

Estudio completo de aplicaciones Visual C++ 6.0 (2 parte)

por

AkirA

Información

Programa:	Varios (ir leyendo)
Tamaño:	<i>El tamaño no importa XDDDDD</i>
Herramientas:	1. Ollydbg 1.09c
Dificultad:	<i>NewBie avanzado</i>

Introducción

Hola amigos!!!! Bienvenidos a la 42 entrega del curso de AkirA.

Desde que hice el tuto sobre VC++ he recibido un montón de preguntas sobre el tema, parece que este tema interesa a mucha gente. Bien, espero que con este tuto aclare conceptos que parece que no quedaron muy claros con el primer tuto.

Cualquier duda escribirme a mi email atalasa@hotmail.com

Nota: si necesitais informacoin sobre Ollydbg buscar en la pagina de Joe Cracker: www.iespana.es/ollydbg esta es la mejor página que hay sobre el tema, de hecho gracias ha ella yo hago todos es tos poryectos en olly

Comentario del Programa

Por supuesto el disclaimer de turno. Vamos a ver, no es que no me haga responsable de la utilización de esta información, es que directamente paso del tema, el único propósito de todo esto es de carácter educativo.

A fin de cuentas, si lo que quieres es crackear un programa pues te bajas el crack y punto, pero si vas a leer este tutorial es porque tu objetivo es aprender. Eso es lo que nos motiva, el comprender como funcionan las cosas o como estan hechas por dentro, y por supuesto el subidón de haberle ganado a un equipo de ingenieros diseccionando un objeto que ellos habian diseñado y del cual no sabemos nada.

Manos a la Obra

Comienza el asedio....

Con el compilador de Visual C ++ podemos hacer mucha clase de proyectos y aplicaciones. Básicamente las mas importantes se dividen en dos: se puede utilizar las MFC, o se puede hacer un programa sin MFC.

(Hay un tercer tipo, pero sirve para hacer programas en MS-DOS, lo cual.....)

Vamos a hablar de los dos tipos, de cómo diferenciarlos y cómo analizarlos.

Lo primero que vamos a hacer es hablar de los programas hechos con MFC. MFC es un conjunto de objetos (códigos y asistentes) que Microsoft nos dan a los programadores para poder desarrollar aplicaciones de una manera mas cómoda y rápida. Así, usando MFC podemos usar los objetos Cedit, Cbutton, CString, etc, etc.

¿ Cómo saber si un programa está hecho con las MFC?

Fácil, los programas hechos con MFC necesitan de varias DLL para poder ejecutarse. Por ejemplo en el tuto sobre visual C++ yo hice un crackme. Cuando le doy a ejecutar, se carga la DLL MFC42.DLL, donde se encontraba la dirección 5f4373f9. Ya quedamos que esa dirección era la que decidía cual era el código del programa principal que debía ejecutarse cuando surgía un evento (pulsar un botón) Si ponemos un breakpoint justo antes de pulsar un botón, rompemos en esa dirección y si damos a f7 podremos trazar dentro de la rutina importante desde la primera línea.

Vamos a ver esto, con un nuevo ejemplo desde el principio para que podais entenderlo mejor.

Cargamos el DriveTest.exe de Wkz_Daedalus y lo primero que debemos ver y muy importante, es lo siguiente. En el EOP siempre, y subrayo lo de siempre (ya sea hecho con con o sin MFC) el código inicial es el mismo para cualquier programa en VC++.

Esto es debido a que al cargar tiene que hacer unas comprobaciones y sobre todo, debe cargar las DLL y este código lo escribe el compilador, no el programador.

Mas cosas: cualquier código que escribimos nosotros, se escribirá siempre por encima del EOP y presumiblemente la gestión de eventos empezara en 401000.

Pero lo más importante es lo siguiente: tanto si el programa tiene MFC, como si no, siempre se llama a GetStartupInfoA y después a GetModuleHandleA y la primera función CALL que haya después de esta última API, será la llamada a el principio de lo que nosotros ya si hemos programado.

004021B6	. 51	PUSH ECX	
004021B7	. FF15 E0734100	CALL DWORD PTR DS:[&KERNEL32.GetStartupInfoA]	pStartupInfo
004021B8	. 8B55 00	MOV EDX,DWORD PTR SS:[EBP-30]	GetStartupInfoA
004021C0	. 83E2 01	AND EDX,1	
004021C3	. 8502	TEST EDX,EDX	
004021C5	. 74 0D	JE SHORT Drivetes.004021D4	
004021C7	. 8B45 D4	MOV EAX,DWORD PTR SS:[EBP-2C]	
004021CA	. 25 FFFF0000	AND EAX,0FFFF	
004021CF	. 8945 84	MOV DWORD PTR SS:[EBP-7C],EAX	
004021D2	. EB 07	JMP SHORT Drivetes.004021D8	
004021D4	> C745 84 0A0000	MOV DWORD PTR SS:[EBP-7C],0A	
004021D8	> 8B4D 84	MOV ECX,DWORD PTR SS:[EBP-7C]	
004021DE	. 51	PUSH ECX	Arg4
004021DF	. 8B55 8C	MOV EDX,DWORD PTR SS:[EBP-74]	
004021E2	. 52	PUSH EDX	Arg3
004021E3	. 6A 00	PUSH 0	Arg2 = 00000000
004021E5	. 6A 00	PUSH 0	pModule = NULL
004021E7	. FF15 D8734100	CALL DWORD PTR DS:[&KERNEL32.GetModuleHandleA]	GetModuleHandleA
004021ED	. 50	PUSH EAX	Arg1
004021EE	. E8 ED000000	CALL Drivetes.004022E0	Drivetes.004022E0

Ahora Run y dejar que cargue tranquilo el programa y se carguen las DLL. Poned el breakpoint en 5f4373f9 y presionad el botón de OK y romperá en nuestra dirección.

5F4373F6	. 8B4D 08	MOV ECX,DWORD PTR SS:[EBP+8]
5F4373F9	. FF55 FC	CALL DWORD PTR SS:[EBP-4]
5F4373FC	. E9 48040000	JMP MFC42D.5F437849

Con un par de f7 llegareis a un Push EBP, más abajo vereis GetDriveTypeA, bueno pues ya sabeis en que consiste la gracia de este crackme.

004023B0	> 55	PUSH EBP	
004023B1	. 8BEC	MOV EBP,ESP	
004023B3	. 83EC 50	SUB ESP,50	
004023B6	. 53	PUSH EBX	
004023B7	. 56	PUSH ESI	
004023B8	. 57	PUSH EDI	
004023B9	. 51	PUSH ECX	
004023BA	. 8D7D B0	LEA EDI,DWORD PTR SS:[EBP-50]	
004023BD	. B9 14000000	MOV ECX,14	
004023C2	. B8 CCCCCCCC	MOV EAX,CCCCCCCC	
004023C7	. F3:AB	REP STOS DWORD PTR ES:[EDI]	
004023C9	. 59	POP ECX	
004023CA	. 894D FC	MOV DWORD PTR SS:[EBP-4],ECX	
004023CD	. C745 F4 1C504	MOV DWORD PTR SS:[EBP-C],Drivetes.00415	ASCII "E:\\"
004023D4	. 8BF4	MOV ESI,ESP	
004023D6	. 8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
004023D9	. 50	PUSH EAX	
004023DA	. FF15 DC734100	CALL DWORD PTR DS:[&KERNEL32.GetDriveTypeA]	RootPathName
004023E0	. 3BF4	CMP ESI,ESP	GetDriveTypeA

Vamos a aprender mas cosas sobre MFC.

Cargar ahora mi crackme, el del primer tuto de VC++. Era un simple crackme que cogía una cadena y la comparaba con otra.

Si repetis el proceso anterior rompereis dentro de la función que gestiona el evento de que se pulsó el botón Registrar. Vale, lo importante es que vereis que en toda esa función, a pesar de ser muy fácil no se llama a ninguna API.

Eso tiene una explicación: cuando se trabaja con MFC, el propio asistente te da la posibilidad de asociar variables a las cajas de texto para manejarlas de forma sencilla y rápida. Suelen ser dos objetos, o Un CString o un Cedit. Este crackme en concreto está hecho con CString, y como os podeis imaginar las funciones del CString están implementadas dentro de la DLL. Por eso vemos llamadas a funciones 389, 255, etc, etc... esto es muy parecido a cuando en Visual Basic vemos la _vbaStrcmp o _vbaStrcat, etc. Pero Olly no nos pone ningun nombre.

Haced una cosa: cargad el crackme con el IDA y cuando termine de analizar, dais a File -Produce- Create MAP y poneis un nombre a ese archivo.map

Ahora en el Olly debereis tener un Plugin llamado MapConv. Lo pinchais y elegid sustituir comentarios por los del archivo.map Ahora esas CALL 389 ya tiene un nombre. Lo único que debéis hacer es hacer vuestras propias pruebas e ir descubriendo para que sirve cada función.

Por ejemplo:

String (+) es la función que compara dos String, equivale en Visual Basic a `_vbaStrCmp`, así que de ahora en adelante cualquier `string(+)` sabréis que sirve para comparar dos Strings XD.

(No pongo más sobre estas funciones, porque estoy haciendo una lista que espero sea grande y definitiva, para un tercer tuto, aunque tu mismo puedes hacerlo XD)

Seguimos aprendiendo cosas sobre las MFC. Estan implementadas sobre DLL y existen varias versiones. Por ejemplo, nosotros hemos visto la MFC42.DLL veamos un ejemplo con otra versión.

Cargamos el VCCrackMe10 Eternal y vemos que carga las MFC40.DLL . Atentos a que también, en el comienzo se llama a las dos APIs de las que hablamos antes.

Si aplicamos el mismo proceso que hicimos en el primer tuto sobre VC++ descubriremos que el salto importante en la versión 4.0 de las MFC está en 61f026f1 en vez de 5f4373f9, como era en 4.2

```
61F026BE 8B4D 08      MOV ECX,DWORD PTR SS:[EBP+8]
61F026C1 FF55 14      CALL DWORD PTR SS:[EBP+14]
61F026C4 EB 41      JMP SHORT MFC40.61F02707
```

Os ayudo: poned breakpoint en 401526 y cuando pulseis el botón mirad el valor de la pila XD

```
00401526 . 55          PUSH EBP
00401527 . 8BEC        MOV EBP,ESP
00401529 . 6A FF       PUSH -1
0040152B . 68 30164000 PUSH UCCrackM.0040169A
00401530 . 50          PUSH EAX
00401531 . 64:8925 00000 MOV DWORD PTR FS:[0],ESP
00401538 . 83EC 20     SUB ESP,20
0040153B . 894D F0     MOV DWORD PTR SS:[EBP-10],ECX
0040153E . 53          PUSH EBX
0040153F . 56          PUSH ESI
00401540 . 8D4D E0     LEA ECX,DWORD PTR SS:[EBP-20]
00401543 . 68 58304000 PUSH UCCrackM.00403058
00401548 . E8 79030000 CALL <JMP.&MFC40.#483>
0040154D . 68 50304000 PUSH UCCrackM.00403050
                                ASCII "Wrong! Try Again"
                                ASCII "Error"
```

Además vereis que las funciones que gestionan nuestro serial son las mismas que antes.

Bueno, segunda parte, analicemos los programas en VC++ que no utiliza MFC.

Cargamos el Crackme! vc++ de Demian.

Fijaos que aquí también al empezar se llama a `GetStartupInfoA` y después a `GetModuleHandleA`. Y que el código normal está por encima de esta dirección.

00401E2E	. 51	PUSH ECX	
00401E2F	. FF15 C4614100	CALL DWORD PTR DS:[<&KERNEL32.GetStartupInfoA]	pStartupInfo GetStartupInfoA
00401E35	. E8 760C0000	CALL CrackME!.00402AB0	
00401E3A	. 8945 9C	MOV DWORD PTR SS:[EBP-64],EAX	
00401E3D	. 8B55 D0	MOV EDX,DWORD PTR SS:[EBP-30]	
00401E40	. 83E2 01	AND EDX,1	
00401E43	. 85D2	TEST EDX,EDX	
00401E45	. 74 0D	JE SHORT CrackME!.00401E54	
00401E47	. 8B45 D4	MOV EAX,DWORD PTR SS:[EBP-2C]	
00401E4A	. 25 FFFF0000	AND EAX,0FFFF	
00401E4F	. 8945 94	MOV DWORD PTR SS:[EBP-6C],EAX	
00401E52	. EB 07	JMP SHORT CrackME!.00401E5B	
00401E54	. C745 94 0A0000	MOV DWORD PTR SS:[EBP-6C],0A	
00401E5B	. 8B4D 94	MOV ECX,DWORD PTR SS:[EBP-6C]	
00401E5E	. 51	PUSH ECX	
00401E5F	. 8B55 9C	MOV EDX,DWORD PTR SS:[EBP-64]	
00401E62	. 52	PUSH EDX	
00401E63	. 6A 00	PUSH 0	
00401E65	. 6A 00	PUSH 0	
00401E67	. FF15 C0614100	CALL DWORD PTR DS:[<&KERNEL32.GetModuleHandleA]	pModule = NULL GetModuleHandleA
00401E6D	. 50	PUSH EAX	
00401E6E	. E8 B0F1FFFF	CALL CrackME!.00401023	

Situaos en GetModuleHandleA. El siguiente CALL es el que llama al programa que sí programa el cracker.

Cuando no se utilizan MFC hay que programar con las APIs verdaderas.

Ahora, veamos un posible método para encontrar el punto en que empieza una función al presionar un botón.

En 401e6e está la call famosa. Trazamos una vez allí con f7 y otro y vemos que llama a DialogParamA.

0040CD9B	. 55	PUSH EBP	
0040CD9C	. 8BEC	MOV EBP,ESP	
0040CD9E	. 56	PUSH ESI	
0040CD9F	. 8BF4	MOV ESI,ESP	
0040CDA1	. 6A 00	PUSH 0	
0040CDA3	. 68 05104000	PUSH CrackME!.00401005	
0040CDA8	. 6A 00	PUSH 0	
0040CDA9	. 6A 65	PUSH 65	
0040CDAC	. 8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	
0040CDAF	. 50	PUSH EAX	
0040CDB0	. FF15 B8624100	CALL DWORD PTR DS:[<&USER32.DialogBoxParamA]	lParam = NULL DlgProc = Crack! hOwner = NULL pTemplate = 65 hInst DialogBoxParamA

Aquí hay dos posibilidades. Podemos ir a la dirección del DlgProc, osea la función que gestiona los eventos del cuadro de diálogo (401005) y buscar allí el botón, o buscar el botón directamente.

Vereis:

Dad a run y cuando cargue pincad e el botón Window del Olly. Allí vereis que el botón Check tiene como ID 3E8.

0024026E		003802A8	000003EB	50010000	00000204	Main	77D2D0E8	Edit
002E024E	&Check	003802A8	000003E8	50010000	00000004	Main	77D350BD	Button
002F028A	E&x it	003802A8	00000002	50010000	00000004	Main	77D350BD	Button

vais al desensamblador y con el derecho search for – all constant, y poneos 3E8 y os saldrá una única línea. Lla pinchais dos veces y vereis que es una comparación con [ebp-8] (en la línea 40cd27 ☺) y que si es verdad salta a 40cd52.

0040CD25	. 74 14	JE SHORT CrackME!.0040CD3B	
0040CD27	. 817D F8 E80300	CMP DWORD PTR SS:[EBP-8],3E8	
0040CD2E	. 74 22	JE SHORT CrackME!.0040CD52	

Bien, por cualquiera de los dos métodos hubiesemos llegado hasta aquí, porque lo que ocurre es que en la DlgProc se mira el evento que se genera. Si es el de se pulso un botón 201 se compara con el Id de cada botón para saber cual es el que se pulso.

Osea que en 40cd52 esta la línea que nos lleva a la rutina importante del crackme que está en 40ca88 (trazar un poquito)

```
0040CA88 | . 55          PUSH EBP
0040CA89 | . 8BEC        MOV EBP,ESP
0040CA8B | . 83EC 28     SUB ESP,28
0040CA8E | . 56          PUSH ESI
```

Pero en fin, como podeis ver, cuando se trabaja sin MFC, se trabajan con APIs directamente, así que cualquier metodo del mundo es válido XD

Ahora, la tercera parte del tuto y posiblemente la más importante:

¿Se puede empaquetar un programa hecho en VC++?

Como poder se puede, pero es una grandísima estupidez XD. Veamos porque.

Carguemos el crackme CME.exe de K3nny. Sabemos que es un crackme en VC++ que está empackado. No tengo ni idea de cual es el empaquetador pero es que me da igual.

Sabemos que el principio de cualquier programa en VC++ es igual y que se llaman a dos funciones al empezar a GetStartupInfoA y después a GetModuleHandleA, bien, pues si dejais que se descomprima, lo único que teneis que hacer es mirar en el código y encontrarlas.

```
004015AD | . 50          PUSH EAX
004015AE | . FF15 34404000 CALL DWORD PTR DS:[404034]
004015B4 | . E8 5C040000 CALL cme.00401A15
004015B9 | . 8945 9C      MOV DWORD PTR SS:[EBP-64],EAX
004015BC | . F645 00 01   TEST BYTE PTR SS:[EBP-30],1
004015C0 | . 74 06        JE SHORT cme.004015C8
004015C2 | . 0FB745 D4    MOVZX EAX,WORD PTR SS:[EBP-2C]
004015C6 | . EB 03        JMP SHORT cme.004015CB
004015C8 | . 6A 0A        PUSH 0A
004015CA | . 58          POP EAX
004015CB | . 50          PUSH EAX
004015CC | . FF75 9C      PUSH DWORD PTR SS:[EBP-64]
004015CF | . 56          PUSH ESI
004015D0 | . 56          PUSH ESI
004015D1 | . FF15 30404000 CALL DWORD PTR DS:[404030]
```

pStartupinfo
GetStartupInfoA

pModule
GetModuleHandleA

Subir un par de líneas hasta que encontréis el típico Push EBP y punto XDD el EOP está en 40150f,

```
0040150F | . 55          PUSH EBP
00401510 | . 8BEC        MOV EBP,ESP
00401512 | . 6A FF        PUSH -1
```

(también se puede poner un BPX en las APIs antes de que se desencripte y con un ctrl.+f9 y será mas rápido que buscar XD)

Ahora se dumpea con lo que querais (con el plugging del Olly funciona) y ya está.

La verdad es que este mismo “fallo” le ocurre a Visual Basic.

Cualquier programa hecho en VB empieza con estas dos lineas

Push (valor numérico)

Call ThuRMain (no está bien escrito, mirarlo en cualquier crackme VB)

Y justo encima hay un montón de Jmp[funciones de VB], osea de tipo ff25.

Ademas ese valor numérico apunta a una cadena Ej: VB5 que le dice la Dll a cargar.

El EOP es siempre el Push (valor numérico) y si no está se escribe XD

Yo se porque packer lo digo XDD.

004014B2	-FF25 20114000	JMP DWORD PTR DS:[401120]	msvbvm60._vbaStrToAnsi
004014B8	-FF25 C4104000	JMP DWORD PTR DS:[4010C4]	msvbvm60.EVENT_SINK_QueryInte
004014BE	-FF25 98104000	JMP DWORD PTR DS:[401098]	msvbvm60.EVENT_SINK_AddRef
004014C4	-FF25 BC104000	JMP DWORD PTR DS:[4010BC]	msvbvm60.EVENT_SINK_Release
004014CA	-FF25 14114000	JMP DWORD PTR DS:[401114]	msvbvm60.ThunRTMain
004014D0	68 B0C24500	PUSH UBCrackM.0045C2B0	
004014D5	E8 F0FFFFFF	CALL UBCrackM.004014CA	JMP to msvbvm60.ThunRTMain
004014DA	0000	ADD BYTE PTR DS:[EAX],AL	

Bueno espero que os haya quedado clar,o como eso de los programa de VC++ y que os hayais divertido con vuestros nuevos conocimientos.

Hasta la proxima ¡!!

Nota:

Bueno espero que hayas aprendido mucho que es de lo que se trata, que te sirva de ejemplo para que tu también puedas hacerlo, desde luego no hay comparado como coger un programa, abrirlo, ver un montón de código por todas partes y manejar lo que otros ingenieros han hecho, tu sólo, por ti mismo.

Bueno , si quieres comentarme algo escíbeme a atalasa@hotmail.com

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más