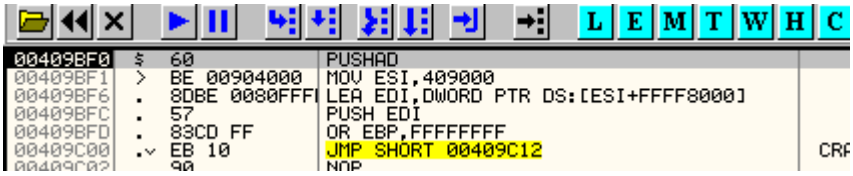
INTRODUCCION AL CRACKING CON OLLYDBG PARTE 34

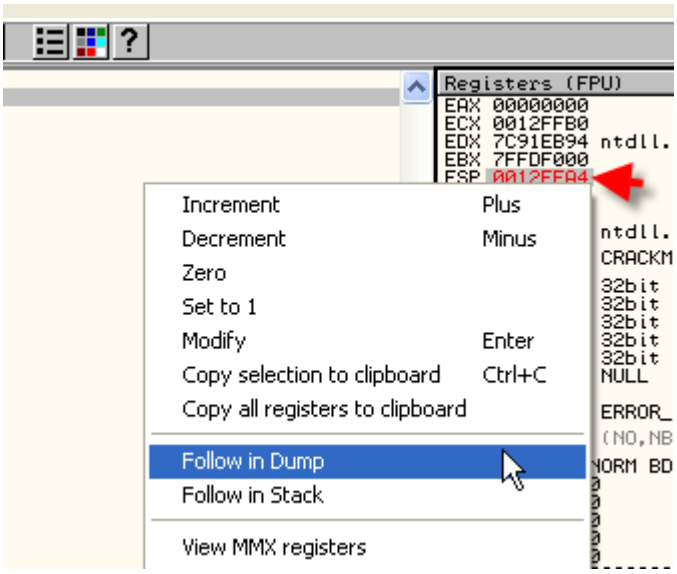
En la parte 33 vimos como funciona el sistema de la IT y IAT, yo se y los que saben reparar una IAT, pueden pensar que no es necesario saber como funciona, ya que hay tools que reparan una iat casi automaticamente, pero yo les aseguro que es bueno saber como funciona y que ocurre en cada momento, porque hay muchos packers que engañan a las tools y las hacen fallar, por lo cual en esos casos hay que saber razonar y pensar que esta pasando para hacer alguna correcion a mano, o poder moverse con facilidad.

Empezaremos con algo facil el Crackme de Cruehead empacado con UPX, realizaremos el proceso de desempacado completo aquí para unir todo lo que vimos y al final repararemos la IAT, para que quede funcional.

Como vimos el primer paso era llegar al OEP, por lo cual abro en OLLYDBG el CC o sea el Crackme de Cruehead.

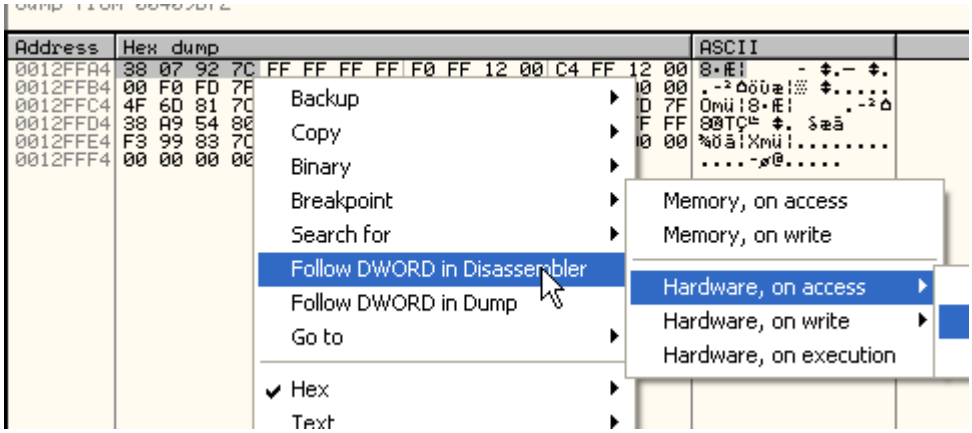


Aplicare el metodo del PUSHAD para llegar al OEP apreto f7 para pasar el PUSHAD.



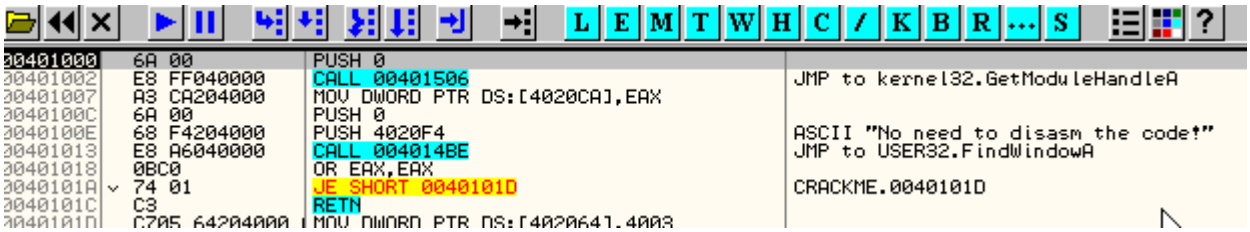
Ahora marco ESP-FOLLOW IN DUMP.

Y en el dump vere los registros que guardo, y marcare los primeros 4 bytes, y pondre un HARDWARE BPX ON ACCESS.



Y doy RUN, parara en el salto al OEP.

Bueno ya estoy en el salto al OEP, apreto f7 y llego al mismo.



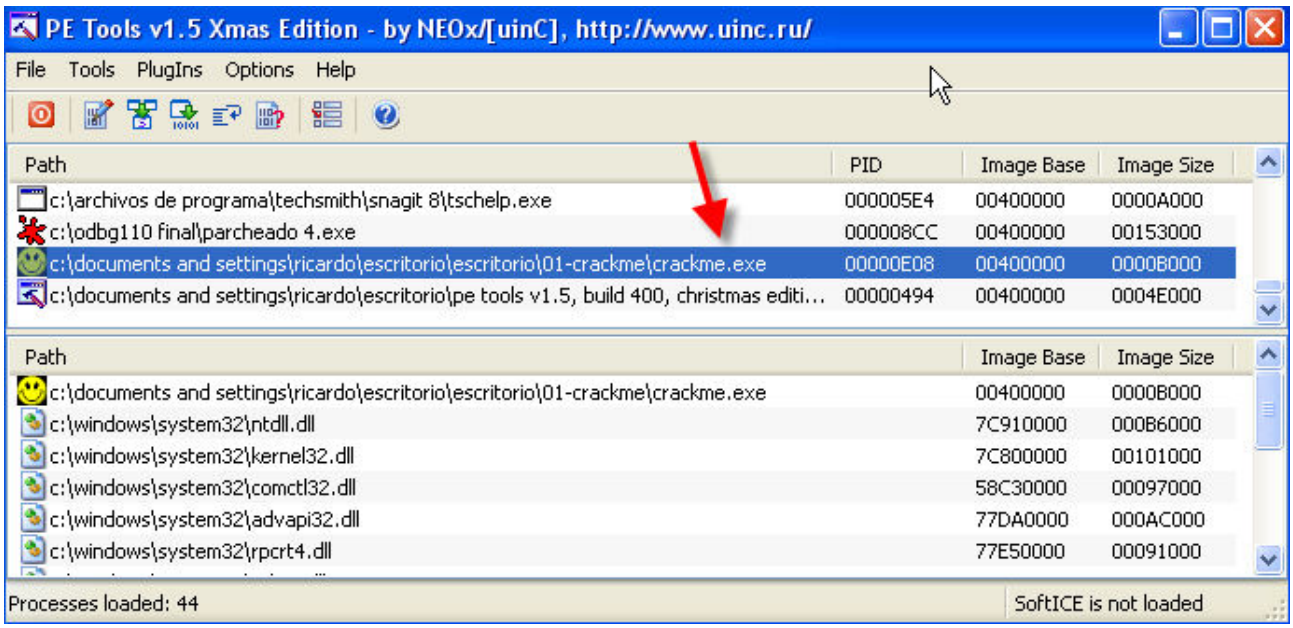
Bueno ya llegamos, al punto donde el programa esta desempacado en memoria ahora lo dumpearemos.

Como vimos en partes anteriores, existen muchos dumpeadores, incluso el OLLYDBG tiene un plugin llamado OLLYDMP que dumpea muy bien, pero como ya vimos como se hace con el LORD PE, ahora utilizaremos PE -TOOLS que pueden bajarse desde:

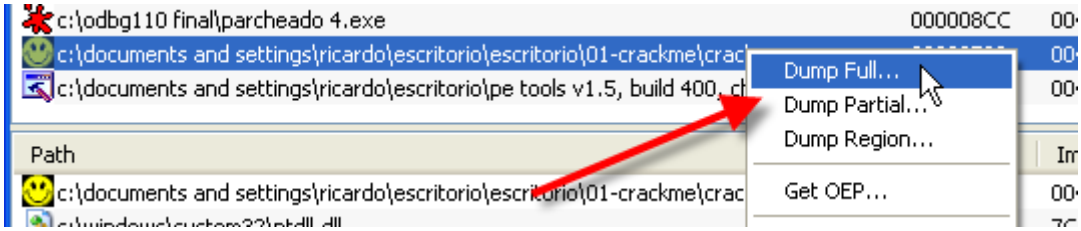
http://www.uinc.ru/files/neox/PE_Tools.shtml

Como nos cansaremos de desempacar, y de practicar mas adelante en futuros desempacados tambien usaremos el plugin OLLYDMP asi aprendemos a usar todos.

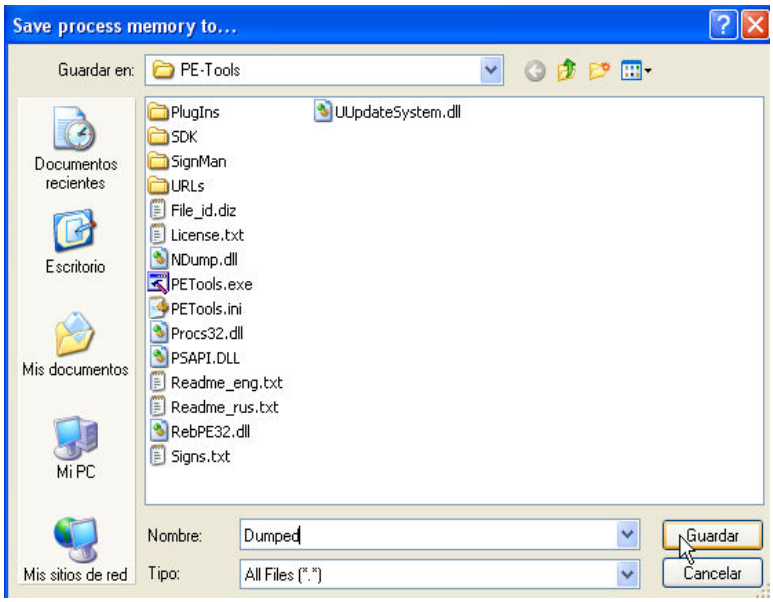
Abramos el PE-TOOLS sin cerrar el OLLYDBG que esta detenido en el OEP.



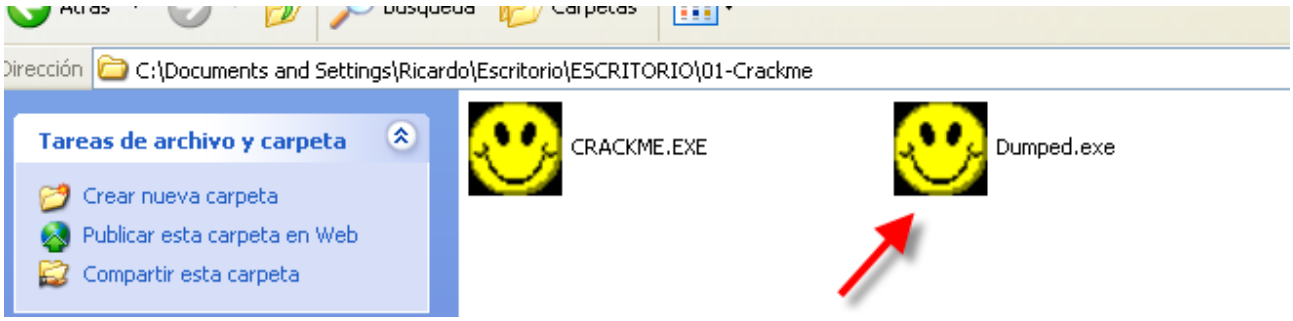
Bueno alli esta el proceso que en este caso se llama crackme.exe ya que perdi el que se llamaba crackmeUPX y lo hice de nuevo y me olvide de cambiarle el nombre, pero es lo mismo, se llame como se llame es el crackme de cruehead empacado con UPX y detenido en el OEP.



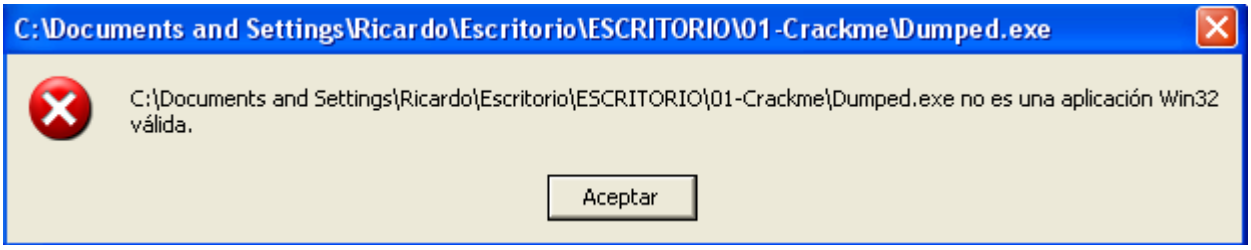
Haciendo click derecho-DUMP FULL



Bueno ya esta dumpeado ahora vamos a la reparacion de la IAT cerramos el PE TOOLS vimos que el DUMPED lo guardo en la carpeta del PE TOOLS, asi que lo buscamos y lo copiamos a la carpeta donde esta el mismo Crackme de Cruehead empacado con UPX.



Bueno ya sabemos que aun falta reparar la iat, igual probamos ver que pasa si lo ejecutamos, hacemos doble click en el dumped.exe y..

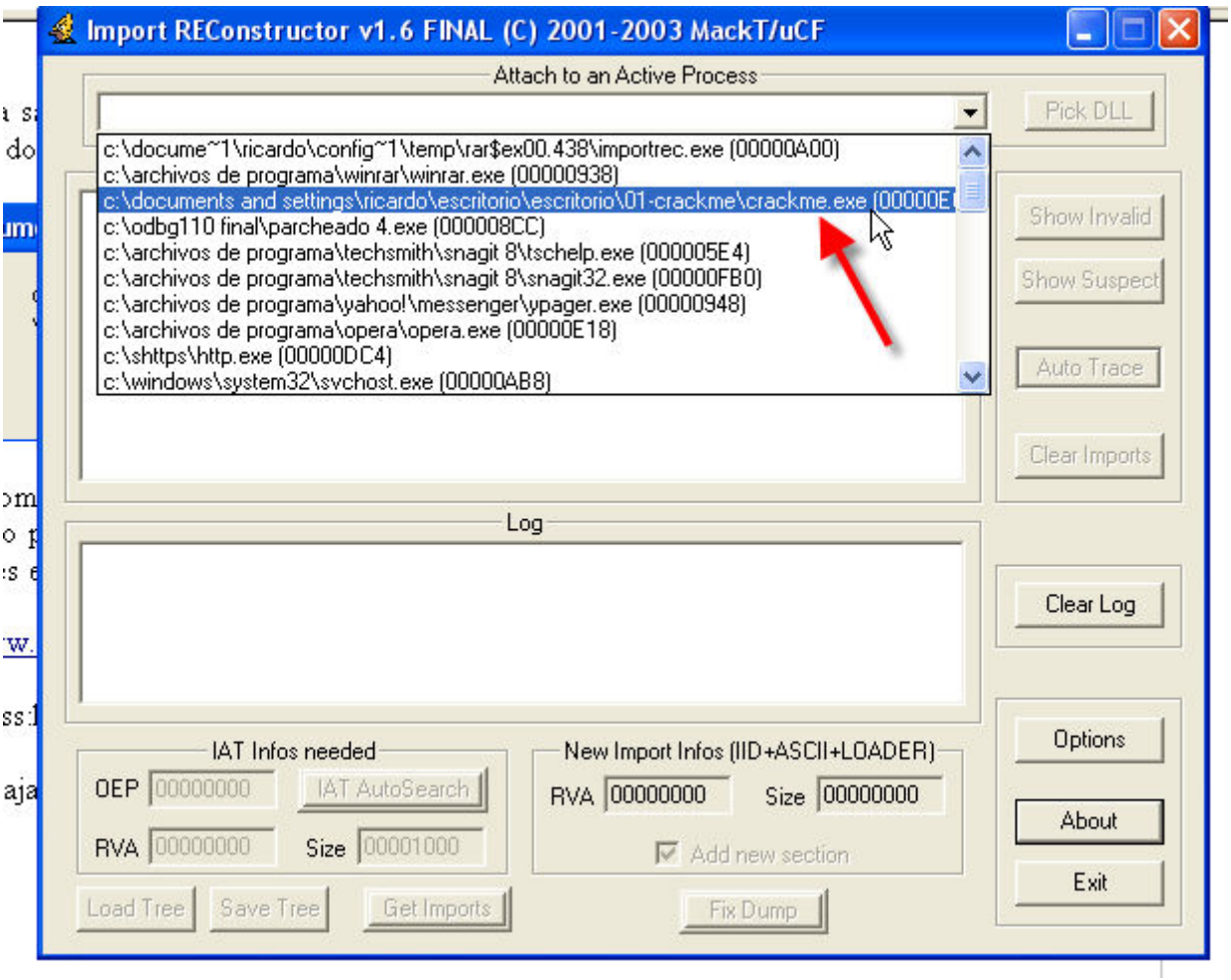


Bueno como ven hay que reparar la IAT, aunque corriera en nuestra maquina, debemos reparar la IAT para que funcione en cualquier maquina y no solo en la nuestra, para ello usaremos el IMPORT RECONSTRUCTOR, por supuesto no cerramos el OLLYDBG con el CC empacado con UPX detenido en el OEP, pues el IMP REC trabaja sobre el.

http://www.ricnar456.dyndns.org/HERRAMIENTAS/F-G-H-I-J-K/ImportReconstructor16f.zip

user y pass:hola

pueden bajarlo desde alli.



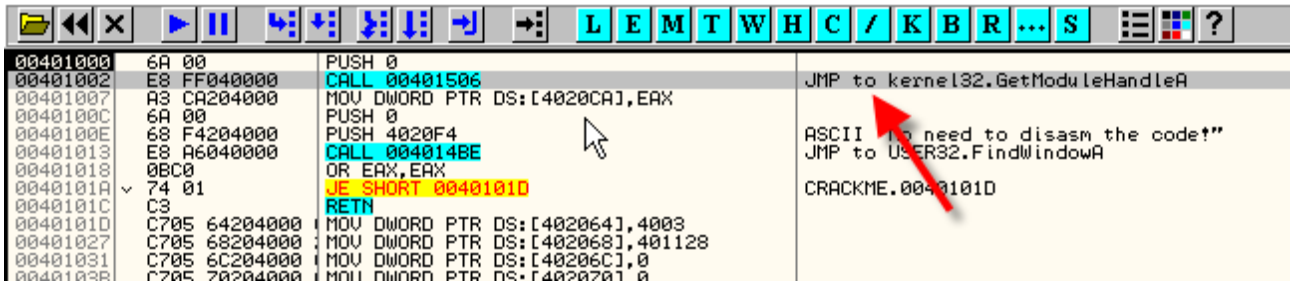
Lo abrimos y buscamos el proceso del crackme de cruehead con UPX que esta detenido en el OEP.

Ahora bien un tema que complica a muchisimos newbies es hallar el inicio y final de la IAT, por supuesto en el crackme que tenemos detenido en el OEP ya vimos que el packer destruyo la IT, por lo cual el primer Image Import descriptor cuyo 4 puntero marca el nombre de la dll y el 5to marca la primera entrada de la iat correspondiente a esa dll, no los tenemos, por lo cual debemos utilizar otros metodos para hallarla.

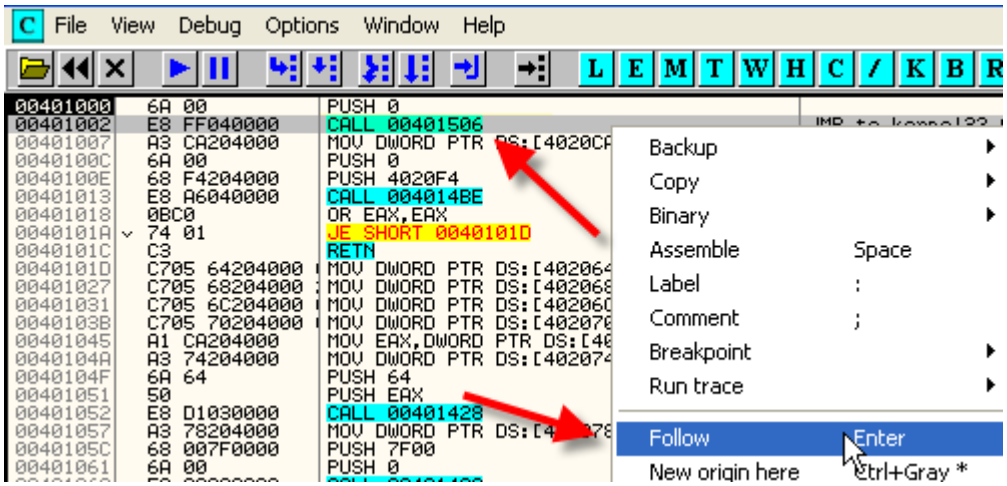
Por supuesto sabemos que las llamadas a las apis generealmente son realizadas con JMPs indirectos o CALLs indirectos del tipo.

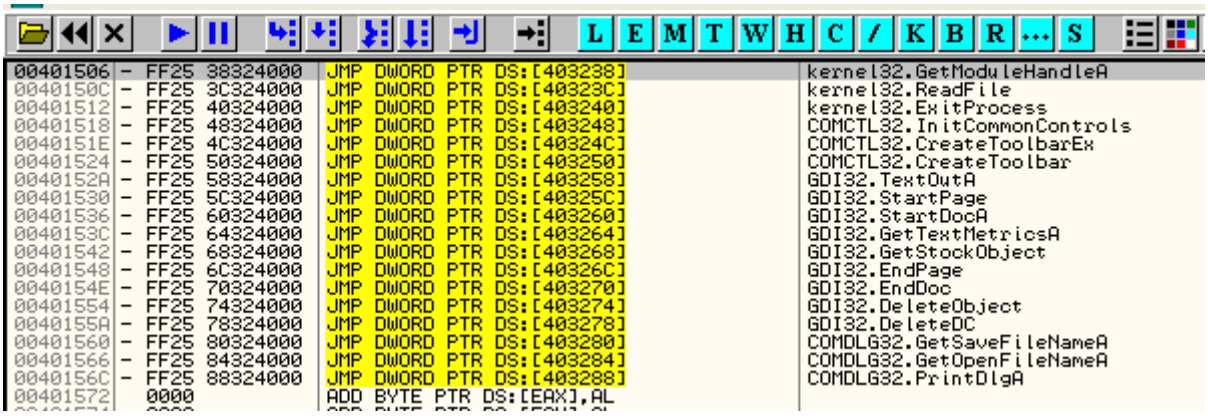
JMP [xxxxxxx] o CALL [xxxxxx]

y como hemos visto en la parte anterior el programa toma la direccion de la api en nuestra maquina de la IAT, que es el deposito de direcciones de las apis en nuestra maquina, busquemos un salto a una api en el empacado, para lo cual miremos en el OLLYDBG con el CC empacado, parado en el OEP.



Alli en la segunda linea vemos un CALL que ollydbg nos dice que ira a una api, aunque previo paso por los JMPS INDIRECTOS, asi que marquemos esa linea y hagamos FOLLOW

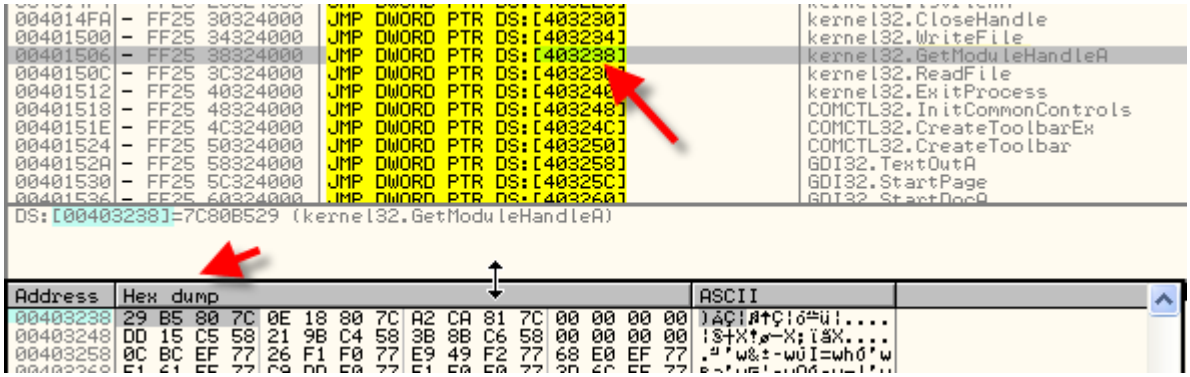




Alli encontramos la tabla de saltos que tomando valores de la IAT, nos lleva a cada API, como vemos estos saltos comienzan con los opcodes FF 25, por lo cual muchas veces veran en tutes que directamente haces un search for bynary string y buscan FF 25, y llegan hasta aquí mas rapidamente.

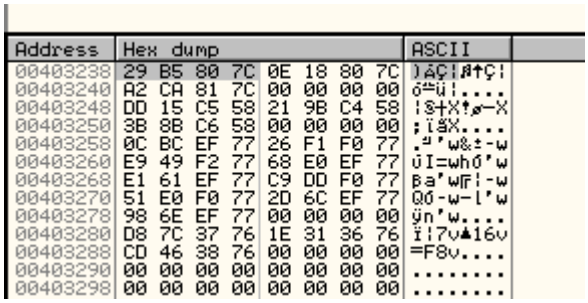
El tema es que no todos los programas usan saltos indirectos para llegar a las apis, por lo cual a veces ese metodo falla, pero lo mejor y que nunca falla es buscar una llamada a una api, y ver de donde toma el valor guardado que nos llevara a la api, y ese valor tiene que estar guardado en la IAT.

Aquí en el ejemplo JMP [403238]

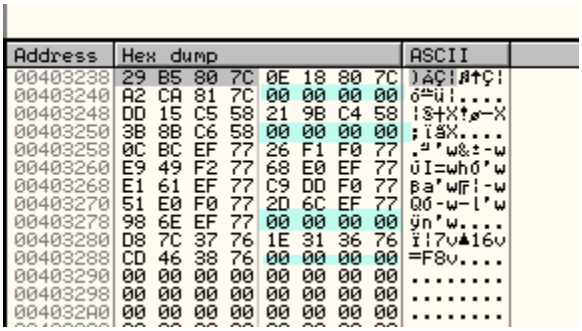


Alli esta bien claro 403238 es una entrada de la IAT donde esta guardada la direccion de la api GetModuleHandleA, de esta forma en el DUMP estamos viendo parte de la IAT, lo que necesitamos es ver donde comienza y donde termina la misma.

Por supuesto mirar todos los JMPS INDIRECTOS y ver cual es la minima y maxima direccion podria ser un metodo, aunque es muy lento, lo mejor es ir al DUMP e ir subiendo de a poco, y como sabemos cada entrada tiene una direccion de una api, en este caso es 7C80B529, que vista al reves es 29 B5 80 7c, cambiare a la vista de dos culumnas para que se aprecie mas.



Alli vemos la organización de la IAT, vimos en la parte anterior que estan continuadas todas las entradas de la misma dll y luego la separacion para comenzar con la siguiente dll, es una entrada con ceros, si marcamos los ceros de separacion.

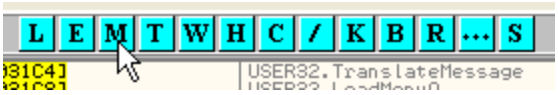


Algunos packers mas sofisticados sobreescriben los ceros con basura para que se haga mas dificil la reconstruccion, total esas entradas no se usan, y como ya no necesita arrancar el programa, no necesita mantener los ceros, pero aquí estan y como ejemplo vemos que entre ellos en una misma dll las direcciones son cercanas ya que van a la misma seccion donde se encuentra esa dll, si ven abajo de

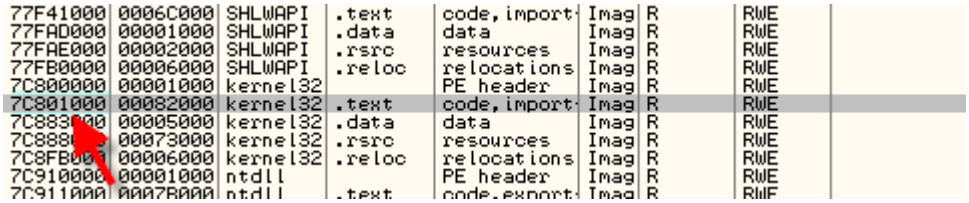
00403238 **29 B5 80 7C** **0E 18 80 7C**)µ€|##€|
00403240 **A2 CA 81 7C** 00 00 00 00

hay tres apis que van a direcciones 7Cxxxxxx y luego esta la entrada con los ceros que separa de la siguiente dll.

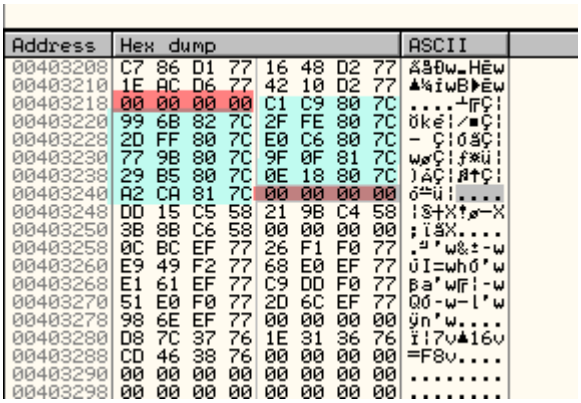
Si vemos en VIEW-M veremos a que seccion CODE de que dll corresponden esas direcciones tipo 7Cxxxxxx



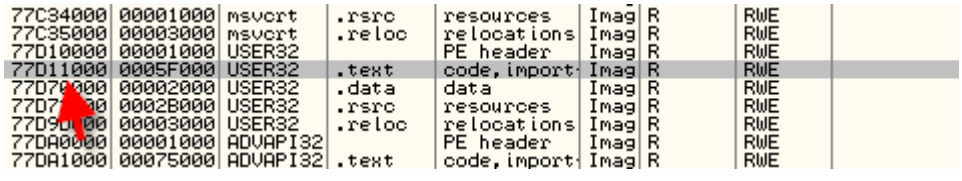
Pues alli vemos que todas esas direcciones estan comprendidas dentro de la seccion code de la kernel32.dll.



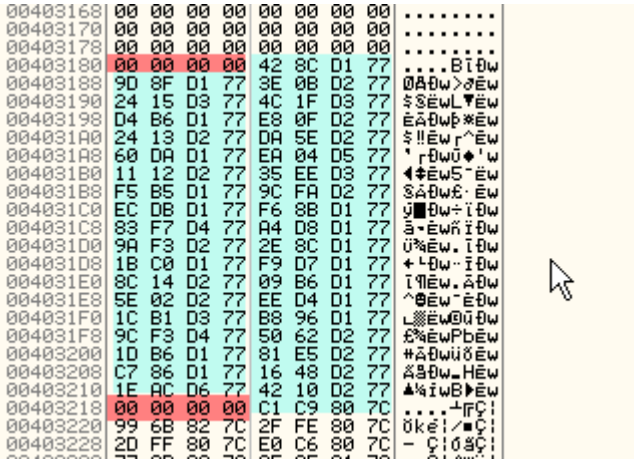
Por supuesto en sus maquinas las dll pueden estar ubicadas en otras direcciones, pero aquí veo que en mi maquina esas entradas corresponde a la seccion CODE de la kernel32.dll o sea que las direcciones de las entradas contiguas caen todas alli en esa dll.



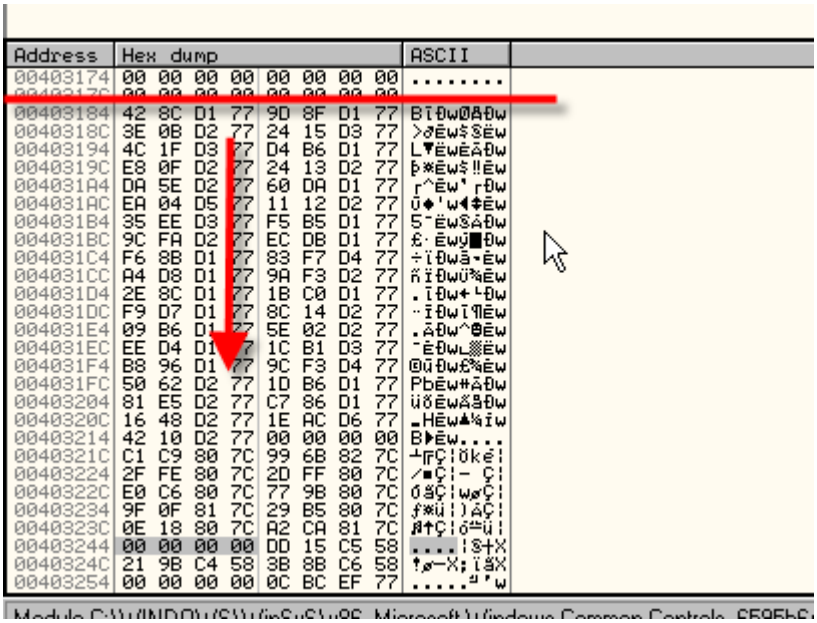
Pues alli vemos todas las entradas correspondientes a la kernel32.dll, vemos la separacion con ceros marcada en rojo y mas arriba hay entradas que van a otra dll en este caso su seccion CODE se encuentra en direcciones cercanas a 77Dxxxxx, miremos en M, a que dll corresponden.



Pues entonces las direcciones que hay arriba de la separacion, corresponden a la seccion CODE de la user32.dll, sigamos subiendo hasta la siguiente separacion.



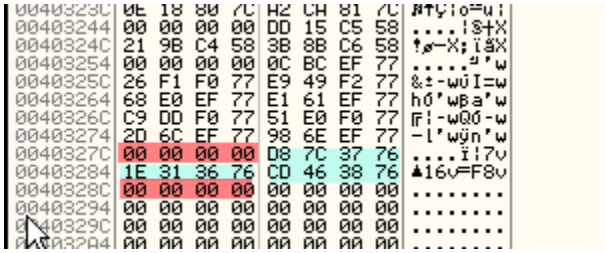
Pues alli vemos todas las apis que caen dentro de la seccion CODE de la user32.dll y la separacion pero arriba ya no hay mas nada quiere decir que el inicio de la iat es 403184, pues esa es la primera entrada valida.



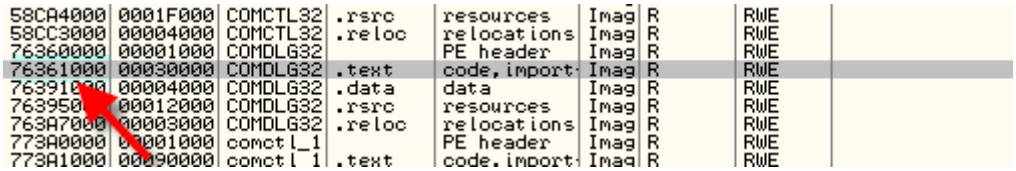
Vemos claramente mas arriba no hay mas entradas que vayan a ninguna dll, y en este caso, ademas hacia arriba hay todos ceros lo cual nos facilita la tarea de ver el inicio de la IAT, algunos packers mas complejos normalmente llenan de basura antes de la IAT y luego de que termine, para que no sea tan facil identificar el inicio, pero si uno sabe que las entradas de la IAT siempre deben ir a la seccion code de una dll, pues el resto en seguida nos damos cuenta lo que es basura, pues no nos lleva a ninguna seccion code de ninguna dll.

Pues alli en la imagen vemos el inicio de la IAT que es 403184, ahora iremos bajando hasta hallar el final de la iat, usando el mismo metodo, mirando siempre que la IAT continuara mientras haya entradas que vayan a una seccion code de una dll.

Tambien mas adelante veremos que hay packers que cambian entradas de la IAT y las redireccionan a rutinas propias, desde la cual saltan a la api, ese caso por supuesto no se da aquí, y lo estudiaremos mas adelante, pero por ahora, sabemos que las entradas de la IAT son direcciones de apis, y que deben llevarnos a secciones code de dlls.



Alli vemos las ultimas entradas de la IAT que van a 76xxxxxx veamos a que dll corresponden.



En este caso corresponden a la sección CODE de COMDLG32.dll y luego de esto no hay mas entradas asi que el final de la IAT para que queden todas las entradas incluidas dentro seria 40328C, podria tomarse tambien que la ultima entrada es 403288 y igual funcionara, pero para mas claridad vemos que termina en 40328c y ponemos esa direccion como fin de la IAT.

Por lo tanto ya tenemos el inicio y final de la IAT

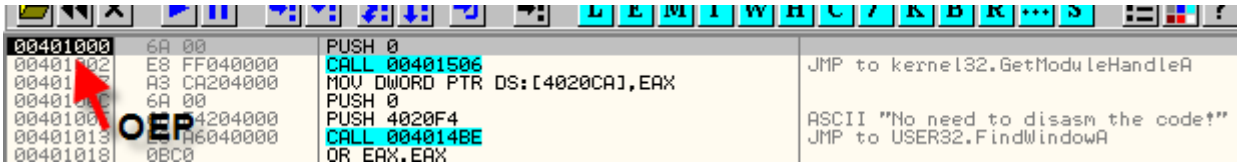
INICIO: 403184
FINAL: 40328C

el IMPORT RECONTRUCTOR nos pide tres datos :
1)el inicio de la IAT, pero hay que restarle a 403184 la imagebase que en este caso es 400000 asi que seria **3184**.
2) El segundo valor que nos pide es el largo de la IAT por lo cual restando el FINAL menos el INICIO tendremos el largo.

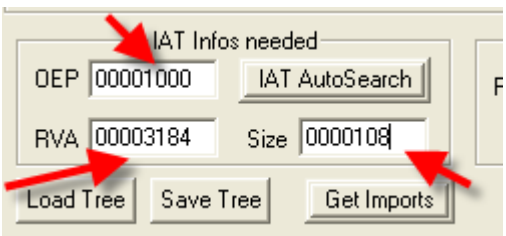
LARGO=40328c-403184=108

por lo cual el segundo dato que nos pide del largo o SIZE sera **108**

3)El tercer dato es el OEP tambien restandole la imagebase seria 401000-400000=**1000**

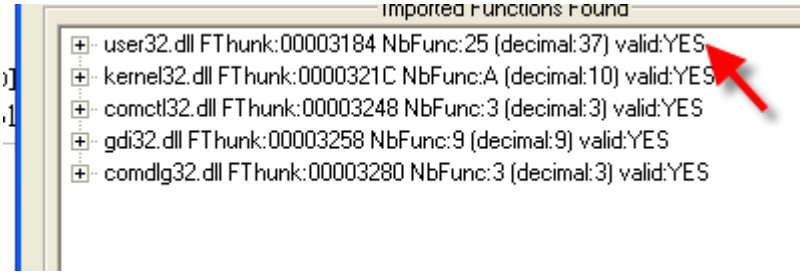


Estos datos los ingresaremos en el IMP REC.



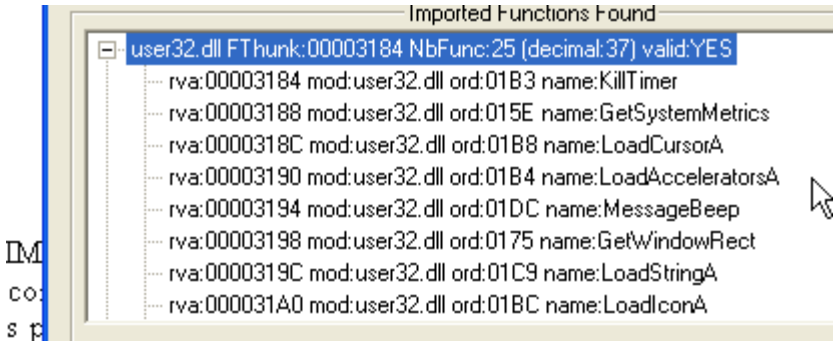
Alli vemos lso daros que pusimos en el IMP REC, en OEP pusimos el 1000 ya que a 401000 le restamos la imagebase, en RVA el inicio de tabla restandole la imagebase tambien y en Size el largo de la IAT.

Ahora apretamos GET IMPORTS

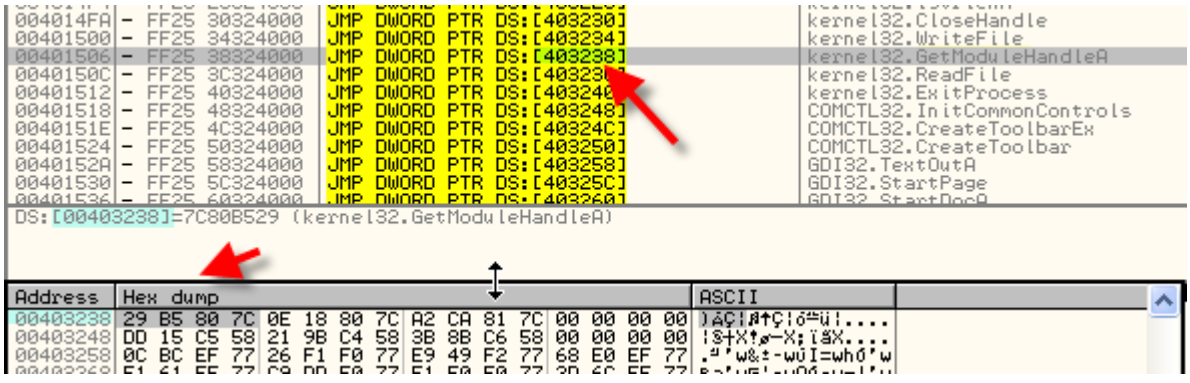


Vemos que el IMP REC lo que hace es hallar que apis pertenece a cada entrada de la IAT, y ademas de hallarla, te comunica si es valida o sea si esta correcta, marcando YES, en el caso de entradas redireccionadas por ciertos packers que no vayan directamente a una API te mostrara NO y en ese caso habra que averiguar esa entrada incorrecta a que api pertenece realmente, arreglarla para que el IMP REC reconozca como entrada correcta y nos diga YES y una vez que esta todo YES como en el caso actual ya podemos reparar el dumpeado.

Antes de hacerlo como nos gusta mirar, veremos que en cada dll, si desplegamos el contenido apretando en el +.

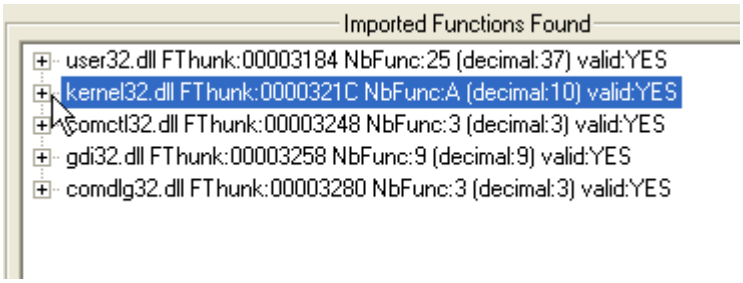


Tenemos cada entrada a que api pertenece y si tenemos ganas podemos ubicar la primera que vimos en la IAT que era la entrada correspondiente a GetModuleHandleA.

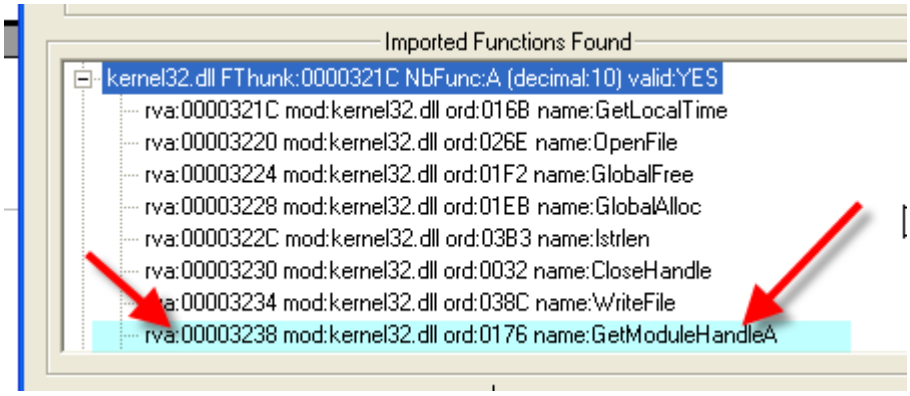


Si recuerdan esa fue la primera entrada que miramos en la IAT la correspondiente a 403238, por supuesto en IMP REC siempre hay que restar la imagebase por lo cual buscaremos 3238 ademas sabemos que pertenece a Kernel32.dll como vemos en la imagen superior al lado del nombre de la

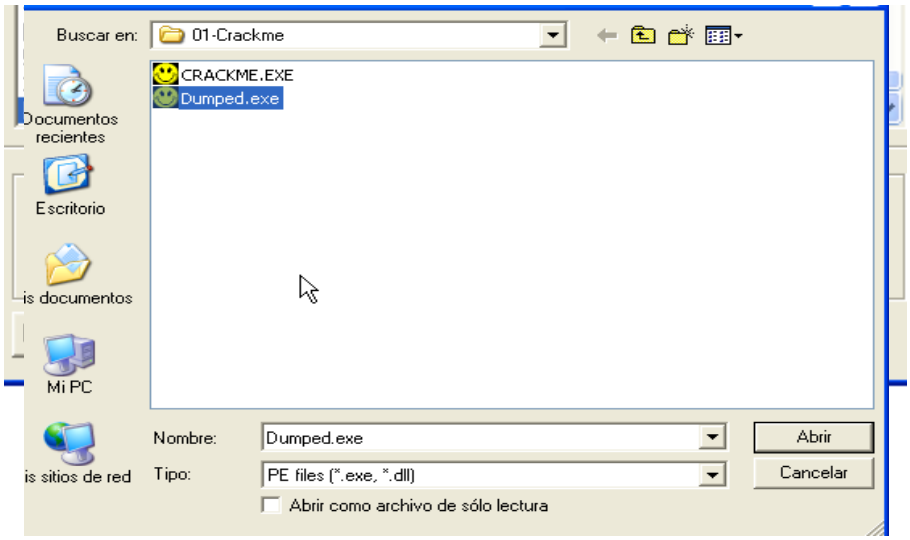
api.



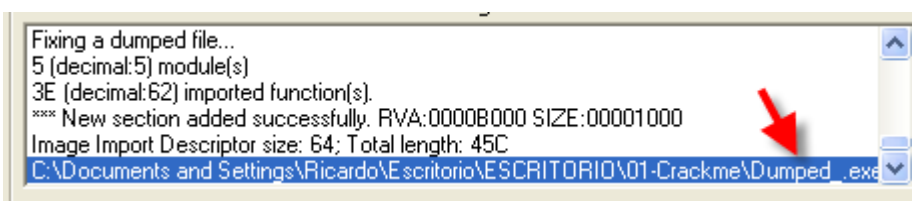
Por lo tanto debemos abrir la kernell32.dll y para ello hacemos click en el + que esta a la izquierda.



Alli vemos la entrada 3238 corresponde a GetModuleHandleA esta todo bien, asi que ahora repararemos el dumpeado, vamos al boton FIX DUMP.



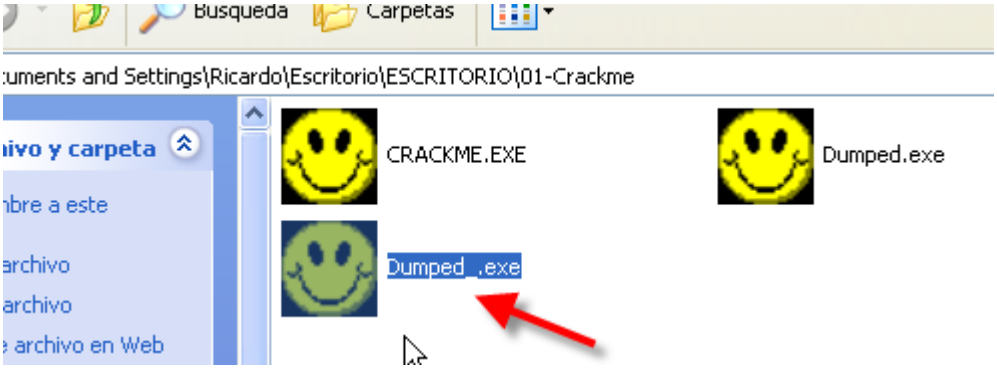
Alli buscamos el dumpeado y lo abrimos



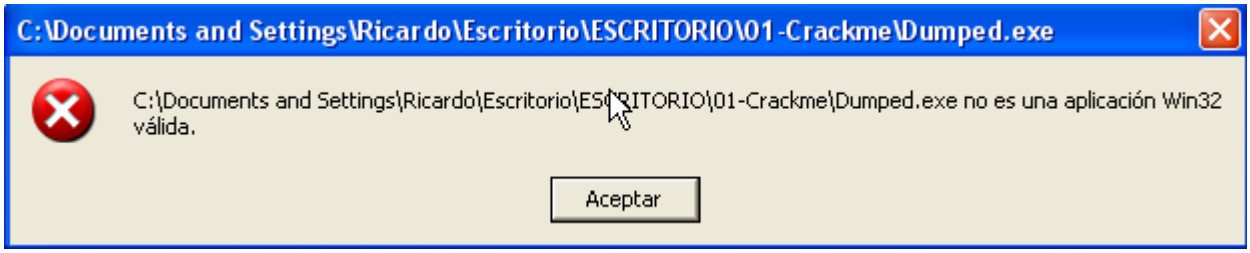
Bueno alli vemos que el IMP REC lo repara aunque no toca el DUMPED que teniamos guardado

sino que crea uno reparado con el nombre DUMPED_.exe.

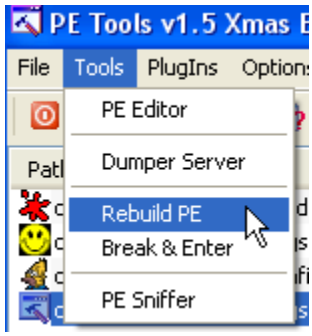
Veamos alli en la carpeta donde esta.



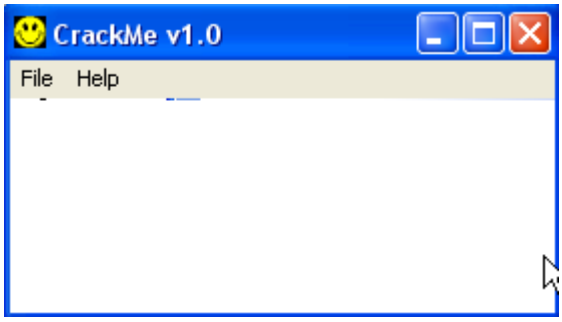
Alli esta podemos intentar ejecutarlo a ver si quedo bien.



Jejejejeje aun parece que falta algo pero a no asustarse que al reparar la IAT muchas veces nos ocurrira esto, el mismo PE TOOLS tiene la solucion abrimos el PE TOOLS



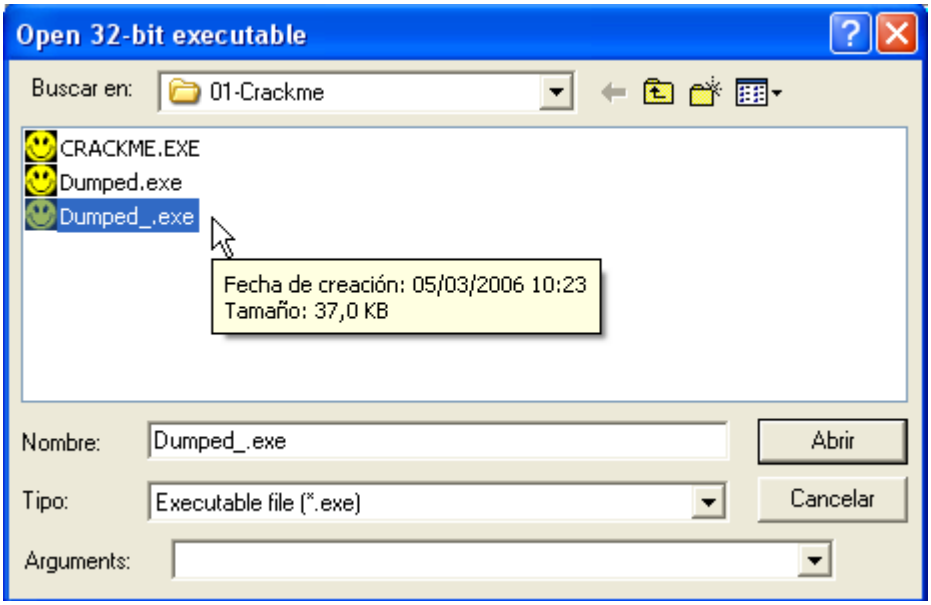
Y vamos a REBUILD PE y buscamos el DUMPED_.EXE y lo repara perfectamente ahora lo ejecutamos yyyy...

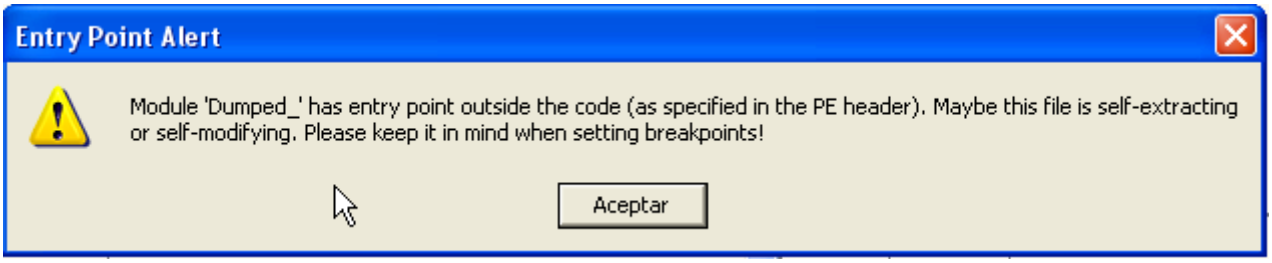


Funcionaaaaaaaaaa y ademas funcionara en cualquier maquina, porque hemos reparado la IAT, de

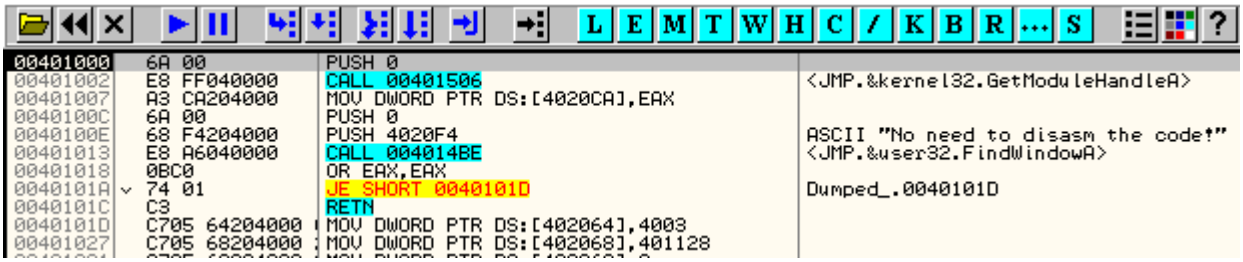
forma que el IMP REC lo que hace es teniendo las apis correctas de cada entrada de la IAT, reescribe los punteros a los nombres de las apis, y arregla la IT poniendo un IID por cada dll como vimos en la parte anterior que debe quedar todo programa para que arranque normalmente sin error.

Si abrimos el DUMPED_.exe en OLLYDBG.

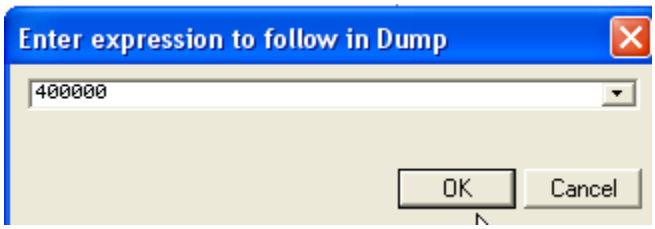


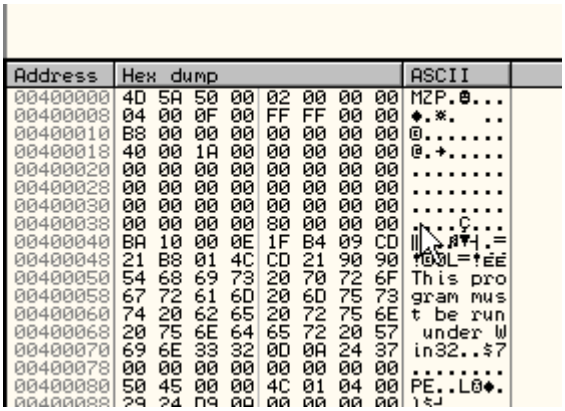


Vemos que OLLYDBG nos dice ahora que el Entry Point, se encuentra fuera de la seccion code y eso es porque el UPX habia cambiado la seccion code a la 3ra, igual podemos arreglar eso.



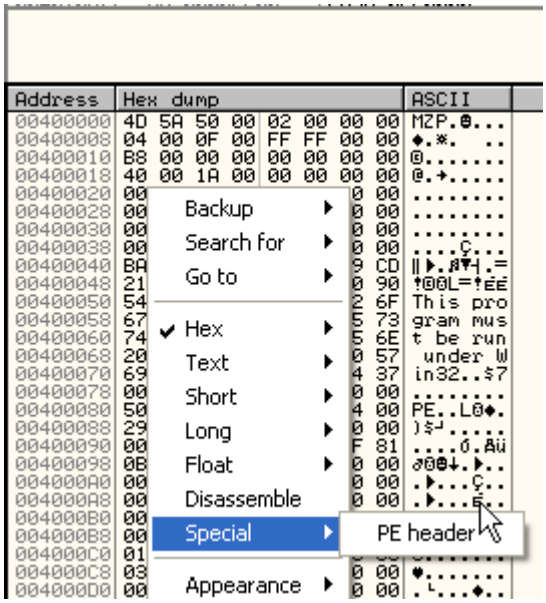
Llegamos al Entry Point y vamos a ver el header con GOTO EXPRESSION=400000

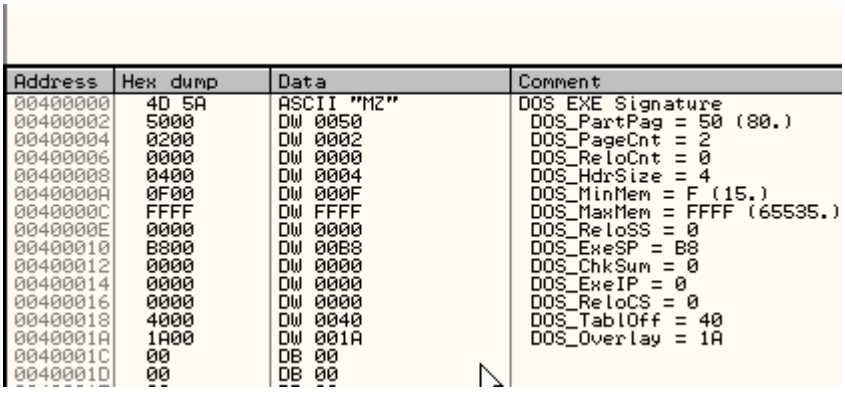




Address	Hex dump	ASCII
00400000	4D 5A 50 00 02 00 00 00	MZP.☻...
00400008	04 00 0F 00 FF FF 00 00	♦.☼. ..
00400010	B8 00 00 00 00 00 00 00	¸.......
00400018	40 00 1A 00 00 00 00 00	@.→.....
00400020	00 00 00 00 00 00 00 00	....
00400028	00 00 00 00 00 00 00 00	....
00400030	00 00 00 00 00 00 00 00	....
00400038	00 00 00 00 80 00 00 00	....Ç...
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..$7
00400078	00 00 00 00 00 00 00 00	....
00400080	50 45 00 00 4C 01 04 00	PE..L☺♦.
00400088	29 24 D9 09 00 00 00 00	)$┘.....

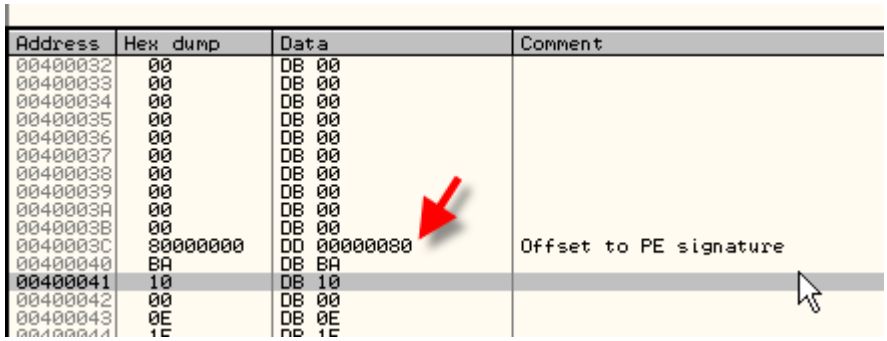
Cambiamos a modo SPECIAL-PE HEADER



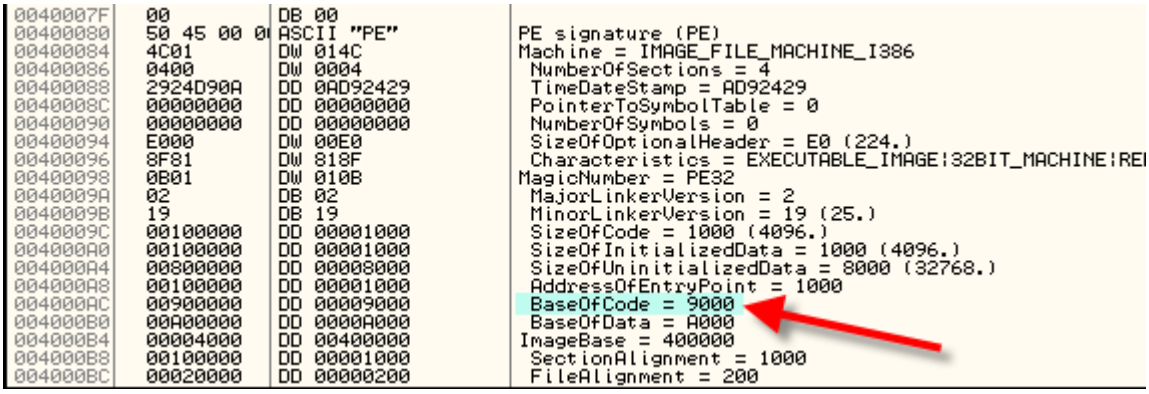


Address	Hex dump	Data	Comment
00400000	4D 5A	ASCII "MZ"	DOS EXE Signature
00400002	5000	DW 0050	DOS_PartPag = 50 (80.)
00400004	0200	DW 0002	DOS_PageCnt = 2
00400006	0000	DW 0000	DOS_ReloCnt = 0
00400008	0400	DW 0004	DOS_HdrSize = 4
0040000A	0F00	DW 000F	DOS_MinMem = F (15.)
0040000C	FFFF	DW FFFF	DOS_MaxMem = 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	

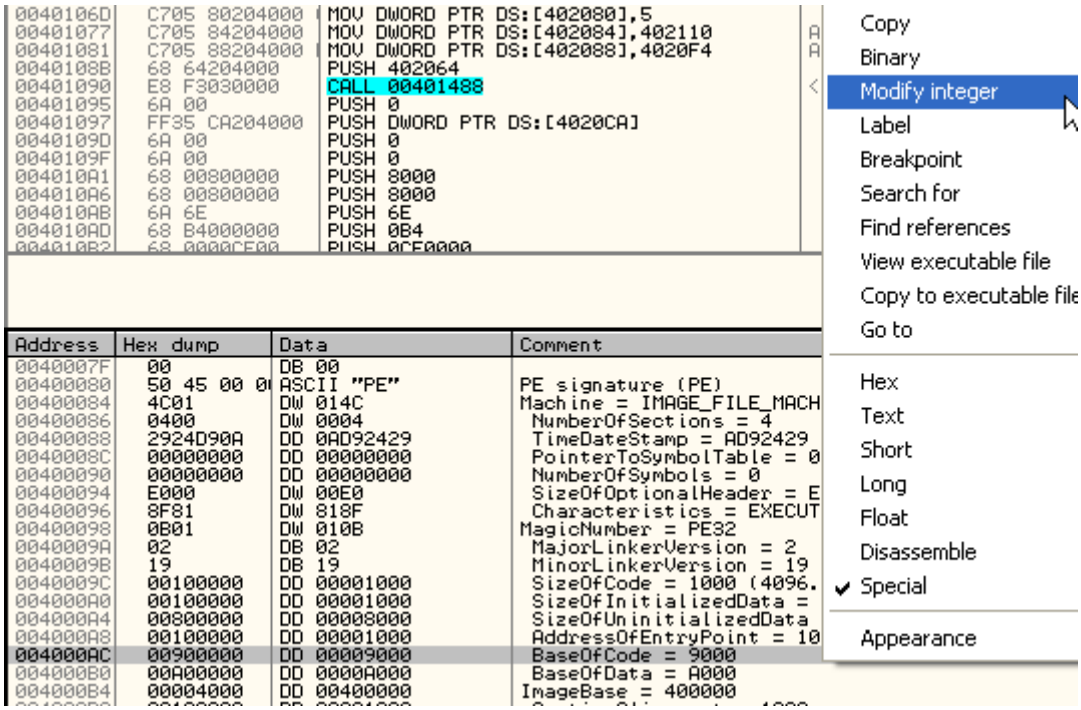
Alli esta bajamos



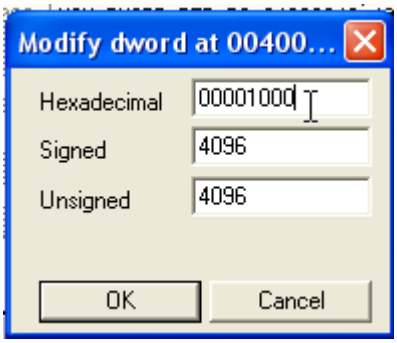
Vemos que la PE SIGNATURE empieza en 80 vayamos alli a 400080.

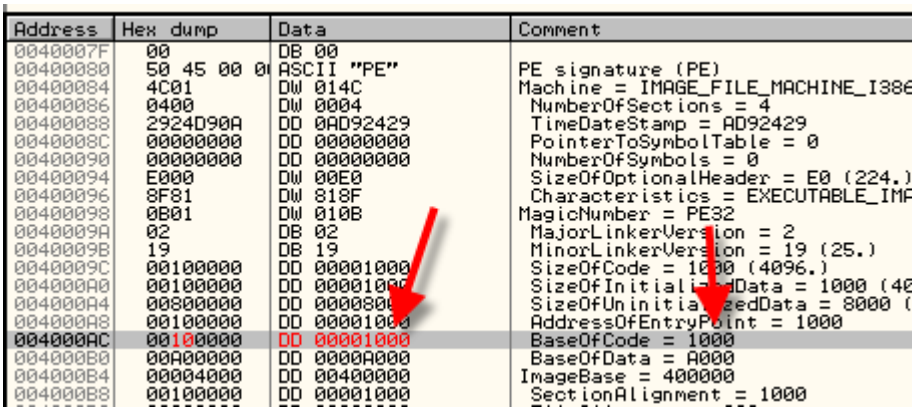


Alli esta el puntero maldito dice BASE OF CODE=9000 o sea queremos que la seccion CODE sea la primera, la que empieza en 401000 asi que cambiamos ese 9000 por 1000.

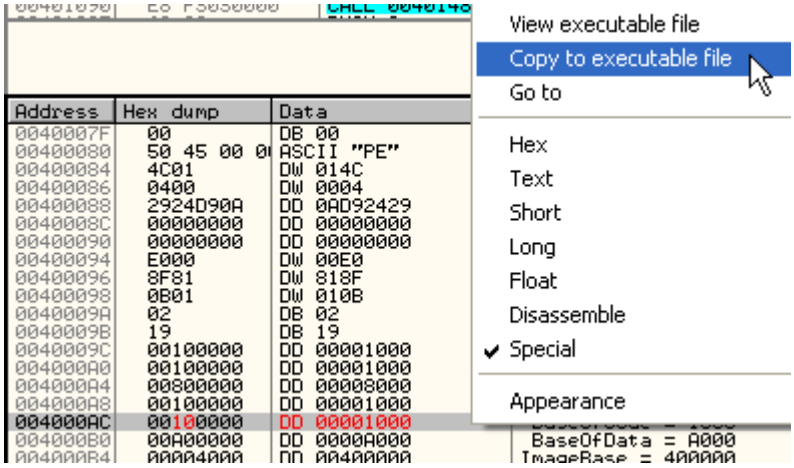


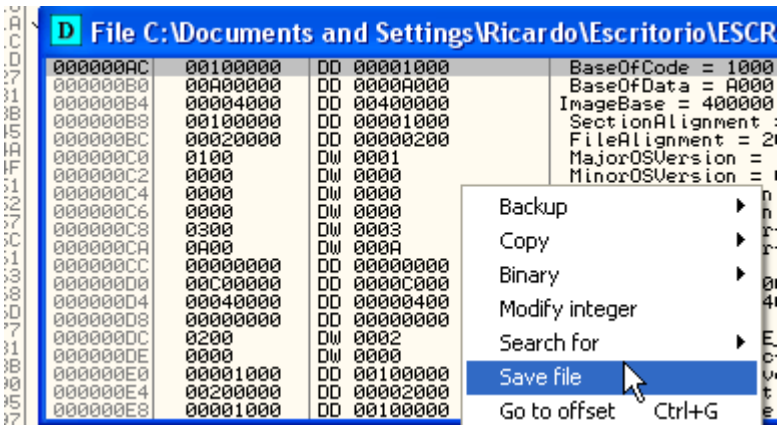
Lo marcamos y vamos a MODIFY INTEGER.

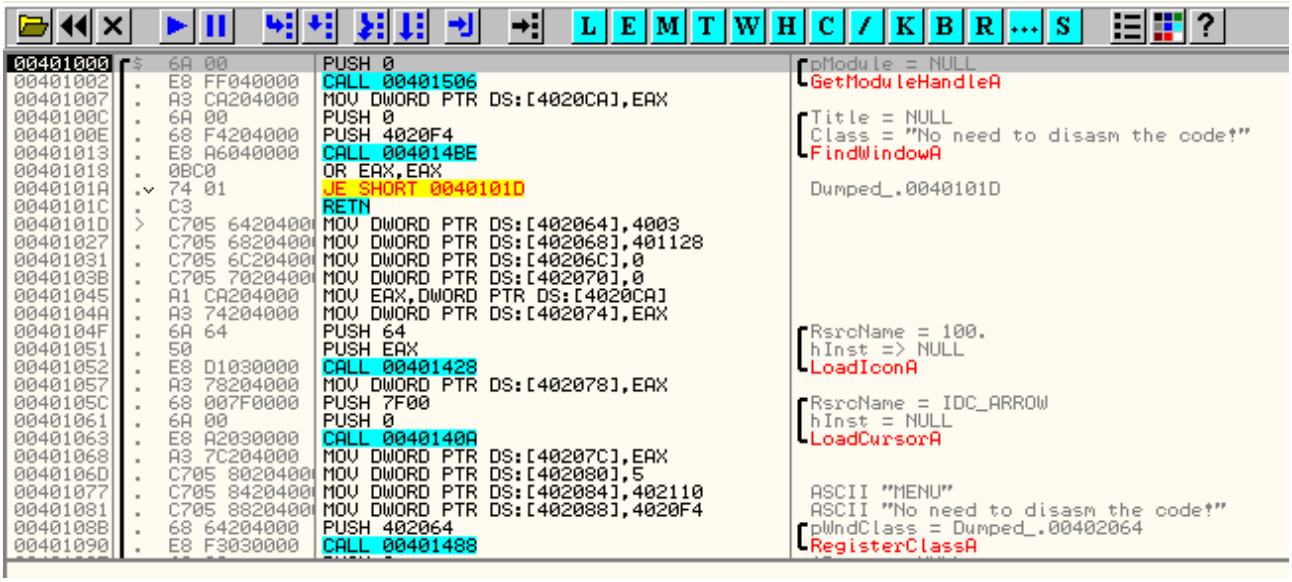




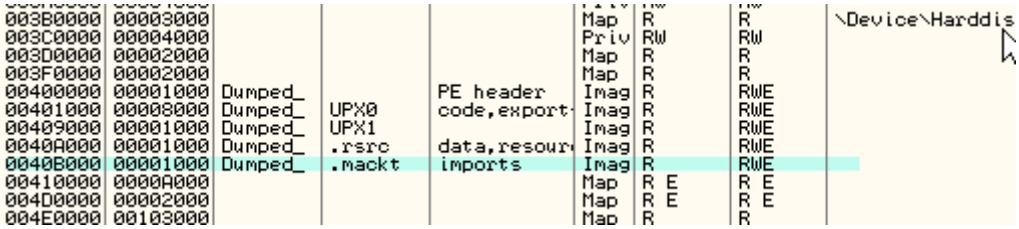
Ahora guardaremos los cambios como siempre click derecho COPY TO EXECUTABLE y en la ventana que se abre click derecho SAVE FILE.



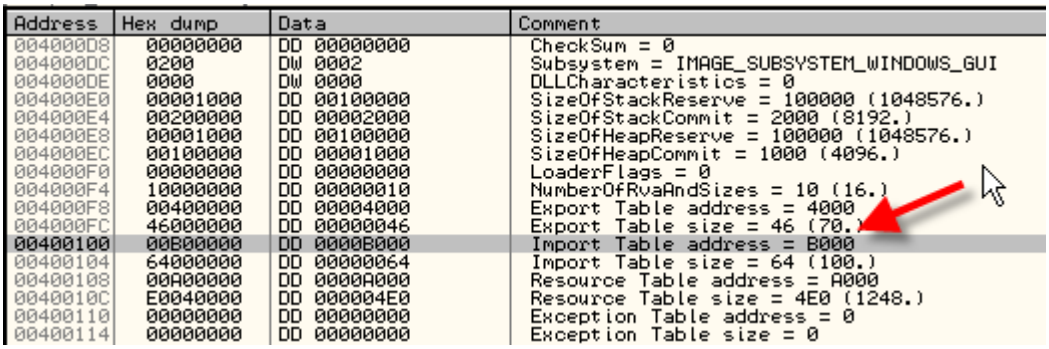




Ahora reiniciamos y vemos que no solo no sale el cartelito molesto, si no que tambien OLLYDBG analiza la seccion ya que al interpretarla como CODE, le realiza el analisis, lo cual suele ser de mucha ayuda.

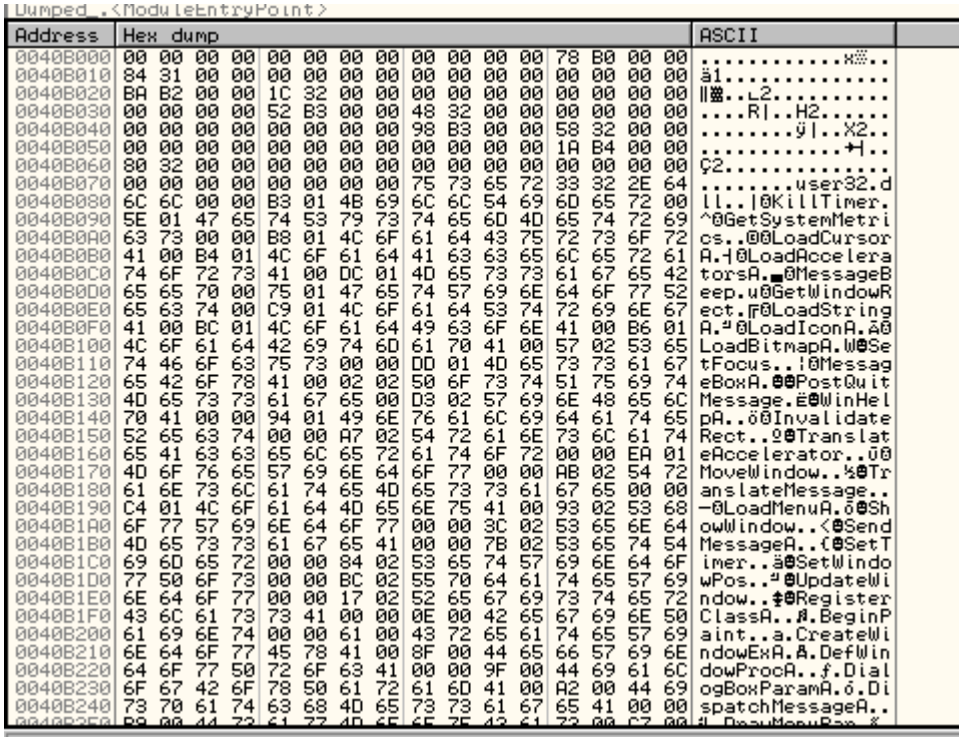


Si vamos a ver las secciones del dumpeado veamos que para repararlas el IMP REC le agrega una seccion nueva llamada mackt donde colocara la nueva IT, comprobemoslo, en el mismo header en modo SPECIAL vayamos a ver el puntero a la IT.



Address	Hex dump	Data	Comment
004000D8	00000000	DD 00000000	CheckSum = 0
004000DC	0200	DW 0002	Subsystem = IMAGE_SUBSYSTEM_WINDOWS_GUI
004000DE	0000	DW 0000	DLLCharacteristics = 0
004000E0	00001000	DD 00100000	SizeOfStackReserve = 100000 (1048576.)
004000E4	00200000	DD 00002000	SizeOfStackCommit = 2000 (8192.)
004000E8	00001000	DD 00100000	SizeOfHeapReserve = 100000 (1048576.)
004000EC	00100000	DD 00001000	SizeOfHeapCommit = 1000 (4096.)
004000F0	00000000	DD 00000000	LoaderFlags = 0
004000F4	10000000	DD 00000010	NumberOfRvaAndSizes = 10 (16.)
004000F8	00400000	DD 00004000	Export Table address = 4000
004000FC	46000000	DD 00000046	Export Table size = 46 (70.)
00400100	00B00000	DD 0000B000	Import Table address = B000
00400104	64000000	DD 00000064	Import Table size = 64 (100.)
00400108	00A00000	DD 0000A000	Resource Table address = A000
0040010C	E0040000	DD 000004E0	Resource Table size = 4E0 (1248.)
00400110	00000000	DD 00000000	Exception Table address = 0
00400114	00000000	DD 00000000	Exception Table size = 0

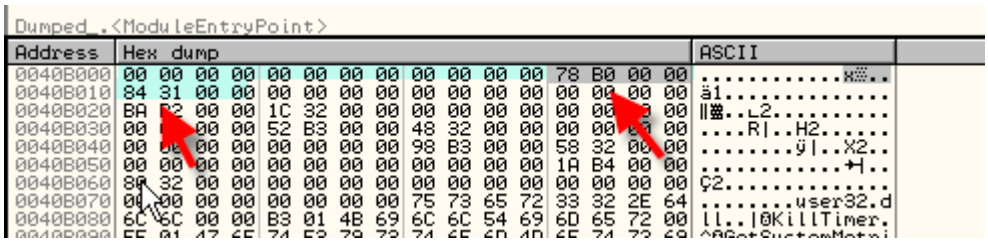
Vemos que la IT esta ahora en B000 o sea 40B000 que es la direccion de la seccion que agrego el IMP REC vayamos a ver alli quitando el modo SPECIAL.



Dumped_.<ModuleEntryPoint>

Address	Hex dump	ASCII
0040B000	00 00 00 00 00 00 00 00 00 00 00 00 78 B0 00 00	............x▒..
0040B010	84 31 00 00 00 00 00 00 00 00 00 00 00 00 00 00	ä1..............
0040B020	BA B2 00 00 1C 32 00 00 00 00 00 00 00 00 00 00	‖▒..L2..........
0040B030	00 00 00 00 52 B3 00 00 48 32 00 00 00 00 00 00	....R|..H2......
0040B040	00 00 00 00 00 00 00 00 98 B3 00 00 58 32 00 00	........ÿ|..X2..
0040B050	00 00 00 00 00 00 00 00 00 00 00 00 1A B4 00 00	............→|..
0040B060	80 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Ç2..............
0040B070	00 00 00 00 00 00 00 00 75 73 65 72 33 32 2E 64	........user32.d
0040B080	6C 6C 00 00 B3 01 4B 69 6C 6C 54 69 6D 65 72 00	ll..|0KillTimer.
0040B090	5E 01 47 65 74 53 79 73 74 65 6D 4D 65 74 72 69	^0GetSystemMetri
0040B0A0	63 73 00 00 B8 01 4C 6F 61 64 43 75 72 73 6F 72	cs..@0LoadCursor
0040B0B0	41 00 B4 01 4C 6F 61 64 41 63 63 65 6C 65 72 61	A.┤0LoadAccelera
0040B0C0	74 6F 72 73 41 00 DC 01 4D 65 73 73 61 67 65 42	torsA.▄0MessageB
0040B0D0	65 65 70 00 75 01 47 65 74 57 69 6E 64 6F 77 52	eep.u0GetWindowR
0040B0E0	65 63 74 00 C9 01 4C 6F 61 64 53 74 72 69 6E 67	ect.╔0LoadString
0040B0F0	41 00 BC 01 4C 6F 61 64 49 63 6F 6E 41 00 B6 01	A.╝0LoadIconA.Ã0
0040B100	4C 6F 61 64 42 69 74 6D 61 70 41 00 57 02 53 65	LoadBitmapA.W@Se
0040B110	74 46 6F 63 75 73 00 00 DD 01 4D 65 73 73 61 67	tFocus..¦0Messag
0040B120	65 42 6F 78 41 00 02 02 50 6F 73 74 51 75 69 74	eBoxA.@@PostQuit
0040B130	4D 65 73 73 61 67 65 00 D3 02 57 69 6E 48 65 6C	Message.Ë@WinHel
0040B140	70 41 00 00 94 01 49 6E 76 61 6C 69 64 61 74 65	pA..öÖInvalidate
0040B150	52 65 63 74 00 00 A7 02 54 72 61 6E 73 6C 61 74	Rect..º@Translat
0040B160	65 41 63 63 65 6C 65 72 61 74 6F 72 00 00 EA 01	eAccelerator..ü0
0040B170	4D 6F 76 65 57 69 6E 64 6F 77 00 00 AB 02 54 72	MoveWindow..½@Tr
0040B180	61 6E 73 6C 61 74 65 4D 65 73 73 61 67 65 00 00	anslateMessage..
0040B190	C4 01 4C 6F 61 64 4D 65 6E 75 41 00 93 02 53 68	-0LoadMenuA.ô@Sh
0040B1A0	6F 77 57 69 6E 64 6F 77 00 00 3C 02 53 65 6E 64	owWindow..<@Send
0040B1B0	4D 65 73 73 61 67 65 41 00 00 7B 02 53 65 74 54	MessageA..{@SetT
0040B1C0	69 6D 65 72 00 00 84 02 53 65 74 57 69 6E 64 6F	imer..ä@SetWindo
0040B1D0	77 50 6F 73 00 00 BC 02 55 70 64 61 74 65 57 69	wPos..╝@UpdateWi
0040B1E0	6E 64 6F 77 00 00 17 02 52 65 67 69 73 74 65 72	ndow..‡@Register
0040B1F0	43 6C 61 73 73 41 00 00 0E 00 42 65 67 69 6E 50	ClassA..♫.BeginP
0040B200	61 69 6E 74 00 00 61 00 43 72 65 61 74 65 57 69	aint..a.CreateWi
0040B210	6E 64 6F 77 45 78 41 00 8F 00 44 65 66 57 69 6E	ndowExA.Å.DefWin
0040B220	64 6F 77 50 72 6F 63 41 00 00 9F 00 44 69 61 6C	dowProcA..ƒ.Dial
0040B230	6F 67 42 6F 78 50 61 72 61 6D 41 00 A2 00 44 69	ogBoxParamA.ó.Di
0040B240	73 70 61 74 63 68 4D 65 73 73 61 67 65 41 00 00	spatchMessageA..

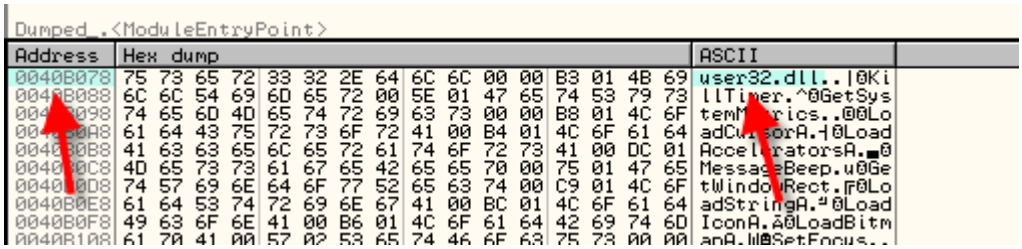
Alli vemos la IT tal cual la vimos en la parte anterior, unicada en otra direccion pero completamente funcional, si queremos podemos hallar los punteros como en la parte anterior para repasar.



Dumped_.<ModuleEntryPoint>

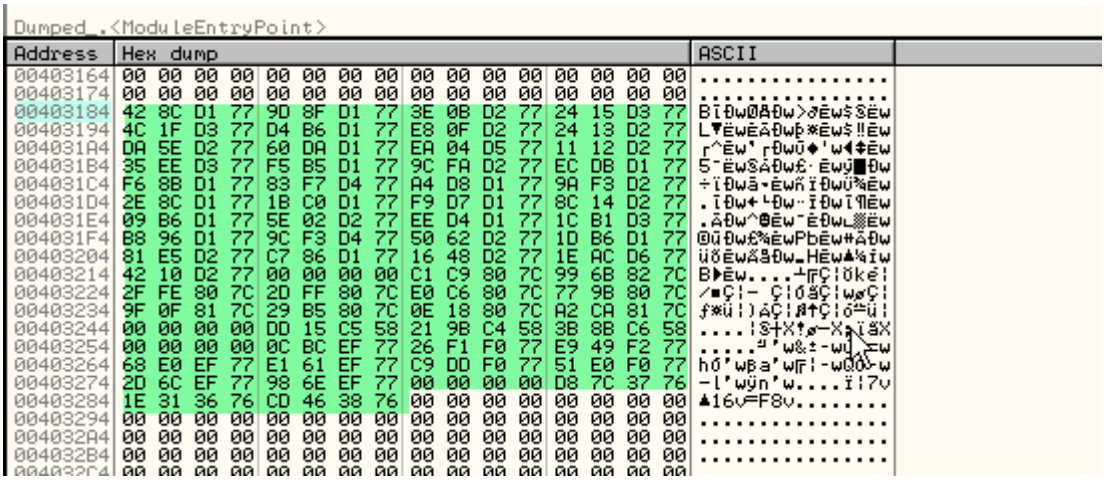
Address	Hex dump	ASCII
0040B000	00 00 00 00 00 00 00 00 00 00 00 00 78 B0 00 00	............x▒..
0040B010	84 31 00 00 00 00 00 00 00 00 00 00 00 00 00 00	ä1..............
0040B020	BA B2 00 00 1C 32 00 00 00 00 00 00 00 00 00 00	‖▒..L2..........
0040B030	00 00 00 00 52 B3 00 00 48 32 00 00 00 00 00 00	....R|..H2......
0040B040	00 00 00 00 00 00 00 00 98 B3 00 00 58 32 00 00	........ÿ|..X2..
0040B050	00 00 00 00 00 00 00 00 00 00 00 00 1A B4 00 00	............→|..
0040B060	80 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Ç2..............
0040B070	00 00 00 00 00 00 00 00 75 73 65 72 33 32 2E 64	........user32.d
0040B080	6C 6C 00 00 B3 01 4B 69 6C 6C 54 69 6D 65 72 00	ll..|0KillTimer.

Alli vemos el primer IID correspondiente a la primera dll, cuyo 4to DWORD nos apunta a su nombre como vimos, en este caso el nombre de la dll estara en B078 o sea 40B078.

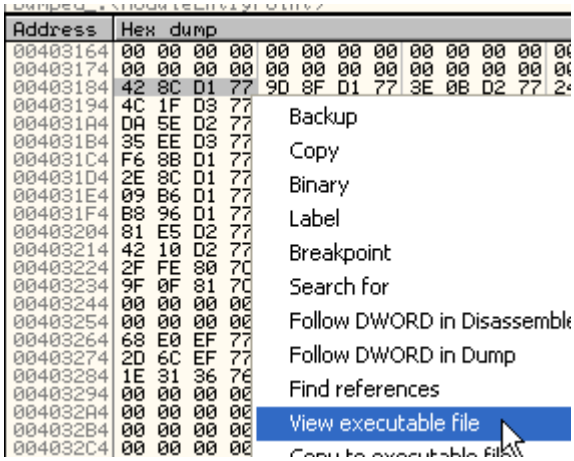


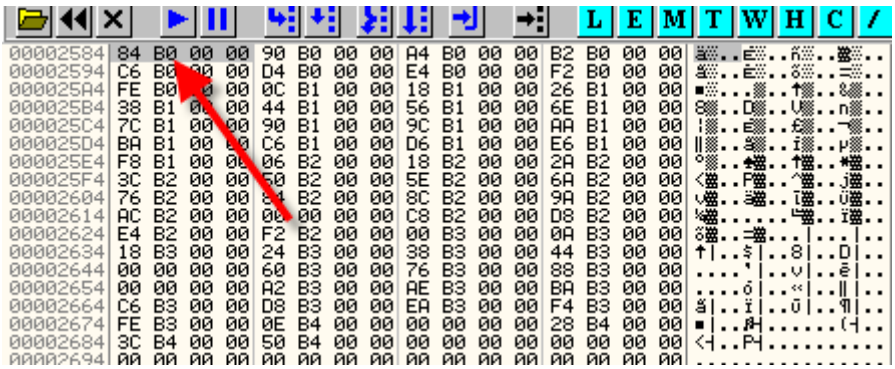
Es user32.dll.

Y el 5to puntero como vimos en la parte anterior apuntara a la primera entrada de la IAT, en este caso el 5to puntero es 3184 por lo tanto la primera entrada de la IAT estara en 403184 como habiamos visto que era el inicio de la IAT al reconstruirla.

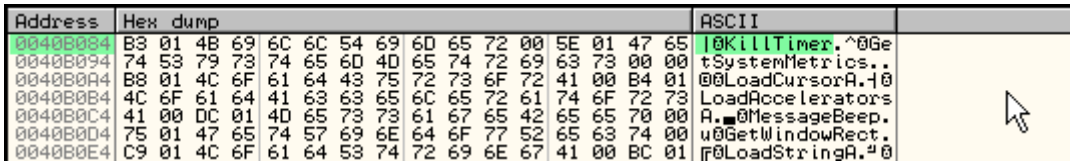


Y por supuesto la primera entrada de la IAT ahora tiene el valor de la api pero si vemos en el ejecutable, antes de ser sobreescrita por el sistema con la direccion de la api en mi maquina, debe tener el puntero al nombre de la api, que llenara dicha entrada si vemos.





En esa entrada de la IAT en el ejecutable, existe el valor B084 que corresponde a 40B084, donde deberia estar el nombre de la API que llenara esta entrada.



La cual es KillTimer.

Vemos que el IMP REC realizo un trabajo estupendo, detecto cada api, construyo la IT nuevamente arreglando todos los punteros, y reconstruyo la lista de nombres de cada api para que al arrancar el sistema sepa que api debe llenar cada entrada de la IAT una verdadera maravilla.

Bueno esta ha sido nuestra primera reconstrucion de una IAT la mas sencilla que puede existir, pero es la base de todo y es importante que tanto la parte anterior como esta, la tengan bien claro, porque a medida que sigamos desempacando nos encontraremos con IATs completemente destrozadas, apis redireccionadas y casos dificiles, para los cuales es imprescindible haber entendido bien comno trabaja todo.
Ademas este packer no tiene antidumps que es la parte que aun no vimos ya que no fue necesario, pero a medida que vayamos avanzando en la complejidad de los packers ya encontraremos casos con ANTIDUMP.
De cualquier manera iremos incrementando la dificultad suavemente para que se les vaya fijando los conceptos asi que no se asusten, vamos de a poquito jejejejeje

Hasta la parte 35
Ricardo Narvaja
05/03/06