INTRODUCCION AL CRACKING CON OLLYDBG PARTE 33

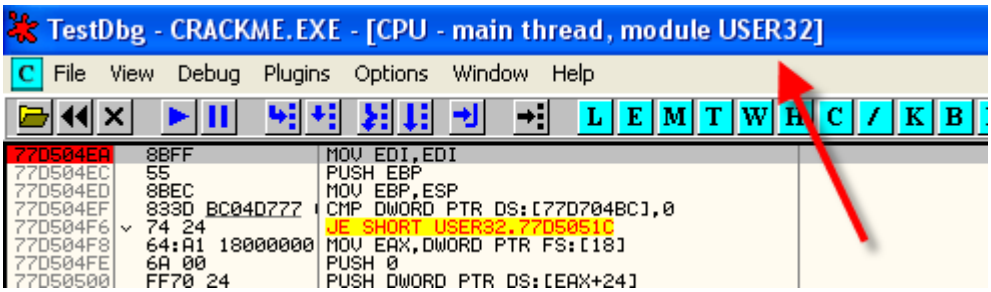
Que es la IAT y como repararla

Antes de ponernos a reparar IATs como loco haremos una introduccion de que es exactamente la IAT y veremos en el CRACKME DE CRUEHEAD original donde esta ubicada y que le hace el packer a la misma, comparandolo con el crackmeUPX empacado.

Primero demos una idea generica de para que sirve todo esto que vamos a explicar:

El tema es el siguiente, como vimos las apis tienen una direccion en nustra maquina, por ejemplo si abro el Crackme de Cruehead original en OLLYDBG y tipeo

Veo que en mi maquina la direccion donde se encuentra la api es 77D504EA, si ustedes ven la direccion de esta api en vuestras maquinas, algunos tendran la misma direccion, otros no, depende de la version de Windows que tengan, y de las actualizaciones que hayan bajado tambien, que como saben son muchisimas, y en cada una al bajar nuevas versiones de la dll que contiene la api en este caso User32.dl, generalmente cambiara la direción.

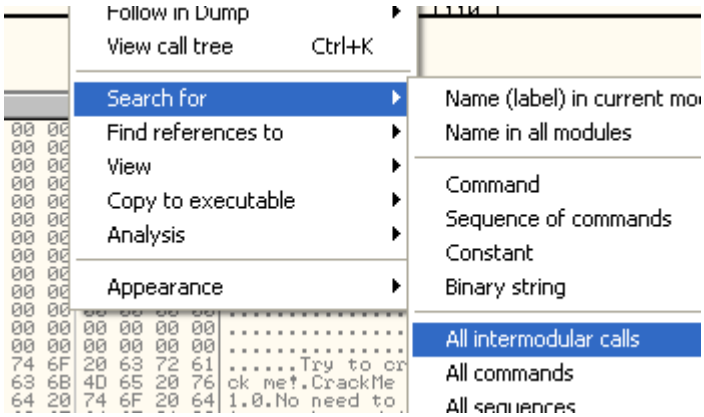


Cada vez que Microsoft saca una nueva version de esta dll, casi siempre cambiara la direccion de las apis que contiene, por lo tanto si yo programara el Crackme de Cruehead, al saltar a la api MessageBoxA, lo haria saltar a 77D504EA, y funcionaria pefcetamente en mi maquina, y en las que tuvieran la suerte de tener la misma version de User32.dll que yo, en el resto de los demas Windows que o bien no sean XP, o bien no tengan la misma version de la User32.dll, saltaria a una direccion donde no estaria la api, lo cual produciria error y no correria.

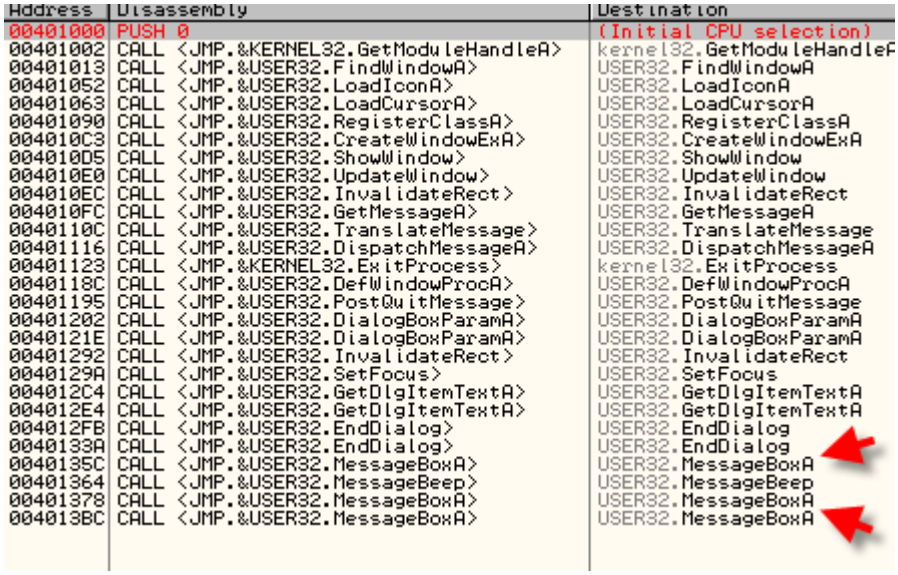
El tema entonces es que el sistema operativo debe suministrar alguna forma de compatibilizar para que mi crackme funcione en otros Windows (dentro de ciertos limites) y en otras versiones de la dll debe funcionar perfectamente.

Para ello se crearon estas famosas tablas llamadas IT (Import Table), y IAT (Import Adress Table)

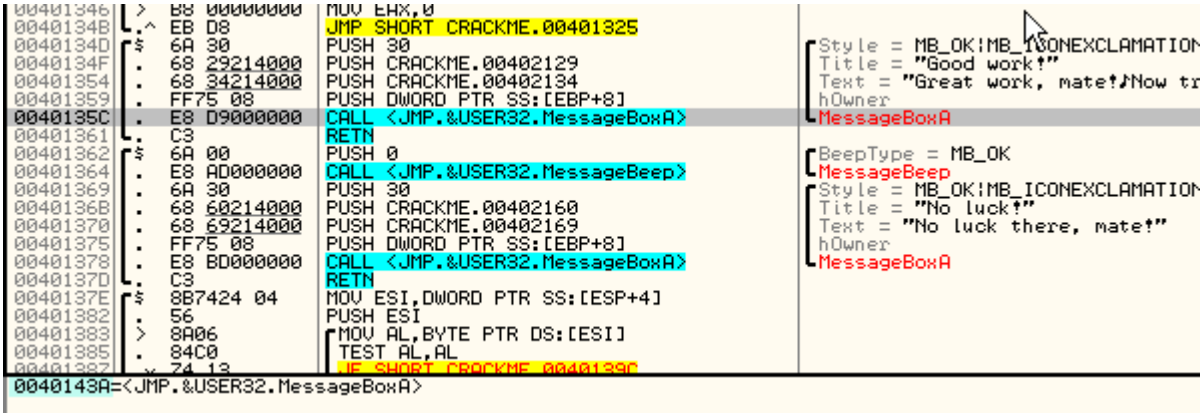
No se austen son puro nombre una vez que uno sabe donde esta ubicada cada una en un programa desempacado y para que sirven no hay mas miedo.



Ahora hagamos SEARCH FOR-ALL INTERMODULAR CALLS y veamos los CALLS que van a otros modulos o dlls, que pueden ser calls a apis.

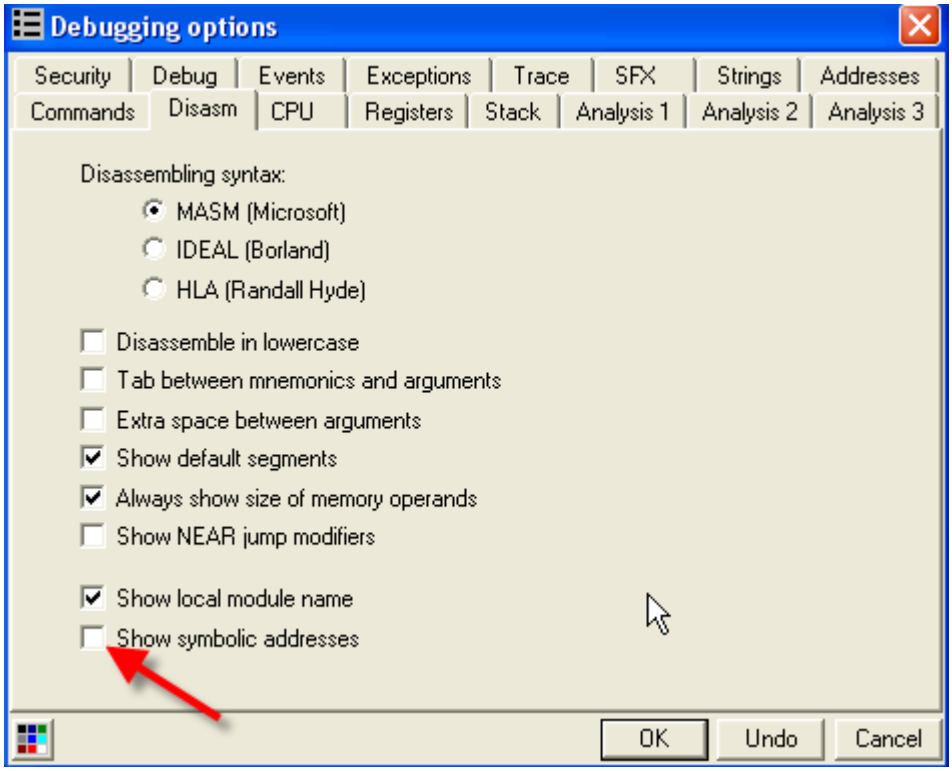


Alli vemos varios Calls a la api MessageBoxA como ejemplo, si hacemos doble click en el primero de ellos

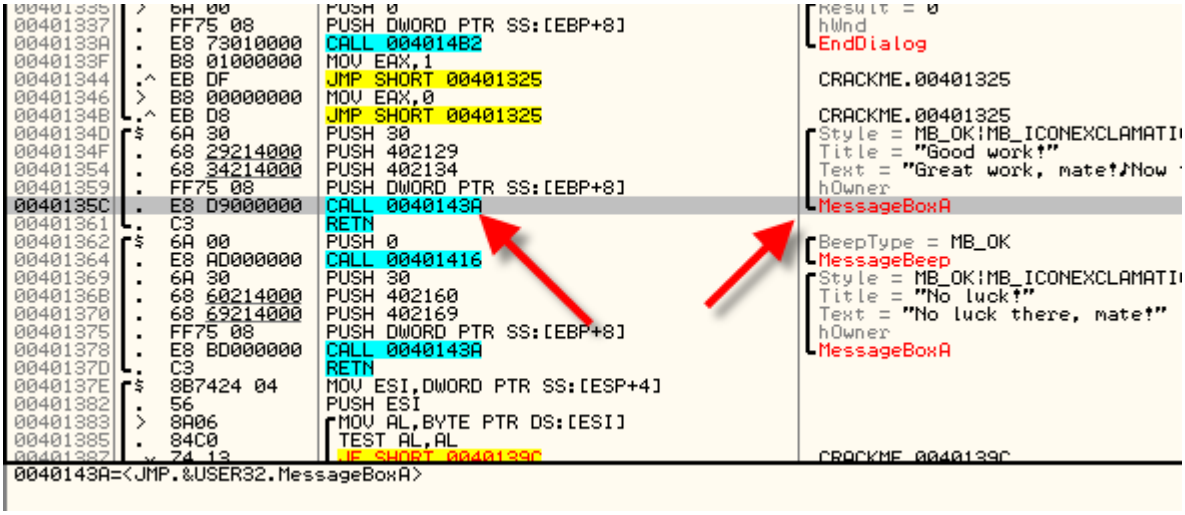


Vemos que en realidad es un CALL 40143A que OLLYDBG para aclararnos nos pone que saltara a la api MessageBoxA, cambiando la notacion por CALL <MessageBoxA> pero no es un call indirecto, es un call directo

CALL 40143A



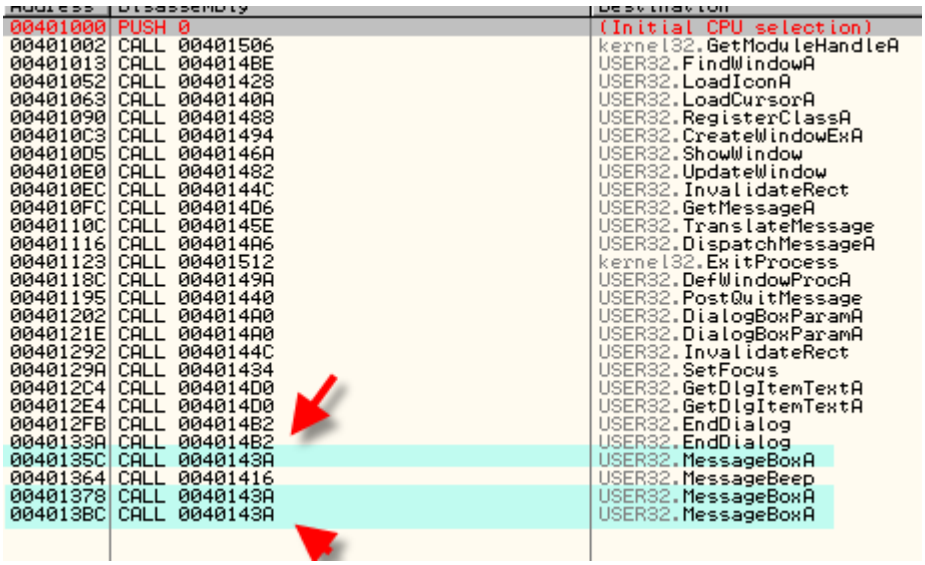
Vemos que en las opciones del OLLYDBG en la pestaña DISASM si le quitamos la tilde a SHOW SIMBOLIC ADDRESSES el listado queda mas claro.



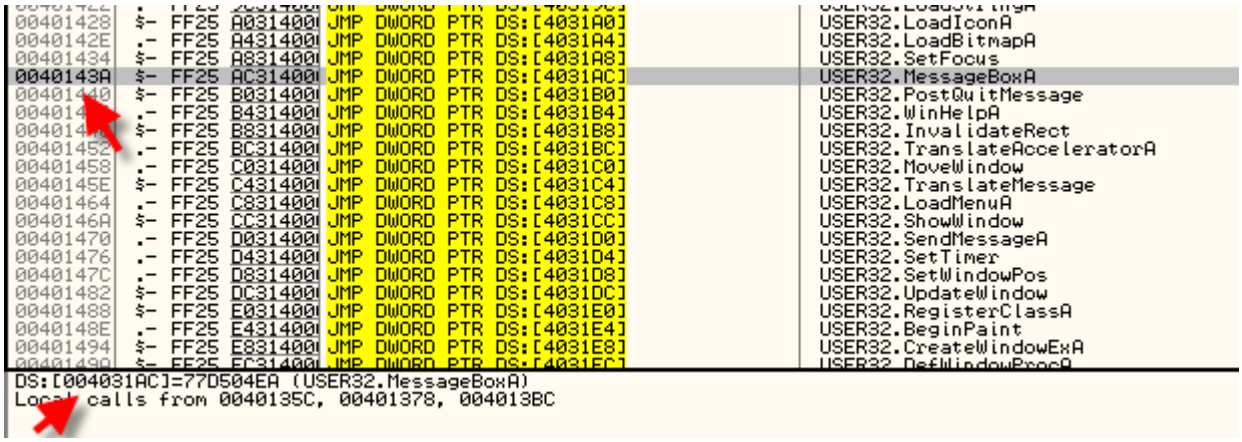
Vemos que igual nos aclara a la derecha los parametros y la api a la cual ira, pero ahora nos muestra el CALL realmente como call directo.

CALL 40143A

En SEARCH FOR INTERMODULARS CALLS

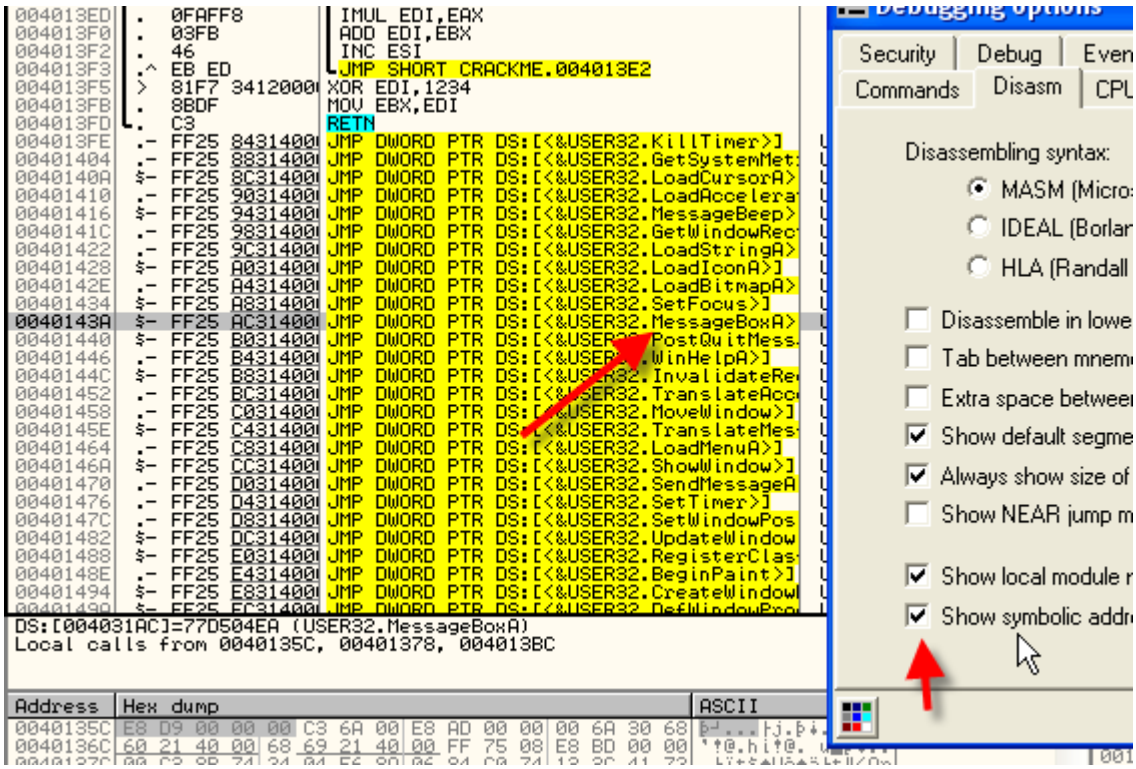


Vemos que en realidad los tres calls a MessageBoxA son calls a 40143A, pues veamos que hay alli.



Aquí vemos el quid de la cuestion, para llegar a la api hace un JMP indirecto, que al tenerlo en esta forma de visualizacion se ve claro que toma el valor a donde saltara de 4031AC

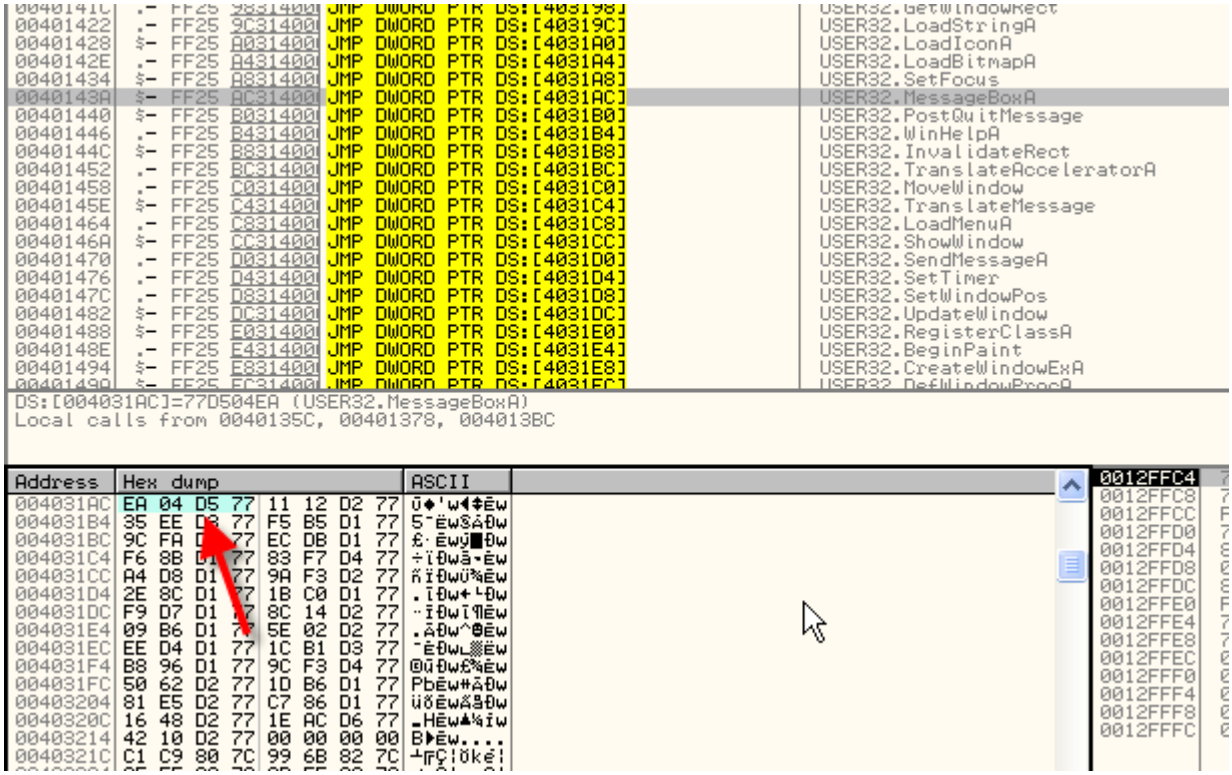
JMP [4031AC]



Tambien vemos que si ponemos la tilde, se nos complica un poco entender que es un salto indirecto, por lo cual la quitaremos mientras estamos trabajando con IATs.

Y aquí esta el truco, el programa salta a la api con un JMP indirecto que toma la direccion que la api tiene en nuestra maquina de 4031AC, y vemos que para todas las otras apis, hay otros JMPS INDIRECTOS similares.

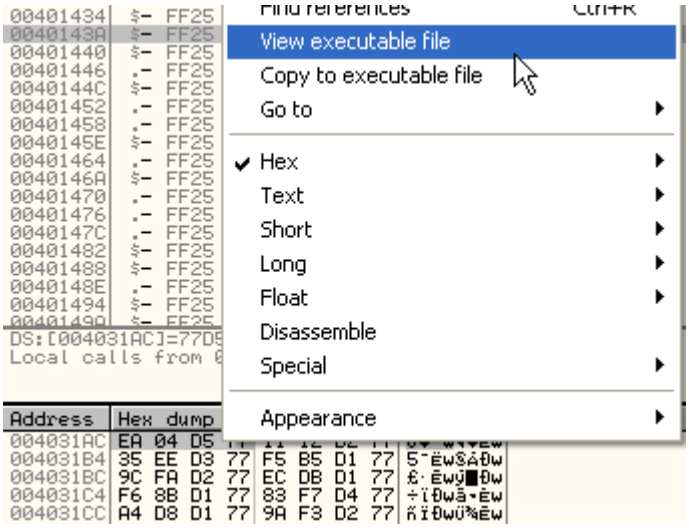
Ahora empezamos a ver un poco mas claro el tema de la compatibilidad entre maquina, cuando el programa tiene que ir a una api, hace un salto indirecto, leyendo la direccion de la api de una especie de deposito, si vemos en el dump este deposito.



Alli vemos realemente que 4031AC es una parte de un deposito que contiene todas las direcciones a las apis en mi maquina, ahora este DEPOSITO es la famosa IAT o IMPORT TABLE ADDRESS, aquí esta la clave de la compatibilidad, quiere decir que todos los programas tendran los mismos JMPS INDIRECTOS a las apis, lo que cambiara es el valor guardado en el deposito, para cada maquina tendra direcciones diferentes que llevaran el salto indirecto, a la direccion correcta de la api.

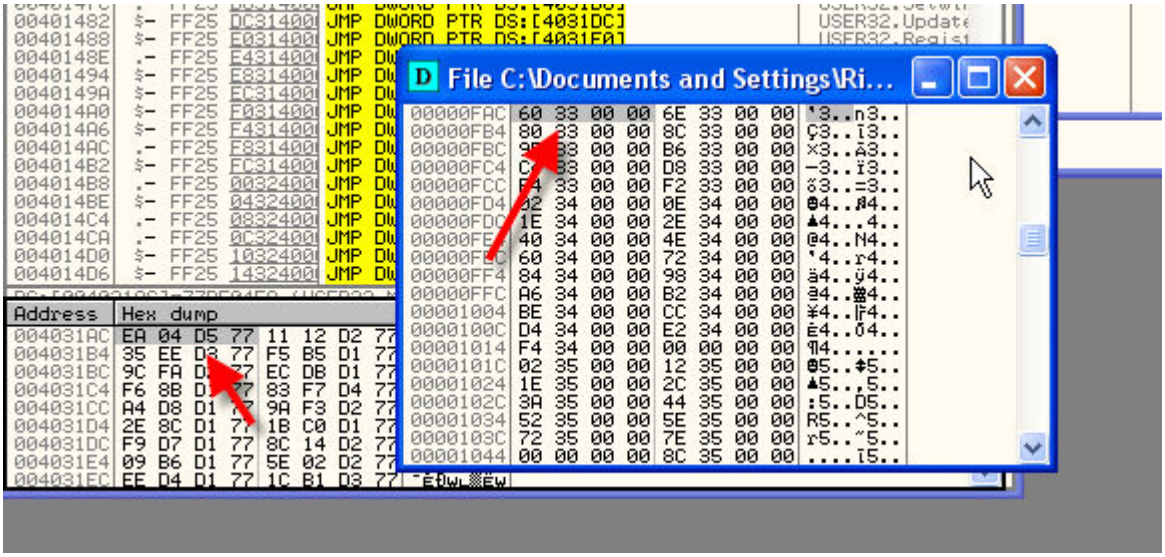
Pero un momento me diran algunos, 4031AC es una direccion del programa, por lo cual si tiene diferentes direcciones para cada maquina, pues entonces el programa seria diferente para cada maquina?

Jeje buena pregunta ahora veremos como funciona el sistema, pero ya sabemos el objetivo, el cual es llenar la IAT de las direcciones correctas para cada maquina.



Sabemos que si hacemos VIEW- EJECUTABLE FILE podemos ver lo que hay en la direccion

4031AC, realmente en el archivo ejecutable guardado en nuestro disco rigido.



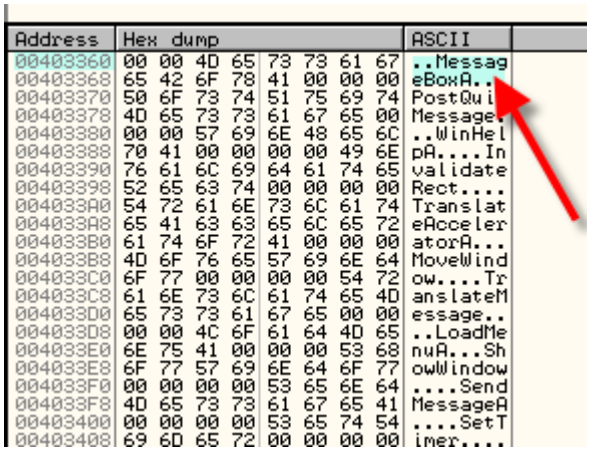
Vemos que el valor en el exe, es 60 33, y que nosotros parados en el entry point ya tenemos en la memoria en la misma posicion la direccion de la api EA 04 D5 77 guardada alli, quiere decir que el sistema al arrancar el programa, tomo el 3360 que habia alli en el OFFSET 0FAC que corresponde a la direccion 4031ac y lo machaco con el valor de la direccion real de la api en mi maquina.

Magia?

No no asi trabaja Windows, cada archivo que arranca, el sistema operativo llena la IAT de las direcciones correctas de las apis en mi maquina, en este caso lleno 4031AC con el valor de la api MessageBoxA y los valores alrededor, con las direcciones de oras apis, para completar la tabla IAT o deposito de direcciones de las apis en mi maquina.

Ahora el sistema no es mago como hace esto, vemos que en 0FAC tiene un valor 3360, le indicara algo al sistema este puntero, para que sepa que api es la que debe llenar alli?

Pues si 3360 es el valor al cual si le sumamos la imagebase queda 403360 y si miramos en el sump esa direccion que vemos?



Pues aquí esta el truco es un puntero a la string MessageBoxA, eso quiere decir que el sistema mira ese puntero, busca que api es por el nombre y con GetProcAddress, saca la direccion de dicha api en

nuestra maquina, y lo completa en la IAT, machacando el 3360, con la direccion real de la api en nuestra maquina, y de esta forma se asegura que el programa funcione para cualquier version de la dll, porque halla la direccion antes de llegar al Entry Point del programa y una vez que llego alli, queda la iat completita con las direcciones de las apis en mi maquina, si miro la iat de otra maquina diferente vere que el contenido de 4031AC es diferente pues la direccion de la api, es diferente, pero cuando haga.

JMP [4031AC]

ira a su MessageBoxA al igual que en mi maquina, el programador no se tiene que preocupar, siempre que haga calls o jmps indirectos al valor que se encuentra en el deposito de la IAT, este estara correcto, al arrancar el sistema coloco en tiempo real la direccion correcta, apuntando a MessageBoxA en todas las maquina para asegurar compatibilidad.

O sea que para que el sistema me complete correctamente la direccion de la api en la iat, debe el archivo tener varios punteros como vimos.

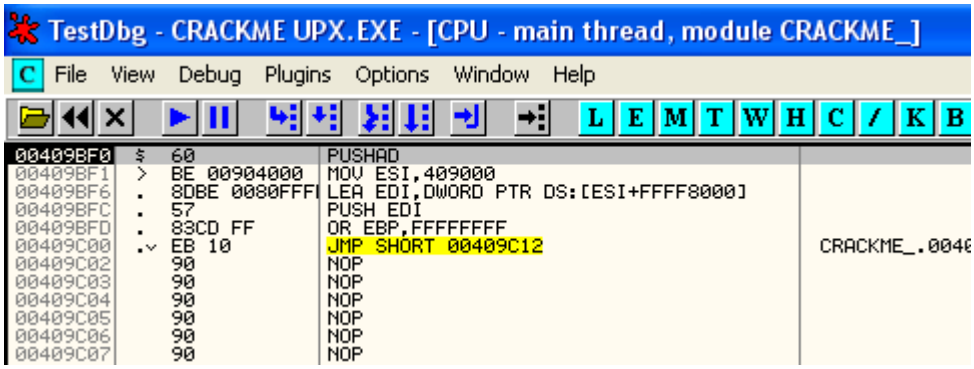
1)en la entrada de la iat en el ejecutable en mi rigido, debe tener un puntero a una cadena de texto que le identifique al sistema que api debe colocar en esa entrada.
2)Por supuesto debe tener la cadena de texto con el nombre de la api

Pues con estos dos puntos, podemos cocluir que el programa arrancara y el sistema me llenara la IAT con los valores correctos. (mas adelante veremos el trabajo cmpleto de este sistema)

Ahora ya vimos como llegamos al OEP de un programa empacado, porque se dice que cuando llegamos alli y dumpeamos un programa debemos reconstruir la IAT, los packers la rompen? Que hacen con ella, pues si la rompen algunos mas otros menos, el truco es que el packer no necesita la iat del programa en si porque arranca desde el mismo desempacador, y puede leer las apis que deberian ir en la IAT, mientras va desempacando el programa, y una vez que leyo cada api, y lleno la iat con las direcciones correctas de cada api, pues las strings que le indicaron cual era cada api, las borrara, ya no las necesita, pues el programa ya arranco y tiene ya en la IAT las direcciones correctas.

Es mas en los packers ni siquiera existen esas strings ya que el programa o bien las guarda encriptadas o bien las guarda en alguna otra direccion que no sean facil de hallar por nosotros los crackers.

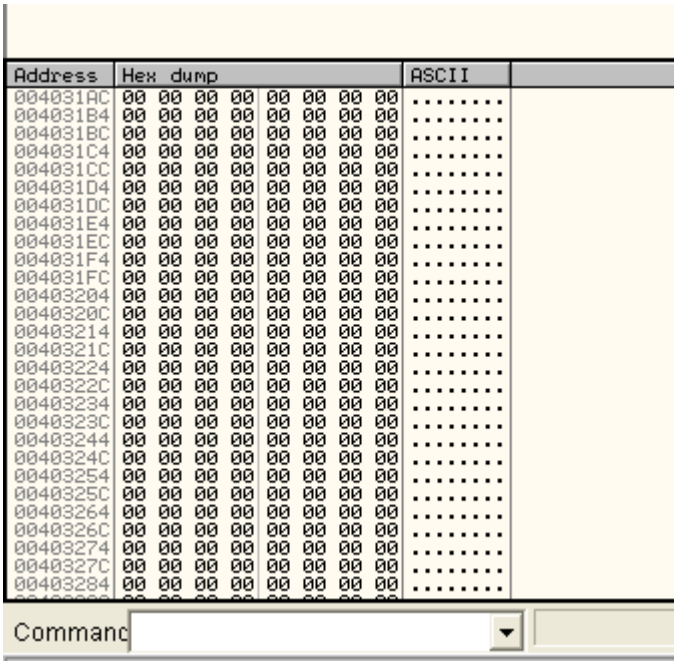
Veamos el ejemplo con el empacado UPX para ver la diferencia y como nos queda al dumpearlo, y porque hay que repararlo.



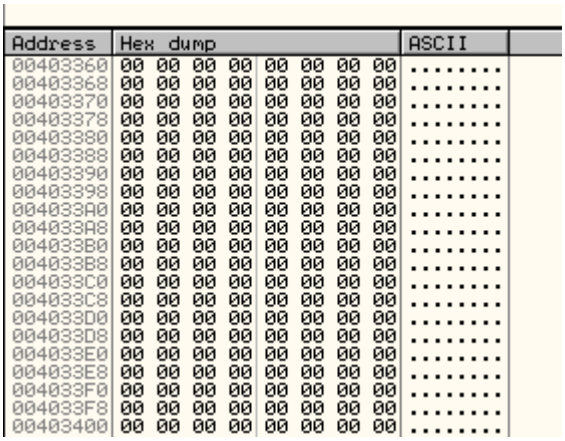
Aquí tenemos el Crackme de Cruehead que esta empacado con UPX que habiamos hecho en partes

anteriores.

Miremos la direccion 4031AC donde estaba el deposito de apis en el original



nada borro todo, y las strings de las apis estaban en 403360 que haya alli?



Nada, de nada, lleguemos al oep y veamos que hay en estas mismas direcciones, ya que para que el programa corra en la iat debe haber algo si no al saltar a la api, y hacer

JMP [4031ac]

y estar vacio dara error, o sea que el packer hara el trabajo que hace normalmente el sistema y llenara la iat, lleguemos al OEP.

```
00409D2D   .  09C0            OR EAX,EAX
00409D2F   .v 74 07           JE SHORT 00409D38                    CRACKME_.00409D38
00409D31   .  8903            MOV DWORD PTR DS:[EBX],EAX
00409D33   .  83C3 04         ADD EBX,4
00409D36   .^ EB E1           JMP SHORT 00409D19                   CRACKME_.00409D19
00409D38   >  FF96 6095000(   CALL DWORD PTR DS:[ESI+9560]
00409D3E   >  61              POPAD
00409D3F   .- E9 BC72FFFF     JMP 00401000                         CRACKME_.00401000
00409D44      00              DB 00
00409D45      00              DB 00
00409D46      00              DB 00
00409D47      00              DB 00
00409D48      00              DB 00
00409D49      00              DB 00
00409D4A      00              DB 00
```

Ponemos un BP en el salto al OEP y damos RUN y llegamos alli

Veamos que hay en el lugar de la iat

```
00401434 - FF25 A8314000   JMP DWORD PTR DS:[4031A8]   USER32.SetFocus
0040143A - FF25 AC314000   JMP DWORD PTR DS:[4031AC]   USER32.MessageBoxA
00401440 - FF25 B0314000   JMP DWORD PTR DS:[4031B0]   USER32.PostQuitMessage
00401446 - FF25 B4314000   JMP DWORD PTR DS:[4031B4]   USER32.WinHelpA
0040144C - FF25 B8314000   JMP DWORD PTR DS:[4031B8]   USER32.InvalidateRect
00401452 - FF25 BC314000   JMP DWORD PTR DS:[4031BC]   USER32.TranslateAcceler
00401458 - FF25 C0314000   JMP DWORD PTR DS:[4031C0]   USER32.MoveWindow
0040145E - FF25 C4314000   JMP DWORD PTR DS:[4031C4]   USER32.TranslateMessage
00401464 - FF25 C8314000   JMP DWORD PTR DS:[4031C8]   USER32.LoadMenuA
0040146A - FF25 CC314000   JMP DWORD PTR DS:[4031CC]   USER32.ShowWindow
00401470 - FF25 D0314000   JMP DWORD PTR DS:[4031D0]   USER32.SendMessageA
00401476 - FF25 D4314000   JMP DWORD PTR DS:[4031D4]   USER32.SetTimer
0040147C - FF25 D8314000   JMP DWORD PTR DS:[4031D8]   USER32.SetWindowPos
00401482 - FF25 DC314000   JMP DWORD PTR DS:[4031DC]   USER32.UpdateWindow
00401488 - FF25 E0314000   JMP DWORD PTR DS:[4031E0]   USER32.RegisterClassA
0040148E - FF25 E4314000   JMP DWORD PTR DS:[4031E4]   USER32.BeginPaint
00401494 - FF25 E8314000   JMP DWORD PTR DS:[4031E8]   USER32.CreateWindowExA
0040149A - FF25 EC314000   JMP DWORD PTR DS:[4031EC]   USER32.DefWindowProcA
004014A0 - FF25 F0314000   JMP DWORD PTR DS:[4031F0]   USER32.DialogBoxParamA
004014A6 - FF25 F4314000   JMP DWORD PTR DS:[4031F4]   USER32.DispatchMessageA
004014AC - FF25 F8314000   JMP DWORD PTR DS:[4031F8]   USER32.DrawMenuBar
004014B2 - FF25 FC314000   JMP DWORD PTR DS:[4031FC]   USER32.EndDialog
004014B8 - FF25 00324000   JMP DWORD PTR DS:[403200]   USER32.EndPaint
DS:[004031AC]=77D504EA (USER32.MessageBoxA)

Address   Hex dump                 ASCII
004031AC  EA 04 D5 77 11 12 D2 77  ú♦'w◄↕Éw
004031B4  35 EE D3 77 F5 B5 D1 77  5ˉËwßÁÐw
004031BC  9C FA D2 77 EC DB D1 77  £·Éwý▀Ðw
004031C4  F6 8B D1 77 83 F7 D4 77  ÷ïÐwâ~Éw
004031CC  A4 D8 D1 77 9A F3 D2 77  ñïÐwü¾Éw
004031D4  2E 8C D1 77 1B C0 D1 77  .îÐw←└Ðw
004031DC  F9 D7 D1 77 8C 14 D2 77  ¨îÐwî¶Éw
```

Vemos que para que el programa funcione el desempacador lleno la iat con las direcciones correctas de la api en mi maquina, pero si nosotros dumpeamos aquí habra un problema, el ejecutable que salga del dumpeado le faltan datos para arrancar.

Ya de por si las strings que identificaban cada api y estaban en 403360 no estan

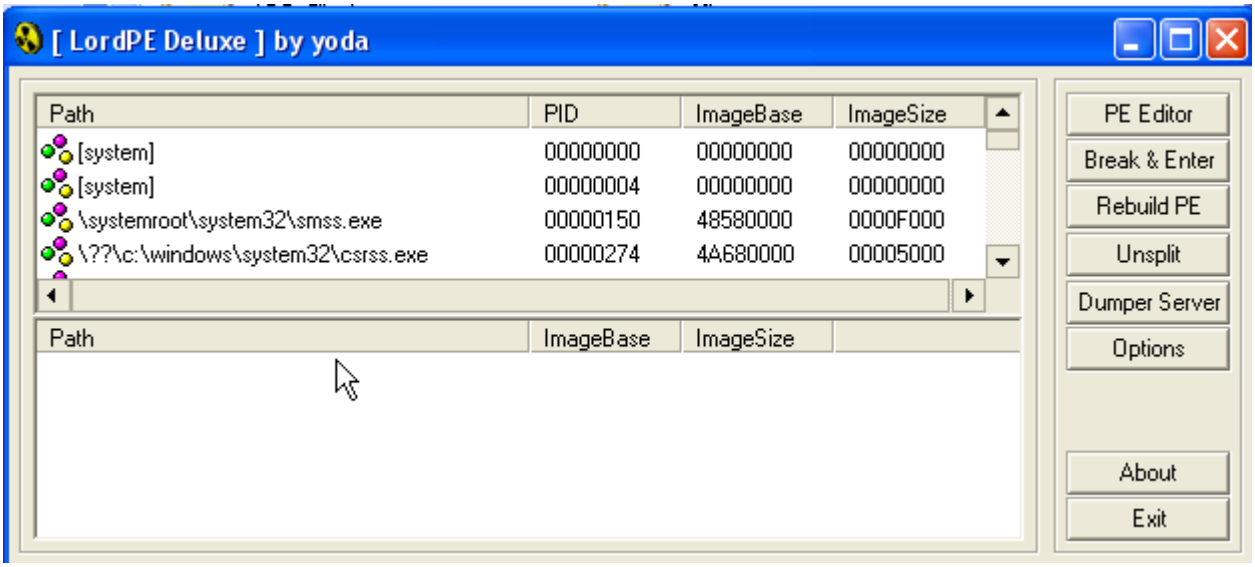
```
Address   Hex dump                 ASCII
00403360  00 00 00 00 00 00 00 00  ........
00403368  00 00 00 00 00 00 00 00  ........
00403370  00 00 00 00 00 00 00 00  ........
00403378  00 00 00 00 00 00 00 00  ........
00403380  00 00 00 00 00 00 00 00  ........
00403388  00 00 00 00 00 00 00 00  ........
00403390  00 00 00 00 00 00 00 00  ........
00403398  00 00 00 00 00 00 00 00  ........
004033A0  00 00 00 00 00 00 00 00  ........
004033A8  00 00 00 00 00 00 00 00  ........
004033B0  00 00 00 00 00 00 00 00  ........
004033B8  00 00 00 00 00 00 00 00  ........
004033C0  00 00 00 00 00 00 00 00  ........
004033C8  00 00 00 00 00 00 00 00  ........
004033D0  00 00 00 00 00 00 00 00  ........
004033D8  00 00 00 00 00 00 00 00  ........
004033E0  00 00 00 00 00 00 00 00  ........
004033E8  00 00 00 00 00 00 00 00  ........
```

Veamos hagamos un dumpeado a ver que queda, pero ya nos damos cuenta que tendremos el

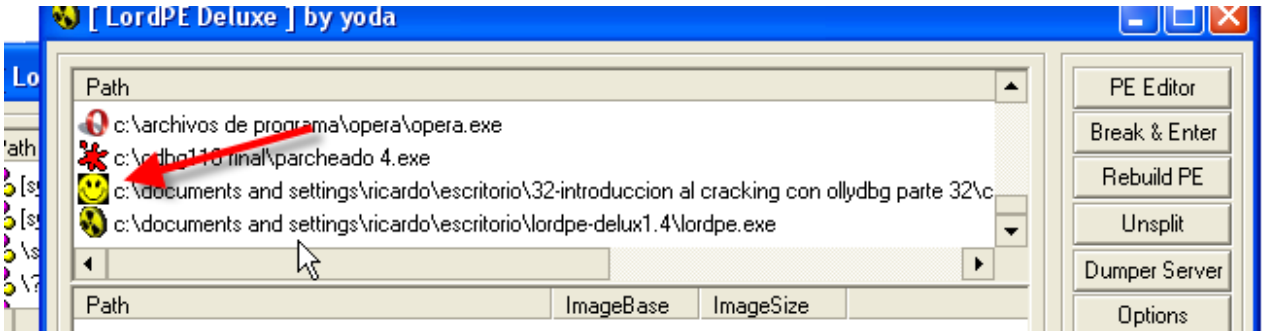
codigo del programa correcto, pero el dumpeado no correra al no poder el sistema llenar la iat por falta de datos.

Por ahora utilizaremos un programa externo al OLLYDBG para dumpear este es el LORDPE DELUXE que se puede bajar desde mi HTTP.

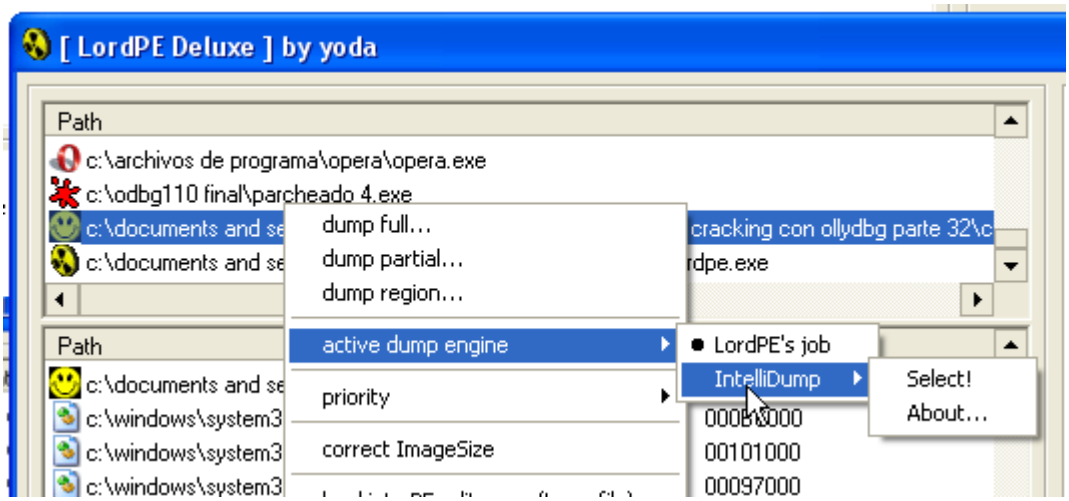
http://www.ricnar456.dyndns.org/HERRAMIENTAS/L-M-N-O-P/lordpe-delux1.4.zip



Alli tenemos al LORD PE DELUXE busquemos el proceso a dumpear que es el crackmeUPX que tenemos detenido en el OEP.

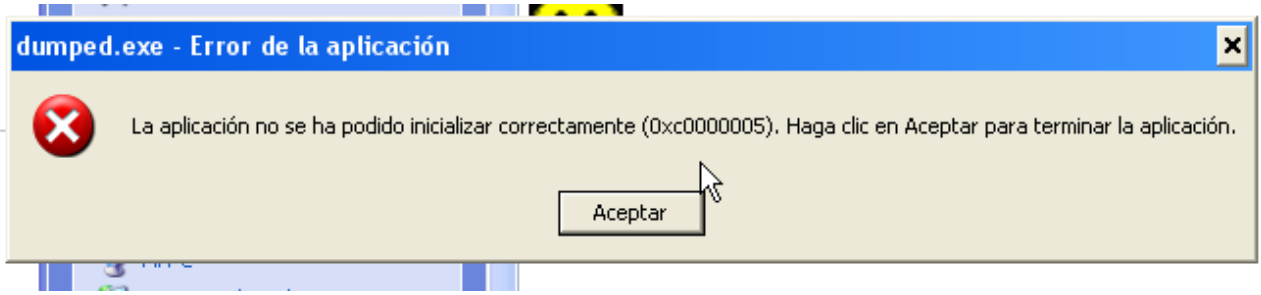


Alli esta el crackmeUPX, marquemoslo



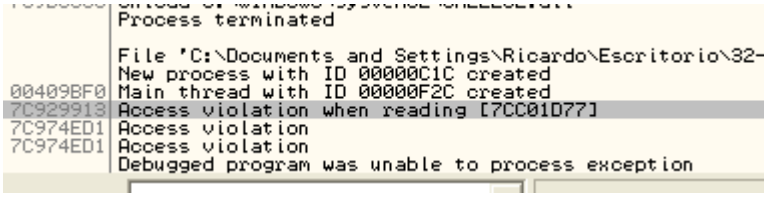
Haciendo click derecho cambiemos a INTELLIDUMP que dicen que es mejor dumpeando, jeje y luego hagamos click en DUMP FULL.

Lo hemos guardado con el nombre DUMPED.exe

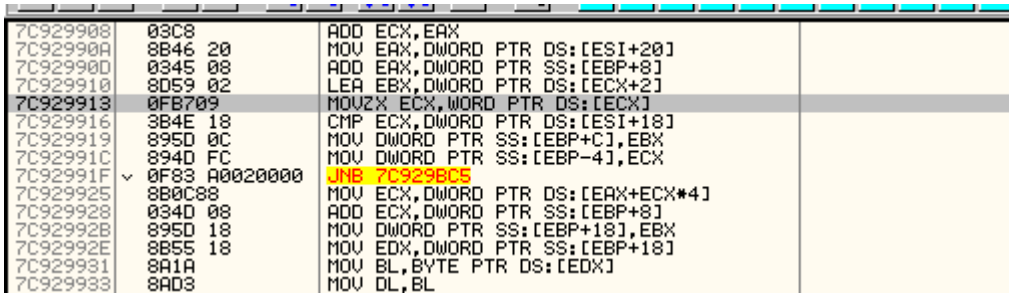


Si corremos el dumpeado vemos que no arranca tratemos de abrirlo en OLLYDBG

Y nos da el mismo error pero si vemos el LOG del OLLYDBG los errores

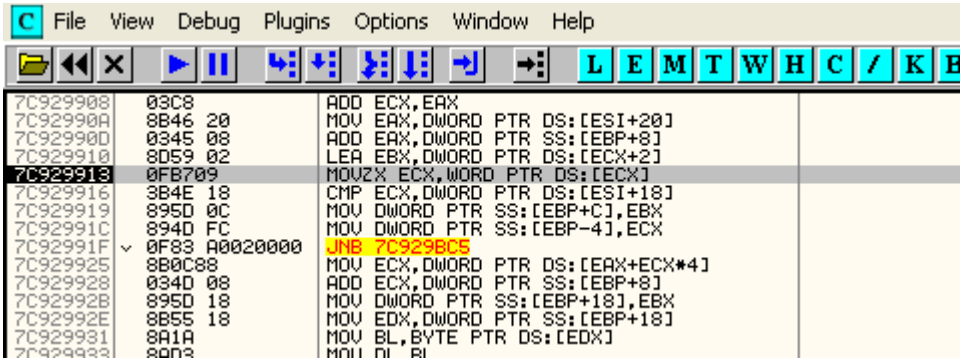


El error se produjo en 7c929913 en mi maquina vayamos a esta direccion, ustedes en su maquina a la direccion que figure en el LOG.



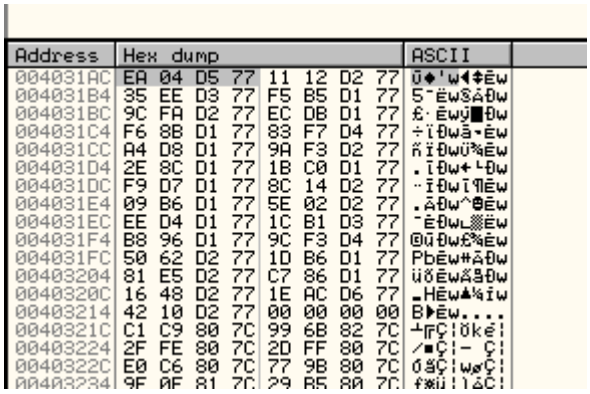
Le pondre un HARDWARE BREAKPINT ON EXECUTION y un BP a ver si para antes de dar el error.

Pues veo que no para, pero tengo otra forma de parar en un error y es quitarle la tilde a las excepciones, al reiniciar



Vemos que si para antes de llegar al Entry Point en el error que no deja arrancar al dumpeado,.

Ya que estamos miremos que hay en la iat

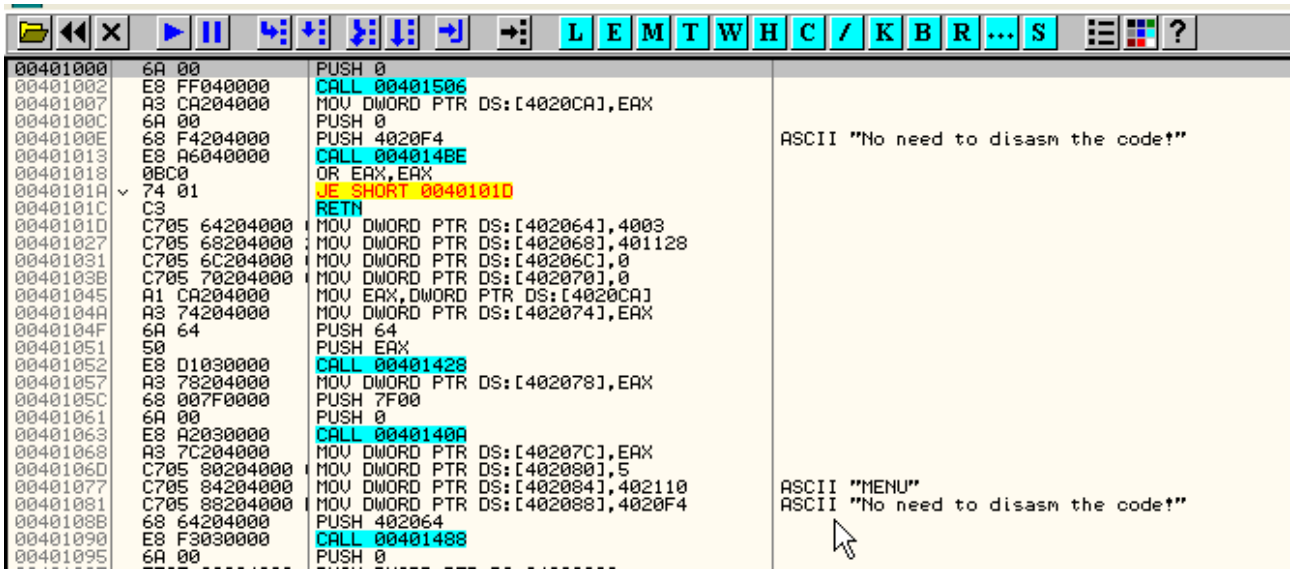


Alli vemos que el dumpeador, logcamente guardo los valores que habia en la iat al momento de llegar al OEP que son los valores correctos en mi maquina de cada API, pero el tema como habiamos visto es que para que un exe arranque sin error y en cualquier maquina, en el ejecutable, alli debe haber un puntero a cada string con el nombre de la api, y el ejecutable contiene el mismo valor este que estamos viendo, que el sistema esta tratando de leer y al no hallar el puntero a la string de la api, le da error.

Pues no hay que ser un genio para darse cuanta que para que arranque el programa hay que restaurar los nombres de las apis, y los punteros a ellas,si quisiera hacerlo a mano es un trabajo terrible deberia cambiar el contenido de 4031AC por un puntero a la string de MessageBoxA y con ciertos arreglos de algunos punteros, pues arrancaria.

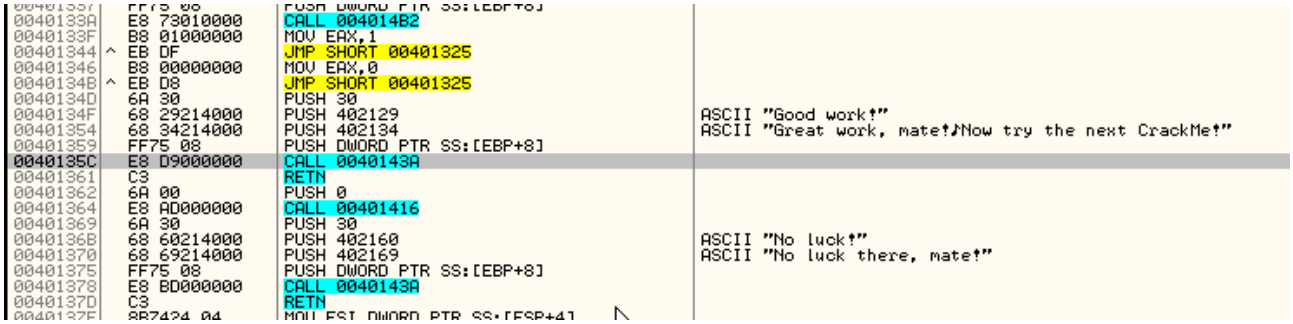
Aquí tenemos planteado el desafio, esto es lo que debemos reparar para que nuestro dumpeado ademas de tener el codigo correcto, que por supuesto lo tiene, corra perfectamente.

De que tiene el codigo correcto no hay duda vayamos a 401000 y miremos que hay alli

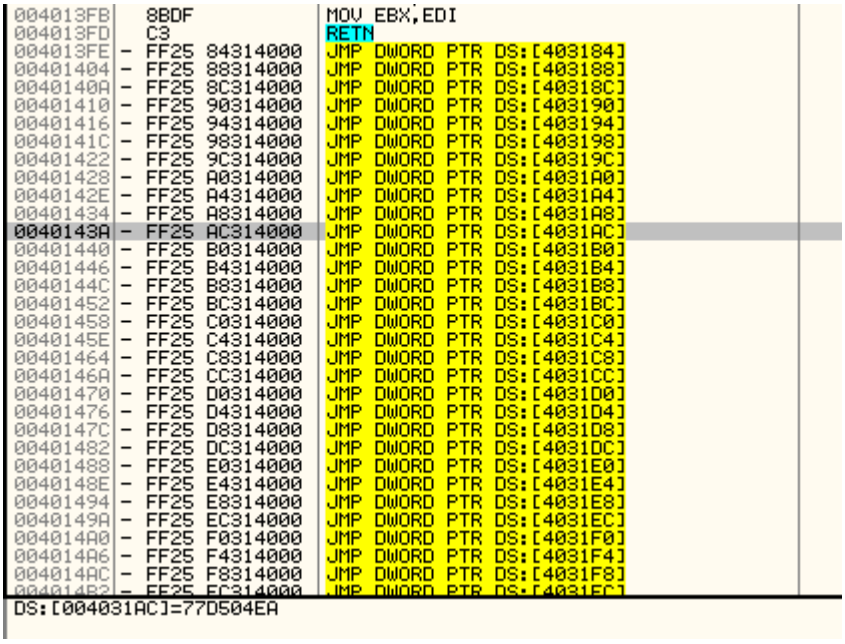


Si vamos a la zona de los saltos a las apis

Alli vemos el call a la MessageBoxA que habiamos visto anteriormente



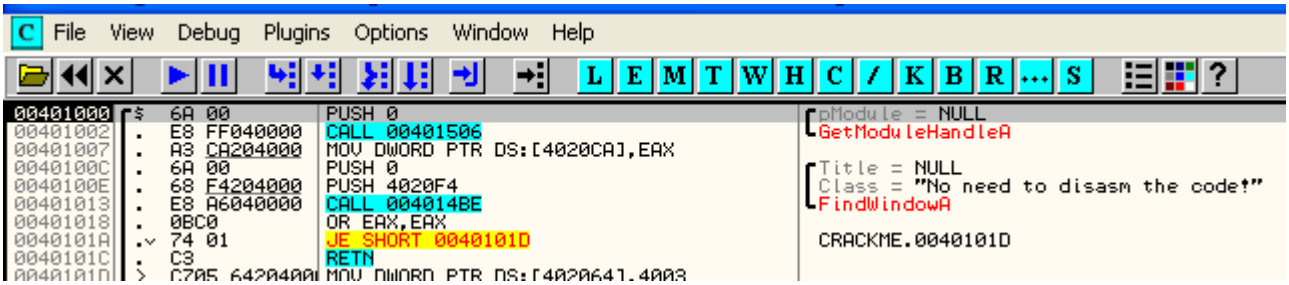
En el CALL el codigo esta correcto, nos lleva al mismo JUMP que antes



Los JUMPS estan lo unico que falta es hacer funcionar el sistema de que llene la IAT con el contenido correcto, para que tenga en la misma antes de correr el programa, los punteros a las strings y que el sistema lo llene con las direciones correctas sin error.

Antes de acometer esta tarea veremos las definiciones y ubicación correcta en el programa original sin empacar de la IAT y la IT.

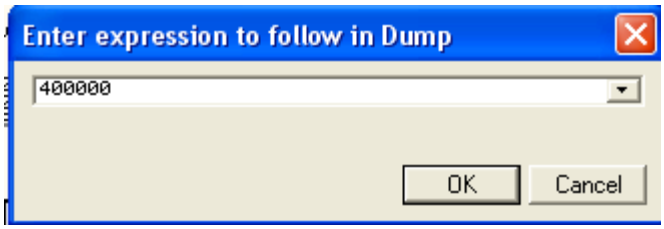
Abramos el CC original en OLLYDBG.

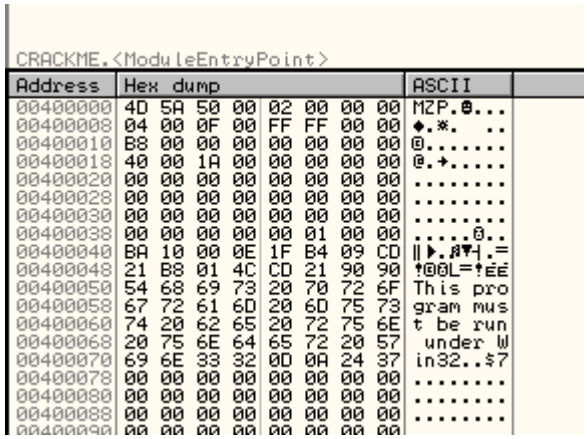


Bueno ubicaremos las partes que hacen que este sistema funcione bien en el original.

En el header, tenemos algunos punteros importantes.

Vayamos al mismo

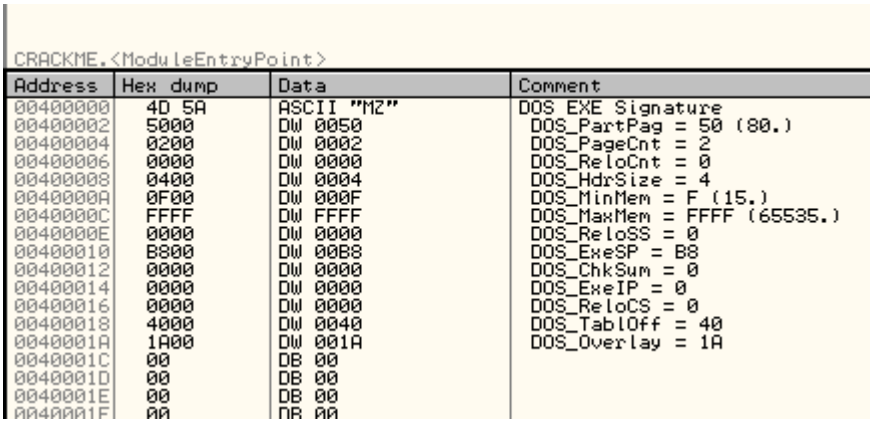




CRACKME.<ModuleEntryPoint>

Address	Hex dump	ASCII
00400000	4D 5A 50 00 02 00 00 00	MZP.☻...
00400008	04 00 0F 00 FF FF 00 00	♦.☼. ..
00400010	B8 00 00 00 00 00 00 00	©.......
00400018	40 00 1A 00 00 00 00 00	@.→.....
00400020	00 00 00 00 00 00 00 00	....
00400028	00 00 00 00 00 00 00 00	....
00400030	00 00 00 00 00 00 00 00	....
00400038	00 00 00 00 00 01 00 00	.....☺..
00400040	BA 10 00 0E 1F B4 09 CD	‖►.♬▼┤.=
00400048	21 B8 01 4C CD 21 90 90	↑©☺L=↑ÉÉ
00400050	54 68 69 73 20 70 72 6F	This pro
00400058	67 72 61 6D 20 6D 75 73	gram mus
00400060	74 20 62 65 20 72 75 6E	t be run
00400068	20 75 6E 64 65 72 20 57	under W
00400070	69 6E 33 32 0D 0A 24 37	in32..$7
00400078	00 00 00 00 00 00 00 00	....
00400080	00 00 00 00 00 00 00 00	....
00400088	00 00 00 00 00 00 00 00	....
00400090	00 00 00 00 00 00 00 00	

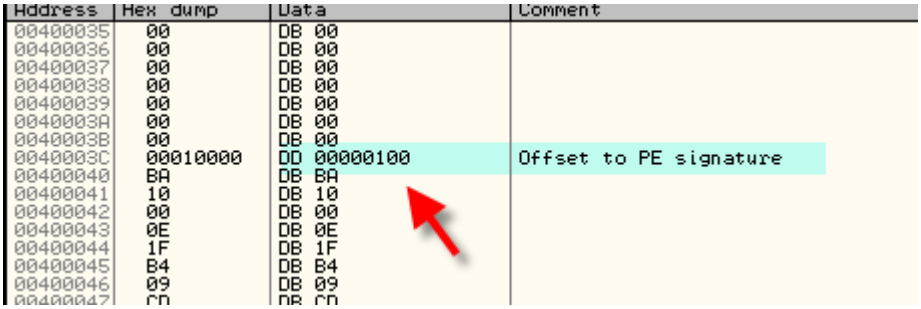
Ahora cambiemoslo a modo SPECIAL-PE HEADER



CRACKME.<ModuleEntryPoint>

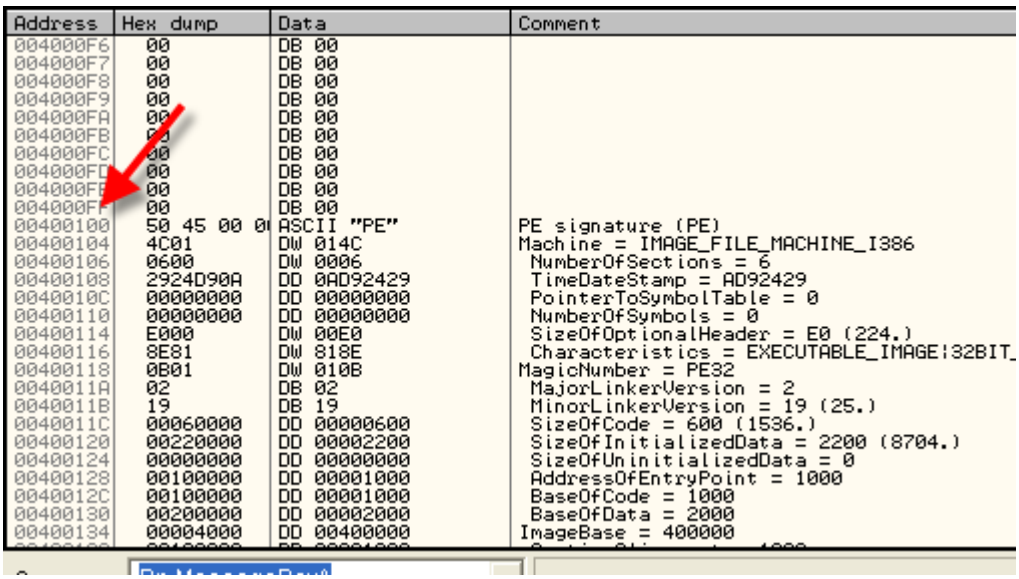
Address	Hex dump	Data	Comment
00400000	4D 5A	ASCII "MZ"	DOS EXE Signature
00400002	5000	DW 0050	DOS_PartPag = 50 (80.)
00400004	0200	DW 0002	DOS_PageCnt = 2
00400006	0000	DW 0000	DOS_ReloCnt = 0
00400008	0400	DW 0004	DOS_HdrSize = 4
0040000A	0F00	DW 000F	DOS_MinMem = F (15.)
0040000C	FFFF	DW FFFF	DOS_MaxMem = FFFF (65535.)
0040000E	0000	DW 0000	DOS_ReloSS = 0
00400010	B800	DW 00B8	DOS_ExeSP = B8
00400012	0000	DW 0000	DOS_ChkSum = 0
00400014	0000	DW 0000	DOS_ExeIP = 0
00400016	0000	DW 0000	DOS_ReloCS = 0
00400018	4000	DW 0040	DOS_TablOff = 40
0040001A	1A00	DW 001A	DOS_Overlay = 1A
0040001C	00	DB 00	
0040001D	00	DB 00	
0040001E	00	DB 00	
0040001F	00	DB 00	

Bajemos



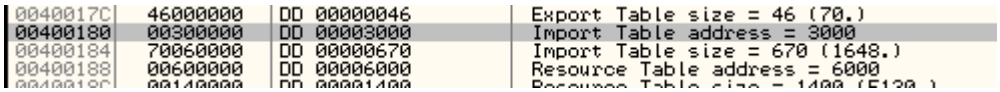
Address	Hex dump	Data	Comment
00400035	00	DB 00	
00400036	00	DB 00	
00400037	00	DB 00	
00400038	00	DB 00	
00400039	00	DB 00	
0040003A	00	DB 00	
0040003B	00	DB 00	
0040003C	00010000	DD 00000100	Offset to PE signature
00400040	BA	DB BA	
00400041	10	DB 10	
00400042	00	DB 00	
00400043	0E	DB 0E	
00400044	1F	DB 1F	
00400045	B4	DB B4	
00400046	09	DB 09	
00400047	CD	DB CD	

Los datos realmente empiezan en 100 donde comienza la PE SIGNATURE



Address	Hex dump	Data	Comment
004000F6	00	DB 00	
004000F7	00	DB 00	
004000F8	00	DB 00	
004000F9	00	DB 00	
004000FA	00	DB 00	
004000FB	00	DB 00	
004000FC	00	DB 00	
004000FD	00	DB 00	
004000FE	00	DB 00	
004000FF	00	DB 00	
00400100	50 45 00 00	ASCII "PE"	PE signature (PE)
00400104	4C01	DW 014C	Machine = IMAGE_FILE_MACHINE_I386
00400106	0600	DW 0006	NumberOfSections = 6
00400108	2924D90A	DD 0AD92429	TimeDateStamp = AD92429
0040010C	00000000	DD 00000000	PointerToSymbolTable = 0
00400110	00000000	DD 00000000	NumberOfSymbols = 0
00400114	E000	DW 00E0	SizeOfOptionalHeader = E0 (224.)
00400116	8E81	DW 818E	Characteristics = EXECUTABLE_IMAGE!32BIT_
00400118	0B01	DW 010B	MagicNumber = PE32
0040011A	02	DB 02	MajorLinkerVersion = 2
0040011B	19	DB 19	MinorLinkerVersion = 19 (25.)
0040011C	00060000	DD 00000600	SizeOfCode = 600 (1536.)
00400120	00220000	DD 00002200	SizeOfInitializedData = 2200 (8704.)
00400124	00000000	DD 00000000	SizeOfUninitializedData = 0
00400128	00100000	DD 00001000	AddressOfEntryPoint = 1000
0040012C	00100000	DD 00001000	BaseOfCode = 1000
00400130	00200000	DD 00002000	BaseOfData = 2000
00400134	00004000	DD 00400000	ImageBase = 400000

El chico no miente alli empieza jeje en 400100 o sea en offset 100.



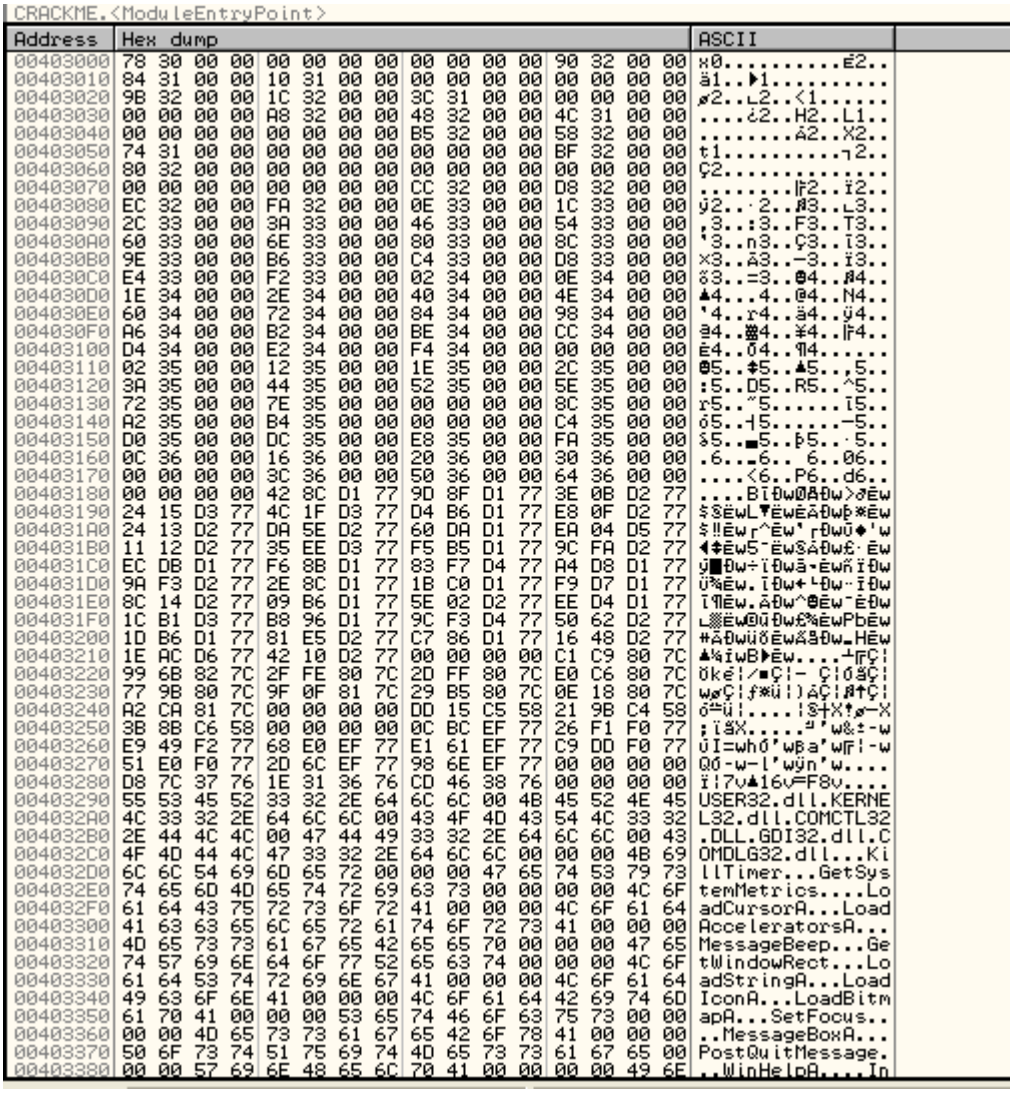
0040017C	46000000	DD 00000046	Export Table size = 46 (70.)
00400180	00300000	DD 00003000	Import Table address = 3000
00400184	70060000	DD 00000670	Import Table size = 670 (1648.)
00400188	00600000	DD 00006000	Resource Table address = 6000

alli tenemos los punteros a la IT o sea la Import Table que no debemos confundirla con la IAT a pesar de que tengan nombres parecidos.

IT= IMPORT TABLE
IAT=IMPORT ADDRESS TABLE

Como ya vimos la IAT es el deposito donde el sistema guarda las direcciones correctas de mi maquina al arrancar, y que es la IT, pues vayamos alli, como vemos dice que empieza en 3000 (403000) y su largo es 670 o sea que termina en 403670, vayamos a mirar.

Como esta fuera del header quitamos el MODO SPECIAL-PE HEADER



Esto que parece un chorizo sin interpretacion posible, es facil de interpretar, si sabemos que es cada cosa, lo detallaremos y veran que esto es muy importante.

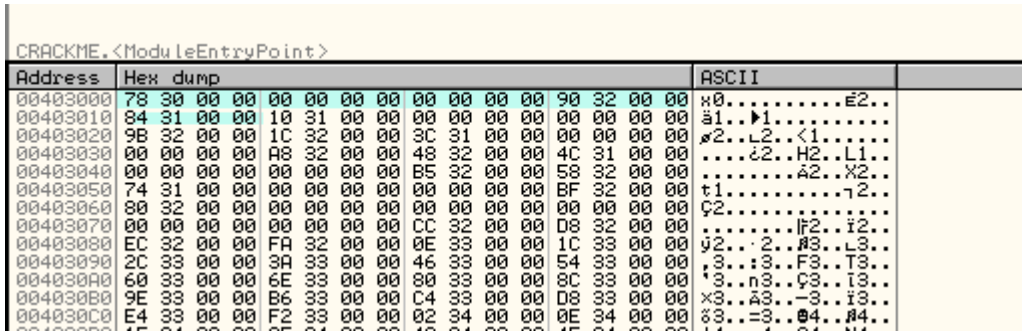
Esto es la famosa IT, aquí la tenemos si comprendemos esto, y aunque no sepamos los nombres de cada puntero, sepamos que es cada cosa y donde apunta, pues estamos salvados.

Lo malo que tenemos para explicar esto en OLLYDBG es que es mucho mas comodo y visible hacerlo en una visualizacion de 5 columnas asi queda ordenado, ya veremos por que.

La famosa IT esta compuesta por los denominados IMAGE IMPORT DESCRIPTORs que son varias lineas de 5 valores dword, de la cual hay una por cada dll que cargara el programa.

Veamos la IT, por eso les digo que esto se ve mejor con 5 columnas para ordenar cada IID uno en cada linea.

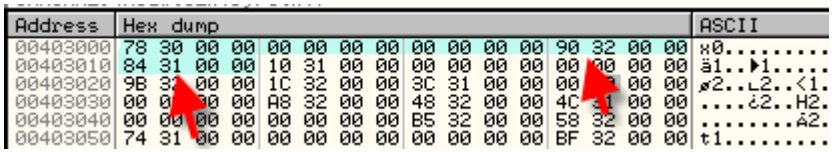
Pero bueno no nos queda otra aquí vemos el primero



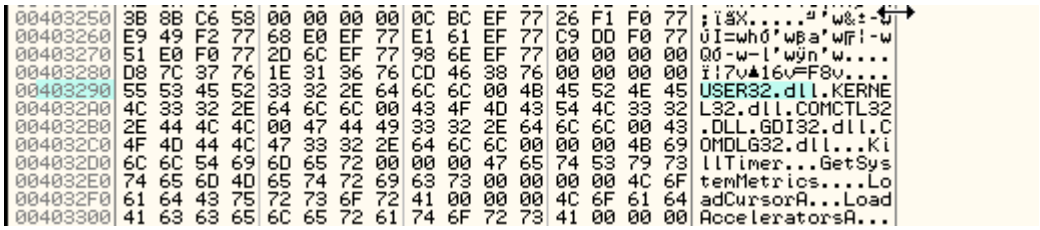
Ese es el primer IID de nuestra IT y tiene 5 DWORDS cuyos nombres son estos

- OriginalFirtsThunks.
- TimeDateStamp.
- ForwarderChain.
- Puntero al nombre de la dll.
- FirtsThunk (Apunta a donde debe buscar en la IAT la primera entrada de dicha dll).

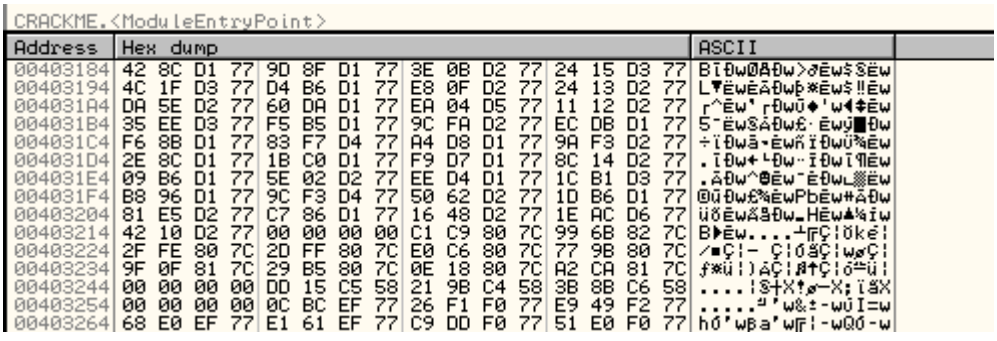
Los tres primeros no son importantes para el cracking hasta siendo cero, normalmente arranca igual el programa, son para usos muy determinados, los importantes son el 4 y el 5to puntero de nuestro primer IID.



Alli esta como vemos el 4to apunta al nombre de la dll a la cual pertenece este IID, veamos cual es la dll en 403290.



Alli esta la primera dll que el sistema buscara es USER32.dll y el 5to puntero marca donde comienza en la IAT las entradas de esta dll, en este caso es 403184.



Alli esta es la IAT, y su primera entrada, todo esto esta dentro de la IT, ya que terminaba en 403670, asi que tenemos que la famosa IT, tiene una linea para cada dll, estas lineas son llamadas IID o IMAGE IMPORT DESCRIPTOR y dentro de la IT esta la IAT o deposito de las direcciones de las apis, todo compacto y para

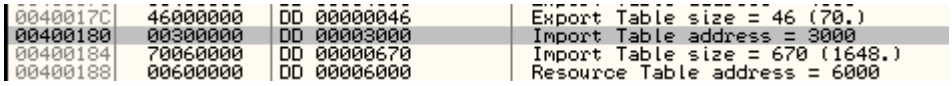
el uso del sistema.

Muchos crackres experimentados diran que no siempre la IAT esta dentro de la IT, ya que el 5to puntero de cada IID puede apuntar a cualquier lado y tranquilamente puede estar fuera, ubicada en cualquier lugar del programa que tenga permiso de escritura, para que cuando arranque el exe, alli el sistema vaya guardando las direcciones correctas de las apis, la cuestion es que ya vemos como trabaja mejor el sistema para llenar la IAT,

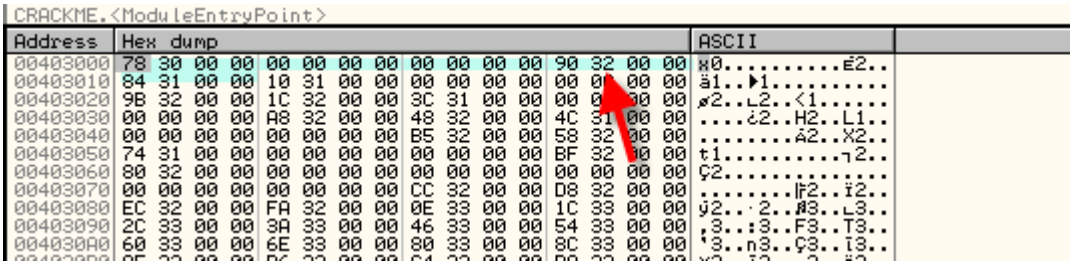
1) Busca la IT
2) Busca la primera IID y se fija en el 4to puntero a que dll pertenece
3) Luego mira en el 5to puntero donde esta la primera entrada de la IAT
4) Alli hay un puntero a la string con el nombre de la api
5) Con GetProcAddress busca la direccion y la guarda en la misma entrada.
6) Cuando encuentra en la IAT una entrada con ceros, eso le indica que termino la primera dll y que debe pasar a mirar el segundo IID para buscar cual es la segunda, y repetir el proceso.

O sea que si el sistema hara esto en imagenes y es importante que lo entiendan bien porque si al sistema le falta algo o encuentra punteros incorrectos dara error y es importante entenderlo a fondo.

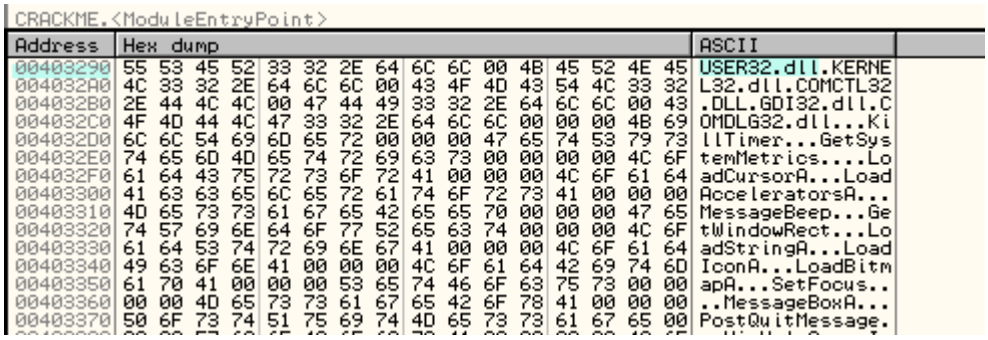
1)Busca la direccion de la IT



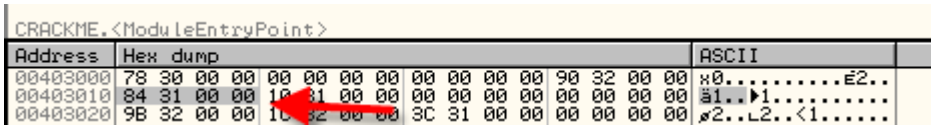
2)Va a esa direccion

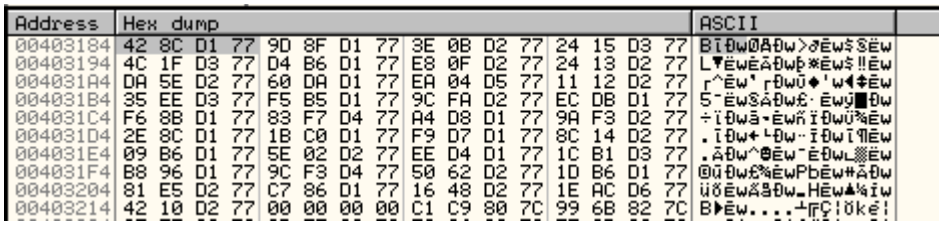


3)En el primer IID busca el 4to DWORD que le apunta al nombre de la dll donde empezara a trabajar en este caso sera USER32.dll.

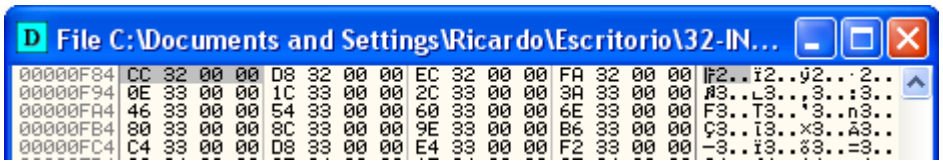


Luego mira el 5to puntero para buscar la primera entrada de la IAT que es 403184 y va alli.

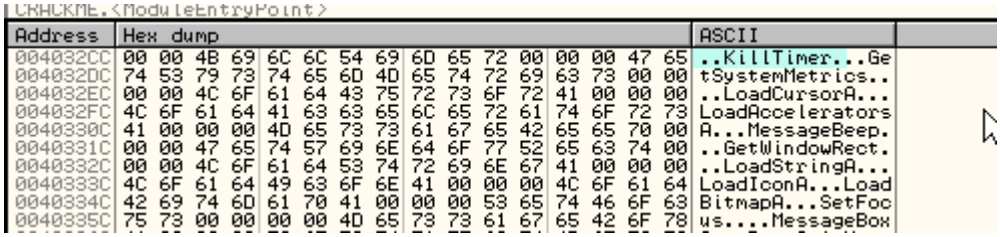




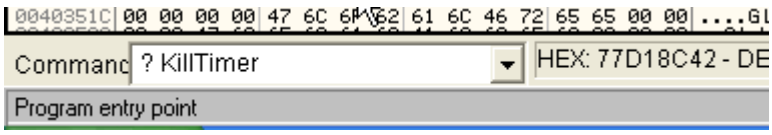
Alli lee el valor que no es ese pues alli ya esta machacado, sino el que esta en el ejecutable, veamos



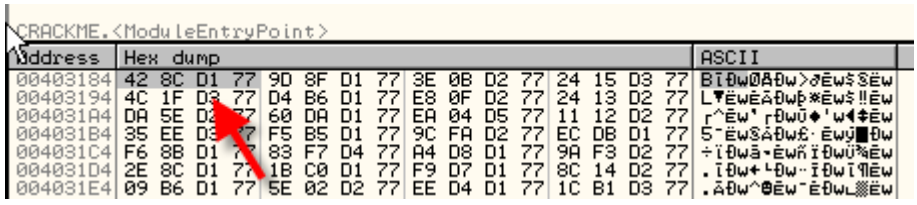
alli ve que el nombre de la primera api esta en 4032CC, va alli



Ve que la primera api es KillTimer con GetProcAddress halla su direccion que nosotros hallamos con el OLLYDBG.

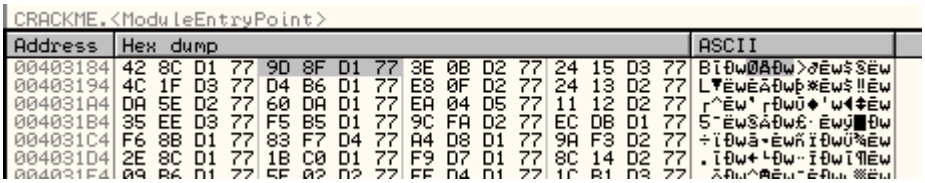


Y ese valor que en mi maquina es 77d18c42 lo guarda en la misma entrada de la IAT machacando el existente.



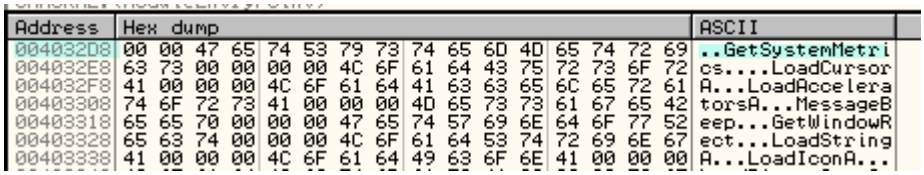
Alli lo vemos es la primera entrada de la IAT ya cuando llego al entry point y muestra el valor que tiene esa api en mi maquina.

Asi saltara a la siguiente entrada de la IAT, que halla facilmente sumandole 4 a esta, y busca el siguiente puntero al nombre de la proxima api.

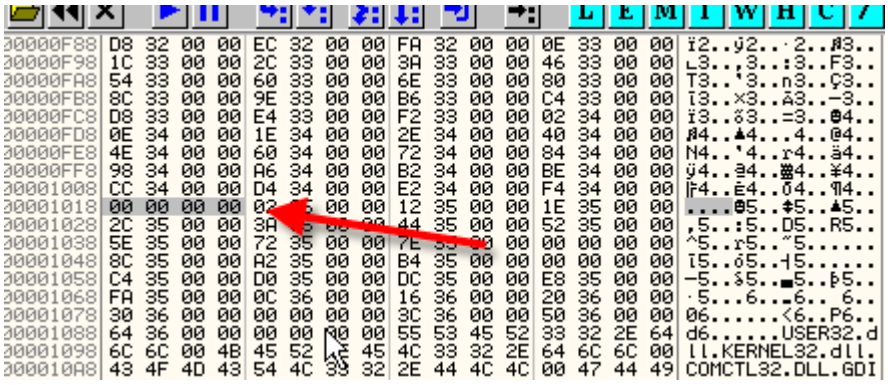


Por supuesto mira en el ejecutable y este es

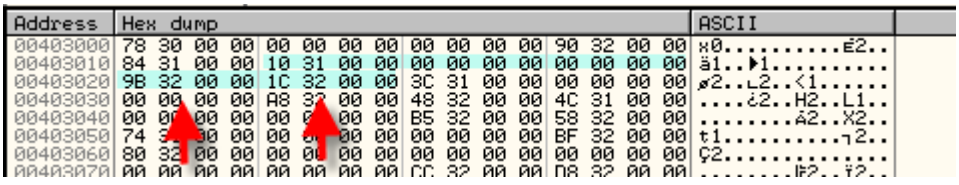
32d8 o sea 4032d8, mira el nombre de la api siguiente alli y como no hallo un cero en la entrada, sabe que continua con las apis de user32.dll.



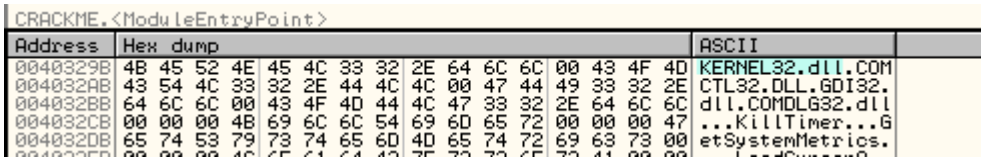
La segunda api es GetSystemMetrics hallara la direccion y la guardara en la segunda entrada de la iat y asi, seguira buscando apis en user32.dll hasta que llegue a una entrada con todos ceros si vemos la iat en el ejecutable.



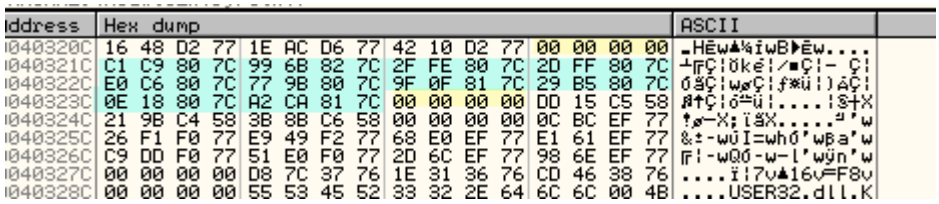
Vemos que seguira buscando en user32.dll hasta que halle ese cero donde retorna al siguiente IID



Cuyos 4 y 5to puntero son esos, el 4to le dice que en 40329b esta el nombre de la 2 dll donde buscara nombres.



Que es Kernel32.dll y la primera entrada de la iat donde buscara sera la que esta a continuacion del cero o sea 40321c.



Alli vemos el cero que determino que no hay mas apis de user32.dll y en 40321c empiezan las de Kernel32.dll donde buscara en el ejecutable el primer puntero a la string y llenara aquí con el valor de la api correcta.

Creo que es muy importante entender bien esto, trate de explicarlo y repetirlo varias veces para que quede pero creo que con solo una leida eso no queda en la mente traten de grabarselo con fuego pues esto es la base de todo el tema de la reconstruccion de las IATs y aunque ya veremos en la parte siguiente que hay tools que nos ayudan a hacer el trabajo pesado sin tener que revisar todo esto, es muy importante saber como trabaja todo, asi cualquier problema o caso raro, no nos quedaremos a un costado del camino y siempre podremos entender que esta pasando.

En la parte siguiente arreglaremos IATs no lloren jeje ya tienen suficiente con comprender como trabajan por ahora jejejejeje.

Hasta la 34
Riardo Narvaja
27/02/06