

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 28

Mas técnicas para Visual Basic (la guerra total)

Personalmente he escuchado muchos crackers que dicen que hacer tal o cual método no es cracking puro, o cracking elegante, o no queda bien, o cosas por el estilo, pues para mi el cracking elegante es el que funciona, y vale todo, es como una guerra y le puedo asegurar que los programadores y sobretodo los packers y protectores no se detienen en pensar que es elegante o no, usan los métodos mas guarros que pueden, sin pensar que puede afectar nuestras máquinas, ya veremos ejemplos de guarradas cometidas por packers en su desesperado intento de que no se pueda desempacar su programa protegido.

Por lo tanto creo que si el enemigo usa misiles, limitarme yo a pelear con un revolver es una clara desventaja, por lo tanto yo uso CUALQUIER METODO lo aclaro desde ya, el que no le guste alguno, pues que busque alguno aprobado por la ACE (Asociación de Crackers Elegantes, jeje), yo no me detengo en pavadas, si funciona bien, no perjudica otros programas, es correcto y sirve.

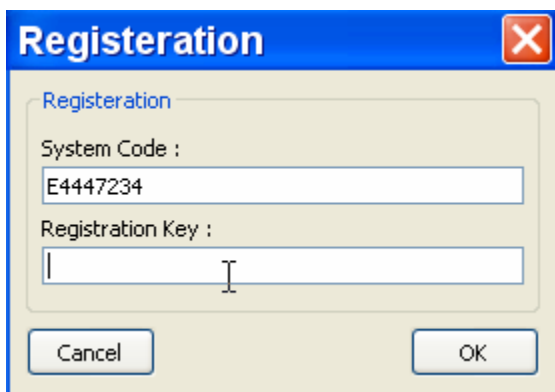
Aclarado el punto vamos a ver el crackme que deje de la semana pasada, que será objeto de estudio para este método, me refiero al



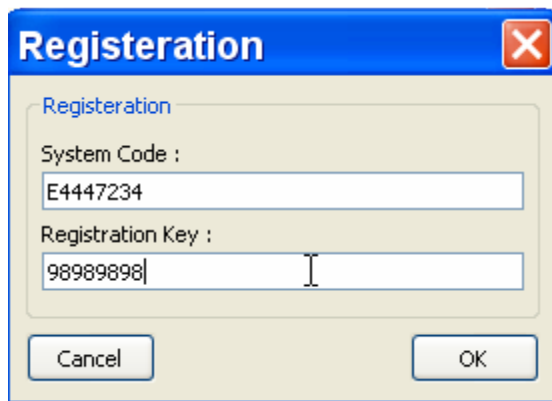
Pues el otro que deje, con el método del 4c se puede quitar la nag perfectamente y la parte del serial esta hecha en PCODE por lo cual lo dejaremos para cuando llegemos a esa instancia.

En este crackme hallar el serial es una pavada, el tema pasa por esa ventana, que no se si es nag o no, pero yo la quiero sacar, el que aplico el método del 4c habrá visto que hay dos forms, y que no hay forma de evitar que aparezcan ambas juntas usando ese método.

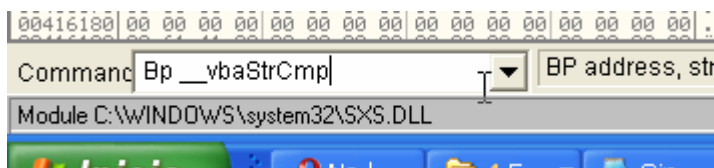
Antes que nada veamos el serial es muy simple.



Llego a la ventana del crackme y tipeo mi serial falso



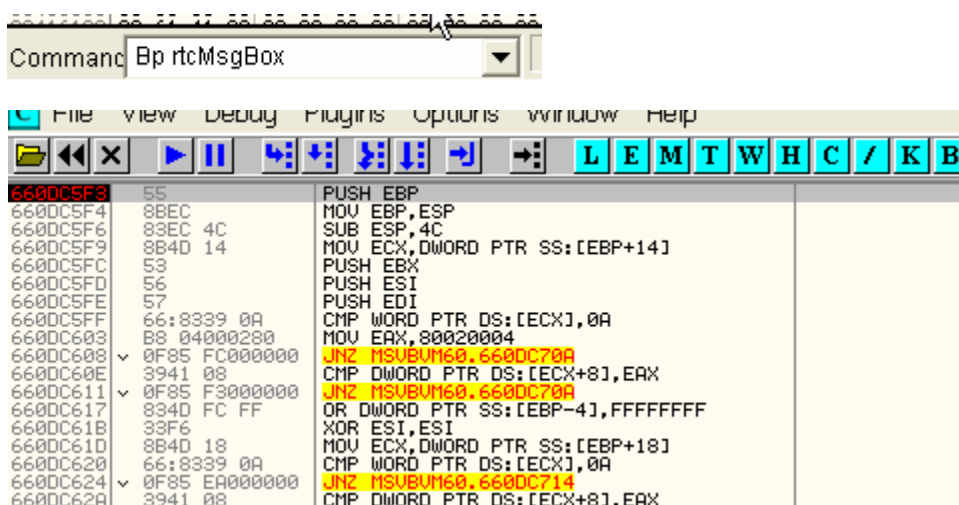
Utilizo antes que nada la api mas usada de comparación __vbaStrCmp



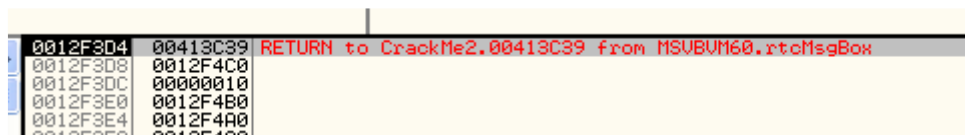
Pongo un BP en ella a ver si encuentro la comparación del serial bueno con el serial malo.

Como para muchísimas veces y tengo fiaca de apretar tantas veces f9, veré si puedo interceptar el cartel de error y de allí llegar a la comparación que debe estar por allí cerca, quito el BP en la comparación y pongo uno en el cartel

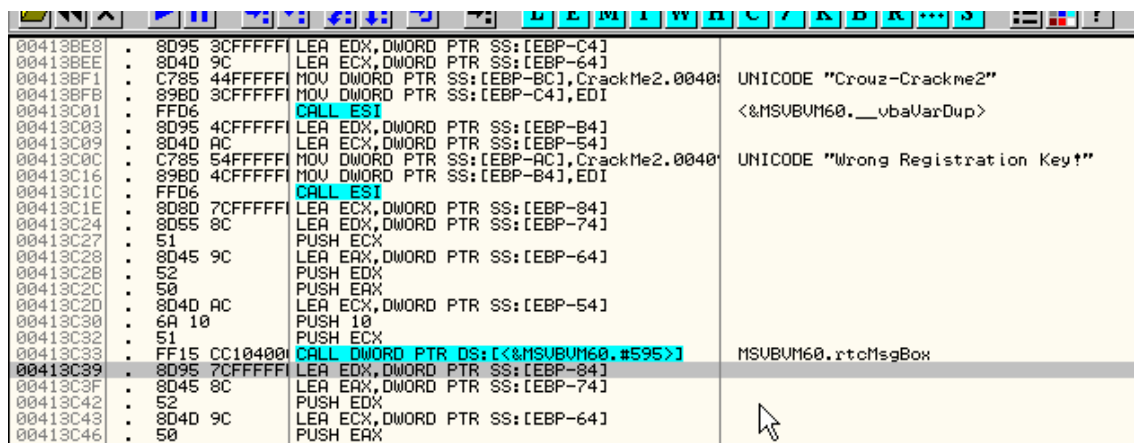
Bp rtcMsgBox



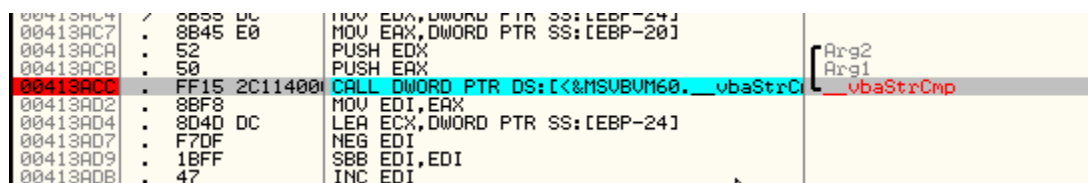
Allí para cuando va a mostrar el cartel de que no acerté el serial, veamos de donde es llamado este call en la primera línea del stack, que es la dirección de retorno de esta api.



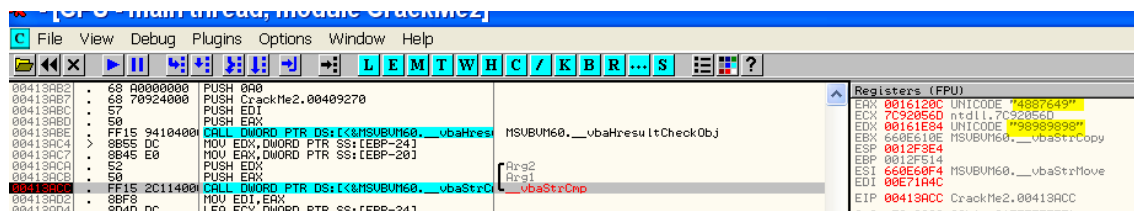
Vayamos allí



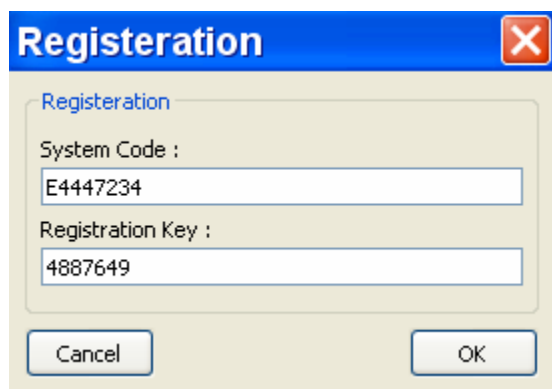
Allí esta donde retornaría de la api justo antes, esta el call a la api y veamos subiendo si hay alguna comparación.

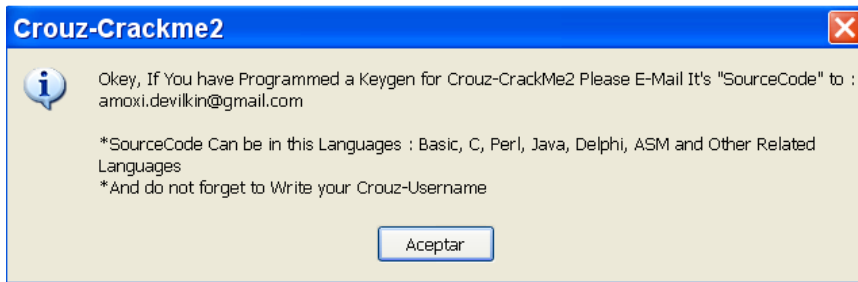


Un poco antes veo esta, pongamos un BP a ver si es la correcta, apreto f9



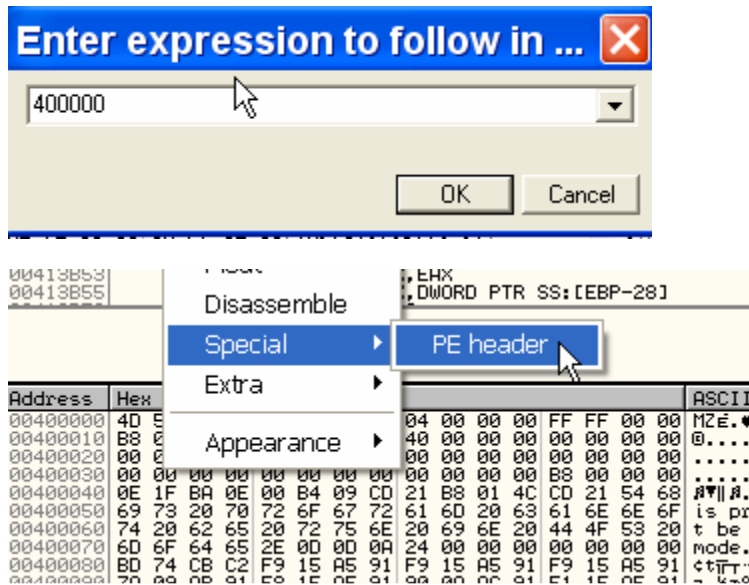
Allí veo que compara mi serial falso con otro numero, en mi caso es 4887649, veamos si es el serial correcto quito todos los BPs y luego de nuevo a la ventana de ingresar serial.





Ahí esta era el serial correcto, el keygen lo dejamos para mas adelante.

Lo primero para quitar la nag, es habilitar la posibilidad de modificar el crackme, o sea darle permiso de escritura a la sección code del mismo para eso vamos al header que comienza en 400000.



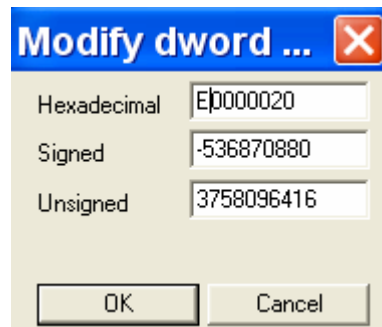
Y lo cambiamos a modo PE HEADER

Address	Hex dump	Data	Comment
004000B5	00	DB 00	
004000B6	00	DB 00	
004000B7	00	DB 00	
004000B8	50 45 00 00	ASCII "PE"	PE signature (PE)
004000BC	4C01	DW 014C	Machine = IMAGE_FILE_MACHINE_I386
004000BE	0300	DW 0003	NumberOfSections = 3
004000C0	297E9142	DD 42917E29	TimeDateStamp = 42917E29
004000C4	00000000	DD 00000000	PointerToSymbolTable = 0
004000C8	00000000	DD 00000000	NumberOfSymbols = 0
004000CC	E000	DW 00E0	SizeOfOptionalHeader = E0 (224.)
004000CE	0F01	DW 010F	Characteristics = EXEQUABLE IMAGE!32B)

Bajamos hasta que comienza la PE SIGNATURE en el header, ahí bajo hasta que veo la primera sección.

Address	Hex dump	Data	Comment
0040019C	00000000	DD 00000000	Delay Import Descriptor size = 0
004001A0	00000000	DD 00000000	COM+ Runtime Header address = 0
004001A4	00000000	DD 00000000	Import Address Table size = 0
004001A8	00000000	DD 00000000	Reserved
004001AC	00000000	DD 00000000	Reserved
004001B0	2E 74 65 71	ASCII ".text"	SECTION
004001B8	78450100	DD 00014578	VirtualSize = 14578 (83320.)
004001BC	00100000	DD 00001000	VirtualAddress = 1000
004001C0	00500100	DD 00015000	SizeOfRawData = 15000 (86016.)
004001C4	00100000	DD 00001000	PointerToRawData = 1000
004001C8	00000000	DD 00000000	PointerToRelocations = 0
004001CC	00000000	DD 00000000	PointerToLineNumbers = 0
004001D0	0000	DW 0000	NumberOfRelocations = 0
004001D2	0000	DW 0000	NumberOfLineNumbers = 0
004001D4	20000060	DD 00000020	Characteristics = CODE EXECUTE READ
004001D8	2E 64 61 71	ASCII ".data"	SECTION
004001E0	00100000	DD 00001000	VirtualSize = 1000 (4096)

Allí vemos las características, sabemos que si cambiamos a E0000020 podremos escribir en dicha sección.



Address	Hex dump	Data	Comment
004001A4	00000000	DD 00000000	Import Address Table size = 0
004001A8	00000000	DD 00000000	Reserved
004001AC	00000000	DD 00000000	Reserved
004001B0	2E 74 65 7D	ASCII ".text"	SECTION
004001B8	78450100	DD 00014578	VirtualSize = 14578 (83320.)
004001BC	00010000	DD 00001000	VirtualAddress = 1000
004001C0	00500100	DD 00005000	SizeOfRawData = 15000 (86016.)
004001C4	00100000	DD 00001000	PointerToRawData = 1000
004001C8	00000000	DD 00000000	PointerToRelocations = 0
004001CC	00000000	DD 00000000	PointerToLineNumbers = 0
004001D0	0000	DW 0000	NumberOfRelocations = 0
004001D2	0000	DW 0000	NumberOfLineNumbers = 0
004001D4	200000E0	DD E0000020	Characteristics = CODE EXECUTE READ WRITE
004001D8	2E 64 61 7D	ASCII ".data"	SECTION
004001E0	00180000	DD 00001800	VirtualSize = 1800 (6152.)

Bueno ahora guardamos los cambios.

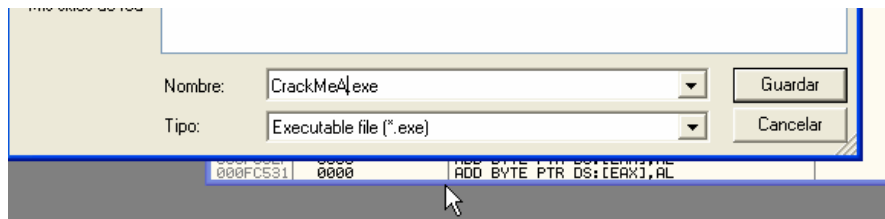
00413B38	68 64984000	PUSH CrackMe2.	Find references
00413B3D	68 58994000	PUSH CrackMe2.	View executable file
00413B42	FFD7	CALL EDI	Copy to executable file
00413B44	8BD0	MOV EDX, EAX	Go to
00413B46	8D4D DC	LEA ECX, DWORD	Hex
00413B49	FFD6	CALL ESI	Text
00413B4B	50	PUSH EAX	Short
00413B4C	68 58994000	PUSH CrackMe2.	Long
00413B51	FFD7	CALL EDI	Float
00413B53	8BD0	MOV EDX, EAX	Disassemble
00413B55	8D4D D8	LEA ECX, DWORD	<input checked="" type="checkbox"/> Special <input type="checkbox"/> Extra

Address	Hex dump	Data
004001A4	00000000	00 00000000
004001A8	00000000	00 00000000
004001AC	00000000	00 00000000
004001B0	2E 74 65	7 ASCII ".text"
004001B3	78450100	00 00014578
004001B8	00100000	00 00010000
004001C0	00500100	00 00015000
004001C4	00100000	00 00010000
004001C8	00000000	00 00000000
004001CC	00000000	00 00000000
004001D0	0000	00 0000
004001D2	0000	00 0000
004001D4	200000E0	00 50000020
004001D8	2E 64 61	7 ASCII ".data"
004001E0	00180000	00 00001800

SECTION
 VirtualSize = 1000 (6152)

The screenshot shows a debugger window with the following content:

- Assembly View:**
 - Address: 000010D4, Disassembly: DD EB000020, Comment: Characteristic = CODE_EXECUTE
 - Address: 000010D8, Disassembly: 2E 64 61 7, Comment: ASCII ".data", Section: SECTION
 - Address: 000010E0, Disassembly: 08180000, Comment: 00001800, VirtualSize = 1800 (6152.)
 - Address: 000010E4, Disassembly: 00000100, Comment: 00016000, VirtualAddress = 16000
 - Address: 000010E8, Disassembly: 08180000, Comment: 00001800, SizeOfRawData = 1800 (4096.)
 - Address: 000010EC, Disassembly: 00000100, Comment: 00016000, PointerToRawData = 16000
 - Address: 000010F0, Disassembly: 00000000, Comment: 00000000, PointerToRelocations = 0
 - Address: 000010F4, Disassembly: 00000000, Comment: 00000000, PointerToLineNumbers = 0
 - Address: 000010F8, Disassembly: 0000, Comment: 0000, NumberOfR... (truncated)
 - Address: 000010FC, Disassembly: 0000, Comment: 0000, Character (truncated)
 - Address: 000010FC, Disassembly: 400000C0, Comment: C0000040, Character (truncated)
 - Address: 00001000, Disassembly: 2E 72 73, Comment: ASCII ".rsrc", Section: SECTION
 - Address: 00002008, Disassembly: 1C2E0000, Comment: 0000021C, VirtualSize (truncated)
 - Address: 0000200C, Disassembly: 00000100, Comment: 00018000, VirtualAd... (truncated)
 - Address: 00002010, Disassembly: 00000000, Comment: 00000000, SizeOfRaw... (truncated)
 - Address: 00002014, Disassembly: 00700100, Comment: 00017000, PointerTo... (truncated)
 - Address: 00002018, Disassembly: 00000000, Comment: 00000000, PointerTo... (truncated)
 - Address: 0000201C, Disassembly: 00000000, Comment: 00000000, PointerTo... (truncated)
 - Address: 00002020, Disassembly: 0000, Comment: 0000, NumberOfR... (truncated)
 - Address: 00002022, Disassembly: 0000, Comment: 0000, NumberOfR... (truncated)
- Instruction List:**
 - 0B06: 994D 8C MOV DWORD PTR SS:[EBP-74],ECX
 - 0B09: 9F84 C80000 JE CrackMe2.00413BDD
 - 0B0F: 8D95 4CFFFFFF LEA EDI, DWORD PTR SS:[EBP-B4]
- Search Results:**
 - Save file (highlighted)
 - Go to offset

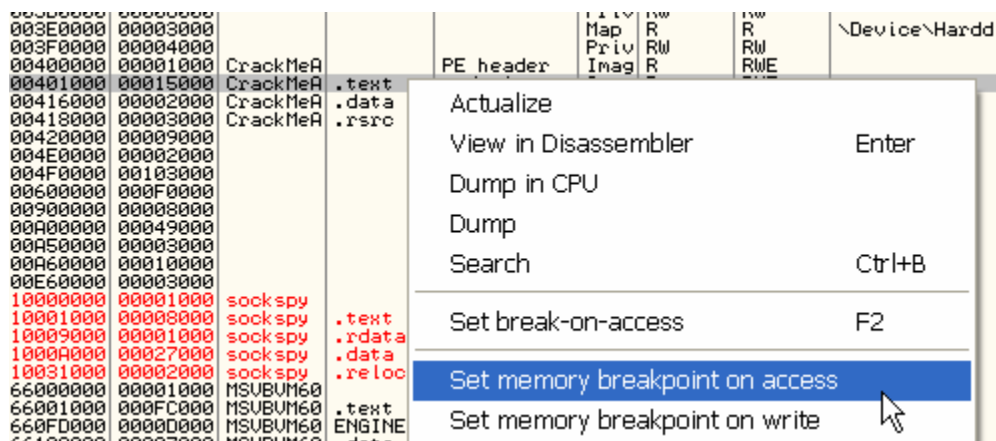


Lo guardo como CrackmeA.exe

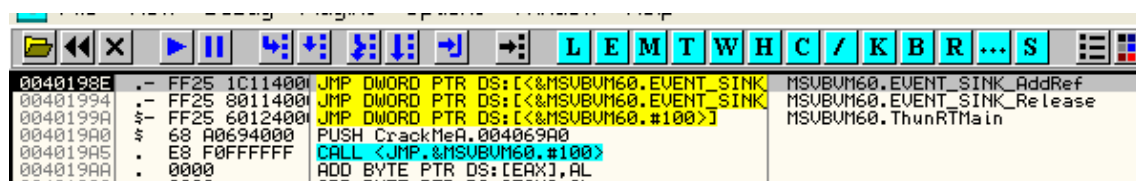


Bueno abro el CrackmeA en el parcheado 5 que es el OLLYDBG modificado para OEPs y VISUAL BASIC.

Pongo un BPM ON ACCESS (que sera on execution) en la seccion CODE.



Demos RUN varias veces para aquí.



Seguimos hasta que paremos en los conocidos JMPS que nos desvían a las diferentes parte del programa.

Address	Disassembly	Comment
00407710	SUB DWORD PTR SS:[ESP+41],3B	
00407718	JMP CrackMeA.0040BD80	
0040771D	SUB DWORD PTR SS:[ESP+41],43	
00407725	JMP CrackMeA.0040BDF0	
0040772A	SUB DWORD PTR SS:[ESP+41],47	
00407732	JMP CrackMeA.0040BEE0	
00407737	SUB DWORD PTR SS:[ESP+41],37	
0040773F	JMP CrackMeA.0040BF50	
00407744	SUB DWORD PTR SS:[ESP+41],0FFFF	
0040774C	JMP CrackMeA.0040C2E0	
00407751	SUB DWORD PTR SS:[ESP+41],3B	
00407759	JMP CrackMeA.0040C470	
0040775E	DB 00	
0040775F	DB 00	

Vemos que la primera vez, para en el primer JMP, y salta a 40bd80, o sea que allí se ejecuta la primera parte del programa, veamos si aparece la nag, antes de volver a los saltos, quitamos el BPM ON ACCESS ya que ahora ejecuta una larga parte en la sección code y si no nos volverá loco parando en cada línea, lo que hacemos es poner BPs en los distintos saltos a las diferentes partes del programa.

Address	Disassembly	Comment
0040770E	DB 00	
0040770F	DB 00	
00407710	SUB DWORD PTR SS:[ESP+41],3B	
00407718	JMP CrackMeA.0040BD80	
0040771D	SUB DWORD PTR SS:[ESP+41],43	
00407725	JMP CrackMeA.0040BDF0	
0040772A	SUB DWORD PTR SS:[ESP+41],47	
00407732	JMP CrackMeA.0040BEE0	
00407737	SUB DWORD PTR SS:[ESP+41],37	
0040773F	JMP CrackMeA.0040BF50	
00407744	SUB DWORD PTR SS:[ESP+41],0FFFF	
0040774C	JMP CrackMeA.0040C2E0	
00407751	SUB DWORD PTR SS:[ESP+41],3B	
00407759	JMP CrackMeA.0040C470	
0040775E	DB 00	

Veamos si sale la nag o si vuelve a otro JMP antes, demos RUN

Address	Disassembly	Comment
0040770E	DB 00	
0040770F	DB 00	
00407710	SUB DWORD PTR SS:[ESP+41],3B	
00407718	JMP CrackMeA.0040BD80	
0040771D	SUB DWORD PTR SS:[ESP+41],43	
00407725	JMP CrackMeA.0040BDF0	
0040772A	SUB DWORD PTR SS:[ESP+41],47	
00407732	JMP CrackMeA.0040BEE0	
00407737	SUB DWORD PTR SS:[ESP+41],37	
0040773F	JMP CrackMeA.0040BF50	
00407744	SUB DWORD PTR SS:[ESP+41],0FFFF	
0040774C	JMP CrackMeA.0040C2E0	
00407751	SUB DWORD PTR SS:[ESP+41],3B	
00407759	JMP CrackMeA.0040C470	
0040775E	DB 00	

Ahora para en el último salto que va a 40c470, sin haber aparecido la nag.

Demos RUN nuevamente



Allí apareció la nag quiere decir que esta parte que se esta ejecutando que empezó en 40c470 es la responsable de la nag, apretemos el botón register.

0040770F	00	DB 00
00407710	816C24 04 3B	SUB DWORD PTR SS:[ESP+4],3B
00407718	E9 63460000	JMP CrackMeA.0040B080
0040771D	816C24 04 43	SUB DWORD PTR SS:[ESP+4],43
00407725	E9 C6460000	JMP CrackMeA.0040B0F0
0040772A	816C24 04 47	SUB DWORD PTR SS:[ESP+4],47
00407732	E9 A9470000	JMP CrackMeA.0040BEE0
00407737	816C24 04 37	SUB DWORD PTR SS:[ESP+4],37
0040773F	E9 0C480000	JMP CrackMeA.0040BF50
00407744	816C24 04 FF	SUB DWORD PTR SS:[ESP+4],0FFFF
0040774C	E9 8F4B0000	JMP CrackMeA.0040C2E0
00407751	816C24 04 3B	SUB DWORD PTR SS:[ESP+4],3B
00407759	E9 124D0000	JMP CrackMeA.0040C470
0040775E	00	DB 00
0040775F	00	DB 00
00407760	01	DB 01
00407761	00	DB 00
00407762	04	DB 04

Al volver de la nag para en el segundo JMP que supuestamente ya iría al inicio del programa, así que podemos probar que ocurre si al JMP anterior que hace salir la nag lo desviamos a 40bdf0, que ocurre, saltara la nag?

0040770F	00	DB 00
00407710	816C24 04 3B	SUB DWORD PTR SS:[ESP+4],3B
00407718	E9 63460000	JMP CrackMeA.0040B080
0040771D	816C24 04 43	SUB DWORD PTR SS:[ESP+4],43
00407725	E9 C6460000	JMP CrackMeA.0040B0F0
0040772A	816C24 04 47	SUB DWORD PTR SS:[ESP+4],47
00407732	E9 A9470000	JMP CrackMeA.0040BEE0
00407737	816C24 04 37	SUB DWORD PTR SS:[ESP+4],37
0040773F	E9 0C480000	JMP CrackMeA.0040BF50
00407744	816C24 04 FF	SUB DWORD PTR SS:[ESP+4],0FFFF
0040774C	E9 8F4B0000	JMP CrackMeA.0040C2E0
00407751	816C24 04 3B	SUB DWORD PTR SS:[ESP+4],3B
00407759	E9 92460000	JMP CrackMeA.0040BDF0
0040775E	00	DB 00
0040775F	00	DB 00
00407760	01	DB 01
00407761	00	DB 00

Allí esta cambie el segundo salto que es el que va a la parte que hace aparecer la nag, por el tercero que debería arrancar el programa guardemos los cambios y veamos que pasa.

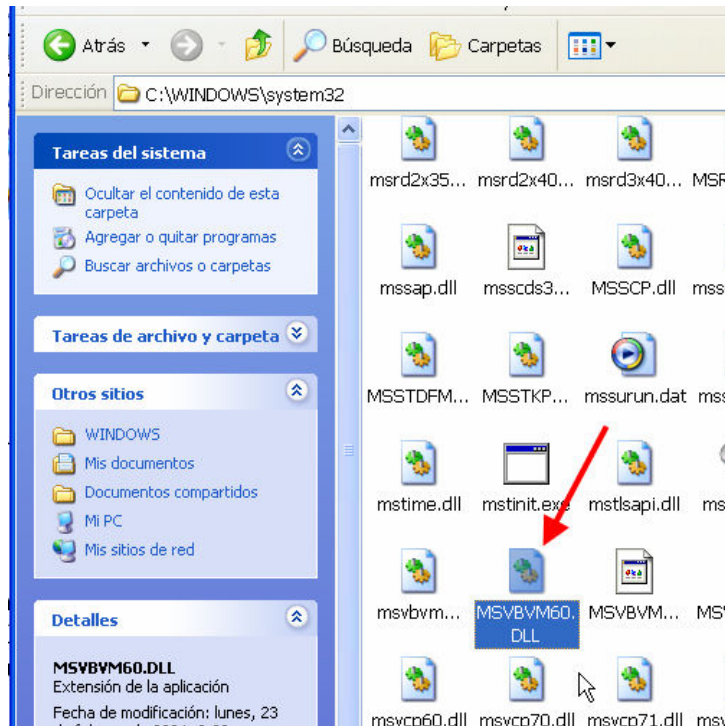
Corramos el CrackmeA ya modificado fuera de OLLY



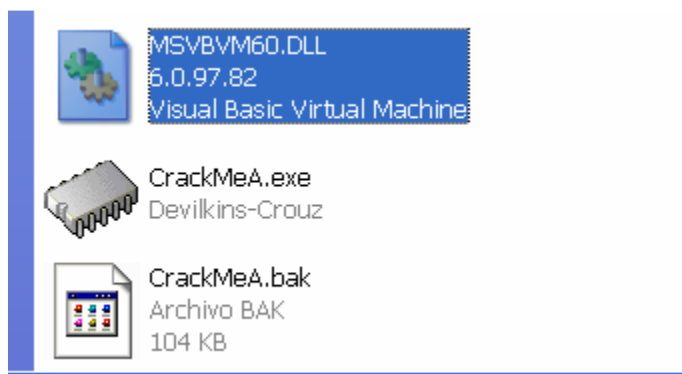
Vemos que nuestra modificación mejoro algo el problema, pero aun no esta terminado o sea, no sale la nag inicial que tenes que apretar el botón register para que aparezca la ventana para ingresar el serial falso, si no que aparecen las dos juntas, adelante la nag, tapando la otra, eso es un gran paso, porque si antes hacíamos a la nag invisible, la otra no aparecería al no haber apretado el botón register, ahora al menos solo nos queda hacer invisible la nag pues la otra ya apareció debajo y quedara sola, visible y funcional, al anular la que la tapa.

Haciendo intentos con los demás JMPs no mejora la cosa, así que no queda otra que injertar. Ahora completemos la batalla, muchos crackers dicen que no corresponde modificar la dll de visual, eso podía ser cierto cuando hablamos de la que se encuentra en system32 y es usada por todos los programas, de forma que si la modificamos, puede afectar a otros programas, pero nosotros no haremos eso, si no que la copiaremos a la carpeta del crackme, de forma que esa solo será usada por el mismo, será una dll especial para el, los restantes programas de mi maquina continuaran usando, la que se encuentra en system32 por supuesto, o si tienen alguna en su propia carpeta pues usaran antes esa.

La cuestión que copiamos la dll de visual que se llama.



A la carpeta del programa

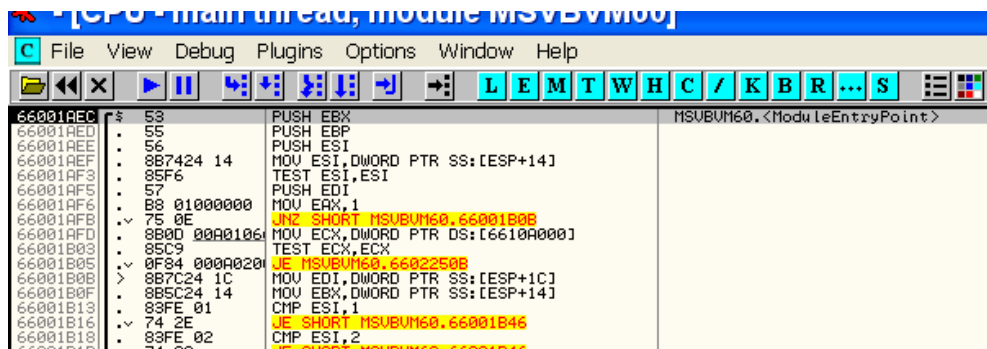
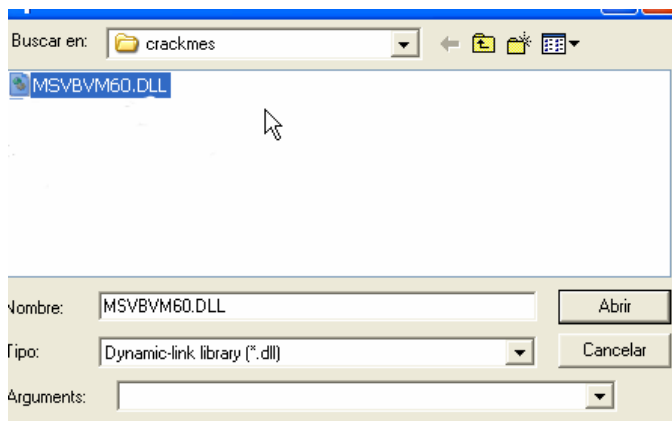


Allí esta junto al CrackmeA, esto puedo realizarlo porque la dll de visual no es una dll de sistema, en el caso que sea una dll de sistema, necesitare cambiar algunas cosas del programa para que acepte una dll en su carpeta.

O sea que cualquier programa que usa una cierta dll que no sea de sistema, primero busca en su carpeta y si no la halla, la busca en system32, en este caso al haber una en su propia carpeta usa esa, lo cual es muy bueno para nosotros jeje ya que nos permite modificarla sin que afecte a otros programas.

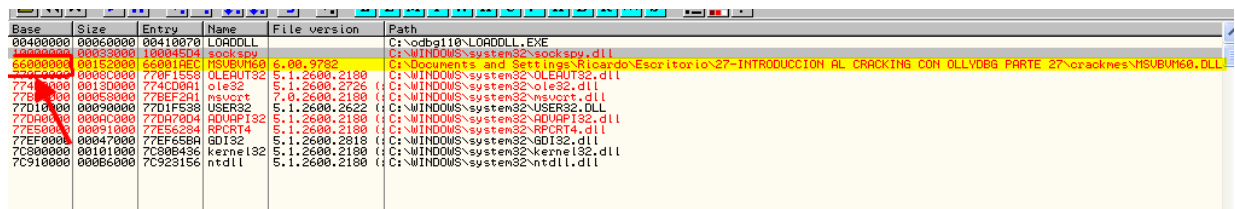
Arranquemos el CrackmeA nuevamente en OLLYDBG, en este caso usamos un OLLYDBG común ya que para escribir injertos o líneas de código a veces los parcheados fallan.

Dynamic-link library (*.dll) con lo cual podremos abrir la dll de Visual.



Allí esta parado en su Entry Point.

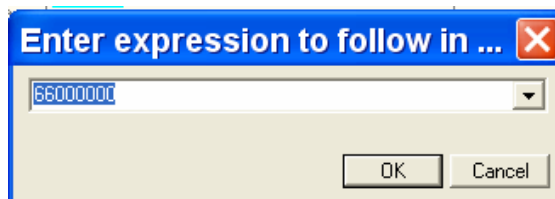
Bueno ahora debemos buscar el header que en este caso no estará ubicado en 400000 como normalmente, miremos la imagebase del archivo que es donde se inicia, apretando el botón E.

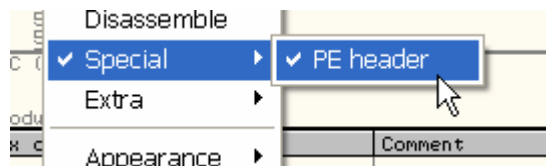


Base	Size	Entry	Name	File version	Path
00400000	00000000	00410070	LOADDLL		C:\odbg110\LOADDLL.EXE
00000000	00000000	00004504	sockspy		C:\WINDOWS\system32\sockspy.dll
66000000	00152000	66001AEC	MSVBVM60	6.00.9792	C:\Documents and Settings\Ricardo\Escritorio\27-INTRODUCCION AL CRACKING CON OLLYDBG PARTE 27\crackmes\MSVBVM60.DLL
00000000	00000000	770F1553	OLEAUT32	5.1.2600.2180	C:\WINDOWS\system32\OLEAUT32.dll
77400000	00130000	774CD0A1	ole32	5.1.2600.2726	C:\WINDOWS\system32\ole32.dll
77000000	00058000	77BEF2A1	msvcrt	7.0.2600.2180	C:\WINDOWS\system32\msvcrt.dll
77010000	00090000	77D1F538	USER32	5.1.2600.2622	C:\WINDOWS\system32\USER32.dll
770A0000	0006C000	77DA70D4	ADUAP132	5.1.2600.2180	C:\WINDOWS\system32\ADUAP132.dll
77E50000	00091000	77E56284	RPCRT4	5.1.2600.2180	C:\WINDOWS\system32\RPCRT4.dll
77EF0000	00047000	77EF65B9	GDI32	5.1.2600.2019	C:\WINDOWS\system32\GDI32.dll
7C800000	00101000	7C80B436	kernel32	5.1.2600.2180	C:\WINDOWS\system32\kernel32.dll
7C910000	000B6000	7C923156	ntdll	5.1.2600.2180	C:\WINDOWS\system32\ntdll.dll

Allí en la columna base encontramos la imagebase de la dll que es 66000000 en mi caso, en sus maquinas puede variar.

Vayamos en el dump a ver el header





Y cambiemos al modo Special- PE HEADER repitiendo el procedimiento que hicimos en el crackme, bajamos buscando la sección CODE.

MSUBUM60, <ModuleEntryPoint>

Address	Hex dump	Data	Comment
66000164	00000000	DD 00000000	Delay Import Descriptor size = 0
66000168	00000000	DD 00000000	COM+ Runtime Header address = 0
6600016C	00000000	DD 00000000	Import Address Table size = 0
66000170	00000000	DD 00000000	Reserved
66000174	00000000	DD 00000000	Reserved
66000178	2E 74 65 74	ASCII ".text"	SECTION
66000180	DAB20F00	DD 000FB2DA	VirtualSize = FB2DA (1028826.)
66000184	00100000	DD 00001000	VirtualAddress = 1000
66000188	00C00F00	DD 000FC000	SizeOfRawData = FC000 (1032192.)
6600018C	00100000	DD 00001000	PointerToRawData = 1000
66000190	00000000	DD 00000000	PointerToRelocations = 0
66000194	00000000	DD 00000000	PointerToLineNumbers = 0
66000198	0000	DW 0000	NumberOfRelocations = 0
6600019A	0000	DW 0000	NumberOfLineNumbers = 0
6600019C	20000060	DD 60000020	Characteristics = CODE EXECUTE READ
660001A0	45 4E 47 47	ASCII "ENGINE"	SECTION
660001A8	F5CD0000	DD 0000CDF5	VirtualSize = CDF5 (52725.)

Allí cambiamos a E0000020 para que tenga permiso de escritura.

Modify dword ...

Hexadecimal: E0000020

Signed: -536870880

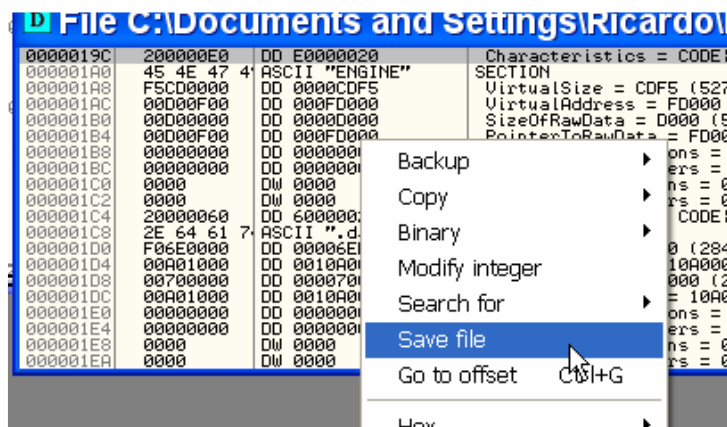
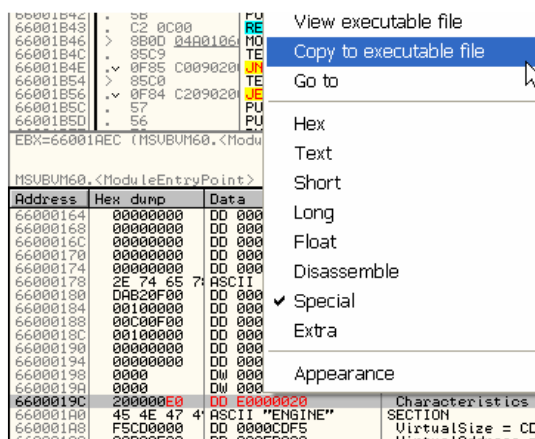
Unsigned: 3758096416

OK Cancel

MSUBUM60, <ModuleEntryPoint>

Address	Hex dump	Data	Comment
66000164	00000000	DD 00000000	Delay Import Descriptor size = 0
66000168	00000000	DD 00000000	COM+ Runtime Header address = 0
6600016C	00000000	DD 00000000	Import Address Table size = 0
66000170	00000000	DD 00000000	Reserved
66000174	00000000	DD 00000000	Reserved
66000178	2E 74 65 74	ASCII ".text"	SECTION
66000180	DAB20F00	DD 000FB2DA	VirtualSize = FB2DA (1028826.)
66000184	00100000	DD 00001000	VirtualAddress = 1000
66000188	00C00F00	DD 000FC000	SizeOfRawData = FC000 (1032192.)
6600018C	00100000	DD 00001000	PointerToRawData = 1000
66000190	00000000	DD 00000000	PointerToRelocations = 0
66000194	00000000	DD 00000000	PointerToLineNumbers = 0
66000198	0000	DW 0000	NumberOfRelocations = 0
6600019A	0000	DW 0000	NumberOfLineNumbers = 0
6600019C	200000E0	DD E0000020	Characteristics = CODE EXECUTE READ WRITE
660001A0	45 4E 47 47	ASCII "ENGINE"	SECTION
660001A8	F5CD0000	DD 0000CDF5	VirtualSize = CDF5 (52725.)
660001AC	00000000	DD 00000000	VirtualAddress = F0000

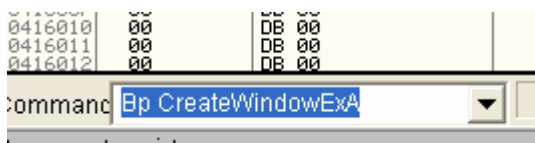
Ahora guardamos los cambios como sabemos.



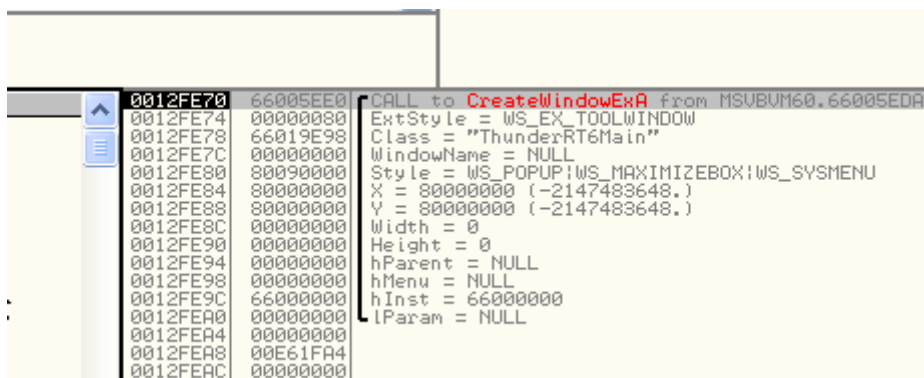
Aquí no le cambiamos el nombre lo guardamos con el mismo nombre si no el crackme no lo cargara

Una vez que ya tenemos ambos el crackme y la dll con posibilidad de escribir en ellos, antes de realizar el injerto, mostrare cual es la idea de lo que tenemos que hacer para que no aparezca la nag.

Abramos en OLLYDBG el crackme que ya teníamos parcheado, para que aparezca la nag encima de la ventana del serial, y pongamos un BP en la api de creación de ventanas Bp CreateWindowExA



Ahora arranco el crackme



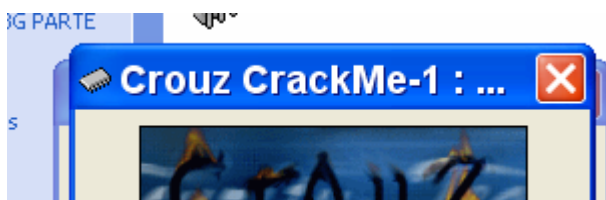
Parara varias veces cuando va creando las diferentes ventanas y botones que tiene el crackme, lleguemos hasta cuando crea la nag.

```

0012FAB0 6605A8DE CALL to CreateWindowExA from MSUBUM60.6605A8D8
0012FAB4 00040000 ExtStyle = WS_EX_APPWINDOW
0012FAB8 0000C28C Class = C28C
0012FABC 00E70740 WindowName = "Crouz CrackMe-1 : Setup"
0012FAC0 02C80000 Style = WS_OVERLAPPED!WS_CLIPCHILDREN!WS_SYSMENU!WS_CAPTION
0012FAC4 00000000 X = 0
0012FAC8 FFFFFFFC Y = FFFFFFFC (-4.)
0012FACC 00000107 Width = 107 (263.)
0012FAD0 000000D6 Height = D6 (214.)
0012FAD4 005E067E hParent = 005E067E ('CrouzCrackMe2',class='ThunderRT6Main')
0012FAD8 00000000 hMenu = NULL
0012FADC 66000000 hInst = 66000000
0012FAE0 00000000 lParam = NULL
0012FAE4 00E65888
0012FAE8 00000000

```

Aquí esta la localizamos fácilmente por el nombre, allí en WindowName dice el titulo de la nag recordemos que era este



Bueno allí se esta creando la nag, si cambio el style o estilo del mismo, veamos que pasa intentando algunos valores veo que si pongo 40000000

```

0012FAB0 6605A8DE CALL to CreateWindowExA from MSUBUM60.6605A8D8
0012FAB4 00040000 ExtStyle = WS_EX_APPWINDOW
0012FAB8 0000C28C Class = C28C
0012FABC 00E70740 WindowName = "Crouz CrackMe-1 : Setup"
0012FAC0 02C80000 Style = WS_OVERLAPPED!WS_CLIPCHILDREN!WS_SYSMENU!WS_CAPTION
0012FAC4 00000000 X = 0
0012FAC8 FFFFFFFC Y = FFFFFFFC (-4.)
0012FACC 00000107 Width = 107 (263.)
0012FAD0 000000D6 Height = D6 (214.)
0012FAD4 005E067E hParent = 005E067E ('CrouzCrackMe2',class='ThunderRT6Main')
0012FAD8 00000000 hMenu = NULL
0012FADC 66000000 hInst = 66000000
0012FAE0 00000000 lParam = NULL
0012FAE4 00E65888
0012FAE8 00000000

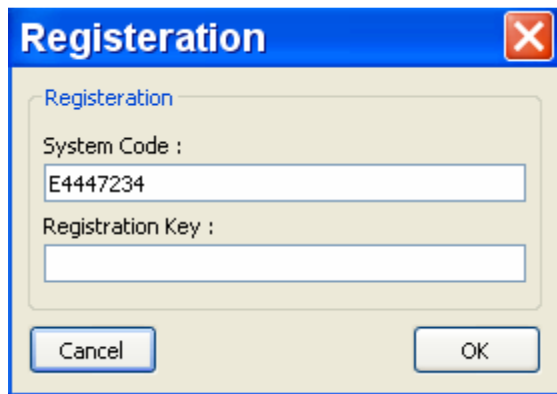
```

```

0012FAB0 6605A8DE CALL to CreateWindowExA from MSUBUM60.6605A8D8
0012FAB4 00040000 ExtStyle = WS_EX_APPWINDOW
0012FAB8 0000C28C Class = C28C
0012FABC 00E70740 WindowName = "Crouz CrackMe-1 : Setup"
0012FAC0 40000000 Style = WS_CHILD
0012FAC4 00000000 X = 0
0012FAC8 FFFFFFFC Y = FFFFFFFC (-4.)
0012FACC 00000107 Width = 107 (263.)
0012FAD0 000000D6 Height = D6 (214.)
0012FAD4 005E067E hParent = 005E067E ('CrouzCrackMe2',class='ThunderRT6Main')
0012FAD8 00000000 hMenu = NULL
0012FADC 66000000 hInst = 66000000
0012FAE0 00000000 lParam = NULL
0012FAE4 00E65888
0012FAE8 00000000

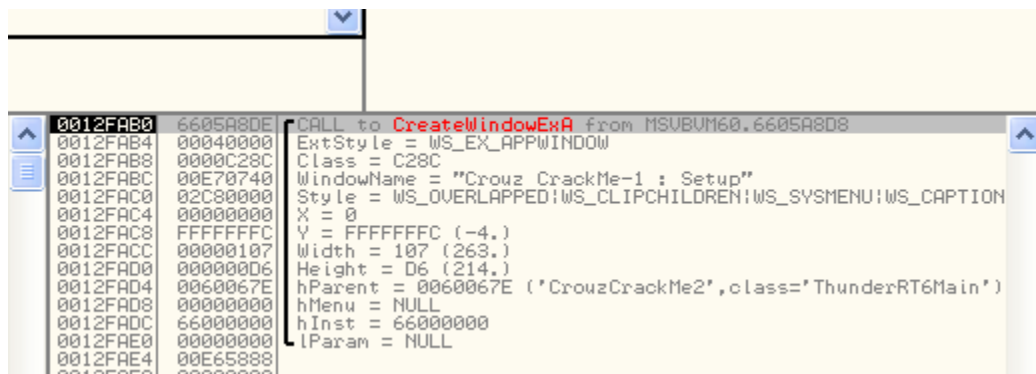
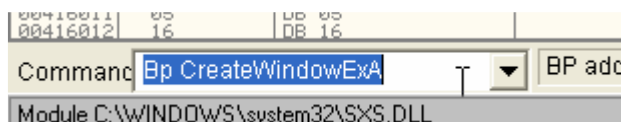
```

Cambia el estilo a WS CHILD ustedes pueden experimentar poniendo diferentes valores, a ver que pasa, ahora quito todos los BP y doy RUN.

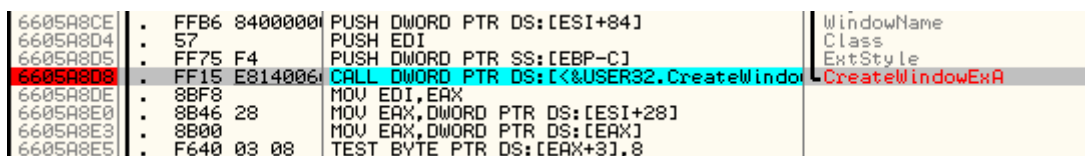


Vemos que sale sola la ventana de registro y la nag no aparece, el tema que injertar en VB es complejo si no cambias la dll, pero si podemos hacerlo y no afectamos ningún otro programa, porque vamos a boxear con las manos atadas, jeje.

Bueno repitamos el procedimiento, hasta parar cuando crea la ventana en la api a injertar CreateWindowExA

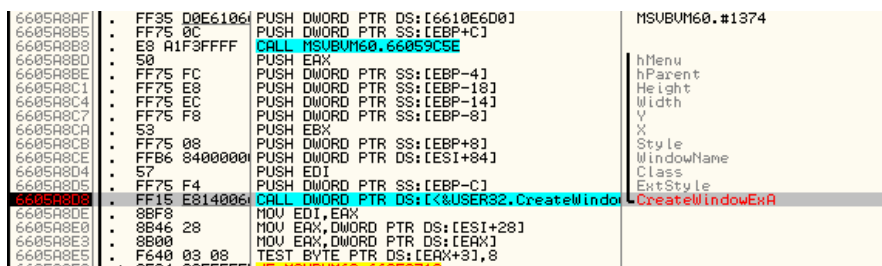


Vemos que la api es llamada desde la dll de Visual Basic, que mejor que injertar allí, veamos de donde viene llamada, la primera línea del stack nos indica de donde se llamo a la api, vayamos allí.

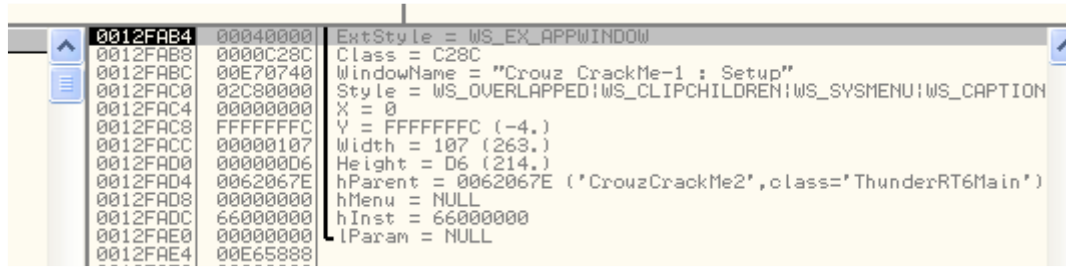


Pongamos un BP allí y quitemos el de la api.

Ahora veamos cuando para, reiniciemos y demos RUN



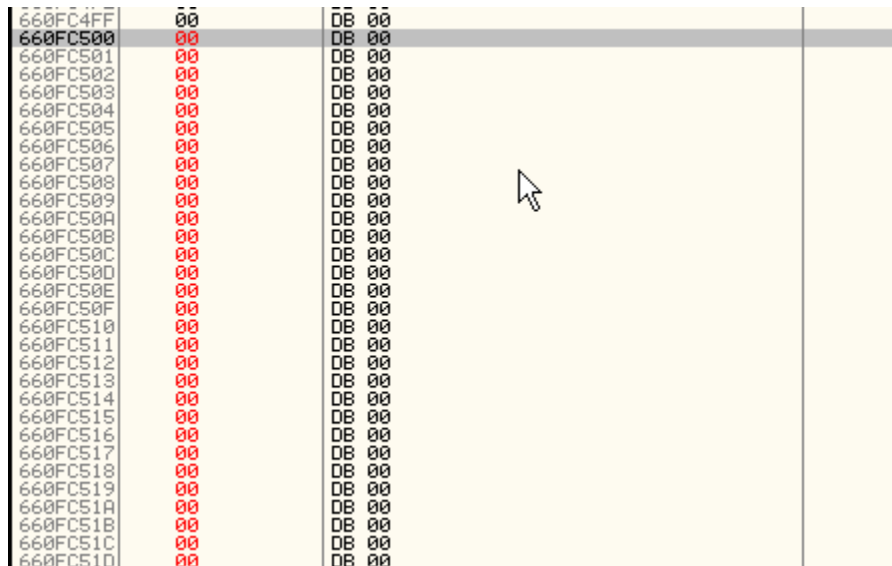
Allí paro y vemos en el stack



Vemos que la primera vez que para es cuando crea la nag, ya que las veces anteriores que paro en la api, fue llamado desde otras direcciones no de aquí.

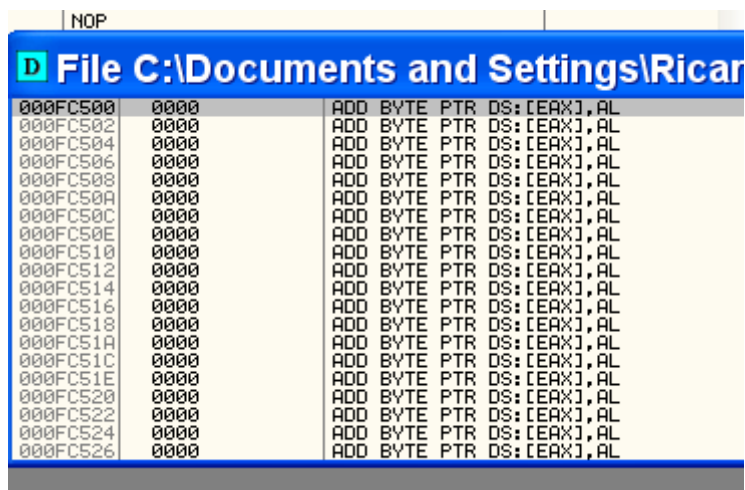
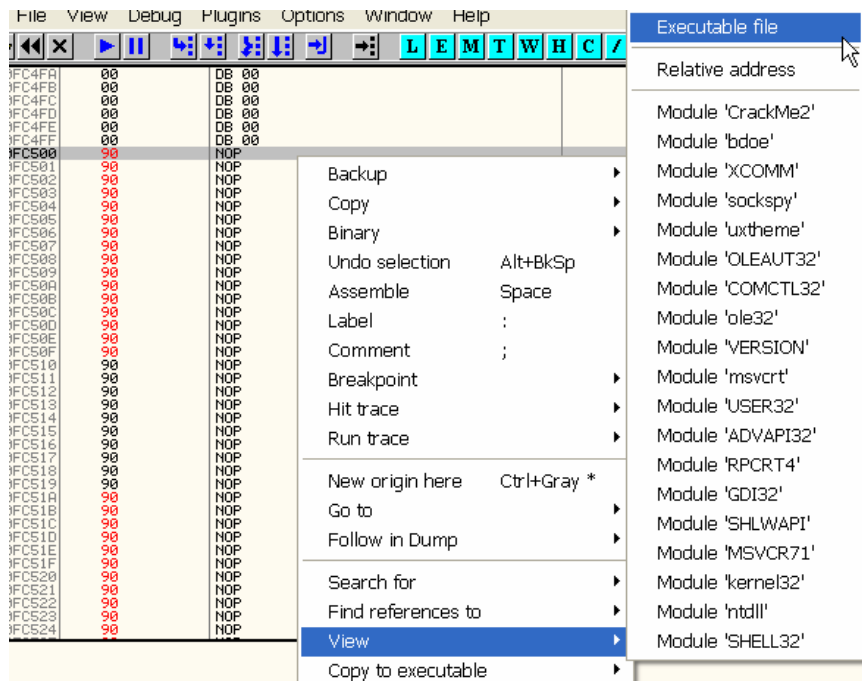
Lamentablemente no es la única vez que pasa por aquí, cuando crea la ventana de registro también lo hace desde aquí, así que debemos ser cuidadosos y hacer un injerto selectivo.

Todo injerto se inicia con un JMP a una zona vacía donde se pueda escribir, si busco al final de la sección, veo que hay una zona vacía en



En mi maquina esta allí en la suya estará al final de la seccion code, ven que hay muchos ceros, lo primero que haré será verificar si sirve.

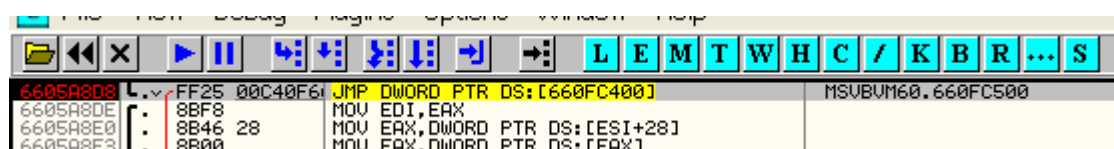
Hago click derecho – VIEW EXECUTABLE para ver la zona, la cual si aparece, me habilita para guardar los cambios en el exe.



Veo que la zona aparece en el ejecutable.

Bueno esto es importante al hacer un injerto, verificar si se puede guardar los cambios en la zona que elegimos porque muchas veces hay partes de una sección que son ceros, pero existen solamente en memoria, y no en el ejecutable, por lo cual no te deja guardar cambios en el exe (ya lo explicaremos mejor cuando veamos desempacado)

.



Primero en donde estaba el call a la api hago un salto al injerto,

```

File View Debug Plugins Options Window Help
[Icons]
6605A8D8 FF25 00C40F61 JMP DWORD PTR DS:[660FC400] MSVBUM60.660FC500
6605A8DE 8BF8 MOV EDI,EAX
6605A8E0 8B46 28 MOV EAX,DWORD PTR DS:[ESI+28]
6605A8E3 8B00 MOV EAX,DWORD PTR DS:[EAX]
6605A8E5 F640 03 08 TEST BYTE PTR DS:[EAX+3],8

```

Lo hago un salto indirecto porque cabe justo sin modificar la siguiente línea luego del call, que era un MOV EDI, EAX, siempre me tengo que fijar que al agregar código no rompa las instrucciones que siguen, y se mantenga la continuidad del programa, si sobrescribo algún byte de MOV EDI, EAX dará error allí al retornar del injerto.

Hice un salto indirecto que toma la dirección donde saltara de 660fc400 que es una dirección un poco anterior a mi injerto donde guardo la dirección de inicio del mismo, podría haber hecho un salto directo, pero bueno, ambas posibilidades son validas (ya lo hice así y me da fiaca cambiarlo si quieren pueen poner JMP 660fc500, jeje)

Address	Hex dump	ASCII
660FC400	00 C5 0F 66 00 00 00 00	.+*f....
660FC408	00 00 00 00 00 00 00 00
660FC410	00 00 00 00 00 00 00 00
660FC418	00 00 00 00 00 00 00 00
660FC420	00 00 00 00 00 00 00 00
660FC428	00 00 00 00 00 00 00 00
660FC430	00 00 00 00 00 00 00 00
660FC438	00 00 00 00 00 00 00 00

Como ven allí guardo la dirección de inicio del injerto, en resumidas cuentas cuando llegue al JMP saltara allí.

0012FAB4	00040000	
0012FAB8	0000C28C	
0012FABC	00E70740	ASCII "Crowz CrackMe-1 : Setup"
0012FAC0	02C80000	
0012FAC4	00000000	
0012FAC8	FFFFFFFF	
0012FACC	00000107	
0012FAD0	00000006	
0012FAD4	00780666	
0012FAD8	00000000	
0012FADC	66000000	OFFSET MSVBUM60.#1374
0012FAE0	00000000	
0012FAE4	00E65888	
0012FAE8	00000000	
0012FAEC	00000000	

Por supuesto en el stack aunque no estén marcados, están los parámetros de la api, allí se ve el nombre de la ventana.

Debemos chequear primero que exista un nombre porque a veces llega aquí con el nombre puesto a cero, y si no chequeamos eso dará error.

0012F9B4	00040000	
0012F9B8	0000C28C	
0012F9BC	00E70740	ASCII "Crowz CrackMe-1 : Setup"
0012F9C0	02C80000	
0012F9C4	00000000	
0012F9C8	FFFFFFFF	
0012F9CC	00000107	
0012F9D0	00000006	
0012F9D4	00780666	

Si hago doble click allí en el stack veo que cambia

\$ ==>	00040000	
\$+4	0000C28C	
\$+8	00E70740	ASCII "Crowz CrackMe-1 : Setup"
\$+C	02C80000	
\$+10	00000000	

Eso quiere decir que la primera línea es ESP, la segunda ESP+4 y así, vemos que la que nos interesa es ESP+8, movemos el valor ese a EAX.

660FC500	8B4424 08	MOV EAX, DWORD PTR SS:[ESP+8]	
660FC504	85C0	TEST EAX, EAX	
660FC506	74 1A	JE SHORT MSUBUM60.660FC522	
660FC508	8138 43726F7	CMP DWORD PTR DS:[EAX], 756F7243	
660FC50E	75 12	JNZ SHORT MSUBUM60.660FC522	
660FC510	90	NOP	
660FC511	90	NOP	
660FC512	90	NOP	
660FC513	90	NOP	
660FC514	90	NOP	
660FC515	90	NOP	
660FC516	90	NOP	
660FC517	90	NOP	
660FC518	90	NOP	
660FC519	90	NOP	
660FC51A	C74424 0C 00	MOV DWORD PTR SS:[ESP+C], 40000000	
660FC522	FF15 E814006	CALL DWORD PTR DS:[<&USER32.CreateWindowExA], USER32.CreateWindowExA	
660FC528	E9 B1E3F5FF	JMP MSUBUM60.6605A8DE	
660FC52D	0000	ADD BYTE PTR DS:[EAX], AL	
660FC52F	0000	ADD BYTE PTR DS:[EAX], AL	
660FC531	0000	ADD BYTE PTR DS:[EAX], AL	
660FC533	0000	ADD BYTE PTR DS:[EAX], AL	
660FC535	00	DB 00	
660FC536	00	DB 00	
660FC537	00	DB 00	
660FC538	00	DB 00	

Voy traceando el injerto para que entiendan como funciona, al ejecutar esa línea en EAX queda

Registers (FPU)	
EAX	00E70740 ASCII "Crouz CrackMe-1 : Setup"
ECX	6610DAC0 MSUBUM60.6610DAC0
EDX	00000007
EBX	00000000
ESP	0012FAB4
EBP	0012FBE0
ESI	00E704DC
EDI	0000C28C
EIP	660FC504 MSUBUM60.660FC504
CS	0023 32bit 0 (FFFFFFFF)

El puntero a la string del título, luego la siguiente línea testea si EAX es cero, lo cual ocurre en el caso de ventanas sin título.

660FC500	8B4424 08	MOV EAX, DWORD PTR SS:[ESP+8]	
660FC504	85C0	TEST EAX, EAX	
660FC506	74 1A	JE SHORT MSUBUM60.660FC522	
660FC508	8138 43726F7	CMP DWORD PTR DS:[EAX], 756F7243	
660FC50E	75 12	JNZ SHORT MSUBUM60.660FC522	
660FC510	90	NOP	
660FC511	90	NOP	
660FC512	90	NOP	
660FC513	90	NOP	
660FC514	90	NOP	
660FC515	90	NOP	
660FC516	90	NOP	
660FC517	90	NOP	
660FC518	90	NOP	
660FC519	90	NOP	
660FC51A	C74424 0C 00	MOV DWORD PTR SS:[ESP+C], 40000000	
660FC522	FF15 E814006	CALL DWORD PTR DS:[<&USER32.CreateWindowExA], USER32.CreateWindowExA	
660FC528	E9 B1E3F5FF	JMP MSUBUM60.6605A8DE	
660FC52D	0000	ADD BYTE PTR DS:[EAX], AL	
660FC52F	0000	ADD BYTE PTR DS:[EAX], AL	
660FC531	0000	ADD BYTE PTR DS:[EAX], AL	
660FC533	0000	ADD BYTE PTR DS:[EAX], AL	
660FC535	00	DB 00	
660FC536	00	DB 00	
660FC537	00	DB 00	
660FC538	00	DB 00	

660FC500	8B4424 08	MOV EAX,DWORD PTR SS:[ESP+8]
660FC504	85C0	TEST EAX,EAX
660FC506	74 1A	JE SHORT MSUBUM60.660FC522
660FC508	8138 43726F7	CMP DWORD PTR DS:[EAX],756F7243
660FC50E	75 12	JNZ SHORT MSUBUM60.660FC522
660FC510	90	NOP
660FC511	90	NOP
660FC512	90	NOP
660FC513	90	NOP
660FC514	90	NOP
660FC515	90	NOP
660FC516	90	NOP
660FC517	90	NOP
660FC518	90	NOP
660FC519	90	NOP
660FC51A	C74424 0C 00	MOV DWORD PTR SS:[ESP+C],40000000
660FC522	FF15 E814006	CALL DWORD PTR DS:[&USER32.CreateWindowExA]
660FC528	E9 B1E3F5FF	JMP MSUBUM60.6605A8DE
660FC52D	0000	ADD BYTE PTR DS:[EAX],AL
660FC52F	0000	ADD BYTE PTR DS:[EAX],AL
660FC531	0000	ADD BYTE PTR DS:[EAX],AL
660FC533	0000	ADD BYTE PTR DS:[EAX],AL
660FC535	00	DB 00
660FC536	00	DB 00

Allí vemos que si EAX es cero no modifica nada y va directo al call a la api.

Si EAX no es cero como en este caso continua, a la línea siguiente

660FC500	8B4424 08	MOV EAX,DWORD PTR SS:[ESP+8]
660FC504	85C0	TEST EAX,EAX
660FC506	74 1A	JE SHORT MSUBUM60.660FC522
660FC508	8138 43726F7	CMP DWORD PTR DS:[EAX],756F7243
660FC50E	75 12	JNZ SHORT MSUBUM60.660FC522
660FC510	90	NOP
660FC511	90	NOP
660FC512	90	NOP
660FC513	90	NOP

Donde compara si es la string que buscamos o sea si los 4 primeros bytes del titulo son 756f7243

Address	Hex dump	ASCII
00E70740	43 72 6F 75 7A 20 43 72	Crowz Cr
00E70748	61 63 6B 4D 65 2D 31 20	ackMe-1
00E70750	3A 20 53 65 74 75 70 00	: Setup.
00E70758	02 00 04 00 CE 01 0A 00	0.0.0.0
00E70760	46 6F 72 6D 31 00 E6 00	Form1.p
00E70768	13 01 02 00 00 10 00 00	!00.0.0
00E70770	78 01 E6 00 78 01 E6 00	x0p.x0p
00E70778	00 00 00 00 00 00 00 00
00E70780	00 00 00 00 00 00 00 00

Como en este caso la comparación es cierta, y es que va a crear la nag, no salta, si fuera cualquier otra ventana, saltaría directo a la api sin hacer cambios.

660FC500	8B4424 08	MOV EAX,DWORD PTR SS:[ESP+8]
660FC504	85C0	TEST EAX,EAX
660FC506	74 1A	JE SHORT MSUBUM60.660FC522
660FC508	8138 43726F7	CMP DWORD PTR DS:[EAX],756F7243
660FC50E	75 12	JNZ SHORT MSUBUM60.660FC522
660FC510	90	NOP
660FC511	90	NOP
660FC512	90	NOP
660FC513	90	NOP
660FC514	90	NOP
660FC515	90	NOP
660FC516	90	NOP
660FC517	90	NOP
660FC518	90	NOP
660FC519	90	NOP
660FC51A	C74424 0C 00	MOV DWORD PTR SS:[ESP+C],40000000
660FC522	FF15 E814006	CALL DWORD PTR DS:[&USER32.CreateWindowExA]
660FC528	E9 B1E3F5FF	JMP MSUBUM60.6605A8DE
660FC52D	0000	ADD BYTE PTR DS:[EAX],AL
660FC52F	0000	ADD BYTE PTR DS:[EAX],AL
660FC531	0000	ADD BYTE PTR DS:[EAX],AL
660FC533	0000	ADD BYTE PTR DS:[EAX],AL

Allí vemos que si no es igual, salta a la api sin cambiar nada, ahora si es igual o sea,es la ventana buscada o nag, continua y cambia el parámetro que esta en ESP+C que es el estilo de la ventana por 40000000

\$ ==>	00040000	
\$+4	0000C28C	
\$+8	00E70740	ASCII "Crouz CrackMe-1 : Setup"
\$+C	02C80000	
\$+10	00000000	
\$+14	FFFFFFFC	
\$+18	00000107	
\$+1C	000000D6	
\$+20	00780666	
\$+24	00000000	
\$+28	66000000	OFFSET MSUBVM60.#1374
\$+2C	00000000	
\$+30	00E65888	
\$+34	00000000	

Allí esta, al ejecutar la linea

660FC500	8B4424 08	MOV EAX,DWORD PTR SS:[ESP+8]	
660FC504	85C0	TEST EAX,EAX	
660FC506	74 1A	JE SHORT MSUBVM60.660FC522	
660FC508	8138 43726F7	CMP DWORD PTR DS:[EAX],756F7243	
660FC50E	75 12	JNZ SHORT MSUBVM60.660FC522	
660FC510	90	NOP	
660FC511	90	NOP	
660FC512	90	NOP	
660FC513	90	NOP	
660FC514	90	NOP	
660FC515	90	NOP	
660FC516	90	NOP	
660FC517	90	NOP	
660FC518	90	NOP	
660FC519	90	NOP	
660FC51A	C74424 0C 00	MOV DWORD PTR SS:[ESP+C],40000000	
660FC522	FF15 E814006	CALL DWORD PTR DS:[&USER32.CreateWindowExA]	USER32.CreateWindowExA
660FC528	E9 B1E3F5FF	JMP MSUBVM60.6605A8DE	
660FC52D	0000	ADD BYTE PTR DS:[EAX],AL	
660FC52F	0000	ADD BYTE PTR DS:[EAX],AL	
660FC531	0000	ADD BYTE PTR DS:[EAX],AL	
660FC533	0000	ADD BYTE PTR DS:[EAX],AL	
660FC535	00	DB 00	
660FC536	00	DB 00	
660FC537	00	DB 00	

Llega a la api y allí vemos como quedaron los parámetros

\$ ==>	00040000	ExtStyle = WS_EX_APPWINDOW
\$+4	0000C28C	Class = C28C
\$+8	00E70740	WindowName = "Crouz CrackMe-1 : Setup"
\$+C	40000000	Style = WS_CHILD
\$+10	00000000	X = 0
\$+14	FFFFFFFC	Y = FFFFFFFC (-4.)
\$+18	00000107	Width = 107 (263.)
\$+1C	000000D6	Height = D6 (214.)
\$+20	00780666	hParent = 00780666 ('CrouzCrackMe2',class='ThunderRT6Main')
\$+24	00000000	hMenu = NULL
\$+28	66000000	hInst = 66000000
\$+2C	00000000	lParam = NULL
\$+30	00E65888	
\$+34	00000000	

Vemos que como cambiamos ESP+C por 40000000 ahora el estilo es WS_CHILD como queríamos.

La siguiente línea es el call a la api que no debe hacerse en forma directa sino al igual que como lo hace en la llamada original recordamos que era.

6605A8D5	FF75 F4	PUSH DWORD PTR SS:[EBP-C]	
6605A8D8	FF15 E8140066	CALL DWORD PTR DS:[&USER32.CreateWindowExA]	USER32.CreateWindowExA
6605A8DE	8BF8	MOV EDI,EAX	
6605A8E0	8B46 28	MOV EAX,DWORD PTR DS:[ESI+28]	
6605A8E3	8B00	MOV EAX,DWORD PTR DS:[EAX]	
6605A8E5	F640 03 08	TEST BYTE PTR DS:[EAX+3],8	
6605A8E9	^ 0F84 29FFFFFF	JE MSUBUM60.6605A718	
6605A8EF	56	PUSH ESI	
6605A8F0	E8 BE4EFFFF	CALL MSUBUM60.6604F7B3	
6605A8F5	66:30 0300	CMP AX,3	
6605A8F9	^ 0F85 19FFFFFF	JNZ MSUBUM60.6605A718	
6605A8FF	8D45 94	LEA EAX,DWORD PTR SS:[EBP-6C]	
6605A902	50	PUSH EAX	
6605A903	57	PUSH EDI	
6605A904	FF15 14140066	CALL DWORD PTR DS:[&USER32.GetWindowRect]	USER32.GetWindowRect
6605A90A	8D45 94	LEA EAX,DWORD PTR SS:[EBP-6C]	
6605A90D	8D4E 58	LEA ECX,DWORD PTR DS:[ESI+58]	
6605A910	50	PUSH EAX	
6605A911	E8 5119FFFF	CALL MSUBUM60.6604C267	
6605A916	^ E9 FDFDFFFF	JMP MSUBUM60.6605A718	
6605A91B	56	PUSH ESI	
6605A91C	5C4FFFFFFF	CALL MSUBUM60.6604F87D	
6605A921	FF75 F4	PUSH DWORD PTR SS:[EBP-C]	
6605A924	56	PUSH ESI	
6605A925	E8 8F52FFFF	CALL MSUBUM60.6604FBB9	
6605A92A	50	PUSH EAX	
6605A92B	8D45 D0	LEA EAX,DWORD PTR SS:[EBP-30]	
6605A92E	FF75 08	PUSH DWORD PTR SS:[EBP+8]	
6605A931	50	PUSH EAX	
6605A932	FF15 D4130066	CALL DWORD PTR DS:[&USER32.AdjustWindowRectEx]	USER32.AdjustWindowRectEx
6605A938	^ E9 AFFEFFFF	JMP MSUBUM60.6605A7EC	
6605A93D	3BD8	CMP EBX,EAX	
6605A93F	^ 0F8E 0AFFFFFF	JLE MSUBUM60.6605A84F	
6605A945	3BD8	MOV EBX,EAX	
6605A947	^ E9 03FFFFFF	JMP MSUBUM60.6605A84F	
DS:[660014E8]=77D2025E (USER32.CreateWindowExA)			

Allí vemos que es un CALL indirecto, que lee el valor correcto de la api de 660014e8, por lo tanto hacemos lo mismo escribimos.

CALL [660014e8]

Y OLLY cambiara y colocara el nombre de la api donde saltara, pero es importante respetar siempre que sea un call indirecto similar al original, para que funcione en cualquier maquina, ya veremos eso con mas profundidad en la parte de desempacados e IATs.

660FC519	90	NOP	
660FC51A	C74424 0C 00	MOV DWORD PTR SS:[ESP+C],40000000	
660FC522	FF15 E8140066	CALL DWORD PTR DS:[&USER32.CreateWindowExA]	USER32.CreateWindowExA
660FC528	^ E9 B1E3F5FF	JMP MSUBUM60.6605A8DE	
660FC52D	0000	ADD BYTE PTR DS:[EAX],AL	
660FC52F	0000	ADD BYTE PTR DS:[EAX],AL	
660FC531	0000	ADD BYTE PTR DS:[EAX],AL	
660FC533	0000	ADD BYTE PTR DS:[EAX],AL	
660FC535	00	DB 00	
660FC536	00	DB 00	

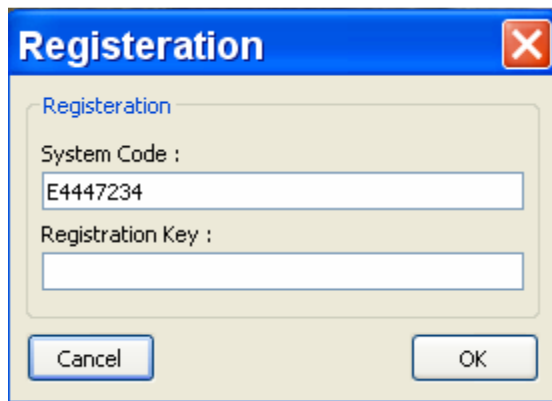
Luego del CALL a la api vuelve a la línea siguiente al call original, que es en mi maquina el MOV EDI,EAX.

6605A8D4	:	57	PUSH EDI	
6605A8D5	:	FF75 F4	PUSH DWORD PTR SS:[EBP-C]	
6605A8D8	✓	FF25 00C40F6	JMP DWORD PTR DS:[660FC400]	MSUBUM60.660FC500
6605A8DE	:	8BF8	MOV EDI,EAX	
6605A8E0	:	8B46 28	MOV EAX,DWORD PTR DS:[ESI+28]	
6605A8E3	:	8B00	MOV EAX,DWORD PTR DS:[EAX]	
6605A8E5	:	F640 03 08	TEST BYTE PTR DS:[EAX+3],8	
6605A8F9	^	0F84 29FFFFFF	JE MSUBUM60.6605A718	

Donde continúa ejecutando el programa

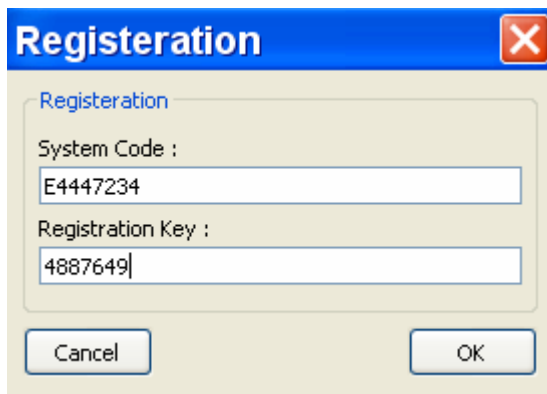
Como vemos lo que he hecho es reemplazar el call a una api, por una rutina propia, que para cualquier ventana que no sea la nag funcione como siempre, solamente una vez que chequea que es la nag a matar, allí le cambia el estilo para que no se vea.

Guardemos todos estos cambios y probemos.

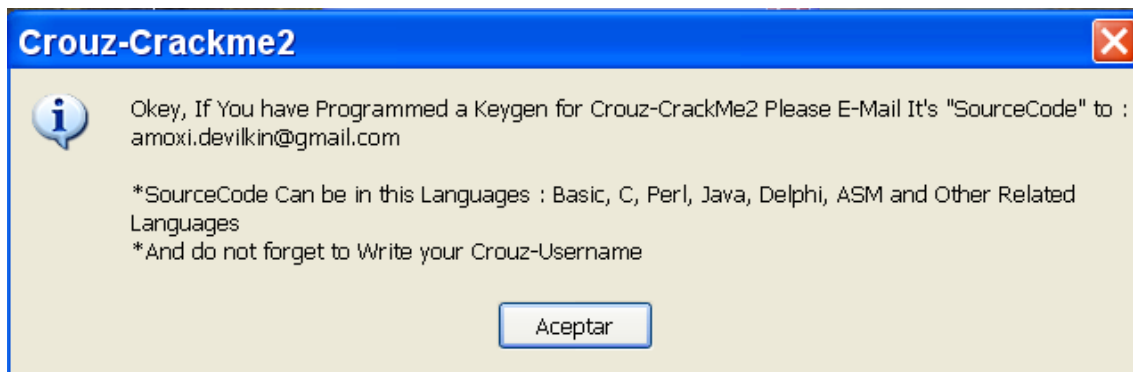


Vemos que con el crackme parcheado y la dll parcheada elimine la nag perfectamente y funciona, probemos el serial correcto anterior que averiguamos.

Lo tipeo en la ventana



Apreto OK



Vemos que funciona perfectamente, el tema es no ser mojigatos y injertar probar y hacer lo que sea mas sencillo, que soluciones hay muchas, pero si podemos hacerla que funcione y no perjudique otro programa, estaríamos jugando casi con las mismas herramientas que el programador (ellos si perjudican otros programas a veces, nosotros no) pero estaríamos mas a mano.

Me gustaría que para practicar vean si pueden quitar la nag del otro crackme que les di en la parte anterior, pero esta vez sin el método del 4c, si no usando este método de parchear el crackme y la dll, a ver si pueden con el.

Hasta la parte 29
Ricardo Narvaja
24 de enero de 2006