

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 34

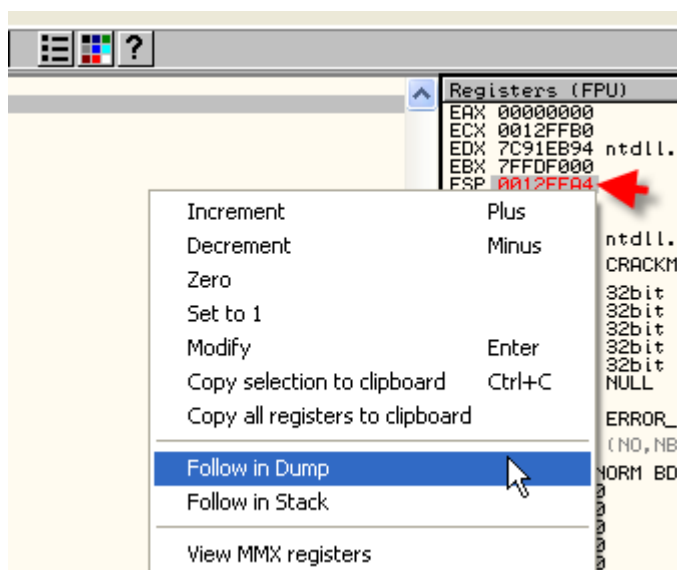
En la parte 33 vimos como funciona el sistema de la IT y IAT, yo se y los que saben reparar una IAT, pueden pensar que no es necesario saber como funciona, ya que hay tools que reparan una iat casi automaticamente, pero yo les aseguro que es bueno saber como funciona y que ocurre en cada momento, porque 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 correccion 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.

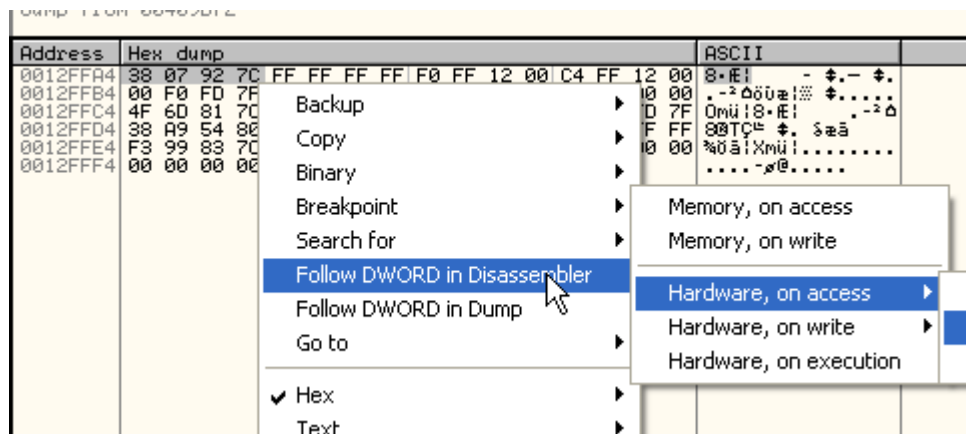


Aplicare el metodo del PUSHAD para llegar al OEP apreto f7 para pasar el PUSHAD.



Ahora marco ESP-FOLLOW IN DUMP.

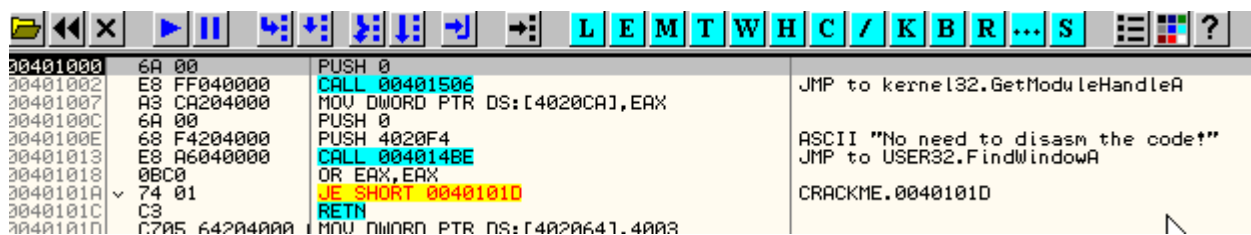
Y en el dump vere los registros que guardo, y marcare los primeros 4 bytes, y pondre un HARDWARE BPX ON ACCESS.



Y doy RUN, parara en el salto al OEP.



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



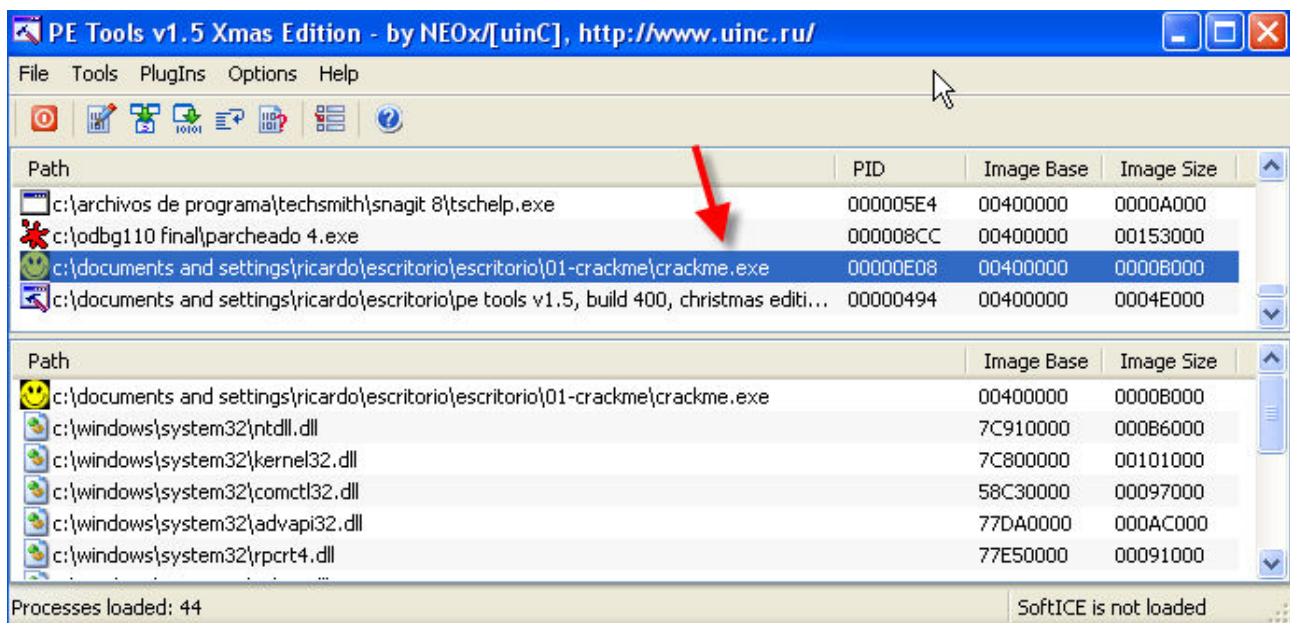
Bueno ya llegamos, al punto donde el programa esta desempacado en memoria ahora lo dumpearemos.

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:

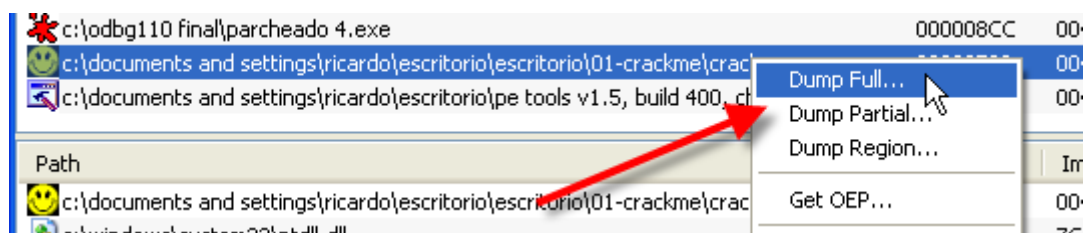
http://www.uinc.ru/files/neox/PE_Tools.shtml

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

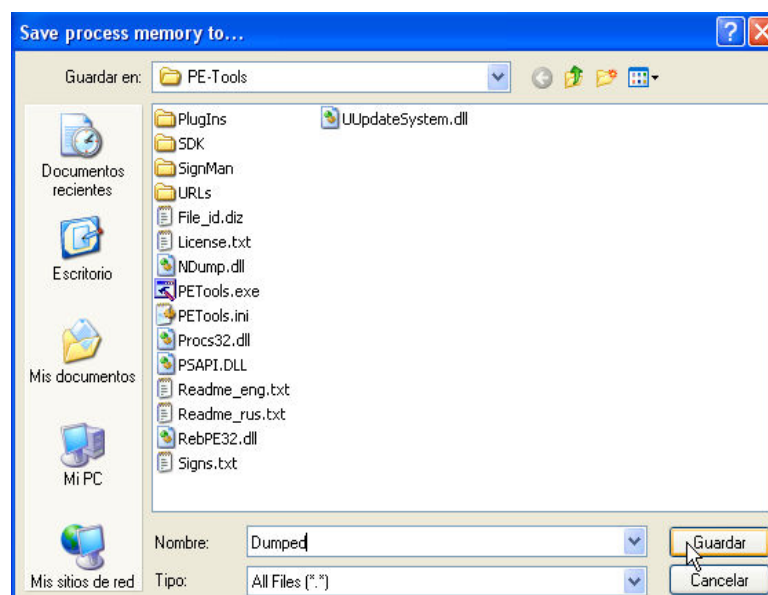
Abramos el PE-TOOLS sin cerrar el OLLYDBG que esta detenido en el OEP.



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.

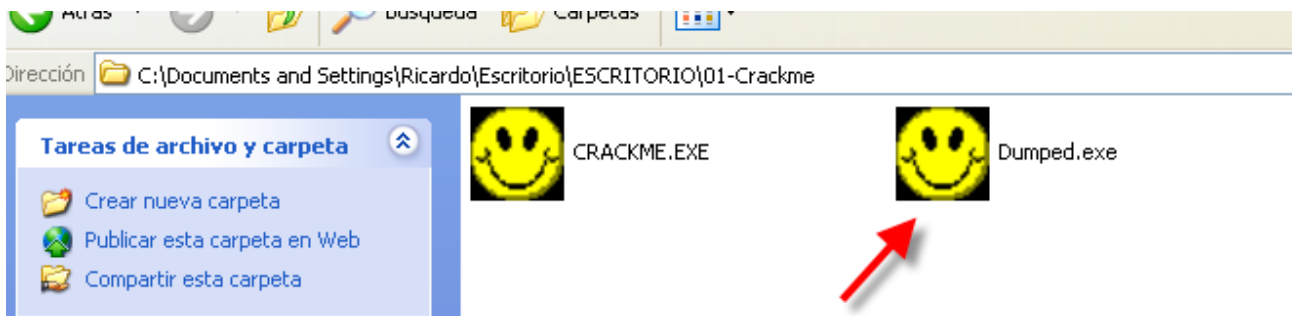


Haciendo click derecho-DUMP FULL

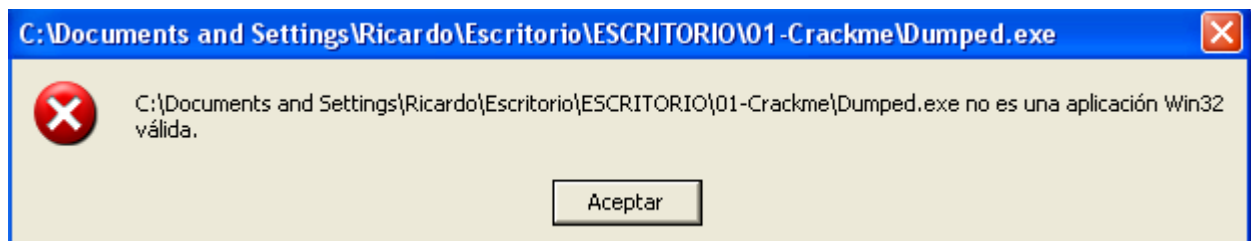




Bueno ya esta dumpeado ahora vamos a la reparacion de la IAT cerramos el PE TOOLS vimos que el DUMPED lo guardo en la carpeta del PE TOOLS, asi que lo buscamos y lo copiamos a la carpeta donde esta el mismo Crackme de Cruehead empacado con UPX.



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..

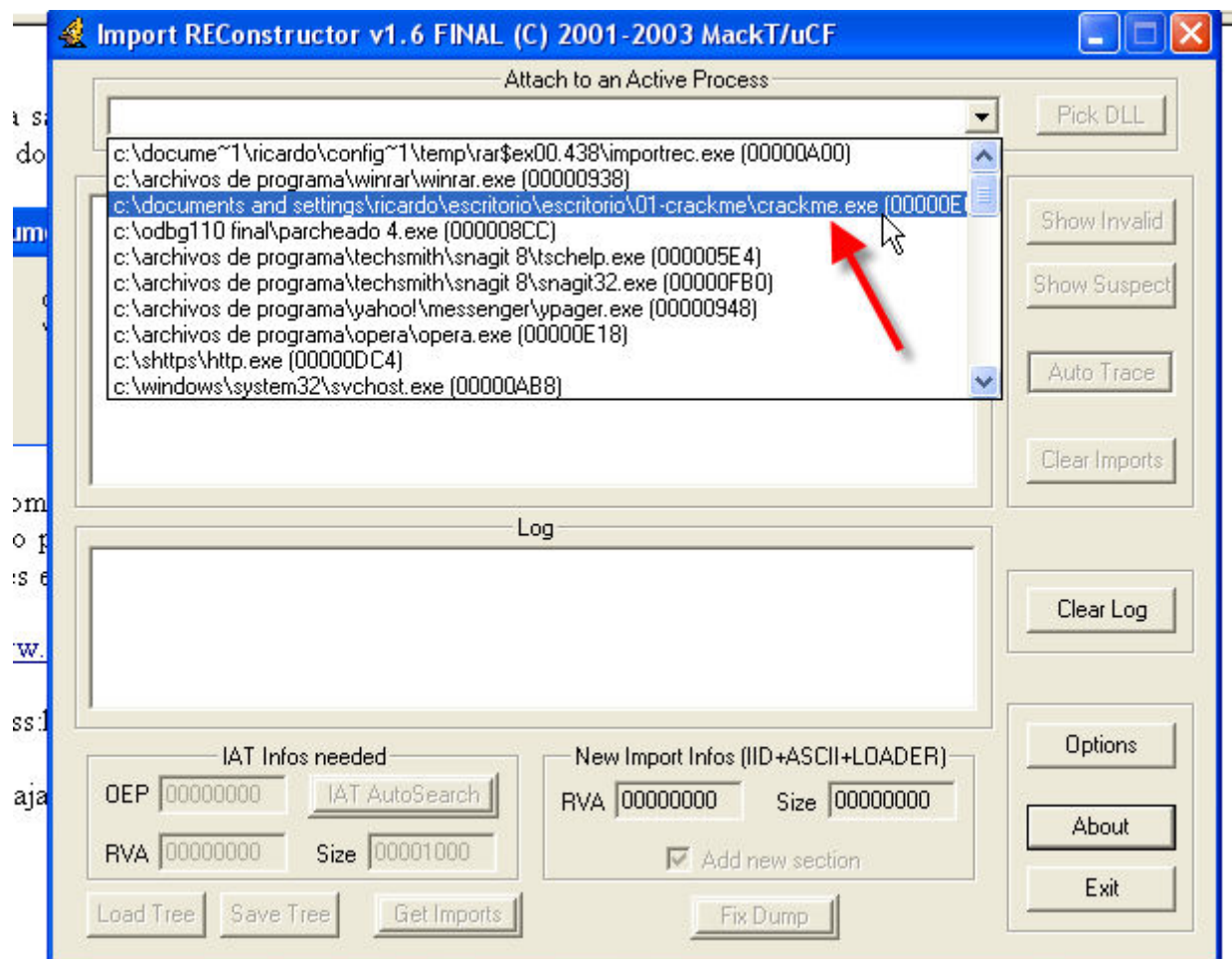


Bueno como ven hay que reparar la IAT, aunque corriera en nuestra maquina, debemos reparar la IAT para que funcione en cualquier maquina y no solo en la nuestra, para ello usaremos el IMPORT RECONSTRUCTOR, por supuesto no cerramos el OLLYDBG con el CC empacado con UPX detenido en el OEP, pues el IMP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

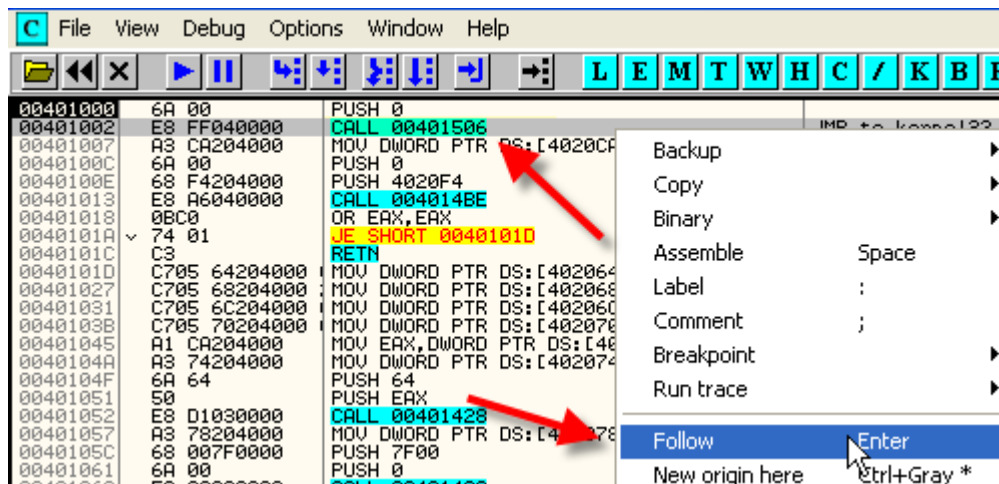
Por supuesto sabemos que las llamadas a las apis generalmente son realizadas con JMPs indirectos o CALLs indirectos del tipo.

JMP [xxxxxxx] o CALL [xxxxxxx]

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 empackado, para lo cual miremos en el OLLYDBG con el CC empackado, parado en el OEP.

00401000	6A 00	PUSH 0	
00401002	E8 FF040000	CALL 00401506	JMP to kernel32.GetModuleHandleA
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA],EAX	
0040100C	6A 00	PUSH 0	
0040100E	68 F4204000	PUSH 4020F4	
00401013	E8 A6040000	CALL 004014BE	
00401018	0BC0	OR EAX,EAX	
0040101A	74 01	JE SHORT 0040101D	
0040101C	C3	RETN	
0040101D	C705 64204000	MOV DWORD PTR DS:[402064],4003	
00401027	C705 68204000	MOV DWORD PTR DS:[402068],401128	
00401031	C705 6C204000	MOV DWORD PTR DS:[40206C],0	
0040103B	C705 70204000	MOV DWORD PTR DS:[402070],0	

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



00401506	- FF25 38324000	JMP DWORD PTR DS:[403238]	kernel32.GetModuleHandleA
0040150C	- FF25 3C324000	JMP DWORD PTR DS:[40323C]	kernel32.ReadFile
00401512	- FF25 40324000	JMP DWORD PTR DS:[403240]	kernel32.ExitProcess
00401518	- FF25 48324000	JMP DWORD PTR DS:[403248]	COMCTL32.InitCommonControls
0040151E	- FF25 4C324000	JMP DWORD PTR DS:[40324C]	COMCTL32.CreateToolBarEx
00401524	- FF25 50324000	JMP DWORD PTR DS:[403250]	COMCTL32.CreateToolBar
0040152A	- FF25 54324000	JMP DWORD PTR DS:[403254]	GDI32.TextOutA
00401530	- FF25 58324000	JMP DWORD PTR DS:[403258]	GDI32.StartPage
00401536	- FF25 5C324000	JMP DWORD PTR DS:[40325C]	GDI32.StartDocA
0040153C	- FF25 60324000	JMP DWORD PTR DS:[403260]	GDI32.GetTextMetricsA
00401542	- FF25 64324000	JMP DWORD PTR DS:[403264]	GDI32.GetStockObject
00401548	- FF25 68324000	JMP DWORD PTR DS:[403268]	GDI32.EndPage
0040154E	- FF25 6C324000	JMP DWORD PTR DS:[40326C]	GDI32.EndDoc
00401554	- FF25 70324000	JMP DWORD PTR DS:[403270]	GDI32.DeleteObject
0040155A	- FF25 74324000	JMP DWORD PTR DS:[403274]	GDI32.DeleteDC
00401560	- FF25 78324000	JMP DWORD PTR DS:[403278]	COMDLG32.GetSaveFileNameA
00401566	- FF25 7C324000	JMP DWORD PTR DS:[40327C]	COMDLG32.GetOpenFileNameA
0040156C	- FF25 80324000	JMP DWORD PTR DS:[403280]	COMDLG32.PrintDlgA
00401572	0000	ADD BYTE PTR DS:[EAX],AL	

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]

004014FA	- FF25 30324000	JMP DWORD PTR DS:[403230]	kernel32.CloseHandle
00401500	- FF25 34324000	JMP DWORD PTR DS:[403234]	kernel32.WriteFile
00401506	- FF25 38324000	JMP DWORD PTR DS:[403238]	kernel32.GetModuleHandleA
0040150C	- FF25 3C324000	JMP DWORD PTR DS:[40323C]	kernel32.ReadFile
00401512	- FF25 40324000	JMP DWORD PTR DS:[403240]	kernel32.ExitProcess
00401518	- FF25 48324000	JMP DWORD PTR DS:[403248]	COMCTL32.InitCommonControls
0040151E	- FF25 4C324000	JMP DWORD PTR DS:[40324C]	COMCTL32.CreateToolBarEx
00401524	- FF25 50324000	JMP DWORD PTR DS:[403250]	COMCTL32.CreateToolBar
0040152A	- FF25 58324000	JMP DWORD PTR DS:[403258]	GDI32.TextOutA
00401530	- FF25 5C324000	JMP DWORD PTR DS:[40325C]	GDI32.StartPage
00401536	- FF25 60324000	JMP DWORD PTR DS:[403260]	GDI32.StartDoc

DS:[00403238]=7C80B529 (kernel32.GetModuleHandleA)

Address	Hex dump	ASCII
00403238	29 B5 80 7C 0E 18 80 7C	!AÇ!#†Ç!
00403240	DD 15 C5 58 21 9B C4 58	!S+X!-X;!S%....
00403248	0C BC EF 77 26 F1 F0 77	.# 'w&:-w
00403250	E9 49 F2 77 68 E0 EF 77	!I=wh0'w
00403258	E1 61 EF 77 C9 DD F0 77	Ba'wfl'-w
00403260	51 E0 F0 77 2D 6C EF 77	Q0-w-l'w
00403268	98 6E EF 77 00 00 00 00	ÿn'w....
00403270	D8 7C 37 76 1E 31 36 76	i!7v▲16v
00403278	CD 46 38 76 00 00 00 00	=F8v....
00403280	00 00 00 00 00 00 00 00
00403288	00 00 00 00 00 00 00 00
00403290	00 00 00 00 00 00 00 00

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 revés es 29 B5 80 7c, cambiare a la vista de dos columnas para que se aprecie mas.

Address	Hex dump	ASCII
00403238	29 B5 80 7C 0E 18 80 7C	!AÇ!#†Ç!
00403240	A2 CA 81 7C 00 00 00 00	ó"ü!....
00403248	DD 15 C5 58 21 9B C4 58	!S+X!-X;!S%....
00403250	3B 8B C6 58 00 00 00 00	!S%....
00403258	0C BC EF 77 26 F1 F0 77	.# 'w&:-w
00403260	E9 49 F2 77 68 E0 EF 77	!I=wh0'w
00403268	E1 61 EF 77 C9 DD F0 77	Ba'wfl'-w
00403270	51 E0 F0 77 2D 6C EF 77	Q0-w-l'w
00403278	98 6E EF 77 00 00 00 00	ÿn'w....
00403280	D8 7C 37 76 1E 31 36 76	i!7v▲16v
00403288	CD 46 38 76 00 00 00 00	=F8v....
00403290	00 00 00 00 00 00 00 00
00403298	00 00 00 00 00 00 00 00

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.

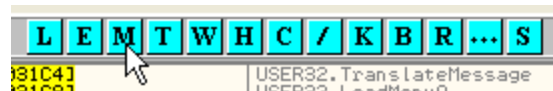
Address	Hex dump	ASCII
00403238	29 B5 80 7C 0E 18 80 7C	!AÇ!#†Ç!
00403240	A2 CA 81 7C 00 00 00 00	ó"ü!....
00403248	DD 15 C5 58 21 9B C4 58	!S+X!-X;!S%....
00403250	3B 8B C6 58 00 00 00 00	!S%....
00403258	0C BC EF 77 26 F1 F0 77	.# 'w&:-w
00403260	E9 49 F2 77 68 E0 EF 77	!I=wh0'w
00403268	E1 61 EF 77 C9 DD F0 77	Ba'wfl'-w
00403270	51 E0 F0 77 2D 6C EF 77	Q0-w-l'w
00403278	98 6E EF 77 00 00 00 00	ÿn'w....
00403280	D8 7C 37 76 1E 31 36 76	i!7v▲16v
00403288	CD 46 38 76 00 00 00 00	=F8v....
00403290	00 00 00 00 00 00 00 00
00403298	00 00 00 00 00 00 00 00
004032A0	00 00 00 00 00 00 00 00

Algunos packers mas sofisticados sobrescriben 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.

77F41000	0006C000	SHLWAPI	.text	code, import	Image	R	RWE	
77FAD000	00001000	SHLWAPI	.data	data	Image	R	RWE	
77FAE000	00002000	SHLWAPI	.rsrc	resources	Image	R	RWE	
77FB0000	00006000	SHLWAPI	.reloc	relocations	Image	R	RWE	
7C800000	00001000	kernel32		PE header	Image	R	RWE	
7C801000	00082000	kernel32	.text	code, import	Image	R	RWE	
7C883000	00005000	kernel32	.data	data	Image	R	RWE	
7C888000	00073000	kernel32	.rsrc	resources	Image	R	RWE	
7C8FB000	00006000	kernel32	.reloc	relocations	Image	R	RWE	
7C910000	00001000	ntdll		PE header	Image	R	RWE	
7C911000	0007B000	ntdll	.text	code, export	Image	R	RWE	

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.

Address	Hex dump	ASCII
00403208	C7 86 D1 77 16 48 D2 77	AS0w..HEw
00403210	1E AC D6 77 42 10 D2 77	Δ%iwBHEw
00403218	00 00 00 00 C1 C9 80 7C	...-IFC
00403220	99 6B 82 7C 2F FE 80 7C	0ke /■C
00403228	2D FF 80 7C E0 C6 80 7C	- C:0SC
00403230	77 9B 80 7C 9F 0F 81 7C	wSC:f*ü
00403238	29 B5 80 7C 0E 18 80 7C	LAQ:RtC
00403240	A2 CA 81 7C 00 00 00 00	öü!....
00403248	DD 15 C5 58 21 9B C4 58	!st%*s-X
00403250	3B 8B C6 58 00 00 00 00	;IS%....
00403258	0C BC EF 77 26 F1 F0 77	."w&:-w
00403260	E9 49 F2 77 68 E0 EF 77	üI=wh0'w
00403268	E1 61 EF 77 C9 DD F0 77	Ba'wF!-w
00403270	51 E0 F0 77 2D 6C EF 77	Q0'-w-l'w
00403278	98 6E EF 77 00 00 00 00	ÿn'w....
00403280	D8 7C 37 76 1E 31 36 76	i!7v▲16v
00403288	CD 46 38 76 00 00 00 00	=F8v....
00403290	00 00 00 00 00 00 00 00
00403298	00 00 00 00 00 00 00 00

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 77Dxxxxxx, miremos en M, a que dll corresponden.

77C34000	00001000	msvcrt	.rsrc	resources	Image	R	RWE	
77C35000	00003000	msvcrt	.reloc	relocations	Image	R	RWE	
77D10000	00001000	USER32		PE header	Image	R	RWE	
77D11000	0005F000	USER32	.text	code, import	Image	R	RWE	
77D70000	00002000	USER32	.data	data	Image	R	RWE	
77D71000	00002B000	USER32	.rsrc	resources	Image	R	RWE	
77D90000	00003000	USER32	.reloc	relocations	Image	R	RWE	
77DA0000	00001000	ADVAPI32		PE header	Image	R	RWE	
77DA1000	00075000	ADVAPI32	.text	code, import	Image	R	RWE	

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.


```

00403168 00 00 00 00 00 00 00 00 .....
00403170 00 00 00 00 00 00 00 00 .....
00403178 00 00 00 00 00 00 00 00 .....
00403180 00 00 00 00 42 8C D1 77 ....Bïðw
00403188 9D 8F D1 77 3E 0B D2 77 >ðËw$Ëw
00403190 24 15 D3 77 4C 1F D3 77 $ËwLËw
00403198 04 B6 D1 77 E8 0F D2 77 Ëðwþ*Ëw
004031A0 24 13 D2 77 DA 5E D2 77 $!Ëw^Ëw
004031A8 60 DA D1 77 EA 04 D5 77 ^ðwü♦Ëw
004031B0 11 12 D2 77 35 EE D3 77 †Ëw5Ëw
004031B8 F5 B5 D1 77 9C FA D2 77 $ðwËËw
004031C0 EC DB D1 77 F6 8B D1 77 üðw÷Ëw
004031C8 83 F7 D4 77 A4 D8 D1 77 ãËwñËw
004031D0 9A F3 D2 77 2E 8C D1 77 üËw.Ëw
004031D8 1B C0 D1 77 F9 D7 D1 77 +Ëw~Ëw
004031E0 8C 14 D2 77 09 B6 D1 77 ïËwîËw
004031E8 5E 02 D2 77 EE D4 D1 77 ^Ëw^Ëw
004031F0 1C B1 D3 77 B8 96 D1 77 LËwüðw
004031F8 9C F3 D4 77 50 62 D2 77 ËËwPbËw
00403200 10 B6 D1 77 81 E5 D2 77 #ðwüðËw
00403208 C7 86 D1 77 16 48 D2 77 ãðw_HËw
00403210 1E AC D6 77 42 10 D2 77 ▲ËwBËw
00403218 00 00 00 00 C1 C9 80 7C ....+FC!
00403220 99 6B 82 7C 2F FE 80 7C öké! /#C!
00403228 2D FF 80 7C E0 C6 80 7C - C!öðC!

```

Pues allí 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.

Address	Hex dump	ASCII
00403174	00 00 00 00 00 00 00 00
0040317C	00 00 00 00 00 00 00 00
00403184	42 8C D1 77 9D 8F D1 77	Bïðwððw
0040318C	3E 0B D2 77 24 15 D3 77	>ðËw\$Ëw
00403194	4C 1F D3 77 04 B6 D1 77	LËwËðw
0040319C	E8 0F D2 77 24 13 D2 77	þ*Ëw\$!Ëw
004031A4	DA 5E D2 77 60 DA D1 77	^ðw' rðw
004031AC	EA 04 D5 77 11 12 D2 77	ü♦Ëw†Ëw
004031B4	35 EE D3 77 F5 B5 D1 77	5Ëw\$ðw
004031BC	9C FA D2 77 EC DB D1 77	ËËwüðw
004031C4	F6 8B D1 77 83 F7 D4 77	÷ËwãËw
004031CC	A4 D8 D1 77 9A F3 D2 77	ñËwüËw
004031D4	2E 8C D1 77 1B C0 D1 77	.Ëw+Ëw
004031DC	F9 D7 D1 77 8C 14 D2 77	~ËwîËw
004031E4	09 B6 D1 77 5E 02 D2 77	.ðw^Ëw
004031EC	EE D4 D1 77 1C B1 D3 77	ËðwLËw
004031F4	B8 96 D1 77 9C F3 D4 77	üðwËËw
004031FC	50 62 D2 77 10 B6 D1 77	PbËw#ðw
00403204	81 E5 D2 77 C7 86 D1 77	üðËwãðw
0040320C	16 48 D2 77 1E AC D6 77	_HËw▲Ëw
00403214	42 10 D2 77 00 00 00 00	Bðw....
0040321C	C1 C9 80 7C 99 6B 82 7C	+FC!öké!
00403224	2F FE 80 7C 2D FF 80 7C	/#C!- C!
0040322C	E0 C6 80 7C 77 9B 80 7C	öðC!wðC!
00403234	9F 0F 81 7C 29 B5 80 7C	f*ü!) ÆC!
0040323C	0E 18 80 7C A2 CA 81 7C	þ†C! öü!
00403244	00 00 00 00 DD 15 C5 58!S+X
0040324C	21 9B C4 58 3B 8B C6 58	!ð-X; !ðX
00403254	00 00 00 00 0C BC EF 77dËw

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 allí 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.

0040323C	0E 18 80 7C	H2 CH 81 7C	RTVto=ui
00403240	00 00 00 00	DD 15 C5 58	...iS+X
00403244	21 9B C4 58	3B 8B C6 58	?-X;13X
00403254	00 00 00 00	0C BC EF 77d'w
0040325C	26 F1 F0 77	E9 49 F2 77	%t-wUI=w
00403264	68 E0 EF 77	E1 61 EF 77	h0'wBa'w
0040326C	C9 0D F0 77	51 E0 F0 77	f:-w00-w
00403274	2D 6C EF 77	98 6E EF 77	-l'wyn'w
0040327C	00 00 00 00	08 7C 37 76	...i!7v
00403284	1E 31 36 76	CD 46 38 76	16v=F8v
0040328C	00 00 00 00	00 00 00 00
00403294	00 00 00 00	00 00 00 00
0040329C	00 00 00 00	00 00 00 00
004032A4	00 00 00 00	00 00 00 00

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

58CA4000	0001F000	COMCTL32	.rsrc	resources	Image	R	RWE
58CC3000	00004000	COMCTL32	.reloc	relocations	Image	R	RWE
76360000	00001000	COMDLG32		PE header	Image	R	RWE
76361000	00030000	COMDLG32	.text	code,import	Image	R	RWE
76391000	00004000	COMDLG32	.data	data	Image	R	RWE
76395000	00012000	COMDLG32	.rsrc	resources	Image	R	RWE
763A7000	00003000	COMDLG32	.reloc	relocations	Image	R	RWE
773A0000	00001000	comctl_1		PE header	Image	R	RWE
773A1000	00090000	comctl_1	.text	code,import	Image	R	RWE

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

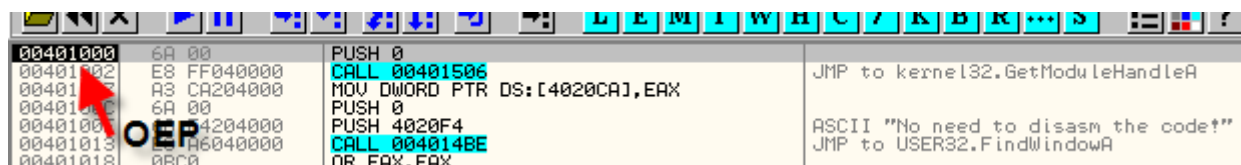
el IMPORT RECONSTRUCTOR nos pide tres datos :

- 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.

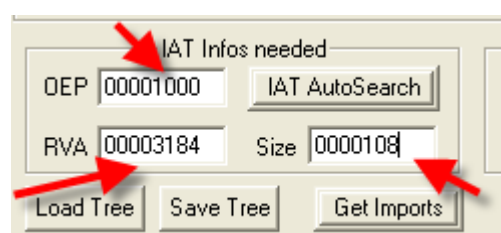
LARGO=40328c-403184=108

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**

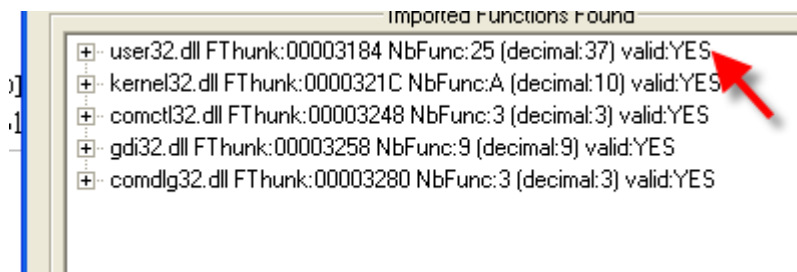


Estos datos los ingresaremos en el IMP REC.



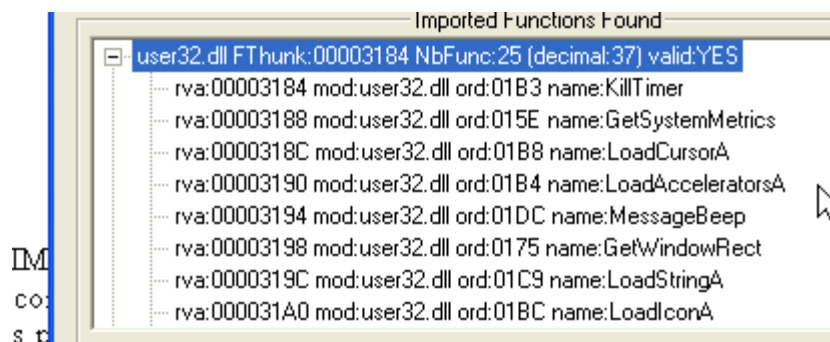
Allí vemos los datos que pusimos en el IMP REC, en OEP pusimos el 1000 ya que a 401000 le restamos la imagebase, en RVA el inicio de tabla restandole la imagebase también y en Size el largo de la IAT.

Ahora apretamos GET IMPORTS

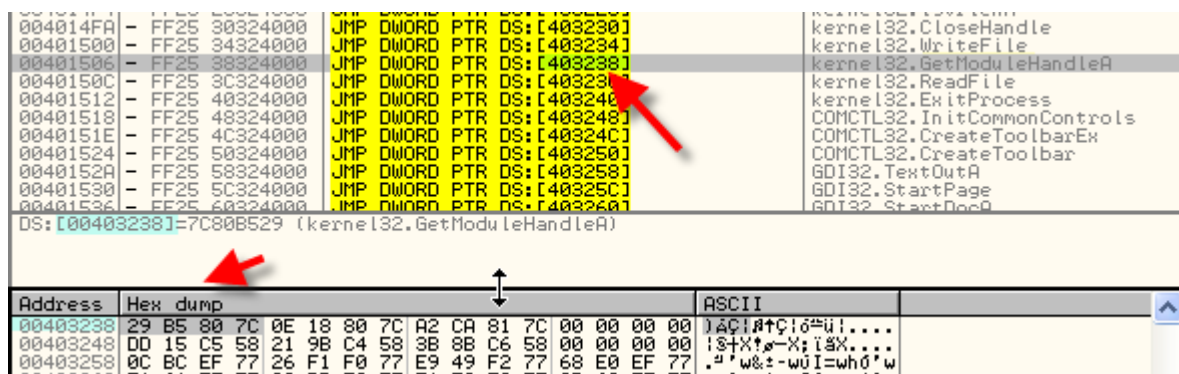


Vemos que el IMP REC lo que hace es hallar que apis pertenece a cada entrada de la IAT, y además de hallarla, te comunica si es válida o sea si está correcta, marcando YES, en el caso de entradas redireccionadas por ciertos packers que no vayan directamente a una API te mostrará NO y en ese caso habrá 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 está 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 +.

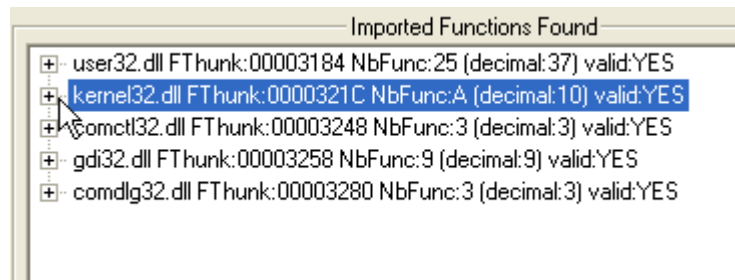


Tenemos cada entrada a que api pertenece y si tenemos ganas podemos ubicar la primera que vimos en la IAT que era la entrada correspondiente a GetModuleHandleA.

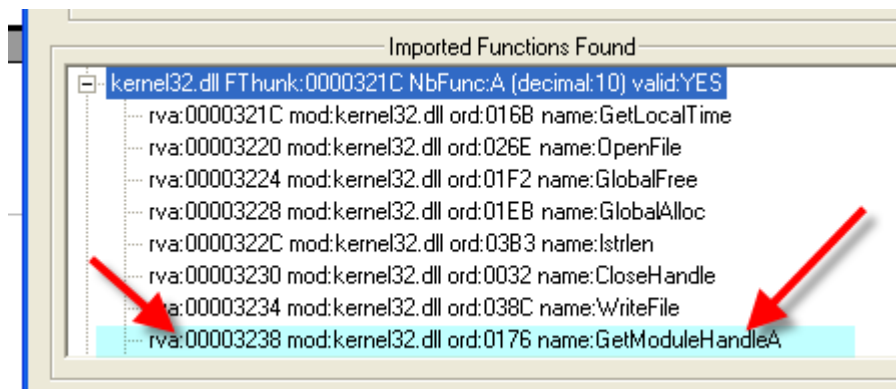


Si recuerdan esa fue la primera entrada que miramos en la IAT la correspondiente a 403238, por supuesto en IMP REC siempre hay que restar la imagebase por lo cual buscaremos 3238 además sabemos que pertenece a Kernel32.dll como vemos en la imagen superior al lado del nombre de la

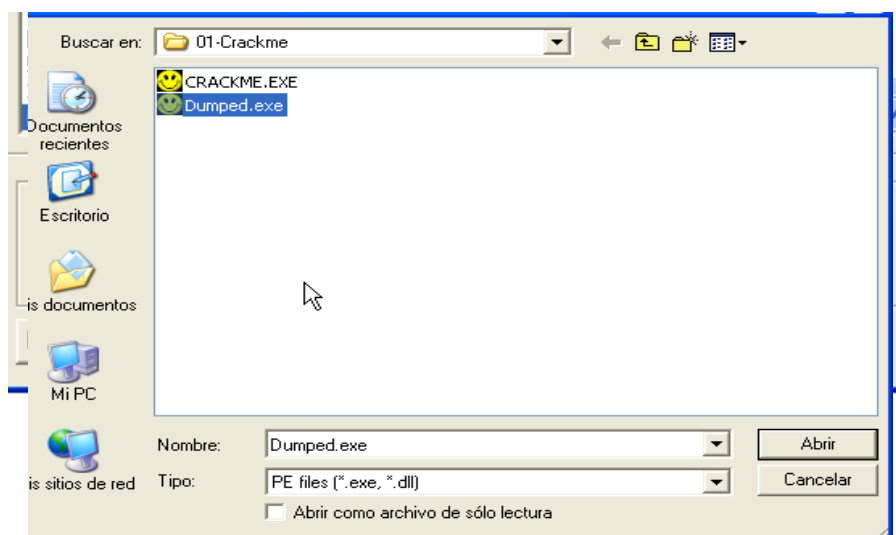
api.



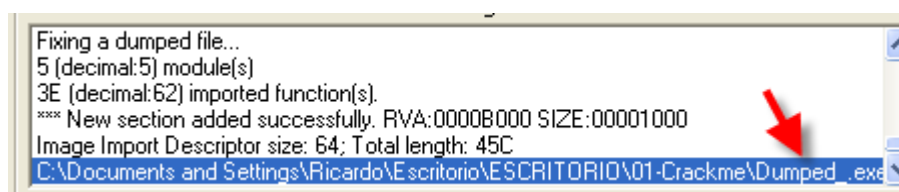
Por lo tanto debemos abrir la kernel32.dll y para ello hacemos click en el + que esta a la izquierda.



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



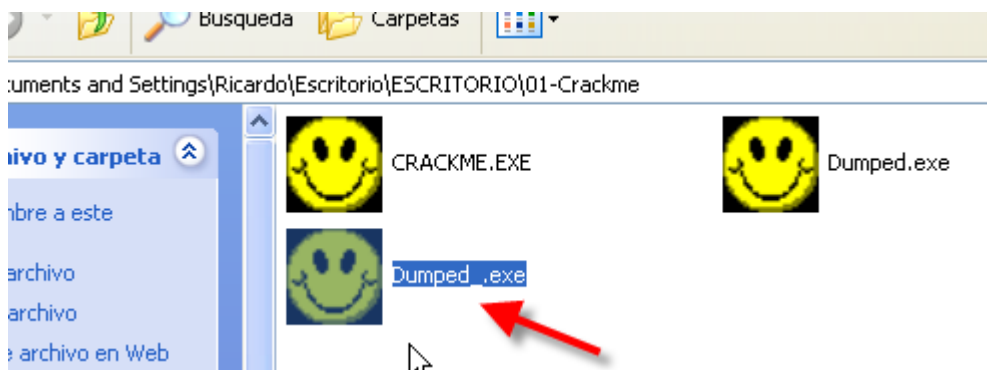
Alli buscamos el dumpeado y lo abrimos



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.

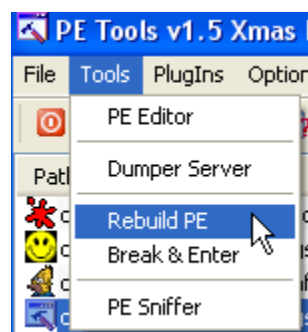
Veamos alli en la carpeta donde esta.



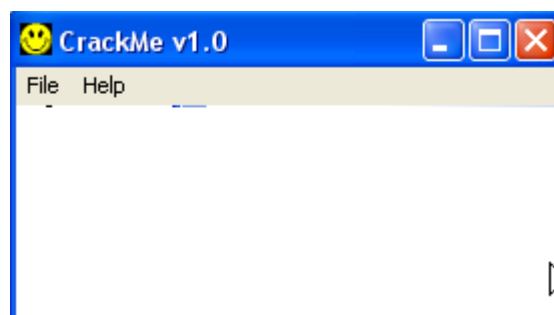
Alli esta podemos intentar ejecutarlo a ver si quedo bien.



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



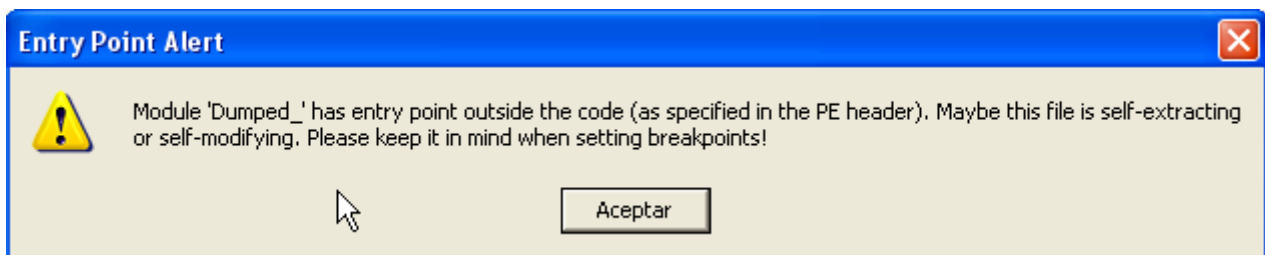
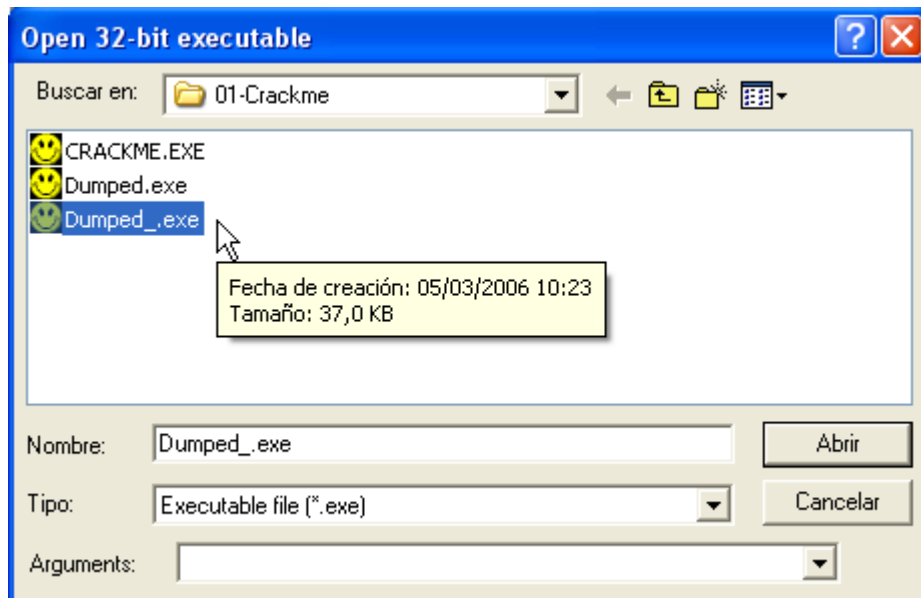
Y vamos a REBUILD PE y buscamos el DUMPED_.EXE y lo repara perfectamente ahora lo ejecutamos yyyy...



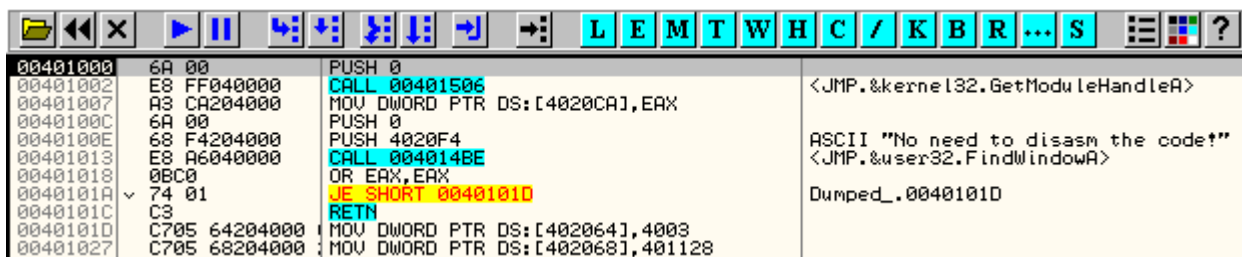
Funcionaaaaaaaaa y ademas funcionara en cualquier maquina, porque hemos reparado la IAT, de

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.

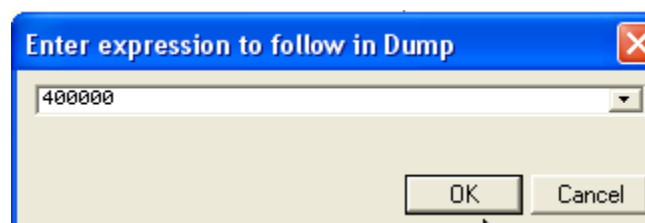
Si abrimos el DUMPED_.exe en OLLYDBG.



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.0...
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 00C...
00400040	BA 10 00 0E 1F B4 09 CD	>.A?+. =
00400048	21 B8 01 4C CD 21 90 90	?@@L=?EE
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 00 0A 24 37	in32..\$7
00400078	00 00 00 00 00 00 00 00
00400080	50 45 00 00 4C 01 04 00	PE..L0♦.
00400088	29 24 09 00 00 00 00 00	!<J

Cambiamos a modo SPECIAL-PE HEADER

Address	Hex dump	ASCII
00400000	4D 5A 50 00 02 00 00 00	MZP.0...
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 00C...
00400040	BA 10 00 0E 1F B4 09 CD	>.A?+. =
00400048	21 B8 01 4C CD 21 90 90	?@@L=?EE
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 00 0A 24 37	in32..\$7
00400078	00 00 00 00 00 00 00 00
00400080	50 45 00 00 4C 01 04 00	PE..L0♦.
00400088	29 24 09 00 00 00 00 00	!<J
00400090	00 00 00 00 00 00 00 00
00400098	00 00 00 00 00 00 00 00
004000A0	00 00 00 00 00 00 00 00
004000A8	00 00 00 00 00 00 00 00
004000B0	00 00 00 00 00 00 00 00
004000B8	00 00 00 00 00 00 00 00
004000C0	01 00 00 00 00 00 00 00
004000C8	03 00 00 00 00 00 00 00
004000D0	00 00 00 00 00 00 00 00

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 00	DB 00	
0040001D	00 00	DB 00	

Alli esta bajamos

Address	Hex dump	Data	Comment
00400032	00	DB 00	
00400033	00	DB 00	
00400034	00	DB 00	
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	80000000	DD 00000000	Offset to PE signature
00400040	BA	DB BA	
00400041	10	DB 10	
00400042	00	DB 00	
00400043	0E	DB 0E	
00400044	1E	DB 1E	

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

0040007F	00	DB 00	
00400080	50 45 00 00	ASCII "PE"	PE signature (PE)
00400084	4C01	DW 014C	Machine = IMAGE_FILE_MACHINE_I386
00400086	0400	DW 0004	NumberOfSections = 4
00400088	2924D90A	DD 0AD92429	TimeDateStamp = AD92429
0040008C	00000000	DD 00000000	PointerToSymbolTable = 0
00400090	00000000	DD 00000000	NumberOfSymbols = 0
00400094	E000	DW 00E0	SizeOfOptionalHeader = E0 (224.)
00400096	8F81	DW 818F	Characteristics = EXECUTABLE_IMAGE 32BIT_MACHINE RE...
00400098	0B01	DW 010B	MagicNumber = PE32
0040009A	02	DB 02	MajorLinkerVersion = 2
0040009B	19	DB 19	MinorLinkerVersion = 19 (25.)
0040009C	00100000	DD 00001000	SizeOfCode = 1000 (4096.)
004000A0	00100000	DD 00001000	SizeOfInitializedData = 1000 (4096.)
004000A4	00800000	DD 00008000	SizeOfUninitializedData = 8000 (32768.)
004000A8	00100000	DD 00001000	AddressOfEntryPoint = 1000
004000AC	00900000	DD 00009000	BaseOfCode = 9000
004000B0	00A00000	DD 0000A000	BaseOfData = A000
004000B4	00004000	DD 00400000	ImageBase = 400000
004000B8	00100000	DD 00001000	SectionAlignment = 1000
004000BC	00020000	DD 00000200	FileAlignment = 200

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.

0040106D	C705 80204000	MOV DWORD PTR DS:[402080],5	
00401077	C705 84204000	MOV DWORD PTR DS:[402084],402110	
00401081	C705 88204000	MOV DWORD PTR DS:[402088],4020F4	
0040108B	68 64204000	PUSH 402064	
00401090	E8 F3030000	CALL 00401488	
00401095	6A 00	PUSH 0	
00401097	FF35 CA204000	PUSH DWORD PTR DS:[4020CA]	
0040109D	6A 00	PUSH 0	
0040109F	6A 00	PUSH 0	
004010A1	68 00800000	PUSH 8000	
004010A6	68 00800000	PUSH 8000	
004010AB	6A 6E	PUSH 6E	
004010AD	68 B4000000	PUSH 0B4	
004010B2	68 0000CF00	PUSH 0CF0000	

Address	Hex dump	Data	Comment
0040007F	00	DB 00	
00400080	50 45 00 00	ASCII "PE"	PE signature (PE)
00400084	4C01	DW 014C	Machine = IMAGE_FILE_MACH
00400086	0400	DW 0004	NumberOfSections = 4
00400088	2924D90A	DD 0AD92429	TimeDateStamp = AD92429
0040008C	00000000	DD 00000000	PointerToSymbolTable = 0
00400090	00000000	DD 00000000	NumberOfSymbols = 0
00400094	E000	DW 00E0	SizeOfOptionalHeader = E
00400096	8F81	DW 818F	Characteristics = EXECUT
00400098	0B01	DW 010B	MagicNumber = PE32
0040009A	02	DB 02	MajorLinkerVersion = 2
0040009B	19	DB 19	MinorLinkerVersion = 19
0040009C	00100000	DD 00001000	SizeOfCode = 1000 (4096.
004000A0	00100000	DD 00001000	SizeOfInitializedData =
004000A4	00800000	DD 00008000	SizeOfUninitializedData =
004000A8	00100000	DD 00001000	AddressOfEntryPoint = 10
004000AC	00900000	DD 00009000	BaseOfCode = 9000
004000B0	00A00000	DD 0000A000	BaseOfData = A000
004000B4	00004000	DD 00400000	ImageBase = 400000

Lo marcamos y vamos a MODIFY INTEGER.

Modify dword at 00400...

Hexadecimal: 00001000

Signed: 4096

Unsigned: 4096

OK Cancel

Address	Hex dump	Data	Comment
0040007F	00	DB 00	
00400080	50 45 00 00	ASCII "PE"	PE signature (PE)
00400084	4C01	DW 014C	Machine = IMAGE_FILE_MACHINE_I386
00400086	0400	DW 0004	NumberOfSections = 4
00400088	2924D90A	DD 0AD92429	TimeDateStamp = AD92429
0040008C	00000000	DD 00000000	PointerToSymbolTable = 0
00400090	00000000	DD 00000000	NumberOfSymbols = 0
00400094	E000	DW 00E0	SizeOfOptionalHeader = E0 (224.)
00400096	8F81	DW 818F	Characteristics = EXECUTABLE_IMAGE
00400098	0B01	DW 010B	MagicNumber = PE32
0040009A	02	DB 02	MajorLinkerVersion = 2
0040009B	19	DB 19	MinorLinkerVersion = 19 (25.)
0040009C	00100000	DD 00001000	SizeOfCode = 1000 (4096.)
004000A0	00100000	DD 00001000	SizeOfInitializedData = 1000 (4096.)
004000A4	00800000	DD 00008000	SizeOfUninitializedData = 8000 (32768.)
004000A8	00100000	DD 00001000	AddressOfEntryPoint = 1000
004000AC	00100000	DD 00001000	BaseOfCode = 1000
004000B0	00A00000	DD 0000A000	BaseOfData = A000
004000B4	00004000	DD 00400000	ImageBase = 400000
004000B8	00100000	DD 00001000	SectionAlignment = 1000

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

Address	Hex dump	Data
0040007F	00	DB 00
00400080	50 45 00 00	ASCII "PE"
00400084	4C01	DW 014C
00400086	0400	DW 0004
00400088	2924D90A	DD 0AD92429
0040008C	00000000	DD 00000000
00400090	00000000	DD 00000000
00400094	E000	DW 00E0
00400096	8F81	DW 818F
00400098	0B01	DW 010B
0040009A	02	DB 02
0040009B	19	DB 19
0040009C	00100000	DD 00001000
004000A0	00100000	DD 00001000
004000A4	00800000	DD 00008000
004000A8	00100000	DD 00001000
004000AC	00100000	DD 00001000
004000B0	00A00000	DD 0000A000
004000B4	00004000	DD 00400000

View executable file

Copy to executable file

Go to

Hex

Text

Short

Long

Float

Disassemble

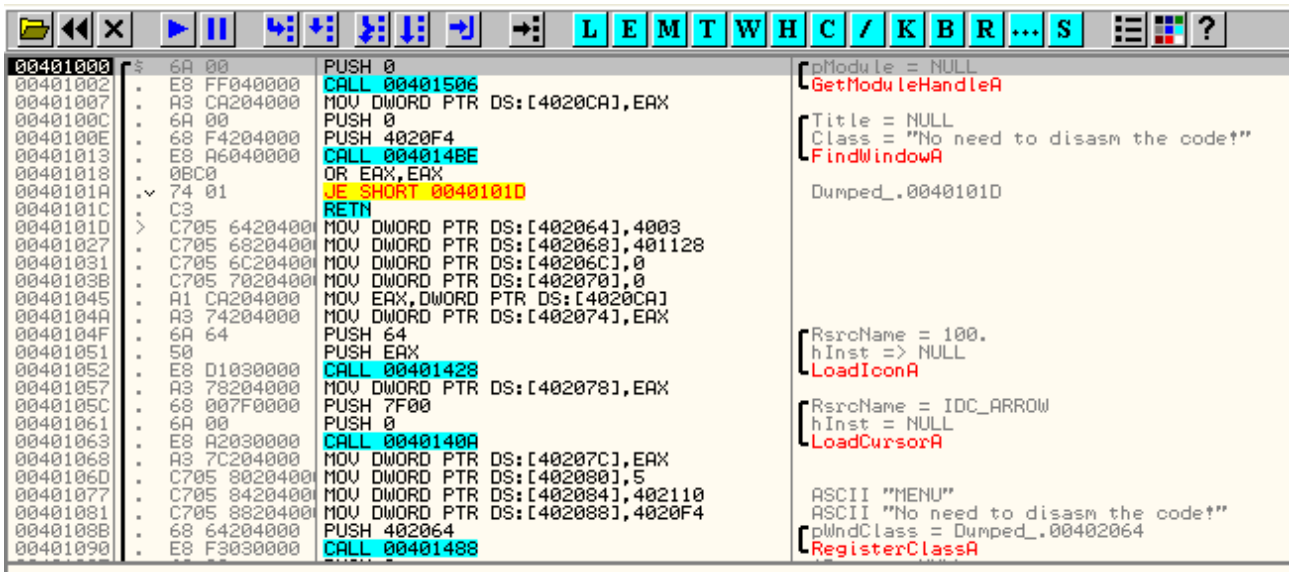
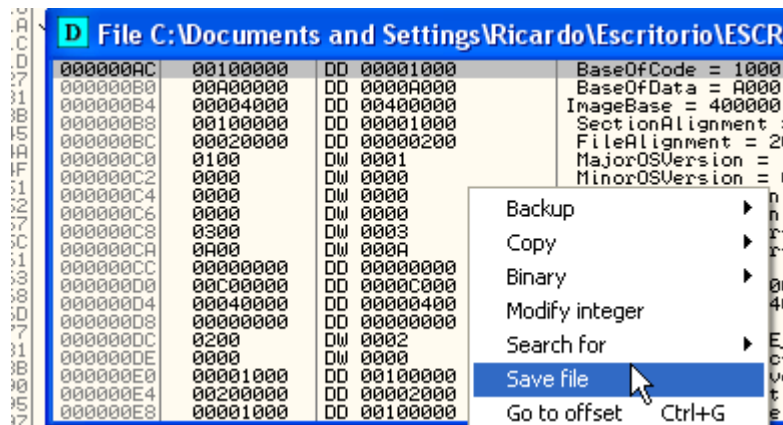
✓ Special

Appearance

BaseOfCode = 1000

BaseOfData = A000

ImageBase = 400000



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.

003B0000	00003000			Map	R	R	\Device\Harddis
003C0000	00004000			Priv	RW	RW	
003D0000	00002000			Map	R	R	
003F0000	00002000			Map	R	R	
00400000	00001000	Dumped_		Imag	R	RWE	
00401000	00008000	Dumped_	UPX0	Imag	R	RWE	
00409000	00001000	Dumped_	UPX1	Imag	R	RWE	
0040A000	00001000	Dumped_	.rsrc	Imag	R	RWE	
0040B000	00001000	Dumped_	.mact	Imag	R	RWE	
00410000	0000A000			Map	R	R	
004D0000	00002000			Map	R	R	
004E0000	00103000			Map	R	R	

Si vamos a ver las secciones del dumpado veamos que para repararlas el IMP REC le agrega una seccion nueva llamada mact 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.

Dumper .<ModuleEntryPoint>				ASCII
Address	Hex dump			
0040B000	00 00 00 00 00 00 00 00 00 00 00 00 78 B0 00 00		K...
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		R1..H2....
0040B040	00 00 00 00 00 00 00 00 98 B3 00 00 58 32 00 00		91..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 48 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..00LoadCursor
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			eeep.u0GetWindowR
0040B0E0	65 63 74 00 C9 01 4C 6F 61 64 53 74 72 69 6E 67			ect..00LoadString
0040B0F0	41 00 BC 01 4C 6F 61 64 49 63 6F 6E 41 00 B6 01			A..0LoadIconA..0
0040B100	4C 6F 61 64 42 69 74 6D 61 70 41 00 57 02 53 65			LoadBitmapA..0Se
0040B110	74 46 6F 63 75 73 00 00 D0 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..0PostQuit
0040B130	4D 65 73 73 61 67 65 00 D3 02 57 69 6E 48 65 6C			Message..0WinHel
0040B140	70 41 00 00 94 01 49 6E 76 61 6C 69 64 61 74 65			pA..00Invalidate
0040B150	52 65 63 74 00 00 A7 02 54 72 61 6E 73 6C 61 74			Rect..00Translat
0040B160	65 41 63 63 65 6C 65 72 61 74 6F 72 00 00 EA 01			eAccelerator..00
0040B170	4D 6F 76 65 57 69 6E 64 6F 77 00 00 AB 02 54 72			MoveWindow..%0T
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..0Sh
0040B1A0	6F 77 57 69 6E 64 6F 77 00 00 3C 02 53 65 6E 64			owWindow..<0Send
0040B1B0	4D 65 73 73 61 67 65 41 00 00 78 02 53 65 74 54			MessageA..<0SetT
0040B1C0	69 6D 65 72 00 00 84 02 53 65 74 57 69 6E 64 6F			iner..ä0SetWindo
0040B1D0	77 50 6F 73 00 00 BC 02 55 70 64 61 74 65 57 69			wPos..#0UpdateWi
0040B1E0	6E 64 6F 77 00 00 17 02 52 65 67 69 73 74 65 72			ndow..#0Register
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 0F 00 44 65 66 57 69 6E			ndowExA..A.DefWin
0040B220	64 6F 77 50 72 6F 63 41 00 00 9F 00 44 69 61 6C			dowProcA..f.Dial
0040B230	6F 67 42 6F 78 50 61 72 61 6D 41 00 A2 00 44 69			ogBoxParamA..ö.Di
0040B240	73 70 61 74 63 68 4D 65 73 73 61 67 65 41 00 00			spatchMessageA..
0040B250	B9 00 44 73 61 72 4D 65 65 7E 43 61 72 00 C7 00			#_DrawMenuBar..%

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 reparar.

Dumper .<ModuleEntryPoint>				ASCII
Address	Hex dump			
0040B000	00 00 00 00 00 00 00 00 00 00 00 00 78 B0 00 00		K...
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		R1..H2....
0040B040	00 00 00 00 00 00 00 00 98 B3 00 00 58 32 00 00		91..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 48 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

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.

Dumped .<ModuleEntryPoint>																		
Address	Hex dump												ASCII					
00400078	75	73	65	72	33	32	2E	64	6C	6C	00	00	B3	01	48	69	user32.dll..!0Ki	
00400088	6C	6C	54	69	6D	65	72	00	5E	01	47	65	74	53	79	73	llTimer.^@getSys	
00400098	74	65	6D	4D	65	74	72	69	63	73	00	00	B8	01	4C	6F	temMetrics..@Lo	
004000A8	61	64	43	75	72	73	6F	72	41	00	B4	01	4C	6F	61	64	adCursorA.#@Load	
004000B8	41	63	63	65	6C	65	72	61	74	6F	72	73	41	00	DC	01	AcceleratorsA.#@	
004000C8	4D	65	73	73	61	67	65	42	65	65	70	00	75	01	47	65	MessageBeep.u@Ge	
004000D8	74	57	69	6E	64	6F	77	52	65	63	74	00	C9	01	4C	6F	tWindowRect.#@Lo	
004000E8	61	64	53	74	72	69	6E	67	41	00	BC	01	4C	6F	61	64	adStringA.#@Load	
004000F8	49	63	6F	6E	41	00	B6	01	4C	6F	61	64	42	69	74	6D	IconA.#@LoadBitm	
00400108	61	70	41	00	57	02	53	65	74	46	6F	63	75	73	00	00	anA.#@SetFocus..	

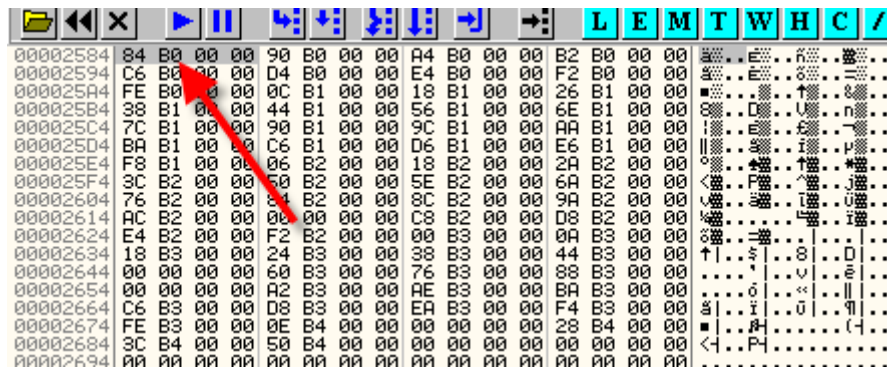
Es user32.dll.

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.

Dumped .<ModuleEntryPoint>																		
Address	Hex dump												ASCII					
00403164	00	00	00	00	00	00	00	00	00	00	00	00	00				
00403174	00	00	00	00	00	00	00	00	00	00	00	00	00				
00403184	42	8C	D1	77	90	8F	D1	77	3E	0B	D2	77	24	15	D3	77	Bi0w0A0w>0Ew\$0Ew	
00403194	4C	1F	D3	77	04	B6	D1	77	E8	0F	D2	77	24	13	D2	77	L*EwE0A0w*Ew\$!!Ew	
004031A4	DA	5E	D2	77	60	DA	D1	77	EA	04	D5	77	11	12	D2	77	r^Ew' r0w0* w0Ew	
004031B4	35	EE	D3	77	F5	B5	D1	77	9C	FA	D2	77	EC	08	D1	77	S^EwS0A0w0Ew\$0Ew	
004031C4	F6	8B	D1	77	83	F7	D4	77	A4	D8	D1	77	9A	F3	D2	77	+i0w0A0w0Ew\$0Ew	
004031D4	2E	8C	D1	77	18	C0	D1	77	F9	D7	D1	77	8C	14	D2	77	.i0w0A0w0Ew\$0Ew	
004031E4	09	B6	D1	77	5E	02	D2	77	EE	D4	D1	77	1C	B1	D3	77	.A0w0Ew0Ew\$0Ew	
004031F4	B8	96	D1	77	9C	F3	D4	77	50	62	D2	77	10	B6	D1	77	@00w0Ew0EwP0Ew#A0w	
00403204	81	E5	D2	77	C7	86	D1	77	16	48	D2	77	1E	AC	D6	77	u0EwA00w0Ew\$A0w	
00403214	42	10	D2	77	00	00	00	00	C1	C9	80	7C	99	68	82	7C	B0Ew...+f0C!0k0E!	
00403224	2F	FE	80	7C	2D	FF	80	7C	E0	C6	80	7C	77	98	80	7C	/0C!- C!00C!w0C!	
00403234	9F	0F	81	7C	29	B5	80	7C	0E	18	80	7C	A2	CA	81	7C	f0u!)A0C!00C!00u!	
00403244	00	00	00	00	DD	15	C5	58	21	9B	C4	58	38	8B	C6	58!S0X0000000000	
00403254	00	00	00	00	0C	BC	EF	77	26	F1	F0	77	E9	49	F2	77" w0000000000	
00403264	68	E0	EF	77	E1	61	EF	77	C9	D0	F0	77	51	E0	F0	77	h0'w00A'wF!-w000-w	
00403274	2D	6C	EF	77	98	6E	EF	77	00	00	00	00	08	7C	37	76	-l'w0n'w...i!70	
00403284	1E	31	36	76	CD	46	38	76	00	00	00	00	00	00	00	00	▲160F000000000000	
00403294	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
004032A4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
004032B4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

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 sobrescrita 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.

Address	Hex dump	
00403164	00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00403174	00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00403184	42 8C D1 77 9D 8F D1 77 3E 0B D2 77 2E	
00403194	4C 1F D3 77	Backup
004031A4	DA 5E D2 77	Copy
004031B4	35 EE D3 77	
004031C4	F6 8B D1 77	Binary
004031D4	2E 8C D1 77	
004031E4	09 B6 D1 77	Label
004031F4	B8 96 D1 77	
00403204	81 E5 D2 77	Breakpoint
00403214	42 10 D2 77	
00403224	2F FE 80 7C	Search for
00403234	9F 0F 81 7C	
00403244	00 00 00 00	Follow DWORD in Disassembl
00403254	00 00 00 00	
00403264	68 E0 EF 77	Follow DWORD in Dump
00403274	2D 6C EF 77	
00403284	1E 31 36 76	Find references
00403294	00 00 00 00	
004032A4	00 00 00 00	View executable file
004032B4	00 00 00 00	
004032C4	00 00 00 00	Copy to executable file



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

Address	Hex dump	ASCII
0040B084	B3 01 4B 69 6C 6C 54 69 6D 65 72 00 5E 01 47 65	!@KillTimer.^@Ge
0040B094	74 53 79 73 74 65 6D 4D 65 74 72 69 63 73 00 00	tSystemMetrics..
0040B0A4	B8 01 4C 6F 61 64 43 75 72 73 6F 72 41 00 B4 01	@@LoadCursorA.1@
0040B0B4	4C 6F 61 64 41 63 63 65 6C 65 72 61 74 6F 72 73	LoadAccelerators
0040B0C4	41 00 DC 01 4D 65 73 73 61 67 65 42 65 65 70 00	A..@MessageBeep.
0040B0D4	75 01 47 65 74 57 69 6E 64 6F 77 52 65 63 74 00	u@GetWindowRect.
0040B0E4	C9 01 4C 6F 61 64 53 74 72 69 6E 67 41 00 BC 01	f@LoadStringA."@

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 reconstruccion 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 completamente destrozadas, apis redireccionadas y casos dificiles, para los cuales es imprescindible haber entendido bien como 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