

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.

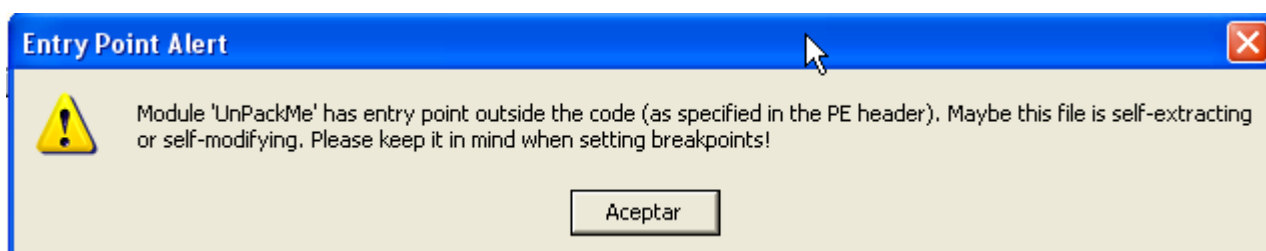
Coloco para practicar la dll del OLLYDUMP en la carpeta de plugins ya que lo dumpeare con el mismo.

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

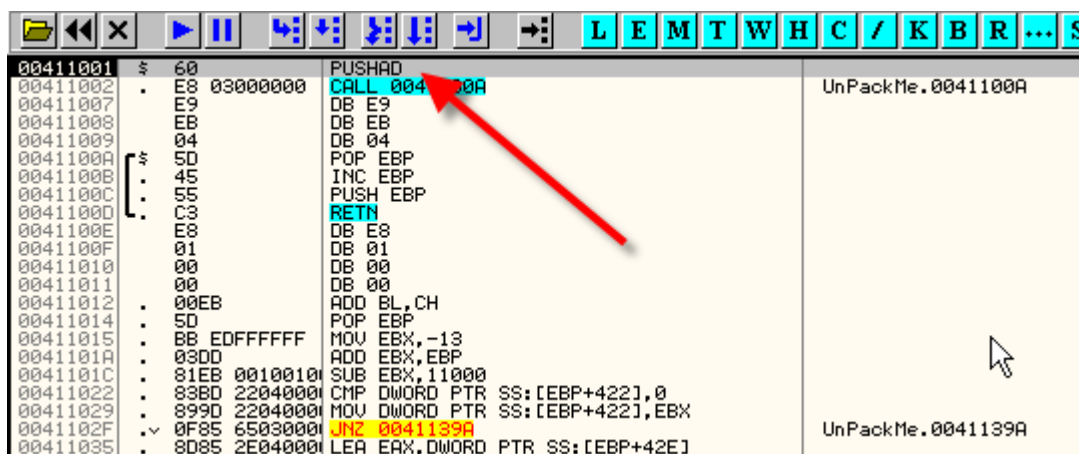
user y pass:hola

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

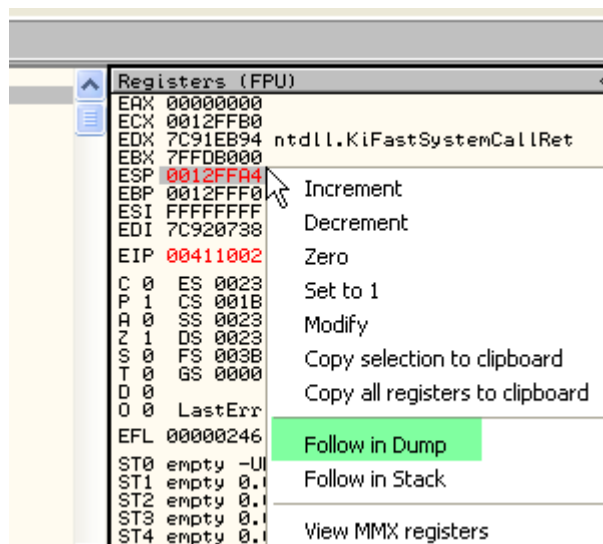
Abremos el OLLYDBG protegido con los plugins para ocultarlo, y lleguemos al OEP con el metodo el PUSHAD.



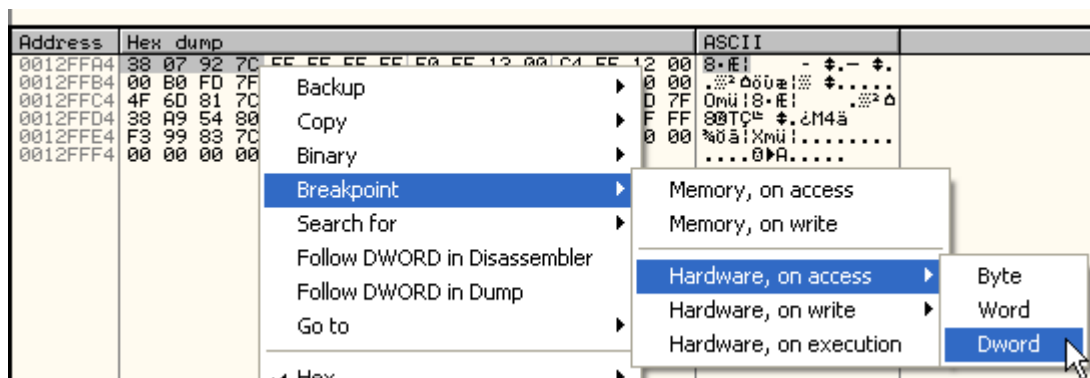
Vemos que aquí nos avisa que el entry point esta fuera de la seccion code como es lo usual, en la mayoría de los packers.



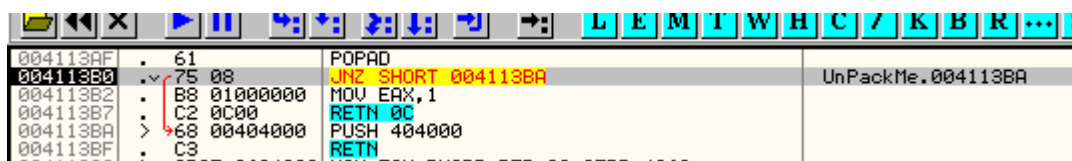
Ahi vemos el PUSHAD inicial al cual pasamos con f7 y luego hacemos.



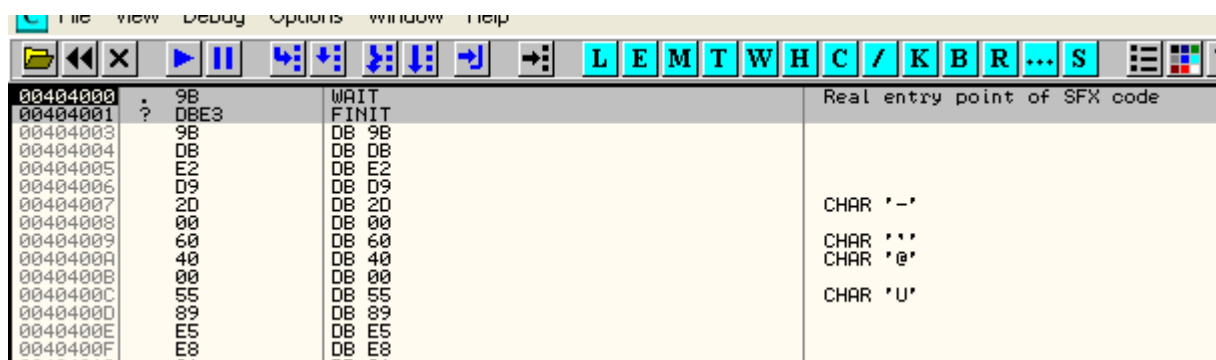
ESP-FOLLOW IN DUMP para colocar un Hardware Breakpoint on access en los valores de los registros que guardo con el PUSHAD en el DUMP.



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 código, le quito el análisis.

The screenshot shows the assembly window of OllyDbg. The 'Analysis' column is empty, indicating that the automatic analysis has been disabled. The assembly code is as follows:

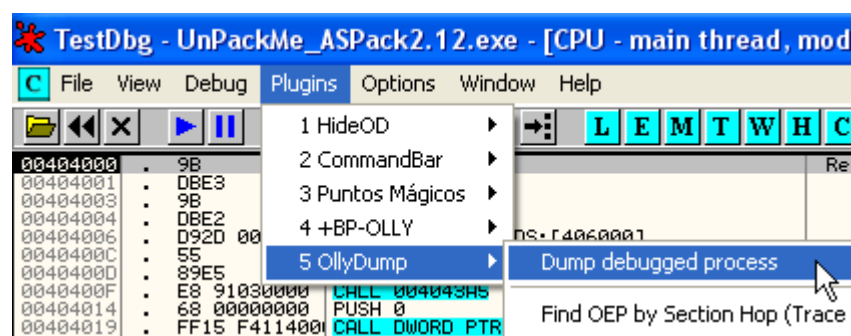
Address	Disassembly	Comment
00404000	9B WAIT	Real entry point of SFX code
00404001	DBE3 FINIT	
00404003	9B WAIT	
00404004	DBE2 FCLEX	
00404006	D92D 00604000 FLDCW WORD PTR DS:[406000]	
0040400C	55 PUSH EBP	
0040400D	89E5 MOV EBP,ESP	
0040400F	E8 91030000 CALL 004043A5	UnPackMe.004043A5
00404014	68 00000000 PUSH 0	
00404019	FF15 F4114000 CALL DWORD PTR DS:[4011F4]	kernel32.GetModuleHandleA
0040401F	A3 07F04000 MOV DWORD PTR DS:[40F007],EAX	
00404024	60 PUSHAD	
00404025	8925 0BF04000 MOV DWORD PTR DS:[40F00B],ESP	
0040402B	E9 30000000 JMP 00404060	UnPackMe.00404060
00404030	8B25 0BF04000 MOV ESP,DWORD PTR DS:[40F00B]	
00404036	61 POPAD	
00404037	E8 A9080000 CALL 004048E5	UnPackMe.004048E5
0040403C	E8 FD030000 CALL 0040443E	UnPackMe.0040443E

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

The screenshot shows the assembly window of OllyDbg with analysis enabled. The 'Analysis' column now contains comments for the code. The assembly code is as follows:

Address	Disassembly	Comment
00404000	9B WAIT	Real entry point of SFX code
00404001	DBE3 FINIT	
00404003	9B WAIT	
00404004	DBE2 FCLEX	
00404006	D92D 00604000 FLDCW WORD PTR DS:[406000]	
0040400C	55 PUSH EBP	
0040400D	89E5 MOV EBP,ESP	
0040400F	E8 91030000 CALL 004043A5	UnPackMe.004043A5
00404014	68 00000000 PUSH 0	
00404019	FF15 F4114000 CALL DWORD PTR DS:[4011F4]	pModule = NULL GetModuleHandleA
0040401F	A3 07F04000 MOV DWORD PTR DS:[40F007],EAX	
00404024	60 PUSHAD	
00404025	8925 0BF04000 MOV DWORD PTR DS:[40F00B],ESP	
0040402B	E9 30000000 JMP 00404060	UnPackMe.00404060
00404030	8B25 0BF04000 MOV ESP,DWORD PTR DS:[40F00B]	
00404036	61 POPAD	
00404037	E8 A9080000 CALL 004048E5	UnPackMe.004048E5
0040403C	E8 FD030000 CALL 0040443E	UnPackMe.0040443E
00404041	89EC MOV ESP,EBP	
00404043	5D POP EBP	
00404044	FF35 D4F14000 PUSH DWORD PTR DS:[40F1D4]	
0040404A	FF15 EC114000 CALL DWORD PTR DS:[4011EC]	ExitCode = 0 ExitProcess
00404050	9B WAIT	
00404051	DBE2 FCLEX	
00404053	D92D 00604000 FLDCW WORD PTR DS:[406000]	

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



OllyDump - UnPackMe_ASPack2.12.exe

Start Address: Size:

Entry Point: -> Modify:

Base of Code: Base of Data:

☒ Fix Raw Size & Offset of Dump Image

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.idata	00001000	00001000	00001000	00001000	C0000040
.rsrc	00002000	00002000	00002000	00002000	C0000040
.text	00002000	00004000	00002000	00004000	C0000040
.data	00001000	00006000	00001000	00006000	C0000040
.bss	00009000	00007000	00009000	00007000	C0000040
IMPOR...	00001000	00010000	00001000	00010000	C0000040
.teddy	00003000	00011000	00003000	00011000	C0000040
.adata	00001000	00014000	00001000	00014000	C0000040

☒ Rebuild Import

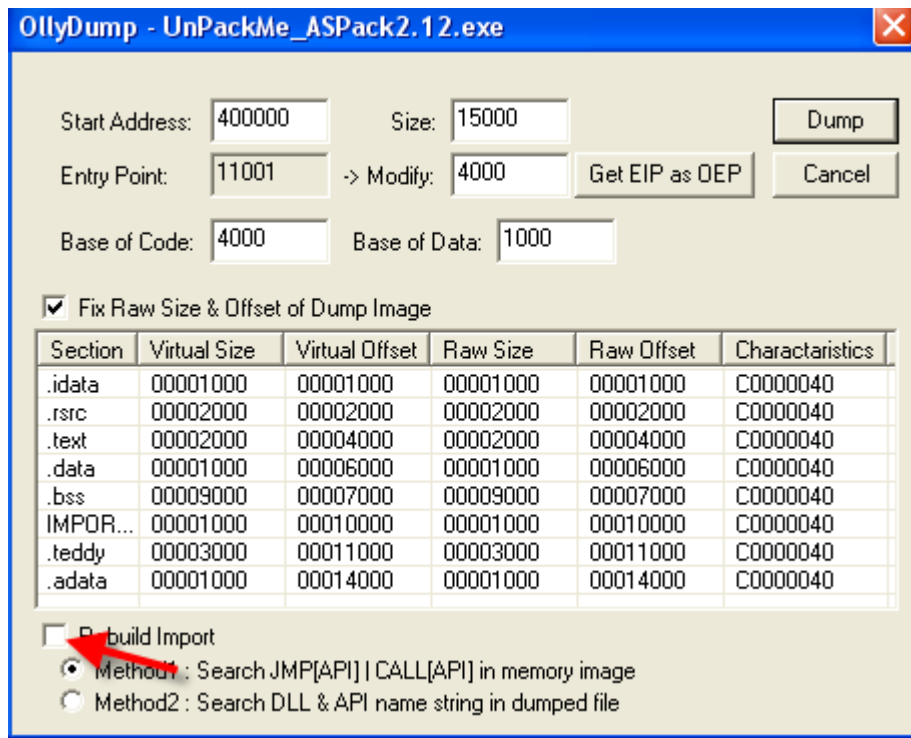
☒ Method1 : Search JMP[API] | CALL[API] in memory image

☐ Method2 : Search DLL & API name string in dumped file

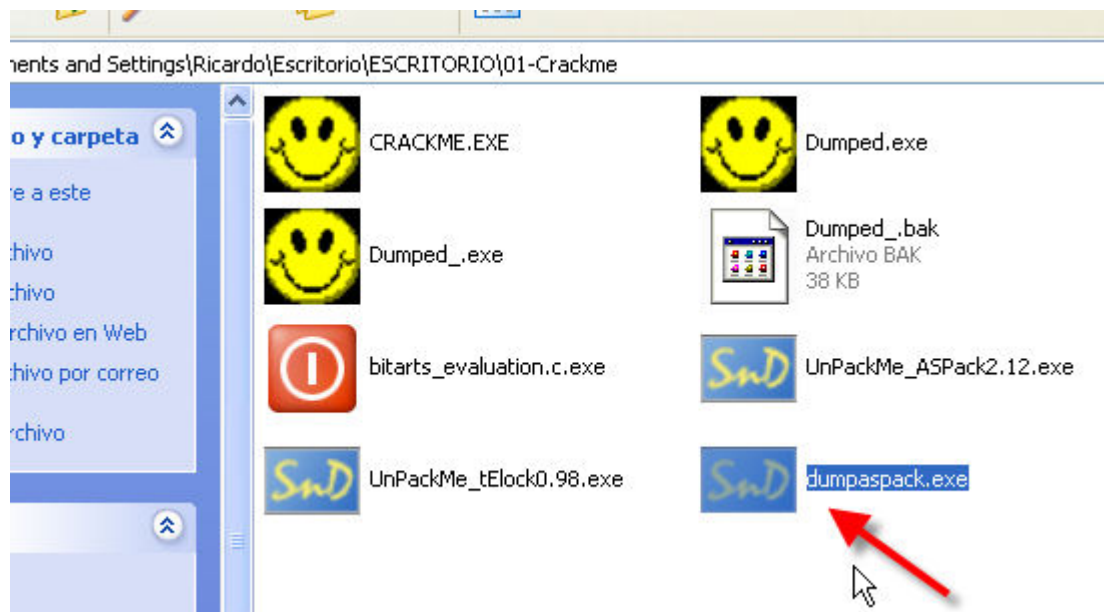
Nos aparece la ventana del plugin, en la cual ya vemos las cositas que podemos modificar, ya ahí podemos arreglar lo de la base of code, sin tener que luego cambiarlo en el header, en la ventana vemos base of code 4000, pero si recordamos este aspack corria en esta seccion que no es la primera, por lo 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.

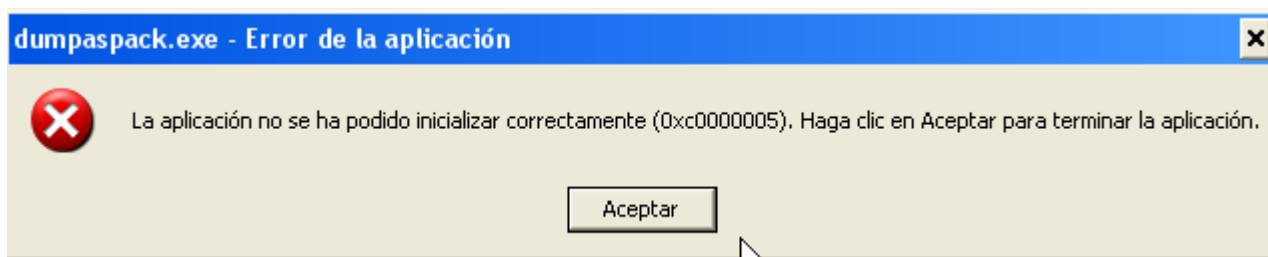
Nosotros le quitaremos la tilde en REBUILD IMPORT ya que lo haremos con el IMP REC que es mas confiable.



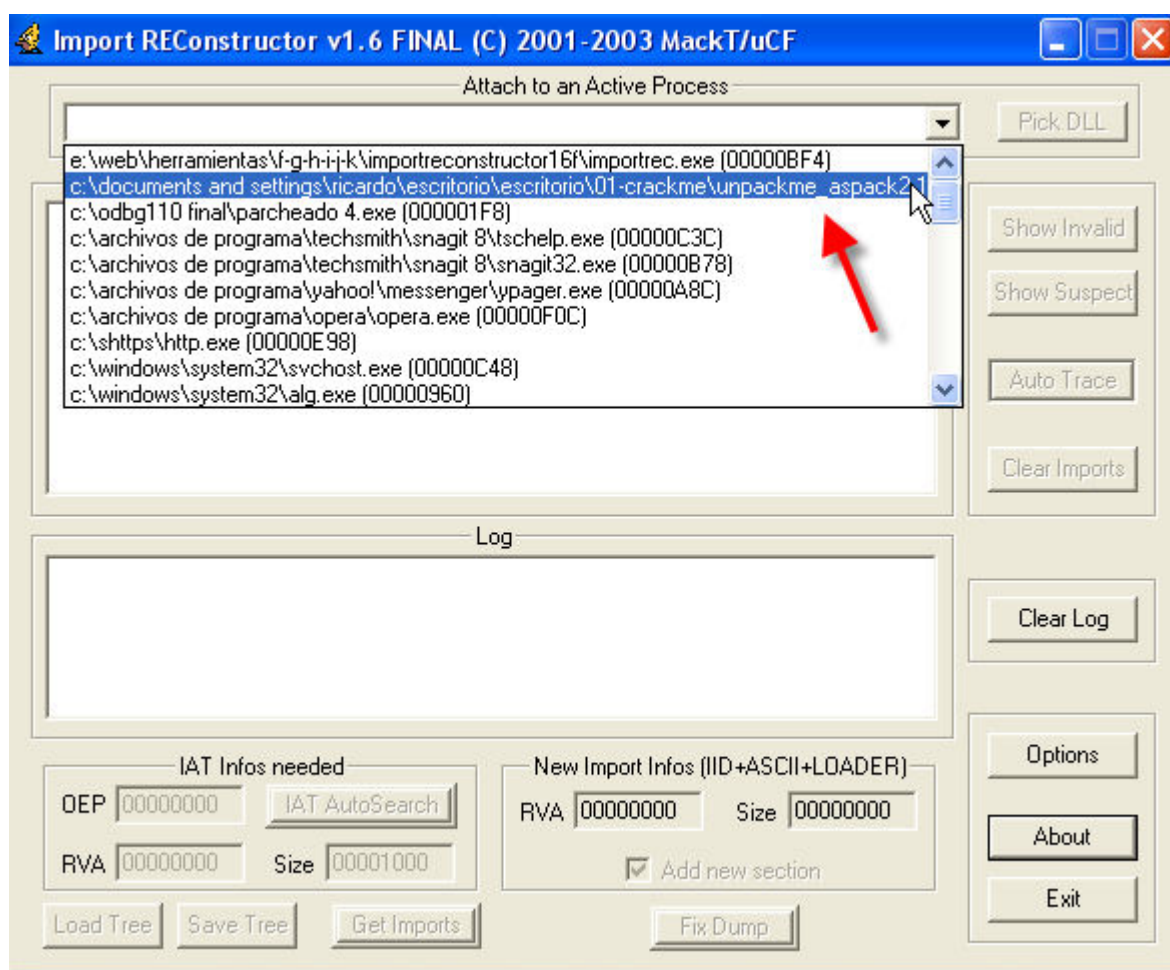
Bueno ahora podemos dumper a ver que tal nos va.



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



Bueno sin cerrar el archivo empaçado 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.

Address	Disassembly	Comment
00404000	9B WAIT	Real entry point of SFX code
00404001	DBE3 FINIT	
00404003	9B WAIT	
00404004	DBE2 FCLEX	
00404006	D92D 00604000 FLDCW WORD PTR DS:[406000]	
0040400C	55 PUSH EBP	
0040400D	89E5 MOV EBP,ESP	
0040400F	E8 91030000 CALL 004043A5	UnPackMe.004043A5
00404014	68 00000000 PUSH 0	pModule = NULL
00404019	FF15 F4114000 CALL DWORD PTR DS:[4011F4]	GetModuleHandleA
0040401F	A3 07F04000 MOV DWORD PTR DS:[40F007],EAX	
00404024	60 PUSHAD	

Si marco dicha linea y hago click derecho-FOLLOW

Address	Disassembly	Comment
0040400F	E8 91030000 CALL 004043A5	UnPackMe.004043A5
00404014	68 00000000 PUSH 0	pModule = NULL
00404019	FF15 F4114000 CALL DWORD PTR DS:[4011F4]	GetModuleHandleA
0040401F	A3 07F04000 MOV DWORD PTR DS:[40F007],EAX	
00404024	60 PUSHAD	
00404025	8925 0BF04000 MOV DWORD PTR DS:[40F00B],ESP	
0040402B	E9 30000000 JMP 00404060	
00404030	8B25 0BF04000 MOV ESP,DWORD PTR DS:[40F00B]	
00404036	61 POPAD	
00404037	E8 A9080000 CALL 004048E5	
0040403C	E8 FD030000 CALL 0040443E	
00404041	89EC MOV ESP,EBP	
00404043	5D POP EBP	
00404044	FF35 D4F14000 PUSH DWORD PTR DS:[40F1D4]	
0040404A	FF15 EC114000 CALL DWORD PTR DS:[4011EC]	
00404050	9B WAIT	
00404051	DBE2 FCLEX	
00404053	D92D 00604000 FLDCW WORD PTR DS:[406000]	
00404059	C3 RETN	
0040405A	DB 00	
0040405B	DB 00	
0040405C	DB 00	
0040405D	DB 00	
0040405E	DB 00	

Backup
 Copy
 Binary
 Assemble
 Label
 Comment
 Breakpoint
 Hit trace
 Run trace
 Follow
 New origin here

Address	Disassembly	Comment
7C80B529	8BFF MOV EDI,EDI	
7C80B52B	55 PUSH EBP	
7C80B52C	8BEC MOV EBP,ESP	
7C80B52E	837D 08 00 CMP DWORD PTR SS:[EBP+8],0	
7C80B532	74 18 JE SHORT 7C80B54C	kernel32.7C80B54C
7C80B534	FF75 08 PUSH DWORD PTR SS:[EBP+8]	
7C80B537	E8 682D0000 CALL 7C80E2A4	kernel32.7C80E2A4
7C80B53C	85C0 TEST EAX,EAX	
7C80B53E	74 08 JE SHORT 7C80B548	kernel32.7C80B548
7C80B540	FF70 04 PUSH DWORD PTR DS:[EAX+4]	
7C80B543	E8 F4300000 CALL 7C80E63C	kernel32.GetModuleHandleW
7C80B548	5D POP EBP	
7C80B549	C2 0400 RETN 4	
7C80B54C	64:A1 18000000 MOV EAX,DWORD PTR FS:[18]	
7C80B552	8B40 30 MOV EAX,DWORD PTR DS:[EAX+30]	
7C80B555	8B40 08 MOV EAX,DWORD PTR DS:[EAX+8]	
7C80B558	EB EE JMP SHORT 7C80B548	kernel32.7C80B548
7C80B55A	90 NOP	

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.


```

0040400C . 55      PUSH EBP
0040400D . 89E5    MOV EBP,ESP
0040400F . E8 91030000 CALL 004043A5
00404014 . 68 00000000 PUSH 0
00404019 . FF15 F4114000 CALL DWORD PTR DS:[4011F4]
0040401F . A3 07F04000 MOV DWORD PTR DS:[40F007],EAX
00404024 . 68 00000000 PUSH 0
00404025 . 9925 0BF04000 MOV DWORD PTR DS:[40F00D],ESP
0040402B . E9 30000000 JMP 00404060
00404030 . 8B25 0BF04000 MOV ESP,DWORD PTR DS:[40F00D]
00404036 . 61      POPAD
00404037 . E8 A9080000 CALL 004048E5
0040403C . E9 FD030000 CALL 0040443E
00404041 . 89EC    MOV ESP,EBP
00404043 . 5D      POP EBP
00404044 . FF35 D4F14000 PUSH DWORD PTR DS:[40F1D4]
0040404A . FF15 EC114000 CALL DWORD PTR DS:[4011EC]
00404050 . 9B      WAIT
00404051 . DBE2    FCLEX
00404053 . D92D 00604000 FLDQW WORD PTR DS:[406000]
00404059 . C3      RETN
0040405A . 00      DB 00
0040405B . 00      DB 00
0040405C . 00      DB 00
0040405D . 00      DB 00
0040405E . 00      DB 00
0040405F . 00      DB 00
00404060 . 68 00000000 PUSH 0
00404065 . B9 4A604000 MOV EAX,40604A
0040406A . 89C2    MOV EDI,EAX
0040406C . 52      PUSH EDI
0040406D . B8 22604000 MOV EAX,406022
00404072 . 89C6    MOV ESI,EAX
00404074 . 56      PUSH ESI
00404075 . E9 1B0F0000 CALL 00404F55
0040407A . 89C7    MOV EDI,ESI
0040407C . 57      PUSH EDI
0040407D . B8 00000000 MOV EAX,00

```

DS:[004011F4]=7C80B529 (kernel32.GetModuleHandleA)

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.

```

00410000 .- FF25 EC114000 JMP DWORD PTR DS:[4011EC] kernel32.ExitProcess
00410006 .- FF25 F4114000 JMP DWORD PTR DS:[4011F4] kernel32.GetModuleHandleA
0041000C .- FF25 9C114000 JMP DWORD PTR DS:[40119C] user32.CallNextHookEx
00410012 .- FF25 F0114000 JMP DWORD PTR DS:[4011F0] kernel32.GetCurrentThreadId
00410018 .- FF25 A0114000 JMP DWORD PTR DS:[4011A0] user32.GetDesktopWindow
0041001E .- FF25 A4114000 JMP DWORD PTR DS:[4011A4] user32.GetWindowRect
00410024 .- FF25 00124000 JMP DWORD PTR DS:[401200] ntdll.RtlAllocateHeap
0041002A .- FF25 04124000 JMP DWORD PTR DS:[401204] kernel32.HeapCompact
00410030 .- FF25 08124000 JMP DWORD PTR DS:[401208] kernel32.HeapCreate
00410036 .- FF25 0C124000 JMP DWORD PTR DS:[40120C] kernel32.HeapDestroy
0041003C .- FF25 10124000 JMP DWORD PTR DS:[401210] ntdll.RtlFreeHeap
00410042 .- FF25 A8114000 JMP DWORD PTR DS:[4011A8] user32.MessageBoxA
00410048 .- FF25 AC114000 JMP DWORD PTR DS:[4011AC] user32.SetWindowsHookExA
0041004E .- FF25 B0114000 JMP DWORD PTR DS:[4011B0] user32.SystemParametersInfoA
00410054 .- FF25 B4114000 JMP DWORD PTR DS:[4011B4] user32.UnhookWindowsHookEx
0041005A .- FF25 14124000 JMP DWORD PTR DS:[401214] kernel32.lstrcatA
00410060 .- FF25 F8114000 JMP DWORD PTR DS:[4011F8] kernel32.GlobalAlloc
00410066 .- FF25 FC114000 JMP DWORD PTR DS:[4011FC] kernel32.GlobalFree
0041006C . 00      DB 00
0041006D . 00      DB 00
0041006E . 00      DB 00

```

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.

Address	Hex dump	ASCII
004011F4	29 B5 80 7C 2D FF 80 7C 1F FE 80 7C 04 05 92 7C	!AQ!-C!C!E!E!
00401204	AE 30 82 7C 29 81 7C 10 11 81 7C 3D 04 92 7C	<0e!))u!>u!>E!
00401214	B9 8F 83 7C 00 00 00 00 00 00 00 00 00 00 00	!Aa!.....
00401224	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
00401244	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

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	"4..04..>4..L4..	
004011E4	58 11 00 00 00 00 00 00 A2 CA 81 7C 37 97 80 7C	X4.....6#u!7uÇ!	
004011F4	29 B5 80 7C 2D FF 80 7C 2F FE 80 7C D4 05 92 7C)4Ç!- Ç!÷Ç!É#E!	
00401204	AE 30 82 7C 29 29 81 7C 10 11 81 7C 3D 04 92 7C	<0é!))û!÷4û!÷#E!	
00401214	B9 8F 83 7C 00 00 00 00 00 00 00 00 00 00 00 00	!Aä!.....	
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	
7C803000	00005000	kernel32	.data	data	Imag	R	RWE	
7C804000	00073000	kernel32	.rsrc	resources	Imag	R	RWE	
7C805000	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	"4..04..>4..L4..	
004011E4	58 11 00 00 00 00 00 00 A2 CA 81 7C 37 97 80 7C	X4.....6#u!7uÇ!	
004011F4	29 B5 80 7C 2D FF 80 7C 2F FE 80 7C D4 05 92 7C)4Ç!- Ç!÷Ç!É#E!	
00401204	AE 30 82 7C 29 29 81 7C 10 11 81 7C 3D 04 92 7C	<0é!))û!÷4û!÷#E!	
00401214	B9 8F 83 7C 00 00 00 00 00 00 00 00 00 00 00 00	!Aä!.....	
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	%>.....0#0wY00w	
004011A4	04 B6 01 77 EA 04 05 77 E9 11 03 77 92 0A D2 77	E4Bw0!w0!EwE.Ew	
004011B4	F3 0D 02 77 00 00 00 00 C2 10 00 00 00 10 00 00	%,Ew.....T...S>..	
004011C4	E6 10 00 00 FA 10 00 00 08 11 00 00 16 11 00 00	p>...>...0!...>..	
004011D4	22 11 00 00 30 11 00 00 3E 11 00 00 4C 11 00 00	"4..04..>4..L4..	
004011E4	58 11 00 00 00 00 00 00 A2 CA 81 7C 37 97 80 7C	X4.....6#u!7uÇ!	
004011F4	29 B5 80 7C 2D FF 80 7C 2F FE 80 7C D4 05 92 7C)4Ç!- Ç!÷Ç!É#E!	
00401204	AE 30 82 7C 29 29 81 7C 10 11 81 7C 3D 04 92 7C	<0é!))û!÷4û!÷#E!	
00401214	B9 8F 83 7C 00 00 00 00 00 00 00 00 00 00 00 00	!Aä!.....	
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	

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.

File view Debug Plugins Options window help									
L E M T W H C / K B R									
Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as	
00010000	00001000				Priv	RW	RW		
00020000	00010000				Priv	RW	RW		
00127000	00010000				Priv	RW	Guar		
00128000	00003000				Priv	RW	Guar		
00130000	00002000				Priv	RWE	RWE		
00140000	00003000				Map	R	R		
00150000	00003000				Priv	RW	RW		
00250000	00006000				Priv	RW	RW		
00260000	00003000				Map	RW	RW		
00270000	00016000				Map	R	R	\Device\Harddis	
00290000	00030000				Map	R	R	\Device\Harddis	
002D0000	00041000				Map	R	R	\Device\Harddis	
00320000	00006000				Map	R	R	\Device\Harddis	
00330000	00041000				Map	R	R		
00380000	00001000				Priv	RWE	RWE		
00390000	00001000				Priv	RWE	RWE		
003A0000	00001000				Priv	RW	RW		
003B0000	00001000				Priv	RW	RW		
00400000	00001000	UnPackMe		PE header	Imag	R	RWE		
00401000	00001000	UnPackMe	.idata		Imag	R	RWE		
00402000	00002000	UnPackMe	.rsrc	resources	Imag	R	RWE		
00404000	00002000	UnPackMe	.text	code	Imag	R	RWE		
00406000	00001000	UnPackMe	.data	code,data	Imag	R	RWE		
00407000	00009000	UnPackMe	.bss	code	Imag	R	RWE		
00410000	00001000	UnPackMe	IMPORTS	code	Imag	R	RWE		
00411000	00003000	UnPackMe	.teddy	SFX, imports	Imag	R	RWE		
00414000	00001000	UnPackMe	.adata		Imag	R	RWE		
00420000	0000A000				Map	R E	R E		
004E0000	00002000				Map	R E	R E		
004F0000	00103000				Map	R	R		
00600000	0013B000				Map	R E	R E		

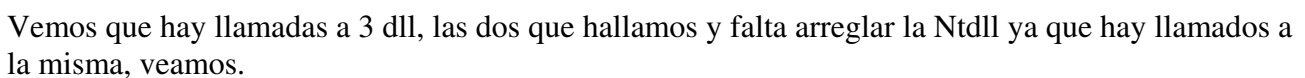
Asi que estas entradas ya que no van ni a una dll, ni a una seccion real, ya que podrian apuntar a alguna seccion creada por el packer, lo cual no es este caso, son basura metida para molestar ya veremos lo que hacemos con ellas, ahora sigamos subiendo.

Hex dump																ASCII													
65	65	00	00	00	00	6C	73	74	72	63	61	74	41	00	00	e	e
55	53	45	52	33	32	2E	44	4C	4C	00	48	45	52	4E	45	U	S	E	R	3	2
4C	33	32	2E	44	4C	4C	00	3C	10	00	00	4E	10	00	00	L	3	2
62	10	00	00	00	00	00	80	10	00	00	94	10	00	00	00	b
AC	10	00	00	00	00	00	00	03	EB	D1	77	ED	E5	D1	77	%
04	B6	D1	77	EA	04	D5	77	E9	11	D3	77	92	0A	D2	77	E	A	0	4
F3	00	D2	77	00	00	00	00	C2	10	00	00	D0	10	00	00	%
E6	10	00	00	FA	10	00	00	08	11	00	00	16	11	00	00	p
22	11	00	00	30	11	00	00	0E	11	00	00	4C	11	00	00	"
58	11	00	00	00	00	00	00	00	00	00	00	00	00	00	00	"
29	B5	80	7C	2D	FF	80	7C	2F	FE	80	7C	D4	05	92	7C)	A	Ç	-	Ç	!	!	!	!	!	!	!	!	!
AE	30	82	7C	29	29	81	7C	10	11	81	7C	3D	04	92	7C	<	0	e	!	!	!	!	!	!	!	!	!	!	!
B9	8F	83	7C	00	00	00	00	00	00	00	00	00	00	00	00	!	A	a	!
AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA

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

004E0000	00002000				Map	R E	R E	
004F0000	00103000				Map	R	R	
00600000	0013B000				Map	R E	R E	
77D10000	00001000	user32		PE header	Imag	R	RWE	
77D11000	0005F000	user32	.text	code,import	Imag	R	RWE	
77D70000	00002000	user32	.data	data	Imag	R	RWE	
77D72000	00002B000	user32	.rsrc	resources	Imag	R	RWE	
77D90000	00003000	user32	.reloc	relocations	Imag	R	RWE	
77EF0000	00001000	GDI32		PE header	Imag	R	RWE	
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	00002000	kernel32	.text	code,import	Imag	R	RWE	
7C803000	00005000	kernel32	.data	data	Imag	R	RWE	
7C808000	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	
7FFD0000	00001000			data block	Priv	RW	RW	
7FFDE000	00001000				Priv	RW	RW	
7FFE0000	00001000				Priv	R	R	

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.



Address	Disassembly	Comment
004045C7	B8 10F04000 MOV EAX,40F010	
004045CC	FF30 PUSH DWORD PTR DS:[EAX]	
004045CE	FF15 00124000 CALL DWORD PTR DS:[401200]	ntdll.RtlAllocateHeap
004045D4	89C2 MOV EDX,EAX	
004045D6	5E POP ESI	
004045D7	8916 MOV DWORD PTR DS:[ESI],EDX	
004045D9	8345 FC CMP DWORD PTR DS:[EBP-4],0	

[illegible]

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.

00404666	B8 10F04000	MOV EAX,40F010	
0040466B	FF30	PUSH DWORD PTR DS:[EAX]	
0040466D	FF15 10124000	CALL DWORD PTR DS:[401210]	ntdll.RtlFreeHeap
00404673	89C2	MOV EDX,EAX	
00404675	5E	POP ESI	
00404676	8916	MOV DWORD PTR DS:[ESI],EDX	

Address	Hex dump	ASCII
004011E0	4C 11 00 00 58 11 00 00 00 00 00 00 A2 CA 81 7C	L<...X<...6u!
004011F0	37 97 80 7C 29 B5 80 7C 2D FF 80 7C 2F FE 80 7C	7uÇ!}AÇ!- Ç! /Ç!
00401200	04 05 92 7C AE 30 82 7C 29 29 81 7C 10 11 81 7C	EÇE!<0e!))ü!>ü!
00401210	3D 04 92 7C B9 8F 83 7C 00 00 00 00 00 00 00	=ÇE! Aç!.....
00401220	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401230	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

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

Address	Hex dump	ASCII
0040118C	80 10 00 00 94 10 00 00 AC 10 00 00 00 00 00 00	Ç>..ö>..%>.....
0040119C	03 EB D1 77 ED E5 D1 77 04 B6 D1 77 EA 04 D5 77	0u0WYö0WEA0W0+~w
004011AC	E9 11 03 77 92 0A D2 77 F3 0D D2 77 00 00 00 00	U<EwE.Ew%>.Ew.....
004011BC	C2 10 00 00 D0 10 00 00 E6 10 00 00 FA 10 00 00	T>..š>..p>..>.....
004011CC	08 11 00 00 16 11 00 00 22 11 00 00 30 11 00 00	0<...<...>...0<...
004011DC	3E 11 00 00 4C 11 00 00 58 11 00 00 00 00 00 00	><...L<...X<.....
004011EC	A2 CA 81 7C 37 97 80 7C 29 B5 80 7C 2D FF 80 7C	6#ü!7uÇ!}AÇ!- Ç!
004011FC	2F FE 80 7C 04 05 92 7C AE 30 82 7C 29 29 81 7C	/Ç!EÇE!<0e!))ü!
00401200	10 11 81 7C 3D 04 92 7C B9 8F 83 7C 00 00 00 00	>ü!>ÇE! Aç!.....
00401210	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401220	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401230	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

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

Address	Hex dump	Instruction	Module Name
00410000	FF25 EC114000	JMP DWORD PTR DS:[4011EC]	kernel32.ExitProcess
00410006	FF25 F4114000	JMP DWORD PTR DS:[4011F4]	kernel32.GetModuleHandleA
0041000C	FF25 9C114000	JMP DWORD PTR DS:[40119C]	user32.CallNextHookEx
00410012	FF25 F0114000	JMP DWORD PTR DS:[4011F0]	kernel32.GetCurrentThreadId
00410018	FF25 A0114000	JMP DWORD PTR DS:[4011A0]	user32.GetDesktopWindow
0041001E	FF25 A4114000	JMP DWORD PTR DS:[4011A4]	user32.GetWindowRect
00410024	FF25 00124000	JMP DWORD PTR DS:[401200]	ntdll.RtlAllocateHeap
0041002A	FF25 04124000	JMP DWORD PTR DS:[401204]	kernel32.HeapCompact
00410030	FF25 08124000	JMP DWORD PTR DS:[401208]	kernel32.HeapCreate
00410036	FF25 0C124000	JMP DWORD PTR DS:[40120C]	kernel32.HeapDestroy
0041003C	FF25 10124000	JMP DWORD PTR DS:[401210]	ntdll.RtlFreeHeap
00410042	FF25 A8114000	JMP DWORD PTR DS:[4011A8]	user32.MessageBoxA
00410048	FF25 AC114000	JMP DWORD PTR DS:[4011AC]	user32.SetWindowsHookExA
0041004E	FF25 B0114000	JMP DWORD PTR DS:[4011B0]	user32.SystemParametersInfoA
00410054	FF25 B4114000	JMP DWORD PTR DS:[4011B4]	user32.UnhookWindowsHookEx
0041005A	FF25 14124000	JMP DWORD PTR DS:[401214]	kernel32.lstrcatA
00410060	FF25 F8114000	JMP DWORD PTR DS:[4011F8]	kernel32.GlobalAlloc
00410066	FF25 FC114000	JMP DWORD PTR DS:[4011FC]	kernel32.GlobalFree
0041006C	00 00 00 00	DB 00	

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

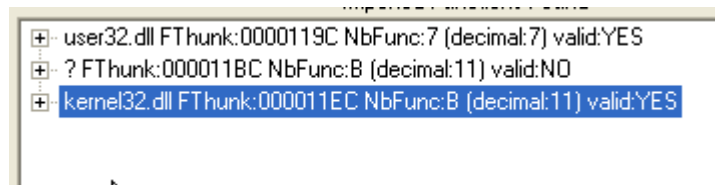
OEP=4000

RVA o INICIO DE LA IAT=119C

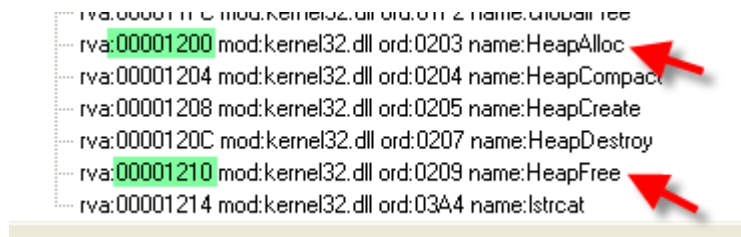
LARGO= FINAL MENOS INICIO= 401218-40119c = 7C

04012FC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
040130C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Command	401218-40119c	HEX: 7C - DEC: 124 - ASCII:

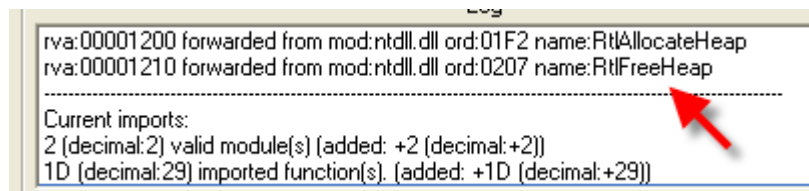
Pongamos estos valores en el IMP REC a ver que pasa, veamos que paso con las dos entradas que estaban mezcladas de la ntdll con las de kernel32.



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

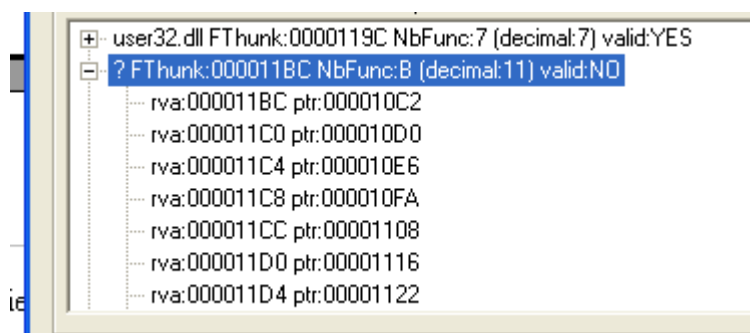


La reemplazo por las similares correspondientes a kernel32.dll y nos dijo algo de esto?

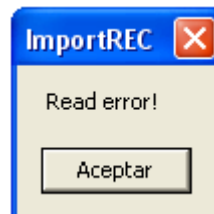
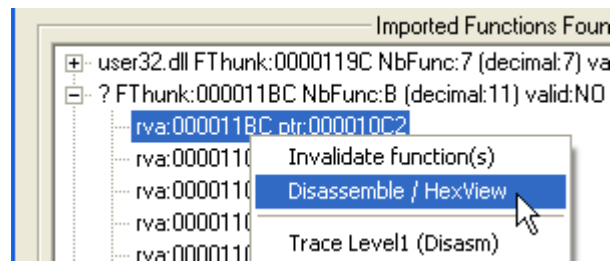


Vemos que si que nos aviso en el log que esas entradas son similares a las de kernel32.dll y seguro cambiadas por el packer para molestar y complicar las cosas.

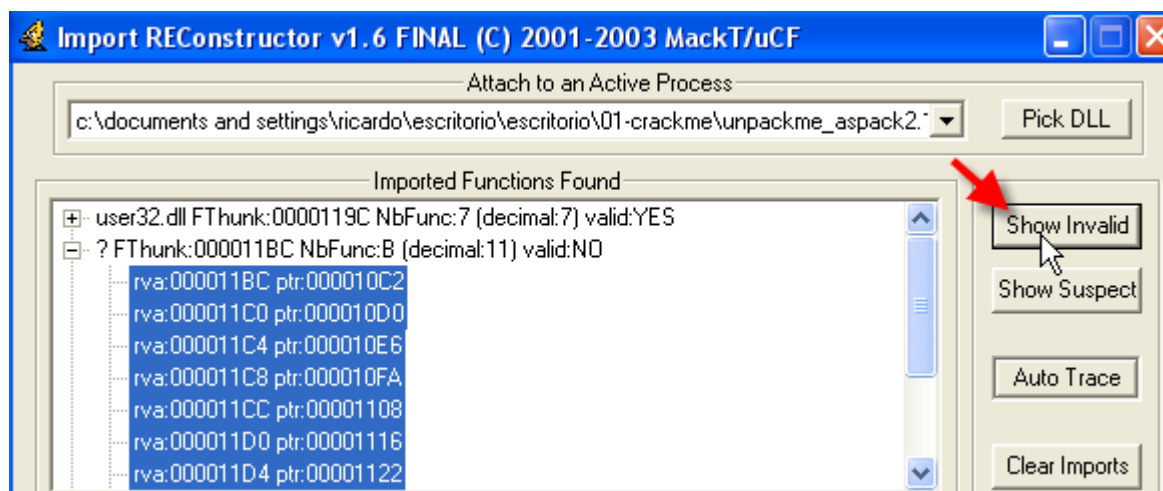
Bueno entonces solo tenemos que quitar la basura de en medio ya que verificamos que esas entradas que dicen NO, son basura, para verificarlo aquí vayamos a una de ellas.



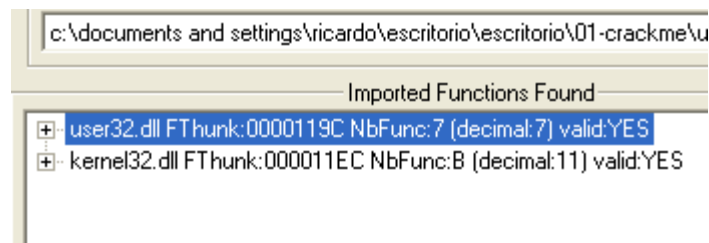
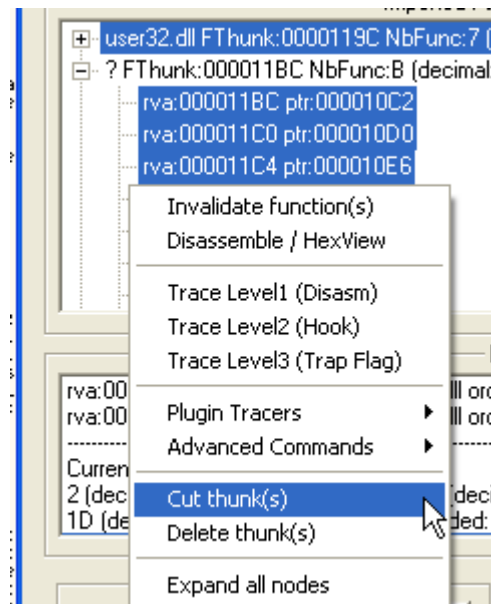
Marco la primera y hago click derecho- DISASSEMBLE-HEX VIEW



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

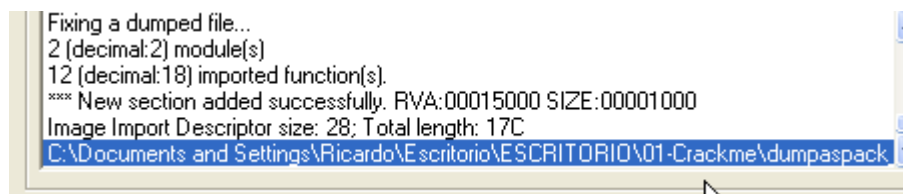


Apretando SHOW INVALID y alli las tenemos a todas marcadas ahora hacemos click derecho CUT THUNKS.

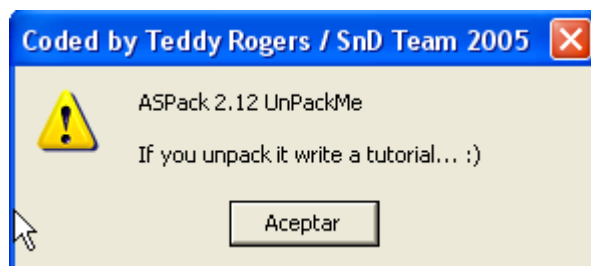


Con lo cual las anulamos a todas esas entradas para que al arrancar el sistema no de error tratando de arrancar apis inexistentes.

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.