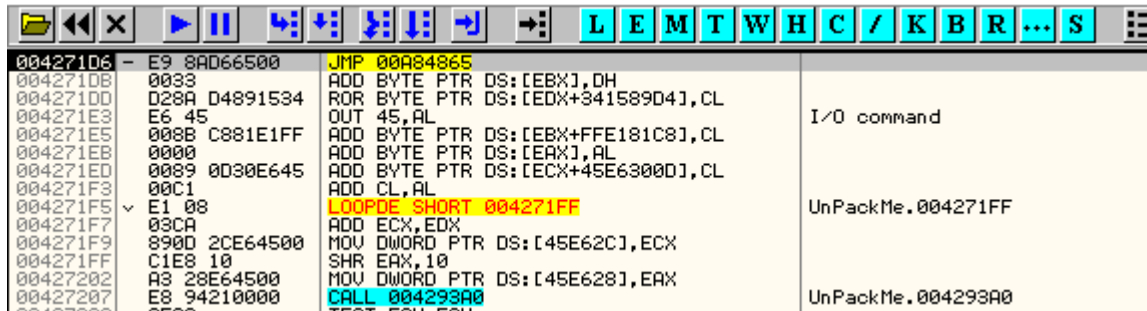


## INTRODUCCION AL CRACKING CON OLLYDBG PARTE 41

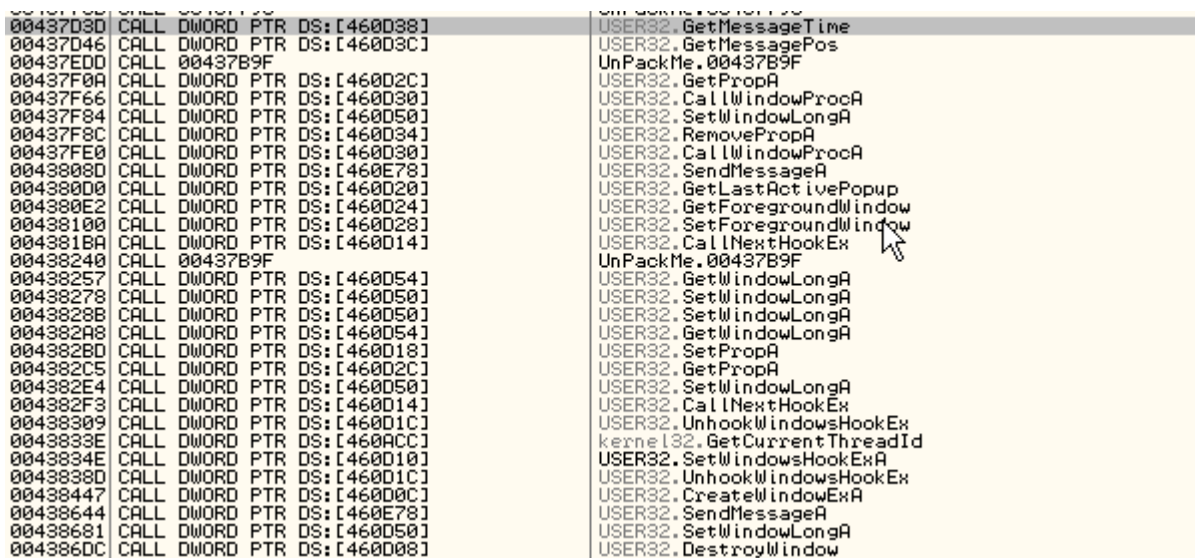
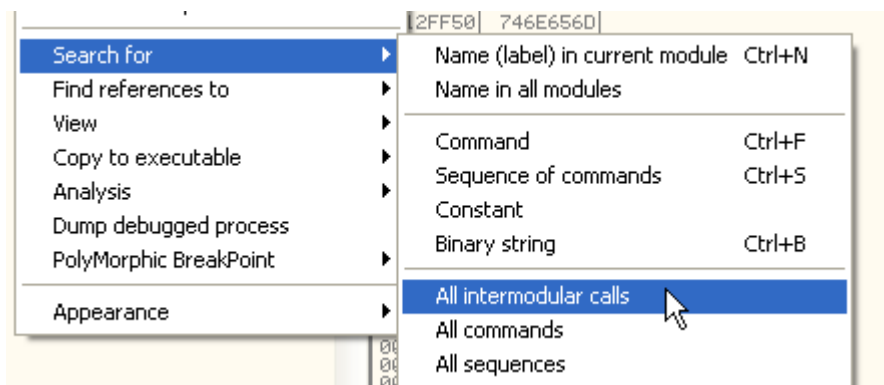
Continuaremos con el dsempacado del pelock que tenemos pendiente, ahora trabajaremos con la IAT para lo cual veremos si existe algun salto magico o similar.

Para ello lleguemos nuevamente hasta el oep.



```
004271D6 - E9 8AD66500 JMP 00A84865
004271D8 0033 ADD BYTE PTR DS:[EBX],DH
004271DD 028A D4891534 ROR BYTE PTR DS:[EDX+341589D4],CL
004271E3 E6 45 OUT 45,AL
004271E5 008B C881E1FF ADD BYTE PTR DS:[EBX+FFE181C8],CL
004271EB 0000 ADD BYTE PTR DS:[EAX],AL
004271ED 0089 0D30E645 ADD BYTE PTR DS:[ECX+45E6300D],CL
004271F3 00C1 ADD CL,AL
004271F5 E1 08 LOOPDE SHORT 004271FF
004271F7 03CA ADD ECX,EDX
004271F9 890D 2CE64500 MOV DWORD PTR DS:[45E62C],ECX
004271FF C1E8 10 SHR EAX,10
00427202 A3 28E64500 MOV DWORD PTR DS:[45E628],EAX
00427207 E8 94210000 CALL 004293A0
00427209 0000 TEST EAX,EAX
```

Bueno ahi llegamos al falso OEP, busquemos llamadas a las apis, miremos con click derecho SEARCH FOR- ALL INTERMODULAR CALLS.



```
00437D3D CALL DWORD PTR DS:[460D38] USER32.GetMessageTime
00437D46 CALL DWORD PTR DS:[460D3C] USER32.GetMessagePos
00437ED0 CALL 00437B9F UnPackMe.00437B9F
00437F0A CALL DWORD PTR DS:[460D2C] USER32.GetPropA
00437F66 CALL DWORD PTR DS:[460D30] USER32.CallWindowProcA
00437F84 CALL DWORD PTR DS:[460D50] USER32.SetWindowLongA
00437F8C CALL DWORD PTR DS:[460D34] USER32.RemovePropA
00437FE0 CALL DWORD PTR DS:[460D30] USER32.CallWindowProcA
00438080 CALL DWORD PTR DS:[460E78] USER32.SendMessageA
004380D0 CALL DWORD PTR DS:[460D20] USER32.GetLastActivePopup
004380E2 CALL DWORD PTR DS:[460D24] USER32.GetForegroundWindow
00438100 CALL DWORD PTR DS:[460D28] USER32.SetForegroundWindow
004381BA CALL DWORD PTR DS:[460D14] USER32.CallNextHookEx
00438240 CALL 00437B9F UnPackMe.00437B9F
00438257 CALL DWORD PTR DS:[460D54] USER32.SetWindowLongA
00438278 CALL DWORD PTR DS:[460D50] USER32.SetWindowLongA
00438288 CALL DWORD PTR DS:[460D50] USER32.SetWindowLongA
004382A8 CALL DWORD PTR DS:[460D54] USER32.SetWindowLongA
004382BD CALL DWORD PTR DS:[460D18] USER32.SetPropA
004382C5 CALL DWORD PTR DS:[460D2C] USER32.GetPropA
004382E4 CALL DWORD PTR DS:[460D50] USER32.SetWindowLongA
004382F3 CALL DWORD PTR DS:[460D14] USER32.CallNextHookEx
00438309 CALL DWORD PTR DS:[460D1C] USER32.UnhookWindowsHookEx
0043833E CALL DWORD PTR DS:[460ACC] kernel32.GetCurrentThreadId
0043834E CALL DWORD PTR DS:[460D10] USER32.SetWindowsHookExA
00438380 CALL DWORD PTR DS:[460D1C] USER32.UnhookWindowsHookEx
00438447 CALL DWORD PTR DS:[460D0C] USER32.CreateWindowExA
00438644 CALL DWORD PTR DS:[460E78] USER32.SendMessageA
00438681 CALL DWORD PTR DS:[460D50] USER32.SetWindowLongA
004386DC CALL DWORD PTR DS:[460D08] USER32.DestroyWindow
```

Alli vemos unas cuantas elijamos una y hagamos doble click en ella para verla en el listado.

00437D28	C2 0C00	RETN 0C	
00437D2B	56	PUSH ESI	
00437D2C	68 8DC24100	PUSH 41C28D	
00437D31	B9 70E24500	MOV ECX, 45E270	
00437D36	E8 A0E30000	CALL 004460DB	UnPackMe.004460DB
00437D3B	8BF0	MOV ESI, EAX	
00437D3D	FF15 380D4600	CALL DWORD PTR DS:[460D38]	USER32.GetMessageTime
00437D43	8946 44	MOV DWORD PTR DS:[ESI+44], EAX	
00437D46	FF15 3C0D4600	CALL DWORD PTR DS:[460D3C]	USER32.GetMessagePos
00437D4C	0FBFC8	MOVSX ECX, AX	
00437D4F	C1E8 10	SHR EAX, 10	
00437D52	0FBFC0	MOVSX EAX, AX	
00437D55	804F 40	CALL DWORD PTR DS:[ESI+40], ECX	

Esto quiere decir que el call a la api toma valores de 460d38 que es una entrada de la IAT, miremosla en el DUMP.

Address	Hex dump	ASCII
00460D38	08 C4 D1 77 94 BF D1 77	08C4D17794BFD177
00460D40	7D BC D1 77 1B C0 D1 77	7DBC D1771BC0D177
00460D48	28 8E D1 77 79 F6 D2 77	288E D17779F6D277
00460D50	0D D6 D1 77 5D 94 D1 77	0DD6 D1775D94D177
00460D58	C3 91 D2 77 E2 16 D2 77	C391 D277E216D277
00460D60	4F 02 D3 77 15 E5 D3 77	4F02 D37715E5D377
00460D68	3D 02 D3 77 AE 90 D2 77	3D02 D377AE90D277
00460D70	88 C1 D1 77 8E 1A D3 77	88C1 D1778E1AD377
00460D78	2F EA D1 77 12 0B D2 77	2FEA D177120BD277
00460D80	12 0B D2 77 44 D0 D3 77	120B D27744D0D377
00460D88	5F F7 D2 77 73 F9 D2 77	5FF7 D27773F9D277
00460D90	DC F6 D2 77 37 F4 D2 77	DCF6 D27737F4D277
00460D98	28 F7 D2 77 42 8C D1 77	28F7 D277428CD177
00460DA0	2E 8C D1 77 8B 14 D3 77	2E8C D1778B14D377
00460DA8	FE EC D3 77 83 F7 D4 77	FE EC D37783F7D477
00460DB0	DE F2 D2 77 DF 1A D3 77	DEF2 D277DF1AD377
00460DB8	F6 F0 D4 77 9C F3 D4 77	F6F0 D4779CF3D477
00460DC0	33 F2 D2 77 6C C9 D1 77	33F2 D2776CC9D177
00460DC8	F6 8B D1 77 B8 96 D1 77	F68B D177B896D177
00460DD0	0C 94 D1 77 61 C6 D3 77	0C94 D17761C6D377
00460DD8	81 55 D3 77 8A 82 D3 77	8155 D3778A82D377

Alli se ve la IAT y alli veo todas entradas correctas subamos a buscar el inicio de la IAT.

Address	Hex dump	ASCII
004607F8	9E BB EB 32 89 1A 41 0D	9EBBEB32891A410D
00460800	88 E2 6F D0 88 4D 6D 50	88E26FD0884D6D50
00460808	C0 BA BB E9 E9 D0 9F 06	C0BABBE9E9D09F06
00460810	1B B7 A8 38 F0 F6 28 3F	1BB7A838F0F6283F
00460818	F0 68 DA 77 18 76 DA 77	F068DA771876DA77
00460820	F4 EA DA 77 E7 EB DA 77	F4EADA77E7EBDA77
00460828	83 78 DA 77 0D 5B D2 70	8378DA770D5BD270
00460830	0D 15 C5 58 2E B0 C3 58	0D15C5582EB0C358
00460838	52 88 F5 31 D4 6A EF 77	5288F531D46AEF77
00460840	66 95 EF 77 89 6A EF 77	6695EF77896AEF77
00460848	F3 AD EF 77 ED D9 EF 77	F3ADEF77EDD9EF77
00460850	99 88 EF 77 C0 B5 EF 77	9988EF77C0B5EF77
00460858	2A 7D EF 77 B2 7C EF 77	2A7DEF77B27CEF77
00460860	77 53 F2 77 1E C9 F1 77	7753F2771EC9F177
00460868	0C BC EF 77 52 D4 EF 77	0CBC EF7752D4 EF77
00460870	FA 8D EF 77 F1 D0 EF 77	FA8DEF77F1D0 EF77
00460878	51 B2 EF 77 26 D5 EF 77	51B2 EF7726D5 EF77
00460880	2A E3 EF 77 5F 39 F2 77	2AE3 EF775F39 F277
00460888	71 B4 EF 77 2E AD EF 77	71B4 EF772EAD EF77
00460890	E1 61 EF 77 B8 85 EF 77	E161 EF77B885 EF77
00460898	CC D2 EF 77 43 70 EF 77	CCD2 EF774370 EF77
004608A0	FB EA F0 77 12 83 EF 77	FB EAF0771283 EF77
004608A8	01 72 F0 77 A9 34 F0 77	0172 F077A934 F077
004608B0	05 93 EF 77 68 EF EF 77	0593 EF7768 EF EF77
004608B8	AA D2 EF 77 B2 6F EF 77	AA D2 EF77B26F EF77
004608C0	3F 38 F2 77 D6 E8 EF 77	3F38 F277D6E8 EF77
004608C8	68 E0 EF 77 00 60 EF 77	68E0 EF770060 EF77

Alli vemos el inicio de la IAT en 460818 ya que mas arriba si marco cualquier entrada y hago FIND REFERENCES no hay referencias, ni van a ningun seccion code de dlls, lamentablemente toda la IAT esta correcta por lo cual parece que no necesitaremos buscar el salto magico, (buaa con lo que habia preparado el script de la parte anterior, pero bueno ya habra otras ocasiones), busquemos el final de la IAT.

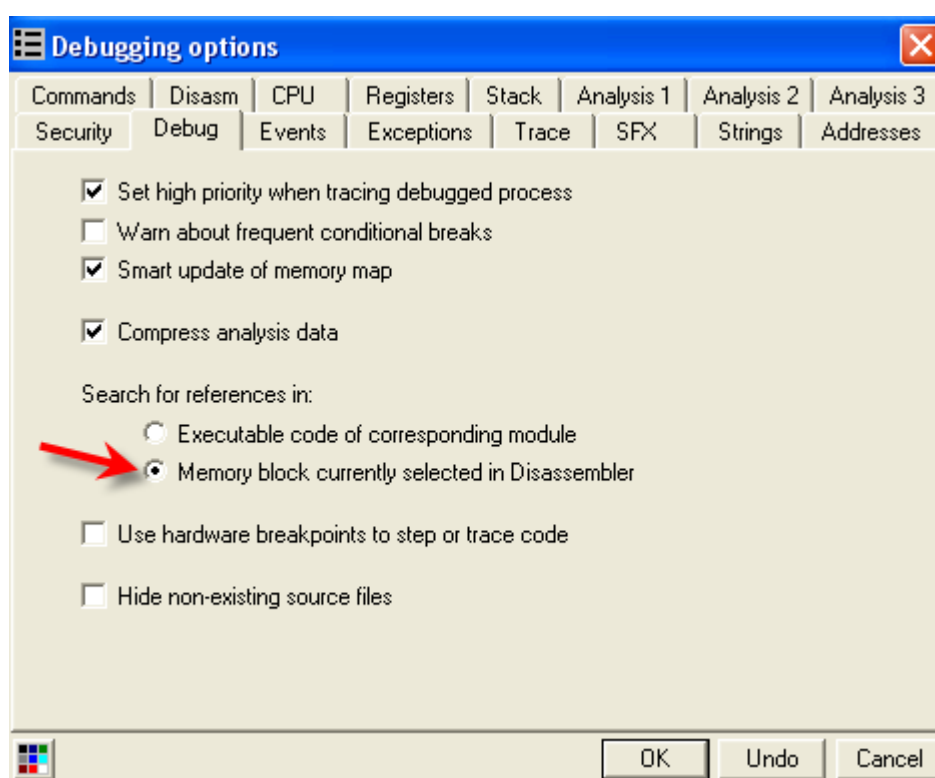
Alli tenemos el final de la IAT



00340000	00010000	UnPackMe			Priv	RW	RW	
00400000	00001000	UnPackMe	.teddy	PE header	Imag	R	RWE	
00401000	0000A000	UnPackMe	.teddy	code	Imag	R	RWE	
00440000	0000C000	UnPackMe	.teddy		Imag	R	RWE	
00457000	00009000	UnPackMe	.teddy		Imag	R	RWE	
00460000	00003000	UnPackMe	.teddy		Imag	R	RWE	
00463000	00008000	UnPackMe	.teddy	resources	Imag	R	RWE	
00468000	0000A000	UnPackMe	.teddy	SFX, imports	Imag	R	RWE	
00480000	00021000				Priv	RW	RW	
00480000	00009000				Map	R E	R E	
00570000	00002000				Map	R E	R E	
00580000	00103000				Map	R	R	
00690000	00001000				Priv	RW	RW	
006A0000	00141000				Map	R E	R E	
009A0000	00001000				Priv	RW	RW	
009B0000	00004000				Priv	RW	RW	
009C0000	00003000				Map	R	R	
009D0000	00001000				Priv	RW	RW	
00A50000	00004000				Priv	RW	RW	
00A60000	00003000				Priv	RW	RW	
00A70000	00002000				Map	R	R	
00A80000	0003C000				Priv	RW	RW	
00AC0000	00001000				Priv	RW	RW	
00BC0000	00002000				Priv	RW	RW	
01280000	00002000				Map	R	R	
58C30000	00001000	COMCTL32		PE header	Imag	R	RWE	
58C31000	00070000	COMCTL32	.text	code, import	Imag	R	RWE	
58CA1000	00003000	COMCTL32	.data	data	Imag	R	RWE	

Vemos que el programa salta a esa seccion y que el programa original arrancado desde el OEP no puede saltar a una seccion afuera del exe sin haberla creado antes con VirtualAlloc o similares, por lo cual concluimos que esa seccion fue creada por el packer y no es parte del codigo original del programa asi como los JMPS que saltan a ella, como por ejemplo el que esta en el falso OEP y hay muchos mas.

Por lo tanto es posible que en esa seccion el packer haya trasladado algunos calls y jmps que toman valores de la IAT y ya que el OLLYDBG si lo tenemos configurado asi.



Busca solo en la seccion que esta visible en el listado en ese momento, pues, como la seccion creada por el packer no esta visible en el listado, no buscara alli.

Probemos buscar las referencias en otras secciones.

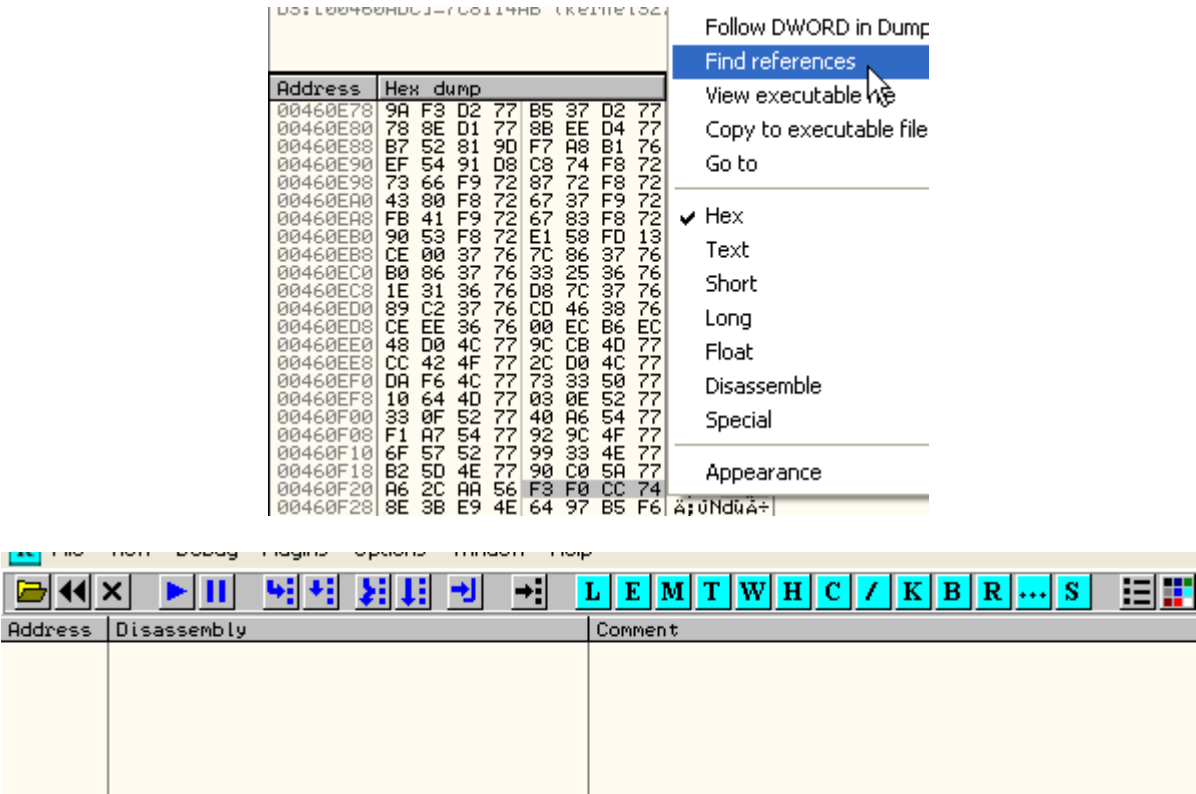
00460F00	33 0F 52 77 40 A6 54 77	3*RW@BTw
00460F08	F1 A7 54 77 92 9C 4F 77	12TwE60w
00460F10	6F 57 52 77 99 33 4E 77	owRW03Hw
00460F18	B2 5D 4E 77 90 C0 5A 77	WJNwe'2w
00460F20	A6 2C AA 56 F3 F0 CC 74	3,-U%-jft
00460F28	8E 3B E9 4E 64 97 B5 F6	uNdüA+
00460F30	9E 7D 09 B0 5F 95 14 1B	xP-öq*
00460F38	61 02 02 16 14 62 59 CC	a0E-1bVlf

00A80000	94	XCHG EAX,ESP	
00A80001	FF15 800E4600	CALL DWORD PTR DS:[460E80]	USER32.GetSysColor
00A80007	E9 211098FF	JMP 0040102D	UnPackMe.0040102D
00A8000C	218D 056275BF	AND DWORD PTR SS:[EBP+BF756205],ECX	
00A80012	888D 809E3A85	MOV BYTE PTR SS:[EBP+853A9E80],CL	
00A80018	77 E9	JA SHORT 00A80003	
00A8001A	6310	ARPL WORD PTR DS:[EAX],DX	
00A8001C	98	CWDE	
00A8001D	FF63 8D	JMP DWORD PTR DS:[EBX-73]	
00A80020	35 8F6EED32	XOR EAX,32ED6E8F	
00A80025	8DB6 EA14DE41	LEA ESI,DWORD PTR DS:[ESI+41DE14EA]	
00A8002B	8DB6 D429BC83	LEA ESI,DWORD PTR DS:[ESI+83BC29D4]	
00A80031	8DB6 A9537807	LEA ESI,DWORD PTR DS:[ESI+77853A9]	
00A80037	E9 6F1098FF	JMP 004010AB	UnPackMe.004010AB
00A8003C	6F	OUTS DX,DWORD PTR ES:[EDI]	I/O command
00A8003D	FF15 30094600	CALL DWORD PTR DS:[460930]	GDI32.DeleteObject
00A80043	E9 871098FF	JMP 004010CF	UnPackMe.004010CF
00A80048	87FF	XCHG EDI,EDI	
00A8004A	15 9C0B4600	ADC EAX,460B9C	
00A8004F	E9 C41098FF	JMP 00401118	UnPackMe.00401118
00A80054	C4FF	LES EDI,EDI	Illegal use of register
00A80056	15 7C0E4600	ADC EAX,460E7C	
00A8005B	E9 FA1098FF	JMP 0040115A	UnPackMe.0040115A
00A80060	FA	CLI	

Alli tenemos una de las secciones creadas por el packer vemos como mi teoria funciona, no hallamos referencias, porque los calls estan trasladados a otras secciones (por eso yo insisto que no hay que usar solo un metodo, en este caso, para ver si una entrada es de la iat o no, necesitamos o bien buscar en las secciones creadas por el packer, o bien mirar si van las entradas a la seccion code de una dll en el mapa de memoria y listo)

Pero bueno ahora tenemos visible una de las secciones creadas por el packer busquemos la referencia en esta.



Nada, pero a no desanimar que hay mas secciones creadas por el packer, pasemos a ver la siguiente.

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped
00380000	00001000				Priv	RWE	RWE	
00390000	00001000				Priv	RWE	RWE	
003A0000	00010000				Priv	RW	RW	
00400000	00001000	UnPackMe		PE header	Imag	R	RWE	
00401000	0004A000	UnPackMe	.teddy	code	Imag	R	RWE	
0044B000	0000C000	UnPackMe	.teddy		Imag	R	RWE	
00457000	00009000	UnPackMe	.teddy		Imag	R	RWE	
00460000	00003000	UnPackMe	.teddy		Imag	R	RWE	
00463000	00008000	UnPackMe	.teddy	resources	Imag	R	RWE	
0046B000	0000A000	UnPackMe	.teddy	SFX, imports	Imag	R	RWE	
00480000	00021000				Priv	RW	RW	
004B0000	00009000				Map	R E	R E	
00570000	00002000				Map	R E	R E	
00580000	00103000				Map	R	R	
00690000	00001000				Priv	RW	RW	
006A0000	00141000				Map	R E	R E	
009A0000	00001000				Priv	RW	RW	
009B0000	00004000				Priv	RW	RW	
009C0000	00003000				Map	R	R	\Device
009D0000	00001000				Priv	RW	RW	
00A50000	00004000				Priv	RW	RW	
00A60000	00003000				Priv	RW	RW	
00A70000	00002000				Map	R	R	
00A80000	0003C000				Priv	RW	RW	
00AC0000	00001000				Priv	RW	RW	
00BCE000	00002000				Priv	RW	RW	
01280000	00002000				Map	R	R	
58C30000	00001000	COMCTL32		PE header	Imag	R	RWE	

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped
00AC0000	EB 02			JMP SHORT 00AC0004				
00AC0002	0C 49			OR AL,49				
00AC0004	EB 02			JMP SHORT 00AC0008				
00AC0006	CD 20			INT 20				
00AC0008	68 2F08817C			PUSH 7C81082F				
00AC000D	EB 02			JMP SHORT 00AC0011				
00AC000F	CD 20			INT 20				
00AC0011	C3			RETN				
00AC0012	8D6424 EC			LEA ESP,DWORD PTR SS:[ESP-14]				
00AC0016	68 2100AC00			PUSH 0AC0021				
00AC001B	E9 2C09D57B			JMP 7C81094C				
00AC0020	0C C1			OR AL,0C1				
00AC0022	F0:008B FFE021			LOCK ADD BYTE PTR DS:[EBX+CD02EBFF],CL				
00AC0029	2055 68			AND BYTE PTR SS:[EBP+68],DL				
00AC002C	35 00AC00EB			XOR EAX,EB00AC00				
00AC0031	010CC3			ADD DWORD PTR DS:[EBX+EAX*8],ECX				
00AC0034	008B ECEB020C			ADD BYTE PTR DS:[EBX+C02EBEC],CL				
00AC003A	49			DEC ECX				
00AC003B	FF75 08			PUSH DWORD PTR SS:[EBP+8]				

Ahora busquemos referencias.

Nada pero como hay unas cuantas secciones y sabemos que un call que lee valores de la entrada 460f24 sera

CALL [460f24]

y los bytes del comando seran

FF 15 24 0f 46 00

por ejemplo si vemos este otro call a una api

00A80031	00B5 A7537007	LEA ESI,DWORD PTR DS:[EBX+70537007]	
00A80037	E9 6F1098FF	JMP 004010AB	UnPackMe.004010AB
00A8003C	6F	OUTS DX,DWORD PTR ES:[EDI]	I/O command
00A8003D	FF15 30094600	CALL DWORD PTR DS:[460930]	GDI32.DeleteObject
00A80043	E9 871098FF	JMP 004010CF	UnPackMe.004010CF
00A80048	87FF	XCHG EDI,EDI	
00A8004A	15 90084600	ADC EAX,4608B9C	
00A8004F	E9 041098FF	JMP 00401118	UnPackMe.00401118
00A80054	0455	INC EAX	Illegal use of register

Vemos que el la secuencia de bytes del call sera FF 15 y a continuacion los bytes de la direccion de la entrada al revés, así que podemos hacer un search de esos bytes por toda la memoria y así podemos buscar si hay una referencia en alguna parte sin tener que cambiar de seccion en seccion.



Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00001000				Priv	RM	RM	
00020000	00001000				Priv	RM	RM	
0012C000	00001000				Priv	RM	Gu	
0012D000	00003000			stack of ma	Priv	RM	Gu	
00130000	00002000				Priv	RWE	RWE	
00140000	00003000				Map	R	R	
00150000	00020000				Priv	RM	RM	
00250000	00006000				Priv	RM	RM	
00260000	00003000				Map	RM	RM	
00270000	00016000				Map	R	R	
00290000	0003D000				Map	R	R	
002D0000	00041000				Map	R	R	
00320000	00006000				Map	R	R	
00330000	00041000				Map	R	R	
00380000	00001000				Priv	RWE	RWE	
00390000	00001000				Priv	RWE	RWE	
003A0000	00010000				Priv	RM	RM	
00400000	00001000	UnPackMe	.teddy	PE header	Inag	R	RWE	
0044B000	0000C000	UnPackMe	.teddy	code	Inag	R	RWE	
00457000	00009000	UnPackMe	.teddy		Inag	R	RWE	
00460000	00003000	UnPackMe	.teddy		Inag	R	RWE	
00463000	00008000	UnPackMe	.teddy	resources	Inag	R	RWE	
0046B000	0000A000	UnPackMe	.teddy	SFX, imports	Inag	R	RWE	
00480000	00021000				Priv	RM	RM	
004B0000	00009000				Map	R E	R E	
00570000	00002000				Map	R E	R E	
00580000	00103000				Map	R	R	
00690000	00001000				Priv	RM	RM	
006A0000	00141000				Map	R E	R E	
009A0000	00001000				Priv	RM	RM	
009B0000	00004000				Priv	RM	RM	
009C0000	00003000				Map	R	R	
009D0000	00001000				Priv	RM	RM	
00A50000	00004000				Priv	RM	RM	
00A60000	00003000				Priv	RM	RM	
00A70000	00002000				Map	R	R	
00A80000	00003C000				Priv	RM	RM	
00AC0000	00001000				Priv	RM	RM	
00BCE000	00002000				Priv	RM	Gu	
01000000	00000000				Map	R	RM	

Vamos al mapa de memoria y alli hacemos click derecho

Type	Access	Initial	Mapped as
Priv	RM	RM	
Priv	RM	RM	
Priv	RM	Gu	
Priv	RM	Gu	
Priv	RWE		
Map	R		
Priv	RM		
Priv	RM		
Map	RM		
Map	R		
Map	R		
Map	R		

Actualize

Dump in CPU

Dump

**Search** Ctrl+B

Set break-on-access F2

Enter binary string to search for

ASCII

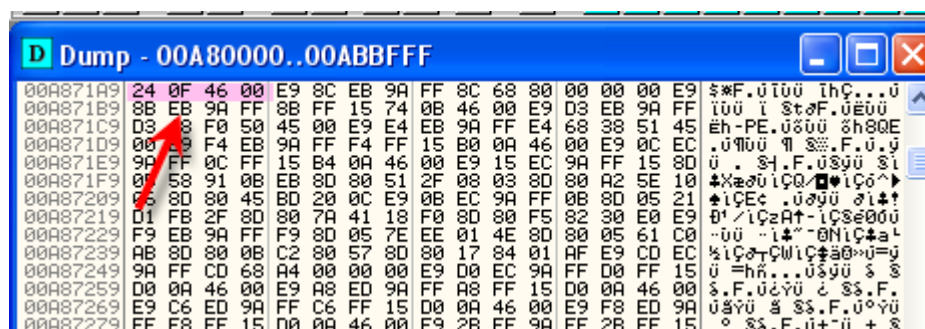
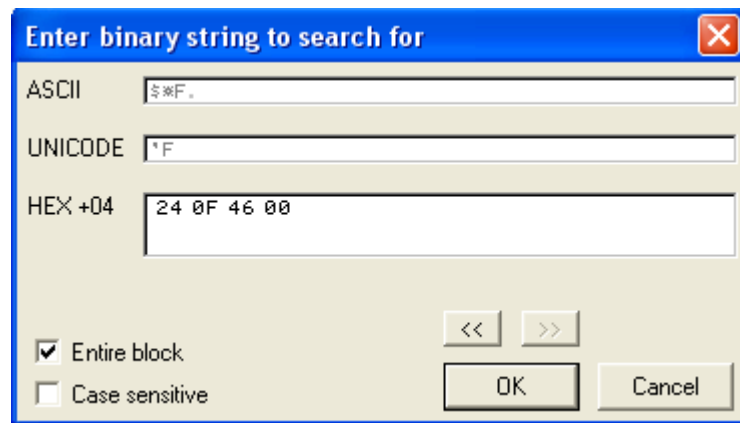
UNICODE

HEX +06

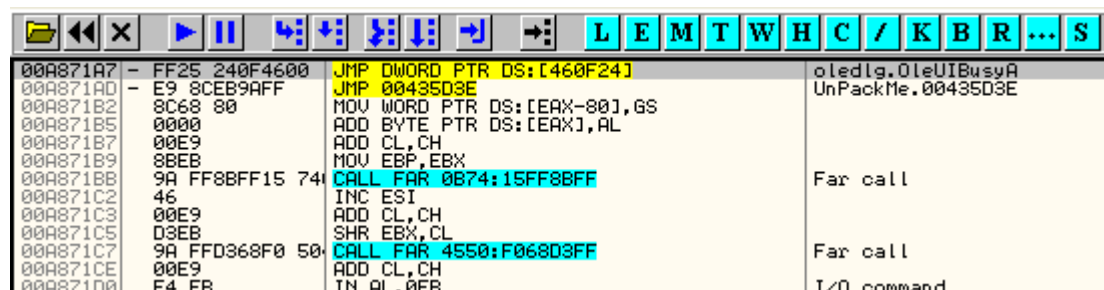
☒ Entire block

☐ Case sensitive

Y escribimos en laparte de HEX la secuencia de bytes que queremos hallar y apretamos OK y no salen resultados grr, pero como somos persistentes y por supuesto hay muchas formas de llamar a una api que no son call indirectos le quitaremos el FF 15 y buscaremos solo la direccion de memoria.



Allí encuentro algo y justo en una de las secciones creadas por el packer, vayamos al listado a ver esta zona.



Allí hallamos la bendita referencia, claro nosotros probamos buscar con FF 15 que es un call indirecto y no probamos con FF 25 que es un JMP indirecto, pero bueno, buscando de esta ultima forma nos sale las referencias para cualquier comando que haya para llamar a la api.

Por lo tanto quiere decir que esa es una entrada de la IAT.



Address	Hex dump	ASCII
00460E78	9A F3 02 77 B5 37 02 77	0%EWÁ7EW
00460E80	78 8E 01 77 8B EE 04 77	xÁ0wí7EW
00460E88	B7 52 81 9D F7 A8 B1 76	ARü0-28U
00460E90	EF 54 91 08 C8 74 F8 72	*Tæi4t0r
00460E98	73 66 F9 72 87 72 F8 72	sf-r9r0r
00460EA0	43 80 F8 72 67 37 F9 72	CÇ0rg7-r
00460EA8	F8 41 F9 72 67 83 F8 72	'A-r9ã0r
00460EB0	90 53 F8 72 E1 58 FD 13	éS0rßX0!!
00460EB8	CE 00 37 76 7C 86 37 76	tr.7v!87v
00460EC0	B0 86 37 76 33 25 36 76	87v3%6v
00460EC8	1E 31 36 76 08 7C 37 76	16vü!7v
00460ED0	89 C2 37 76 CD 46 38 76	ët7vF8v
00460ED8	CE EE 36 76 00 EC B6 EC	tr6v.9ãv
00460EE0	48 00 4C 77 9C CB 4D 77	HsLw6trMw
00460EE8	CC 42 4F 77 2C D0 4C 77	lfB0w,3Lw
00460EF0	DA F6 4C 77 73 33 50 77	r÷Lws3Pw
00460EF8	10 64 4D 77 03 0E 52 77	ldMw08Rw
00460F00	33 0F 52 77 40 A6 54 77	3wRw03Tw
00460F08	F1 A7 54 77 92 9C 4F 77	±0TwE60w
00460F10	6F 57 52 77 99 33 4E 77	0wRw03Hw
00460F18	B2 5D 4E 77 90 C0 5A 77	8JHwE12w
00460F20	A6 2C A9 56 F3 F0 CC 74	ã,-U8-1F8
00460F28	8E 3B E9 4E 64 97 B5 F6	ã:üNdüA÷
00460F30	9E 7D 09 B0 5F 95 14 1B	xJ-80q+
00460F38	61 02 02 16 14 62 59 CC	a0E-7bVlf
00460F40	A9 4F 28 35 EF 76 0B 48	00(5'00H
00460F48	CB 6F 5C A7 98 A9 2B 3C	tr0\000+<
00460F50	CD A6 F6 5F 53 A3 16 A3	=ã÷ Su_u

Y allí termina la IAT ya que mas abajo antes de ponerme a buscar referencias ya veo que no hay valores que vayan a la seccion code de ninguna dll con lo cual ya directamente no son entradas de la IAT, la cual termina en 460f28.

Bueno los datos de la IAT son

LARGO=FINAL -INICIO= 460f28 - 460818= 710



Asi que poniendo en limpio

**OEP=271B0**  
**RVA o INICIO=60818**  
**SIZE O LARGO=710**

Ahora vemos que la IAT esta toda correcta asi que llegaremos nuevamente al falso oep, le agregaremos los stolen bytes y dumparemos.

Aquí llego al falso OEP.

Los stolen bytes eran

55 8B EC 6A FF 68 60 0E 45 00 68 C8 92 42 00 64 A1 00 00 00 00 50 64 89 25 00 00 00 00 83 C4  
A8 53 56 57 89 65 E8

Los pego en el dumpeado a partir de 4271b0 como habiamos visto en la parte 39.

004271AF	90	NOP	
004271B0	55	PUSH EBP	
004271B1	8BEC	MOV EBP,ESP	
004271B3	6A FF	PUSH -1	
004271B5	68 600E4500	PUSH 450E60	
004271BA	68 C8924200	PUSH 4292C8	
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
004271C5	50	PUSH EAX	
004271C6	64:8925 000000	MOV DWORD PTR FS:[0],ESP	
004271CD	83C4 A8	ADD ESP,-58	
004271D0	53	PUSH EBX	
004271D1	56	PUSH ESI	
004271D2	57	PUSH EDI	
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
004271D6	E9 8AD66500	JMP 00A84865	
004271D8	0033	ADD BYTE PTR DS:[EBX],DH	
004271DD	D28A D4891534	ROR BYTE PTR DS:[EDX+341589D4],CL	I/O command
004271E3	E6 45	OUT 45,AL	
004271E5	008B C881E1FF	ADD BYTE PTR DS:[EBX+FFE181C8],CL	
004271EB	0000	ADD BYTE PTR DS:[EAX],AL	
004271ED	0089 0D30E645	ADD BYTE PTR DS:[ECX+45E6300D],CL	
004271F3	00C1	ADD CL,AL	
004271F5	E1 08	LOOPDE SHORT 004271FF	UnPackMe.004271FF
004271F7	03CA	ADD ECX,EDX	
004271F9	890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX	
004271FF	C1E8 10	SHR EAX,10	
00427202	A3 28E64500	MOV DWORD PTR DS:[45E628],EAX	
00427207	E8 94210000	CALL 004293A0	UnPackMe.004293A0
0042720C	85C0	TEST EAX,EAX	
0042720E	75 0A	JNZ SHORT 0042721A	UnPackMe.0042721A
00427210	6A 1C	PUSH 1C	

Address	Hex dump	ASCII
00427190	F9 00 00 01 00 75 08 80	..0.uQ
00427198	CC 02 EB 03 80 CC 03 F7	00000000
004271A0	C2 00 00 04 00 74 03 80	T...t
004271A8	CC 10 C3 90 90 90 90 90	00000000
004271B0	55 8B EC 6A FF 68 60 0E	U00j h'A
004271B8	45 00 68 C8 92 42 00 64	E.hEB.d
004271C0	A1 00 00 00 00 50 64 89	.....Pd
004271C8	25 00 00 00 00 83 C4 A8	.....A
004271D0	53 56 57 89 65 E8 E9 8A	SUW00000
004271D8	D6 65 00 00 33 D2 8A D4	ie...0000
004271E0	89 15 34 E6 45 00 8B C8	034pE.00
004271E8	81 E1 FF 00 00 89 0D	00...000
004271F0	30 E6 45 00 C1 E1 08 03	0pE.+000
004271F8	CA 89 0D 2C E6 45 00 C1	00...pE.0

Ahora dumpearlo cambiando el OEP a 4271b0

OllyDump - UnPackMe\_PELock1.06.d.exe

Start Address: 400000 Size: 75000 Dump

Entry Point: 6B05C -> Modify: 271b0 Get EIP as OEP Cancel

Base of Code: 1000 Base of Data: C

☒ Fix Raw Size & Offset of Dump Image

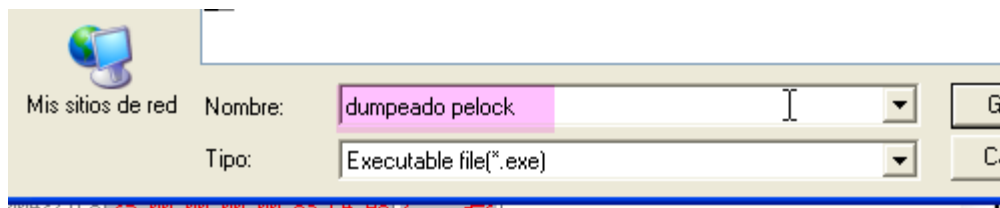
Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.teddy	0004A000	00001000	0004A000	00001000	C00000E0
.teddy	0000C000	0004B000	0000C000	0004B000	C00000E0
.teddy	00009000	00057000	00009000	00057000	C00000E0
.teddy	00003000	00060000	00003000	00060000	C00000E0
.teddy	00008000	00063000	00008000	00063000	C00000E0
.teddy	0000A000	0006B000	0000A000	0006B000	C00000E0

☐ Rebuild Import

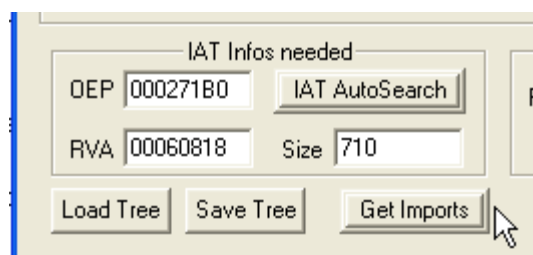
☒ Method1 : Search JMP[API] | CALL[API] in memory image

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

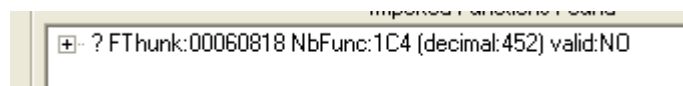
Allí cambie el OEP para que se corrija y le quito la tilde en REBUILD IMPORTS y apreto dump para dumpear.



Allí lo guardamos como dumpeado pelock, ahora podemos repararle la IAT con este mismo que tenemos detenido en el OEP ya que la misma estaba correcta, así que abro el IMP REC.



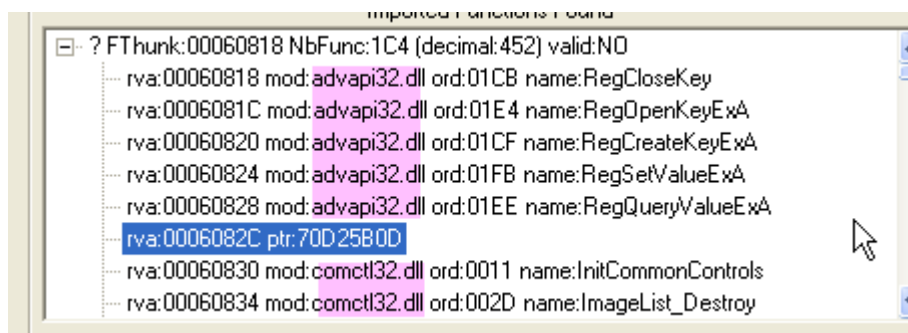
Le colocamos los datos de la IAT que averiguamos y apretamos GET IMPORTS.



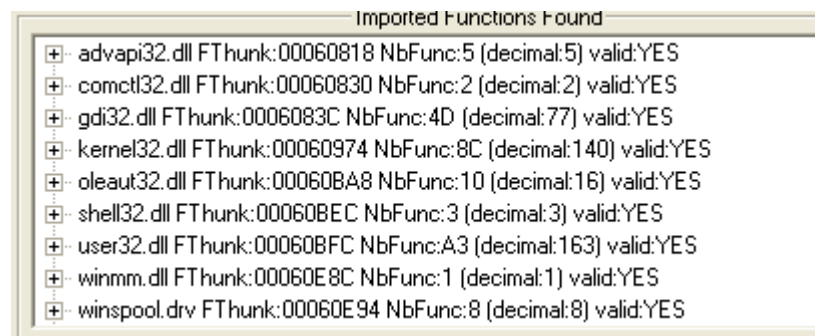
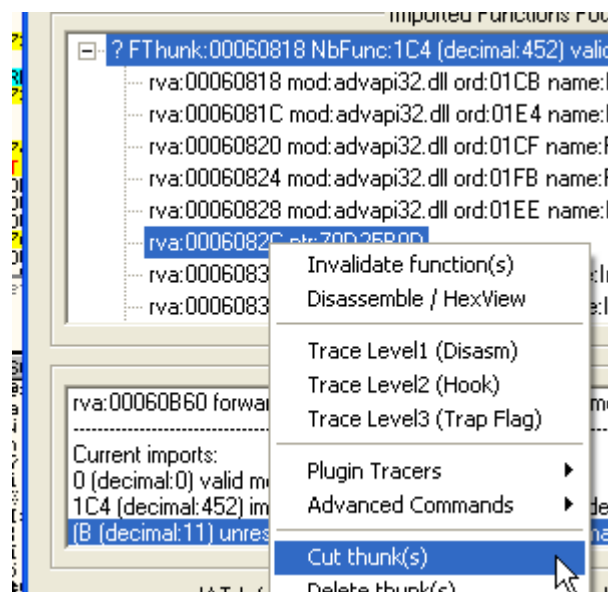
Vemos que dice que NO, pero si recordábamos no hay entradas redireccionadas el problema es que el packer cambió los ceros que separan las diferentes entradas de cada dll por basura, si miramos.

Address	Hex dump	ASCII
0046091C	1A 40 F2 77 55 EA EF 77	+@=wUü'w
00460924	C5 61 EF 77 70 E6 EF 77	+a'wpy'w
0046092C	F0 81 EF 77 2D 6C EF 77	-ü'w-l'w
00460934	98 6E EF 77 4F 83 EF 77	ÿn'w0â'w
0046093C	09 ED EF 77 EB AA EF 77	.ÿ'wü-r'w
00460944	26 69 F0 77 B1 95 EF 77	&l-wÿo'w
0046094C	6F B0 EF 77 8A 5A EF 77	cÿ'weZ'w
00460954	E9 49 F2 77 26 F1 F0 77	üI=w&:-w
0046095C	C9 D0 F0 77 51 E0 F0 77	ÿI'-w00-w
00460964	33 8C EF 77 6C EC EF 77	3I'wü'w
0046096C	29 94 EF 77 69 05 E6 3E	)ö'wü!p>
00460974	6B 17 80 7C C1 C9 80 7C	kÿC!-ÿC
0046097C	69 10 81 7C EE 1E 80 7C	i,ü!-ÿC
00460984	8D 2C 81 7C 40 7A 94 7C	i,ü!@zö
0046098C	E1 EA 81 7C A2 CA 81 7C	püü!ö±ü
00460994	16 1E 80 7C 43 99 80 7C	-ÿC!C0C
0046099C	10 11 81 7C 29 29 81 7C	ÿü!))ü
004609A4	14 9B 80 7C 81 9A 80 7C	ÿÿC!üüC
004609AC	FB 2C 82 7C AE 94 83 7C	' ,é!<öâ
004609B4	2B 2E 83 7C C4 CE 80 7C	+ ,ä!-ÿC
004609BC	8D 2B 82 7C 8F 8F 81 7C	+ ,ä!2-ÿC

Allí se ve claro en vez de haber ceros entre las diferentes entradas de cada dll, hay basura, esto el IMP REC lo arregla fácilmente, apretemos SHOW INVALID.

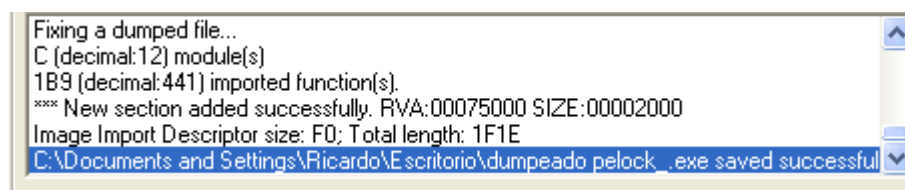


Ahi corroboramos lo que deciamos estan las entradas de advapi32.dll y la separacion con las de comctl32.dll es basura, asi que como tenemos todas esas basuras marcadas al haber apretado SHOW INVALID, hacemos click deerecho CUT THUNKS y anulara todas esas entradas.



Como vemos al anular las entradas basura en IMP REC ya no las toma en cuenta y la tabla se arregla, quedando esas entradas basura sin efecto.

Ahora apreto FIX DUMP para arreglar el dumpeado.



Y alli creo el dumpeado pelock\_.exe con la IAT reparada, , y ahora si corro el archivo reparado da error que ocurre, abramoslo en OLLYDBG.sin cerrar el otro que tengo detenido en el OEP aun.

Como me da un error de formato lo paso en el LORDPE en la opcion REBUILDPE para que repare el header, y ahora si podemos verlo en OLLYDBG correctamente.

```

File View Debug Plugins Options Window Help
[Icons] [L] [E] [M] [T] [W] [H] [C] [ / ] [K] [B] [R] [ ... ] [S]
004271B0 55 PUSH EBP
004271B1 8BEC MOV EBP,ESP
004271B3 6A FF PUSH -1
004271B5 68 600E4500 PUSH 450E60
004271B8 68 C8924200 PUSH 4292C8
004271BF 64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
004271C5 50 PUSH EAX
004271C6 64:8925 00000000 MOV DWORD PTR FS:[0],ESP
004271CD 83C4 A8 ADD ESP,-58
004271D0 53 PUSH EBX
004271D1 56 PUSH ESI
004271D2 57 PUSH EDI
004271D3 8965 E8 MOV DWORD PTR SS:[EBP-18],ESP
004271D6 - E9 8AD66500 JMP 00A84865
004271D8 0033 ADD BYTE PTR DS:[EBX],DH
004271DD 028A D4891534 ROR BYTE PTR DS:[EDX+341589D4],CL
004271E3 E6 45 OUT 45,AL
004271E5 008B C881E1FF ADD BYTE PTR DS:[EBX+FFE181C8],CL
004271F1 0033 ADD BYTE PTR DS:[EBX],AL

```

Alli lo vemos parado en su EP en forma correcta miremos la IAT.

Address	Hex dump	ASCII
00460818	F0 6B DA 77 1B 76 DA 77	-k rw+Urw
00460820	F4 EA DA 77 E7 EB DA 77	10 rWpUrw
00460828	83 78 DA 77 00 00 00 00	ax rw...
00460830	DD 15 C5 58 2E BD C3 58	!\$+X.c!X
00460838	00 00 00 00 D4 6A EF 77	...Ej'w
00460840	66 95 EF 77 89 6A EF 77	fö'wëj'w
00460848	F3 AD EF 77 ED 09 EF 77	%i'wY'w
00460850	99 8B EF 77 C0 B5 EF 77	0i'wL'a'w
00460858	2A 7D EF 77 B2 7C EF 77	*}w!w
00460860	77 53 F2 77 1E C9 F1 77	WS=wAf'w
00460868	0C BC EF 77 52 D4 EF 77	.wRE'w
00460870	FA 8D EF 77 F1 DD EF 77	.l'w!w
00460878	51 B2 EF 77 26 D5 EF 77	0w&'w
00460880	2A E3 EF 77 5F 39 F2 77	*0'w_9=w
00460888	71 B4 EF 77 2E AD EF 77	q!w.i'w
00460890	E1 61 EF 77 B8 85 EF 77	pa'w0a'w
00460898	CC D2 EF 77 43 70 EF 77	lFE'wCp'w
004608A0	FB EA F0 77 12 83 EF 77	'U-w0a'w
004608A8	01 72 F0 77 A9 34 F0 77	0r-w04-w
004608B0	D5 93 EF 77 68 EF EF 77	'0'wh'w
004608B8	AA D2 EF 77 B2 6F EF 77	-E'w0'o'w
004608C0	3F 38 F2 77 D6 E8 EF 77	?8=wip'w
004608C8	68 E0 EF 77 00 60 EF 77	h0'w.'w
004608D0	90 5B EF 77 6D AC EF 77	E['wM'w
004608D8	94 6C F0 77 22 8D EF 77	0l-w''l'w
004608E0	3D C8 F1 77 3D 6D F0 77	=B=wM-w
004608E8	6F C0 EF 77 85 7B EF 77	oL'w0c'w
004608F0	26 D9 EF 77 FB 5E EF 77	&J'w1^'w
004608F8	36 8A EF 77 FC 8A EF 77	6ë'w2ë'w
00460900	0F 62 EF 77 49 5E EF 77	*b'w1^'w
00460908	97 5D EF 77 1A 9A EF 77	üJ'w+ü'w
00460910	6B FA EF 77 7B C9 F0 77	k.'wCf-w
00460918	00 92 F2 77 10 40 F2 77	ü=w00-w

Alli se ve la IAT correctamente reparada por el IMP REC, vemos que el mismo reemplazo los valores basura por ceros para que la separacion quede perfecta entre las entradas de las diferentes dlls, sin embargo al correrlo aquí en OLLYDBG da error que ocurre.

Pues muy facil aquí viene el proximo punto que debemos estudiar, luego de reparar un dumpeado, la IAT y arreglar los stolen bytes si tiene, aun nos queda un truco mas que tienen muchos packers el ANTIDUMP.

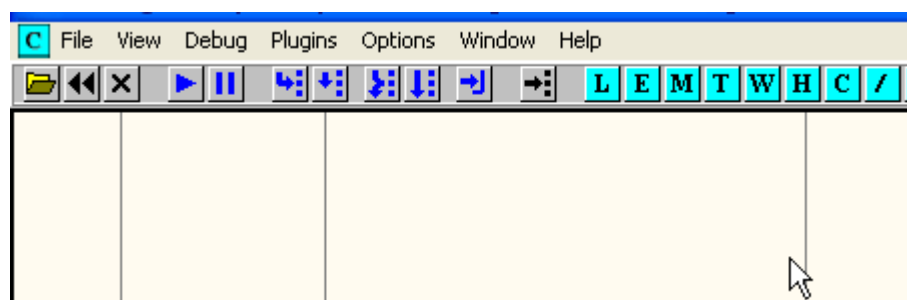
Hay muchos tipos de antidump algunos son simplemente un chequeo de si el programa esta dumpeado, chequeando la imagesize para ver el largo, o la cantidad de secciones, para ver si el IMP REC, agrego la seccion que siempre coloca para reparar la IAT, pero en este caso, el antidump son varias secciones que el packer creo en tiempo de ejecucion y que desvia ciertas partes del programa alli, y al dumpear no tenemos dichas secciones, por lo cual el dumpeado no corre, lo vemos claramente en el EP.

004271B0	55	PUSH EBP	
004271B1	8BEC	MOV EBP,ESP	
004271B3	6A FF	PUSH -1	
004271B5	68 600E4500	PUSH 450E60	
004271B8	68 C8924200	PUSH 4292C8	
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
004271C5	50	PUSH EAX	
004271C6	64:8925 000000	MOV DWORD PTR FS:[0],ESP	
004271CD	83C4 A8	ADD ESP,-58	
004271D0	53	PUSH EBX	
004271D1	56	PUSH ESI	
004271D2	57	PUSH EDI	
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
004271D6	E9 8AD66500	JMP 00A84865	
004271DB	0033	ADD BYTE PTR DS:[EBX],DH	
004271DD	D28A D4891534	ROR BYTE PTR DS:[EDX+341589D4],CL	
004271E3	E6 45	OUT 45,AL	I/O command
004271F5	0000 C881E1FF	ADD BYTE PTR DS:[EBX+FFF181C8],CL	

El programa comenzara a ejecutarse pero al llegar al JMP dara error traceemos hasta alli.

004271CD	83C4 A8	ADD ESP,-58	
004271D0	53	PUSH EBX	
004271D1	56	PUSH ESI	
004271D2	57	PUSH EDI	
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
004271D6	E9 8AD66500	JMP 00A84865	
004271DB	0033	ADD BYTE PTR DS:[EBX],DH	
004271DD	D28A D4891534	ROR BYTE PTR DS:[EDX+341589D4],CL	
004271E3	E6 45	OUT 45,AL	I/O command
004271F5	0000 C881E1FF	ADD BYTE PTR DS:[EBX+FFF181C8],CL	

Vemos que si apreto F7 salta a una seccion inexistente.



Y no puede continuar mas si miro el original, veo que el JMP si nos lleva a una zona que creo el packer en tiempo de ejecucion.

004271D6	E9 8AD66500	JMP 00A84865	
004271DB	0033	ADD BYTE PTR DS:[EBX],DH	
004271DD	D28A D4891534	ROR BYTE PTR DS:[EDX+341589D4],CL	
004271E3	E6 45	OUT 45,AL	
004271E5	0000 C881E1FF	ADD BYTE PTR DS:[EBX+FFF181C8],CL	
004271EB	0000	ADD BYTE PTR DS:[EAX],AL	
004271ED	0009 0D30E645	ADD BYTE PTR DS:[ECX+45E645],AL	
004271F3	00C1	ADD CL,AL	
004271F5	E1 08	LOOPDE SHORT 004271FF	
004271F7	03CA	ADD ECX,EDX	
004271F9	890D 2CE64500	MOV DWORD PTR DS:[45E62C0D],ECX	
004271FF	C1E8 10	SHR EAX,10	
00427202	A3 28E64500	MOV DWORD PTR DS:[45E62800],EAX	
00427207	E8 94210000	CALL 004293A0	
0042720C	85C0	TEST EAX,EAX	
0042720E	75 0A	JNZ SHORT 0042721A	
00427210	6A 1C	PUSH 1C	
00427212	E8 49010000	CALL 00427360	
00427217	83C4 04	ADD ESP,4	
0042721A	E8 D12F0000	CALL 0042A1F0	
0042721F	85C0	TEST EAX,EAX	

Hago click derecho-FOLLOW y me muestra donde saltara el JMP.



Address	Disassembly	Comment
00A84865	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
00A8486B	E9 6C299AFF	JMP 004271DC
00A84870	6C	INS BYTE PTR ES:[EDI],DX
00A84871	FF15 84094600	CALL DWORD PTR DS:[460984]
00A84877	E9 C8299AFF	JMP 00427244
00A8487C	C8 FF1580	ENTER 15FF,80
00A84880	0946 00	OR DWORD PTR DS:[ESI],EAX
00A84883	E9 532A9AFF	JMP 004272DB
00A84888	53	PUSH EBX
00A84889	8D05 E4E5F1B3	LEA EAX,DWORD PTR DS:[B3F1E5E4]
00A8488F	8D80 261A0E4C	LEA EAX,DWORD PTR DS:[EAX+4C0E1A26]
00A84895	E9 562A9AFF	JMP 004272F0
00A8489A	56	PUSH ESI
00A8489B	FF15 9C0B4600	CALL DWORD PTR DS:[460B9C]
00A848A1	E9 562A9AFF	JMP 004272FC
00A848A6	56	PUSH ESI
00A848A7	68 FF000000	PUSH 0FF
00A848AC	E9 CF2A9AFF	JMP 00427380
00A848B1	CF	IRETD
00A848B2	FF15 E08D4500	CALL DWORD PTR DS:[458DE0]
00A848B8	E9 C92A9AFF	JMP 00427386
00A848BD	C9	LEAVE
00A848BE	68 80000000	PUSH 80
00A848C3	E9 7D2B9AFF	JMP 00427445
00A848C8	7D 8D	JGE SHORT 00A84857
00A848CA	05 6EF5C6DD	ADD EAX,DDC6F56E

Module
kernel32.GetVersion
UnPackMe.004271DC
I/O command
kernel32.GetCommandLineA
UnPackMe.00427244
UnPackMe.004272DB
UnPackMe.004272F0
UnPackMe.004272FC
kernel32.GetModuleHandleA
UnPackMe.00427380
UnPackMe.00427400
UnPackMe.00427386
UnPackMe.00427445

Alli vemos que ejecuta un CALL a una api y luego vuelve al codigo, hay muchas formas de reparar esto, pero la mas sencilla es agregarle al dumpeado la seccion que le falta y ubicarla en la misma posicion, como se hace esto, pues vamos a ver.

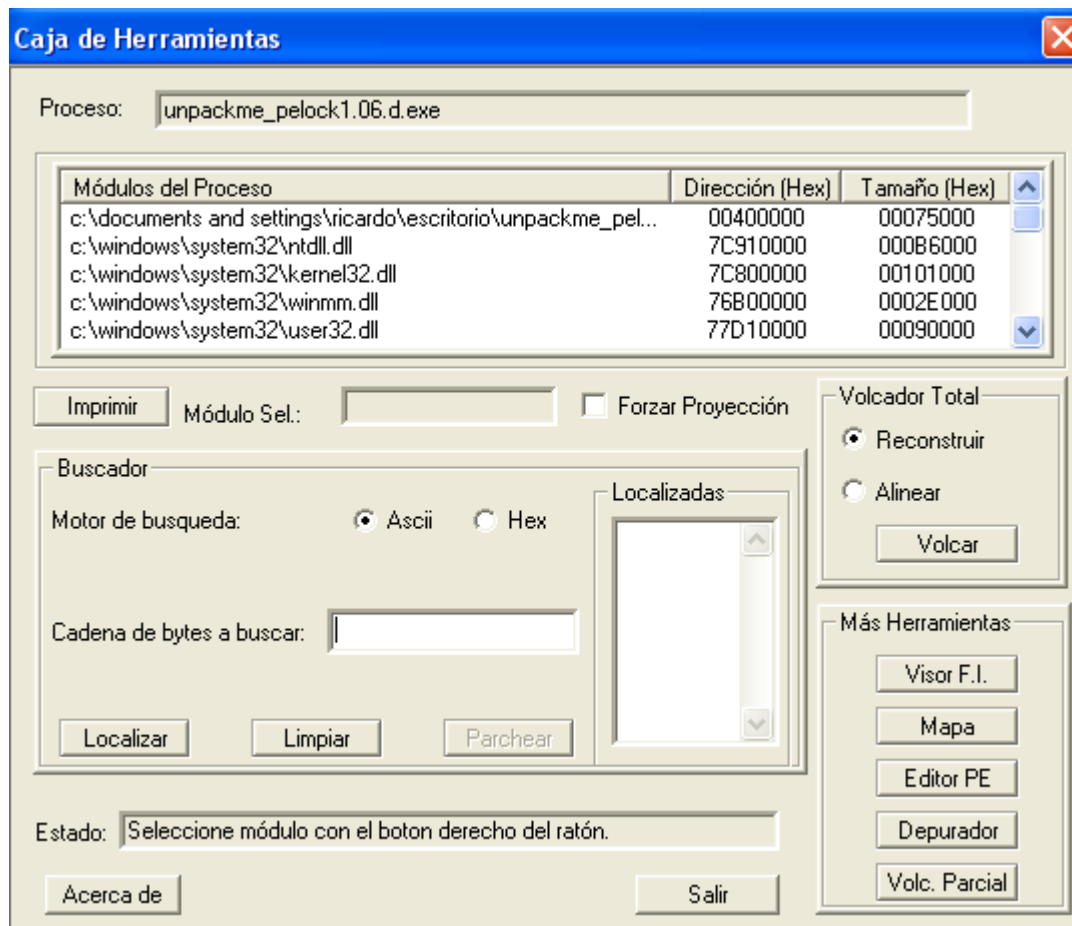
Miremos en el original la seccion faltante, primero le agregaremos esta que es la que nos da error al inicio si llega a faltar alguna mas la agregaremos despues.

Necesitamos DUMPEAR solo la seccion que nos falta, para ello utilizaremos el PUPE que es una muy buena herramienta para dumppear partes de procesos o secciones sueltas entre muchos usos, asi que buscamos el pupe que estara adjunto al tutorial.

Cerremos el OLLYDBG con el dumpeado y dejamos el original que esta detenido en el falso OEP y lo abrimos en el PUPE.

Procesos en ejecución	ID Proceso	ID Modulo	Nº Threads	Prioridad
pupe.exe	00000FA8	00000000	00000001	Normal
lordpe.exe	00000C78	00000000	00000002	Normal
importrec.exe	00000998	00000000	00000002	Normal
unpackme_pelock1.06.d.exe	000008B0	00000000	00000002	Normal
petools.exe	00000B18	00000000	00000001	Normal
winhlp32.exe	00000F34	00000000	00000001	Normal
winhlp32.exe	00000F14	00000000	00000001	Normal

Hacemos click derecho- CAJA DE HERRAMIENTAS

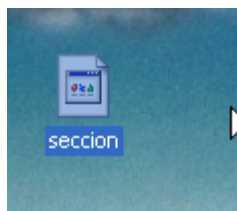


Allí está abierto el proceso ahora vayamos al mapa de memoria para dumppear la sección faltante, vamos a MAPA.

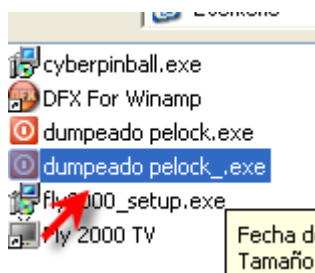
Recordamos que la sección faltante empezaba en A80000 así que la buscamos en el mapa del pape.



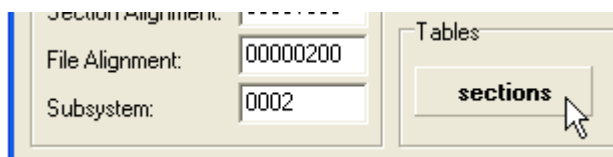
Allí está apretamos VOLCAR y la guardo con el nombre sección.



Ahora abro el PE EDITOR y la agregare al dumpeado.



Busco el dumpeado reparado y lo abro en el PE EDITOR.



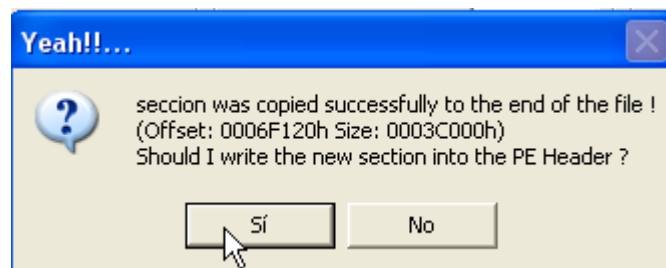
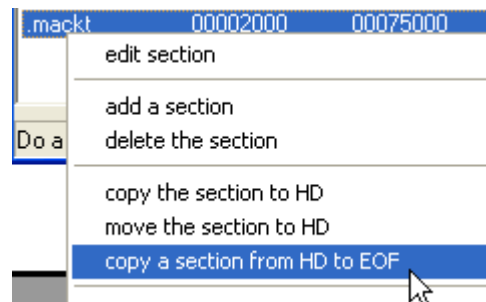
Apreto sections.

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.teddy	0004A000	00001000	00049183	00000400	C00000E0
.teddy	0000C000	0004B000	0000BEDB	00049600	C00000E0
.teddy	00009000	00057000	000056E0	00055600	C00000E0
.teddy	00003000	00060000	000010DB	0005AE00	C00000E0
.teddy	00008000	00063000	0000789D	0005C000	C00000E0
.teddy	0000A000	0006B000	000096F8	00063A00	C00000E0
.mackt	00002000	00075000	00001F20	0006D200	E0000060

Do a right mouse click on a sectionname for more options...

Alli vemos las secciones del dumpeado que son las mismas del original, mas la seccion mackt que agrego el IMP REC para arreglar la IAT, pues bien agreguemosle la seccion faltante al final.

Hago click derecho y apreto COPY A SECTION FROM HD TO EOF (End of file)



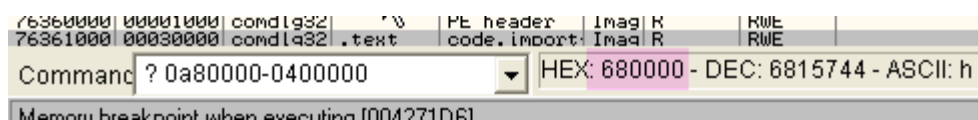
Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.teddy	0004A000	00001000	00049183	00000400	C00000E0
.teddy	0000C000	00048000	0000BEDB	00049600	C00000E0
.teddy	00009000	00057000	000056E0	00055600	C00000E0
.teddy	00003000	00060000	000010DB	0005AE00	C00000E0
.teddy	00008000	00063000	0000789D	0005C000	C00000E0
.teddy	0000A000	00068000	000096F8	00063A00	C00000E0
.mack	00002000	00075000	00001F20	0006D200	E0000060
seccion	0003C000	00077000	0003C000	0006F120	C0000040

Do a right mouse click on a sectionname for more options...

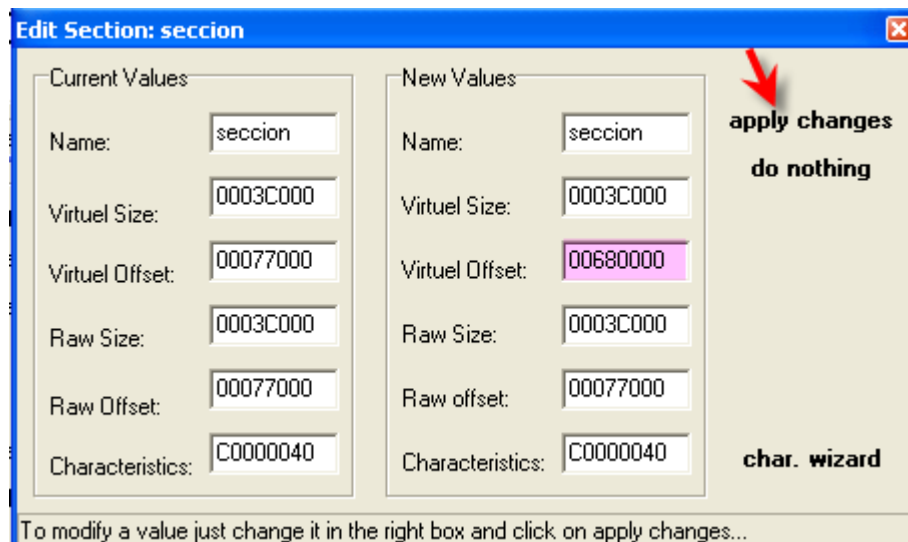
Ahora solo nos queda cambiarle el VIRTUAL OFFSET a 0A80000 para que arranque alli ya que es la direccion en la que arrancara.



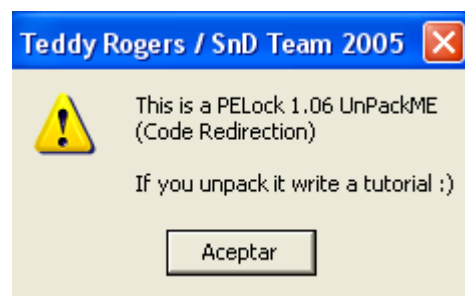
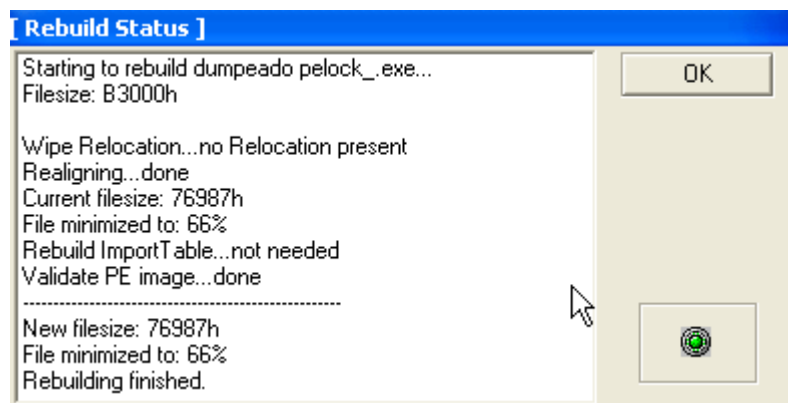
Ahora debo cambiar la direccion del Virtual Offset a 0a80000 restandole 400000 quedaria.



680000 seria la direccion del virtual offset de la nueva seccion.



Ahora le hacemos una pasada para que repare el header el LORDPE nuevamente con la opcion REBUILD PE.



Lo ejecutamos y vemos que funciona si lo abrimos en OLLYDBG.

```

004271B0 $ 55 PUSH EBP
004271B1 . 8BEC MOV EBP,ESP
004271B3 . 6A FF PUSH -1
004271B5 . 68 60E4500 PUSH 450E60
004271BA . 68 C8924200 PUSH 4292C8
004271BF . 64:A1 000000 MOV EAX,DWORD PTR FS:[0]
004271C5 . 50 PUSH EAX
004271C6 . 64:8925 0000 MOV DWORD PTR FS:[0],ESP
004271CD . 83C4 A8 ADD ESP,-58
004271D0 . 53 PUSH EBX
004271D1 . 56 PUSH ESI
004271D2 . 57 PUSH EDI
004271D3 . 8965 E8 MOV DWORD PTR SS:[EBP-18],ESP
004271D6 . E9 8AD66500 JMP 00A84865
004271D8 . 00 DB 00
004271DC . 33D2 XOR EDX,EDX
004271DE . 8AD4 MOV DL,AH

```

Veamos el mapa de memoria

003A0000	00001000				Priv	RW	RW	
003B0000	00001000				Priv	RW	RW	
003C0000	00004000				Priv	RW	RW	
003D0000	00004000				Priv	RW	RW	
003E0000	00003000				Map	R	R	\Device\Harddi
003F0000	00002000				Map	R	R	
00400000	00001000	dumpeado	PE header		Imag	R	RWE	
00401000	00004000	dumpeado	code	.teddy	Imag	R	RWE	
0044B000	0000C000	dumpeado		.teddy	Imag	R	RWE	
00457000	00009000	dumpeado		.teddy	Imag	R	RWE	
00460000	00003000	dumpeado		.teddy	Imag	R	RWE	
00463000	00008000	dumpeado	resources	.teddy	Imag	R	RWE	
0046B000	0000A000	dumpeado		.teddy	Imag	R	RWE	
00475000	0060B000	dumpeado	imports	.mact	Imag	R	RWE	
00A80000	0003C000	dumpeado	data	section	Imag	R	RWE	
00C00000	0000B000				Map	R E	R E	
00C00000	00002000				Map	R E	R E	

Vemos que el LORDPE para hacerlo al dumpeado funcional agrando la seccion anterior para hacerlas contiguas con la que agregue, y luego viene la seccion que le faltaba que va desde 0a80000 en adelante y que tiene el codigo que dumpeamos con el PUPE.

```

004271BF . 64:A1 000000 MOV EAX,DWORD PTR FS:[0]
004271C5 . 50 PUSH EAX
004271C6 . 64:8925 0000 MOV DWORD PTR FS:[0],ESP
004271CD . 83C4 A8 ADD ESP,-58
004271D0 . 53 PUSH EBX
004271D1 . 56 PUSH ESI
004271D2 . 57 PUSH EDI
004271D3 . 8965 E8 MOV DWORD PTR SS:[EBP-18],ESP
004271D6 . E9 8AD66500 JMP 00A84865
004271D8 . 00 DB 00
004271DC . 33D2 XOR EDX,EDX
004271DE . 8AD4 MOV DL,AH
004271E0 . 8915 34E64500 MOV DWORD PTR DS:[45E6]
004271E6 . 8BC8 MOV ECX,EAX
004271E8 . 81E1 FF000000 AND ECX,0FF
004271EE . 890D 30E64500 MOV DWORD PTR DS:[45E6]
004271F4 . C1E1 08 SHL ECX,8
004271F7 . 03CA ADD ECX,EDX
004271F9 . 890D 2CE64500 MOV DWORD PTR DS:[45E6]
004271FF . C1E8 10 SHR EAX,10
00427202 . A3 28E64500 MOV DWORD PTR DS:[45E6]
00427207 . E8 94210000 CALL 004293A0
0042720C . 85C0 TEST EAX,EAX
0042720E . 75 0A JNZ SHORT 0042721A
00427210 . 6A 1C PUSH 1C
00427212 . E9 40910000 CALL 004573C0

```

Si miramos el salto vemos que ahora tenemos alli el codigo faltante.



Address	Disassembly	Comment
00A84865	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
00A84868	E9 6C299AFF	JMP 004271DC
00A84870	6C	INS BYTE PTR ES:[EDI],DX
00A84871	FF15 84094600	CALL DWORD PTR DS:[460984]
00A84877	E9 C8299AFF	JMP 00427244
00A8487C	C8 FF1580	ENTER 15FF,80
00A84880	0946 00	OR DWORD PTR DS:[ESI],EAX
00A84883	E9 532A9AFF	JMP 004272DB
00A84888	53	PUSH EBX
00A84889	8D05 E4E5F1B3	LEA EAX,DWORD PTR DS:[B3F1E5E4]
00A8488F	8D80 261A0E4C	LEA EAX,DWORD PTR DS:[EAX+4C0E1A26]
00A84895	E9 562A9AFF	JMP 004272F0
00A8489A	56	PUSH ESI
00A8489B	FF15 9C0B4600	CALL DWORD PTR DS:[460B9C]
00A848A1	E9 562A9AFF	JMP 004272FC
00A848A6	56	PUSH ESI
00A848A7	68 FF000000	PUSH 0FF

Si la imagen quedo muy grande y se agrando mucho el archivo se puede parchear todo lo que necesitamos y luego empacarlo con UPX que lo dejara pequeño y funcional y podremos desempacarlo cuando querramos ya el GUIPEX tiene la opcion de empacar y desempacar asi que lo tenemos bien dominado, aunque aquí no es necesario ya que el dumpeado, reparado y con la seccion agregada y la memoria intermedia allocada quedo en menos de 500k.

Otra opcion para los que quieren estudiar otra forma de reparar el antidump, es agregar la o las seccion faltante a continuacion de la ultima para que no se agrande mucho el exe, y luego cambiar el oep en el dumpeado y hacer un injerto para ubicarla en su posicion creandola con VirtualAlloc y copiando los bytes que agregamos alli, este metodo esta muy bien explicado por marciano en el ultimo concurso de crackslatinos si alguien quiere mirarlo se los recomiendo pues cuanto mas metodos conozcamos mejor, el link es.

[http://www.ricnar456.dyndns.org/CRACKING/NUEVO%20CURSO/CONCURSOS%20Y%20CO%20LABORACIONES/CONCURSOS/CONCURSOS%202004-2005/CONCURSO%2078/PELock%20v1.06%20\(All%20protections\)%20-%20por%20marciano.rar](http://www.ricnar456.dyndns.org/CRACKING/NUEVO%20CURSO/CONCURSOS%20Y%20CO%20LABORACIONES/CONCURSOS/CONCURSOS%202004-2005/CONCURSO%2078/PELock%20v1.06%20(All%20protections)%20-%20por%20marciano.rar)

user y pass:hola

En dicho unpackme de pelock que es parecido al que yo hice, la tabla esta redireccionada que era lo que yo queria tambien reparar con el script, pero mi version no tiene la IAT redireccionada por lo cual aplicaremos el script en una futura ocasión ya que habra miles de packers que lo necesiten.

Alli marciano utiliza el otro metodo que es hacer un injerto que cargue la seccion faltante en vez de cambiar el Virtual Offset, muchas veces yo he utilizado ese metodo y es muy bueno, aunque si son muchas las seccion faltantes es un poco cansador ya que hay que agregar una a una, con el metodo de agregarlo con el lordpe nos queda habilitada toda la seccion de memoria entre la ultima seccion y la seccion faltante.

De esta forma con el metodo del PeEditor conviene agregar siempre primero la seccion mas lejana, ya que de esta forma, al reparar con el LORDPE, toda la memoria entre la ultima seccion y el inicio de la seccion faltante quedara allocada sin problemas y podremos copiar del original detenido en el OEP y pegar los bytes en el dumpeado y luego guardar los cambios en este ultimo de cualquier seccion intermedia faltante.

Creo que es claro el ejemplo de que es lo que ocurre al agregar la seccion mas lejana, si esta empieza en 0a80000 como vemos en la imagen de abajo del mapa de memoria del original, el lordpe agranda la seccion anterior el dumpeado para que termine justo antes de 0a80000. Lo cual si nos falta alguna seccion mas antes de a80000 como las que tenemos en rosado, la podemos copiar y pegar facilmente y guardar los cambios.

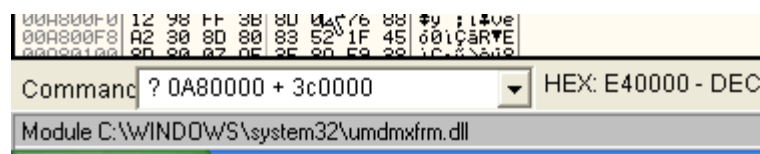
## MAPA DE MEMORIA DEL ORIGINAL

003F0000	00001000				Priv	RWE	RWE	
003A0000	00001000	UnPackMe		PE header	Priv	RW	RW	
00400000	00001000			code	Imag	R	RWE	
00401000	00004000	UnPackMe	.teddy		Imag	R	RWE	
0044B000	0000C000	UnPackMe	.teddy		Imag	R	RWE	
00457000	00009000	UnPackMe	.teddy		Imag	R	RWE	
00460000	00003000	UnPackMe	.teddy		Imag	R	RWE	
00463000	00008000	UnPackMe	.teddy	resources	Imag	R	RWE	
0046B000	0000A000	UnPackMe	.teddy	SFX, imports	Imag	R	RWE	
00480000	00021000				Priv	RW	RW	
004B0000	0000B000				Map	R E	R E	
00570000	00002000				Map	R E	R E	
00580000	00103000				Map	R	R	
00690000	00001000				Priv	RW	RW	
006A0000	00176000				Map	R E	R E	
009A0000	00001000				Priv	RW	RW	
009B0000	00004000				Priv	RW	RW	
009C0000	00003000				Map	R	R	
009D0000	00001000				Priv	RW	RW	
00A50000	00004000				Priv	RW	RW	
00A60000	00003000				Priv	RW	RW	
00A70000	00002000				Map	R	R	
00A80000	0003C000				Priv	RW	RW	
00A90000	00001000				Priv	RW	RW	
00B00000	00002000				Priv	RW	RW	

## MAPA DE MEMORIA DEL DUMPEADO

003F0000	00002000				Map	R	R	
00400000	00001000	dumpeado		PE header	Imag	R	RWE	
00401000	00004000	dumpeado	.teddy	code	Imag	R	RWE	
0044B000	0000C000	dumpeado	.teddy		Imag	R	RWE	
00457000	00009000	dumpeado	.teddy		Imag	R	RWE	
00460000	00003000	dumpeado	.teddy		Imag	R	RWE	
00463000	00008000	dumpeado	.teddy	resources	Imag	R	RWE	
0046B000	0000A000	dumpeado	.teddy		Imag	R	RWE	
00475000	0000B000	dumpeado	.mackt	imports	Imag	R	RWE	
00A80000	0003C000	dumpeado	seccion	data	Imag	R	RWE	
00AC0000	0000B000				Map	R E	R E	
00B80000	00002000				Map	R E	R E	

Alli se ve que cualquier seccion del original anterior a A80000 existe en el dumpeado y esta allocada ya que LORDPE agrando la seccion mackt, hasta hacerla contigua a la que agregamos, con los cual abarcamos en forma continua y tenemos allocada toda la memoria desde 401000 hasta  $A80000 + 3c0000$  (largo de la ultima seccion)= 0E40000



O sea cualquier seccion que vaya entre 401000 y E40000 ya la tenemos allocada en el dumpeado si necesitamos agregar bytes solo copiando y pegando del original detenido en el OEP al dumpeado y guardando los cambios ya lo tendremos corriendo.

Veremos ejemplos usando ambos metodos, ya que ambos son muy buenos, asi que ahora tienen dos tutes para leer, este y el de marciano, y practiquen ambos, porque hay veces que es mejor usar uno y en otras es mejor usar el otro según las circunstancias.

13/04/06