStrCmp que es una posible comparación de strings, pongamos un BP allí y quitemos el BPM antes de dar RUN.

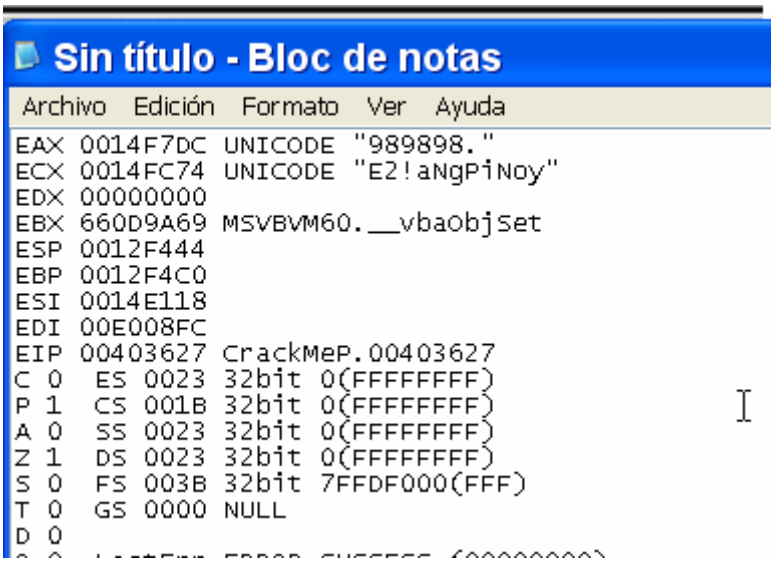




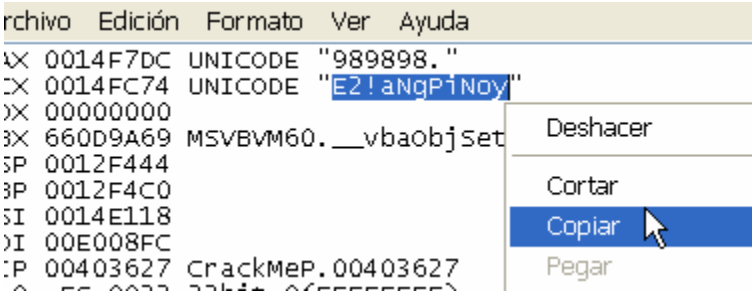
Vemos que compara la string que ingrese 989898 con el serial correcto, copiémoslo.



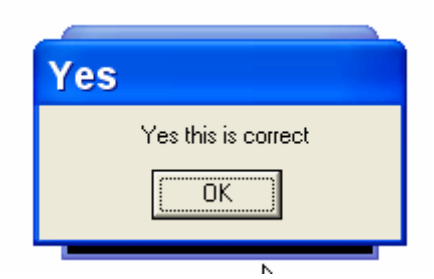
Y peguémoslo en el bloc de notas



De allí lo podemos copiar al serial correcto fácilmente



Probémoslo pegándolo en la ventana del crackme



Bueno resuelto, por supuesto hay que guardar los cambios del nopeo de la nag para que corra para siempre sin la misma.

En la parte 27 seguiremos profundizando el cracking en Visual Basic, con ejemplos mas complejos y ya empezaran a trabajar ustedes tambien.

Hasta la parte 27
Ricardo Narvaja
10 de enero de 2006