

Estudio completo de aplicaciones Visual C++ 6.0

por

AkirA

Información

Programa:	Crackme de mi cosecha (ponerse en contacto conmigo)
Tamaño:	El tamaño no importa XDDDDD
Herramientas:	Ollydbg 1.09c
Dificultad:	NewBie avanzado

Introducción

Hola amigos!!!! Bienvenidos a la 37 entrega del curso de AkirA.

Bueno, hace tiempo que deseaba echarle el guante a este compilador y por lo que parece he dado con un método que nos puede abrir grandes puertas.

Para poder dar con él me hice un pequeño crackme superbásico, pero utilizando todos los asistentes que trae este compilador, de esta forma, coincidiría con el 90% de los programas.

Todavía quedan cosas por investigar, pero espero que esto sea más que suficiente para que disfrutéis.

Cualquier duda escribirme a mi email atalasa@hotmail.com

Nota: si necesitáis información 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 proyectos 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 están hechas por dentro, y por supuesto el subidón de haberle ganado a un equipo de ingenieros diseccionando un objeto que ellos habían diseñado y del cual no sabemos nada.

Manos a la Obra

Comienza el asedio....

Hola amigos y bienvenidos a la 37 entrega del curso de AkirA. La intención de este tutorial es encontrar un método general que nos ayude a estudiar y a comprender los secretos de cualquier aplicación que este hecha en Visual C++ 6.0.

Llevaba tiempo queriendo encontrar algo así y en cuanto me ha venido la idea brillante y he comprobado que funciona no me he esperado ni a dormir para escribirlo.

Aun no puedo deciros el porqué exactamente del truco, ni como funciona todo por dentro, pero si me he dado cuenta de que es un paso muy importante que nos va a solucionar grandes problemas y que ya se ha abierto el cerrojo de este compilador.

Al formatear el ordenador de mi casa e intentar ejecutar una aplicación hecha en MFC y con el VC++ me salió un mensaje diciendo “Te faltan las Runtime muchacho”. Eso fue definitivo.

Si por casualidad no te acuerdas de que era eso, te remito a que te vuelvas a leer el Tuto sobre Visual Fox Pro 5.0 o el de Visual basic 5.0 y 6.0 o el de Delphi.

Bien, el paso siguiente era encontrar el punto exacto (si es que existía) en el que las Runtime (las dll que tienen normalmente el control) capturaban un evento (Ej: El usuario a pulsado un botón) y buscaban que línea de código del programa original debía ejecutarse y en que punto exacto le pasaban el control.

Para tal tarea me hice yo mismo un pequeño crackme en este compilador. Era lo más simple posible: un cuadro de diálogo que tenía una caja de texto y un botón de aceptar y otro de cancelar. Y por supuesto las típicas MessageBoxA de “Bien” y “mal”. El texto sin encriptar me daría la clave.

Compilé y abrí el ejecutable con el Ollydbg y eché un vistazo.

Rápidamente encontré los Mensajes de Bien y Mal. Y unas líneas más arriba encontré el típico Push EBP, etc, que indican que es el comienzo de una función.

Address	Hex dump	Disassembly	Registers
0040191E	CC	INT3	EAX 0041518C
0040191F	CC	INT3	ECX 0012FE70
00401920	> 55	PUSH EBP	EDX 00000000
00401921	. 8BEC	MOV EBP,ESP	EBX 00000000
00401923	. 6A FF	PUSH -1	ESP 0012F620
00401925	. 68 0B324000	PUSH Copia_de.004032A8	EBP 0012F654
0040192A	. 64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	ESI 00143593
00401930	. 50	PUSH EAX	EDI 0012F8BC
00401931	. 64:8925 00000000	MOV DWORD PTR FS:[0],ESP	EIP 00401920
00401938	. 83EC 50	SUB ESP,50	C 0 ES 0023
0040193B	. 53	PUSH EBX	P 1 CS 001B
0040193C	. 56	PUSH ESI	A 0 SS 0023
0040193D	. 57	PUSH EDI	Z 1 DS 0023
0040193E	. 51	PUSH ECX	S 0 FS 0038
0040193F	. 8D7D A4	LEA EDI,DWORD PTR SS:[EBP-5C]	T 0 GS 0000
00401942	. B9 14000000	MOV ECX,14	D 0
00401947	. B8 CCCCCCCC	MOV EAX,CCCCCCCC	O 0 LastErr
0040194C	. 33AB	REP STOS DWORD PTR ES:[EDI]	EFL 00000246
0040194E	. 59	POP ECX	ST0 empty 0.0
0040194F	. 894D F0	MOV DWORD PTR SS:[EBP-10],ECX	ST1 empty +UH
00401952	. 6A 01	PUSH 1	ST2 empty -5.
00401954	. 8B4D F0	MOV ECX,DWORD PTR SS:[EBP-10]	ST3 empty +UH
00401957	. E8 58030000	CALL <JMP.&MFC42D.#5056>	ST4 empty +UH
0040195C	. 68 AC524100	PUSH Copia_de.004152AC	ST5 empty -UH
00401961	. 8D4D EC	LEA ECX,DWORD PTR SS:[EBP-14]	ST6 empty 8.5
00401964	. E8 45030000	CALL <JMP.&MFC42D.#487>	ST7 empty 0.0
00401969	. C745 FC 00000000	MOV DWORD PTR SS:[EBP-4],0	FST 4000
00401970	. 68 A0524100	PUSH Copia_de.004152A0	FCW 027F
00401975	. 8D4D E8	LEA ECX,DWORD PTR SS:[EBP-18]	
00401978	. E8 31030000	CALL <JMP.&MFC42D.#487>	
0040197D	. C645 FC 01	MOV BYTE PTR SS:[EBP-4],1	
00401981	. 68 90524100	PUSH Copia_de.00415290	
00401984	. 8D4D E4	LEA ECX,DWORD PTR SS:[EBP-1C]	
EBP=0012F654 Jump from 0040102D			
0012F620	5F4373FC	RETURN to MFC42D.5F4373FC	
00416000	0000	ADD BYTE PTR DS:[EAX],AL	0012F624
00416002	0000	ADD BYTE PTR DS:[EAX],AL	0012F628
00416004	0000	ADD BYTE PTR DS:[EAX],AL	0012F62C
00416006	0000	ADD BYTE PTR DS:[EAX],AL	0012F630
00416008	0000	ADD BYTE PTR DS:[EAX],AL	0012F634
0041600A	0000	ADD BYTE PTR DS:[EAX],AL	0012F638
0041600C	0000	ADD BYTE PTR DS:[EAX],AL	0012F63C
0041600E	0000	ADD BYTE PTR DS:[EAX],AL	0012F640
00416010	0000	ADD BYTE PTR DS:[EAX],AL	0012F644
00416012	0000	ADD BYTE PTR DS:[EAX],AL	0012F648
00416014	0000	ADD BYTE PTR DS:[EAX],AL	0012F64C
00416016	0000	ADD BYTE PTR DS:[EAX],AL	0012F650

Estas dos líneas calculan una dirección y saltan a ella:

0040101E	.vE9 00050000	JMP Copia_de.00401600
00401023	.vE9 48060000	JMP Copia_de.00401670
00401028	.vE9 63010000	JMP Copia_de.00401190
0040102D	.vE9 EE080000	JMP Copia_de.00401920
00401032	.vE9 29030000	JMP Copia_de.00401360
00401037	.vE9 E4030000	JMP Copia_de.00401420
0040103C	.vE9 AF000000	JMP Copia_de.004010F0

Y allí podemos observar un JMP precioso. En 40102D JMP 401920. Magnífico!!!!!! Todas las sospechas confirmadas!!!!!! Otro compilador que ha caído!!!!

Pero si os dais cuenta no es el único JMP que hay. Hay muchos otros JMP: una tabla de ellas.

Parece ser que el VC++ es un compilador de lo más ordenado!! El programa principal comienza y hace una serie de operaciones estándar y en cierto punto le pasa el control a las Runtimes.

Cuando el usuario produce algún evento, las runtimes calculan la dirección de la función a ejecutar. Dirección que la sacan de esa tabla de JMP's, ¡magnífico, mas ordenado imposible!

Pero ahora observemos que ocurre si pincho en cancelar, ya que a ese botón no le escribí ninguna función y debe ejecutarse código por defecto.

Pinchamos y salta el Olly en 5F4373F9

Magnifico!!!! Nueva confirmación!!!!

Trazamos con F7 y estamos en esas dos líneas. Trazamos con F7 y saltamos a ... sorpresa!! Es una tabla de JMP como esperábamos pero ahora apuntan a direcciones de una Dll. De la Runtime XDD

00401B52	\$.FF25 50744100	JMP DWORD PTR DS:[<&MFC42D.#574>]	MFC42D.#574
00401B58	\$.FF25 54744100	JMP DWORD PTR DS:[<&MFC42D.#684>]	MFC42D.#684
00401B5E	\$.FF25 58744100	JMP DWORD PTR DS:[<&MFC42D.#4195>]	MFC42D.#4195
00401B64	\$.FF25 5C744100	JMP DWORD PTR DS:[<&MFC42D.#3629>]	MFC42D.#3629
00401B6A	\$.FF25 60744100	JMP DWORD PTR DS:[<&MFC42D.#4017>]	MFC42D.#4017
00401B70	\$.FF25 64744100	JMP DWORD PTR DS:[<&MFC42D.#4753>]	MFC42D.#4753
00401B76	\$.FF25 68744100	JMP DWORD PTR DS:[<&MFC42D.#3362>]	MFC42D.#3362

Esto encaja muy bien en nuestra teoría.

Parece ser que el compilador escribe todos los eventos por defecto que sean necesarios. Todos implementados en la DLL. Pero que si sobrescribes alguna función por defecto, el compilador te crea una nueva tabla de JMP que apuntan a tus funciones en el programa principal.

Todavía no he tenido tiempo de hacer muchas comprobaciones ni de sacarle partido a esta nueva información, pero si hay una cosa clara, "Hemos conseguida dar con un truco igual de bueno que el de delphi o el de Vbasic".

A partir de ahora, cuando queramos saber que código del programa original se ejecuta al apretar un botón solo hay que poner un breakpoint en 5F4373F9, trazar dos líneas con F7 y llegaremos a una tabla de JMP's que nos dirá cual es la dirección buena XDD

Bueno, espero que le saquéis jugo a esta nueva información

Hasta la próxima!!!

Nota:

Espero que te hayas divertido y sobre todo 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 nada comparado con coger un programa, abrirlo, ver un montón de código por todas partes y manejar lo que otros ingenieros han hecho, tú solo, por ti mismo.

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

Quiero agradecer a Ricardo Narvaja y a Makkako por su increíble esfuerzo de escribir tantísimos y buenos tutoriales que nos han llevado a aprender tanto, agradecer al Profesor X sus famosas compilaciones, y también agradecer a Joe Cracker su página muy, muy actualizada y el esfuerzo de divulgación del Olly que esta haciendo, ha sido muy importante en mis progresos como cracker.

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.