INTRODUCCION AL CRACKING CON OLLYDBG PARTE 35

Seguiremos practicando y desempacando cada vez con packers mas dificiles, aumentando levemente el grado de dificultad.

El siguiente packer en la escala de dificultad es el aspack, casi muy parecido al UPX, y para el cual ya tenemos el crackme UnPackMe_ASPack2.12.exe que se envio en partes anteriores, al cual ademas ya le habiamos encontrado el OEP.

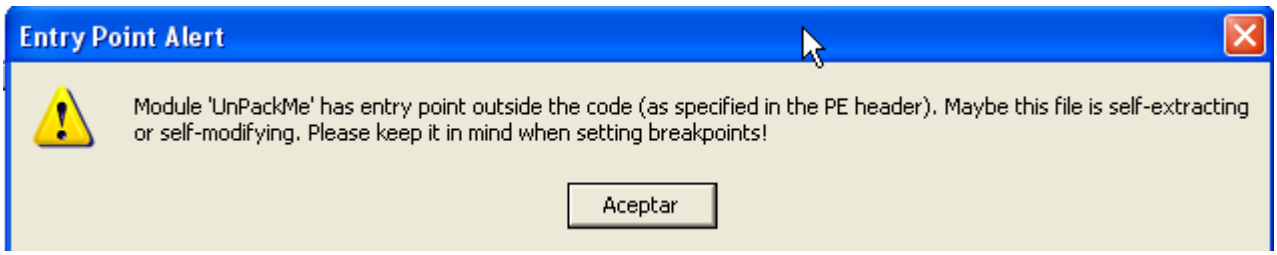
Coloco para practicar la dll del OLLYDUMP en la carpeta de plugins ya que lo dumpeare con el mismo.

http://www.ricnar456.dyndns.org/HERRAMIENTAS/L-M-N-O-P/Plugins_Olly/OllyDump%20parche%20de%20parasito.rar

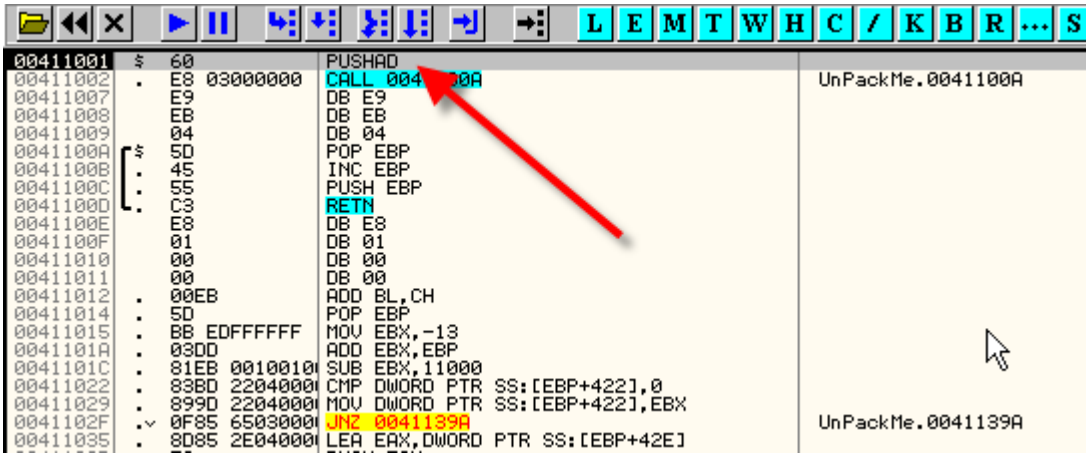
user y pass:hola

Esa es la ultima version, parcheado algun bug que tenia por Parasito.

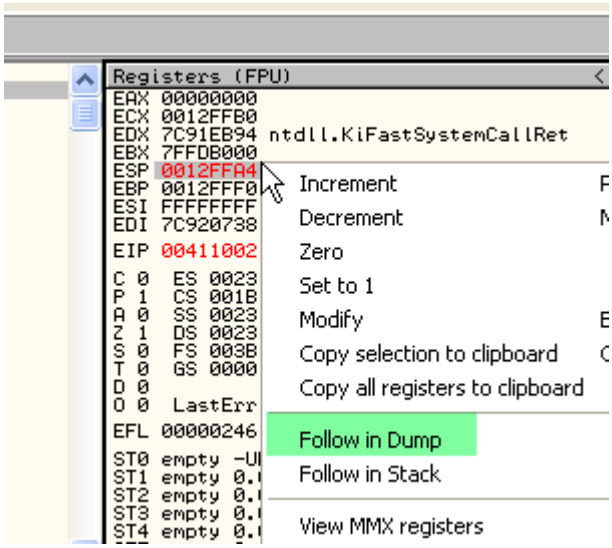
Abremos el OLLYDBG protegido con los plugins para ocultarlo, y lleguemos al OEP con el metodo el PUSHAD.



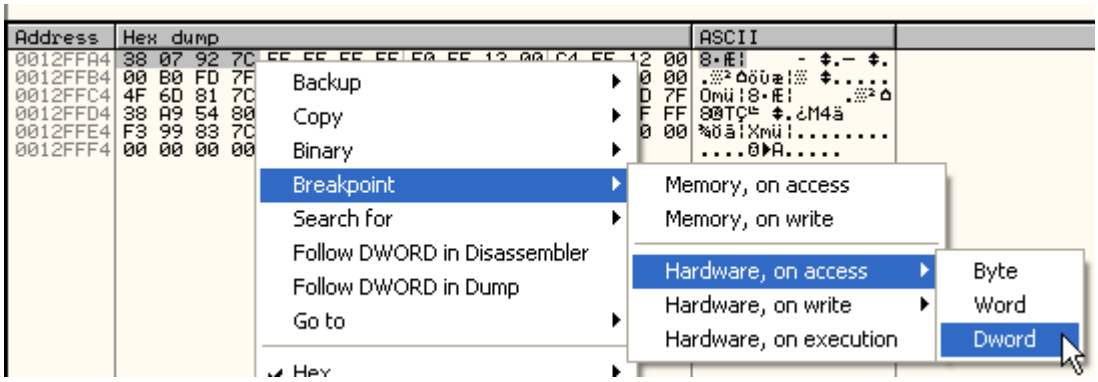
Vemos que aquí nos avisa que el entry point esta fuera de la seccion code como es lo usual, en la mayoria de los packers.



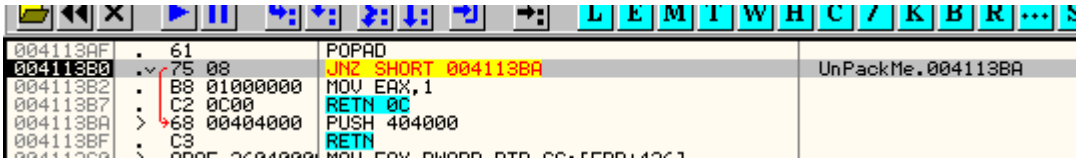
Ahi vemos el PUSHAD inicial al cual pasamos con f7 y luego hacemos.



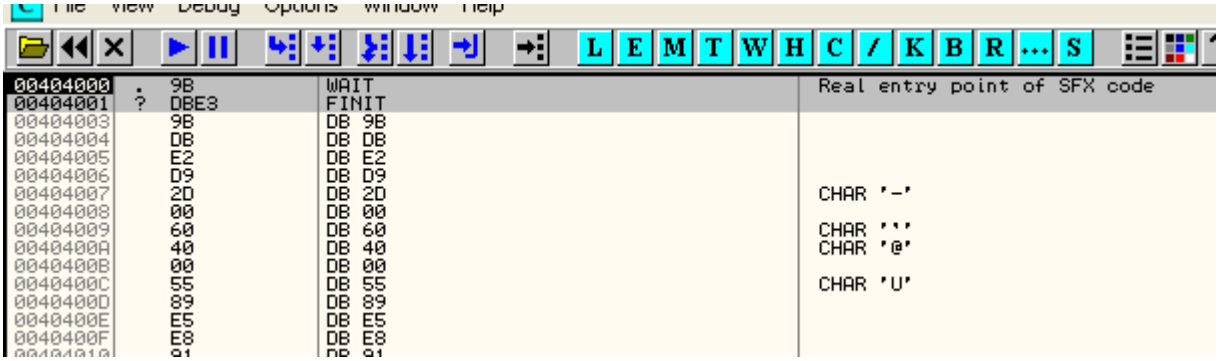
ESP-FOLLOW IN DUMP para colocar un Hardware Breakpoint on accesss en los valores de los registros que guardo con el PUSHAD en el DUMP.



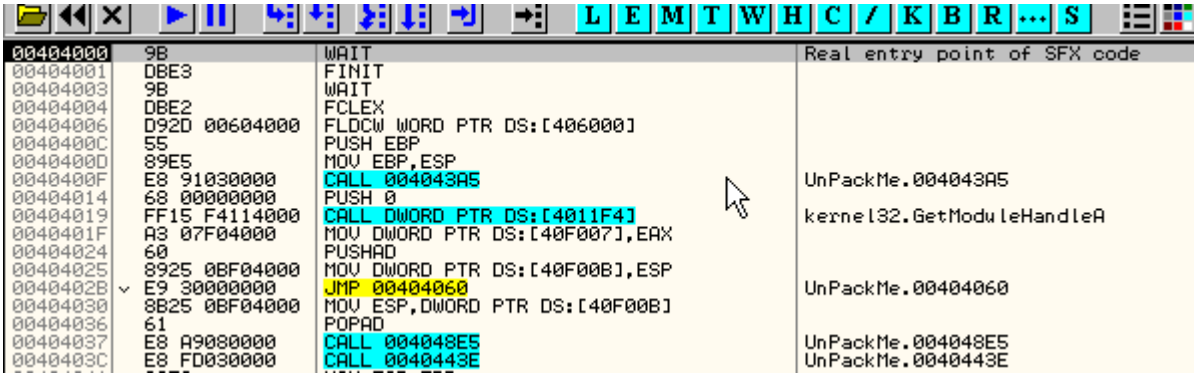
Luego apeto F9 con lo cual doy RUN



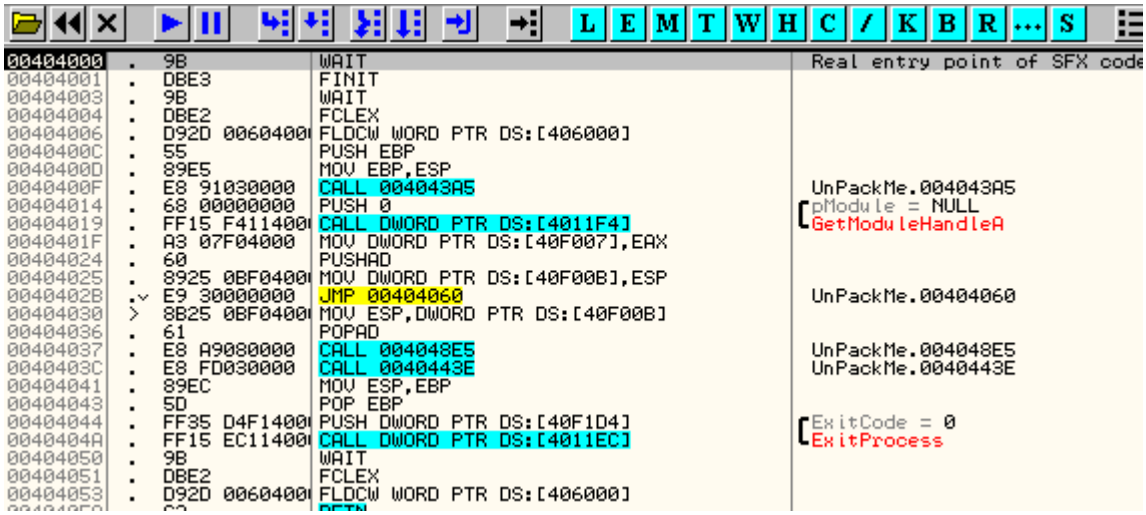
Con lo cual para justo despues del POPAD que restaura los valores guardados a los registros, traceo hasta llegar al OEP con f7.



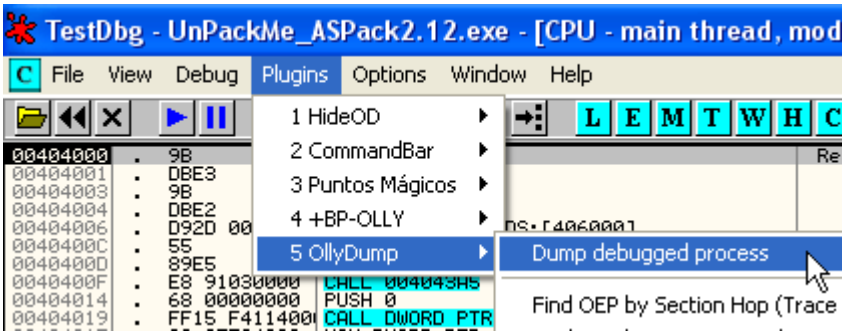
Como veo que no se entiende el codigo, le quito el analisis.

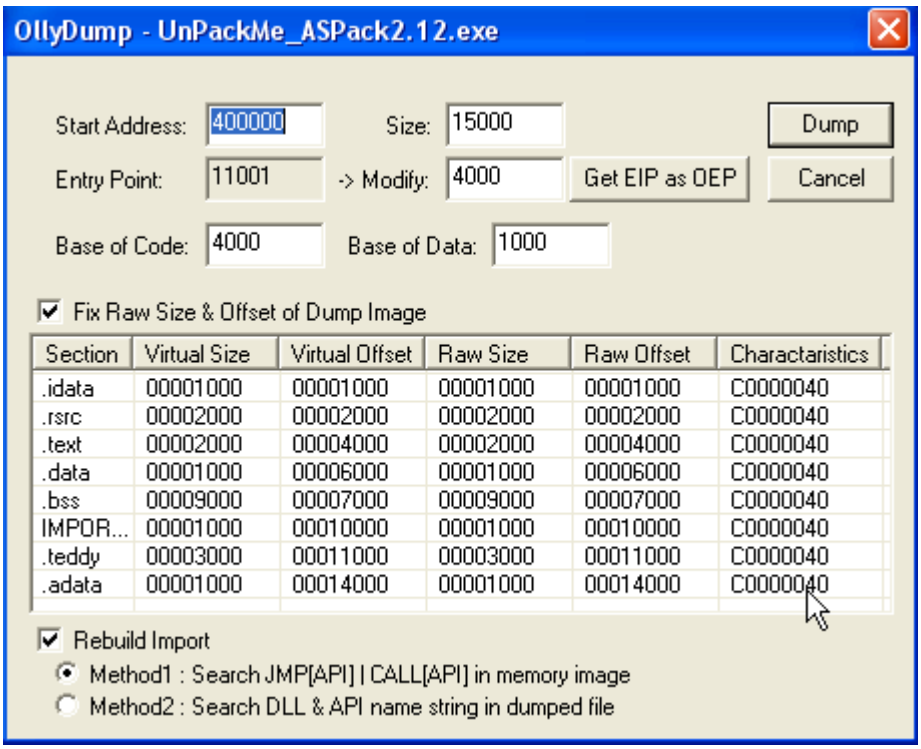


Y vemos que si lo vuelvo a analizar mejora aun mas.



Luego procedere al dumpeado voy al menu PLUGINS y alli busco el OLLYDUMP

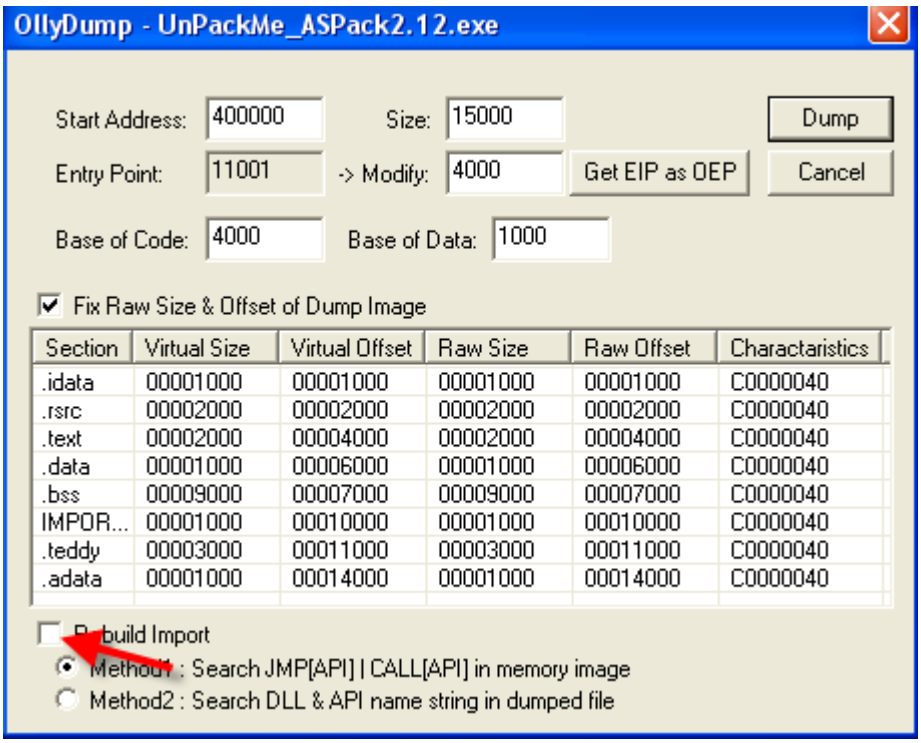




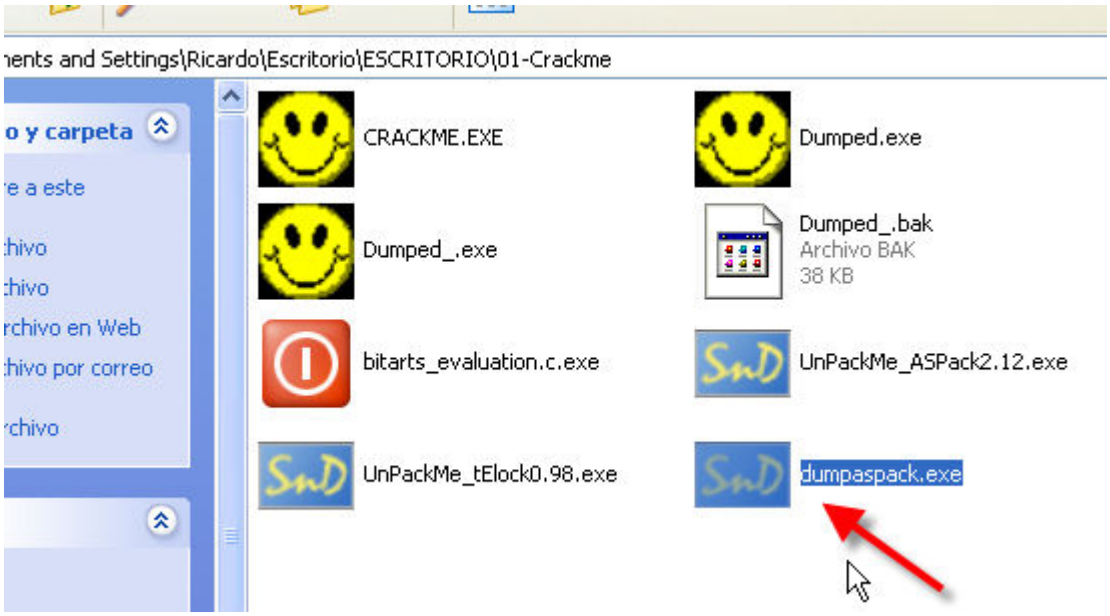
Nos aparece la ventana del plugin, en la cual ya vemos las cositas que podemos modificar, ya ahi podemos arreglar lo de la base of code, sin tener que luego cambiarlo en el header, en la ventana vemos base of code 4000, pero si recordamos este aspack corria en esta seccion que no es la primera, por lo cual le dejamos la base of code en 4000 que corresponde a 404000 que es la seccion donde esta el OEP y corre el programa.

Otro de los temas es la tilde de REBUILD IMPORT que esta abajo, el OLLYDUMP trata de hacer el trabajo del IMP REC para lo cual tiene dos opciones METHOD1 y METHOD2, que en algun packer sencillo puede funcionar, el que quiere a veces ganar tiempo, puede hacer un dumpeado con cada uno de estos metodos y ver si alguno funciona, aunque no es muy certero, pero alguna vez puede funcionar.

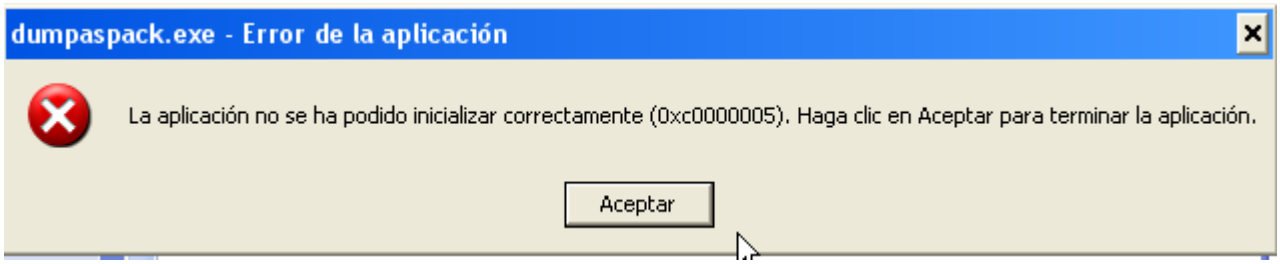
Nosotros le quitaremos la tilde en REBUILD IMPORT ya que lo haremos con el IMP REC que es mas confiable.



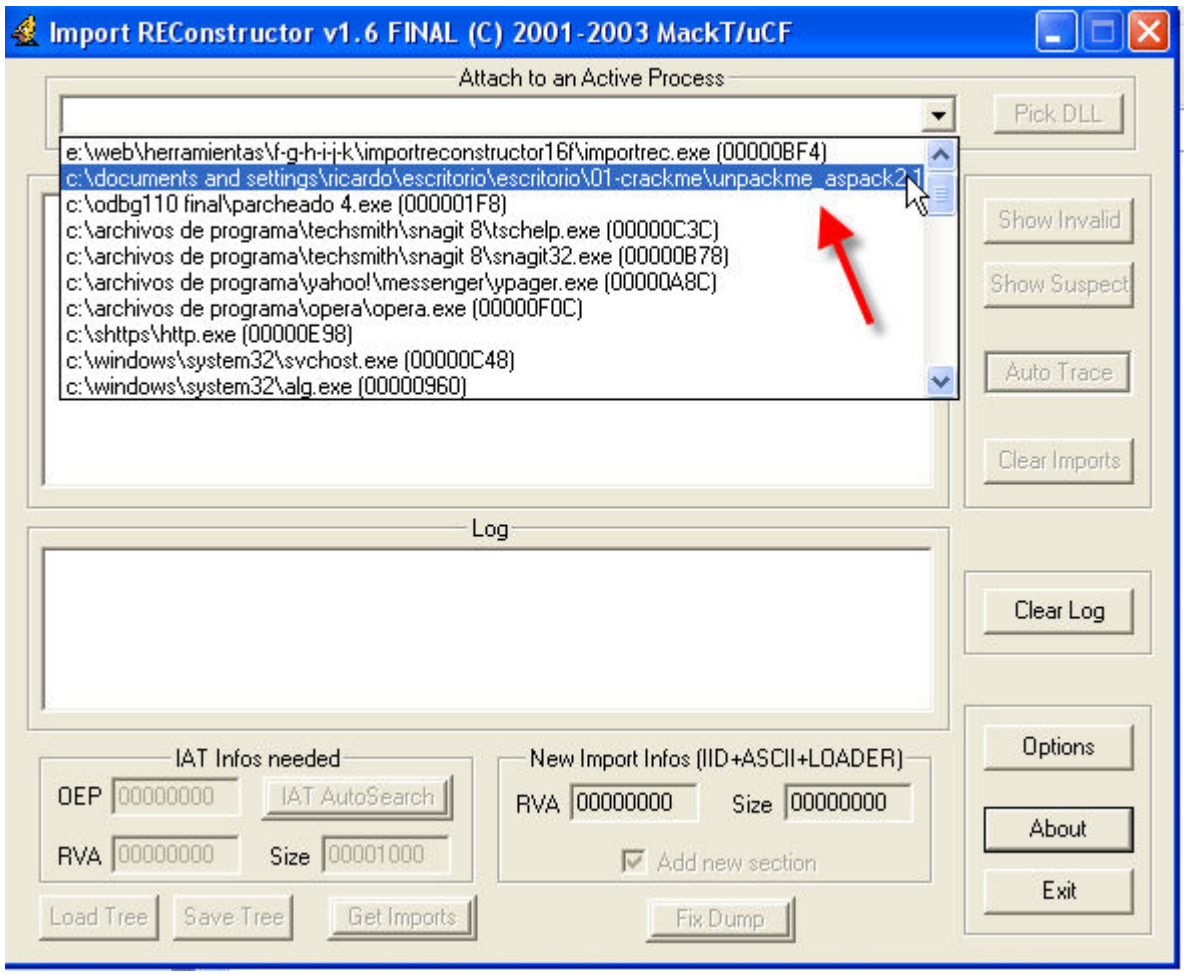
Bueno ahora podemos dumpear a ver que tal nos va.



Pues alli esta el dumpeado si lo ejecuto sin reparar la iat o bien correra solo en mi maquina con mucha suerte, o bien dara error veamos.



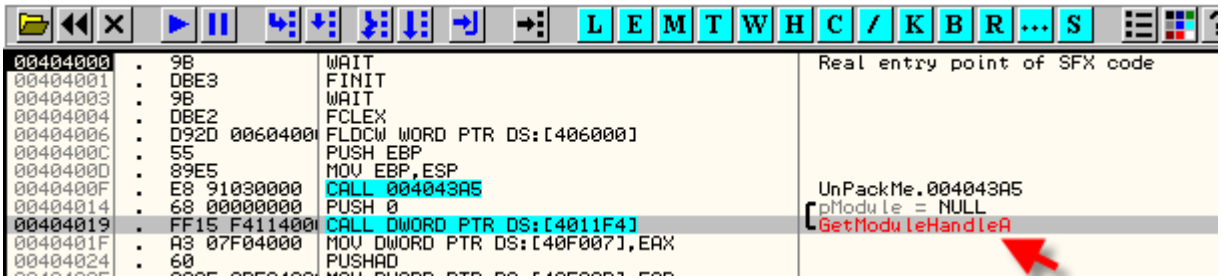
Bueno sin cerrar el archivo empacado que esta detenido en el OEP abrimos el IMP REC y elegimos dicho proceso en la lista del menu desplegable.



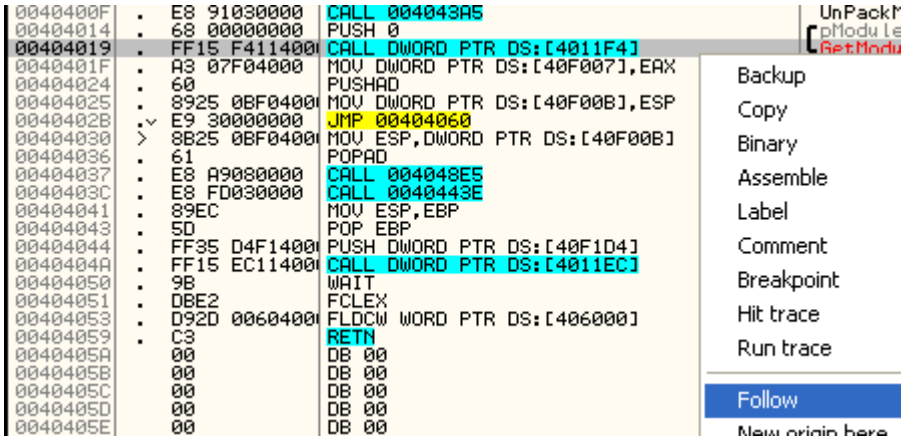
Volvemos al OLLYDBG para hallar los valores de INICIO DE IAT, LARGO y el OEP.

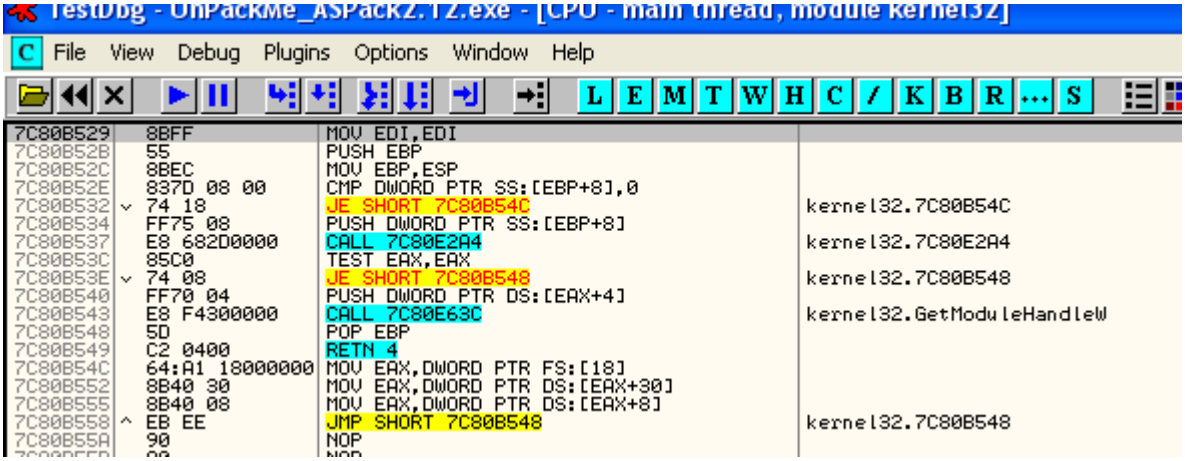
OEP es 404000 o sea que en el IMP REC ya que le debemos restar la imagebase que es 400000, quedara 4000.

Buscaremos el inicio y final de la iat como vimos para ellos hay que buscar una llamada a una api, justo abajo del oep esta la llamada a GetModulehandleA.



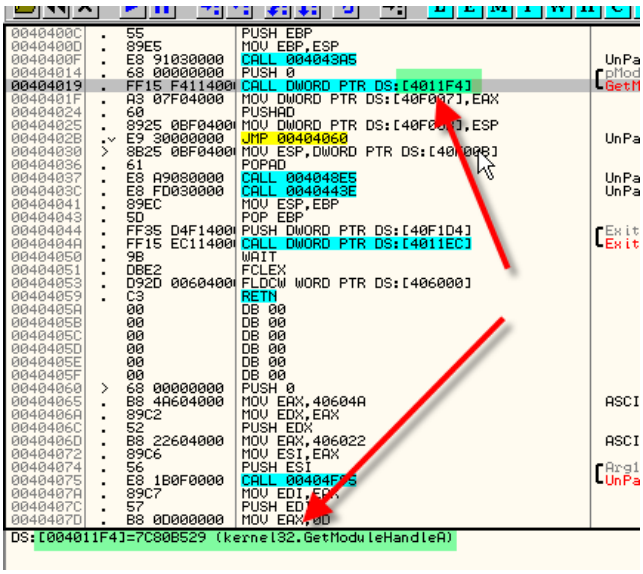
Si marco dicha linea y hago click derecho-FOLLOW



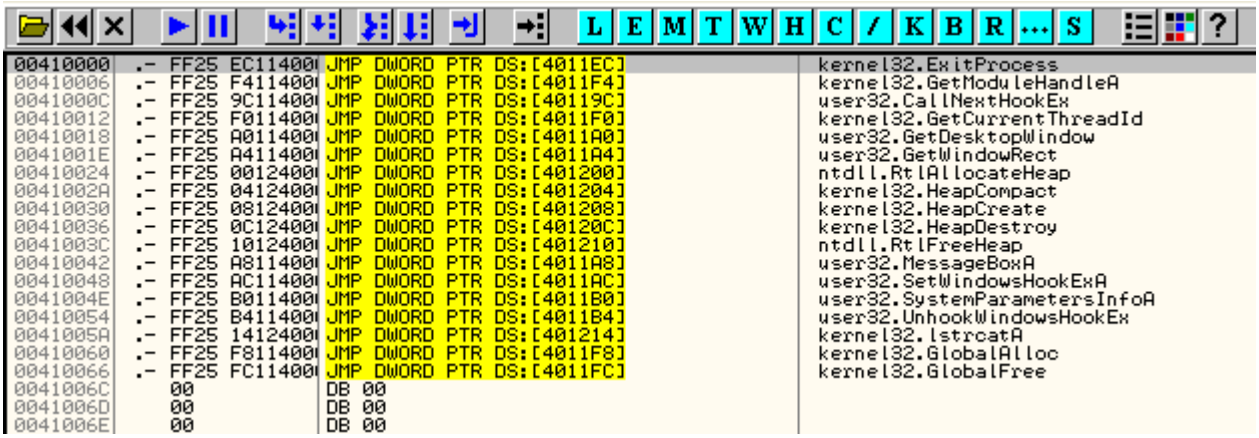


Veo que va directamente a la api sin JMPS INDIRECTOS intermedios por lo menos en esta llamada , (ya que si buscamos veremos que si hay JMPS INDIRECTOS lo que pasa es que no siempre los usa como en este caso)

Quiere decir aquí usa un CALL INDIRECTO, para saltar a la api, por lo cual es facil deducir que lee la direccion de la API directamente de la IAT para saltar a la misma correctamente.

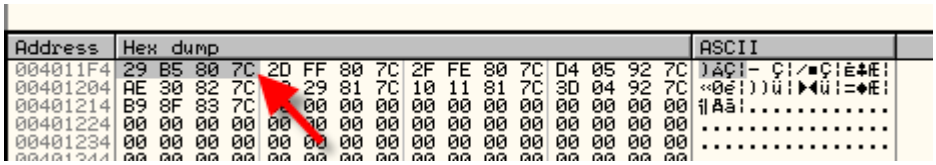


Bueno es facil de ver que 4011F4 es una entrada de la IAT donde guarda la direccion de la API, GetModuleHandleA.
El que gusta de ver los JMPS INDIRECTOS, buscando con FF 25 tambien los hallara.

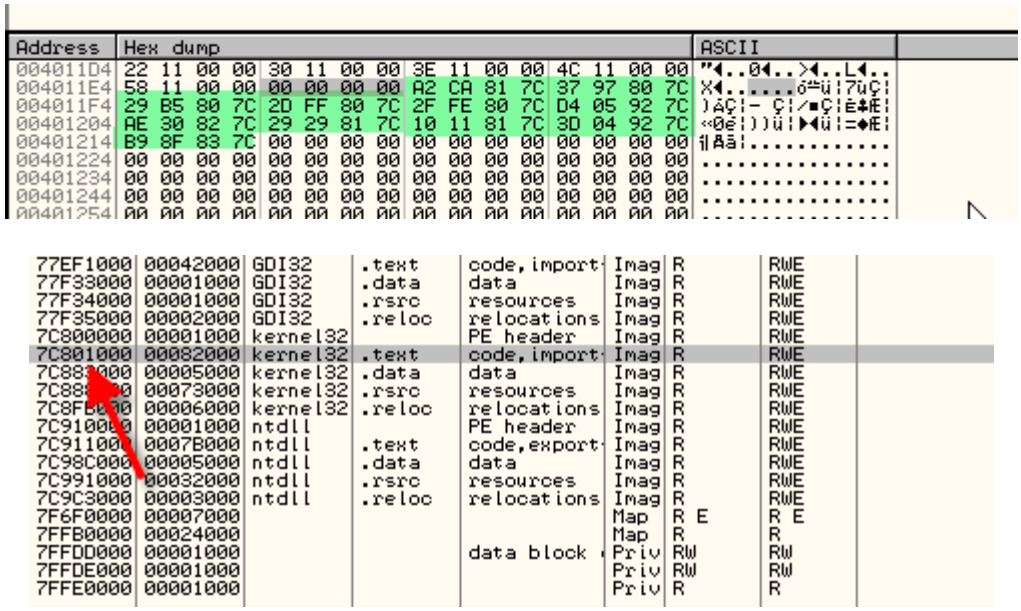


Y llegara al mismo resultado ya que el JMP a GetModulehandleA lee valores de la misma entrada de la IAT.

Vayamos en el DUMP a mirar dicha entrada y la IAT en general.



Alli vemos todas las entradas que son compañeras de la que miramos inicialmente, todas van a la seccion code de la misma dll, miremos en VIEW-M a que dll corresponden.

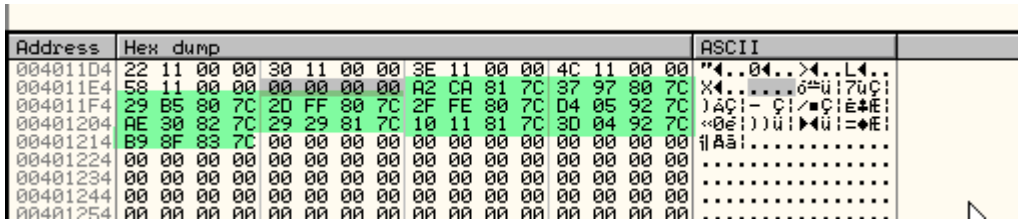


Address	Hex dump	ASCII
004011D4	22 11 00 00 30 11 00 00 3E 11 00 00 4C 11 00 00	"◀..0◀..>◀..L◀..
004011E4	58 11 00 00 00 00 00 00 A2 CA 81 7C 37 97 80 7C	X◀.....ó╩ü¦7ùÇ¦
004011F4	29 B5 80 7C 2D FF 80 7C 2F FE 80 7C D4 05 92 7C	)ÁÇ¦- Ç¦/■Ç¦È♣Æ¦
00401204	AE 30 82 7C 29 29 81 7C 10 11 81 7C 3D 04 92 7C	«0é¦))ü¦▶◀ü¦=♦Æ¦
00401214	B9 8F 83 7C 00 00 00 00 00 00 00 00 00 00 00 00	╣Åâ¦............
00401224	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00401234	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00401244	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00401254	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....

77EF1000	00042000	GDI32	.text	code,import	Imag	R	RWE
77F33000	00001000	GDI32	.data	data	Imag	R	RWE
77F34000	00001000	GDI32	.rsrc	resources	Imag	R	RWE
77F35000	00002000	GDI32	.reloc	relocations	Imag	R	RWE
7C800000	00001000	kernel32		PE header	Imag	R	RWE
7C801000	00082000	kernel32	.text	code,import	Imag	R	RWE
7C883000	00005000	kernel32	.data	data	Imag	R	RWE
7C888000	00073000	kernel32	.rsrc	resources	Imag	R	RWE
7C8FB000	00006000	kernel32	.reloc	relocations	Imag	R	RWE
7C910000	00001000	ntdll		PE header	Imag	R	RWE
7C911000	0007B000	ntdll	.text	code,export	Imag	R	RWE
7C98C000	00005000	ntdll	.data	data	Imag	R	RWE
7C991000	00032000	ntdll	.rsrc	resources	Imag	R	RWE
7C9C3000	00003000	ntdll	.reloc	relocations	Imag	R	RWE
7F6F0000	00007000				Map	R E	R E
7FFB0000	00024000				Map	R	R
7FFDD000	00001000			data block	Priv	RW	RW
7FFDE000	00001000				Priv	RW	RW
7FFE0000	00001000				Priv	R	R

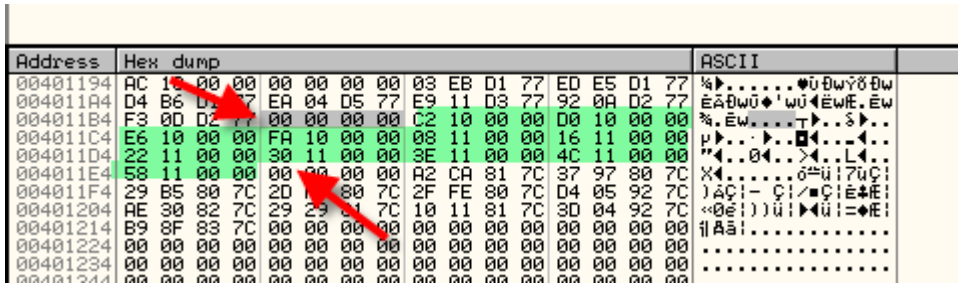
Todas caen dentro de dicha seccion, por lo cual vemos que son las entradas que corresponden a Kernel32.dll ya que apuntan a su seccion CODE.

Alli mismo podemos ver el final de la IAT ya que debajo de 401218 no hay mas nada, asi que el final de la iat es 401218, ahora nos queda hallar el inicio.



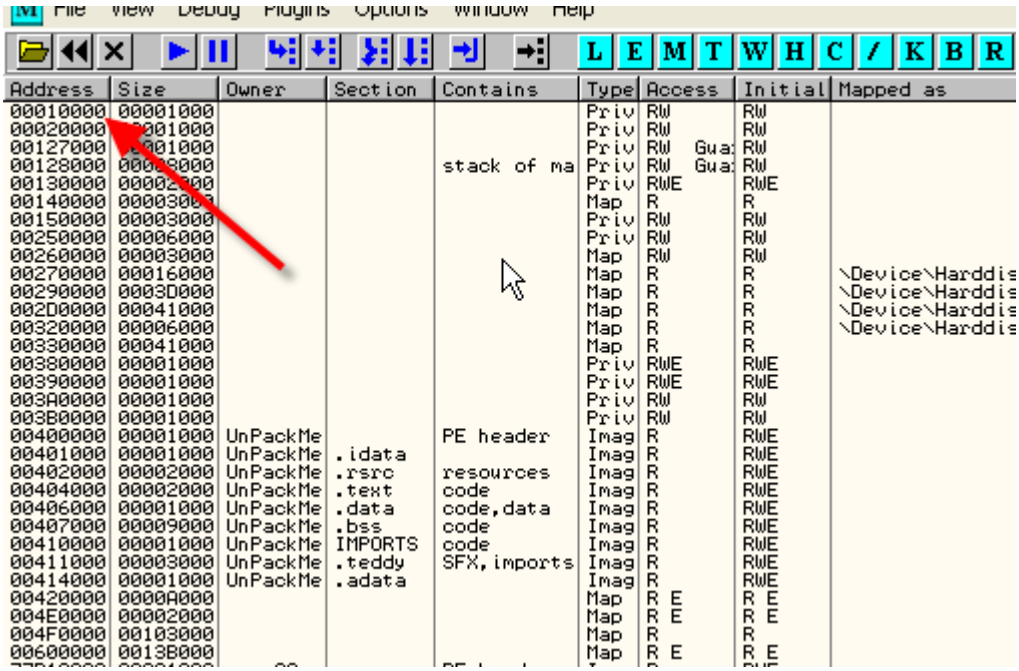
Address	Hex dump	ASCII
004011D4	22 11 00 00 30 11 00 00 3E 11 00 00 4C 11 00 00	"◀..0◀..>◀..L◀..
004011E4	58 11 00 00 00 00 00 00 A2 CA 81 7C 37 97 80 7C	X◀.....ó╩ü¦7ùÇ¦
004011F4	29 B5 80 7C 2D FF 80 7C 2F FE 80 7C D4 05 92 7C	)ÁÇ¦- Ç¦/■Ç¦È♣Æ¦
00401204	AE 30 82 7C 29 29 81 7C 10 11 81 7C 3D 04 92 7C	«0é¦))ü¦▶◀ü¦=♦Æ¦
00401214	B9 8F 83 7C 00 00 00 00 00 00 00 00 00 00 00 00	╣Åâ¦............
00401224	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00401234	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00401244	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00401254	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....

Vemos la separacion de ceros y antes otro grupo de entradas.

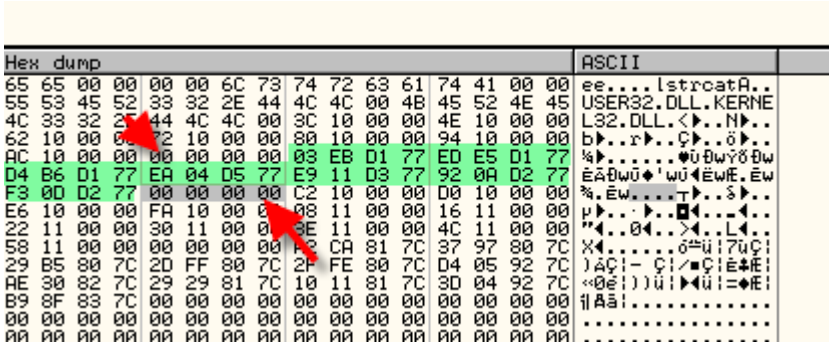


Address	Hex dump	ASCII
00401194	AC 10 00 00 00 00 00 00 03 EB D1 77 ED E5 D1 77	¼▶.....♥ûÐwýõÐw
004011A4	D4 B6 D1 77 EA 04 D5 77 E9 11 D3 77 92 0A D2 77	ÈáÐwÜ♦'wÚ◀ÉwÆ.Éw
004011B4	F3 0D D2 77 00 00 00 00 C2 10 00 00 D0 10 00 00	¾.Éw....┬▶..ð▶..
004011C4	E6 10 00 00 FA 10 00 00 08 11 00 00 16 11 00 00	µ▶.. ▶..◘◀...◀..
004011D4	22 11 00 00 30 11 00 00 3E 11 00 00 4C 11 00 00	"◀..0◀..>◀..L◀..
004011E4	58 11 00 00 00 00 00 00 A2 CA 81 7C 37 97 80 7C	X◀.....ó╩ü¦7ùÇ¦
004011F4	29 B5 80 7C 2D FF 80 7C 2F FE 80 7C D4 05 92 7C	)ÁÇ¦- Ç¦/■Ç¦È♣Æ¦
00401204	AE 30 82 7C 29 29 81 7C 10 11 81 7C 3D 04 92 7C	«0é¦))ü¦▶◀ü¦=♦Æ¦
00401214	B9 8F 83 7C 00 00 00 00 00 00 00 00 00 00 00 00	╣Åâ¦............
00401224	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00401234	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....

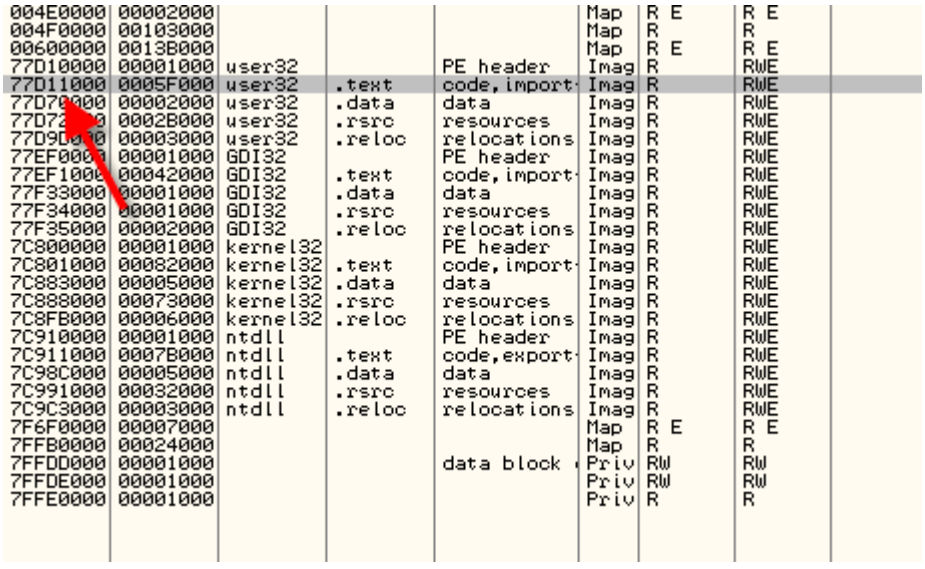
Que son exactamente estas entradas, vemos que en las direcciones adonde apuntan (10xx o 11xx) no hay dlls ni nada ya que la mas baja direccion en el mapa de memoria es 10000.



Asi que estas entradas ya que no van ni a una dll, ni a una seccion real, ya que podrian apuntar a alguna seccion creada por el packer, lo cual no es este caso, son basura metida para molestar ya veremos lo que hacemos con ellas, ahora sigamos subiendo.

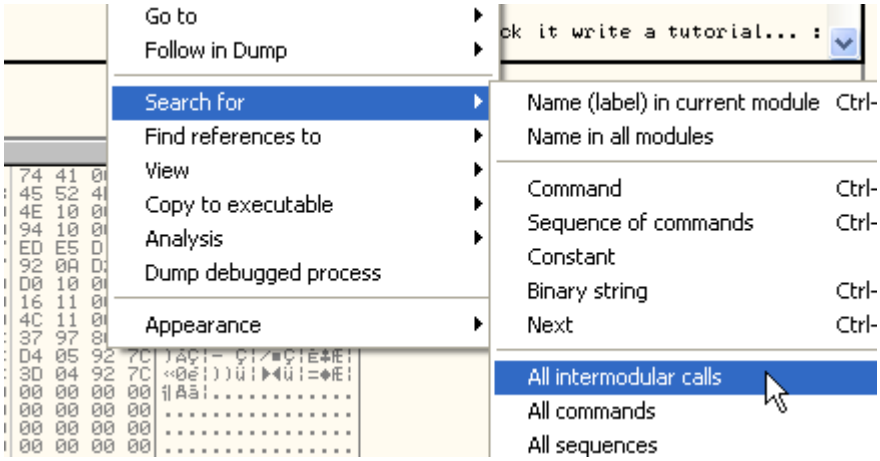


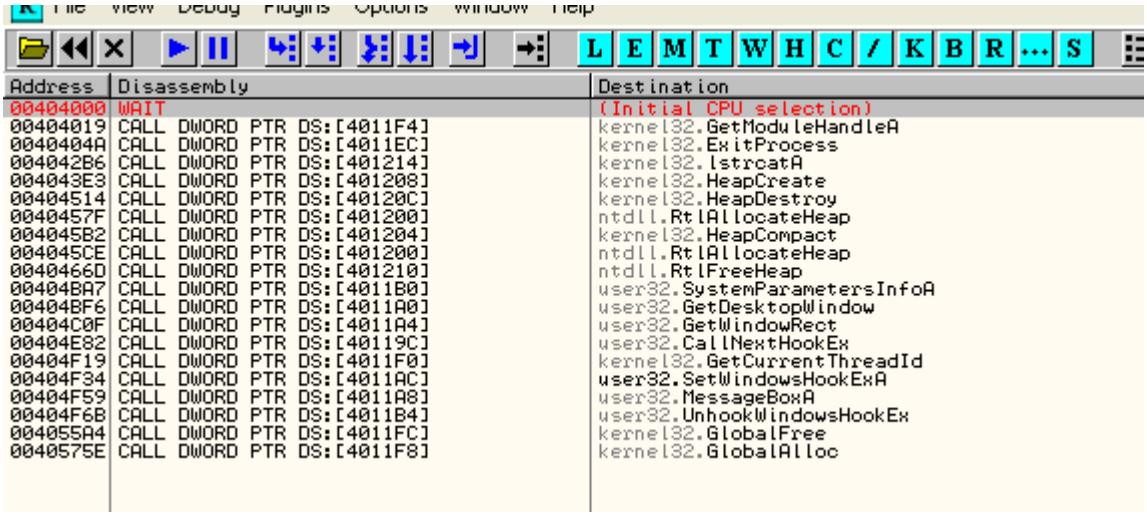
Vemos que entre ceros hay otro grupo de entradas que apuntan a direcciones 77Dxxxxx veamos en el mapa de memoria a que dll pertenecen.



Vemos que pertenecen a la User32.dll

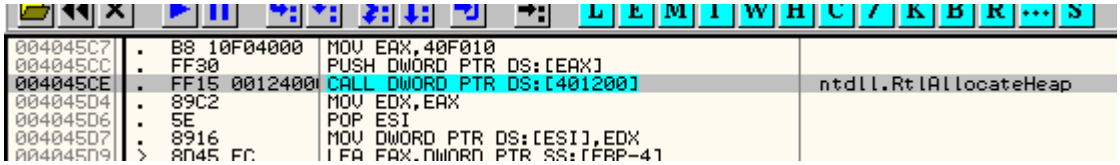
Tambien vemos que hay mas dlls en el mapa de memoria como GDI32 y Ntdll las cuales pueden haber sido cargadas por el packer para su uso, y no la usa el programa, para verificar esto, hagamos en el mismo listado SEARCH FOR – ALL INTERMODULAR CALLS.



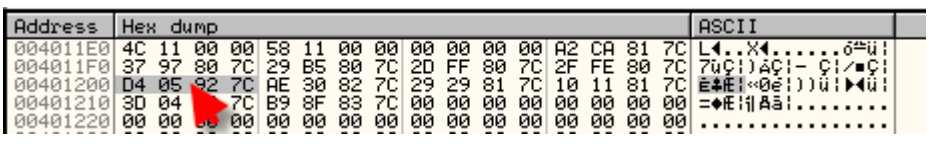


Vemos que hay llamadas a 3 dll, las dos que hallamos y falta arreglar la Ntdll ya que hay llamados a la misma, veamos.

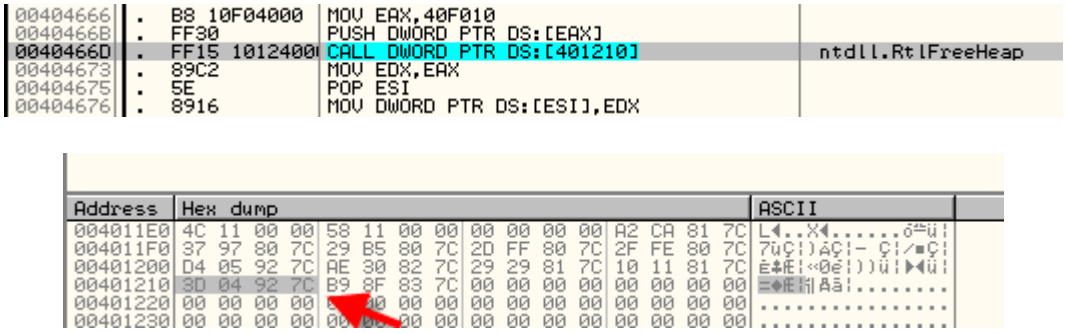
Si vamos a los calls de esas dll por ejemplo.



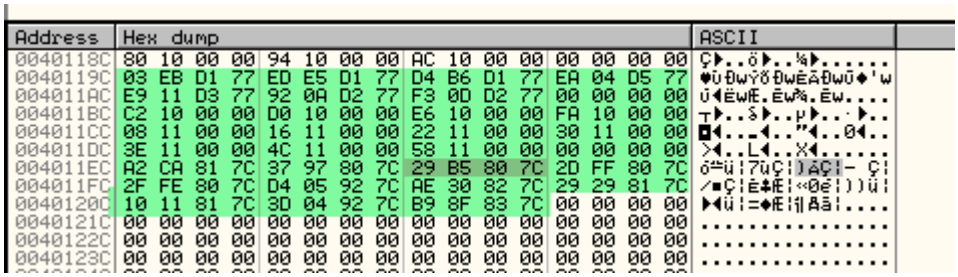
Vemos que la entrada esta en 401200 o sea que esta mezclada con las de kernel32.dll



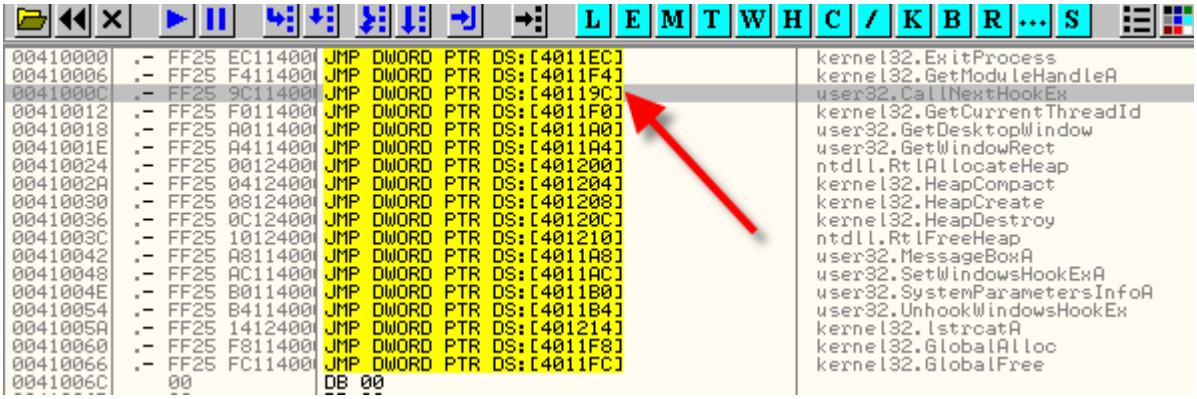
Lo mismo la otra de dicha dll esta tambien mezclada con las de kernel32.dll no nos dimos cuenta por la proximidad de las secciones code de ambas pero es asi.



Bueno vemos algunos problemas en esto de cualquier forma el inicio de la IAT, es el que abarca todas las entradas asi que



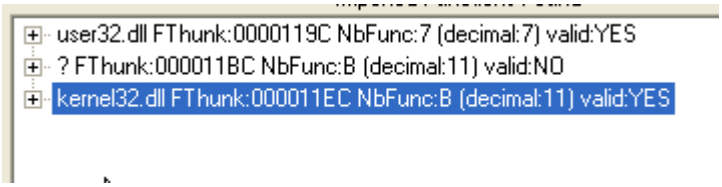
El inicio de la Iat es 40119C lo que concuerda con el menor valor hallado en la tabla de saltos.



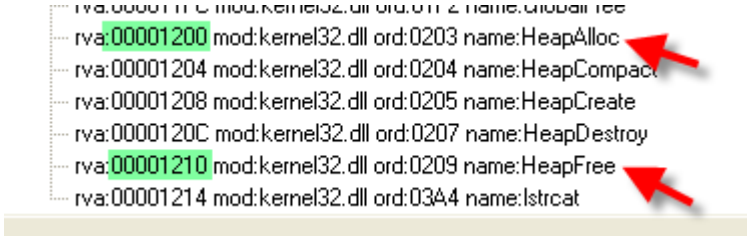
Vemos que es el mas pequeño de todos estos valores, asi que ya tenemos

OEP=4000
RVA o INICIO DE LA IAT=119C
LARGO= FINAL MENOS INICIO= 401218-40119c = 7C

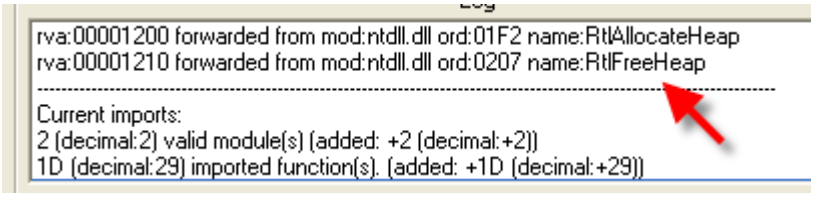
Pongamos estos valores en el IMP REC a ver que pasa, veamos que paso con las dos entradas que estaban mezcladas de la ntdll con las de kernel32.



Vemos la parte que dice NO que corresponde a las entradas basura, y abajo vemos que solo tiene entradas para kernel32, si miramos las entradas raras que correspondian a 401200 y 401210, vemos que

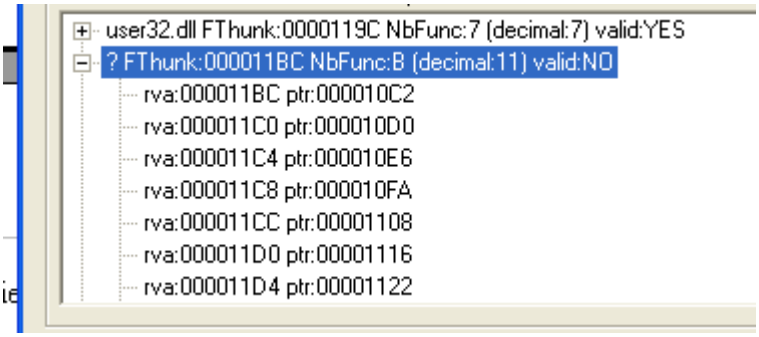


La reemplazo por las similares correspondientes a kernel32.dll y nos dijo algo de esto?

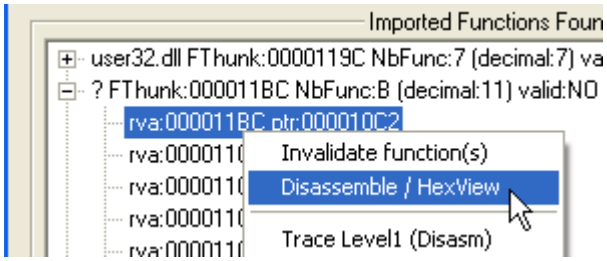


Vemos que si que nos aviso en el log que esas entradas son similares a las de kernel32.dll y seguro cambiadas por el packer para molestar y complicar las cosas.

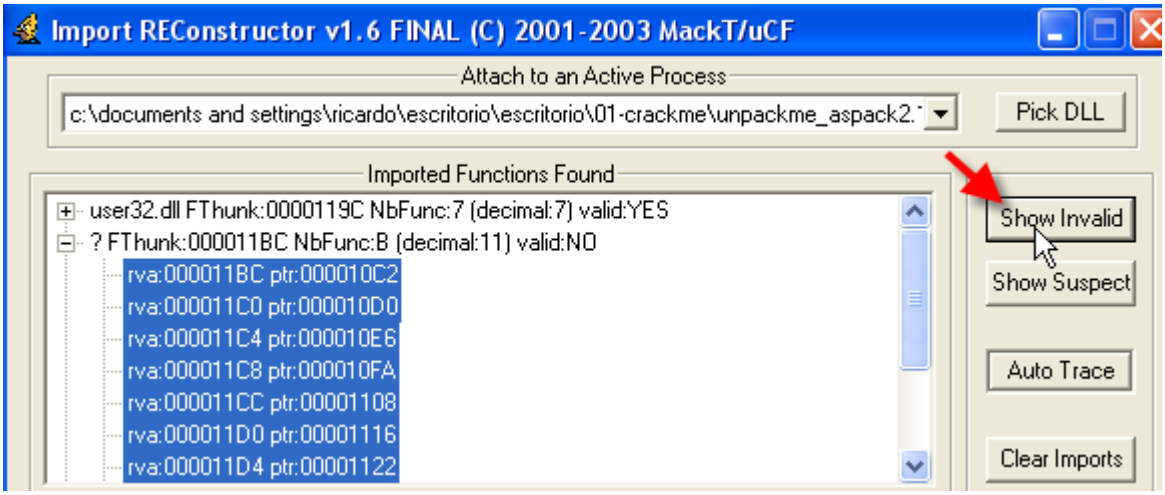
Bueno entonces solo tenemos que quitar la basura de en medio ya que verificamos que esas entradas que dicen NO, son basura, para verificarlo aquí vayamos a una de ellas.



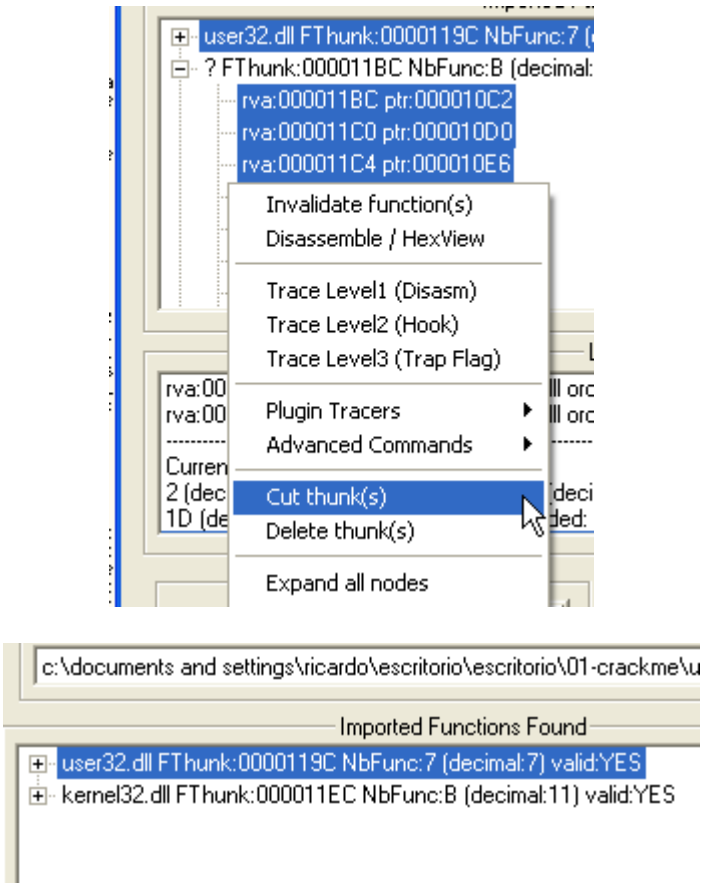
Marco la primera y hago click derecho- DISASSEMBLE-HEX VIEW



Alli vemos que no cae en ningun lugar que exista, asi que marcamos todas las entradas basura.

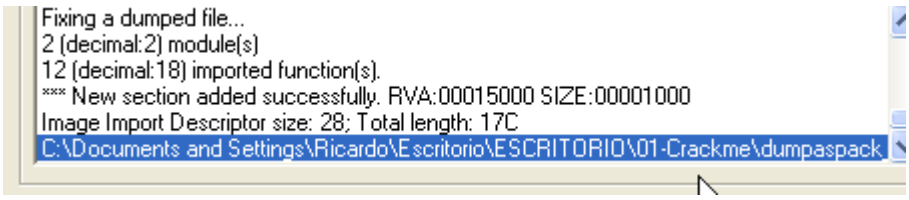


Apretando SHOW INVALID y alli las tenemos a todas marcadas ahora hacemos click derecho CUT THUNKS.

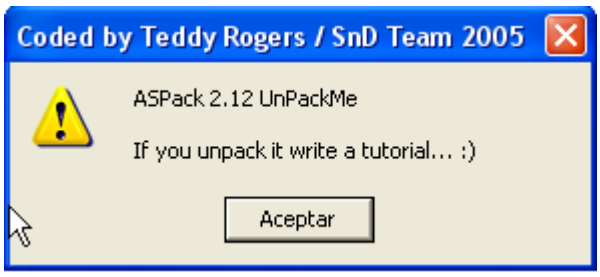


Con lo cual las anulamos a todas esas entradas para que al arrancar el sistema no de error tratando de arrancar apis inexistentes.

Ahora si, apreto FIX DUMP ya que tengo todas las entradas marcadas como YES o sea validas.



y me crea el dumpaspack_.exe el archivo que supuestamente ya estaria reparado, veamos ejecutemoslo.



Y si funciona perfectamente por lo cual terminamos con el segundo packer para ir aumentando la

dificultad progresivamente y muy levemente.

Hasta la parte 36 con otro packer
Ricardo Narvaja.