

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-N-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 puedes hacer es buscar el nombre de la api en GOOGLE y ver si tienes 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 específica 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) `__vbavarcats` - 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` - Finalización 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

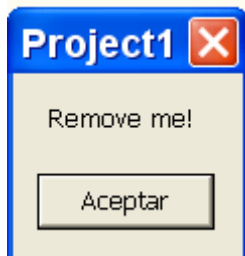
Pues para los que tienen problemas para usar el SMART CHECK eso los ayudara, aunque en este tute no veremos su uso, creo que ningún tute de SMART CHECK da una ayuda sobre ciertas expresiones usadas por el mismo, creo que eso servirá, por eso lo agregue aquí.

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.

Address	Disassembly	Comment
00401370	PUSH CrackMeP.0040153C	
00401375	CALL <JMP.&MSUBUM60.#100>	
0040137A	ADD BYTE PTR DS:[EAX],AL	
0040137C	ADD BYTE PTR DS:[EAX],AL	
0040137E	ADD BYTE PTR DS:[EAX],AL	
00401380	XOR BYTE PTR DS:[EAX],AL	
00401382	ADD BYTE PTR DS:[EAX],AL	
00401384	TNC FAX	

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.

Address	Disassembly	Comment
004050A0	5C 00 50 00 72 00 6F 00	\.P.
004050A8	6A 00 65 00 63 00 74 00	j.e.

Command: Bp rtcMsgBox

Module: C:\Archivos de programa\Yahoo\Mes

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

Address	Disassembly	Comment
660DC5F3	PUSH EBP	
660DC5F4	MOV EBP,ESP	
660DC5F6	SUB ESP,4C	
660DC5F9	MOV ECX,DWORD PTR SS:[EBP+14]	
660DC5FC	PUSH EBX	
660DC5FD	PUSH ESI	
660DC5FE	PUSH EDI	
660DC5FF	CMP WORD PTR DS:[ECX],0A	
660DC603	MOV EAX,80020004	
660DC608	JNZ MSUBUM60.660DC70A	
660DC60E	CMP DWORD PTR DS:[ECX+8],EAX	

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.

Address	Disassembly	Comment
0012F900	004032D9	RETURN to CrackMeP.004032D9 from MSUBUM60.rtcMsgBox
0012F904	0012F9C4	
0012F908	00000000	
0012F90C	0012F9B4	
0012F910	0012F9A4	

Vayamos a ver de donde fue llamada esta api.

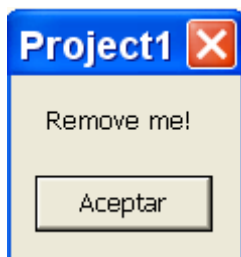
Address	Disassembly	Comment
004032C7	MOV DWORD PTR SS:[EBP-2C],EAX	
004032CC	MOV DWORD PTR SS:[EBP-3C],8	
004032D3	CALL DWORD PTR DS:[<&MSUBUM60.#595>]	MSUBUM60.rtcMsgBox
004032D9	LEA ECX,DWORD PTR SS:[EBP-28]	
004032DC	CALL DWORD PTR DS:[<&MSUBUM60.__vbaFree	MSUBUM60.__vbaFreeStr
004032E2	LEA EAX,DWORD PTR SS:[EBP-6C]	
004032E5	LEA ECX,DWORD PTR SS:[EBP-5C]	

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

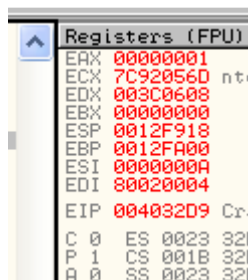
004032C6	. 8975 B4	MOV DWORD PTR SS:[EBP-4C],ESI	
004032C9	. 895D D4	MOV DWORD PTR SS:[EBP-2C],EBX	
004032CC	. C745 C4 0800	MOV DWORD PTR SS:[EBP-3C],8	
004032D3	. FF15 54104000	CALL DWORD PTR DS:[<&MSUBUM60.#595>]	MSUBUM60.rtcMsgBox
004032D9	. 8D4D D8	LEA ECX,DWORD PTR SS:[EBP-28]	
004032DC	. FF15 0C114000	CALL DWORD PTR DS:[<&MSUBUM60.__vbaFreeStr	MSUBUM60.__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	
004032F1	. 8D45 C4	LEA EDI,DWORD PTR SS:[EBP-3C]	

004032C9	. 895D D4	MOV DWORD PTR SS:[EBP-2C],EBX	
004032CC	. C745 C4 0800	MOV DWORD PTR SS:[EBP-3C],8	
004032D3	. FF15 54104000	CALL DWORD PTR DS:[<&MSUBUM60.#595>]	MSUBUM60.rtcMsgBox
004032D9	. 8D4D D8	LEA ECX,DWORD PTR SS:[EBP-28]	
004032DC	. FF15 0C114000	CALL DWORD PTR DS:[<&MSUBUM60.__vbaFreeStr	MSUBUM60.__vbaFreeStr
004032E2	. 8D45 94	LEA EAX,DWORD PTR SS:[EBP-6C]	
004032E5	. 8D4D A4	LEA ECX,DWORD PTR SS:[EBP-5C]	
004032F1	. 50	PUSH EAX	

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

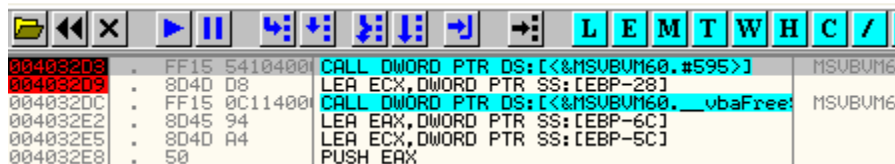


Acepto y para en el retorno.

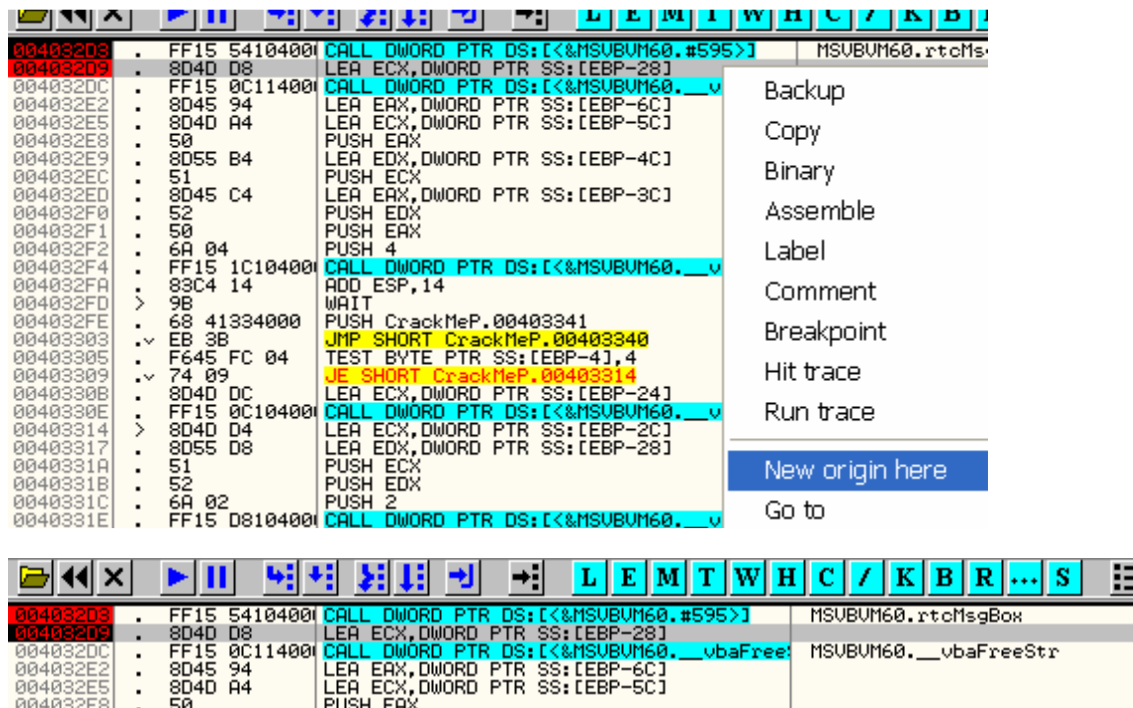


EAX =1 significa que la api se ejecuto con éxito, ahora reiniciemos, intentaremos nopear la nag de alguna forma.

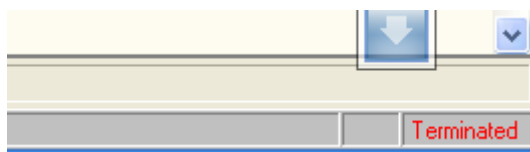
Reiniciamos y para en el call



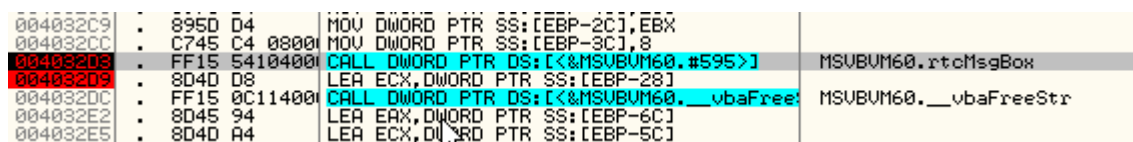
Si hago en la línea de retorno CLICK DERECHO -NEW ORIGIN HERE continuara ejecutándose desde allí, sin salir el cartel, es como una simulación de que ocurriría si noopeo el call.



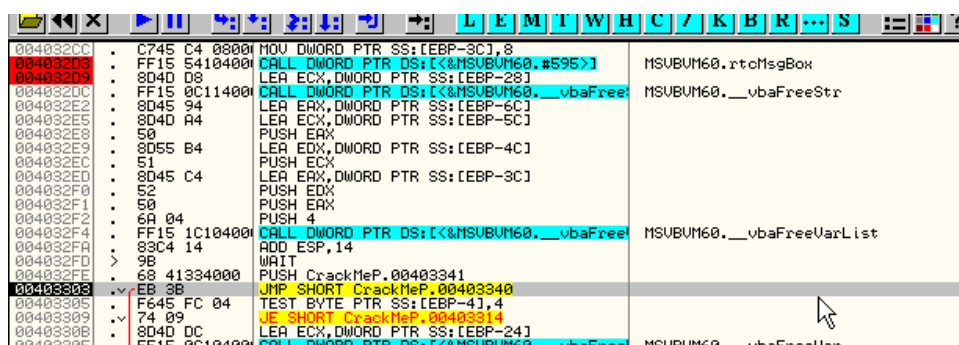
Allí esta demos Run a ver que pasa



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



Llegemos hasta 4032d9, y sigamos traceando con f8



Llego al JMP y sigo traceando con F8

0040332F	FF15 1C104000	CALL DWORD PTR DS:[&MSUBUM60.__vbaFree	MSUBUM60.__vbaFreeVarList
0040332FA	83C4 14	ADD ESP,14	
0040332FD	9B	WAIT	
0040332FE	68 41334000	PUSH CrackMeP.00403341	
004033303	EB 3B	JMP SHORT CrackMeP.00403340	
004033305	F645 FC 04	TEST BYTE PTR SS:[EBP-13],1	
004033309	74 09	JE SHORT CrackMeP.00403314	
00403330B	8D4D DC	LEA ECX,DWORD PTR SS:[EBP-24]	
00403330E	FF15 0C104000	CALL DWORD PTR DS:[&MSUBUM60.__vbaFree	MSUBUM60.__vbaFreeVar
004033314	8D4D D4	LEA ECX,DWORD PTR SS:[EBP-2C]	
004033317	8D55 D8	LEA EDX,DWORD PTR SS:[EBP-28]	
00403331A	51	PUSH ECX	
00403331B	52	PUSH EDX	
00403331C	6A 02	PUSH 2	
00403331E	FF15 08104000	CALL DWORD PTR DS:[&MSUBUM60.__vbaFree	MSUBUM60.__vbaFreeStrList
004033324	8D45 94	LEA EAX,DWORD PTR SS:[EBP-6C]	
004033327	8D4D A4	LEA ECX,DWORD PTR SS:[EBP-5C]	
00403332A	50	PUSH EAX	
00403332B	8D55 B4	LEA EDX,DWORD PTR SS:[EBP-4C]	
00403332E	51	PUSH ECX	
00403332F	8D45 C4	LEA EAX,DWORD PTR SS:[EBP-3C]	
004033332	52	PUSH EDX	
004033333	50	PUSH EAX	
004033334	6A 04	PUSH 4	
004033336	FF15 1C104000	CALL DWORD PTR DS:[&MSUBUM60.__vbaFree	MSUBUM60.__vbaFreeVarList
00403333C	83C4 20	ADD ESP,20	
00403333F	C3	RET	
00403340	C3	RET	RET used as a jump to 00403341
00403341	8B4D 0C	MOV ECX,DWORD PTR SS:[EBP+C]	
00403344	8B55 DC	MOV EDX,DWORD PTR SS:[EBP-24]	

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.

0040333F	C3	RET	
00403340	C3	RET	RET used as
00403341	8B4D 0C	MOV ECX,DWORD PTR SS:[EBP+C]	
00403344	8B55 DC	MOV EDX,DWORD PTR SS:[EBP-24]	
00403347	8B45 E0	MOV EAX,DWORD PTR SS:[EBP-20]	
0040334A	5F	POP EDI	
0040334B	8911	MOV DWORD PTR DS:[ECX],EDX	
0040334D	8B55 E4	MOV EDX,DWORD PTR SS:[EBP-1C]	
00403350	5E	POP ESI	
00403351	5B	POP EBX	
00403352	8941 04	MOV DWORD PTR DS:[ECX+4],EAX	
00403355	8B45 E8	MOV EAX,DWORD PTR SS:[EBP-18]	
00403358	8951 08	MOV DWORD PTR DS:[ECX+8],EDX	
0040335B	8941 0C	MOV DWORD PTR DS:[ECX+C],EAX	
0040335E	8B4D EC	MOV ECX,DWORD PTR SS:[EBP-14]	
00403361	33C0	XOR EAX,EAX	
00403363	64:890D 0000	MOV DWORD PTR FS:[0],ECX	
0040336A	8BE5	MOV ESP,EBP	
0040336C	5D	POP EBP	
0040336D	C2 0800	RET 8	
00403370	E9 67DEFFFF	JMP <JMP.&MSUBUM60.__vbaFPEXception>	
00403375	90	NOP	

Llegamos hasta el RETN 8

00402F94	56	PUSH ESI	
00402F95	FF91 0C070000	CALL DWORD PTR DS:[ECX+70C]	
00402F9B	8D8D 74FFFFFF	LEA ECX,DWORD PTR SS:[EBP-8C]	
00402FA1	FF15 0C104000	CALL DWORD PTR DS:[&MSUBUM60.__vbaFree	MSUBUM60.__vbaFreeVar
00402FA7	E8 D0F2FFFF	CALL CrackMeP.0040227C	
00402FAC	8985 2CFFFFFF	MOV DWORD PTR SS:[EBP-D4],EAX	

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

00402F8D	8D95 74FFFFFF	LEA EDX,DWORD PTR SS:[EBP-8C]	
00402F93	52	PUSH EDX	
00402F94	56	PUSH ESI	
00402F95	FF91 0C070000	CALL DWORD PTR DS:[ECX+70C]	
00402F9B	8D8D 74FFFFFF	LEA ECX,DWORD PTR SS:[EBP-8C]	
00402FA1	FF15 0C104000	CALL DWORD PTR DS:[&MSUBUM60.__vbaFree	MSUBUM60.__vbaFreeVar
00402FA7	E8 D0F2FFFF	CALL CrackMeP.0040227C	
00402FAC	8985 2CFFFFFF	MOV DWORD PTR SS:[EBP-D4],EAX	
00402FB2	8B35 34104000	MOV ESI,DWORD PTR DS:[&MSUBUM60.__vbaS	MSUBUM60.__vbaSetSystemError
00402FB8	FFD6	CALL ESI	<&MSUBUM60.__vbaSetSystemError>
00402FBA	39BD 2CFFFFFF	CMP DWORD PTR SS:[EBP-D4],EDI	
00402FC0	74 10	JC SHORT CrackMeP.00402FDB	

00402F8B	8B0E	MOV ECX,DWORD PTR DS:[ESI]	
00402F8D	8D95 74FFFFFF	LEA EDX,DWORD PTR SS:[EBP-8C]	
00402F93	52	PUSH EDX	
00402F94	56	PUSH ESI	
00402F95	FF91 0C070000	CALL DWORD PTR DS:[ECX+70C]	CrackMeP.00401E74
00402F9B	8D8D 74FFFFFF	LEA ECX,DWORD PTR SS:[EBP-8C]	
00402FA1	FF15 0C104000	CALL DWORD PTR DS:[<&MSUBUM60.__vbaFreeVar	MSUBUM60.__vbaFreeVar
00402FA7	E8 D0F2FFFF	CALL CrackMeP.0040227C	
00402FAC	8985 2CFFFFFF	MOV DWORD PTR SS:[EBP-4],EAX	
00402FB2	8B35 34104000	MOV ESI,DWORD PTR DS:[<&MSUBUM60.__vbaSetSystemError	
00402FB8	FFD6	CALL ESI	
00402FBA	39BD 2CFFFFFF	CMP DWORD PTR SS:[EBP-4],EDI	
00402FBC	74 18	JE SHORT CrackMeP.00402FDB	
00402FC0	6A 01	PUSH 1	
00402FC2	E8 F7F2FFFF	CALL CrackMeP.004022C0	
00402FC9	FFD6	CALL ESI	
00402FCB	EB 00	JMP SHORT CrackMeP.00402FDB	
00402FCD	6A 01	PUSH 1	

Nopeemos el call a ver si corre sin la nag

00402F8B	8B0E	MOV ECX,DWORD PTR DS:[ESI]	
00402F8D	8D95 74FFFFFF	LEA EDX,DWORD PTR SS:[EBP-8C]	
00402F93	52	PUSH EDX	
00402F94	56	PUSH ESI	
00402F95	FF91 0C070000	CALL DWORD PTR DS:[ECX+70C]	CrackMeP.00401E74
00402F9B	8D8D 74FFFFFF	LEA ECX,DWORD PTR SS:[EBP-8C]	
00402FA1	FF15 0C104000	CALL DWORD PTR DS:[<&MSUBUM60.__vbaFreeVar	MSUBUM60.__vbaFreeVar
00402FA7	E8 D0F2FFFF	CALL CrackMeP.0040227C	
00402FAC	8985 2CFFFFFF	MOV DWORD PTR SS:[EBP-4],EAX	
00402FB2	8B35 34104000	MOV ESI,DWORD PTR DS:[<&MSUBUM60.__vbaSetSystemError	
00402FB8	FFD6	CALL ESI	
00402FBA	39BD 2CFFFFFF	CMP DWORD PTR SS:[EBP-4],EDI	
00402FBC	74 18	JE SHORT CrackMeP.00402FDB	
00402FC0	6A 01	PUSH 1	
00402FC2	E8 F7F2FFFF	CALL CrackMeP.004022C0	
00402FC9	FFD6	CALL ESI	
00402FCB	EB 00	JMP SHORT CrackMeP.00402FDB	
00402FCD	6A 01	PUSH 1	

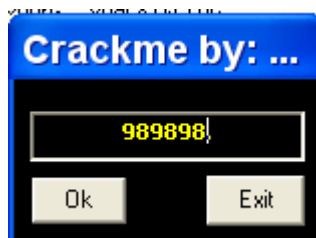
00402F8B	8B0E	MOV ECX,DWORD PTR DS:[ESI]	
00402F8D	8D95 74FFFFFF	LEA EDX,DWORD PTR SS:[EBP-8C]	
00402F93	52	PUSH EDX	
00402F94	56	PUSH ESI	
00402F95	90	NOP	
00402F96	90	NOP	
00402F97	90	NOP	
00402F98	90	NOP	
00402F99	90	NOP	
00402F9A	90	NOP	
00402F9B	8D8D 74FFFFFF	LEA ECX,DWORD PTR SS:[EBP-8C]	
00402FA1	FF15 0C104000	CALL DWORD PTR DS:[<&MSUBUM60.__vbaFreeVar	MSUBUM60.__vbaFreeVar
00402FA7	E8 D0F2FFFF	CALL CrackMeP.0040227C	
00402FAC	8985 2CFFFFFF	MOV DWORD PTR SS:[EBP-4],EAX	
00402FB2	8B35 34104000	MOV ESI,DWORD PTR DS:[<&MSUBUM60.__vbaSetSystemError	MSUBUM60.__vbaSetSystemError
00402FB8	FFD6	CALL ESI	
00402FBA	39BD 2CFFFFFF	CMP DWORD PTR SS:[EBP-4],EDI	
00402FBC	74 18	JE SHORT CrackMeP.00402FDB	
00402FCD	6A 01	PUSH 1	

Demos RUN

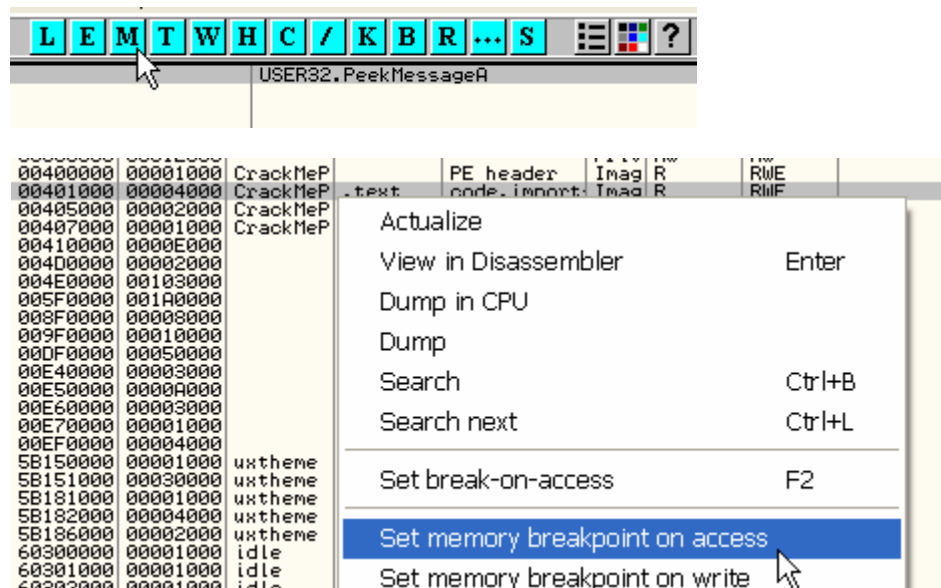


Bye bye nag, jeje ya veremos mas adelante muchos ejemplos de remover nags, pero bueno este es sencillo salvo ese pequeño truco, la cuestión es ir saliendo desde la nag hacia los retornos de los calls, para ver si se puede nopear alguno de los calls anteriores, para evitar la aparición de la misma.

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

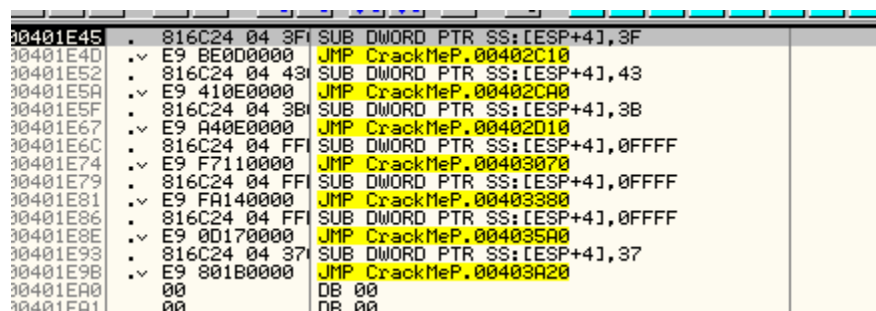


Ahora veremos el uso que le damos al OLLY este parcheado, lo mejor es intentar poner un BPM ON ACCESS (execute) en la sección code, vayamos a M.

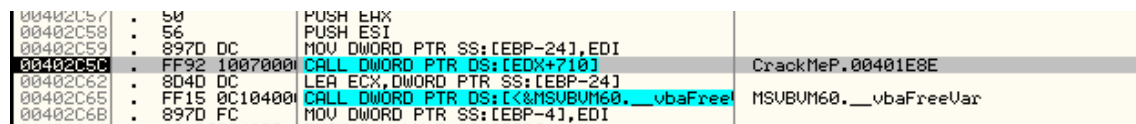


Recordemos que en este OLLY eso es un BPM ON EXECUTION, no para cuando lee y escribe en la sección ejecutando la dll de visual y eso es lo que necesitamos, volver rápido al programa y evitar las miles de veces que para ON READ antes de volver.

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

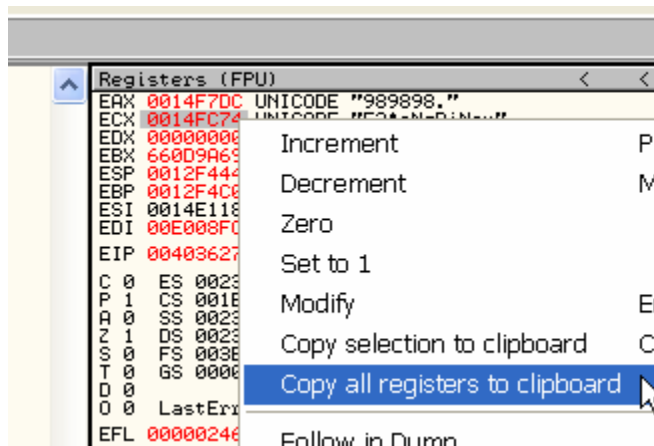
004035A0	> 55	PUSH EBP	
004035A1	. 8BEC	MOV EBP,ESP	
004035A3	. 83EC 0C	SUB ESP,0C	
004035A6	. 68 D6114000	PUSH <JMP.&MSUBUM60.__vbaExceptionHandler>	SE handler installation
004035AB	. 64:A1 000000	MOV EAX,DWORD PTR FS:[0]	
004035B1	. 50	PUSH EAX	
004035B2	. 64:8925 0000	MOV DWORD PTR FS:[0],ESP	
004035B3	. 83EC 54	SUB ESP,54	
004035BC	. 53	PUSH EBX	
004035BD	. 56	PUSH ESI	
004035BE	. 57	PUSH EDI	
004035BF	. 8965 F4	MOV DWORD PTR SS:[EBP-C],ESP	
004035C2	. C745 F9 9811	MOV DWORD PTR SS:[EBP-8],CrackMeP.00401	
004035C3	. 8B4D 0C	MOV ECX,DWORD PTR SS:[EBP+C]	
004035C5	. 8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]	
004035CF	. 33C0	XOR EAX,EAX	
004035D1	. 56	PUSH ESI	
004035D2	. 8901	MOV DWORD PTR DS:[ECX],EAX	
004035D4	. 8B16	MOV EDX,DWORD PTR DS:[ESI]	
004035D6	. 8945 DC	MOV DWORD PTR SS:[EBP-24],EAX	
004035D9	. 8945 D8	MOV DWORD PTR SS:[EBP-28],EAX	
004035DC	. 8945 D4	MOV DWORD PTR SS:[EBP-2C],EAX	
004035DF	. 8945 D0	MOV DWORD PTR SS:[EBP-30],EAX	
004035E2	. 8945 AC	MOV DWORD PTR SS:[EBP-54],EAX	
004035E5	. FF92 00030000	CALL DWORD PTR DS:[EDX+300]	
004035EB	. 8B1D 58104000	MOV EBX,DWORD PTR DS:[<&MSUBUM60.__vbaObjSet	MSUBUM60.__vbaObjSet
004035F1	. 50	PUSH EAX	
004035F2	. 8D45 D0	LEA EAX,DWORD PTR SS:[EBP-30]	
004035F5	. 50	PUSH EAX	
004035F6	. FFD3	CALL EBX	<&MSUBUM60.__vbaObjSet>
004035F8	. 8BF8	MOV EDI,EAX	
004035FA	. 8D55 D8	LEA EDX,DWORD PTR SS:[EBP-28]	
004035FD	. 52	PUSH EDX	
004035FE	. 57	PUSH EDI	
004035FF	. 8B0F	MOV ECX,DWORD PTR DS:[EDI]	
00403601	. FF91 A0000000	CALL DWORD PTR DS:[ECX+A0]	
00403607	. 85C0	TEST EAX,EAX	
00403609	. DBE2	FCLEX	
0040360B	. 7D 12	JGE SHORT CrackMeP.0040361F	
0040360D	. 68 A0000000	PUSH 0A0	
00403612	. 68 C4244000	PUSH CrackMeP.004024C4	
00403617	. 57	PUSH EDI	
00403618	. 50	PUSH EAX	
00403619	. FF15 38104000	CALL DWORD PTR DS:[<&MSUBUM60.__vbaHres	MSUBUM60.__vbaHresultCheckObj
0040361F	. 8B45 D8	MOV EAX,DWORD PTR SS:[EBP-28]	
00403622	. 8B4E 34	MOV ECX,DWORD PTR DS:[ESI+34]	
00403625	. 50	PUSH EAX	
00403626	. 51	PUSH ECX	
00403627	. FF15 78104000	CALL DWORD PTR DS:[<&MSUBUM60.__vbaStrC	MSUBUM60.__vbaStrCmp
0040362D	. 8BF8	MOV EDI,EAX	

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.

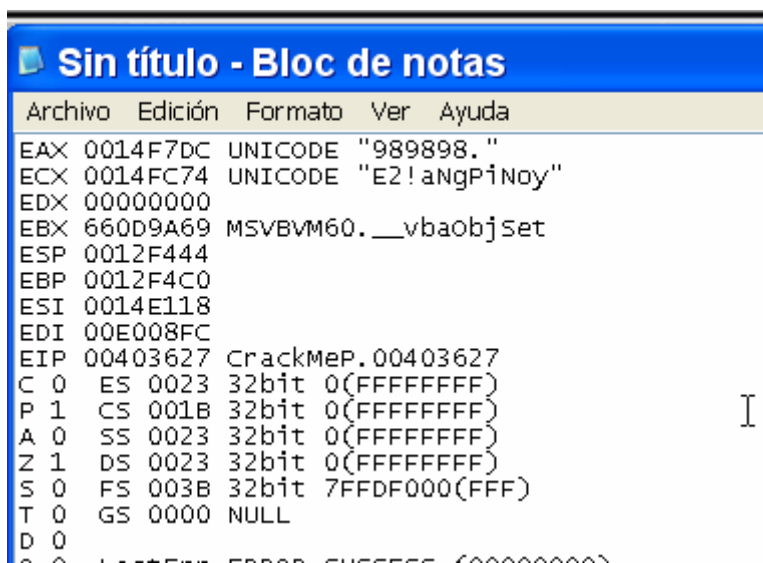
00403625	. 50	PUSH EAX	
00403626	. 51	PUSH ECX	
00403627	. FF15 78104000	CALL DWORD PTR DS:[<&MSUBUM60.__vbaStrC	MSUBUM60.__vbaStrCmp
0040362D	. 8BF8	MOV EDI,EAX	
0040362F	. 8D4D D8	LEA ECX,DWORD PTR SS:[EBP-28]	
00403632	. F7DF	NEG EDI	
00403634	. 1BFF	SBB EDI,EDI	

Registers (FPU)			
EAX	0014F7DC	UNICODE	"989898."
ECX	0014FC74	UNICODE	"E2!aNgP iNoy"
EDX	00000000		
EBX	660D9A69	MSUBUM60.__vbaObjSet	
ESP	0012F444		
EBP	0012F4C0		
ESI	0014E118		
EDI	00E008FC		
EIP	00403627	CrackMeP.00403627	
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)

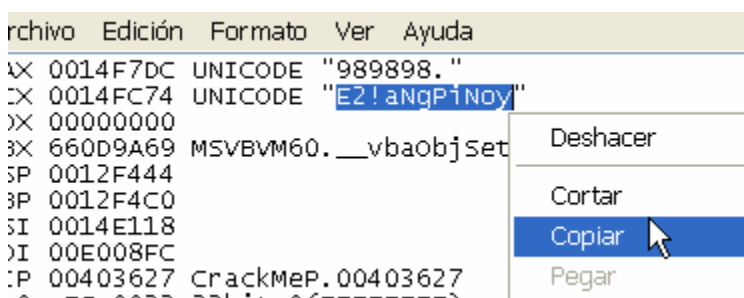
Vemos que compara la string que ingrese 989898 con el serial correcto, copiémoslo.



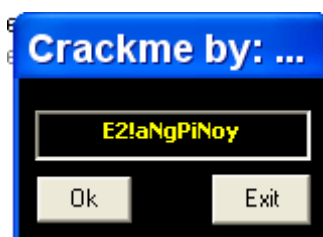
Y peguemoslo en el bloc de notas

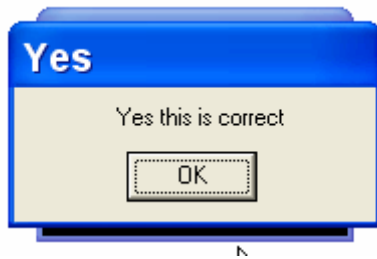


De allí lo podemos copiar al serial correcto fácilmente



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