



Newbie Vs UnknowPacker - Revisitado - by El Cid



Newbie Vs UnknowPacker - Revisitado - by El Cid

Programa	L2Walker v10.23
Download	http://ricardonarvaja.info/WEB/CONCURSOS_VIEJOS/CONCURSOS_2004-2006/CONCURSO_33/
Descripción	
Herramientas	OllyDbg v2.0, OllyDbg v1.10, ImpRect, Cerebro, Atención.
Dificultad	Baja-Media
Compresor	Yoda's Crypter (v1.2-1.3)
Protección	Packer
Objetivos	Desempacado y Reconstrucción automática de la IAT por el método del <u>JMP-CALL MÁGICO</u>
Fecha:	16/10/2010
Autor	El Cid
Referencias	<ol style="list-style-type: none">1. NCR/CRC! [ReVeRsEr]: Newbie_Vs_UnknowPacker, Partes I a V2. R. Narvaja: INTRODUCCION AL CRACKING CON OLLYDBG PARTE 333. Formato PE (por Mark Pietrek)4. Windows Win32 API reference



Newbie Vs UnknowPacker – Revisitado – by El Cid

NOTA PREVIA


Disclaimer

- Este documento puede verse, en una primera lectura, como un sistema de debuggear un programa y del sistema de protección del packer Yoda's. Ésa es sólo una visión parcial de la realidad. La información contenida en el presente documento, sirve igualmente para establecer medidas anti-debugging más adecuadas y eficaces, para el estudio técnico de los citados sistemas de protección, su funcionalidad y eficacia. Igualmente es un banco de trabajo para el análisis de la estructura de los programas en Windows, concepto y estructura del formato PE y un sin fin de aplicaciones adicionales que sería prolijo referir aquí.
- Sin embargo, dado que todo lo anterior solo tiene como objetivo el aprendizaje y la práctica de la programación bajo Windows, y resto de los elementos citados antes, bajo un prisma exclusivamente didáctico y técnico – científico, si pensais utilizar el programa bajo una base comercial, debeis comprarlo y abandonar ahora mismo la lectura del presente documento.
- En absoluto me hago responsable del mal uso que cada cual pueda hacer de la información técnica facilitada.



INDICE

1. Cuestión Previa: ¿Por qué este tuto?	4
2. Recordatorio: ¿en qué consiste el método del JMP- CALL Mágico?	5
3. Herramientas usadas	8
4. Datos de partida y recordatorio de la localización del OEP	9
5. Localización de la IAT	11
6. Inspección de la IAT	15
7. Primeros pasos	19
8. Localización y formación de nombres y direcciones de las APIs en entradas buenas de la IAT	26
8.1. Datos de las DLLs	26
8.2. Localización y descryptado de APIs correspondientes a entradas buenas de la IAT	32
8.3. Escritura de la dirección de una API en una zona buena de la IAT	38
8.4. Análisis del cambio de zona: de Buena a Mala	39
9. Localización y descryptado de APIs correspondientes a entradas malas de la IAT	43
10. Estrategia para la reparación de las entradas malas	48
11. Injerto del programa	51
11.1. Reparación de la primera zona Mala de la IAT	55
11.2. Construcción de la IAT completa	58
12. Dumpeado y reparación final	68
13. Notas finales	72

	<p align="center"><u>Newbie Vs UnknowPacker – Revisitado – by</u> <u>El Cid</u></p>
--	---

1. Cuestión Previa ¿Por qué este tuto?

Quizá alguien se pregunte ¿Por qué un tuto sobre un programa con un packer ya resuelto?

Permitidme esta pequeña digresión a modo de explicación.

Debo admitir que los packers no han sido nunca “*santo de mi devoción*”. Hay otras facetas de la Ingeniería Inversa que me atraen mucho más y me parecen más interesantes.

Sin embargo trato de verlos de vez en cuando, para seguirles un poco la pista aunque sea a distancia.

Con esta idea revisé la documentación disponible y me llamó la atención un conjunto de 5 tutos:

Newbie_Vs_UnknowPacker Partes 1 a 5

, escritos y muy bien por cierto, por NCR (Teorías [446](#), 447, 452, 453 y 472).

El nombre me animó, ya que de él, parecía deducirse que estaban escritos para los no expertos como yo y que además debían estar explicados concienzudamente ya que tenía 5 partes, que era lo que yo quería, para no tener que pensar demasiado por mi cuenta. No sabía lo que me esperaba, que tampoco fue para tanto pero, desde luego, sí me hizo pensar (y aprender).

Bueno el caso es que me fui empapando la parte 1, la 2 y la 3.

Al acabar ésta tercera parte, la cosa ya estaba bastante clara al menos en el plano conceptual. Se había encontrado el OEP (Original Entry Point) y sólo restaba reparar la IAT (Import Address Table), es decir, reponer las entradas redireccionadas por el packer, de manera que contuvieran las direcciones reales de las APIs y no las creadas por el packer. Con ello podríamos después dumper y tener la versión desempacada del programa.

En ese momento, como me remordía un poco la conciencia por lo poco que había hecho por mí mismo hasta entonces, decidí que éso (es decir reparar la IAT) bien lo podía hacer yo y que si me atascaba pues siempre podría mirar en las partes 4 y 5 siguientes y ¡listo! Así que me puse manos a la obra.

Lo primero fue pensar cómo lo podía hacer. Hacerlo a mano era impensable, dado el gran número de entradas redireccionadas que tiene la IAT. Utilizar otro tipo de herramientas automáticas tampoco me gusta, cuando son tantas entradas, porque no te enteras de casi nada de lo que ocurre y además, las más de las veces, se cuelga el equipo. Además y por si fuera poco al final casi siempre hay que hacer retoques a mano y entonces ya no compensa tanto.

Me quedaba un método clásico, que suele ser de aplicación general (que a mí son las cosas que más me interesan, no me seducen los “truquitos” que sólo valen para un caso) y que es muy automático si se da con el quid. El método en cuestión no es ni más ni menos que el llamado y muy conocido:

Método del JMP-CALL MÁGICO ([Refª 2](#))

Este sistema lo había estudiado bastante en el pasado y me parecía que ofrecía buenas posibilidades. Así que me puse a ello y al cabo de varias horas de trabajo analizando el código del programa, había averiguado cosas, sí, pero no había conseguido dar con el meollo de la cuestión ni mucho menos resolver el packer, por lo



Newbie Vs UnknowPacker - Revisitado - by El Cid

que decidí echar una miradita a los 2 últimos ficheros de NCR esperando que se aclararan mis dudas.

¡Cual no fue mi sorpresa al ver que el sistema usado en ellos no tenía nada que ver con el que yo estaba intentando! Mi gozo en un pozo como se suele decir. En esa situación sólo tenía dos opciones: abandonar, conformándome con lo visto y averiguado hasta ese momento o tratar de solucionar definitivamente el packer por el método que yo estaba intentando.

Como podeis intuir, esto último es lo que hice y lo que sigue es el resultado.

2. Recordatorio: ¿en qué consiste el método del JMP-CALL Mágico?

2.1. La IAT

Como se sabe la IAT, es una tabla que contiene las direcciones de las funciones (APIs) que el programa importa de las DLLs. Son las funciones que necesita para operar, porque las llamará en el transcurso de la ejecución del mismo. Funciones tales como Abrir ficheros, Cerrarlos y muchas cosas más.


Aquí teneis una imagen de un trozo de la IAT de un programa, con una serie de APIs, sus direcciones en mi equipo y las DLL a que pertenecen.

77DA6C07	ADVAPI32.RegCloseKey
00000000	
58C74834	COMCTL32.ImageList_LoadImageW
58C40205	COMCTL32.ImageList_Create
58C403D8	COMCTL32.ImageList_Destroy
58C365CF	COMCTL32.InitCommonControls
58C4DFF1	COMCTL32.ImageList_Draw
58C3C7F4	COMCTL32.ImageList_ReplaceIcon
58C41FF8	COMCTL32.ImageList_AddMasked
58C72EC5	COMCTL32.ImageList_Add
58C51226	COMCTL32._TrackMouseEvent
58C72FF1	COMCTL32.ImageList_GetImageInfo
58C6229A	COMCTL32.ImageList_GetIcon
00000000	
77EF7786	GDI32.CreateRectRgn
77EFB479	GDI32.GetPixel
77EF7AA0	GDI32.SelectClipRgn
77EF8B03	GDI32.SetTextAlign
77EFA0FB	GDI32.MoveToEx
77EFD9F7	GDI32.LineTo
77EFD749	GDI32.GetRgnBox
77EFA237	GDI32.CreatePolygonRgn
77EFE0E3	GDI32.Polyline
77EF6A56	GDI32.IntersectClipRect
77EFAAF3	GDI32.GetDIBits
77EF8494	GDI32.GetTextCharsetInfo
77EF8EF7	GDI32.GetTextColor
77EFEE05	GDI32.Polygon
77EF8EA3	GDI32.GetBkColor
77EFE671	GDI32.DPtoLP
77EF8E34	GDI32.GetMapMode
77EF9C2D	GDI32.CombineRgn
77EF9DA4	GDI32.SetRectRgn
77EF85E3	GDI32.PatBlt

En los programas que no están empacados, todas las entradas de la IAT son similares a las de arriba y contienen la dirección real de una API, en el equipo en que se ejecuta el programa.

Sin embargo en los programas empacados, algunas de las entradas son sustituidas por el packer que las cambia y las deriva a otras posiciones suyas (creadas por él en tiempo de ejecución en Secciones privadas normalmente) y que finalmente terminan, claro, en las direcciones de la API correspondiente.

En este caso, nuestro trabajo consiste en reemplazar, reparar, cada una de esas entradas redireccionadas, malas las llamaremos por simplificar, por el correspondiente valor que sea la dirección real de la API de que se trate, que llamaremos buena.

	<p align="center"><u>Newbie Vs UnknowPacker - Revisitado - by</u> <u>El Cid</u></p>
---	---

2.2. Formas de reparar la IAT

Sólo citaré 2:

- Método manual, al que también podemos denominar como “de chinos”, “heroico” o con apelativos similares.

Consiste en ir trazando (mediante F7 / F8) cada una de las entradas malas de la IAT hasta llegar a la API en cuestión y a continuación substituir en la IAT, el valor existente por la dirección real de la API. No es un método practicable para programas medianos o grandes. Pensemos en que un programa puede tener fácilmente 200-300 entradas redireccionadas en la IAT o más. Podemos estimar que para saber en qué API termina cada llamada, haya que pulsar unas 30 veces la tecla F7 (y me quedo corto en ocasiones), con lo que estaríamos hablando de unas 6000-9000 pulsaciones de esta tecla, lo que no parece razonable. Eso sin contar con que podríamos cometer errores de pulsación que nos arruinarían el trabajo en cualquier momento, que el programa podría incorporar técnicas antitraceo, detección de BPs en entrada de APIS, etc.

Este sistema sólo es válido para realizar prácticas de aprendizaje, para ver cómo se redirecciona una API o para casos muy concretos, en que haya que reparar sólo unas pocas entradas.

- Método del JMP-CALL Mágico

Es un método muy interesante, muy elegante conceptualmente y prácticamente automático, es decir con muy poca intervención manual. Como dato os adelanto que la reparación automática completa de la IAT de este programa, sólo ha requerido 34 bytes de código: 24 para el injerto y 10 simplemente de llamada al mismo. No me digais que no es interesante (¿excitante?) pensar que con sólo 34 bytes tenemos todo arreglado. A mí por lo menos sí me lo parece.

Pero es que hay más, porque de esos 24 bytes del injerto, 19 son pura copia de los del programa original y los otros 5 son un JMP para volver al programa donde lo dejamos. O sea que realmente nosotros lo que aportamos de nuestra cosecha esprácticamente NADA. Bueno sí, es como el buen mecánico de un coche que tocando un tornillito de nada, hace que el motor marche como un reloj suizo, que lo difícil no es tocar el tornillo sino saber qué tornillo hay que tocar cuando se levanta la tapa del motor.

Y ese es ahora nuestro trabajo: averiguar donde hay que tocar el programa mínimamente para que nos produzca el efecto deseado (arreglar la IAT).

2.3. Breve descripción del Método del JMP-CALL Mágico

Hemos dicho antes que los packers cambian algunas entradas de la IAT y las redireccionan a posiciones suyas, en lugar de apuntar a las direcciones reales de las APIs. Aquí vemos un ejemplo de ambos tipos de entradas de un trozo de una IAT:



Newbie Vs UnknowPacker - Revisitado - by El Cid

EF	61	EF	77	29	5E	EF	77	77	50	EF	77	A1	6A	EF	77
B1	A7	EF	77	C1	61	EF	77	E2	A8	EF	77	74	78	EF	77
EC	D1	F1	77	B0	6E	F0	77	0B	D1	F1	77	59	6F	F0	77
77	C0	EF	77	4C	7B	EF	77	36	6B	F0	77	1B	82	EF	77
17	60	F2	77	00	00	00	00	00	00	B3	00	20	00	B3	00
40	00	B3	00	60	00	B3	00	80	00	B3	00	A0	00	B3	00
C0	00	B3	00	E0	00	B3	00	00	01	B3	00	20	01	B3	00
40	01	B3	00	60	01	B3	00	80	01	B3	00	A0	01	B3	00
C0	01	B3	00	E0	01	B3	00	00	02	B3	00	20	02	B3	00
40	02	B3	00	60	02	B3	00	80	02	B3	00	A0	02	B3	00
C0	02	B3	00	E0	02	B3	00	00	03	B3	00	20	03	B3	00
40	03	B3	00	60	03	B3	00	80	03	B3	00	A0	03	B3	00
C0	03	B3	00	E0	03	B3	00	00	04	B3	00	20	04	B3	00
40	04	B3	00	60	04	B3	00	80	04	B3	00	A0	04	B3	00

Las entradas verdes son normales o buenas (son direcciones reales de APIs) que el programa ha calculado y colocado en la tabla y las azul oscuro son redireccionadas o malas (son direcciones de zonas del packer donde él luego calcula la dirección de la API real) que el programa ha puesto ahí y que después desviará a sus secciones donde calculará la dirección real. La azul claro, como sabemos, es la separación entre ambas.

Pero entonces espera, puede decir alguien, ¿quiere eso decir que el programa escribe tanto las direcciones buenas como las malas? La respuesta es un rotundo SÍ.

De alguna manera él **decide** en unos casos poner en las entradas IAT, direcciones reales (buenas) de APIs y en otros poner valores que apuntan a sus zonas (malas).

He subrayado la palabra “decide” porque es la clave de la cuestión, ya que normalmente ¿cómo decide un programa entre dos opciones alternativas?

Pues mediante el consabido IF seguido de los JMPs correspondientes, es decir sería algo así como:


```
IF      (esta DLL es de entradas buenas) → JMP destino_bueno
ELSE → JMP destino_malo
```

Disculpadme por el pseudocódigo macarrónico, pero creo que se entiende.

Debo aclarar que las entradas buenas y malas no están mezcladas normalmente dentro de la misma DLL. Quiere eso decir que las direcciones correspondientes a una DLL serán todas buenas o todas malas. De ahí el esquema del pseudocódigo de arriba.

Entonces creo que es inmediato darse cuenta de que si somos capaces de detectar el lugar del programa en que se bifurcan los tratamientos de salto bueno y salto malo, lo que deberemos hacer simplemente, es encaminar el tratamiento de todas las entradas de IAT por el lado bueno y el propio programa nos hará el trabajo de escribir la IAT completa con entradas buenas.

De la idea anterior proviene el nombre del método del JMP-mágico.

	<p align="center"><u>Newbie Vs UnknowPacker - Revisitado - by</u> <u>El Cid</u></p>
--	---

No me digais que la idea no es lógica, simple y al mismo tiempo atractiva y potente.

Para hacer lo anterior tenemos, al menos 2, sistemas:

- Parchear adecuadamente el salto concreto, si existe, que nos lleve siempre a la parte de las entradas buenas de la IAT y calcule y escriba en ella, las direcciones reales de las APIs.
- Engañar al packer haciéndole creer que todas las entradas son buenas. El resultado es el mismo que con el anterior pero nos ahorramos tener que buscar el “salto” propiamente dicho.

El primer sistema es el más directo pero a veces no es tan fácil de implementar porque puede que ni siquiera exista un único JMP, JNE, JE o similar que discrimine entre ambas opciones. Desde luego, si existe y lo encontramos, es lo mejor pues bastaría con cambiar 1 ó 2 bytes para tener el trabajo hecho.

Pero a veces es difícil o simplemente el código de ambas opciones está tan imbricado y entrelazado que se ven implicados varios saltos y la lógica puede no estar clara. Tengamos en cuenta que los diseñadores del programa tratan de ofuscar estas instrucciones.

En estos casos es a veces más fácil buscar elementos que nos permitan engañar al programa haciéndole creer que todas las entradas son buenas y sea él quien se encargue luego de tomar el camino adecuado.

Se podría decir entonces, que se trata del método del salto mágico a ciegas, es decir sin saber realmente dónde está, lo que parece más mágico todavía.

Ya vereis que todo es bastante simple.


En este caso además, los nombres de las APIs están codificados, lo que ha sido un pequeño inconveniente al principio pero después nos ha servido de ayuda para localizar y distinguir mejor el camino malo del camino bueno.

3. Herramientas usadas

Son las clásicas en estos casos, es decir: OllyDbg v1.1, v2.0 e Import Reconstructor. Sin embargo debemos hacer un comentario respecto de Olly. ¿Por qué se dice que se van a usar los dos Ollys? Por lo siguiente.

Durante el análisis del programa se usan muchos MBPs *simultáneos* con lo que la utilización de Olly v2.0, es casi obligatoria. No quiero ni imaginar lo que hubiera sido trabajar con el 1.1 con sólo un MBP. Suerte que el programa no utiliza sistemas antidebugging de manera intensiva, ya que con la v2.0 de Olly, que no admite plugins, hubiera sido costoso contrarrestarlas. Sólo nos deberemos preocupar en el inicio de cada Run, de poner a cero binario el Byte de IsDebuggerPresent, es decir el apuntado por [EBX+2], porque éso sí lo comprueba y si no lo hacemos se termina la ejecución y nos sacará del programa.

Hay además a lo largo del código alguna comprobación de BPs en la entrada de las APIs que hay que evitar (ya veremos dónde y cómo) y también comprueba el entorno de depuración por medio de la instrucción RDTSC, haciéndonos caer en bucles infinitos si no estamos atentos.

	<p align="center"><u>Newbie Vs UnknowPacker - Revisitado - by</u> <u>El Cid</u></p>
--	---

Una vez reparada la IAT usaremos la v1.10 para los dumpeados, ya que con la v2.0 no es posible hacerlos que yo sepa.

4. Datos de partida y recordatorio de la localización del OEP

Daremos por conocido el OEP y la forma de llegar a él que básicamente se trata del método de las excepciones.

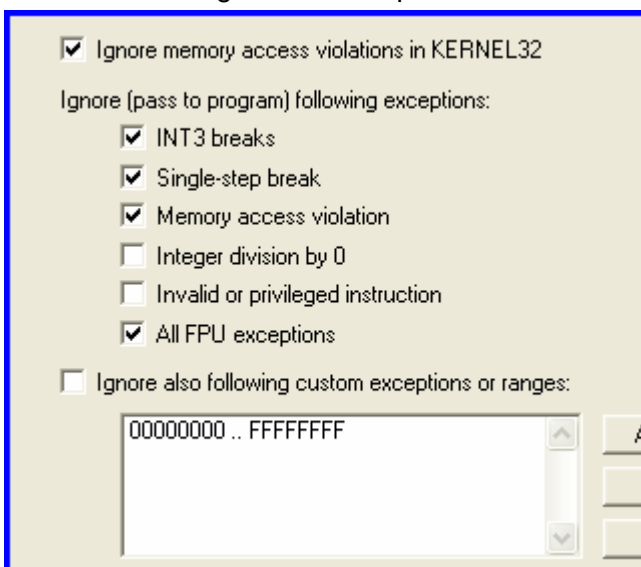
Si quereis conocer los detalles del proceso leed los 3 tutos de NCR citados.

Como recordatorio rápido de ello, expondremos aquí brevísimamente la forma de llegar al OEP.

Usaremos por ahora Olly v1.10 (porque se llega más fácilmente al OEP que con la v2.0).

Configuramos Olly así:

Marcamos las siguientes excepciones:



Ésto lo hacemos, porque sabemos del estudio de NCR, que la última excepción antes de llegar al OEP, no es de ninguno de estos tipos, por lo que marcamos las casillas correspondientes para saltarlas. Así llegaremos antes al OEP. Si no lo supiéramos, quitaríamos todas las marcas y pararíamos muchas más veces pero el sistema es el mismo.

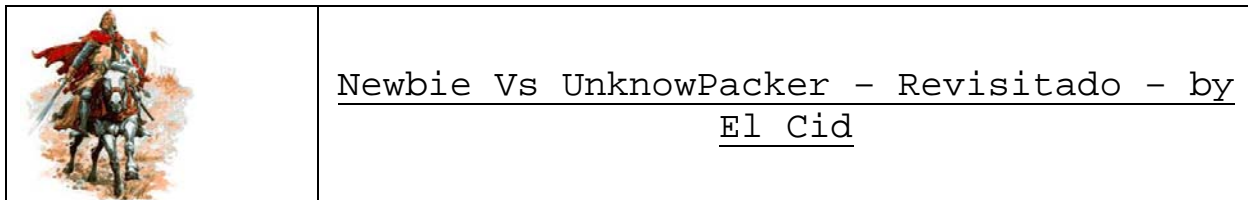
En cuanto a plugins, el único que es imprescindible, aunque más tarde, es OllyDump aunque yo también tengo puestos Hide Debugger y CommandBar que siempre viene bien.

Empezamos.

Cargamos el programa en Olly v1.10. **Damos Run**

Cada vez que paramos en una excepción, la pasamos con Shift+F9 y continuamos.

En mi caso a la cuarta vez que para, estamos aquí:



Address	Hex dump	Disassembly
00393DE0	6285 443D4000	BOUND EAX, QWORD PTR SS:[EBP+403D44]
00393DE6	EB F8	JMP SHORT 00393DE0
00393DE8	8B08	MOV EBX, EAX
00393DEA	81C4 00010000	ADD ESP, 100
00393DF0	6A 00	PUSH 0
00393DF2	53	PUSH EBX
00393DF3	8D85 BF244000	LEA EAX, DWORD PTR SS:[EBP+4024BF]
00393DF9	50	PUSH EAX
00393DFA	8B85 46384000	MOV EAX, DWORD PTR SS:[EBP+403846]
00393E00	E9 FA050000	JMP 003943FF
00393E05	E9 89853B37	JMP 3774C393

Este punto, como ya he hecho previamente los tutos de NCR, sé que es justamente el anterior a llegar al OEP. Por ello pongo ahora un Break on Access en la sección de código del programa:

003D0000	00005000				Priv	RW	RW	
003E0000	00003000				Map	R	R	\Device\Harddi
003F0000	00002000				Priv	RW	RW	
00400000	00001000	L2Walker	PE header		Imag	RW	RWE	
00401000	0001F000	L2Walker	code					
00620000	00022000	L2Walker	data, r					
00642000	00008000	L2Walker	SFX, in					
00650000	0000A000							
00710000	00002000							
00720000	00103000							
00830000	0011A000							
00B30000	00001000							
00B40000	00002000							
00B50000	00001000							
00BD0000	00005000							
00BE0000	00001000							
00BF0000	00002000							
00C00000	00003000							

Y doy Shift+F9, aterrizando aquí:

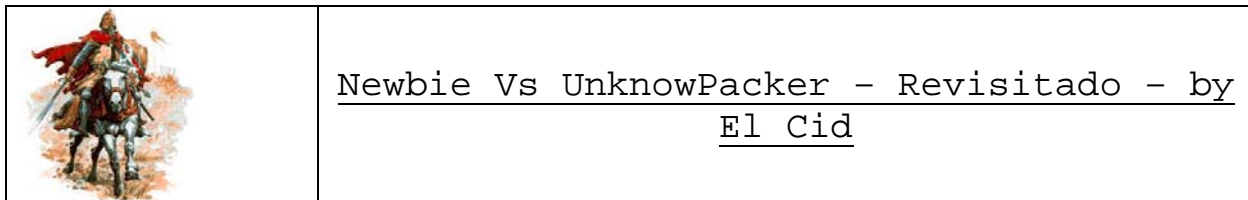
Address	Hex dump	Disassembly
00451715	6A 60	PUSH 60
00451717	68 C8665500	PUSH L2Walker.005566C8
0045171C	E8 07290000	CALL L2Walker.004540F8
00451721	BF 94000000	MOV EDI, 94
00451726	8BC7	MOV EAX, EDI
00451728	E8 23140000	CALL L2Walker.00452B50
0045172D	8965 E8	MOV DWORD PTR SS:[EBP-18], ESP
00451730	8BF4	MOV ESI, ESP
00451732	893E	MOV DWORD PTR DS:[ESI], EDI
00451734	56	PUSH ESI
00451735	FF15 6CD24D00	CALL NEAR DWORD PTR DS:[4DD26C]
0045173B	8B4E 10	MOV ECX, DWORD PTR DS:[ESI+10]
0045173E	890D A4DE6100	MOV DWORD PTR DS:[61DEA4], ECX
00451744	8B46 04	MOV ECX, DWORD PTR DS:[ESI+4]

Éste es el OEP

Si no supiéramos que ésa era la última excepción antes del OEP, hubiéramos visto que al dar Shift+F9 desde ese punto, arrancarí el programa, con lo que reiniciando y repitiendo el proceso, al llegar a ella ahora sí sabríamos que era la última y pondríamos el Set Break-On-Access dicho.

Vamos a continuar haciendo algunas averiguaciones con Olly v1.10 antes de pasar a la v2.0.

Como estamos parados en el OEP, el packer ha tenido que generar ya todas las direcciones de las APIs, buenas o malas, para que el programa las pueda llamar cuando comience su ejecución. Por tanto vamos a buscarlas que es equivalente a buscar la IAT.



5. Localización de la IAT

(Nota: quien sepa localizarla por sí mismo puede pasar al [Apartado 7](#))

El primer paso para reparar la IAT es localizarla dentro del programa.

Sabemos que una forma de hacerlo es mirar en la cabecera del programa que está en formato PE.

Vamos a hacer ésto lo primero.

Si miramos en la ventana M de Olly, vemos que el Header del programa comienza en la posición 400000:

M Memory map									
Address	Size	Owner	Section	Contains	Type	Access	Initial	M.	
003C0000	00001000				Priv	RW	RW		
003D0000	00005000				Priv	RW	RW		
003E0000	00003000				Map	R	R		
003F0000	00002000				Priv	RW	RW		
00400000	00001000	L2Walker		PE header	Imag	RW	RWE		
00401000	0021F000	L2Walker		code	Imag	RW	RWE		
00620000	00022000	L2Walker		data,resource	Imag	RW	RWE		
00642000	00008000	L2Walker		SFX, imports	Imag	RW	RWE		
00650000	00007000				Map	R E	R E		

Así que hacemos: GoTo → Expression 400000

Y en la ventana de volcado hex, hacemos:



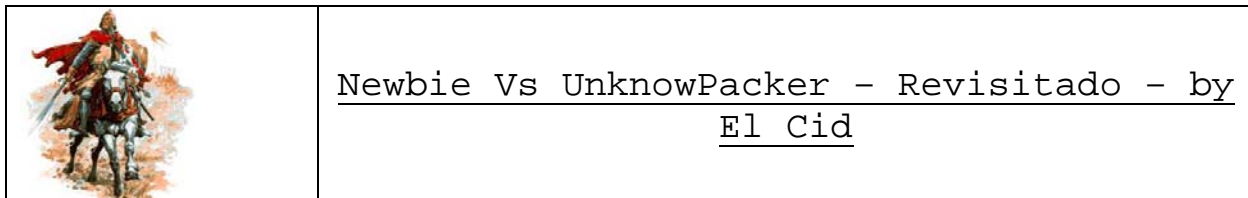
Y tenemos lo siguiente:

Address	Hex dump	Data	Comment
00400000	4D 5A	ASCII "MZ"	DOS EXE Signature
00400002	9000	DW 0090	DOS_PartPag = 90 (144.)
00400004	0300	DW 0003	DOS_PageCnt = 3
00400006	0000	DW 0000	DOS_ReloCnt = 0
00400008	0400	DW 0004	DOS_HdrSize = 4
0040000A	0000	DW 0000	DOS_MinMem = 0
0040000C	FFFF	DW FFFF	DOS_MaxMem = FFFF (65535.)
0040000E	0000	DW 0000	DOS_ReloSS = 0
00400010	B800	DW 00B8	DOS_ExeSP = B8
00400012	0000	DW 0000	DOS_ChkSum = 0
00400014	0000	DW 0000	DOS_ExeIP = 0
00400016	0000	DW 0000	DOS_ReloCS = 0
00400018	4000	DW 0040	DOS_Tabloff = 40
0040001A	0000	DW 0000	DOS_Overlay = 0
0040001C	00	DB 00	
0040001D	00	DB 00	
0040001E	00	DB 00	

Éste es el inicio de la cabecera del fichero del programa.

Sabemos que en el Offset "3C" hex (o sea en 40003C) se encuentra el valor del Offset de la "PE signature". Vamos allí a mirar, viendo ésto:

00400038	00	DB 00	
00400039	00	DB 00	
0040003A	00	DB 00	
0040003B	00	DB 00	
0040003C	00010000	DW 00000100	Offset to PE signature
00400040	0E	DB 0E	
00400041	1F	DB 1F	
00400042	BA	DB BA	
00400043	0E	DB 0E	



, o sea que el Offset es 100. Pues vamos a la posición 400100 y veremos ésto:

Address	Hex dump	Data	Comment
00400100	50 45 00 00	ASCII "PE"	PE signature (PE)
00400104	4C01	DW 014C	Machine = IMAGE_FILE_MACHINE_I386
00400106	0300	DW 0003	NumberOfSections = 3
00400108	39ABC041	DD 41C0AB39	TimeDateStamp = 41C0AB39
0040010C	00000000	DD 00000000	PointerToSymbolTable = 0
00400110	00000000	DD 00000000	NumberOfSymbols = 0
00400114	E000	DW 00E0	SizeOfOptionalHeader = E0 (224.)
00400116	0F01	DW 010F	Characteristics = EXECUTABLE_IMAGE 32BIT_MACHINE RELOCS_STRIP
00400118	0E01	DW 010E	Maginumber = PE32
0040011A	07	DB 07	MajorLinkerVersion = 7
0040011B	0A	DB 0A	MinorLinkerVersion = A (10.)
0040011C	00C00000	DD 000DC000	SizeOfCode = DC000 (901120.)
00400120	00501600	DD 00165000	SizeOfInitializedData = 165000 (1462272.)
00400124	00000000	DD 00000000	SizeOfUninitializedData = 0
00400128	EA9D2400	DD 00249DEA	AddressOfEntryPoint = 249DEA
0040012C	00000000	DD 00000000	BaseOfCode = 00000
00400130	00000000	DD 00000000	BaseOfData = 00000
00400134	00004000	DD 00400000	ImageBase = 400000
00400138	00100000	DD 00001000	SectionAlignment = 1000
0040013C	00020000	DD 00002000	FileAlignment = 2000
00400140	0400	DW 0004	MajorOSVersion = 4
00400142	0000	DW 0000	MinorOSVersion = 0
00400144	0000	DW 0000	MajorImageVersion = 0
00400146	0000	DW 0000	MinorImageVersion = 0
00400148	0400	DW 0004	MajorSubsystemVersion = 4
0040014A	0000	DW 0000	MinorSubsystemVersion = 0
0040014C	00000000	DD 00000000	Reserved
00400150	00A02400	DD 0024A000	SizeOfImage = 24A000 (2400256.)
00400154	00100000	DD 00001000	SizeOfHeaders = 1000 (4096.)
00400158	00000000	DD 00000000	Checksum = 0
0040015C	0200	DW 0002	Subsystem = IMAGE_SUBSYSTEM_WINDOWS_GUI
0040015E	0000	DW 0000	DLLCharacteristics = 0
00400160	00001000	DD 00100000	SizeOfStackReserve = 100000 (1048576.)
00400164	00100000	DD 00001000	SizeOfStackCommit = 1000 (4096.)
00400168	00001000	DD 00001000	SizeOfHeapReserve = 100000 (1048576.)
0040016C	00100000	DD 00001000	SizeOfHeapCommit = 1000 (4096.)
00400170	00000000	DD 00000000	LoaderFlags = 0
00400174	10000000	DD 00000010	NumberOfRvaAndSizes = 10 (16.)
00400178	00000000	DD 00000000	Export Table address = 0
0040017C	00000000	DD 00000000	Export Table size = 0
00400180	40202400	DD 00242040	Import Table address = 242048
00400184	2C010000	DD 0000012C	Import Table size = 12C (300.)
00400188	00002200	DD 00220000	Resource Table address = 220000
0040018C	001C0200	DD 00021C00	Resource Table size = 21C00 (138240.)
00400190	00000000	DD 00000000	Exception Table address = 0
00400194	00000000	DD 00000000	Exception Table size = 0
00400198	00000000	DD 00000000	Certificate File pointer = 0
0040019C	00000000	DD 00000000	Certificate Table size = 0
004001A0	00000000	DD 00000000	Relocation Table address = 0
004001A4	00000000	DD 00000000	Relocation Table size = 0
004001A8	00000000	DD 00000000	Debug Data address = 0

Vemos cómo nos indica Olly que es la PE signature y más abajo he marcado la dirección de la IT (Import Table) que no debemos confundir con la IAT que es la que buscamos.

Bueno el caso es que esa dirección es: $400000 + 242048 = 642048$. Pues vamos allí a mirar:

Address	Hex dump	ASCII
00642048	70 20 24 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00642058	70 20 24 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00642068	00 00 00 00 00 00 00 00 B0 AD 80 7C B1 B6 80 7C	00 00 00 00 00 00 00 00 B0 AD 80 7C B1 B6 80 7C

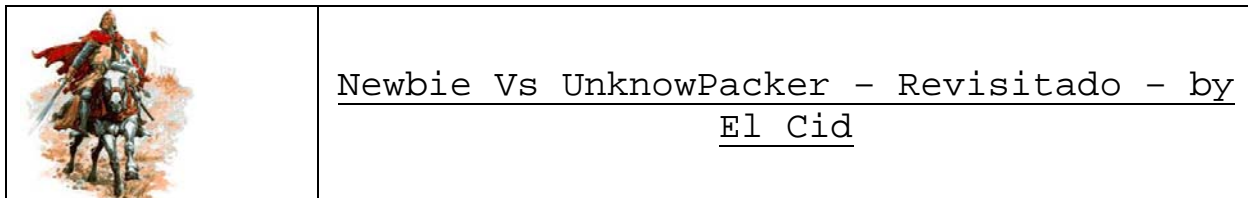
Sabemos que en esa dirección hay un grupo de 5 DWORDs y que la 5ª precisamente es la dirección (Offset) de la IAT.

Pues vamos allí (a $400000 + 242070 = 642070$) a mirar:

Address	Hex dump	ASCII
00642070	B0 AD 80 7C B1 B6 80 7C 77 10 80 7C 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00642080	4B 45 52 4E 45 4C 33 32 2E 64 6C 6C 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00642090	47 65 74 50 72 6F 63 41 64 64 72 65 73 73 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
006420A0	00 47 65 74 40 6F 64 75 6C 65 48 61 6E 64 6C 65	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
006420B0	41 00 00 00 4C 6F 61 64 4C 69 62 72 61 72 79 41	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
006420C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
006420D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Aquí tenemos lo que buscamos:

- Azul cyan: 3 direcciones de API
- Amarillo: campo separador

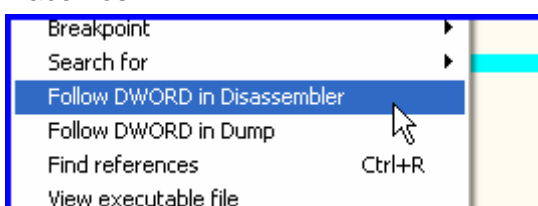


- **Naranja:** Nombre de la DLL involucrada
- Cuadro **rojo:** Nombres de las APIs

Si por ejemplo en la segunda DWord de la IAT:

Address	Hex dump
00642050	00 00 00 00 80 20 24 00 70 20 24 00 00 00 00
00642060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00642070	00 AD 80 7C B1 B6 80 7C 77 10 80 7C 00 00 00
00642080	4B 45 52 4E 45 4C 33 32 7E 64 6C 6C 00 00 00
00642090	47 65 74 50 72 6F 63 41 74 64 72 65 73 73 00
006420A0	00 47 65 74 4D 6F 64 75 6C 65 48 61 6E 64 6C

Hacemos:



En la ventana de código de Olly, vamos aquí y vemos que corresponde a la API GetModuleHandle que es precisamente el nombre que veíamos en la imagen de la IAT de un poco más arriba:

7C80B6B0	90	NOP
7C80B6B1	8BFF	MOV EDI, EDI
7C80B6B3	55	PUSH EBP
7C80B6B4	8BEC	MOV EBP, ESP
7C80B6B6	837D 08 00	CMPL DWORD PTR SS:[EBP+8], 0
7C80B6B8	74 18	JE SHORT kernel32.7C80B6D4
7C80B6BC	FF75 08	PUSH DWORD PTR SS:[EBP+8]
7C80B6BF	E8 C0290000	CALL kernel32.7C80E084
7C80B6C4	85C0	TEST EAX, EAX
7C80B6C6	74 08	JE SHORT kernel32.7C80B6D0
7C80B6C8	FF70 04	PUSH DWORD PTR DS:[EAX+4]
7C80B6CB	E8 7D2D0000	CALL kernel32.GetModuleHandleW
7C80B6D0	5D	POP EBP
7C80B6D1	C2 0400	RETN 4
7C80B6D4	64:A1 18000000	MOV EAX, DWORD PTR FS:[18]
7C80B6DA	8B40 30	MOV EAX, DWORD PTR DS:[EAX+30]
7C80B6DD	8B40 08	MOV EAX, DWORD PTR DS:[EAX+8]
7C80B6E0	EB EE	JMP SHORT kernel32.7C80B6D0
7C80B6E2	90	NOP
7C80B6E3	90	NOP



Newbie Vs UnknowPacker - Revisitado - by El Cid

Resumamos en una única imagen el camino seguido hasta la IAT a modo de recordatorio:

M Memory map

Address	Size	Owner	Section	Contains	Type	Access	Initial	M
003C0000	00001000				Priv	RW	RW	
003D0000	00005000				Priv	RW	RW	
003E0000	00003000				Map	R	R	
003F0000	00002000				Priv	RW	RW	
00400000	00001000	L2Walker		PE header	Image	RW	RWE	
00401000	0021F000	L2Walker		code	Image	RW	RWE	
00620000	00022000	L2Walker		data,resource	Image	RW	RWE	
00642000	00008000	L2Walker		SFX, imports	Image	RW	RWE	
00650000	00007000				Map	R	R	
00660000	00000000							

Address Hex dump Data Comment

00400000	40 5A	ASCII "MZ"	DOS EXE Signature
00400002	9000	DW 0090	DOS_PartPag = 90 (144.)
00400004	0300	DW 0003	DOS_PageCnt = 3
00400006	0000	DW 0000	DOS_ReloCnt = 0
00400008	0400	DW 0004	DOS_HdrSize = 4
0040000A	0000	DW 0000	DOS_MinMem = 0
0040000C	FFFF	DW FFFF	DOS_MaxMem = FFFF (65535.)
0040000E	0000	DW 0000	DOS_ReloSS = 0
00400010	B800	DW 00B8	DOS_ExeSP = B8
00400012	0000	DW 0000	DOS_ChkSum = 0
00400014	0000	DW 0000	DOS_ExeIP = 0
00400016	0000	DW 0000	DOS_ReloCS = 0
00400018	4000	DW 0040	DOS_Tabloff = 40
0040001A	0000	DW 0000	DOS_Overlay = 0
0040001C	00	DB 00	
0040001D	00	DB 00	
0040001E	00	DB 00	

Address Hex dump Data Comment

00400038	00	DB 00	
00400039	00	DB 00	
0040003A	00	DB 00	
0040003B	00	DB 00	
0040003C	00010000	DD 00000100	Offset to PE signature
00400040	0E	DB 0E	
00400041	1F	DB 1F	
00400042	BA	DB BA	
00400043	0E	DB 0E	

Address Hex dump Data Comment

00400100	50 45 00 00	ASCII "PE"	PE signature (PE)
00400104	4C01	DW 014C	Machine = IMAGE_FILE_
00400106	0300	DW 0003	NumberOfSections = 3
00400108	39AB041	DD 41C0AB39	TimeDateStamp = 41C0
0040010C	00000000	DD 00000000	PointerToSymbolTable
00400110	00000000	DD 00000000	NumberOfSymbols = 0

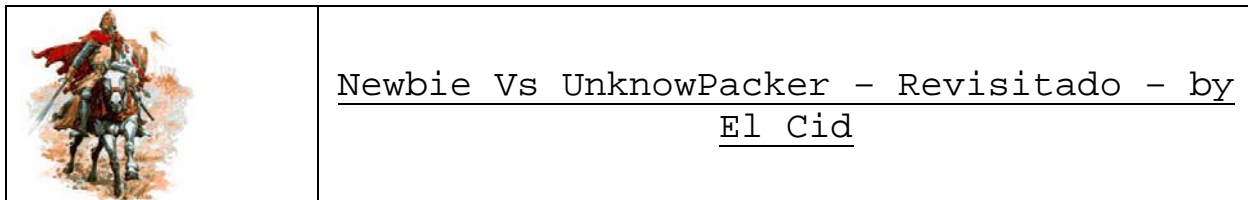
Address Hex dump Data Comment

00400178	00000000	DD 00000000	Export Table address = 0
0040017C	00000000	DD 00000000	Export Table size = 0
00400180	48202400	DD 00242048	Import Table address = 242048
00400184	2C010000	DD 0000012C	Import table size = 12C (300)
00400188	00002200	DD 00220000	Resource Table address = 220000
0040018C	001C0200	DD 00021C00	Resource Table size = 21C00 (1382)
00400190	00000000	DD 00000000	Exception Table address = 0
00400194	00000000	DD 00000000	Exception Table size = 0

Address Hex dump

00642048	70 20 24 00 00 00 00 00 00 00 00 80 20 24 00	
00642058	70 20 24 00 00 00 00 00 00 00 00 80 20 24 00	
00642068	00 00 00 00 00 00 00 00 B0 AD 80 7C B1 B6 80 7C	

Dirección IAT



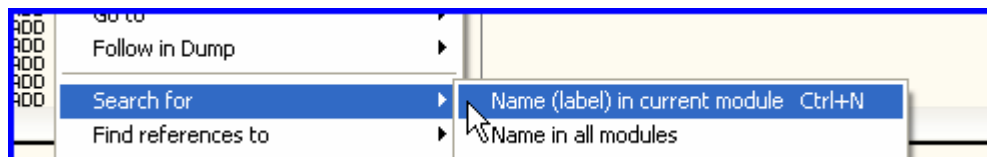
6. Inspección de la IAT

Vamos a examinar un poco la IAT encontrada. De nuevo repetimos la imagen:

Direcciones de APIs		Separador		Librería DLL		Nombres de APIs	
Address	Hex dump						ASCII
00642070	B0 AD 80 7C B1 B6 80 7C 77 10 80 7C 00 00 00 00						...C:\w...
00642080	4B 45 52 4E 45 4C 33 32 2E 64 6C 6C 00 00 00 00						KERNEL32.dll...
00642090	47 65 74 50 72 6F 63 41 64 64 72 65 73 00 00 00						GetProcAddress...
006420A0	00 47 65 74 40 6F 64 75 6C 65 48 61 6E 64 6C 65						.GetProcAddress...
006420B0	41 00 00 00 4C 6E 61 64 4C 69 62 72 61 72 79 41						...LoadLibraryA...
006420C0	00 00 00 00 00 00 00 00 00 00 00 00 00 60 E8 00					
006420D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00					

Vemos que tenemos 3 entradas, luego habrá 3 APIs, lo cual nos parece bien poco. Si nos fijamos en los nombres de las APIs, no son cualquiera, sino que con ellas podríamos obtener todas las demás y éso es lo que va a hacer el packer precisamente y además crear la tabla en otro lugar. En realidad lo habrá hecho ya, puesto que estamos en el OEP.

Si hacemos:



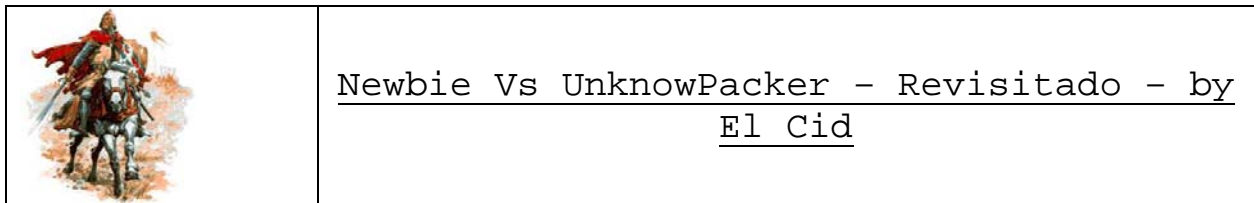
, vemos los mismos 3 nombres de APIs y también vemos la dirección de comienzo del programa, pero del módulo selfextractor del packer (el EP), no el OEP.

Names in L2Walker			
Address	Section	Type	Name
00642074		Import	KERNEL32.GetModuleHandleA
00642078		Import	KERNEL32.GetProcAddress
0064207C		Import	KERNEL32.LoadLibraryA
00649DEA		Export	<ModuleEntryPoint>

Address	Hex dump	Disassembly	Comment
00649DEA	60	PUSHAD	
00649DEB	E8 00000000	CALL L2Walker.00649DF0	
00649DF0	5D	POP EBP	
00649DF1	81ED F31D4000	SUB EBP, L2Walker.00401DF3	
00649DF7	B9 7B090000	MOV ECX, 97B	
00649DFC	8DBD 3B1E4000	LEA EDI, DWORD PTR SS:[EBP+401E3B]	
00649E02	8BF7	MOV ESI, EDI	
00649E04	AC	LODS BYTE PTR DS:[ESI]	
00649E05	61	POPAD	
00649E06	E9 C282FFFF	JMP L2Walker.006420CD	
00649E0B	0000	ADD BYTE PTR DS:[EBX], AL	

Bueno ya vemos que tanto si miramos en el header del PE, como si buscamos Names de APIs, Olly sólo encuentra 3, pero nosotros sabemos que debe haber muchas más, aunque estén en otro sitio.

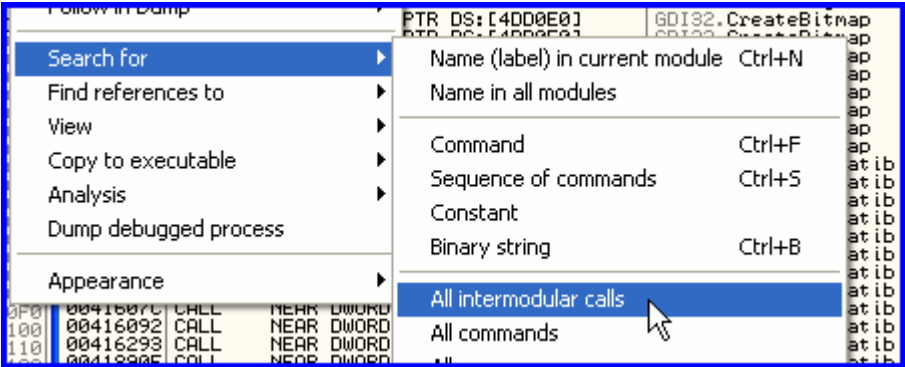
Desde el inicio del tuto, llevamos tiempo afirmando que una de las cosas que hace el packer es colocar las entradas de la IAT, que son direcciones de comienzo de APIs, en otro sitio, en secciones privadas suyas, en definitiva en otros módulos del programa



(aquí la palabra módulo debe entenderse como unidades o bloques de memoria, no en el sentido habitual de conjunto de instrucciones que se ejecutan de manera independiente para producir un efecto específico).

Por tanto vamos a buscar las llamadas existentes entre módulos a ver lo que encontramos.

Hacemos:



, y obtenemos la ventana R de Olly donde vemos al principio ésto:

Address	Disassembly	Destination
00401006	CALL NEAR DWORD PTR DS:[4DD194]	DS:[0040D194]=00B30360
00401019	CALL NEAR DWORD PTR DS:[4DD198]	DS:[0040D198]=00B30380
00401045	CALL NEAR DWORD PTR DS:[4DD19C]	DS:[0040D19C]=00B303A0
00401096	CALL NEAR DWORD PTR DS:[4DD1A0]	DS:[0040D1A0]=00B303C0
004010FC	CALL NEAR DWORD PTR DS:[4DD18C]	DS:[0040D18C]=00B30320
00401262	CALL NEAR DWORD PTR DS:[4DD550]	DS:[0040D550]=00B41020
00401B98	CALL NEAR DWORD PTR DS:[4DD550]	DS:[0040D550]=00B41020
00401E7D	CALL NEAR DWORD PTR DS:[4DD554]	DS:[0040D554]=00B41040
004024BC	CALL NEAR DWORD PTR DS:[4DD17C]	DS:[0040D17C]=00B302A0
004024C8	CALL NEAR DWORD PTR DS:[4DD180]	DS:[0040D180]=00B302C0
004024DF	CALL NEAR DWORD PTR DS:[4DD184]	DS:[0040D184]=00B302E0
004026A6	CALL NEAR DWORD PTR DS:[4DD178]	DS:[0040D178]=00B30280
00403A11	CALL NEAR DWORD PTR DS:[4DD18C]	DS:[0040D18C]=00B30320

, en la zona intermedia esto otro:

Address	Disassembly	Destination
00401006	CALL NEAR DWORD PTR DS:[4DD194]	DS:[0040D194]=00B30360
00401019	CALL NEAR DWORD PTR DS:[4DD198]	DS:[0040D198]=00B30380
00401045	CALL NEAR DWORD PTR DS:[4DD19C]	DS:[0040D19C]=00B303A0
00401096	CALL NEAR DWORD PTR DS:[4DD1A0]	DS:[0040D1A0]=00B303C0
004010FC	CALL NEAR DWORD PTR DS:[4DD18C]	DS:[0040D18C]=00B30320
00401262	CALL NEAR DWORD PTR DS:[4DD550]	DS:[0040D550]=00B41020
00401B98	CALL NEAR DWORD PTR DS:[4DD550]	DS:[0040D550]=00B41020
00401E7D	CALL NEAR DWORD PTR DS:[4DD554]	DS:[0040D554]=00B41040
004024BC	CALL NEAR DWORD PTR DS:[4DD17C]	DS:[0040D17C]=00B302A0
004024C8	CALL NEAR DWORD PTR DS:[4DD180]	DS:[0040D180]=00B302C0
004024DF	CALL NEAR DWORD PTR DS:[4DD184]	DS:[0040D184]=00B302E0
004026A6	CALL NEAR DWORD PTR DS:[4DD178]	DS:[0040D178]=00B30280
00403A11	CALL NEAR DWORD PTR DS:[4DD18C]	DS:[0040D18C]=00B30320

, y hacia el final ésto:



Newbie Vs UnknowPacker - Revisitado - by El Cid

0043E800	CALL	NEAR	DWORD	PTR	DS:[400570]	WINMM.timeGetTime
0043F57A	CALL	NEAR	DWORD	PTR	DS:[400570]	WINMM.timeGetTime
00442208	CALL	NEAR	DWORD	PTR	DS:[400570]	WINMM.timeGetTime
0044273D	CALL	NEAR	DWORD	PTR	DS:[400570]	WINMM.timeGetTime
004437F5	CALL	NEAR	DWORD	PTR	DS:[400570]	WINMM.timeGetTime
004456CF	CALL	NEAR	DWORD	PTR	DS:[400570]	WINMM.timeGetTime
0044591C	CALL	NEAR	DWORD	PTR	DS:[400570]	WINMM.timeGetTime
00447B37	CALL	NEAR	DWORD	PTR	DS:[400570]	WINMM.timeGetTime
00447F81	CALL	NEAR	DWORD	PTR	DS:[400570]	WINMM.timeGetTime
004894E0	CALL	NEAR	DWORD	PTR	DS:[400028]	COMCTL32._TrackMouseEvent
0048DF49	CALL	NEAR	DWORD	PTR	DS:[400028]	COMCTL32._TrackMouseEvent
004AF716	CALL	NEAR	DWORD	PTR	DS:[400028]	COMCTL32._TrackMouseEvent
004B02DA	CALL	NEAR	DWORD	PTR	DS:[400028]	COMCTL32._TrackMouseEvent
004165FE	CALL	NEAR	DWORD	PTR	DS:[4002F8]	MSG32.TransparentBlt
0044FF22	CALL	NEAR	DWORD	PTR	DS:[400310]	OLEAUT32.VariantChangeType
0044FFE1	CALL	NEAR	DWORD	PTR	DS:[400310]	OLEAUT32.VariantChangeType
00450205	CALL	NEAR	DWORD	PTR	DS:[400310]	OLEAUT32.VariantChangeType
00450289	CALL	NEAR	DWORD	PTR	DS:[400310]	OLEAUT32.VariantChangeType
004D40E9	CALL	NEAR	DWORD	PTR	DS:[400310]	OLEAUT32.VariantChangeType
004D444A	CALL	NEAR	DWORD	PTR	DS:[400310]	OLEAUT32.VariantChangeType
0044FF76	CALL	NEAR	DWORD	PTR	DS:[40030C]	OLEAUT32.VariantClear
004500B6	CALL	NEAR	DWORD	PTR	DS:[40030C]	OLEAUT32.VariantClear
004500D8	CALL	NEAR	DWORD	PTR	DS:[40030C]	OLEAUT32.VariantClear
00450205	CALL	NEAR	DWORD	PTR	DS:[40030C]	OLEAUT32.VariantClear

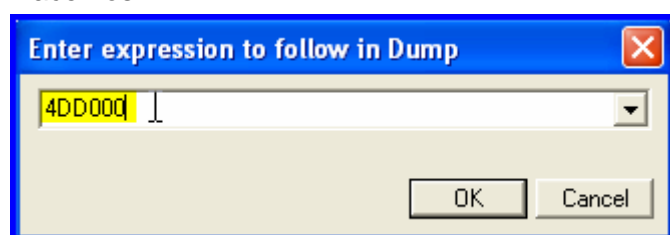
En esta última imagen se ve claramente que hay CALLs a APIs de WINMM.DLL, de COMCTL.DLL y de OLEAUT32.DLL y Olly así nos lo indica. Vemos que las posiciones implicadas son 4DDxxx, donde las xxx toman diferentes valores. En la imagen de la zona intermedia también se ve el mismo esquema de CALLs aunque Olly no nos indica nada al respecto y en la del comienzo lo mismo.

Vamos a la más baja dirección de la ventana de entre las mostradas [4DD000]:

004D1C67	CALL	NEAR	DWORD	PTR	DS:[400000]	ADVAPI32.RegCloseKey
004D1DA0	CALL	NEAR	DWORD	PTR	DS:[400000]	ADVAPI32.RegCloseKey
004D1DB6	CALL	NEAR	DWORD	PTR	DS:[400000]	ADVAPI32.RegCloseKey
004D2264	CALL	NEAR	DWORD	PTR	DS:[400000]	ADVAPI32.RegCloseKey
004D23A7	CALL	NEAR	DWORD	PTR	DS:[400000]	ADVAPI32.RegCloseKey
004D2D00	CALL	NEAR	DWORD	PTR	DS:[400000]	ADVAPI32.RegCloseKey
004D2D51	CALL	NEAR	DWORD	PTR	DS:[400000]	ADVAPI32.RegCloseKey
004D2E47	CALL	NEAR	DWORD	PTR	DS:[400000]	ADVAPI32.RegCloseKey
004D2F45	CALL	NEAR	DWORD	PTR	DS:[400000]	ADVAPI32.RegCloseKey
00418D8D	CALL	NEAR	DWORD	PTR	DS:[40009C]	GDI32.SelectObject
00418E62	CALL	NEAR	DWORD	PTR	DS:[40009C]	GDI32.SelectObject
00418FE2	CALL	NEAR	DWORD	PTR	DS:[40009C]	GDI32.SelectObject

Vamos a esa zona a mirar lo que hay:

Hacemos





Newbie Vs UnknowPacker - Revisitado - by El Cid

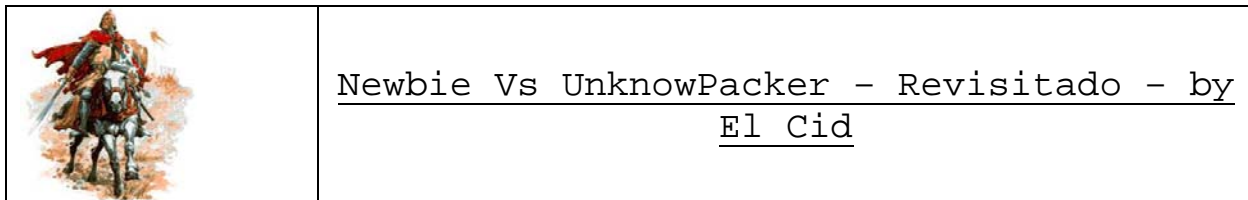
, con lo que llegamos aquí:

Address	Hex dump	ASCII
0040CFC0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040CFD0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040CFE0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040CFF0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D000	07 6C DA 77 00 00 00 00 34 48 C7 58 05 02 C4 58	...Lrw...4H&X&0-X
0040D010	D8 03 C4 58 CF 65 C3 58 F1 DF C4 58 F4 C7 C3 58	i0-X&eTX:~X&H&TX
0040D020	F8 1F C4 58 C5 2E C7 58 26 12 C5 58 F1 2F C7 58	0v-Xt.AX&+X:/AX
0040D030	9A 22 C6 58 00 00 00 00 86 77 EF 77 79 B4 EF 77	0"5X...5w'wy'w
0040D040	A0 7A EF 77 D3 8B EF 77 FB AD EF 77 F7 D9 EF 77	az'w&I'w'&'w'w
0040D050	49 D7 EF 77 37 A2 EF 77 E3 E0 EF 77 56 6A EF 77	Ii'w7&'w0&'wUj'w
0040D060	F3 AA EF 77 94 84 EF 77 F7 8E EF 77 05 EE EF 77	%'w&5'w-2'w&'w
0040D070	A3 8E EF 77 71 E6 EF 77 34 8E EF 77 2D 9C EF 77	u&'w&P'w4&'w-6'w
0040D080	A4 9D EF 77 E3 85 EF 77 7C 82 EF 77 0A 70 EF 77	R0'w0&'wI&'w.p'w
0040D090	E0 5F EF 77 79 6F EF 77 5F 6E EF 77 70 58 EF 77	0'wyo'w_n'wpI'w
0040D0A0	C6 C3 F0 77 75 AC EF 77 FA 68 EF 77 FA 8A EF 77	5t'w&'w-k'w-ll'w
0040D0B0	A1 9A EF 77 36 AD EF 77 1E EA EF 77 FF 84 EF 77	Iu'w6&'w40'w-1'w
0040D0C0	A5 61 EF 77 6A 5A EF 77 34 90 EF 77 E8 94 EF 77	Na'wjZ'w4&'wB0'w
0040D0D0	7F 90 EF 77 DB 5E EF 77 70 8A EF 77 36 8B EF 77	0&'w&'w&E'w&I'w
0040D0E0	EF 61 EF 77 29 5E EF 77 77 5D EF 77 A1 6A EF 77	'a'w)^(w&J'wiJ'w
0040D0F0	B1 A7 EF 77 C1 61 EF 77 E2 A8 EF 77 74 78 EF 77	8&'w-a'w0&'w&X'w
0040D100	EC D1 F1 77 B0 6E F0 77 0B D1 F1 77 59 6F F0 77	y&:w&n-w&B:wYo-w
0040D110	77 C0 EF 77 4C 7B EF 77 36 68 F0 77 1B 82 EF 77	w'wLc'w&k-w&e'w
0040D120	17 60 F2 77 00 00 00 00 00 00 B3 00 20 00 B3 00	+~w.....
0040D130	40 00 B3 00 60 00 B3 00 80 00 B3 00 A0 00 B3 00	0.....C...a... ..
0040D140	C0 00 B3 00 E0 00 B3 00 00 01 B3 00 20 01 B3 00	L...0...0...0... ..
0040D150	40 01 B3 00 60 01 B3 00 80 01 B3 00 A0 01 B3 00	00...0...C0...a0... ..
0040D160	C0 01 B3 00 E0 01 B3 00 00 02 B3 00 20 02 B3 00	40...00...0...0... ..
0040D170	40 02 B3 00 60 02 B3 00 80 02 B3 00 A0 02 B3 00	00...0...0...0... ..
0040D180	C0 02 B3 00 E0 02 B3 00 00 03 B3 00 20 03 B3 00	40...00...0...0... ..
0040D190	40 03 B3 00 60 03 B3 00 80 03 B3 00 A0 03 B3 00	00...0...0...0... ..
0040D1A0	C0 03 B3 00 E0 03 B3 00 00 04 B3 00 20 04 B3 00	40...0...0...0... ..
0040D1B0	40 04 B3 00 60 04 B3 00 80 04 B3 00 A0 04 B3 00	00...0...0...0... ..
0040D1C0	C0 04 B3 00 E0 04 B3 00 00 05 B3 00 20 05 B3 00	40...0...0...0... ..
0040D1D0	40 05 B3 00 60 05 B3 00 80 05 B3 00 A0 05 B3 00	00...0...0...0... ..
0040D1E0	C0 05 B3 00 E0 05 B3 00 00 06 B3 00 20 06 B3 00	40...0...0...0... ..
0040D1F0	40 06 B3 00 60 06 B3 00 80 06 B3 00 A0 06 B3 00	00...0...0...0... ..
0040D200	C0 06 B3 00 E0 06 B3 00 00 07 B3 00 20 07 B3 00	40...0...0...0... ..
0040D210	40 07 B3 00 60 07 B3 00 80 07 B3 00 A0 07 B3 00	00...0...0...0... ..
0040D220	C0 07 B3 00 E0 07 B3 00 00 08 B3 00 20 08 B3 00	L...0...0...0... ..
0040D230	40 08 B3 00 60 08 B3 00 80 08 B3 00 A0 08 B3 00	00...0...0...0... ..
0040D240	C0 08 B3 00 E0 08 B3 00 00 09 B3 00 20 09 B3 00	40...0...0...0... ..
0040D250	40 09 B3 00 60 09 B3 00 80 09 B3 00 A0 09 B3 00	00...0...0...0... ..
0040D260	C0 09 B3 00 E0 09 B3 00 00 0A B3 00 20 0A B3 00	L...0...0...0... ..
0040D270	40 0A B3 00 60 0A B3 00 80 0A B3 00 A0 0A B3 00	00...0...0...0... ..
0040D280	C0 0A B3 00 E0 0A B3 00 00 0B B3 00 20 0B B3 00	L...0...0...0... ..
0040D290	40 0B B3 00 60 0B B3 00 80 0B B3 00 A0 0B B3 00	00...0...0...0... ..
0040D2A0	C0 0B B3 00 E0 0B B3 00 00 0C B3 00 20 0C B3 00	40...0...0...0... ..
0040D2B0	40 0C B3 00 60 0C B3 00 80 0C B3 00 A0 0C B3 00	00...0...0...0... ..
0040D2C0	C0 0C B3 00 E0 0C B3 00 00 0D B3 00 20 0D B3 00	L...0...0...0... ..
0040D2D0	40 0D B3 00 60 0D B3 00 80 0D B3 00 A0 0D B3 00	00...0...0...0... ..
0040D2E0	C0 0D B3 00 E0 0D B3 00 00 0E B3 00 20 0E B3 00	L...0...0...0... ..
0040D2F0	40 0E B3 00 60 0E B3 00 80 0E B3 00 A0 0E B3 00	00...0...0...0... ..
0040D300	00 00 00 00 E5 79 0F 77 A7 4B 0F 77 20 49 0F 77	00>3v&43v
0040D310	FF 6B 0F 77 80 49 0F 77 80 48 0F 77 7E 4C 0F 77	...0y#w&K#w I#w
0040D320	6B 4D 0F 77 05 4C 0F 77 00 00 00 00 FF 31 7A 7E	k#w&I#w&H#w"L#w
0040D330	C8 71 75 7E 00 00 00 00 79 67 F4 77 8F 6D F4 77	kM#w&L#w.... 1z"
0040D340	97 6F F4 77 15 83 F4 77 00 00 00 00 00 00 B4 00	"qu"....yg&Iw&A&Iw
0040D350	20 00 B4 00 40 00 B4 00 60 00 B4 00 80 00 B4 00	u0&Iw&3&Iw.....+.

He marcado las zonas con 3 colores distintos:

- Zonas **cyan** compuestas de DWords en las que si hacemos Follow DWORD in Disassembler aparecemos en una API.
- Zonas **fucsia** en las que si hacemos lo mismo, Olly ya no nos ofrece esa posibilidad y pertenecen a secciones privadas creadas por el packer:

00720000	00103000				Map	R	L	R	L	
00830000	000BD000				Map	R	E	R	E	
00B30000	00001000				Priv	RW		RW		
00B40000	00002000				Priv	RW		RW		
00B50000	00001000				Priv	RW		RW		
00BD0000	00005000				Priv	RW		RW		
00BE0000	00001000				Priv	RW		RW		
00BF0000	00002000				Map	R		R		
00C00000	00003000				Priv	R		RW		



Si hubiéramos mirado al principio de cargar el prog no aparecerían. Aquí está la imagen recién cargado el programa y efectivamente no aparecen dichas secciones:

00200000	00000000				Map	R		R
002C0000	00041000				Map	R		R
00310000	00006000				Map	R		R
00320000	00041000				Map	R		R
00400000	00001000	L2Walker		PE header	Img	R		RWE
00401000	0021F000	L2Walker	<Sect_0>	Code	Img	RWE	Copi	RWE
00620000	00022000	L2Walker	<Sect_1>	Data,resources	Img	RW	Copi	RWE
00642000	00008000	L2Walker	<Sect_2>	SFX, imports	Img	RW	Copi	RWE
7C800000	00001000	kernel32		PE header	Img	R		RWE
7C801000	00083000	kernel32	.text	Code, imports, expo	Img	R	E	RWE
7C804000	0000F000	kernel32	.data	Data	Img	RW	Copi	RWE

- Zonas **amarillas** de separación entre las anteriores

Por encima de la zona marcada en cyan sólo hay ceros binarios y por debajo parece que sigue la misma estructura anterior.

Por tanto estamos en una zona en que *unas entradas van a APIs y otras a secciones privadas del packer, hay muchas entradas, están agrupadas por DLLs, o sea que blanco y en botella es* ¡¡ la IAT!!

La auténtica IAT, la completa, no la birria de IAT que hemos encontrado al principio que sólo tenía 3 entradas.

Bien, no es necesario hacer ningún acto de fé, sólo hay que trazar alguna de las entradas malas y ver que efectivamente nos conducen a una API, eso sí, después de bastantes vueltas y revueltas, para marearnos, de alguna trampa antitraceo (con la instrucción RDTSC) y de algunas anti BPs (detección de 0xCC) en la entrada de APIs y cosas así, de modo que quien lo haga que tenga cuidado de no caer en ellas.

Si seguimos bajando en la tabla de Olly vemos que esa misma estructura se mantiene hasta la posición 4DD61B inclusive (o 4DD61F si contamos el campo separador final, es indiferente) que es el final de la IAT:

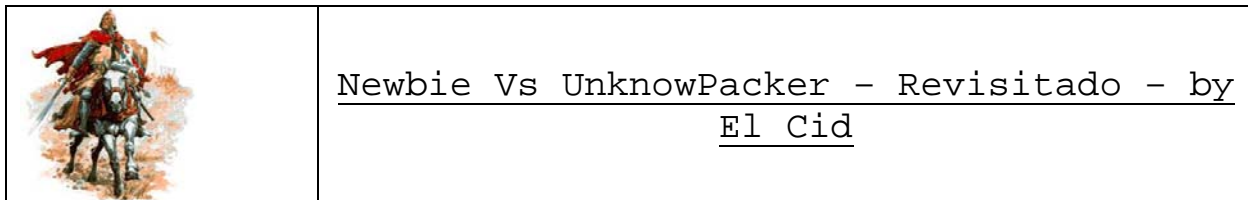
004005A0	00 01 BE 00 20 01 BE 00 40 01 BE 00 60 01 BE 00
004005B0	80 01 BE 00 A0 01 BE 00 00 00 00 00 B5 06 BB 4E
004005C0	4E 07 BB 4E B0 2C BB 4E 0A C2 BE 4E B3 71 BA 4E
004005D0	9A BC BB 4E 7D B1 BE 4E 73 2A BB 4E 7E 69 BA 4E
004005E0	09 E1 BA 4E 8F C1 BE 4E B4 88 BB 4E 1D 72 BA 4E
004005F0	5D 34 BB 4E E8 34 BB 4E 79 7A BB 4E 86 E3 C3 4E
00400600	05 04 BF 4E A5 98 BF 4E 9A 2A C4 4E A9 69 BA 4E
00400610	00 00 00 00 FD 2C 4D 77 10 64 4D 77 00 00 00 00
00400620	18 25 55 00 28 D6 4D 00 05 00 00 00 00 00 00

Después vemos otras entradas de diferente aspecto y que, en principio, consideramos fuera de la IAT.

Para posteriores referencias yo me he copiado una imagen de la IAT que no la incluyo aquí porque abulta mucho pero que la podeis ver en el [Anexo 1](#), al final del tuto.

7. Primeros pasos

Ahora que ya sabemos donde está la IAT tenemos que averiguar las zonas del programa desde donde escribe en ella, porque entonces podremos tratar de intervenir para que se escriba lo que nosotros queremos.



Entonces lo que haremos es poner MBPs en toda la IAT para que Olly se detenga cuando se vaya a escribir allí. Como nos interesa, por claridad, tratar las diversas zonas de la IAT de manera independiente, nos conviene poner un MBP en cada zona de la IAT y éso sólo lo podremos hacer con Olly v2.0 Por lo tanto y hasta nuevo aviso usaremos esta versión 2.0. El que quiera puede seguir con Olly 1.10 pero deberá adaptar las instrucciones que vayamos dando a esa versión y resulta un poco más incómodo.

Cargamos entonces el progy en Olly v2.0:

```

00649DE2  61          DB 61
00649DE3  1F          DB 1F
00649DE4  2D 6D 48 32 ASCII "~mH2i",0
00649DEA  60          PUSHAD
00649DEB  E8 00000000 CALL 00649DF0
00649DF0  5D          POP EBP
00649DF1  81ED F31D400 SUB EBP,00401DF3
00649DF7  B9 7B090000 MOV ECX,97B
00649DFC  8DBD 3B1E400 LEA EDI,[EBP+401E3B]
00649E02  8BF7        MOV ESI,EDI
00649E04  AC          LODS BYTE PTR DS:[ESI]
00649E05  61          POPAD
00649E06  E9 C282FFFF JMP 006420CD
00649E0B  00          DB 00
00649E0C  00          DB 00
00649E0D  00          DB 00

```

Aquí lo tenemos detenido en el EP del self-extractor.

Antes de nada, como este Olly no tiene plugins que lo oculten, pondremos a cero el byte de IsDebuggerPresent.

Para ello, hacemos Follow in Dump con el contenido de EBX:

```

EDX 7C91E514 ntdll.KiFastSystemCall
EBX 7FFD7000
ESP 0012F Increment
EBP 0012F
ESI 5B153 Decrement
EDI 0012B
EIP 00649 Zero
C 0 ES 0 Set to 1
P 1 CS 0
A 0 SS 0 Modify...
Z 1 DS 0
S 0 FS 0 Copy to clipboard
T 0 GS 0 Copy all registers
D 0
O 0 Last
EFL 00000 Follow in Dump
ST0 empty

```

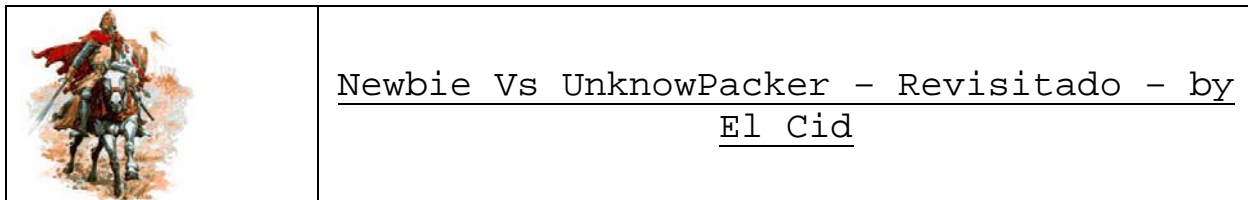
Y en la posición [EBX+2] del Dump ponemos 00:

L2Walker.<ModuleEntryPoint>	
Address	Hex dump
7FFD7000	00 00 00 FF FF FF FF
7FFD7010	00 00 02 00 00 00 00 00
7FFD7020	00 10 91 7C E0 10 91 7C
7FFD7030	00 00 00 00 00 00 00 00

Ésto lo haremos siempre que reiniciemos este Olly, lo diga el tuto o no lo diga, porque quizá se me pueda olvidar pero es necesario para no arriesgarnos a arruinar nuestro trabajo de horas.

A continuación, definimos un MBP On Write para cada zona de la IAT vista antes, es decir, en las siguientes posiciones, que vemos también en la ventana M de Olly:

4DD000-4DD004	
4DD008-4DD034	
4DD038-4DD124	
4DD128-4DD2F4	
4DD2F8-4DD300	



```

4DD304-4DD328
4DD32C-4DD334
4DD338-4DD348
4DD34C-4DD56C
4DD570-4DD574
4DD578-4DD57C
4DD580-4DD5B8
4DD5BC-4DD610
4DD614-4DD61C

```

Ahora ya podemos ver desde dónde escribe en la IAT. **Damos Run** y Olly se detiene aquí:

```

0037366D  PUSH ECX
0037366E  SHR ECX,2
00373671  REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00373673  POP ECX
00373674  AND ECX,00000003
00373677  REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00373679  POP ESI
0037367A  PUSH EBX
0037367B  PUSH 8000
00373680  PUSH 0
00373682  PUSH ESI
00373683  LEA EAX,[EBP+L2Walker.401D4F]
00373689  PUSH EAX

```

ECX=00050C00
[00C0C000]=0
[004DD000]=0

Vemos que efectivamente va a escribir en la posición 4DD000, la primera de la IAT, sin embargo también vemos que va a escribir un valor nulo, que no es lo que nosotros esperábamos, porque si miramos en la IAT que hemos salvado antes (ver [Anexo 1](#)) el valor que debe contener esta posición es 77DA6C07.

Bueno pues continuemos. **Damos Run** y Olly se detiene aquí:

```

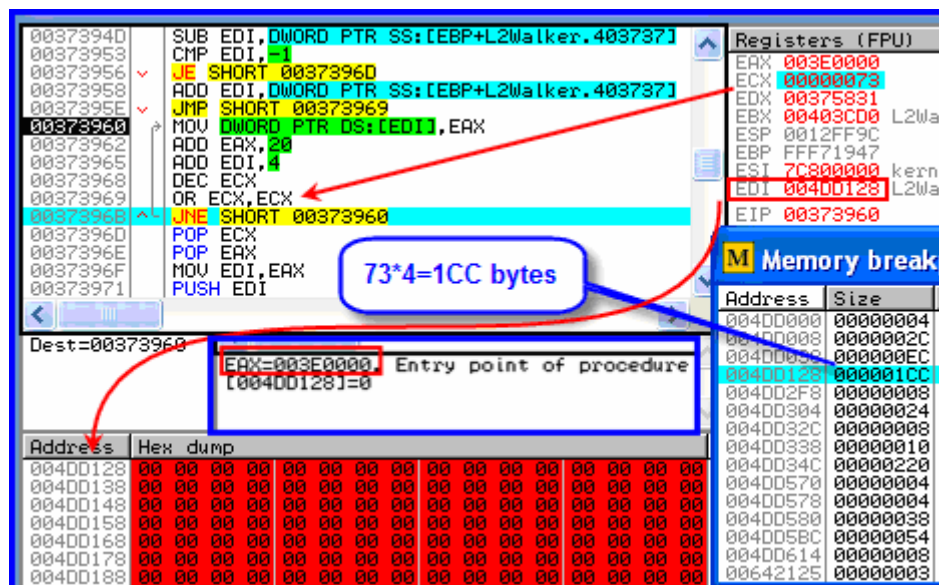
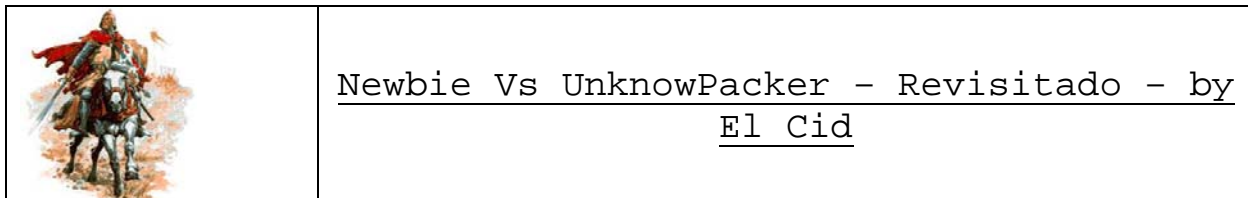
00373668  JMP 566242F8
0037366D  PUSH ECX
0037366E  SHR ECX,2
00373671  REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00373673  POP ECX
00373674  AND ECX,00000003
00373677  REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00373679  POP ESI
0037367A  PUSH EBX
0037367B  PUSH 8000
00373680  PUSH 0

```

ECX=00050BF0
[00C0C040]=0
[004DD040]=0

Es el mismo lugar del programa y de nuevo escribe ceros en la posición 4DD040 y siguientes, mediante DWords. Da la sensación que está inicializando a ceros toda la zona de la IAT.

Voy dando Run, hasta que vea que el valor que va a escribir es diferente de cero. Podría hacerse con un BMP condicional pero como no son muchas veces las que va a parar no merece la pena, así que sigo dando Run mirando de reojo el valor que se escribe. A mí me han salido 23 veces más que hay que dar Run y paramos aquí:



En esta figura, en la que he juntado y/o solapado varias ventanas de Olly, he intentado resumir la situación. Vemos que se escribe desde la posición 373960 del prog. El valor que se escribe está en EAX y vale 3E0000. Este valor ya concuerda con el de la IAT definitiva como podemos ver en la imagen del Anexo 1. Fijaos en que digo concuerda y no digo que es igual, porque como se trata de direcciones virtuales de memoria y estamos en otro Run distinto, las direcciones pueden variar y de hecho varían. Sin embargo las direcciones reales de APIs siempre serán las mismas. Ésto no nos debe preocupar.

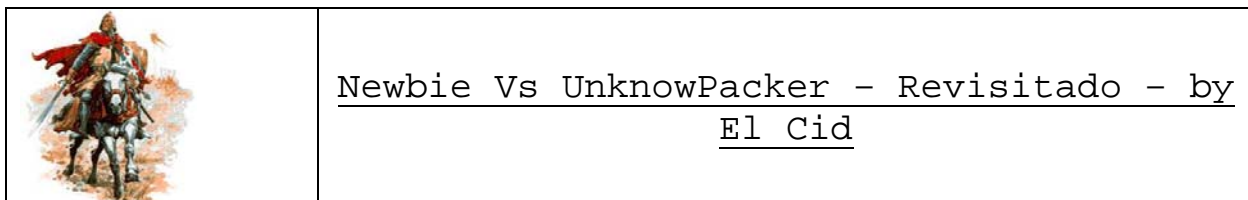
Por si alguien no se ha dado cuenta todavía, ésta es claramente una zona MALA, es decir de entradas redireccionadas de la IAT, ya que el valor que se escribe (3E0000) no corresponde a dirección real de API alguna.

Siguiendo con la figura, vemos que se hace un bucle de escritura que se repite -- según el valor de ECX -- 73 hex veces, o sea que se escriben otras tantas dobles palabras, que son exactamente:

$$73 \text{ hex} * 4 = 1CC \text{ hex bytes,}$$

, el mismo valor que vemos en la ventana M de Olly para esa zona, o sea que va a escribir esta zona de la IAT completa.

Bien, pues dejemos que la escriba sin interrumpirle, es decir, desactivemos el MBP de esta zona de la IAT:



Newbie Vs UnknowPacker - Revisitado - by El Cid

M Memory breakpoints					
Address	Size	Module	Type	Status	Comment
0040D000	00000004	L2Walker	W	Active	
0040D008	0000002C	L2Walker	W	Active	
0040D038	000000FC	L2Walker	W	Active	
0040D128	000001CC	L2Walker	W	Disabled	
0040D2F8	00000008	L2Walker	W	Active	
0040D304	00000024	L2Walker	W	Active	
0040D32C	00000008	L2Walker	W	Active	
0040D338	00000010	L2Walker	W	Active	
0040D34C	00000020	L2Walker	W	Active	
0040D570	00000004	L2Walker	W	Active	
0040D578	00000004	L2Walker	W	Active	
0040D580	00000038	L2Walker	W	Active	
0040D58C	00000054	L2Walker	W	Active	
0040D614	00000008	L2Walker	W	Active	
00642125	00000003	L2Walker	E	Disabled	

Address	Hex dump	A
0040D128	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
0040D138	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
0040D148	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
0040D158	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
0040D168	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
0040D178	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
0040D188	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
0040D198	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
0040D1A8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
0040D1B8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.

, **demos Run** y veremos cómo se rellena la zona anterior con los valores, todos ellos, malos (vista parcial de la zona):

Address	Hex dump
0040D128	00 00 3E 00 20 00 3E 00 40 00 3E 00 60 00 3E 00
0040D138	00 00 3E 00 A0 00 3E 00 C0 00 3E 00 E0 00 3E 00
0040D148	00 01 3E 00 20 01 3E 00 40 01 3E 00 60 01 3E 00
0040D158	00 01 3E 00 A0 01 3E 00 C0 01 3E 00 E0 01 3E 00
0040D168	00 02 3E 00 20 02 3E 00 40 02 3E 00 60 02 3E 00
0040D178	00 02 3E 00 A0 02 3E 00 C0 02 3E 00 E0 02 3E 00
0040D188	00 03 3E 00 20 03 3E 00 40 03 3E 00 60 03 3E 00
0040D198	00 03 3E 00 A0 03 3E 00 C0 03 3E 00 E0 03 3E 00
0040D1A8	00 04 3E 00 20 04 3E 00 40 04 3E 00 60 04 3E 00
0040D1B8	00 04 3E 00 A0 04 3E 00 C0 04 3E 00 E0 04 3E 00
0040D1C8	00 05 3E 00 20 05 3E 00 40 05 3E 00 60 05 3E 00
0040D1D8	00 05 3E 00 A0 05 3E 00 C0 05 3E 00 E0 05 3E 00
0040D1E8	00 06 3E 00 20 06 3E 00 40 06 3E 00 60 06 3E 00
0040D1F8	00 06 3E 00 A0 06 3E 00 C0 06 3E 00 E0 06 3E 00

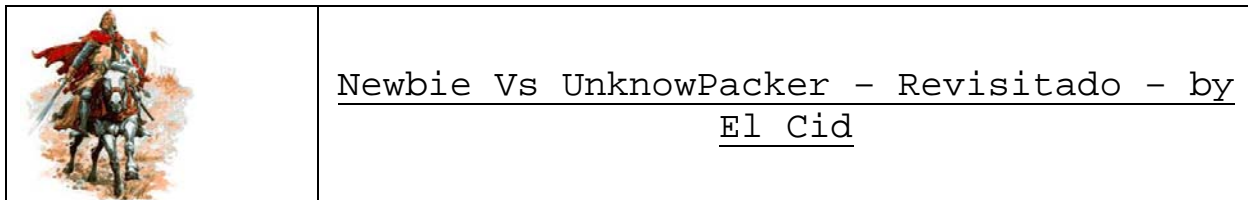
Además el proggy se detiene aquí:

00373904	MOV DWORD PTR DS:[EDI],EAX	
00373906	POP EDX	
00373907	MOVZX EAX, BYTE PTR DS:[EDX-1]	
00373908	ADD EDX,EAX	
00373909	INC EDX	
0037390E	ADD EDI,4	
00373911	POP ECX	
00373912	LOOP SHORT 00373897	
00373914	JMP 00373D83	
00373919	MOV ECX,DWORD PTR DS:[EDX]	
0037391B	AND ECX,7FFFFFFF	
00373921	PUSH ECX	
00373922	PUSH EDX	
00373923	SHL ECX,5	
00373926	PUSH 4	

EAX=77DA6C07 (ADVAPI32.RegCloseKey)		
[0040D000]=0		

Address	Hex dump	ASCII
0040D000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Registers (FPU)	
EAX	77DA6C07
ECX	7C929280
EDX	7C98E178
EBX	003D04AC
ESP	0012FF9C
EBP	FFF71947
ESI	77DA0000
EDI	0040D000
EIP	00373904
C	1 ES 0023 3
P	0 CS 001B 3
A	1 SS 0023 3
Z	0 DS 0023 3
S	1 FS 003B 3
T	0 GS 0000 0
O	0 LastErr 0
O	0
EFL	00010293
ST0	empty -1.0



Vemos que se va a escribir en la posición 4DD000 (origen de la IAT). El valor que se escribe es el de EAX, pero ahora es una dirección real de una API que corresponde a la API RegCloseKey, como fielmente nos indica Olly.

También es un bucle, como vemos con la instrucción LOOP un poco más abajo. ¿Cuántas veces se efectúa el bucle? Lleguemos hasta ese punto con F7:

```

00373904 MOV DWORD PTR DS:[EDI],EAX
00373906 POP EDX
00373907 MOVZX EAX,BYTE PTR DS:[EDX-1]
00373908 ADD EDX,EAX
0037390D INC EDX
0037390E ADD EDI,4
00373911 POP ECX
00373912 LOOP SHORT 00373897
00373914 JMP 00373D83
00373919 MOV ECX,DWORD PTR DS:[EDX]
0037391B AND ECX,7FFFFFFF
00373921 PUSH ECX
00373922 PUSH EDX
00373923 SHL ECX,5
00373926 PUSH 4

```

Registers (FPU)

EAX	0000000B
ECX	00000001
EDX	00375F77 AS
EBX	003D04AC
ESP	0012FFA4
EBP	FFF71947
ESI	77DA0000 AD
EDI	004D0004 L2
EIP	00373912

Jump is not taken
Dest=00373897
ECX=1

Address	Hex dump	ASCII
0040D000	07 6C DA 77 00 00 00 00 00 00 00 00 00 00 00 00
0040D010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Aquí vemos que ECX=1, pero sabemos que la instrucción LOOP va a restar 1 a ECX antes de ejecutarse, o sea :

$ECX \leftarrow ECX - 1$, es decir, va a resultar $ECX = 0$, y ya no ejecuta más el LOOP y continua. En definitiva se ha hecho 1 vez y ... ¡Qué “casualidad”!, vemos que esa zona de la IAT tiene también una única entrada ya que a continuación de la posición viene el campo separador de ceros binarios. La zona es BUENA porque la dirección de API que contiene apunta realmente a una API.

Como la zona se ha completado, **damos Run** y se detiene aquí:

```

00373948 PUSH EAX
0037394C PUSH ECX
0037394D SUB EDI,DWORD PTR SS:[EBP+L2Walker.403737]
00373953 CMP EDI,-1
00373956 JE SHORT 0037396D
00373958 ADD EDI,DWORD PTR SS:[EBP+L2Walker.403737]
0037395E JMP SHORT 00373969
00373960 MOV DWORD PTR DS:[EDI],EAX
00373962 ADD EAX,20
00373965 ADD EDI,4
00373968 DEC ECX
00373969 OR ECX,ECX
0037396B JNE SHORT 00373960
0037396D POP ECX
0037396E POP EAX

```

Registers (FPU)

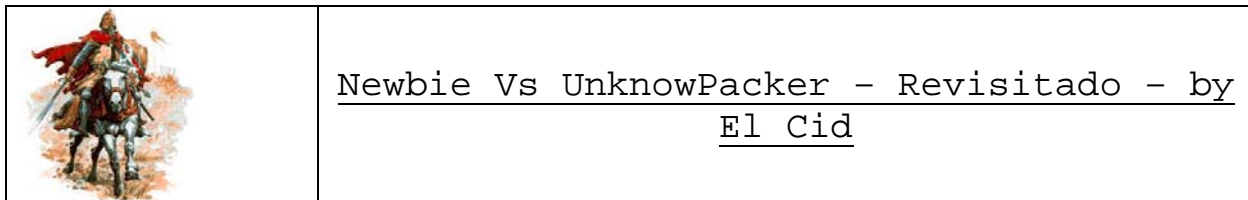
EAX	003F0000
ECX	00000000
EDX	00375F87
EBX	003D04AC
ESP	0012FF9C
EBP	FFF71947
ESI	7E390000 US
EDI	004D034C L2
EIP	00373960

EAX=003F0000, Entry point of procedure
[0040D034C]=0

Address	Hex dump	ASCII
0040D034C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D035C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D036C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D037C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Vemos los mismos elementos de antes: se escribe en la posición 4DD34C que es el inicio de una zona de la IAT, el valor que se escribe está en EAX y vale 3F0000, que es un valor MALO o sea redireccionado y se escriben 88 hex DWords que equivalen a:

88 hex *4= 220h bytes



Newbie Vs UnknowPacker - Revisitado - by El Cid

, como vemos en la ventana M de Olly:

M Memory breakpoints					
Address	Size	Module	Type	Status	Comment
004DD000	00000004	L2Walker	W	Active	
004DD008	0000002C	L2Walker	W	Active	
004DD038	000000EC	L2Walker	W	Active	
004DD128	000001CC	L2Walker	W	Disabled	
004DD2F8	00000008	L2Walker	W	Active	
004DD304	00000024	L2Walker	W	Active	
004DD32C	00000008	L2Walker	W	Active	
004DD338	00000010	L2Walker	W	Active	
004DD347	00000220	L2Walker	W	Active	
004DD570	00000004	L2Walker	W	Active	
004DD578	00000004	L2Walker	W	Active	
004DD580	00000038	L2Walker	W	Active	
004DD5BC	00000054	L2Walker	W	Active	

Desactivamos el MBP de esta zona y damos Run, con lo que la zona se rellena completamente como vemos en la fig siguiente (parcial):

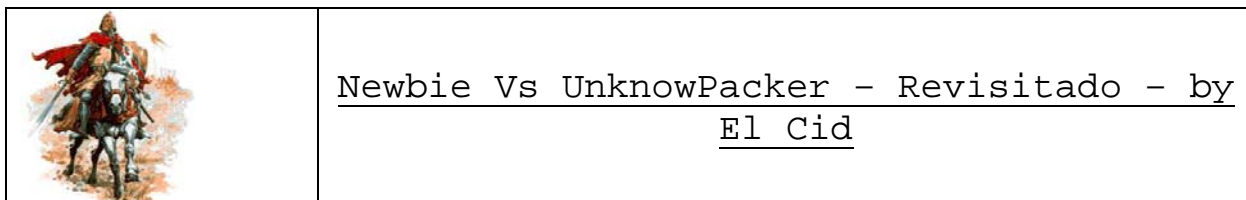
Address	Hex dump	ASCII
004DD34C	00 00 3F 00 20 00 3F 00 40 00 3F 00 60 00 3F 00	..?..?..?..?
004DD35C	80 00 3F 00 A0 00 3F 00 C0 00 3F 00 E0 00 3F 00	0..?..?..?..?
004DD36C	00 01 3F 00 20 01 3F 00 40 01 3F 00 60 01 3F 00	0..?..?..?..?
004DD37C	80 01 3F 00 A0 01 3F 00 C0 01 3F 00 E0 01 3F 00	0..?..?..?..?
004DD38C	00 02 3F 00 20 02 3F 00 40 02 3F 00 60 02 3F 00	0..?..?..?..?
004DD39C	80 02 3F 00 A0 02 3F 00 C0 02 3F 00 E0 02 3F 00	0..?..?..?..?
004DD3AC	00 03 3F 00 20 03 3F 00 40 03 3F 00 60 03 3F 00	0..?..?..?..?
004DD3BC	80 03 3F 00 A0 03 3F 00 C0 03 3F 00 E0 03 3F 00	0..?..?..?..?
004DD3CC	00 04 3F 00 20 04 3F 00 40 04 3F 00 60 04 3F 00	0..?..?..?..?
004DD3DC	80 04 3F 00 A0 04 3F 00 C0 04 3F 00 E0 04 3F 00	0..?..?..?..?
004DD3EC	00 05 3F 00 20 05 3F 00 40 05 3F 00 60 05 3F 00	0..?..?..?..?
004DD3FC	80 05 3F 00 A0 05 3F 00 C0 05 3F 00 E0 05 3F 00	0..?..?..?..?

Además paramos aquí:

00373904	8907	MOV DWORD PTR DS:[EDI],EAX	Registers (FPU)	
00373906	5A	POP EDX	EAX 77EF7786 GDI32.CreateRectRgn	
00373907	0FB642 FF	MOVZX EAX, BYTE PTR DS:[EDX+1]	ECX 7C929280 ntdll.7C929280	
00373908	03D0	ADD EDX,EAX	EDX 7C98E178 ntdll.7C98E178	
0037390D	42	INC EDX	EBX 003D0930 ASCII "hx9"	
0037390E	83C7 04	ADD EDI,4	ESP 0012FF9C	
00373911	59	POP ECX	EBP FFF71947	
00373912	E2 83	LOOP SHORT 00373897	ESI 77EF0000 GDI32.77EF0000	
00373914	✓ E9 6A040000	JMP 00373D83	EDI 004DD038 L2Walker.004DD038	
00373919	8B0A	MOV ECX,DWORD PTR DS:[EDX]	EIP 00373904	
0037391B	81E1 FFFFFFFF	AND ECX,FFFFFFFF	C 1 ES 0023 32bit 0(FFFFFFFF)	
00373921	51	PUSH ECX	P 0 CS 001B 32bit 0(FFFFFFFF)	
00373922	52	PUSH EDX	A 1 SS 0023 32bit 0(FFFFFFFF)	
00373923	C1E1 05	SHL ECX,5	Z 0 DS 0023 32bit 0(FFFFFFFF)	
00373926	6A 04	PUSH 4	S 1 FS 003B 32bit 7FFDF000(FFF)	
			T 0 GS 0000 NULL	
			D 0	
			O 0 LastErr 00000002 ERROR_FILE	
			EFL 00010293 (NO,B,NE,BE,S,P,O,L)	
			ST0 empty +UNORM 0002 0000002C 00	
EAX=77EF7786 (GDI32.CreateRectRgn)				
[004DD038]=0				
Address	Hex dump	Address	Value	Comments
004DD038	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0012FF9C	0037677F	
004DD048	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0012FFA0	0000003B	
004DD058	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0012FFA4	7C929280	RETURN from ntdll.7
004DD068	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0012FFA8	7FFDF000	
004DD078	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0012FFAC	0012FFF0	

Ahí va a escribir la primera entrada de esta zona de la IAT (4DD038). Como antes, como siempre podríamos ya decir, vemos que se escribe el valor contenido en EAX, que Olly nos dice fielmente que es la dirección de la API CreateRectRgn de GDI32.DLL.

¿Cuántas entradas de la IAT se van a rellenar? Vemos que el bucle lo controla la instrucción LOOP de unas líneas más abajo y sabemos que esta instrucción se controla con el registro ECX que se va a establecer justo antes del LOOP mediante el POP ECX que vemos. Como vemos que más arriba hay otro POP EDX, el valor de ECX será el segundo valor que haya en la pila en este momento, que ahí en la figura



anterior lo tenemos marcado de azul: 3B hex. Así que se rellenarán 3B DWords lo que equivale a :

$$3B \times 4 = 0EC \text{ h bytes}$$

, que es el valor que vemos en la ventana de MBPs de Olly.

00400000	00000004	L2Walker	W	Disabled	
00400008	0000002C	L2Walker	W	Active	
00400034	0000005C	L2Walker	W	Active	
00400128	000001CC	L2Walker	W	Disabled	
004002F8	00000008	L2Walker	W	Active	

Así que todo concuerda por ahora.

Ahora nos preguntamos ¿Qué sabemos hasta ahora respecto de las APIs de la IAT? Pues sabemos los sitios del programa desde los que se escriben las entradas buenas y malas, pero no sabemos, cómo se obtienen esas direcciones ni donde lo hace el programa, ni mucho menos en base a que se decide si una zona de la IAT va a tener entradas buenas o malas. Así que hemos averiguado bastantes cosas pero aún nos falta por saber otras cuantas, para poder reparar la IAT y desempacar el programa.

Si consideramos que ahora mismo estamos parados al comienzo del relleno de una zona BUENA de la IAT, que además tiene muchas entradas, no cabe duda de que el programa tendrá que repetir muchas veces el ciclo completo de obtención de la dirección de cada API y su escritura en la IAT, por lo que es buen momento para observar con detenimiento lo que hace, lo que quiere decir trazar con F7 ó F8 según el caso.

8. Localización y formación de nombres y direcciones de las APIs en entradas buenas de la IAT

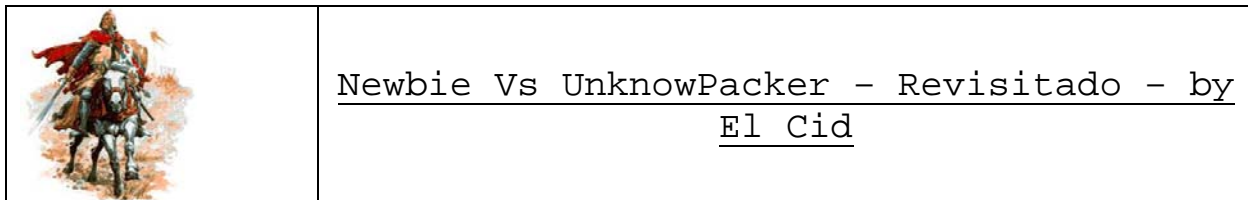
8.1. Datos de las DLLs

Como decimos, estamos parados al inicio del relleno de la zona BUENA de la IAT que comienza en 4DD038, pero al final del proceso de la primera entrada que debe ser, obviamente, escribir la dirección en la Tabla. Por tanto empezaremos a trazar con esta idea en la mente, de que el programa lo que hará será repetir un cierto ciclo para la obtención de la segunda entrada de esta zona de la IAT, hasta que llegue a este mismo sitio para escribirla.

En este momento y sólo como recordatorio, sabemos que las únicas 3 APIs que nos podemos encontrar son éstas que vimos al principio:

Address	Section	Type	Name
00642074		Import	KERNEL32.GetModuleHandleA
00642070		Import	KERNEL32.GetProcAddress
00642078		Import	KERNEL32.LoadLibraryA
006490E4		Export	<ModuleEntryPoint>

, ya que todas las demás aún no existen. (Estrictamente hablando se podría llamar a las APIs que ya están generadas en la tabla, pero no es el caso y más vale no dar ideas).



Antes de empezar a trazar y para prevenirnos de posibles efectos indeseados pongamos un BP en el punto donde estamos, de escritura en la IAT:

```

00373904 8907 MOV DWORD PTR DS:[EDI],EAX
00373906 5A POP EDX
00373907 0FB642 FF MOVZX EAX,BYTE PTR DS:[EDX-1]
00373908 03D0 ADD EDX,EAX
0037390D 42 INC EDX
0037390E 83C7 04 ADD EDI,4
00373911 59 POP ECX
00373912 E2 83 LOOP SHORT 00373897
00373914 E9 6A040000 JMP 00373D83
00373919 8B0A MOV ECX,DWORD PTR DS:[EDX]

```

, así si nos equivocamos y damos por ejemplo F8 en un CALL en lugar de F7, en el peor de los casos siempre parará al llegar a escribir y tendremos otra oportunidad en el siguiente ciclo del bucle.

Debo advertir, que en los tutos I, II, y III de NCR se avisa de que no debemos mantener los BPs mucho tiempo porque parece que el packer los detecta. A mí en esta zona del programa no me ha pasado, por lo que lo pondremos y si alguien observa algún problema pues que los quite de vez en cuando.

Comenzamos. **Damos F7** 2 veces, hasta llegar aquí que por ahora no es correr mucho, je, je, ya lo sé, pero bueno luego correremos:

```

00373904 MOV DWORD PTR DS:[EDI],EAX
00373906 POP EDX
00373907 MOVZX EAX,BYTE PTR DS:[EDX-1]
00373908 ADD EDX,EAX
0037390D INC EDX
0037390E ADD EDI,4
00373911 POP ECX
00373912 LOOP SHORT 00373897
00373914 JMP 00373D83
00373919 MOV ECX,DWORD PTR DS:[EDX]
0037391B AND ECX,7FFFFFFF
00373921 PUSH ECX
00373922 PUSH EDX
00373923 SHL ECX,5

```

Registers (FPU)

EAX 77EF7786 GDI32.CreateRectRgn

ECX 7C929280 ntdll.7C929280

EDX **0037677F**

EBX 003D093D ASCII "hx9""

ESP 0012FFA0

EBP FFF71947

ESI 77EF0000 GDI32.77EF0000

EDI 004DD038 L2Walker.004DD038

EIP 00373907

C 1 ES 0023 32bit 0(FFFFFFFF)

P 0 CS 001B 32bit 0(FFFFFFFF)

D 1 SS 0023 32bit 0(FFFFFFFF)

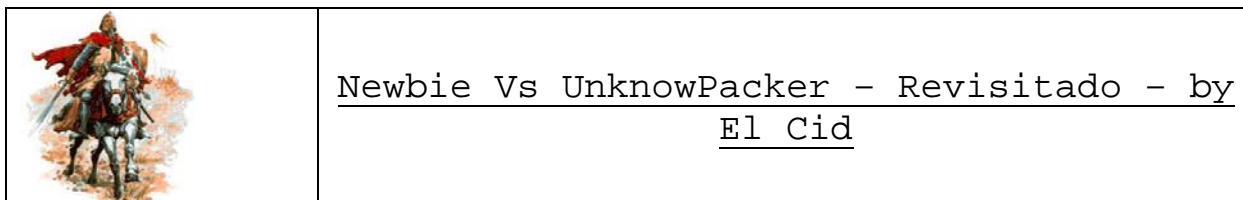
Vemos que acaba de recuperar EDX (con POP EDX), como un poco después recuperará ECX. ECX sabemos que es un valor importante: el número de veces que se repite el ciclo, o sea el nº de entradas en esta zona de la IAT, es decir el número de APIs que contiene. Entonces no es descabellado pensar que si recupera EDX será porque también es un valor importante que necesitará y usará para algo.

Pues vayamos a esa zona de la memoria a ver lo que hay. En EDX hacemos Follow in Dump:

Es esta zona:

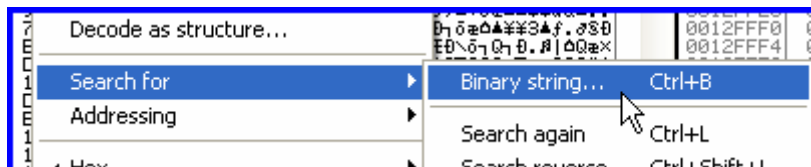
Address	Hex dump	ASCII
0037673F	FE 9F BE 51 BF D1 D2 B5 52 93 5C 9F 55 91 91 D1	■f#01 0EÁRô\ fUææ0
0037674F	BE 00 0B B5 7F 51 1F D2 D1 FE DF 75 51 7F 00 0B	%.ðÀ0QÆ0■u00.ð
0037675F	15 D1 BF 93 5C 9F 95 1E BE 1E 7F 00 38 00 00 00	3B,6\fo▲æΔ.8s..
0037676F	09 47 44 49 33 32 2E 44 4C 4C 00 38 00 00 00 00	.GDI32.DLL;....
0037677F	97 B1 53 03 71 53 B5 53 93 71 B5 13 32 00 08 17	USeqSASoqA!!2.0+
0037678F	53 71 F5 D2 F0 53 72 00 00 95 53 72 53 93 71 97	Sq3E-Sr..òSrSòqu
0037679F	72 D2 F1 B5 13 32 00 0C 95 53 71 75 53 F0 71 D7	xÉ+À!!2..òSquS-qî
003767AF	72 D2 13 32 00 08 56 12 31 53 75 12 57 F0 00 06	xÉ!!2.0U+1Su+lw.-+
003767BF	76 D2 32 53 75 12 00 09 17 53 71 B5 13 32 B7 12	vE2Su+...#SqA!!2A+

Si miramos un poco por encima del byte que señala EDX (byte azul), vemos claramente el nombre de la DLL en la que estamos que es GDI32.DLL, ya lo hemos visto antes al escribir la primera entrada de esta zona. También vemos más cosas que no sabemos identificar por ahora.

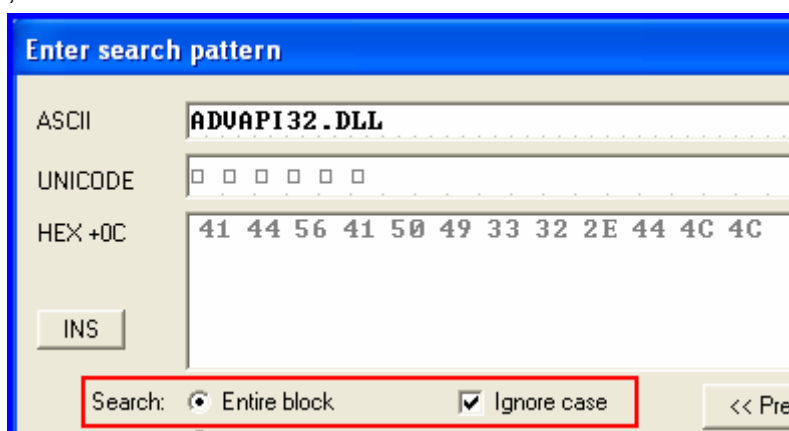


Pero si está el nombre la DLL que estamos tratando, nos preguntamos si estaría también, por ahí cerca, el de la anterior DLL que vimos, que era ADVAPI32 (si no lo recordais, miradlo [aquí](#)) cuya API se escribía en la pos 4DD000 y solo tenía una entrada en la IAT.

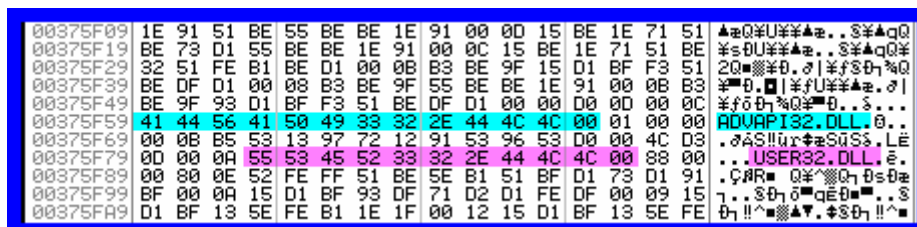
Pues vamos a verlo. En el hex Dump donde estamos hacemos:



, con:



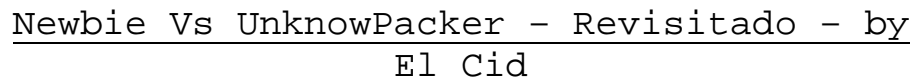
, y Olly nos muestra ésto:



O sea que sí, también estaba el nombre de esta DLL.

Esta figura es muy interesante e importante y es la clave principal de la resolución del Packer, por lo que la he enmarcado con grosor doble. La analizaremos con detalle y con calma para que no se pierda nadie.

- Lo primero que vemos es que un poco más delante de nuestro string de la DLL, se ve otro, en concreto el USER32.DLL. ¡Anda que suerte! Se podría decir “2 por el precio de 1”.
- Ahora pensamos, “claro como esa DLL (ADVAPI32) solo tiene una entrada en la IAT pues en seguida tiene que venir la siguiente, que parece que es USER32”. Y así es. Además, como somos curiosos y observadores, vemos que a continuación de la cadena ADVAPI32.DLL, terminada en cero binario, aparece un humilde “01” que nos llama mucho la atención porque, repito, sabemos que esta DLL solo tiene una entrada en la API. ¿Será este valor el que indique el número de entradas de cada DLL en la IAT? Pues miremos. El 01 visto, sabemos que realmente en formato de DWORD debe ser “00 00 00 01” que en memoria aparecerá como “01 00 00 00” como vemos en esta fig:



Ahora, sin embargo, estamos parados al comienzo de la escritura de una zona de la IAT que corresponde a GDI32. Esta zona es buena como la de ADVAPI32, y hemos visto que tiene 3B entradas. Pues vamos a ver si los datos concuerdan. Retrocedamos hasta donde estábamos en el hex Dump a ver:

29



Newbie Vs UnknowPacker - Revisitado - by El Cid

Enter search pattern

ASCII **.DLL**

UNICODE ☐ ☐

HEX +04 2E 44 4C 4C

INS

Search: ☒ Entire block ☒ Ignore case

Aparecen 2 ó 3 pero no tienen lo que buscamos, así que seguimos buscando “siguiente” con CTRL+L

, y a la tercera o la cuarta vez que damos Ctrl + L vemos esto:

003757EC	CD 19 61 CF 8B 85 53 37 40 00 85 C0 74 07 61 B8	=+aDtiS7@.ã!t.a@
003757FC	00 00 00 00 C3 6A 00 6A 00 FF B5 53 38 40 00 80	...tj.j. AS8@.i
0037580C	80 02 3E 40 00 8D BD 2E 16 40 00 2B CF 33 C0 F3	!E>@.i!c...@.+@3!%
0037581C	AA AB C3 28 D1 0D 00 0C 4B 45 52 4E 45 4C 33 32	-%t{0...KERNEL32
0037582C	2E 44 4C 4C 00 73 00 00 80 0C 93 D1 BF B2 51 9F	.DLL...C.ôhQf
0037583C	BF 05 7F 7F 1E 7F 00 0C 15 D1 BF B2 51 9F BF D5	7'ôôôô...SôhQf
0037584C	7F 7F 1E 7F 00 0C B2 1E 51 B1 B2 5E 71 7F 51 7F	ôôôô...SôhQf
0037585C	5C 55 00 13 15 D1 BF 93 5C 9F BF D1 DE B5 5E 7F	\U.!!SôhQfôhQf
0037586C	D1 91 BF 1E 7F 5C 55 00 14 15 D1 BF 13 5E FE B1	0a7!ôô\U.!!SôhQf
0037587C	1E 1F 9F B5 5E 7F D1 91 BF 1E 7F 5C 55 00 12 15	AVf!ôôôhQf!ôô\U.!!S

Aquí ya hay algo que tiene que ver con 73h. Vamos, de hecho aparece el 73h claramente pero en el byte alto de la Dword hay un 80h que no sabemos lo que significa, por ahora. Además vemos que se trata del KERNEL32.

- Vamos a hacer lo mismo con la otra zona MALA que ya hemos pasado, que os recuerdo estaba en 4DD34C y tenía 88h entradas. El que no lo recuerde puede verlo [aquí](#). Desde donde estamos damos 2 veces CTRL+L y:

Address	Hex dump	ASCII
00375F42	BE BE 1E 91 00 0B B3 BE 9F 93 D1 BF F3 51 BE DF	%#A%.ô!%fôhQf
00375F52	D1 00 00 00 00 00 0C 41 44 56 41 50 49 33 32 2E	0...S...ADVAPI32.
00375F62	44 4C 4C 00 01 00 00 0B B5 53 13 97 72 12 91	DLL.0...@AS!ur#
00375F72	53 96 53 00 00 4C D3 00 00 0A 55 53 45 52 33 32	SqS\$.Lê...USER32
00375F82	2E 44 4C 4C 00 88 00 00 80 0E 52 FE FF 51 BE 5E	.DLL.ê...C@R# Q#^
00375F92	B1 51 BF D1 73 D1 91 BF 00 0A 15 D1 BF 93 DF 71	Qh0sôh...SôhQf
00375FA2	D2 01 FE DF 00 09 15 D1 BF 13 5E FE B1 1E 1F 00	ê0#...Sôh!^!%AV.
00375FB2	12 15 D1 BF 13 5E FE B1 1E 1F 33 BE 51 91 D1 DE	#Sôh!^!%AV3#0a0i
00375FC2	D1 55 BF 00 00 52 9F 52 91 1F 5F 5F 91 00 1F 92	0a...@ôôôôôôôôôô

¡Aquí está! Y con el mismo 80h en el “msb” (most significant byte)

Entonces, empezamos a vislumbrar una regla:

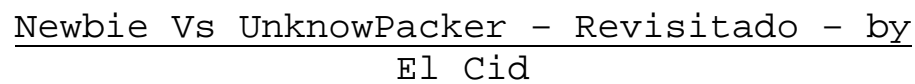
- Si la zona de la DLL en la IAT es BUENA la Dword a continuación del nombre de la DLL, lleva simplemente su longitud en DWords.
- Si la zona de la DLL en la IAT es MALA la Dword a continuación del nombre de la DLL, lleva su longitud en DWords con un “80”h añadido en su msb.

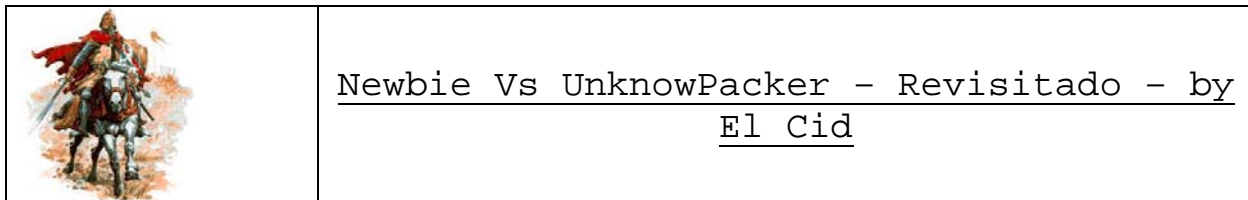
Por supuesto estaremos alerta de que esta hipótesis pueda no cumplirse.

Aquí están el resto de las DLLs del programa encontradas haciendo CTRL+L:

Resto de DLLs que usa el programa y número de entradas en la IAT de cada una

Address	Hex dump	ASCII
00376A86	13 57 F0 00 10 95 53 71 35 D2 53 11 F1 12 B1 71	!!W-.!ôSq5ES!+!#q
00376A96	16 B1 13 57 F0 00 06 57 91 93 D3 F1 53 00 0B B5	...!!W-.!ôôôô!S.ôA
00376AA6	53 93 71 35 D2 91 D2 B3 72 53 00 09 F5 71 35 D2	Sôq5Eê!rS...Sq5E
00376AB6	91 D2 B3 72 53 00 70 05 00 00 09 57 49 4E 4D 4D	êê!rS.p'...WINMM
00376AC6	2E 44 4C 4C 00 01 00 00 00 0B 71 D2 52 53 17 53	.DLL.0...ôqERS\$S
00376AD6	71 75 D2 52 53 00 F8 D2 00 00 0B 4D 53 49 4D 47	quERS.°ê...@MSIMG
00376AE6	33 32 2E 44 4C 4C 00 02 00 00 00 0E 75 B1 D3 32	32.DLL.0...!u#ê2

31



Vemos que todas son buenas menos una mala (la marcada en rosa que lleva 80h en el msb).

Creo que hemos sacado bastante información de la figura anterior que os he indicado que era importante. Pero aún sacaremos más.

Volvemos a dicha figura, que repito aquí, por comodidad y recordamos que corresponde, al momento actual donde estamos detenidos (en 373807), pero a la zona de memoria que usó el programa cuando escribió las entradas de la IAT correspondientes a la ADVAPI32.DLL y que la estamos usando porque como sólo tiene una entrada de una API se aprecian las cosas mucho mejor que en otras con muchas entradas que queda todo más enmascarado.

00375F09	1E 91 51 BE 55 BE BE 1E 91 00 00 15 BE 1E 71 51	ΔQ%U%Δ%..S%ΔQ
00375F19	BE 73 D1 55 BE BE 1E 91 00 0C 15 BE 1E 71 51 BE	%s0U%Δ%..S%ΔQ%
00375F29	32 51 FE B1 BE D1 00 0B B3 BE 9F 15 D1 BF F3 51	2Q%Δ%Δ.%f%Δ%Q
00375F39	BE DF D1 00 0B B3 BE 9F 55 BE BE 1E 91 00 0B B3	%Δ%Δ.%f%U%Δ%Δ.%f
00375F49	BE 9F 93 D1 BF F3 51 BE DF D1 00 00 0D 0D 0C	%fδb%Q%Δ%Δ%..S%..
00375F59	41 44 56 41 50 49 33 32 2E 44 4C 4C 00 01 00 00	ADVAPI32.DLL.0..
00375F69	00 0B B5 53 13 97 72 12 91 53 96 53 D0 00 4C D3	.ΔAS!UrΔSΔS.LÉ
00375F79	00 00 0A 55 53 45 52 33 32 2E 44 4C 4C 00 88 00	..USER32.DLL.é.
00375F89	00 80 0E 52 FE FF 51 BE 5E B1 51 BF D1 73 D1 91	.ÇR= Q%Δ%QΔ%Δ%Δ%
00375F99	BF 00 0A 15 D1 BF 93 DF 71 D2 D1 FE DF 00 09 15	γ..SΔΔ%Δ%Δ%Δ%Δ%
00375FA9	D1 BF 13 5E FE B1 1E 1F 00 12 15 D1 BF 13 5E FE	bγ!!^Δ%Δ%Δ%Δ%Δ%

Si miramos un poco delante del nombre de la DLL, vemos unos bytes en naranja, que incluyen los “0DD000” que nos resultan familiares.

¿No recordais la posición 4DD000 que hemos visto [aquí](#)? Era precisamente el origen de la IAT y el valor que vemos ahora, 0DD000, no parece sino el Offset del mismo.

Luego, acabamos de descubrir que 1 byte por delante del nombre de la DLL hay una DWORD con el Offset de la zona de la IAT donde se escribirán las entradas de esa DLL.

Nos gusta ser prudentes e inmediatamente vamos a comprobar ésto. Sólo tenemos que mirar en la Tabla de figuras que acabamos de ver y ver que son los mismos valores que veíamos antes [aquí](#).

Por ejemplo en la última figura de la Tabla vemos el valor: 2C D3 0D 00, que en hex puro es: 00 0D D3 2C y si ahora le sumamos la image base: 00 40 00 00, tendremos:

$$00\ 40\ 00\ 00 + 00\ 0D\ D3\ 2C = 00\ 4D\ D3\ 2C$$

Que es una de las líneas de la Tabla que veíamos antes en la [pág 20](#).

Bueno hemos averiguado ya bastantes cosas (con sólo 2 pulsaciones de F7). Ahora antes de pasar a otras cosas, os diré por si no lo habías notado ya, que el byte marcado en amarillo en la figura anterior (375F58) es la longitud en caracteres del nombre de la DLL. Podemos ir mirando en las figuras incluídas antes y lo comprobaremos.

Bueno, puede decir alguno, todo esto es muy interesante, pero ¿y las APIs? ¿ dónde están las APIs? Sigamos un poco a ver si las encontramos, que va a ser que sí.

8.2. Localización y descryptado de APIs correspondientes a entradas buenas de la IAT

Recordemos antes de seguir, que estamos tratando la GDI32.DLL de la que ya se ha escrito la primera entrada de la IAT en 4DD038.

Estamos parados en Olly aquí:



Newbie Vs UnknowPacker - Revisitado - by El Cid

The screenshot shows a debugger window with assembly code on the left and registers on the right. Annotations include:

- A red box around the instruction `MOVZX EAX, BYTE PTR DS:[EDX-1]` and the subsequent `ADD EDX, EAX` and `INC EDX` instructions.
- A blue box around the instruction `JMP 00373083`.
- A blue box around the instruction `MOV ECX, DWORD PTR DS:[EDX]`.
- A blue box around the instruction `AND ECX, 7FFFFFFF`.
- A blue box around the instruction `PUSH ECX`.
- A blue box around the instruction `PUSH EDX`.
- A blue box around the instruction `SHL ECX, 5`.
- A blue box around the instruction `PUSH 4`.
- A blue box around the instruction `MOV ECX, DWORD PTR DS:[EDX]`.
- A blue box around the instruction `AND ECX, 7FFFFFFF`.
- A blue box around the instruction `PUSH ECX`.
- A blue box around the instruction `PUSH EDX`.
- A blue box around the instruction `SHL ECX, 5`.
- A blue box around the instruction `PUSH 4`.
- A blue box around the instruction `MOV ECX, DWORD PTR DS:[EDX]`.
- A blue box around the instruction `AND ECX, 7FFFFFFF`.
- A blue box around the instruction `PUSH ECX`.
- A blue box around the instruction `PUSH EDX`.
- A blue box around the instruction `SHL ECX, 5`.
- A blue box around the instruction `PUSH 4`.

Registers (FPU) window shows:

- EAX: 77EF7786 GDI32.CreateRectRgn
- ECX: 7C929280 ntdll.7C929280
- EDX: 0037677F
- EBX: 003D093D ASCII "hx9"
- ESP: 0012FFA0
- EBP: FFF71947
- ESI: 77EF0000 GDI32
- EDI: 0040D038 L2Walker.0040D038

Address Hex dump window shows:

Address	Hex dump	ASCII
0037673E	51 FE 9F BE 51 BF D1 D2 B5 52 93 5C 9F 55 91 91	Q=f%0;0EAr0\fuæ
0037674E	01 BE 00 08 B5 7F 51 1F D2 D1 FE DF 75 51 7F 00	0%.000Q7E0=uQ0.
0037675E	0B 15 D1 BF 93 5C 9F 95 1E BE 1E 7F 00 38 00 00	0B15\foA%0.8%.
0037676E	00 09 47 44 49 33 32 2E 44 4C 4C 00 3B 00 00 00	..GDI32.DLL;...
0037677E	00 97 B1 53 D3 71 53 B5 53 93 71 B5 13 32 00 00	uSsqSAS0qA!2.0
0037678E	17 53 71 F5 D2 F0 53 72 00 00 95 53 72 53 93 71	\$SqS0-Sr..0SrS0q
0037679E	97 72 D2 F1 B5 13 32 00 0C 95 53 71 75 53 F0 71	urE!A!2..0SquS-q
003767AE	D7 72 D2 13 32 00 0C 95 12 31 53 75 12 57 F0 00	irE!2.0U!1Su0W-

En la figura he marcado las principales relaciones que hay entre los diversos campos. Esta es la situación de partida.

La zona marcada de amarillo termina en un cero binario y podría ser un string pero no vemos que tengan ningún sentido los caracteres.

Lo más importante es que, después de ejecutarse las 3 instrucciones enmarcadas:

- EDX quedará direccionado a 37677F
- EDI quedará direccionado a la nueva posición de la IAT.

Damos F7 hasta llegar a la instrucción LOOP:

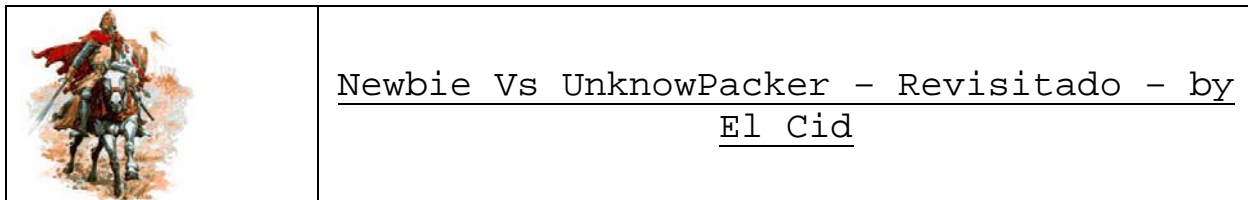
The screenshot shows the same debugger window after execution. The instruction `LOOP SHORT 00373897` is highlighted. The registers window shows:

- EAX: 00000000
- ECX: 0000003B
- EDX: 00376780
- EBX: 003D093D ASCII "hx9"
- ESP: 0012FFA4
- EBP: FFF71947
- ESI: 77EF0000 GDI32.77EF0000
- EDI: 0040D03C L2Walker.0040D03C
- EIP: 00373912

Address Hex dump window shows:

Address	Hex dump	ASCII
0037673E	51 FE 9F BE 51 BF D1 D2 B5 52 93 5C 9F 55 91 91	Q=f%0;0EAr0\fuæ
0037674E	01 BE 00 08 B5 7F 51 1F D2 D1 FE DF 75 51 7F 00	0%.000Q7E0=uQ0.
0037675E	0B 15 D1 BF 93 5C 9F 95 1E BE 1E 7F 00 38 00 00	0B15\foA%0.8%.
0037676E	00 09 47 44 49 33 32 2E 44 4C 4C 00 3B 00 00 00	..GDI32.DLL;...
0037677E	00 97 B1 53 D3 71 53 B5 53 93 71 B5 13 32 00 00	uSsqSAS0qA!2.0
0037678E	17 53 71 F5 D2 F0 53 72 00 00 95 53 72 53 93 71	\$SqS0-Sr..0SrS0q
0037679E	97 72 D2 F1 B5 13 32 00 0C 95 53 71 75 53 F0 71	urE!A!2..0SquS-q
003767AE	D7 72 D2 13 32 00 0C 95 12 31 53 75 12 57 F0 00	irE!2.0U!1Su0W-
003767BE	06 76 D2 32 53 75 12 00 09 17 53 71 B5 13 32 B7	uVE2Su+..\$SqA!2A
003767CE	13 50 00 10 07 B1 53 03 71 53 55 13 73 00 13 13	+..NuS0S0S0S0S0

Aquí vemos los valores de EDX y EDI que habíamos previsto. Además vemos que después del byte azul 08, hay unos bytes que terminan también en cero binario y que he marcado de amarillo y naranja.



Pues bien, pensando que el packer para determinar la API de esta entrada vaya a utilizar los bytes marcados, voy a poner un MBP on access en todos ellos, así:

Address	Hex dump	ASCII
00376740	91 01 BE 00 0B B5 7F 51 1F D2 D1 FE DF 75 51 7F	æ0%,.8A0QVÉ0= uQ0
00376750	00 0B 15 01 BF 93 5C 9F 95 1E BE 1E 7F 00 38 00	.880h0\fo8#A0.88
00376760	00 00 09 47 44 49 33 32 2E 44 4C 4C 00 38 00 00	...GD132.DLL;...
00376770	00 00 97 B1 53 03 71 53 B5 53 93 71 B5 13 32 00	...uSéqSASôqA!!2.
00376780	08 17 53 71 F5 02 F0 53 72 00 00 95 53 72 53 93	Q#Sq3E-Sr..ôSrSô
00376790	71 97 72 02 F1 B5 13 32 00 0C 95 53 71 75 53 F0	qûrE!A!2..ôSquS-
003767A0	71 07 72 02 13 32 00 08 56 12 31 53 75 12 57 F0	qîrE!2. U#1Su#W-
003767B0	00 06 76 02 32 53 75 12 00 09 17 53 71 B5 13 32	.#vE2Su+.#SqA!2
003767C0	B7 12 F0 00 10 97 B1 53 03 71 53 F5 12 72 00 13	A+-,W#SéqS8+r!!
003767D0	12 32 B5 13 32 00 08 F5 12 72 00 72 02 32 53 00	#2A!2.88+r8rE2S.
003767E0	11 06 32 71 53 B1 91 53 93 71 97 72 02 F1 B5 53	!i2qS#SôqûrE!AS

Ahora la idea es dar run a ver si accede a alguno de ellos y ver lo que hace. En todo caso como tenemos puesto el BP en 373904 (que es donde escribe la entrada API en la IAT), si nos equivocamos y no para en estos bytes, parará en el BP indicado y ya pensaremos otra cosa para el siguiente ciclo. Esta es la ventaja de trabajar con una zona de IAT que tenga muchas entradas como ya he dicho, es como descubrir un juego de prestidigitación que nos repite muchas veces el mago, al final, si estamos atentos, lo cazaremos.

Bueno pues damos Run. Paramos aquí:

003738C8	LEA EDI,[EBP+L2Walker.403BC7]
003738D1	XOR EAX,EAX
003738D3	LODS BYTE PTR DS:[ESI]
003738D4	JMP SHORT 003738DD
003738D6	ROL AL,8
003738D9	NOT AL
003738DB	STOS BYTE PTR ES:[EDI]
003738DC	LODS BYTE PTR DS:[ESI]
003738DD	OR EAX,EAX
003738DF	JNE SHORT 003738D6
003738E1	STOS BYTE PTR ES:[EDI]
003738E2	POPAD
003738E3	LEA EDX,[EBP+L2Walker.403BC7]

Address	Hex dump
00376740	91 01 BE 00 0B B5 7F 51 1F D2 D1 FE DF 75 51 7F
00376750	00 0B 15 01 BF 93 5C 9F 95 1E BE 1E 7F 00 38 00
00376760	00 00 09 47 44 49 33 32 2E 44 4C 4C 00 38 00 00
00376770	00 00 97 B1 53 03 71 53 B5 53 93 71 B5 13 32 00
00376780	08 17 53 71 F5 02 F0 53 72 00 00 95 53 72 53 93
00376790	71 97 72 02 F1 B5 13 32 00 0C 95 53 71 75 53 F0
003767A0	71 07 72 02 13 32 00 08 56 12 31 53 75 12 57 F0

, está accediendo al primer byte de la zona MBP, ¡qué ordenado es nuestro chico!, je, je.

Veamos lo que hace con él.

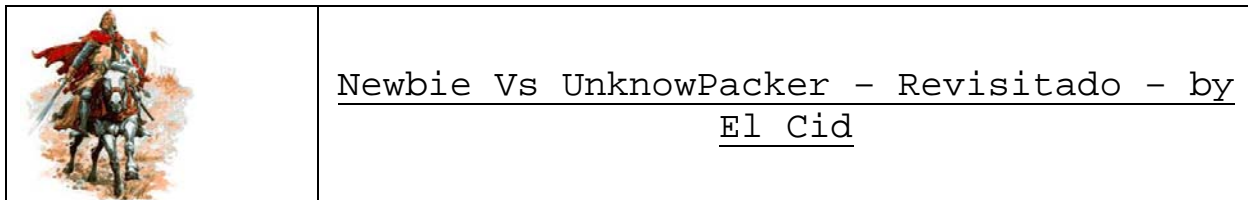
1º Lo carga (LODS) en EAX (que previamente ha borrado con XOR EAX, EAX)

2º Salta a 3738DD y mira a ver si EAX es cero. Caso contrario continúa.

Vamos a llegar hasta aquí con F7:

003738C8	LEA EDI,[EBP+L2Walker.403BC7]
003738D1	XOR EAX,EAX
003738D3	LODS BYTE PTR DS:[ESI]
003738D4	JMP SHORT 003738DD
003738D6	ROL AL,8
003738D9	NOT AL
003738DB	STOS BYTE PTR ES:[EDI]
003738DC	LODS BYTE PTR DS:[ESI]
003738DD	OR EAX,EAX
003738DF	JNE SHORT 003738D6
003738E1	STOS BYTE PTR ES:[EDI]
003738E2	POPAD

Aquí tenemos claramente un bucle que:



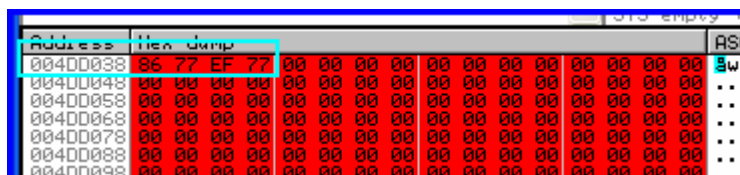
- Hace algunas operaciones con el byte tomado (ROL y NOT)
- Lo guarda en algún sitio (señalado por EDI)
- Vuelve a cargar otro byte (de ESI)
- Comprueba si es cero y si no lo es va a otro ciclo
- Si lo es guarda el último byte y acaba el ciclo.

Debemos recordar que tanto la instrucción LODS como la STOS, progresan automáticamente el índice (ESI y EDI respectivamente).

Lo primero que deducimos es que de la zona marcada antes con el MBP, nos bastaría solo con la que va hasta el primer 00h (la amarilla), ya que el bucle acaba ahí y luego irá a otro.

Vamos a ver una imagen general que incluya los registros, para hacer una observación importante. Estamos aquí:

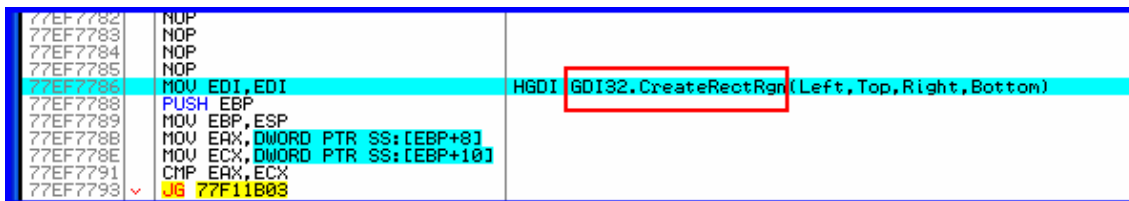
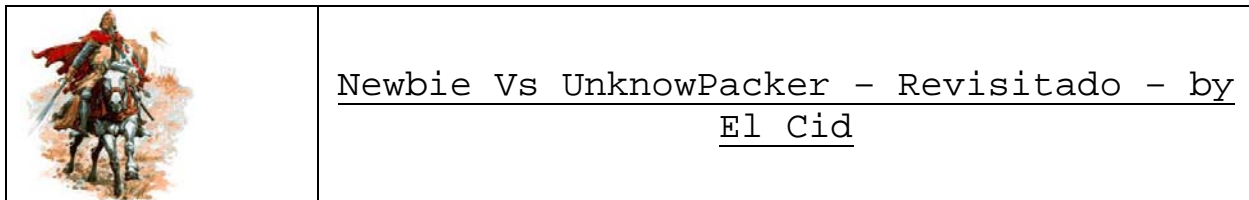
Si nos fijamos en EDI vemos que está apuntando a un string con el nombre de una API. Pero ¿no habíamos dicho que estábamos al comienzo de un ciclo para precisamente obtener el nombre de una API? ¡¡ Si, en realidad casi no hemos hecho nada desde que escribimos la dirección de la API anterior en la IAT, que fue la primera de la zona por cierto!! Entonces, es un poco raro, que sin empezar, como quien dice, ya tengamos el nombre ahí formadito. Vamos a hacer una cosa, vamos a mirar cual era la API que se escribió en primer lugar en esta zona. Si recordamos estaba en la pos 4DD038, todavía están puestos allí los MBPs. Tenemos mil formas de ir allí. Vamos allí:



Ahora marcamos la DWord 77EF7786 y hacemos Follow in Disassembler:



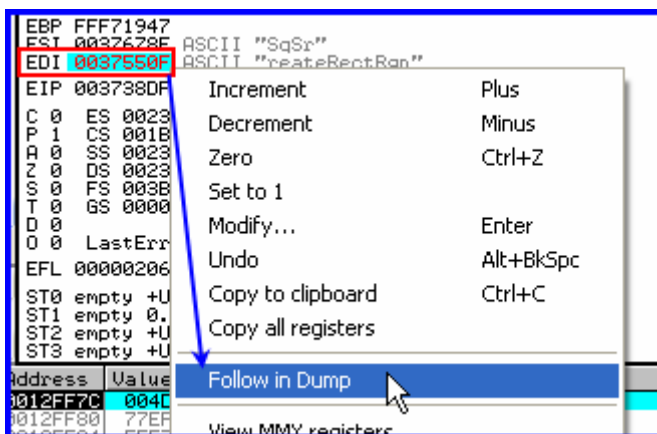
Y vemos ésto:



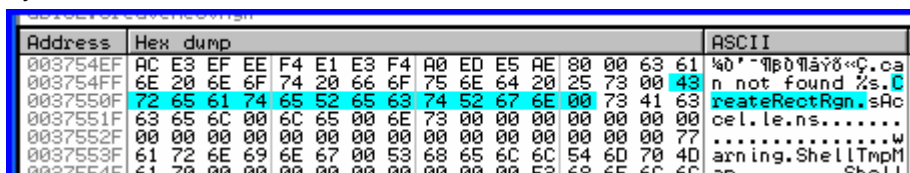
¡Anda! ¡¡Es la misma!! Ahora lo entiendo, resulta que lo que estamos viendo apuntado por EDI, es la API anterior, de la que ya ha calculado la dirección y la ha escrito en la IAT.

Vamos allí a mirar lo que hay:

Hacemos:



, y vemos ésto:



Aquí está el nombre de la API anterior, acabado en un cero binario. No sabemos si el nombre de las APIs lo formará siempre en esa misma posición, o irá variando de unas a otras o por DLLs o por cualquier otro criterio.

Bueno, pues una vez visto ésto, vamos a seguir trazando un poco a ver lo que averiguamos.

Desde 3738DF donde estamos parados, con EAX=17h, damos F7 hasta llegar a:



Newbie Vs UnknowPacker - Revisitado - by El Cid

```
003738D1 XOR EAX,EAX
003738D3 LODS BYTE PTR DS:[ESI]
003738D4 JMP SHORT 003738DD
003738D6 ROL AL,8
003738D9 NOT AL
003738DB STOS BYTE PTR ES:[EDI]
003738DC LODS BYTE PTR DS:[ESI]
003738DD OR EAX,EAX
003738DF JNE SHORT 003738D6
003738E1 STOS BYTE PTR ES:[EDI]
003738E2 POPAD
003738E3 LEA EDX,[EBP+L2Walker.403BC7]
003738E9 PUSH EDX
003738EA PUSH ESI
003738EB LEA ECX,[EBP+L2Walker.403BC7]
003738ED JNE SHORT 003738D6
```

AL=47 ('G')
[0037550F]=72 ('r')

Ha manipulado EAX=17 hasta obtener el valor AL=47, que Olly nos dice que es la letra “G”. O sea que lo que está haciendo es desencriptar la API que estaba encriptada en las posiciones que vimos que seguían a los nombres de las DLLs. Además vemos que lo va a ubicar en la pos 37550F, que pertenece a la misma zona de antes que comenzaba en 37550E.

Para que se forme la API en una pasada, hacemos lo siguiente:

- Desactivar este MBP

Address	Hex dump	ASCII
0037678F	53 71 F5 02 F0 53 72 00 0D 95 53 72 53 93 71 97	Sq3E-Sr..ôSrSôqu
0037679F	72 02 F1 B5 13 32 00 0C 95 53 71 75 53 F0 71 07	rE+Â!!2..ôSquS-qî
003767AF	72 02 13 32 00 08 56 12 31 53 75 12 57 F0 00 06	rE!!2.üU*1Su#W-.+
003767BF	76 02 32 53 75 12 00 09 17 53 71 B5 13 32 B7 12	wE2Su\$...\$Snâ!!2â\$

Memory breakpoints					
Address	Size	Module	Type	Status	Comment
0037678F	00000018	kernel32.dll	RW	Disabled	
00451715	00000001	L2Walker	W	Disabled	OEP
00451716	00000001	L2Walker	W	Disabled	

- Poner un BP en la salida del bucle

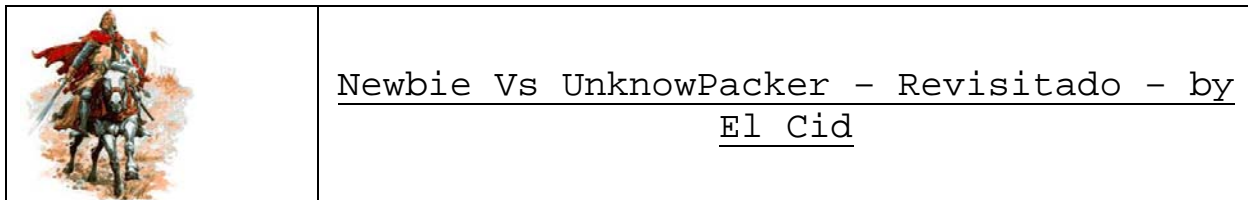
```
003738D1 XOR EAX,EAX
003738D3 LODS BYTE PTR DS:[ESI]
003738D4 JMP SHORT 003738DD
003738D6 ROL AL,8
003738D9 NOT AL
003738DB STOS BYTE PTR ES:[EDI]
003738DC LODS BYTE PTR DS:[ESI]
003738DD OR EAX,EAX
003738DF JNE SHORT 003738D6
003738E1 STOS BYTE PTR ES:[EDI]
003738E2 POPAD
003738E3 LEA EDX,[EBP+L2Walker.403BC7]
003738E9 PUSH EDX
```

- Dar Run y después dar F7: estamos aquí

```
003738D1 XOR EAX,EAX
003738D3 LODS BYTE PTR DS:[ESI]
003738D4 JMP SHORT 003738DD
003738D6 ROL AL,8
003738D9 NOT AL
003738DB STOS BYTE PTR ES:[EDI]
003738DC LODS BYTE PTR DS:[ESI]
003738DD OR EAX,EAX
003738DF JNE SHORT 003738D6
003738E1 STOS BYTE PTR ES:[EDI]
003738E2 POPAD
003738E3 LEA EDX,[EBP+L2Walker.403BC7]
003738E9 PUSH EDX
```

Y en la zona de antes, vemos la API descodificada:

Address	Hex dump	ASCII
003754EE	F3 AC E3 EF EE F4 E1 E3 F4 A0 ED E5 AE 80 00 63	%d0'~p0qay8<<C.c
003754FE	61 6E 20 6E 6F 74 20 66 6F 75 6E 64 20 25 73 00	an not found %s.
0037550E	47 65 74 50 69 78 65 6C 00 74 52 67 6E 00 73 41	GetPixel.tRgn.sA
0037551E	63 63 65 6C 00 6C 65 00 6E 73 00 00 00 00 00 00	ccel.le.ns.....
0037552E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0037553E	77 61 72 6E 69 6E 67 00 53 68 65 6C 6C 54 6D 70	warning.ShellTmp
0037554E	4D 61 70 00 00 00 00 00 00 00 00 00 53 68 65 6C	Map.....Shell

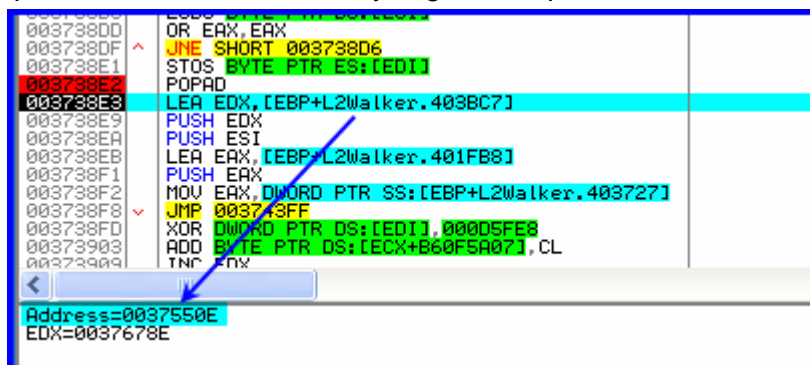


Ahora quitamos el BP de 3738E2 para no ser detectados.

8.3. Escritura de la dirección buena de una API en la IAT

Una vez desencriptado el nombre de la API, por lógica, lo que hará el packer es obtener su dirección para después escribirla en la IAT.

Deberemos, por tanto trazar. Damos F7 y llegamos aquí:



```

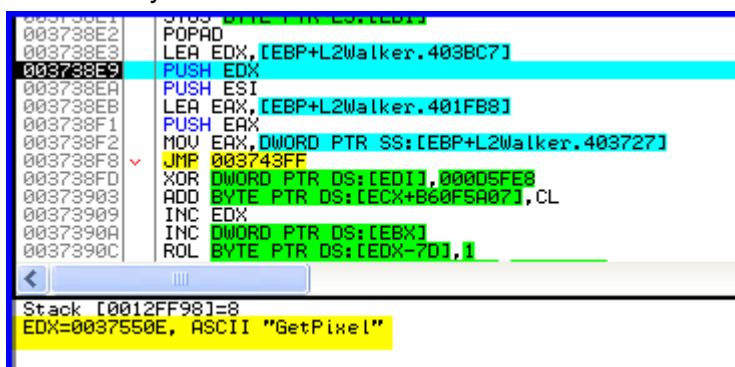
003738D0  OR EAX,EAX
003738D1  JNE SHORT 003738D6
003738E1  STOS BYTE PTR ES:[EDI]
003738E2  POPAD
003738E3  LEA EDX,[EBP+L2Walker.403BC7]
003738E9  PUSH EDX
003738EA  PUSH ESI
003738EB  LEA EAX,[EBP+L2Walker.401FB8]
003738F1  PUSH EAX
003738F2  MOV EAX,DWORD PTR SS:[EBP+L2Walker.403727]
003738F8  JMP 003743FF
003738FD  XOR DWORD PTR DS:[EDI],00005FE8
00373903  ADD BYTE PTR DS:[ECX+B60F5A07],CL
00373909  INC EDX
0037390A  INC DWORD PTR DS:[EBX]
0037390C  ROL BYTE PTR DS:[EDX-7D],1

```

Address=0037550E
EDX=0037678E

, donde vemos que en EDX va a poner el valor 37550E con lo que [EDX]→String_Nombre_Api

Damos F7 y lo vemos:



```

003738E1  STOS BYTE PTR ES:[EDI]
003738E2  POPAD
003738E3  LEA EDX,[EBP+L2Walker.403BC7]
003738E9  PUSH EDX
003738EA  PUSH ESI
003738EB  LEA EAX,[EBP+L2Walker.401FB8]
003738F1  PUSH EAX
003738F2  MOV EAX,DWORD PTR SS:[EBP+L2Walker.403727]
003738F8  JMP 003743FF
003738FD  XOR DWORD PTR DS:[EDI],00005FE8
00373903  ADD BYTE PTR DS:[ECX+B60F5A07],CL
00373909  INC EDX
0037390A  INC DWORD PTR DS:[EBX]
0037390C  ROL BYTE PTR DS:[EDX-7D],1

```

Stack [0012FF98]=8
EDX=0037550E, ASCII "GetPixel"

Podemos seguir trazando con F8 o dar Run lo que prefiera cada uno. Se puede hacer con F8, pero no con F7 porque hay CALLs que detectan el traceo y se entra en bucles infinitos (con RDTSC). Haciéndolo con F8 se ve como para saber la dirección de la API llama a GetProcAddress con el nombre que ha desencriptado, en esta ocasión GetPixel. Pero ésto no es necesario para lo que resta, ni para desempacar el programa. Yo doy Run que es más rápido, llegando aquí:



Newbie Vs UnknowPacker - Revisitado - by El Cid

```
003738FF CALL 00374663
00373904 MOV DWORD PTR DS:[EDI],EAX
00373906 POP EDX
00373907 MOVZX EAX,BYTE PTR DS:[EDX-1]
00373908 ADD EDX,EAX
0037390D INC EDX
0037390E ADD EDI,4
00373911 POP ECX
00373912 LOOP SHORT 00373897
00373914 JMP 00373D83
00373919 MOV ECX,DWORD PTR DS:[EDX]
0037391B AND ECX,7FFFFFFF
00373921 PUSH ECX
00373922 PUSH EDX
```

EAX=77EFB479 (GDI32.GetPixel)
[004DD03C]=0

Address	Hex dump	As
004DD038	86 77 EF 77 00 00 00 00 00 00 00 00 00 00 00 00	...
004DD048	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
004DD058	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
004DD068	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
004DD078	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...

Es el lugar del que hemos partido, o sea que hemos hecho un ciclo completo del bucle que concluirá a continuación con la instrucción MOV, con la escritura de la entrada en la IAT.

Damos F7 y :

```
003738FF CALL 00374663
00373904 MOV DWORD PTR DS:[EDI],EAX
00373906 POP EDX
00373907 MOVZX EAX,BYTE PTR DS:[EDX-1]
00373908 ADD EDX,EAX
0037390D INC EDX
0037390E ADD EDI,4
00373911 POP ECX
00373912 LOOP SHORT 00373897
00373914 JMP 00373D83
00373919 MOV ECX,DWORD PTR DS:[EDX]
```

Top of stack [0012FF9C]=0037678E
EDX=ntdll.7C98E178

Address	Hex dump	As
004DD038	86 77 EF 77 79 B4 EF 77 00 00 00 00 00 00 00 00	...
004DD048	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
004DD058	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...

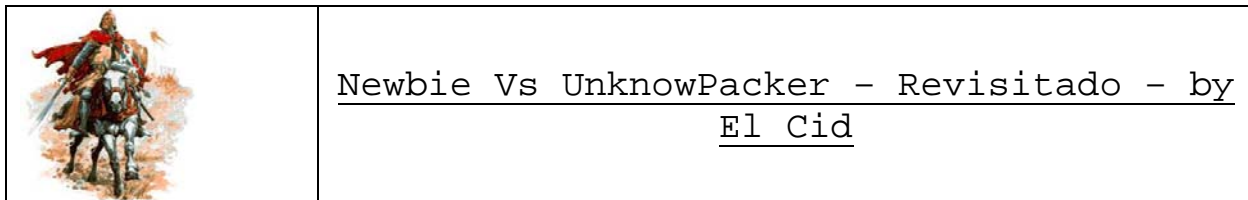
, efectivamente ha puesto la segunda entrada correcta en esta zona de la IAT. Podemos ahora ir viendo como se rellena la zona de entradas buenas, es decir, direcciones reales de APIs, quitando todos los BPs que tengamos puestos y dejando tan solo el MBP en la zona de memoria correspondiente a esta parte de la IAT, es decir 4DD038-4DD124 y cada vez pararemos en una entrada, así hasta el final de la zona.

El que quiera puede hacerlo, sin embargo nosotros no lo haremos pues lo que ahora nos interesa, es solucionar el problema de las entradas malas y una vez que sepamos como hacerlo, entonces sí regresaremos a este punto para la creación de la IAT tanto para entradas buenas como para malas (pero arregladas por nosotros).

8.4. Análisis del cambio de zona: de Buena a Mala

Vamos a ver ahora un aspecto que resulta muy importante en este sistema del JMP-CALL MÁGICO. Se trata del cambio o transición de una zona buena a otra mala o viceversa.

¿Por qué es importante? Pues porque si las entradas de zona buena reciben un tratamiento y las de mala otro distinto (entradas redireccionadas) no cabe duda que en



las instrucciones del programa que encontremos cuando se acaba una zona de un tipo y comienza otra zona de un tipo distinto, habrá muchas probabilidades de que se encuentre el salto mágico.

Aquí tenemos una situación bastante favorable ya que tenemos una zona buena de la IAT (la 4DD000) con una única entrada, por lo que nada más escribirla, el programa deberá ir a las instrucciones que decimos. Pues vamos a analizar trazando un poco estas instrucciones.

AVISO IMPORTANTE: si alguien ha ido poniendo comentarios en Olly, explicando líneas de código, aclarando condiciones, saltos que se toman o que no, etc, etc, LOS PERDERÁ TODOS, ya que al estar empacado el programa, una vez reiniciado Olly no es capaz de colocarlos en su sitio de nuevo y aunque están dentro del .UDD no nos servirán. Lo digo para que estéis avisados y no os quedéis con la cara que me quedé yo, cuando me pasó lo que acabo de decir.

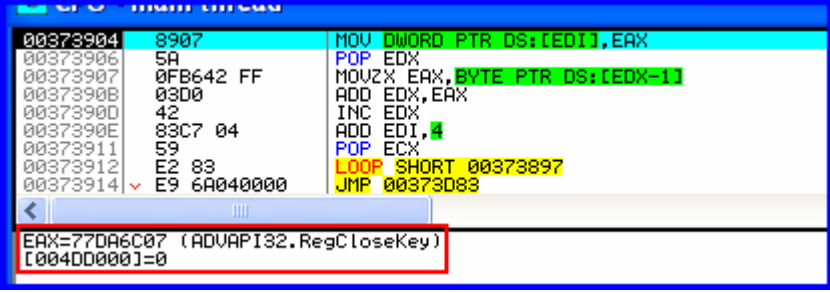
El que avisa no es traidor, je, je

Reiniciamos y Ponemos a cero el byte de IsDebuggerPresent

Ponemos un MBP en 4DD000.

Damos Run

Olly se detiene allí la primera vez que va a escribir ceros, que no nos interesa. **Damos Run** de nuevo y vuelve a parar ahí:



```

00373904 8907 MOV DWORD PTR DS:[EDI],EAX
00373906 5A POP EDX
00373907 0FB642 FF MOVZX EAX,BYTE PTR DS:[EDX-1]
0037390B 03D0 ADD EDX,EAX
0037390D 42 INC EDX
0037390E 83C7 04 ADD EDI,4
00373911 59 POP ECX
00373912 E2 83 LOOP SHORT 00373897
00373914 E9 6A040000 JMP 00373D83
  
```

EAX=77DA6C07 (ADVAPI32.RegCloseKey)
[004DD000]=0

, donde vemos que se va a escribir la dirección de la API en la IAT.

Bien, vemos que pertenece a la ADVAPI32.DLL, como ya sabíamos.

¿Cuál será la siguiente DLL a procesar? Realmente ya lo hemos visto pero lo repetiremos.

Vayamos a esta sección de la memoria y busquemos esta cadena de caracteres.

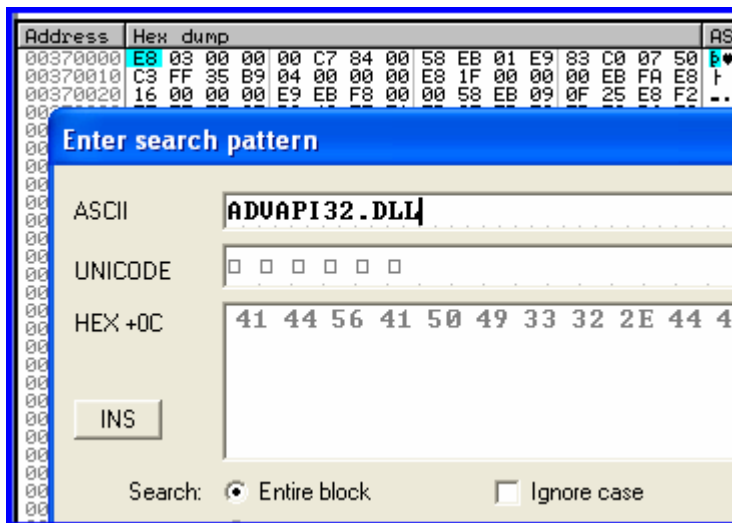
En:

Address	Hex dump
00370000	E8 03 00 00 00 C7 84 00 58 EB 01 E9 83 C0 07 50
00370010	C3 FF 35 B9 04 00 00 00 E8 1F 00 00 00 EB FA E8
00370020	16 00 00 00 E9 EB F8 00 00 58 EB 09 0F 25 E8 F2
00370030	FF FF 0F B9 49 75 F1 EB 05 EB F9 EB F0 D6 E8
00370040	10 00 00 00 00 13 01 00 FF 01 00 00 00 00 FF 04

, hacemos:



Newbie Vs UnknowPacker - Revisitado - by El Cid



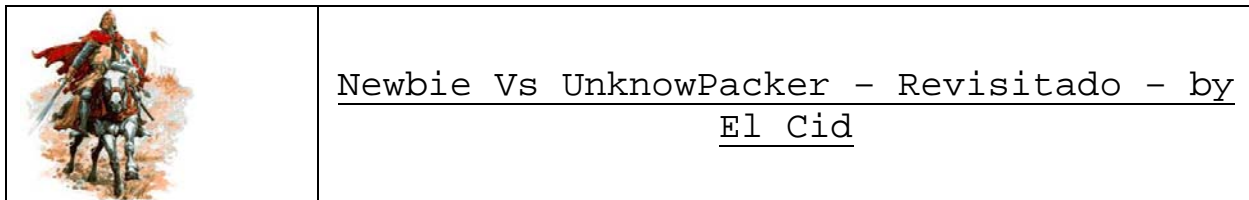
, y vamos aquí:

Address	Hex dump	ASCII
00375F59	41 44 56 41 50 49 33 32 2E 44 4C 00 01 00 00	ADVAPI32.DLL.0..
00375F69	00 0B B5 53 13 97 72 12 91 53 96 53 D0 00 4C D3	.0AS!!ur#eS0S3.LÉ
00375F79	00 00 0A 55 53 45 52 33 32 2E 44 4C 00 88 00	...USER32.DLL.e.
00375F89	00 00 0E 52 FE FF 51 BE 5E B1 51 BF D1 73 D1 91	.ÇAR= Q#^Q70s0æ
00375F99	BF 00 0A 15 D1 BF 93 DF 71 D2 D1 FE DF 00 09 15	7...\$070qE0#...\$
00375FA9	D1 BF 13 5E FE B1 1E 1F 00 12 15 D1 BF 13 5E FE	07!!^#▲7.#\$07!!^#

, donde hemos encontrado la cadena y ya vemos la siguiente que es USER32.DLL. Entonces vamos a poner un MBP en USER32.DLL para que Olly se detenga cuando acceda a este string:

Address	Hex dump	ASCII
00375F59	41 44 56 41 50 49 33 32 2E 44 4C 00 01 00 00	ADVAPI32.DLL.0..
00375F69	00 0B B5 53 13 97 72 12 91 53 96 53 D0 00 4C D3	.0AS!!ur#eS0S3.LÉ
00375F79	00 00 0A 55 53 45 52 33 32 2E 44 4C 00 88 00	...USER32.DLL.e.
00375F89	00 00 0E 52 FE FF 51 BE 5E B1 51 BF D1 73 D1 91	.ÇAR= Q#^Q70s0æ
00375F99	BF 00 0A 15 D1 BF 93 DF 71 D2 D1 FE DF 00 09 15	7...\$070qE0#...\$

Porque cuando estemos accediendo al nombre de la siguiente DLL, podemos considerar que el siguiente ciclo ya ha comenzado o está a punto de comenzar.



Damos Run y paramos aquí:

```

CPU - main thread, module ntdll
7C91126F 0BFF OR EDI,EDI
7C911271 74 1E JE SHORT 7C911291
7C911273 83C9 FF OR ECX,FFFFFFFF
7C911276 33C0 XOR EAX,EAX
7C911278 F2:AE REPNE SCAS BYTE PTR ES:[EDI]
7C91127A F7D1 NOT ECX
7C91127C 81F9 FFFF0000 CMP ECX,0FFFF
7C911282 76 05 JBE SHORT 7C911289
7C911284 B9 FFFF0000 MOV ECX,0FFFF

ECX=FFFFFFFF (decimal -1.)
AL=00
[00375F7C]=55 ('U')

Address Hex dump ASCII
00375F59 41 44 56 41 50 49 33 32 2E 44 4C 4C 00 01 00 00 ADVAPI32.DLL.0..
00375F69 00 0B B5 53 13 97 72 12 91 53 96 53 00 00 4C D3 .AS!ur#sSs.L.
00375F79 00 00 0A 55 53 45 52 33 32 2E 44 4C 4C 00 88 00 ...USER32.DLL.
00375F89 00 00 0E 52 FE FF 51 BE 5E B1 51 BF D1 73 D1 91 .CAR= 0#^0h 0s0#
00375F99 BF 00 0A 15 D1 BF 93 DF 71 D2 D1 FE DF 00 09 15 7..30h0=qE0#...3

```

Estamos dentro del módulo NTDLL. Ahora quitamos el MBP puesto que ya ha cumplido su papel y así no parará más veces en él y damos (Execute until return + F7), 3 veces hasta volver al código del usuario o sea del packer. Olly tiene una opción que hace precisamente esto (Alt+F9) pero a mí no me gusta usarla porque en alguna ocasión se me ha colgado (aquí, sin ir más lejos, he hecho la prueba y se me cuelga). Aparecemos aquí:

```

00373855 0BC0 OR EAX,EAX
00373857 75 1E JNE SHORT 00373877
00373859 56 PUSH ESI
0037385A 8085 271F4000 LEA EAX,[EBP+L2Walker.401F27]
00373860 50 PUSH EAX
00373861 8B85 2F374000 MOV EAX,DWORD PTR SS:[EBP+L2Walker.40372F]
00373867 E9 930B0000 JMP 003743FE
0037386C FF15 0BC07505 CALL DWORD PTR DS:[575C00B]
00373872 E9 9D0D0000 JMP 00374614
00373877 0FB64E FF MOVZX ECX,BYTE PTR DS:[ESI-1]
0037387B 03F1 ADD ESI,ECX
0037387D 8BD6 MOV EDX,ESI
0037387F 8BF0 MOV ESI,EAX
00373881 42 INC EDX
00373882 8B0A MOV ECX,DWORD PTR DS:[EDX]
00373884 81E1 00000000 AND ECX,00000000
0037388A 0BC9 OR ECX,ECX
0037388C 0F85 87000000 JNE 00373919
00373892 8B0A MOV ECX,DWORD PTR DS:[EDX]
00373894 83C2 04 ADD EDX,4
00373897 51 PUSH ECX
00373898 0FB602 MOVZX EAX,BYTE PTR DS:[EDX]
0037389B 0BC0 OR EAX,EAX
0037389D 75 27 JNE SHORT 003738C6
0037389F 42 INC EDX
003738A0 52 PUSH EDX
003738A1 8B02 MOV EAX,DWORD PTR DS:[EDX]

```

Justo debajo hay un JNE que, en un primer momento, pensamos que podría ser el JMP mágico pero enseguida lo descartamos porque lo único que hace es comprobar si el Handle de la DLL es nulo, (pero fijaos en los 2 JNE que hay un poco más abajo y a ver si las direcciones a donde van 373919 y 3738C6 os resultan familiares a lo largo del tuto, je, je).

Pero bueno el caso es que esta zona es por la que se pasa cuando se cambia de DLL. Y fijémonos que se pasará siempre por ella (en transiciones de B→Mala y viceversa, así como de B→Buena. No hay M→Mala), puesto que el programa todavía no sabe si la DLL es de entradas buenas o malas (lo mira a continuación si os quereis entretener en comprobarlo).

Y ahora nos preguntamos ¿Cuál es la primera DLL que trata? Si en esta misma zona de la memoria hacemos:

Enter search pattern

ASCII

UNICODE
☐ ☐

HEX +00

Search:
☒ Entire block
 ☐ Ignore case
☐ Forward

, vemos:

Address	Hex dump	ASCII
0037580C	8D 02 3E 40 00 8D BD 2E 16 40 00 2B CF 33 C0 F3	1E>@.i<...@.+B3U%
0037581C	AA AB C3 28 D1 0D 00 0C 4B 45 52 4E 45 4C 33 32	~%t(0...KERNEL32
0037582C	2E 44 4C 4C 00 73 00 00 80 0C 93 D1 BF B2 51 9F	.DLL.s..C.δ0j0.f
0037583C	BF 05 7F 7F 1E 7F 00 0C 15 D1 BF B2 51 9F BF D5	γ'δδΔδ...Σ0j00.fγ'
0037584C	7F 7F 1E 7F 00 0C B2 1E 51 B1 B2 5E 71 7F 51 7F	δδΔδ...Σ0j000Qδδ
0037585C	5C 55 00 13 15 D1 BF 93 5C 9F BF D1 DE B5 5E 7F	\U.!!Σ0jδ\jγθiΔ^δ
0037586C	D1 91 BF 1E 7F 5C 55 00 14 15 D1 BF 13 5E FE B1	0æγΔδ\U.0Σ0γ!!^Σ
0037587C	1E 1F 9F B5 5E 7F D1 91 BF 1E 7F 5C 55 00 12 15	ΔγfΔ^δ0æγΔδ\U.δΣ

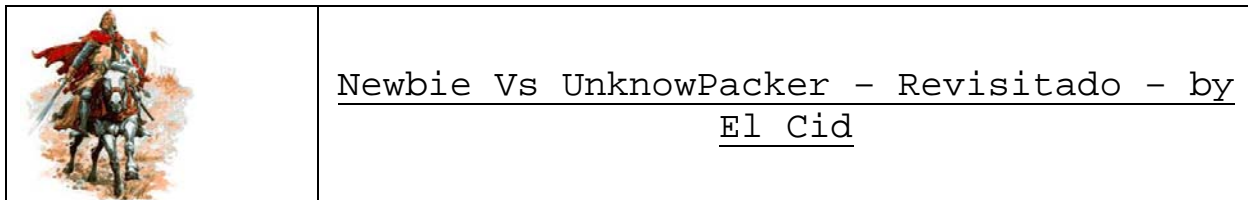
, que es Kernel32 y que es Mala por el 80h que tiene en su longitud. Ésto es muy importante para nosotros y lo usaremos al final del tuto para resolver el packer.

Y ésto es lo único que necesitamos saber por ahora.

9. Localización y descryptado de APIs correspondientes a entradas malas de la IAT
- Vamos ahora a estudiar las entradas malas. Para ello, como antes, trataremos de caer en una zona mala y empezar a trazar analizando lo que veamos.
- Sabemos que ya hemos pasado por 2 zonas malas y que de aquí al final de la IAT sólo nos queda 1 más.
- No sabemos el orden de las DLLs en la IAT, pero si fuera el mismo que el que hemos visto en esta [Tabla](#), tendremos que pasar por otras 6 zonas buenas antes de llegar a la mala que nos queda. Éso supone muchas entradas y bastantes probabilidades de equivocarnos en algo, echar a perder el trabajo hecho y malgastar el tiempo. Por tanto soy partidario de reiniciar todo (total no hemos parcheado nada, ni perderemos por tanto ningún trabajo hecho) ya que sabemos que la primera zona de la IAT que se trata es Mala, así que creo que llegaremos antes empezando de nuevo y además nos servirá de repaso.

Por tanto ¡¡allá vamos!!

REINICIAMOS



Ponemos a cero el byte de IsDebuggerPresent:

L2Walker.<ModuleEntryPoint>									
Address	Hex dump								
7FFDE000	00	00	00	00	FF	FF	FF	FF	00
7FFDE010	00	00	02	00	00	00	00	00	00
7FFDE020	00	10	91	7C	E0	10	91	7C	01
7FFDE030	00	00	00	00	00	00	00	00	00
7FFDE040	C0	05	99	7C	01	00	00	00	00
7FFDE050	00	00	6F	7F	88	06	6F	7F	00
7FFDE060	00	00	5D	7F	01	00	00	00	00

Configuramos así los MBPs, es decir todos desactivados excepto el referente a la zona que comienza en 4DD128, que sabemos que es la primera Mala por la que se pasa.

Memory breakpoints					
Address	Size	Module	Type	Status	Com
004DD000	00000004	L2Walker	W	Disabled	
004DD008	0000002C	L2Walker	W	Disabled	
004DD038	000000EC	L2Walker	W	Disabled	
004DD128	000001CC	L2Walker	W	Active	
004DD2F8	00000008	L2Walker	W	Disabled	
004DD304	00000024	L2Walker	W	Disabled	
004DD32C	00000008	L2Walker	W	Disabled	
004DD338	00000010	L2Walker	W	Disabled	
004DD34C	00000220	L2Walker	W	Disabled	
004DD570	00000004	L2Walker	W	Disabled	
004DD578	00000004	L2Walker	W	Disabled	
004DD580	00000038	L2Walker	W	Disabled	
004DD58C	00000054	L2Walker	W	Disabled	
004DD614	00000008	L2Walker	W	Disabled	

La idea es lanzar la ejecución y esperar a que pare cuando acceda por primera vez a esta zona y a partir de ahí trazar y ver lo que podemos averiguar.

Si alguien está pensando que se podía haber puesto un BP en la zona 37550E, que es donde descripta el nombre de las APIs (al menos para las zonas buenas), no lo podemos hacer porque esas posiciones de memoria no existen todavía. Las crea el packer después.

Así que **Damos Run** y paramos aquí:

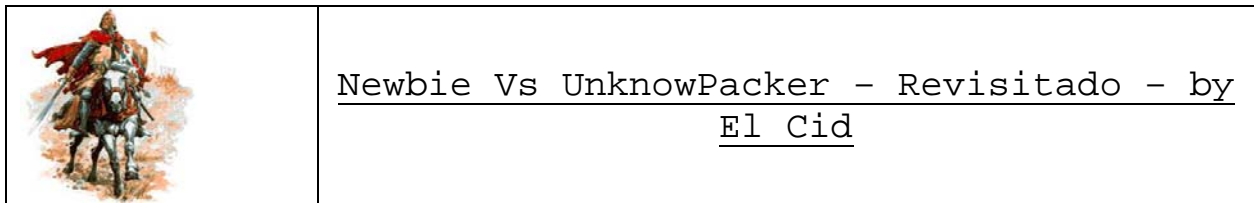
0037394C	PUSH ECX
0037394D	SUB EDI, DWORD PTR SS:[EBP+L2Walker.403737]
00373953	CMP EDI, -1
00373956	JE SHORT 00373960
00373958	ADD EDI, DWORD PTR SS:[EBP+L2Walker.403737]
0037395E	JMP SHORT 00373969
00373960	MOV DWORD PTR DS:[EDI], EAX
00373962	ADD EAX, 20
00373965	ADD EDI, 4
00373968	DEC ECX
00373969	OR ECX, ECX
0037396B	JNE SHORT 00373960
0037396D	POP ECX
0037396E	POP EAX

EAX=003E0000, Entry point of procedure [004DD128]=0

Address	Hex dump															ASCII
004DD128	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	...
004DD138	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	...
004DD148	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	...
004DD158	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	...

, donde vemos que va a escribir una entrada mala en la IAT.

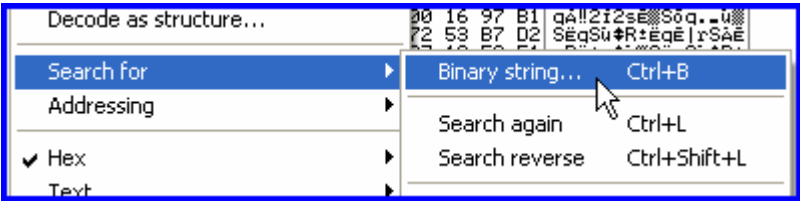
En este punto habíamos estado ya al principio del tuto y sabemos que esta DLL tiene 73 hex entradas en la IAT. Como es mala, debe haber una DWord 80 00 00 73 como "longitud" de la zona IAT correspondiente, que en formato memoria será: 73 00 00 80.



Pues buscaremos esa cadena binaria en la zona que contiene los nombres encriptados de las APIs, que recordamos que estaba (por ejemplo el de GDI32), en 376770. Vamos a esta posición:

Address	Hex dump	ASCII
00376770	47 44 49 33 32 2E 44 4C 4C 00 3B 00 00 00 00 97	GDI32.DLL;....u
00376780	B1 53 03 71 53 B5 53 93 B5 13 32 00 08 17 53	qSëqSASôqA!2. S
00376790	71 F5 02 F0 53 72 00 00 53 72 53 93 71 97 72	qSë-Sr..ôSrSôqur
003767A0	D2 F1 B5 13 32 00 0C 95 53 71 75 53 F0 71 07 72	E!A!2..ôSquS-qir
003767B0	D2 13 32 00 08 56 12 31 53 75 12 57 F0 00 06 76	E!2. U!1Su!W-.+v

, y hacemos:



Enter search pattern

ASCII

s . .

UNICODE

s

HEX +04

73 00 00 80

INS

Search:

☒ Entire block
☐ Forward

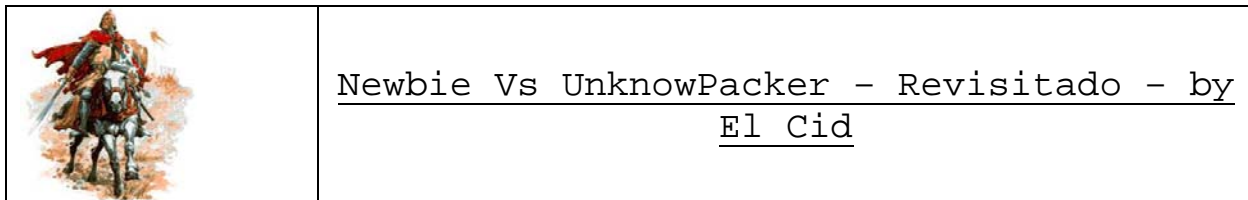
☒ Ignore case

, y vemos ésto:

Address	Hex dump	ASCII
003757F1	85 53 37 40 00 85 C0 74 07 61 B8 00 00 00 00 C3	âS7@.â!t.a@....t
00375801	6A 00 6A 00 FF B5 53 38 40 00 80 00 D2 3E 40 00	j..j. AS8@.i!e>@.
00375811	80 B0 2E 16 40 00 2B CF 33 C0 F3 A8 AB C3 28 D1	!c..@..+@3!~!t!@
00375821	00 00 0C 48 45 52 4E 45 4C 33 32 2E 44 4C 4C 00	...KERNEL32.DLL.
00375831	73 00 00 80 0C 93 D1 BF B2 51 9F BF D5 7F 7F 1E	...C.ôhQfj'ôôô.
00375841	7F 00 0C 15 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00	ô..ôhQfj'ôôô.
00375851	0C B2 1E 51 B1 B2 5E 71 7F 51 7F 5C 55 00 13 15	.##C##^qôQô\U.!!S
00375861	D1 BF 93 5C 9F BF D1 DE B5 5E 7F D1 91 BF 1E 7F	ô!ô\!f!ô!ô^ôô!ô
00375871	5C 55 00 14 15 D1 BF 13 5E FE B1 1E 1F 9F B5 5E	\U.!!Sô!!!^##!fô^
00375881	7F D1 91 BF 1E 7F 5C 55 00 12 15 D1 BF D2 1E B1	ôô!ô!ô\U.ôôô!ô

Tenemos todos los elementos vistos antes.
Las zonas marcadas en morado serán 2 APIs codificadas. Ponemos sendos MBPs en cada una y lo mismo hacemos con la posición 37550E, que ahora ya existe. O sea ésto:

Address	Hex dump	ASCII
003757F1	85 53 37 40 00 85 C0 74 07 61 B8 00 00 00 00 C3	âS7@.â!t.a@....t
00375801	6A 00 6A 00 FF B5 53 38 40 00 80 00 D2 3E 40 00	j..j. AS8@.i!e>@.
00375811	80 B0 2E 16 40 00 2B CF 33 C0 F3 A8 AB C3 28 D1	!c..@..+@3!~!t!@
00375821	00 00 0C 48 45 52 4E 45 4C 33 32 2E 44 4C 4C 00	...KERNEL32.DLL.
00375831	73 00 00 80 0C 93 D1 BF B2 51 9F BF D5 7F 7F 1E	...C.ôhQfj'ôôô.
00375841	7F 00 0C 15 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00	ô..ôhQfj'ôôô.
00375851	0C B2 1E 51 B1 B2 5E 71 7F 51 7F 5C 55 00 13 15	.##C##^qôQô\U.!!S
00375861	D1 BF 93 5C 9F BF D1 DE B5 5E 7F D1 91 BF 1E 7F	ô!ô\!f!ô!ô^ôô!ô
00375871	5C 55 00 14 15 D1 BF 13 5E FE B1 1E 1F 9F B5 5E	\U.!!Sô!!!^##!fô^



, y ésto:

Address	Hex dump	ASCII
0037550E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0037551E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0037552E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0037553E	77 61 72 6E 69 6E 67 00 53 68 65 6C 6C 54 6D 70	warning.ShellTm
0037554E	4D 61 7A 00 00 00 00 00 00 00 00 00 53 68 65 6C	Map.....Shel

Además desactivamos el MBP en la zona IAT 4DD128 porque ya sabemos que va a escribir unos valores, redireccionados, que no nos sirven para nada, así que si quiere escribirlos que los escriba el muchacho, que ya los cambiaremos nosotros después.

Damos Run y paramos aquí:

The screenshot shows a debugger window with assembly code on the left and registers on the right. Annotations include:

- Long en caract. API**: Points to the register ESI containing the value 00375836.
- Desencriptado para zona Mala**: Points to the instruction `JNE SHORT 003739F3`.
- Destino Nombre API desencriptado**: Points to the instruction `STOS BYTE PTR ES:[EDI]`.
- [00375836]=93**: Points to the value 93 in the memory dump at address 00375836.

Vemos los mismos elementos básicos que veíamos antes:

- El bucle de desencriptado, que es diferente del de las entradas buenas (tiene 2 instrucciones más)
- La posición de destino (EDI) para el nombre de API desencriptado, que es la misma que la de las entradas buenas
- La posición que marca el número de caracteres del nombre de la API que, en este caso, es 0Ch (12 decimal)

Damos Run de nuevo:



Newbie Vs UnknowPacker - Revisitado - by El Cid

Se detiene aquí:

```
003739F8  C0C0 03  ROL AL,3
003739FB  F6D0     NOT AL
003739FD  AA      STOS BYTE PTR ES:[EDI]
003739FE  49      DEC ECX
003739FF  33C0    XOR EAX,EAX
00373A01  0BC9    OR ECX,ECX
00373A03  75 EE   JNE SHORT 003739F8
00373A05  AA      STOS BYTE PTR ES:[EDI]
00373A06  61      POPAD

AL: 53 ('S')
[0037550E]=00

Address Hex dump ASCII
0037550E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0037551E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0037552E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0037553E 77 61 72 6E 69 6E 67 00 53 68 65 6C 6C 54 6D 70 warn
```

, justamente cuando va a poner el carácter "S" en la primera posición del campo.

Damos Run:

```
003739FB  F6D0     NOT AL
003739FD  AA      STOS BYTE PTR ES:[EDI]
003739FE  49      DEC ECX
003739FF  33C0    XOR EAX,EAX
00373A01  0BC9    OR ECX,ECX
00373A03  75 EE   JNE SHORT 003739F8
00373A05  AA      STOS BYTE PTR ES:[EDI]
00373A06  61      POPAD

AL: 65 ('e')
[0037550F]=00

Address Hex dump ASCII
0037550E 53 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0037551E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0037552E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

, y vemos que va a poner el siguiente.

Seguimos dando Run hasta que veamos que va a poner un cero binario que será el "carácter" final del nombre de la API (o finalizador de string), que es aquí:

```
003739F6  2C 55   SUB AL,55
003739F8  C0C0 03  ROL AL,3
003739FB  F6D0     NOT AL
003739FD  AA      STOS BYTE PTR ES:[EDI]
003739FE  49      DEC ECX
003739FF  33C0    XOR EAX,EAX
00373A01  0BC9    OR ECX,ECX
00373A03  75 EE   JNE SHORT 003739F8
00373A05  AA      STOS BYTE PTR ES:[EDI]
00373A06  61      POPAD
00373A07  8D95 C73B4000 LEA EDI,[EBP+L2Walker.403BC7]

AL: 00
[0037551A]=00


Address Hex dump ASCII
0037550E 53 65 74 4C 61 73 74 45 72 72 6F 72 00 00 00 00 .....
0037551E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... SetLastError...
```

Ya vemos que la API decodificada es SetLastError.

De la misma manera que en el caso de las entradas buenas, lo que esperamos que haga ahora el packer es obtener su dirección para después escribirla no en la IAT, sino en las posiciones que él se haya generado o se vaya a generar para ese fin.

Sin embargo, a diferencia del caso de entradas buenas, ahora no nos interesa trazar ya que sabemos (intuimos, pero fuertemente), que la historia terminará escribiendo la dirección de la API por cualquier sitio y no en la IAT como nos interesa.

Podemos trazar para quedarnos más tranquilos, pero realmente, ¡qué remedio le queda al packer que obtener las direcciones de las APIs!

	<p style="text-align: center;"><u>Newbie Vs UnknowPacker – Revisitado – by</u> <u>El Cid</u></p>
--	--

Por otra parte, como nuestra estrategia va a ser la segunda de las expuestas [aquí](#) (pág 8), casi nos da lo mismo, en principio, lo que ocurra en esta parte de entradas malas, dado que nosotros derivaremos todo el flujo del packer por la parte de las entradas buenas.

Por tanto ha llegado el momento de plantear lo que debemos hacer, que por otro lado, es bien simple.

10. Estrategia para la reparación de las entradas malas

Resumimos, de todo lo visto hasta aquí, lo que resulta relevante para la reparación de la IAT:

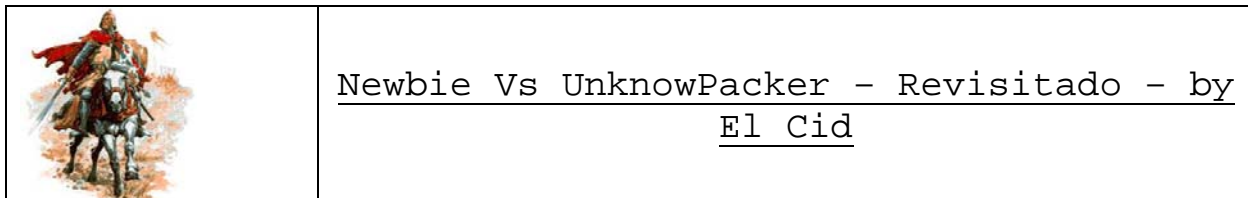
- Si la zona de la IAT correspondiente a una DLL es BUENA, la Dword a continuación del nombre de la DLL en la zona de memoria con las APIs encriptadas correspondiente a esta DLL, es, simplemente, su longitud en DWords.
- Si la zona de la IAT correspondiente a una DLL es MALA, la Dword a continuación del nombre de la DLL en la zona de memoria con las APIs encriptadas correspondiente a esta DLL, es su longitud en DWords con un "80"h añadido en su msb.
- El desencriptado de las APIs correspondientes a zonas buenas y malas es diferente.
- Los nombres de las DLLs y de las APIs se encuentran en una sección privada creada por el packer en ejecución.

Por tanto una estrategia para que el programa genere, por sí solo, las entradas correctas para las zonas malas de la IAT sería la siguiente:

Modificar el dato de las longitudes de DLLs de las 3 zonas malas de la IAT cambiando, en las DWord correspondientes, los respectivos bytes 80h por ceros binarios (00). Éso nos asegura, creemos, que el packer las interpretará como buenas y las tratará como tales, dirigiéndose a la parte del programa que escribe direcciones reales de las APIs en la IAT.

Sin embargo ésto requerirá que hagamos un ajuste adicional, derivado de que la descodificación de los nombres de las APIs ya hemos dicho que es distinta para zonas buenas y malas.

Por tanto para paliar este inconveniente deberemos, además, injertar el programa para los casos de entradas malas, ya que si no, al desencriptarse incorrectamente el nombre de las APIs, la función GetProcAddress retornará un valor nulo y se generará un error a continuación.



Vamos primeramente a verificar que simplemente por el cambio de la DWord que expresa la longitud de la zona IAT de una zona mala, el flujo del programa se dirige a la zona que trata las entradas buenas.

Reiniciamos y cambiamos el byte de IsDebuggerPresent a 00

Debemos tener en cuenta que muchas de las direcciones con las que tenemos que trabajar no existen al iniciar, porque las crea el packer después. Por tanto, a veces, tendremos que hacer las cosas dando rodeos o haciendo paradas intermedias porque no las podemos hacer directamente al inicio.

Activamos el MBP de 4DD128, para que pare allí que como sabemos es la primera zona de la IAT que escribe el packer y es MALA. Además, sabemos que antes de escribir las entradas redireccionadas, escribe primero allí unos ceros lo que nos viene muy bien para parar antes de que empiece a escribir entradas malas.

Damos Run y paramos aquí:

```

00373668  JMP 566242F8
0037366D  PUSH ECX
0037366E  SHR ECX, 2
00373671  REP MOVS DWORD PTR ES:[EDI], DWORD PTR DS:[ESI]
00373673  POP ECX
00373674  AND ECX, 00000003
00373677  REP MOVS BYTE PTR ES:[EDI], BYTE PTR DS:[ESI]
00373679  POP ESI
0037367A  PUSH EBX

```

ECX=00050BB0
[00C0C140]=0
[0040D140]=0

No es relevante ningún dato, lo único es que ahora sabemos que ya están generadas las APIs encriptadas.

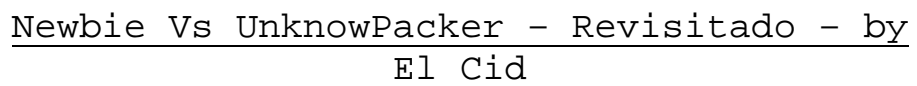
Sabemos que la primera zona de la IAT en escribirse, es mala y corresponde a KERNEL32.DLL. Además ya hemos visto que las APIs encriptadas de esta DLL están después de la pos 375824h, pero esta dirección de memoria que no existía al comienzo, ahora ya existe, como vemos aquí en la ventana M de Olly:

Memory map				
Address	Size	Owner	Section	Contains
002C0000	00041000			
00310000	00006000			
00320000	00041000			
00370000	00007000			
00380000	00001000			
00390000	00001000			
003A0000	00001000			
003B0000	00005000			

Ahora desactivamos el MBP de 4DD128, porque ya no nos hace falta.

Vamos a la pos 375824h, buscamos el nombre de la misma (KERNEL32.DLL) y miramos la DWord a continuación del nombre, que sabemos que es la longitud de la misma (en DWords) y parcheamos el byte 80h por 00, como se ve en la fig:

Address	Hex dump	ASCII
00375824	4B 45 52 4E 45 4C 33 32 2E 44 4C 4C 00 73 00 00	KERNEL32.DLL.s..
00375834	00 0C 93 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00 0C	..õþÛQfj'ðððð..
00375844	15 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00 0C B2 1E	ÛþÛQfj'ðððð..Ûþ
00375854	51 B1 B2 5E 71 7F 51 7F 5C 55 00 13 15 D1 BF 93	ÛþÛQfj'ððððU.!!Ûþõ
00375864	5C 9F BF D1 DE B5 5E 7F D1 91 BF 1E 7F 5C 55 00	ÛþÛQfj'ððððU.



Lo ponemos (con marcar 1 posición hubiera bastado):

Así podremos comprobar si solo por el hecho de parchear 80h a 00 el packer se dirige a la zona buena en vez de a la mala.

50



Newbie Vs UnknowPacker - Revisitado - by El Cid

, y vemos que el supuesto nombre de la API es un galimatías sin sentido que provocará el fallo de GetProcAddress (en lugar de devolver la dirección de la API, devolverá un valor nulo) y el subsiguiente error del programa.

Por tanto, cuando estemos tratando una zona mala, tendremos que injertar la zona de descriptado buena, para que el nombre de la API se descripte como lo haría si hubiese ido a la zona del programa que trata las entradas malas. Esto parece un trabalenguas pero si se piensa un momento es lo más lógico y simple.

11. Injerto del programa

Ya hemos visto las instrucciones de descriptado de nombres, para las zonas buenas y malas de la IAT. Recordamos que son éstas:

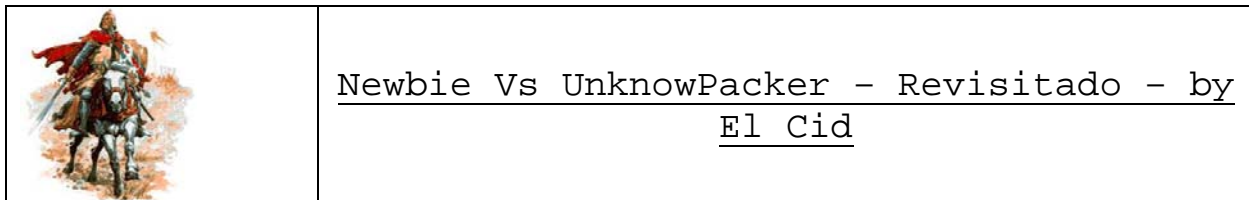
Zona BUENA		Zona MALA	
003738C8 LEA EDI, [EBP+L2Walker.403BC7] 003738D1 XOR EAX, EAX 003738D3 LODS BYTE PTR DS:[ESI] 003738D4 JMP SHORT 003738DD 003738D6 ROL AL, 8 003738D9 NOT AL 003738DB STOS BYTE PTR ES:[EDI] 003738DC LODS BYTE PTR DS:[ESI] 003738DD OR EAX, EAX 003738DF JNE SHORT 003738D6 003738E1 STOS BYTE PTR ES:[EDI] 003738E2 POPAD 003738E3 LEA EDI, [EBP+L2Walker.403BC7]	Instrucciones de descriptado para Nombres de APIs de zonas BUENAS	003739ED MOVZX ECX, BYTE PTR DS:[ESI-1] 003739F1 JMP SHORT 00373A01 003739F3 LODS BYTE PTR DS:[ESI] 003739F4 XOR AL, 79 003739F6 SUB AL, 55 003739F8 ROL AL, 8 003739FB NOT AL 003739FD STOS BYTE PTR ES:[EDI] 003739FE DEC ECX 003739FF XOR EAX, EAX 00373A01 OR ECX, ECX 00373A03 JNE SHORT 003739F3 00373A05 STOS BYTE PTR ES:[EDI] 00373A06 POPAD	Instrucciones de descriptado para Nombres de APIs de zonas MALAS

, como vemos el descriptado de nombre de APIs de zonas malas tiene 2 instrucciones más que el de de zonas buenas, que son 4 bytes más como vemos aquí:

003739EB 33C0 XOR EAX, EAX 003739ED 0FB64E FF MOVZX ECX, BYTE PTR DS:[ESI-1] 003739F1 EB 0E JMP SHORT 00373A01 003739F3 AC LODS BYTE PTR DS:[ESI] 003739F4 34 79 XOR AL, 79 003739F6 2C 55 SUB AL, 55 003739F8 C0C0 03 ROL AL, 8 003739FB F6D0 NOT AL 003739FD AA STOS BYTE PTR ES:[EDI] 003739FE 49 DEC ECX 003739FF 33C0 XOR EAX, EAX	
---	--

Esto supone un leve contratiempo, porque si hubiera sido al revés es decir, si el descriptado de zonas buenas hubiese tenido más instrucciones (o bytes) que el de zonas malas, sólo con NOPear las instrucciones de más hubiéramos resuelto el asunto, pero así nos va a producir un pequeño inconveniente, ya que no podremos parchear directamente la zona buena para que quede como la mala, porque no caben los bytes. Por ello no nos queda otra solución que realizar un injerto, entendiendo por tal un salto desde donde estamos a otra zona del programa, donde hacemos lo que tengamos que hacer, para luego volver a este lugar.

Lo primero que tenemos que hacer es buscar un sitio libre para nuestro injerto. No son muchos bytes los que necesitamos, pero como el programa está todavía en parte, empackado, no hay muchos bytes libres, aparte de que sería peligroso usar zonas del programa que también use el packer porque nadie nos asegura que éste no nos



sobreescriba encima y nos deje sin injerto. Así que buscaremos hueco en la sección Code del programa principal:

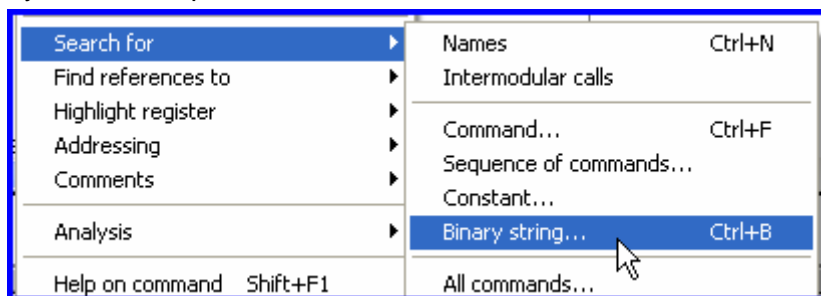
00310000	00006000		
00320000	00041000		
00400000	00001000	L2Walker	PE header
00401000	0021F000	L2Walker	<Sect_0> Code
00620000	00022000	L2Walker	<Sect_1> Data, resources
00642000	00008000	L2Walker	SFX, imports
7C800000	00001000	kernel32	PE header

Para buscar espacio dentro de un programa, acostumbro a hacerlo a mano. La razón es que cada vez que he querido usar el Topo, mi antivirus salta como un gato montés y se lo carga. Todavía no he podido determinar sin género de dudas, si verdaderamente el Topo está infectado o es un “falso positivo”. El caso es que lo he hecho a mano.

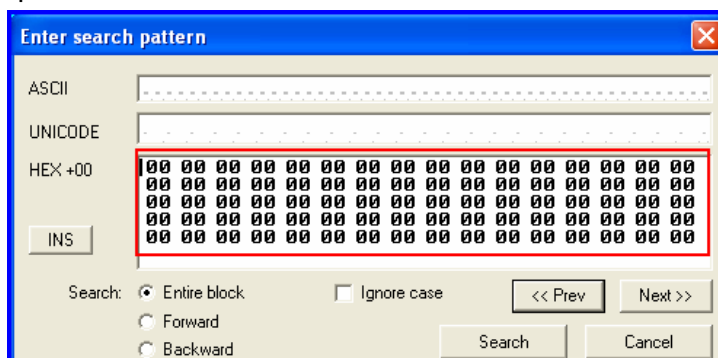
Vamos a la Section de código:

00401000	AB	STOS	DWORD PTR ES:[EDI]
00401001	68 5E8A5BC8	PUSH	C85B8A5E
00401006	26:7E FF	JLE	SHORT 00401008
00401009	40	INC	EAX
0040100A	31C3	XOR	EBX,EAX
0040100C	AF	SCAS	DWORD PTR ES:[EDI]

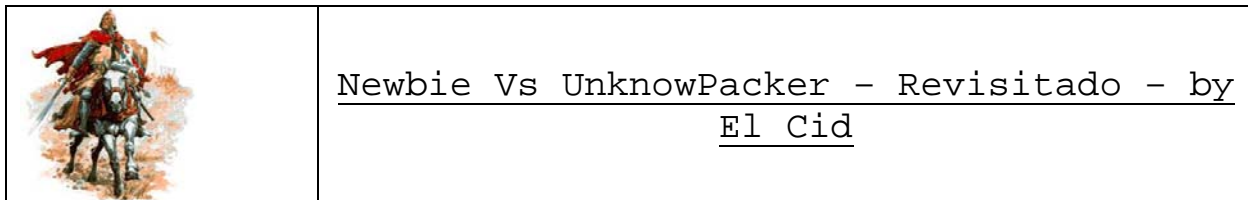
, y situados aquí hacemos:



, y ponemos un número abundante de ceros binarios, mayor cuanto más espacio queramos encontrar:



, y aquí tenemos nuestra zona libre:



004DC2AA	0000	ADD BYTE PTR DS:[EAX],AL
004DC2AC	0000	ADD BYTE PTR DS:[EAX],AL
004DC2AE	0000	ADD BYTE PTR DS:[EAX],AL
004DC2B0	0000	ADD BYTE PTR DS:[EAX],AL
004DC2B2	0000	ADD BYTE PTR DS:[EAX],AL
004DC2B4	0000	ADD BYTE PTR DS:[EAX],AL
004DC2B6	0000	ADD BYTE PTR DS:[EAX],AL
004DC2B8	0000	ADD BYTE PTR DS:[EAX],AL
004DC2BA	00	DB 00
004DC2BB	00	DB 00
004DC2BC	00	DB 00
004DC2BD	00	DB 00
004DC2BE	00	DB 00
004DC2BF	00	DB 00
004DC2C0	00	DB 00
004DC2C1	00	DB 00
004DC2C2	00	DB 00
004DC2C3	00	DB 00
004DC2C4	00	DB 00
004DC2C5	00	DB 00

Ahora, yo suelo buscar un byte de inicio que sea fácil de recordar, por ejemplo que termine en 1 ó 2 ceros si existe. En nuestro caso he elegido el 4DC2D0, pero vosotros podéis elegir el que más os guste.

Vamos ahora a repetir básicamente lo hecho en el apartado anterior, justo hasta el momento en que vaya a desenscriptar las APIs, que entonces, pondremos nuestro injerto.

Reiniciamos y cambiamos el byte de IsDebuggerPresent a 00

Activamos el MBP de 4DD128, ya hemos explicado antes para qué.

Damos Run y paramos aquí, como antes:

00373668	JMP 566242F8
0037366D	PUSH ECX
0037366E	SHR ECX,2
00373671	REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00373673	POP ECX
00373674	AND ECX,00000003
00373677	REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00373679	POP ESI
0037367A	PUSH EBX

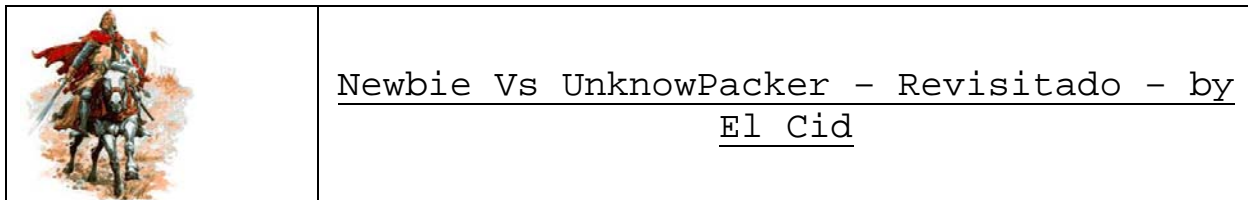
ECX=00050BB0
[00C0C140]=0
[004DD140]=0

Ahora desactivamos el MBP de 4DD128, porque ya no nos hace falta.

Vamos a la pos 375824h, buscamos el nombre de la DLL que sabemos es KERNEL32 y miramos la DWord a continuación del nombre, que sabemos que es la longitud de la misma (en DWords) y parcheamos el byte 80h por 00, como se ve en la fig:

Address	Hex dump	ASCII
00375824	4B 45 52 4E 45 4C 33 32 2E 44 4C 4C 00 73 00 00	KERNEL32.DLL. s..
00375834	00 0C 93 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00 0C	..δθηΒΩη'δδδδ..
00375844	15 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00 0C B2 1E	ΣθηΒΩη'δδδδ..Σ▲
00375854	51 B1 B2 5E 71 7F 51 7F 5C 55 00 13 15 D1 BF 93	0Ση^qδδδ\U.!!Σθηδ
00375864	5C 9F BF D1 DE B5 5E 7F D1 91 BF 1E 7F 5C 55 00	\ηθiδ^δθη▲δ\U.

Ponemos un MBP en el primer carácter de la primera API de esta zona mala como vemos en la figura para que pare cuando vaya a leerlo:



Address	Hex dump	ASCII
00375804	00 FF B5 53 38 40 00 8D 8D D2 3E 40 00 8D BD 2E	. AS8@.i1E>@.ic.
00375814	16 40 00 2B CF 33 C0 F3 AA AB C3 28 D1 00 00 0C	..@.+@3%~%t{0...
00375824	4B 45 52 4F 45 4C 33 32 2F 44 4C 4C 00 73 00 00	KERNEL32.DLL.s...
00375834	00 0C 93 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00 0C	..@0~@Qf'@0@0..
00375844	15 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00 0C B2	..@0~@Qf'@0@0..
00375854	51 B1 B2 5E 71 7F 51 7F 5C 55 00 13 15 D1 BF	
00375864	5C 9F BF D1 DE B5 5E 7F D1 91 BF 1E 7F 5C 55	
00375874	14 15 D1 BF 13 5E FE B1 1E 1F 9F B5 5E 7F D1	

1ª API encriptada de KERNEL32

Damos Run y paramos aquí:

Assembly code snippet:

```

003738C6 INC EDX
003738C7 PUSH EDX
003738C8 PUSHAD
003738C9 MOV ESI,EDX
003738CB LEA EDI,[EBP+L2Walker.403BC7]
003738D1 XOR EAX,EAX
003738D3 LODS BYTE PTR DS:[ESI]
003738D4 JMP SHORT 003738DD
003738D6 ROL AL,3
003738D9 NOT AL
003738DB STOS BYTE PTR ES:[EDI]
003738DC LODS BYTE PTR DS:[ESI]
003738DD OR EAX,EAX
003738DF JNE SHORT 003738D6
003738E1 STOS BYTE PTR ES:[EDI]
003738E2 POPAD
003738E3 LEA EDX,[EBP+L2Walker.403BC7]
003738E5 BRNED EAX

```

Register window values:

- EAX: 00000000
- ECX: 00000073
- EDX: 00375836
- EBX: 00403CD0
- ESP: 0012FF7C
- EBP: FFF71947
- ESI: 00375836
- EDI: 0037550E
- EIP: 003738D3

Hex dump snippet:

Address	Hex dump	ASCII
00375804	00 FF B5 53 38 40 00 8D 8D D2 3E 40 00 8D BD 2E	. AS8@.i1E>@.ic.
00375814	16 40 00 2B CF 33 C0 F3 AA AB C3 28 D1 00 00 0C	..@.+@3%~%t{0...
00375824	4B 45 52 4F 45 4C 33 32 2F 44 4C 4C 00 73 00 00	KERNEL32.DLL.s...
00375834	00 0C 93 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00 0C	..@0~@Qf'@0@0..
00375844	15 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00 0C B2	..@0~@Qf'@0@0..

, donde vemos que efectivamente estamos accediendo al primer carácter encriptado de la API.

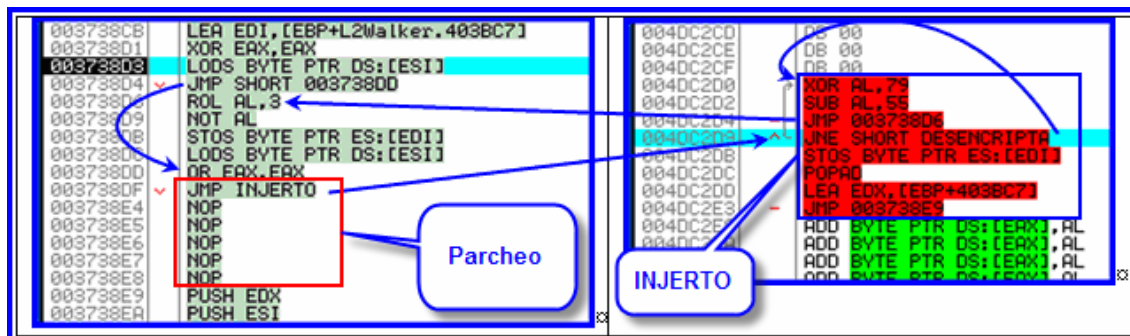
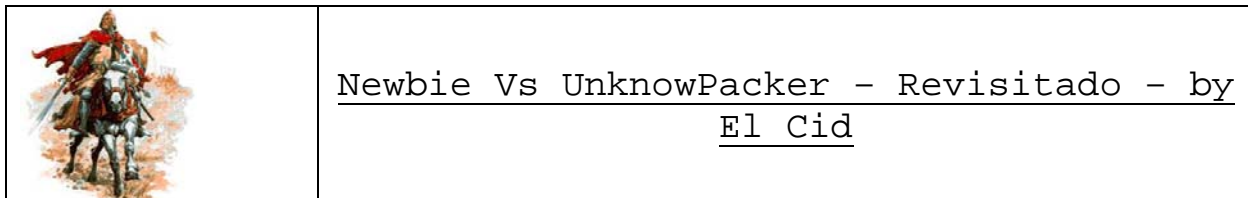
Pues ahora sería el momento de preparar nuestro injerto y la llamada al mismo.

NO HACE FALTA QUE LO HAGAIS PORQUE LO VAMOS A VER Y REPETIR DE NUEVO DESDE EL PRINCIPIO EN EL APARTADO SIGUIENTE, CON ENTENDERLO ES SUFICIENTE.

Iríamos al punto que hemos escogido como inicio, en mi caso el 4DC2D0 y comenzaríamos la codificación del injerto que en nuestro caso se reduce simplemente a copiar las 2 instrucciones que faltan del desencriptado de zonas malas. El resto de instrucciones que vemos son los saltos JMP, JNE y resto de instrucciones que hemos machacado en el código original ya que el salto largo ha ocupado 5 bytes en lugar de los 2 que era el salto corto.

Además deberíamos parchear la zona de desencriptado de APIs malas para que se desencripten adecuadamente.

Lo vemos hecho y después lo comentamos que creo que así se entenderá mejor. Además, como se puede ver, he puesto la etiqueta INJERTO en 4DC2D9 y DESENCRIPTA en 4DC2D0, para que se visualice mejor el flujo.



Vemos que el flujo del programa viene desde 3738D3 en donde lee un carácter de la API encriptada y salta a ver si era el último (nulo) mediante la OR EAX,EAX activando o no el Z Flag. Ahora saltamos a nuestro injerto donde miramos el flag a ver si era nulo y caso de no serlo vamos a desencriptar de la forma como se hace en las zonas malas, con: XOR AL, 79 y SUB AL,55 y después saltamos al resto de código que es común, almacenamos al carácter mediante STOS BYTE y leemos un carácter nuevo con LODS BYTE lo que recomienza el ciclo.

Bien, ha llegado el momento cumbre de realizar y probar nuestro injerto.

Vamos a recapitular cuidadosamente todos los pasos a dar, para que nadie luego venga con reclamaciones de que “a mí no me funciona el Salto Mágico” y esas cosas que ya sé yo lo que pasa, je, je.

11.1. Reparación de la primera zona Mala de la IAT

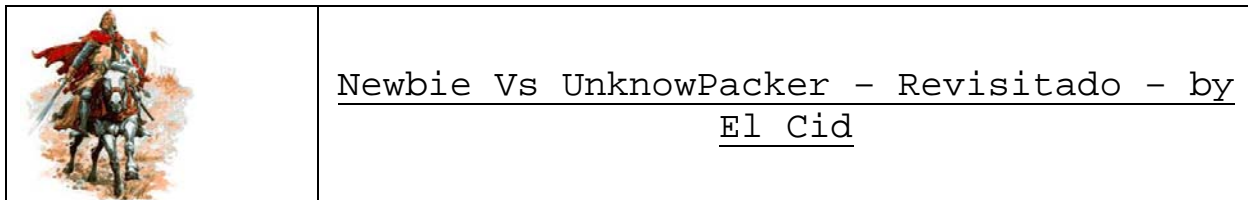
Haremos una lista de las cosas que hay que hacer y a quien las haga todas yo le aseguro que funciona. La lista de acciones es un poco larga pero es sólo para el relleno de la primera zona IAT, porque hay que tener en cuenta que todavía no está desempacado del todo el programa, pero el resto de zonas de la IAT son más fáciles.

Recomiendo seguir el mismo orden que se indica porque si no podría fallar el sistema (queda fuera de contexto explicarlo aquí, si alguien está interesado tiene que ver con la forma de direccionar de MASM).

La lista de acciones es la siguiente que no hace sino recopilar todo lo visto hasta aquí:

- Reiniciar
- Poner a cero binario el byte de IsDebuggerPresent (je, je, creo que no se me ha olvidado ni una vez avisarlo)
- Activar o poner un MBP en la pos 4DD128
- **Dar Run** y esperar a que pare. (en 373671, o en 393671 si definió ahí la sección privada el packer)
- Quitar el MPB de 4DD128
- Ir a la sección que creó el packer 37xxx (OJO, ya digo que a veces es la 39xxx, se distingue porque en la ventana M de Olly, se ve que tiene una longitud de 7000h bytes).

002C0000	00041000			
00310000	00006000			
00320000	00041000			
00370000	00007000			
00380000	00001000			
00390000	00001000			
003A0000	00001000			



- Buscar allí la cadena KERNEL32.DLL y poner un MBP en él

003757EC	CD 19 61 CF 8B 85 53 37 40 00 85 C0 74 07 61 B8	=a0iãS7@.ã't.ã0
003757FC	00 00 00 00 C3 6A 00 6A 00 FF B5 53 38 40 00 8D	...fj.j. AS8@.i
0037580C	8D 02 3E 40 00 8D BD 2E 16 40 00 2B CF 33 C0 F3	ie>@.ic..@.+83%ã
0037581C	AA AB C3 28 D1 00 00 0C 4B 45 52 4E 45 4C 33 32	~ãt(0...KERNEL32
0037582C	2E 44 4C 4C 00 73 00 00 0C 93 D1 BF B2 51 9F	.DLL.s...õhã0.f
0037583C	BF D5 7F 7F 1E 7F 00 0C 15 D1 BF B2 51 9F BF D5	ã'õããã..ãõãããã.fã
0037584C	7F 7F 1E 7F 00 0C B2 1E 51 B1 B2 5E 71 7F 51 7F	ãããã..ãããããããããããã
0037585C	5C 55 00 13 15 D1 BF 93 5C 9F BF D1 DE B5 5E 7F	\U.!!ãõããããããããããã
0037586C	D1 91 BF 1E 7F 5C 55 00 14 15 D1 BF 13 5E FE B1	0ãããããããããããããããã

- En la misma zona, cambiar el byte 80h por 00 como se ve arriba en la figura.
- Dar Run (para aquí):

7C911273	83C9 FF	OR ECX,FFFFFFF
7C911276	83C0	XOR EAX,EAX
7C911278	F2:BE	REPNE SCAS BYTE PTR ES:[EDI]
7C91127A	F7D1	NOT ECX
7C91127C	81F9 FFFF000	CMP ECX,0FFFF
7C911282	76 05	JBE SHORT 7C911289

- Quitar el MBP del nombre del KERNEL32.DLL
- Hacer (Execute Until Return + F7) 3 veces hasta llegar aquí:

00373855	0BC0	OR EAX,EAX
00373857	75 1E	JNE SHORT 00373877
00373859	56	PUSH ESI
0037385A	8D85 271F4000	LEA EAX,[EBP+L2Walker.401F27]
00373860	50	PUSH EAX
00373861	8B85 2F374000	MOV EAX,DWORD PTR SS:[EBP+L2Walker.40372F]
00373867	E9 930B0000	JMP 003743FF
0037386C	FF15 0BC07505	CALL DWORD PTR DS:[575C00B]
00373872	E9 900D0000	JMP 00374614
00373877	0FB64E FF	MOVZX ECX,BYTE PTR DS:[ESI-1]

- Dar F7 y Poner un BP en 373857. OJO ésto último es muy, muy, importante si no dará error (luego se entenderá por qué).
- Escribir el Injerto en la pos 4DC2D0 o en la que cada uno haya elegido. (El injerto no debe escribirse antes porque el packer puede detectarlo o machacarlo).

004DC2CE	00	DB 00
004DC2CF	00	DB 00
004DC2D0	34 79	XOR AL,79
004DC2D2	2C 55	SUB AL,55
004DC2D4	E9 FD75E9FF	JMP 003738D6
004DC2D9	75 F5	JNE SHORT 004DC2D0
004DC2DB	AA	STOS BYTE PTR ES:[EDI]
004DC2DC	61	POPAD
004DC2DD	8D95 C73B400	LEA EDX,[EBP+403BC7]
004DC2E3	E9 0176E9FF	JMP 003738E9
004DC2E8	00	DB 00
004DC2E9	00	DB 00

- Parchear el código para llamar al injerto, dado que la primera zona IAT es mala. (OJO el Label INJERTO de Olly hace referencia a la pos 4DC2D9 del Injerto que está señalada en la fig anterior)

003738DC	AC	LODS BYTE PTR DS:[ESI]
003738DD	0BC0	OR EAX,EAX
003738DF	E9 F5091600	JMP INJERTO
003738E4	90	NOP
003738E5	90	NOP
003738E6	90	NOP
003738E7	90	NOP
003738E8	90	NOP
003738E9	52	PUSH EDX
003738EA	56	PUSH ESI

- Antes de dar Run podemos ir a la zona de la IAT que se va a rellenar y la veremos cambiar delante de nuestros ojos. Dar Run



Newbie Vs UnknowPacker - Revisitado - by El Cid

ANTES	DESPUÉS																																								
<table><tr><th>Address</th><th>Hex dump</th></tr><tr><td>004DD128</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td></tr><tr><td>004DD138</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td></tr><tr><td>004DD148</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td></tr><tr><td>004DD158</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td></tr><tr><td>004DD168</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td></tr><tr><td>004DD178</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td></tr><tr><td>004DD188</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td></tr></table> <div>SIGUE ...</div>	Address	Hex dump	004DD128	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	004DD138	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	004DD148	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	004DD158	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	004DD168	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	004DD178	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	004DD188	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	<table><tr><th>Address</th><th>Hex dump</th></tr><tr><td>004DD128</td><td>30 FE 91 7C 21 FE 91 7C 77 10 80 7C FA 4E 81 7C</td></tr><tr><td>004DD138</td><td>73 13 82 7C DF B4 80 7C B1 B6 80 7C 87 00 80 7C</td></tr><tr><td>004DD148</td><td>EE AB 80 7C 3F FC 80 7C E7 ED 80 7C 20 25 80 7C</td></tr><tr><td>004DD158</td><td>13 CE 81 7C 47 06 81 7C 3D FD 80 7C 29 FF 80 7C</td></tr><tr><td>004DD168</td><td>92 FE 80 7C EA 11 81 7C E0 10 91 7C 00 10 91 7C</td></tr><tr><td>004DD178</td><td>42 24 80 7C C5 9F 80 7C A7 CC 80 7C 79 BC 80 7C</td></tr><tr><td>004DD188</td><td>7A 13 92 7C 01 9F 80 7C 19 2A 81 7C 25 A4 80 7C</td></tr><tr><td>004DD198</td><td>72 02 80 7C 25 99 80 7C 9E 97 80 7C 00 34 83 7C</td></tr><tr><td>004DD1A8</td><td>13 DC 81 7C 54 8A 83 7C 9B C4 81 7C 29 7D 83 7C</td></tr><tr><td>004DD1B8</td><td>20 9F 80 7C DF BC 80 7C 11 9E 80 7C B7 27 81 7C</td></tr><tr><td>004DD1C8</td><td>30 99 80 7C B8 92 80 7C 37 A4 80 7C ED 47 84 7C</td></tr></table> <div>SIGUE ...</div>	Address	Hex dump	004DD128	30 FE 91 7C 21 FE 91 7C 77 10 80 7C FA 4E 81 7C	004DD138	73 13 82 7C DF B4 80 7C B1 B6 80 7C 87 00 80 7C	004DD148	EE AB 80 7C 3F FC 80 7C E7 ED 80 7C 20 25 80 7C	004DD158	13 CE 81 7C 47 06 81 7C 3D FD 80 7C 29 FF 80 7C	004DD168	92 FE 80 7C EA 11 81 7C E0 10 91 7C 00 10 91 7C	004DD178	42 24 80 7C C5 9F 80 7C A7 CC 80 7C 79 BC 80 7C	004DD188	7A 13 92 7C 01 9F 80 7C 19 2A 81 7C 25 A4 80 7C	004DD198	72 02 80 7C 25 99 80 7C 9E 97 80 7C 00 34 83 7C	004DD1A8	13 DC 81 7C 54 8A 83 7C 9B C4 81 7C 29 7D 83 7C	004DD1B8	20 9F 80 7C DF BC 80 7C 11 9E 80 7C B7 27 81 7C	004DD1C8	30 99 80 7C B8 92 80 7C 37 A4 80 7C ED 47 84 7C
Address	Hex dump																																								
004DD128	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00																																								
004DD138	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00																																								
004DD148	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00																																								
004DD158	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00																																								
004DD168	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00																																								
004DD178	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00																																								
004DD188	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00																																								
Address	Hex dump																																								
004DD128	30 FE 91 7C 21 FE 91 7C 77 10 80 7C FA 4E 81 7C																																								
004DD138	73 13 82 7C DF B4 80 7C B1 B6 80 7C 87 00 80 7C																																								
004DD148	EE AB 80 7C 3F FC 80 7C E7 ED 80 7C 20 25 80 7C																																								
004DD158	13 CE 81 7C 47 06 81 7C 3D FD 80 7C 29 FF 80 7C																																								
004DD168	92 FE 80 7C EA 11 81 7C E0 10 91 7C 00 10 91 7C																																								
004DD178	42 24 80 7C C5 9F 80 7C A7 CC 80 7C 79 BC 80 7C																																								
004DD188	7A 13 92 7C 01 9F 80 7C 19 2A 81 7C 25 A4 80 7C																																								
004DD198	72 02 80 7C 25 99 80 7C 9E 97 80 7C 00 34 83 7C																																								
004DD1A8	13 DC 81 7C 54 8A 83 7C 9B C4 81 7C 29 7D 83 7C																																								
004DD1B8	20 9F 80 7C DF BC 80 7C 11 9E 80 7C B7 27 81 7C																																								
004DD1C8	30 99 80 7C B8 92 80 7C 37 A4 80 7C ED 47 84 7C																																								

Aquí tenemos esta zona de la IAT que antes tenía las entradas redireccionadas, ya con todas las entradas arregladas cada una de ellas apuntando a la API correspondiente.

¡¡Por fin!! Después de tanto trabajo, lo hemos conseguido.

No me resisto a poner la zona entera de la IAT, es ésta:

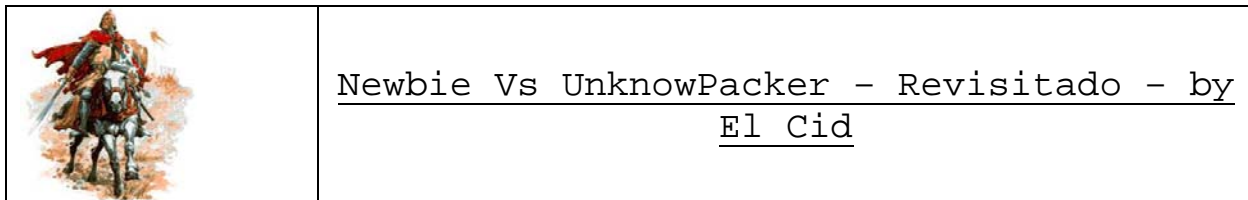
004DD0F8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD108	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD118	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD128	30 FE 91 7C 21 FE 91 7C 77 10 80 7C FA 4E 81 7C
004DD138	73 13 82 7C DF B4 80 7C B1 B6 80 7C 87 00 80 7C
004DD148	EE AB 80 7C 3F FC 80 7C E7 ED 80 7C 20 25 80 7C
004DD158	13 CE 81 7C 47 06 81 7C 3D FD 80 7C 29 FF 80 7C
004DD168	92 FE 80 7C EA 11 81 7C E0 10 91 7C 00 10 91 7C
004DD178	42 24 80 7C C5 9F 80 7C A7 CC 80 7C 79 BC 80 7C
004DD188	7A 13 92 7C 01 9F 80 7C 19 2A 81 7C 25 A4 80 7C
004DD198	72 02 80 7C 25 99 80 7C 9E 97 80 7C 00 34 83 7C
004DD1A8	13 DC 81 7C 54 8A 83 7C 9B C4 81 7C 29 7D 83 7C
004DD1B8	20 9F 80 7C DF BC 80 7C 11 9E 80 7C B7 27 81 7C
004DD1C8	30 99 80 7C B8 92 80 7C 37 A4 80 7C ED 47 84 7C
004DD1D8	30 8E 83 7C 07 51 83 7C 36 62 83 7C 75 63 83 7C
004DD1E8	89 9E 80 7C F4 9A 80 7C C6 2B 81 7C 08 0F 81 7C
004DD1F8	EE 1E 80 7C 61 0E 81 7C A7 CC 80 7C 93 6F 81 7C
004DD208	2D 2F 81 7C 6B CF 81 7C 87 DF 81 7C 4A 30 86 7C
004DD218	49 2F 81 7C D0 04 92 7C 16 1E 80 7C 68 C0 80 7C
004DD228	EA CD 81 7C E9 74 92 7C E1 B9 80 7C 66 2D 81 7C
004DD238	61 9A 80 7C D0 1A 80 7C C4 00 92 7C E5 17 80 7C
004DD248	2D FF 91 7C FA A7 93 7C 1F AC 80 7C 65 1C 83 7C
004DD258	76 E8 80 7C FC E7 80 7C BA 67 83 7C E7 36 81 7C
004DD268	43 09 83 7C EE 2A 81 7C C6 BD 80 7C D8 0B 83 7C
004DD278	40 97 80 7C D2 60 82 7C D6 97 80 7C 3F 99 80 7C
004DD288	57 98 80 7C 18 C1 80 7C 3F 29 83 7C 27 A0 80 7C
004DD298	7A 97 83 7C 51 F8 85 7C 17 84 83 7C 94 0D 83 7C
004DD2A8	FB 98 80 7C 8A 97 80 7C 76 97 80 7C 0E 18 80 7C
004DD2B8	97 00 81 7C 9E 0B 81 7C 51 26 81 7C A9 23 83 7C
004DD2C8	04 23 83 7C 8E 20 83 7C 87 0A 81 7C 0E DE 80 7C
004DD2D8	05 DE 80 7C 9D 99 80 7C C9 23 81 7C 01 4D 83 7C
004DD2E8	50 97 80 7C AF 2D 81 7C D5 9B 80 7C 00 00 00 00
004DD2F8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD308	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Por supuesto que quedan más zonas, ya que ésta era sólo la primera pero era la más difícil, precisamente por ello, ya que teníamos que ajustar nuestro sistema. Ahora todo será coser y cantar.

Antes de continuar, ¿habeis visto dónde se ha detenido Olly? Sí, en el BP 373857 que os he dicho que era tan importante.

Como hemos visto antes, esta posición podemos entenderla como el inicio del tratamiento de una DLL y efectivamente ahora comprobamos que es así. Se acaba de completar una zona de la IAT y si hemos parado aquí, es porque el programa va ahora a tratar la siguiente DLL.

La siguiente DLL, tenemos que mirar si es de entradas buenas o malas en la IAT, pero ya os adelanto que es buena y por lo tanto habrá que descryptar los nombres de las APIs como tales.



Pero en este momento la decodificación de nombres de APIs está dispuesta (con nuestro injerto) para las entradas malas de la DLL anterior, luego si no hacemos nada en cuanto vaya a descriptar el nombre de la primera API “BUENA” y obtenga una cadena “en japonés” o similar a lo visto antes, nos dará un error como una casa. Ésta es la razón por la que he dicho antes que este BP era muy importante.

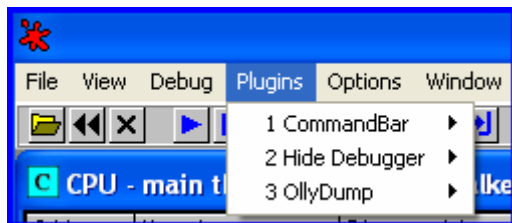
11.2. Construcción de la IAT completa

Una vez visto cómo se repara una zona mala de la IAT por medio de nuestro injerto, lo que haremos será construir el resto de la IAT, las zonas buenas sin modificarlas y las malas derivando el flujo a nuestro injerto como hemos visto. Finalmente cuando tengamos la IAT completamente reparada y escrita correctamente llegaremos hasta el OEP y dumpearemos el proceso.

Pero dado que con la v2.0 no podemos usar el OllyDump, nos conviene en este momento volver a nuestro buen amigo Olly v1.10, ahora que ya sabemos perfectamente lo que tenemos que hacer en cada punto del programa, así nuestro trabajo será más fácil.

Pues vamos allá. Cerramos Olly v2.0 y abrimos el v1.10

Debemos tener el plugin OllyDump, y otro para ocultarlo como Hide Debugger. Éstos son los que yo he puesto:



Empezamos y veréis que seguimos prácticamente los pasos vistos antes, con unas pequeñas variaciones que os indicaré:

- Cargamos el proggy en Olly v1.10
- Ponemos un MBP on Access en toda la zona 4DD128-4DD2F4 (ver [Tabla](#))
- **Damos Run** y paramos aquí como antes:

Address	Hex	Disassembly	Comment
00393671	REP	MOVS DWORD PTR ES:[EDI], DWORD PTR DS:[ESI]	
00393673	POP	EAX	
00393674	AND	EAX, 3	
00393677	REP	MOVS BYTE PTR ES:[EDI], BYTE PTR DS:[ESI]	
00393679	POP	ESI	
0039367A	PUSH	EBX	
0039367B	PUSH	8000	
00393680	PUSH	0	

Sin embargo ahora vemos que está trabajando en la sección 390000, en lugar de en la 370000. Por lo tanto deberemos transformar todas las direcciones 37xxxx que veíamos antes en 39xxxx de ahora para que todo sea igual. Vosotros debeis adaptarlo a las direcciones que tengais en vuestro equipo.

- Quitamos el MBP



Newbie Vs UnknowPacker - Revisitado - by El Cid

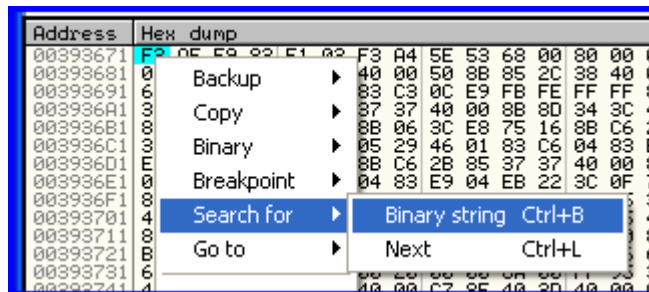
- Hacemos en este punto Follow in Dump



, y vamos a:

Address	Hex dump
00393671	F3 A5 59 83 E1 03 F3 A4 5E 53 68 00
00393681	00 56 8D 85 4F 1D 40 00 50 88 85 2C
00393691	6A 0D 00 00 E9 5B 83 C3 0C E9 FB FE
003936A1	30 3C 40 00 03 B5 37 37 40 00 8B 8D
003936B1	83 E9 05 EB 5B 66 8B 06 3C E8 75 16
003936C1	37 37 40 00 83 C0 05 29 46 01 83 C6

- Ahora buscamos el string KERNEL32.DLL mediante:



, y:

Enter binary string to search for

ASCII:

UNICODE:

HEX +03:

☒ Entire block

☐ Case sensitive

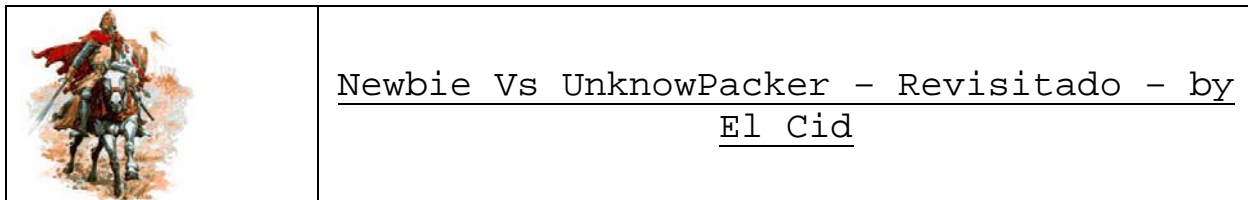
<< >> OK

Aquí está:

Address	Hex dump	ASCII
00395824	4B 45 52 4E 45 4C 33 32 2E 44 4C 4C 00 73 00 00	KERNEL32.DLL.s..
00395834	80 0C 93 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00 0C	..60h0f1'0000..
00395844	15 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00 0C B2 1E	30h0f1'0000..
00395854	51 B1 B2 5E 71 7F 51 7F 5C 55 00 13 15 D1 BF 93	00000000U.!!30h0
00395864	5C 9F BF D1 DE B5 5E 7F D1 91 BF 1E 7F 5C 55 00	\f10i0000U.!!30h0
00395874	14 15 D1 BF 13 5E FE B1 1E 1F 9F B5 5E 7F D1 91	130h11^0000U.!!30h0
00395884	BF 1E 7F 5C 55 00 12 15 D1 BF D2 1E B1 DF BE D1	7000U.30hE0000

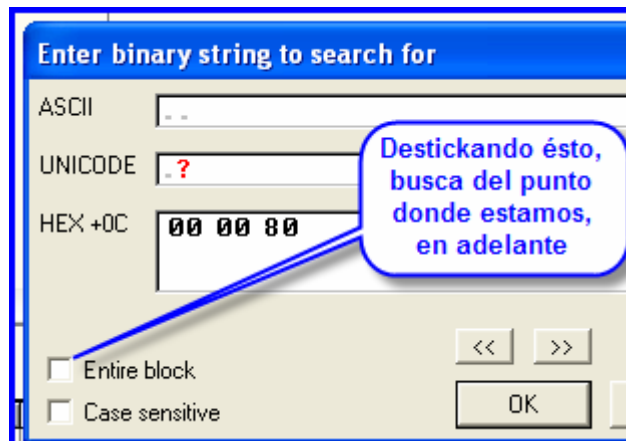
- Cambiamos el byte 80h por 00:

Address	Hex dump	ASCII
00395824	4B 45 52 4E 45 4C 33 32 2E 44 4C 4C 00 73 00 00	KERNEL32.DLL.s..
00395834	0C 93 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00 0C	..60h0f1'0000..
00395844	15 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00 0C B2 1E	30h0f1'0000..
00395854	51 B1 B2 5E 71 7F 51 7F 5C 55 00 13 15 D1 BF 93	00000000U.!!30h0
00395864	5C 9F BF D1 DE B5 5E 7F D1 91 BF 1E 7F 5C 55 00	\f10i0000U.!!30h0
00395874	14 15 D1 BF 13 5E FE B1 1E 1F 9F B5 5E 7F D1 91	130h11^0000U.!!30h0
00395884	BF 1E 7F 5C 55 00 12 15 D1 BF D2 1E B1 DF BE D1	7000U.30hE0000



- En este momento variamos un poco lo de antes por mera comodidad. Como sabemos que hay otras 2 zonas IAT MALAS y tendremos que volver a esta zona de la memoria para cambiar sus 2 bytes 80h por 00, pues los cambiamos ya y así no tenemos que volver y así nos desentendemos de ello, porque si luego se nos olvida pues se nos chafa todo y tenemos que volver a empezar (que a mí ya me pasó).

Por tanto buscamos la cadena 000080, a partir de este punto en adelante:



, hemos encontrado el siguiente:

Address	Hex dump	ASCII
00395F68	00 00 0B B5 53 13 97 72 12 91 53 96 53 D0 00 4C	..AS!ür#SUS.L
00395F78	D3 0D 00 0A 55 53 45 52 33 32 2E 44 4C 4C 00 88	É...USER32.DLL.É
00395F88	00 00 80 0E 52 FE FF 51 BE 5E B1 51 BF D1 73 D1	...R= Q%~Qh0s0
00395F98	91 BF 00 0A 15 D1 BF 93 DF 71 D2 D1 FE DF 00 09	...S0h0"qE0■..
00395FA8	15 D1 BF 13 5E FE B1 1E 1F 00 12 15 D1 BF 13 5E	S0h!!^■▲▼.S0h!!^

, aquí está y lo cambiamos:

Address	Hex dump	ASCII
00395F68	00 00 0B B5 53 13 97 72 12 91 53 96 53 D0 00 4C	..AS!ür#SUS.L
00395F78	D3 0D 00 0A 55 53 45 52 33 32 2E 44 4C 4C 00 88	É...USER32.DLL.É
00395F88	00 00 00 0E 52 FE FF 51 BE 5E B1 51 BF D1 73 D1	...R= Q%~Qh0s0
00395F98	91 BF 00 0A 15 D1 BF 93 DF 71 D2 D1 FE DF 00 09	...S0h0"qE0■..
00395FA8	15 D1 BF 13 5E FE B1 1E 1F 00 12 15 D1 BF 13 5E	S0h!!^■▲▼.S0h!!^
00395FB8	FE B1 1E 1F 33 BE 51 91 D1 DE D1 FE BF 00 08 52	■▲▼3%Q#0i0■.QR
00395FC8	0E 53 04 1F 55 5E FF 04 00 4F 00 00 00 00 00 00	0E53041F555EFF04004F0000000000000000000000

Seguimos ahora con CTRL+L:

Address	Hex dump	ASCII
00396CA1	04 00 00 00 00 00 09 00 00 00 00 00 0C 00 00 00	♦.....
00396CB1	00 00 08 00 00 00 00 00 06 00 00 00 00 00 07 00+.....Ç
00396CC1	00 00 00 00 0A 00 00 00 00 00 02 00 00 00 00 80@.....Ç
00396CD1	05 00 00 0A 57 53 32 5F 33 32 2E 44 4C 4C 00 0E	'...WS2_32.DLL.#
00396CE1	00 00 80 00 12 00 00 00 00 00 00 00 00 00 00 00	...Ç+...δ.....
00396CF1	09 00 00 00 00 00 03 00 00 00 00 00 13 00 00 00♥.....!!..
00396D01	00 00 10 00 00 00 00 0A 00 00 00 00 00 15 00 00>.....\$.
00396D11	00 00 00 00 04 00 00 00 00 00 73 00 00 00 00 00♦.....s.....
00396D21	17 00 00 00 00 00 6F 00 00 00 00 00 34 00 00 00	♦.....o.....4....

, y lo cambiamos:

Address	Hex dump	ASCII
00396CA1	04 00 00 00 00 00 09 00 00 00 00 00 0C 00 00 00	♦.....
00396CB1	00 00 08 00 00 00 00 00 06 00 00 00 00 00 07 00+.....Ç
00396CC1	00 00 00 00 0A 00 00 00 00 00 02 00 00 00 00 80@.....Ç
00396CD1	05 00 00 0A 57 53 32 5F 33 32 2E 44 4C 4C 00 0E	'...WS2_32.DLL.#
00396CE1	00 00 00 00 12 00 00 00 00 00 00 00 00 00 00 00+...δ.....
00396CF1	09 00 00 00 00 00 03 00 00 00 00 00 13 00 00 00♥.....!!..
00396D01	00 00 10 00 00 00 00 0A 00 00 00 00 00 15 00 00>.....\$.
00396D11	00 00 00 00 04 00 00 00 00 00 73 00 00 00 00 00♦.....s.....
00396D21	17 00 00 00 00 00 6F 00 00 00 00 00 34 00 00 00	♦.....o.....4....

, y ya sabemos que no hay más zonas malas que cambiar.



Newbie Vs UnknowPacker - Revisitado - by El Cid

- Si intentamos ahora, como antes, poner un MBP en el string de la DLL. KERNEL32, nos dará error, quizá lo detecta. Lo que haremos será poner un HBP en la DWORD correspondiente a su longitud, es decir en:

Address	Hex dump	ASCII
00395824	4B 45 52 4E 45 4C 33 32 2E 44 4C 4C 00 73 00 00	KERNEL32.DLL....
00395834	00 0C 93 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00 0C	...00000000...
00395844	15 D1 BF B2 51 9F BF D5 7F 7F 1E 7F 00 0C B2 1E	...00000000...
00395854	51 B1 B2 5E 71 7F 51 7F 5C 55 00 13 15 D1 BF 93	...00000000...
00395864	5C 9F BF D1 DE B5 5E 7F D1 91 BF 1E 7F 5C 55 00	...00000000...
00395874	14 15 D1 BF 13 5E 5E B1 1E 15 9F 5E 5E 7F D1 91	...00000000...

, donde obviamente vemos el byte 00 que acabamos de cambiar. Así pues tenemos:

#	Base	Size	Stop on	Follow	Delete
1	00395831	1	Access	Follow 1	Delete 1
2				Follow 2	Delete 2

- Damos Run y paramos aquí:

Address	Hex	Disassembly
0039387D	MOV	EDX, ESI
0039387F	MOV	ESI, EAX
00393881	INC	EDX
00393882	MOV	ECX, DWORD PTR DS:[EDX]
00393884	AND	ECX, 80000000
0039388A	OR	ECX, ECX
0039388C	JNZ	00393919
00393892	MOV	ECX, DWORD PTR DS:[EDX]
00393894	ADD	EDX, 4
00393897	PUSH	ECX

- Vemos que ya está comprobando si es zona buena o mala por el valor del byte 80h que nosotros hemos parcheado a 00. Vamos a 393857 y ponemos un BP:

Address	Hex	Disassembly
00393857	JNZ	SHORT 00393877
00393859	PUSH	ESI
0039385A	LEA	EAX, DWORD PTR SS:[EBP+401F27]
00393860	PUSH	EAX
00393861	MOV	EAX, DWORD PTR SS:[EBP+40372F]
00393867	JMP	003943FF
0039386C	CALL	NEAR DWORD PTR DS:[575C00B]
00393872	JMP	00394614
00393877	MOVZX	ECX, BYTE PTR DS:[ESI-1]
0039387B	ADD	ESI, ECX

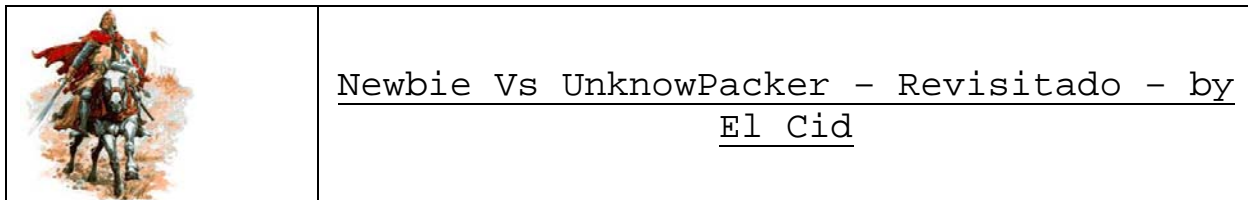
Olly nos avisa que esa posición está fuera de la sección de código pero aceptamos. Quitamos el HBP por si lo detecta.

- Preparamos el Injerto y la llamada al mismo:

003938D4	EB 07	JMP	SHORT 003938DD
003938D6	C0C0 03	ROL	AL, 3
003938D9	F6D0	NOT	AL
003938DB	AA	STOS	BYTE PTR ES:[EDI]
003938DC	AC	LDS	BYTE PTR DS:[ESI]
003938DD	0BC0	OR	EAX, EAX
003938DF	E9 F5891400	JMP	L2Walker.004DC2D9
003938E4	90	NOP	
003938E5	90	NOP	
003938E6	90	NOP	
003938E7	90	NOP	
003938E8	90	NOP	
003938E9	52	PUSH	EDX
003938EA	56	PUSH	ESI

004DC2C8	0000	ADD	BYTE PTR DS:[EAX], AL
004DC2CA	0000	ADD	BYTE PTR DS:[EAX], AL
004DC2CC	0000	ADD	BYTE PTR DS:[EAX], AL
004DC2CE	0000	ADD	BYTE PTR DS:[EAX], AL
004DC2D0	34 79	XOR	AL, 79
004DC2D2	2C 55	SUB	AL, 55
004DC2D4	E9 FD75EBFF	JMP	003938D6
004DC2D9	75 F5	JNZ	SHORT L2Walker.004DC2D0
004DC2DB	AA	STOS	BYTE PTR ES:[EDI]
004DC2DD	61	POPAD	
004DC2DD	8095 C73B4000	LEA	EDX, DWORD PTR SS:[EBP+403BC7]
004DC2E3	E9 0176EBFF	JMP	003938E9
004DC2E8	0000	ADD	BYTE PTR DS:[EAX], AL
004DC2EA	0000	ADD	BYTE PTR DS:[EAX], AL
004DC2EC	0000	ADD	BYTE PTR DS:[EAX], AL
004DC2EE	0000	ADD	BYTE PTR DS:[EAX], AL

Observad que los bytes del injerto (al trabajar en la sección 39xxxx, son ligeramente distintos, pero las instrucciones en ASM son las mismas)



- Antes de dar Run visualizaremos la zona IAT que vamos a reparar para estar seguros, al instante, de que se ha reparado:

Aquí vemos el inicio, todavía sin rellenar:

Address	Hex dump
00400128	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400138	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400148	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400158	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400168	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400178	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400188	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400198	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

- **Damos Run**, paramos nuevamente en el BP 393857 y la IAT se ha transformado así:

Address	Hex dump
00400128	30 FE 91 7C 21 FE 91 7C 77 1D 80 7C FA 4E 81 7C
00400138	73 13 82 7C DF B4 80 7C B1 B6 80 7C 87 00 80 7C
00400148	EE AB 80 7C 3F FC 80 7C E7 ED 80 7C 20 25 80 7C
00400158	13 CE 81 7C 47 06 81 7C 3D FD 80 7C 29 FF 80 7C
00400168	92 FE 80 7C EA 11 81 7C E0 10 91 7C 00 10 91 7C
00400178	42 24 80 7C C5 9F 80 7C A7 CC 80 7C 79 BC 80 7C
00400188	7A 13 92 7C 01 9F 80 7C 19 2A 81 7C 25 A4 80 7C
00400198	72 02 80 7C 25 99 80 7C 9E 97 80 7C C0 34 83 7C
004001A8	13 DC 81 7C 54 8A 83 7C 9B C4 81 7C 29 7D 83 7C
004001B8	20 9F 80 7C DF BC 80 7C 11 9E 80 7C 87 27 81 7C
004001C8	30 99 80 7C B8 92 80 7C 37 A4 80 7C ED 47 84 7C
004001D8	30 8E 83 7C 07 51 83 7C 36 62 83 7C 75 63 83 7C
004001E8	89 9E 80 7C F4 9A 80 7C C6 28 81 7C 08 0F 81 7C
004001F8	EE 1E 80 7C 61 0E 81 7C A7 CC 80 7C 93 6F 81 7C
00400208	2D 2F 81 7C 6B CF 81 7C 87 DF 81 7C 4A 30 86 7C
00400218	49 2F 81 7C DD 04 92 7C 16 1E 80 7C 68 C0 80 7C
00400228	EA CD 81 7C E9 74 92 7C E1 B9 80 7C 66 2D 81 7C
00400238	61 9A 80 7C D0 1A 80 7C C4 00 92 7C E5 17 80 7C
00400248	2D FF 91 7C FA A7 93 7C 1F AC 80 7C 65 1C 83 7C
00400258	76 E8 80 7C FC E7 80 7C BA 67 83 7C E7 36 81 7C
00400268	43 09 83 7C EE 2A 81 7C C6 B0 80 7C D8 08 83 7C
00400278	40 97 80 7C D2 60 82 7C D6 97 80 7C 3F 99 80 7C
00400288	57 9B 80 7C 18 C1 80 7C 3F 29 83 7C 27 A0 80 7C
00400298	7A 97 83 7C 51 F8 85 7C 17 84 83 7C 94 00 83 7C
004002A8	FB 98 80 7C 8A 97 80 7C 76 97 80 7C 0E 18 80 7C
004002B8	97 0D 81 7C 9E 08 81 7C 51 26 81 7C A9 23 83 7C
004002C8	04 23 83 7C 8E 20 83 7C 87 0A 81 7C 0E DE 80 7C
004002D8	05 DE 80 7C 9D 99 80 7C C9 23 81 7C 01 40 83 7C
004002E8	50 97 80 7C AF 2D 81 7C D5 9B 80 7C 00 00 00 00

Aquí la tenemos, calentita y recién rellenada con las entradas correctas que llaman directamente a las APIs correspondientes.

- Estamos parados en el BP 393857 y por lo tanto vamos a comenzar otra DLL que sabemos que es buena (y si dudamos miramos EDI=4DD000). Por lo tanto no necesitamos llamar al injerto. Así que quitamos la llamada que hemos puesto, la mejor manera es marcar las instrucciones y hacer Undo Selection (sólo en la llamada al injerto, porque el injerto en sí mismo lo dejamos para usarlo cuando llegue otra zona mala de la IAT).

003938D0	0BC0	OR	EAX, EAX	ADVAPI32
003938DF	E9 F5891400	JMP	L2Walker.004DC2D9	
003938E4	90	NOP		
003938E5	90	NOP		
003938E6	90	NOP		
003938E7	90	NOP		
003938E8	90	NOP		
003938E9	52	PUSH	EDX	
003938EA	56	PUSH	ESI	
003938EB	8085 B81F4000	LEA	EAX,	
003938F1	50	PUSH	EAX	
003938F2	000F 00000000	MOVB	EBX,	

, y nos vuelve a quedar como estaba, así:



Newbie Vs UnknowPacker - Revisitado - by El Cid

003938D4	EB 07	JMP	SHORT 003938D0
003938D6	C0C0 03	ROL	AL, 3
003938D9	F6D0	NOT	AL
003938DB	AA	STOS	BYTE PTR ES:[EDI]
003938DC	AC	LODS	BYTE PTR DS:[ESI]
003938DD	0BC0	OR	EAX, EAX
003938DF	75 F5	JNZ	SHORT 003938D6
003938E1	AA	STOS	BYTE PTR ES:[EDI]
003938E2	61	POPAD	
003938E3	8D95 C73B4000	LEA	EDX, DWORD PTR SS:[EBP+403BC7]

Ahora damos RUN, volvemos a parar en BP 393857 y se ha rellenado esta zona de la IAT:

Address	Hex dump
0040D000	07 6C DA 77 00 00 00 00 00 00 00 00 00 00 00 00
0040D010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

- Estamos parados en el BP 393857 y vamos a comenzar otra DLL que sabemos que es Mala (y si dudamos miramos EDI=4DD34C). Luego tenemos que volver a llamar al injerto, así que volvemos a parchear como la primera vez. Queda así, de nuevo:

003938DC	AC	LDS	BYTE PTR DS:[ESI]
003938DD	0BC0	OR	EAX, EAX
003938DF	E9 F5891400	JMP	L2Walker.004DC2D9
003938E4	90	NOP	
003938E5	90	NOP	
003938E6	90	NOP	
003938E7	90	NOP	
003938E8	90	NOP	
003938E9	52	PUSH	EDX
003938EA	56	PUSH	ESI
003938EB	58	PUSH	EDI

Damos RUN, paramos de nuevo en el BP 393857 y la IAT:

0040D2C8	04 23 83 7C 8E 20 83 7C 87 0A 81 7C 0E DE 80 7C	◆#A!A A!C.U!A!C!
0040D2D8	05 DE 80 7C 9D 99 80 7C C9 23 81 7C 01 4D 83 7C	!iC!00C!f#u!0M!A!
0040D2E8	50 97 80 7C AF 2D 81 7C 05 9B 80 7C 00 00 00 00	PuQ!>-u!>C!....
0040D2F8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D308	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D318	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D328	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D338	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040D348	00 00 00 00 FS B5 39 7E 8C 0C 3A 7E 7D BC 39 7EA9"l.:?A9"
0040D358	87 03 38 7E 27 BE 39 7E 62 07 3A 7E 3F B5 39 7E	C?:"*9"b.:?A9"
0040D368	1B C0 39 7E 69 D8 39 7E 56 90 39 7E A2 0D 3A 7E	+L9"i9"Ue9"0.: "
0040D378	29 D9 39 7E A9 C4 39 7E 72 02 3A 7E 86 13 3A 7E)!9"0-9"0: "B!:"
0040D388	68 EF 3C 7E 7A 14 3A 7E B3 F2 3A 7E C4 F6 3A 7E	h"<"2q:"l=":"-:"
0040D398	05 C5 39 7E EE 50 3E 7E 2F B8 39 7E 94 BF 39 7E	+!9" "P>"!9"0!9"
0040D3A8	08 C4 39 7E 1E F2 39 7E EA DA 39 7E 1C F2 3A 7E	0-9"0-9"0!9"l=":"
0040D3B8	CE 3D 3A 7E D8 D8 39 7E 07 D9 39 7E 85 3E 3A 7E	if="!i9"i9">:"
0040D3C8	3A 15 38 7E 48 BE 39 7E 0E 97 39 7E 60 DA 39 7E	:3:"K*9"0u9"i9"
0040D3D8	33 B9 39 7E F0 BE 39 7E AF C2 38 7E 5B F8 39 7E	3!9"0-9"i9"i9"0"
0040D3E8	DA 94 39 7E 45 5D 3A 7E B1 B8 3D 7E 5B BA 3D 7E	i09"El:"0!="["l="
0040D3F8	09 B9 3D 7E 75 5D 3A 7E EC DB 39 7E A2 BD 39 7E	!l="u!:"0!9"0c9"
0040D408	99 00 3D 7E 8F 3D 3A 7E 52 F8 3C 7E C9 59 3A 7E	0!="BA:"R<"FV:"
0040D418	5A 8A 3A 7E 58 D6 39 7E D5 EE 39 7E 88 C1 39 7E	Ze:"X!9"i-9"0!9"
0040D428	21 90 39 7E 09 B6 39 7E 1D B6 39 7E DC 8B 3A 7E	te9"iA9"iA9"i!:"
0040D438	7D FB 3A 7E F6 8B 39 7E D1 E1 3A 7E 86 5F 3D 7E	J!:"i9"0B:"0="
0040D448	96 F1 3C 7E 68 03 38 7E E2 C2 39 7E AB B8 3D 7E	0!<"h0:"0!9"0!9"
0040D458	95 B7 3D 7E C8 EF 39 7E FD BE 39 7E B2 C2 39 7E	0A="0!9"0!9"0!9"
0040D468	AB 8E 39 7E 38 1F 38 7E C6 B5 39 7E 55 FA 39 7E	0A9"0!9"0!9"0!9"
0040D478	3D EF 39 7E 26 BF 39 7E 1D C7 39 7E 8E BD 39 7E	=!9"0!9"0!9"0!9"
0040D488	75 E8 39 7E FE 95 38 7E 70 D8 39 7E D2 DC 39 7E	0!9"0!9"0!9"0!9"
0040D498	49 D7 39 7E DD 7D 3E 7E 55 FE 38 7E BC 22 3A 7E	i!9"i!>"0!9"0!9"
0040D4A8	3E DA 39 7E BC C8 3E 7E 3D EF 39 7E 2E 8C 39 7E	>!9"0!9"0!9"0!9"
0040D4B8	78 D8 3A 7E 52 F0 39 7E 0F F9 3A 7E 07 E9 38 7E	xi:"R-9"0!9"0!9"
0040D4C8	8F 8C 3A 7E 28 13 38 7E E8 D9 39 7E 1E C2 39 7E	Al:"(!:"0!9"0!9"
0040D4D8	53 5F 3D 7E B2 FF 39 7E 6C BF 39 7E 4D 3D 3A 7E	S="0!9"0!9"0!9"
0040D4E8	76 BD 39 7E 57 C2 39 7E EA F8 3A 7E C8 BD 39 7E	0c9"0!9"0!9"0!9"
0040D4F8	C7 86 39 7E 90 86 39 7E 6E C6 39 7E 41 BD 39 7E	0A9"0!9"0!9"0!9"
0040D508	CE D6 39 7E 58 BF 39 7E EA D6 39 7E 64 5D 3D 7E	0!9"0!9"0!9"0!9"
0040D518	29 8C 3A 7E 9C 8F 39 7E 7D 1A 38 7E 37 02 38 7E	Ji:"0A9"0!9"0!9"
0040D528	25 02 38 7E 5E 0F 38 7E 56 0D 38 7E D4 D9 39 7E	0!9"0!9"0!9"0!9"
0040D538	87 F7 39 7E 31 B6 39 7E F9 D7 39 7E A4 D8 39 7E	0-9"iA9"i9"i9"
0040D548	42 8C 39 7E 4E EB 39 7E AE B6 39 7E 2F B7 39 7E	B!9"Nu9"0!9"0!9"
0040D558	65 C4 39 7E D4 B6 39 7E EF FA 3A 7E 3C F4 3C 7E	e-9"0A9"0!9"0!9"
0040D568	78 8E 39 7E 00 00 00 00 00 00 00 00 00 00 00	0A9".....
0040D578	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00



Newbie Vs UnknowPacker - Revisitado - by El Cid

, se ha rellenado desde el punto indicado. Ahora sí corremos, eh?, je, je.

- La siguiente zona es Buena (véase EDI=4DD038), por lo que no necesitamos llamar al injerto, luego **Undo-Selection** y **Damos Run**. Volvemos a parar en el BP 393857 y la IAT:

004DCFF8	00 00 00 00 00 00 00 00 07 6C DA 77 00 00 00 00
004DD008	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD018	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD028	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD038	86 77 EF 77 79 B4 EF 77 A0 7A EF 77 D3 8B EF 77
004DD048	FB AD EF 77 F7 D9 EF 77 49 D7 EF 77 37 A2 EF 77
004DD058	E3 E0 EF 77 56 6A EF 77 F3 AA EF 77 94 84 EF 77
004DD068	F7 8E EF 77 05 EE EF 77 A3 8E EF 77 71 E6 EF 77
004DD078	34 8E EF 77 20 9C EF 77 A4 9D EF 77 E3 85 EF 77
004DD088	7C 82 EF 77 0A 70 EF 77 E0 5F EF 77 79 6F EF 77
004DD098	5F 6E EF 77 70 5B EF 77 C6 C3 F0 77 75 AC EF 77
004DD0A8	FA 6B EF 77 FA BA EF 77 A1 9A EF 77 36 AD EF 77
004DD0B8	1E EA EF 77 FF B4 EF 77 A5 61 EF 77 6A 5A EF 77
004DD0C8	34 90 EF 77 E8 94 EF 77 7F 90 EF 77 DB SE EF 77
004DD0D8	70 8A EF 77 36 8B EF 77 EF 61 EF 77 29 5E EF 77
004DD0E8	77 5D EF 77 A1 6A EF 77 B1 A7 EF 77 C1 61 EF 77
004DD0F8	E2 A8 EF 77 74 78 EF 77 EC D1 F1 77 B0 6E F0 77
004DD108	0B D1 F1 77 59 6E F0 77 77 C0 FE 77 4C 7B EF 77
004DD118	36 6B F0 77 1B 82 EF 77 17 60 F2 77 00 00 00 00

, vemos que se sigue rellenando.

- La siguiente zona es Buena (véase EDI=4DD570), luego solo tenemos que **dar RUN**, con lo que la IAT:

004DD560	EF FA 3A 7E 3C F4 3C 7E 78 8E 39 7E 00 00 00 00
004DD570	5B 4E B0 76 00 00 00 00 00 00 00 00 00 00 00 00
004DD580	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD590	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD5A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

, se rellena en la única entrada que tiene en esta zona.

- La siguiente zona es Buena (véase EDI=4DD2F8), luego solo tenemos que **dar RUN**, con lo que la IAT:

Address	Hex dump
004DD2F8	F0 10 33 76 9C 11 33 76 00 00 00 00 00 00 00 00
004DD308	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD318	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD328	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD338	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD348	00 00 00 00 F5 B5 39 7E 8C 0C 3A 7E 7D BC 39 7E
004DD358	87 03 3B 7E 27 BE 39 7E 62 07 3A 7E 3F B5 39 7E

, se rellenan las 2 entradas que tiene en esta zona.

- La siguiente zona es Buena (véase EDI=4DD008), luego solo tenemos que **dar RUN**, con lo que en la IAT:

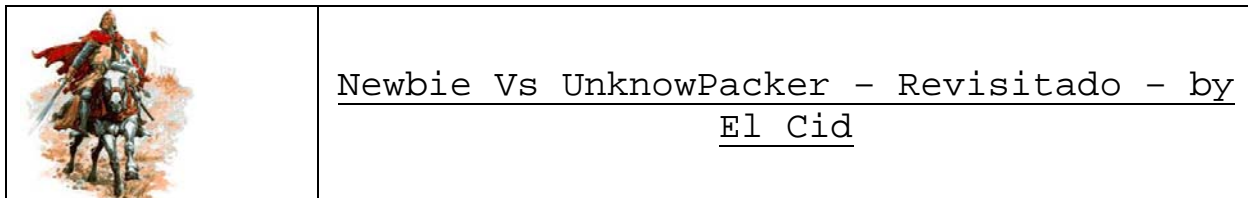
Address	Hex dump
004DD008	34 48 C7 58 05 02 C4 58 D8 03 C4 58 CF 65 C3 58
004DD018	F1 DF C4 58 F4 C7 C3 58 F8 1F C4 58 C5 2E C7 58
004DD028	26 12 C5 58 F1 2F C7 58 9A 22 C6 58 00 00 00 00
004DD038	86 77 EF 77 79 B4 EF 77 A0 7A EF 77 D3 8B EF 77
004DD048	FB AD EF 77 F7 D9 EF 77 49 D7 EF 77 37 A2 EF 77
004DD058	E3 E0 EF 77 56 6A EF 77 F3 AA EF 77 94 84 EF 77

, se rellenan las 11 entradas que tiene en esta zona.

- La siguiente zona es Buena (véase EDI=4DD338), luego solo tenemos que **dar RUN**, con lo que en la IAT:

Address	Hex dump
004DD338	79 67 F4 77 8F 6D F4 77 97 6F F4 77 15 83 F4 77
004DD348	00 00 00 00 F5 B5 39 7E 8C 0C 3A 7E 7D BC 39 7E

, se rellenan las 4 entradas que tiene en esta zona.



Newbie Vs UnknowPacker - Revisitado - by El Cid

- La siguiente zona es Buena (véase EDI=4DD614), luego solo tenemos que **dar RUN**, con lo que en la IAT:

Address	Hex dump
004DD5F4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD604	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD614	FD 2C 4D 77 10 64 4D 77 00 00 00 00 18 25 55 00
004DD624	28 D6 4D 00 05 00 00 00 00 00 00 00 00 00 00 00

, se rellenan las 2 entradas que tiene en esta zona.

- La siguiente zona es Buena (véase EDI=4DD304), luego solo tenemos que **dar RUN**, con lo que en la IAT:

Address	Hex dump
004DD2C4	A9 23 83 7C 04 23 83 7C 8E 20 83 7C 87 0A 81 7C
004DD2D4	0E DE 80 7C 05 DE 80 7C 9D 99 80 7C C9 23 81 7C
004DD2E4	01 4D 83 7C 50 97 80 7C AF 2D 81 7C D5 9B 80 7C
004DD2F4	00 00 00 00 FA 1A 33 76 9C 11 33 76 00 00 00 00
004DD304	E5 79 0F 77 A7 4B 0F 77 20 49 0F 77 FF 6B 0F 77
004DD314	80 49 0F 77 80 48 0F 77 7E 4C 0F 77 6B 4D 0F 77
004DD324	05 4C 0F 77 00 00 00 00 00 00 00 00 00 00 00
004DD334	00 00 00 00 79 67 F4 77 8F 6D F4 77 97 6F F4 77
004DD344	15 83 F4 77 00 00 00 00 F5 B5 39 7E 8C 0C 3A 7E
004DD354	7D BC 39 7E 87 03 3B 7E 27 BE 39 7E 62 07 3A 7E
004DD364	3F B5 39 7E 1B C0 39 7E 69 D8 39 7E 56 90 39 7E

, se rellenan las 9 entradas que tiene en esta zona.

- La siguiente zona es Mala (y si dudamos miramos EDI=4DD580) y además es la última mala. Luego tenemos que volver a llamar al injerto, así que volvemos a parchear como la primera vez. Queda así, de nuevo:

003938DC	AC	LDS	BYTE PTR DS:[ESI]
003938DD	0BC0	OR	EAX, EAX
003938DF	E9 F5891400	JMP	L2Walker.004DC2D9
003938E4	90	NOP	
003938E5	90	NOP	
003938E6	90	NOP	
003938E7	90	NOP	
003938E8	90	NOP	
003938E9	52	PUSH	EDX
003938EA	56	PUSH	ESI
003938EB	55	LEA	EDI, [ESI+4]

Damos Run, con lo que en la IAT:

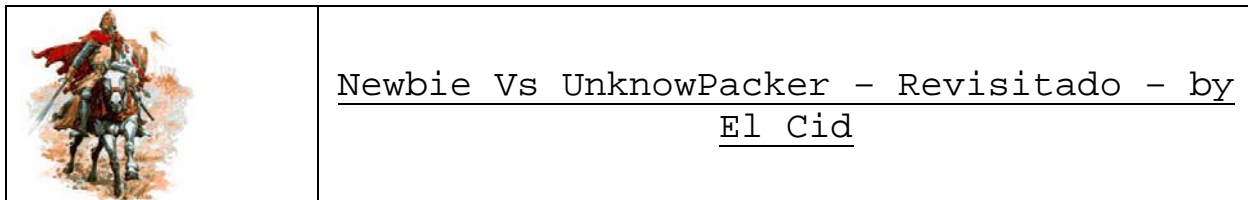
004DD564	3C F4 3C 7E 78 8E 39 7E 00 00 00 00 5B 4E B0 76
004DD574	00 00 00 00 00 00 00 00 00 00 00 00 C0 2D A3 71
004DD584	F4 2B A3 71 66 2B A3 71 39 96 A3 71 8A 42 A3 71
004DD594	5A 61 A3 71 19 45 A3 71 A1 3E A3 71 6A 40 A3 71
004DD5A4	4D 66 A3 71 91 3B A3 71 DC 94 A3 71 04 4F A3 71
004DD5B4	28 44 A3 71 00 00 00 00 00 00 00 00 00 00 00 00
004DD5C4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD5D4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DD5E4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

, se rellenan las 14 entradas que tiene esta zona.

- La siguiente zona es Buena (véase EDI=4DD5BC), por lo que no necesitamos llamar al injerto, luego **Undo-Selection** y **Damos Run**. Volvemos a parar en el BP 393857 y en la IAT:

004DD5A4	4D 66 A3 71 91 3B A3 71 DC 94 A3 71 04 4F A3 71
004DD5B4	28 44 A3 71 00 00 00 00 B5 06 B8 4E 4E 07 B8 4E
004DD5C4	B0 2C B8 4E 0A C2 BE 4E B3 71 B8 4E 9A BC B8 4E
004DD5D4	7D B1 BE 4E 73 2A B8 4E 7E 69 B8 4E 09 E1 B8 4E
004DD5E4	8F C1 BE 4E B4 88 B8 4E 1D 72 B8 4E 5D 34 B8 4E
004DD5F4	E8 34 B8 4E 79 70 B8 4E 86 F3 C3 4E 05 04 BF 4E
004DD604	B5 9B BF 4E 9A 20 C4 4E 09 69 B8 4E 00 00 00 00
004DD614	FD 2C 4D 77 10 64 4D 77 00 00 00 00 18 25 55 00

, se rellenan las 21 entradas que tiene esta zona.



Newbie Vs UnknowPacker - Revisitado - by El Cid

- La siguiente zona es Buena (véase EDI=4DD32C), luego solo tenemos que **dar RUN**, con lo que en la IAT:

004DD31C	7E 4C 0F 77 6B 4D 0F 77 05 4C 0F 77 00 00 00 00
004DD320	FF 31 7A 7E C8 71 75 7E 00 00 00 00 79 67 F4 77
004DD33C	8F 6D F4 77 97 6F F4 77 15 83 F4 77 00 00 00 00
004DD34C	F5 B5 39 7E 8C 0C 3A 7E 7D BC 39 7E 87 03 3B 7E
004DD35C	27 BE 39 7E 62 07 3A 7E 3F B5 39 7E 1B C0 39 7E

, se rellenan las 2 entradas que tiene esta zona.

- La siguiente zona es Buena (véase EDI=4DD578) y además ES LA ÚLTIMA QUE FALTA POR RELLENAR EN LA IAT. No sabemos por tanto si el programa volverá al BP 393857 o irá a hacer otras cosas, pero lo que sí sabemos es que terminará llegando al OEP, luego solo tenemos que poner un BP en el OEP 451715, quitar el BP que tenemos es 393857 porque ya no lo necesitamos:

Address	Hex dump	Disassembly
00451715	6A 60	PUSH 60
00451717	C8 C8665500	PUSH L2Walker.005566C8
0045171C	E8 07290000	CALL L2Walker.004540F8
00451721	BF 94000000	MOV EDI, 94
00451726	8BC7	MOV EAX, EDI
00451728	E8 23140000	CALL L2Walker.
0045172D	8965 E8	MOV DWORD PTR

BP en OEP

, y **dar RUN**, con lo que se terminará de rellenar la IAT por completo:

Address	Hex dump
004DD578	90 53 F8 72 00 00 00 00 C0 2D A3 71 F4 2B A3 71
004DD588	66 2B A3 71 39 96 A3 71 8A 42 A3 71 5A 61 A3 71
004DD598	19 45 A3 71 A1 3E A3 71 6A 40 A3 71 4D 66 A3 71
004DD5A8	91 3B A3 71 DC 94 A3 71 04 4F A3 71 28 44 A3 71
004DD5B8	00 00 00 00 B5 06 BB 4E 4E 07 BB 4E B0 2C BB 4E
004DD5C8	0A C2 BE 4E B3 71 BA 4E 9A BC BB 4E 7D B1 BE 4E



Newbie Vs UnknowPacker - Revisitado - by El Cid

y pararemos en el OEP:

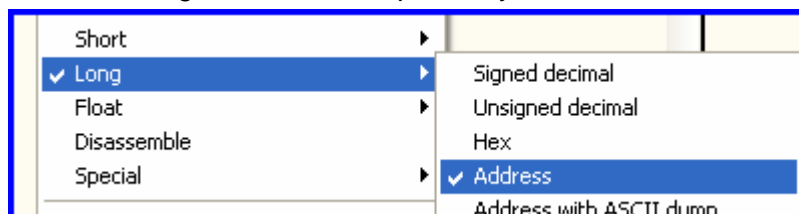
Address	Hex dump	Disassembly
00451715	6A 60	PUSH 60
00451717	68 C8665500	PUSH L2Walker.005
0045171C	E8 D7290000	CALL L2Walker.004
00451721	BF 94000000	MOV EDI, 94
00451726	8BC7	MOV EAX, EDI
00451728	E8 23140000	CALL L2Walker.004
0045172D	8965 E8	MOV DWORD PTR SS:[EBP-20], E8

Estamos
parados en
OEP

Si queremos ver la IAT completamente llena con todas las entradas correctas, aquí la tenemos:

004D0000	07 6C DA 77 00 00 00 00 34 48 C7 58 05 02 C4 58	004D0340	97 6F F4 77 15 83 F4 77 00 00 00 00 F5 B5 39 7E
004D0010	D8 03 C4 58 CF 65 C3 58 F1 DF C4 58 F4 C7 C3 58	004D0350	8C 0C 3A 7E 7D BC 39 7E 87 03 38 7E 27 BE 39 7E
004D0020	F8 1F C4 58 C5 2E C7 58 26 12 C5 58 F1 2F C7 58	004D0360	62 07 3A 7E 3F B5 39 7E 1B C0 39 7E 69 D8 39 7E
004D0030	9A 22 C6 58 00 00 00 00 86 77 EF 77 79 B4 EF 77	004D0370	56 90 39 7E A2 00 3A 7E 29 09 39 7E A9 C4 39 7E
004D0040	A0 7A EF 77 03 8B EF 77 FB AD EF 77 F7 D9 EF 77	004D0380	72 02 3A 7E 86 13 3A 7E 68 EF 3C 7E 7A 14 38 7E
004D0050	49 07 EF 77 37 A2 EF 77 E3 80 EF 77 56 6A EF 77	004D0390	B3 F2 3A 7E C4 F6 3A 7E 05 C5 39 7E EE 50 3E 7E
004D0060	F3 0A EF 77 94 84 EF 77 F7 8E EF 77 05 EE EF 77	004D03A0	2F B8 39 7E 94 BF 39 7E 08 C4 39 7E 1E F2 39 7E
004D0070	A3 8E EF 77 71 E6 EF 77 34 8E EF 77 2D 9C EF 77	004D03B0	EA DA 39 7E 1C F2 3A 7E CE 3D 3A 7E D8 D8 39 7E
004D0080	A4 9D EF 77 E3 85 EF 77 7C 82 EF 77 0A 70 EF 77	004D03C0	07 D9 39 7E 85 3E 3A 7E 3A 15 38 7E 48 BE 39 7E
004D0090	E0 5F EF 77 79 6F EF 77 5F 6E EF 77 70 5B EF 77	004D03D0	0E 97 39 7E 60 DA 39 7E 33 B9 39 7E F0 BE 39 7E
004D00A0	C6 C3 F0 77 75 AC EF 77 FA 68 EF 77 FA BA EF 77	004D03E0	AF C2 3E 7E 58 F8 39 7E DA 94 39 7E 45 5D 3A 7E
004D00B0	A1 9A EF 77 36 AD EF 77 1E 9A EF 77 FF B4 EF 77	004D03F0	B1 B8 3D 7E 5B BA 3D 7E 09 B9 3D 7E 75 5D 3A 7E
004D00C0	A5 61 EF 77 6A 5A EF 77 34 90 EF 77 E8 94 EF 77	004D0400	EC D8 39 7E A2 6D 39 7E 99 00 3D 7E 8F 8F 3A 7E
004D00D0	7F 90 EF 77 D8 5E EF 77 70 8A EF 77 36 8B EF 77	004D0410	52 F8 3C 7E C9 59 3A 7E 5A 8A 3A 7E 58 D6 39 7E
004D00E0	EF 61 EF 77 29 5E EF 77 77 5D EF 77 A1 6A EF 77	004D0420	D5 EE 39 7E 88 C1 39 7E 21 90 39 7E 09 B6 39 7E
004D00F0	B1 A7 EF 77 C1 61 EF 77 E2 A8 EF 77 74 78 EF 77	004D0430	10 B6 39 7E DC 88 3A 7E 7D FB 3A 7E F6 8B 39 7E
004D0100	EC D1 F1 77 80 6E F0 77 0B D1 F1 77 59 6F F0 77	004D0440	D1 E1 3A 7E 86 5F 3D 7E 96 F1 3C 7E 68 03 88 7E
004D0110	77 C0 EF 77 4C 7B EF 77 36 6B F0 77 1B 82 EF 77	004D0450	E2 C2 39 7E AB B8 3D 7E 95 B7 3D 7E C8 EF 39 7E
004D0120	77 6D F2 77 00 00 00 00 30 FE 91 7C 21 FE 91 7C	004D0460	F0 BE 39 7E B2 C2 39 7E AB BE 39 7E 3B 1F 38 7E
004D0130	77 10 80 7C FA 4E 81 7C 73 13 82 7C DF B4 80 7C	004D0470	CD C5 39 7E 55 FA 39 7E 3D EF 39 7E 26 BF 39 7E
004D0140	B1 B6 80 7C 87 D0 80 7C EE AB 80 7C 3F FC 80 7C	004D0480	1D C7 39 7E 8E BD 39 7E 75 E8 39 7E 95 38 7E
004D0150	E7 ED 80 7C 20 25 80 7C 13 CE 81 7C 47 06 81 7C	004D0490	7D DB 39 7E D2 DC 39 7E 49 07 39 7E DD 7D 3E 7E
004D0160	3D FD 80 7C 29 FF 80 7C 92 FE 80 7C EA 11 81 7C	004D04A0	55 FE 3E 7E BC 22 3A 7E 3E DA 39 7E BC C8 3E 7E
004D0170	E0 10 91 7C 00 10 91 7C 42 24 80 7C C5 9F 80 7C	004D04B0	3D EF 39 7E 2E 8C 39 7E 78 D8 3A 7E 52 F0 39 7E
004D0180	A7 CC 80 7C 79 BC 80 7C 7A 13 92 7C 01 9F 80 7C	004D04C0	0F F9 3A 7E 07 E9 38 7E 8F 8C 3A 7E 28 13 88 7E
004D0190	19 2A 81 7C 25 A4 80 7C 72 D2 80 7C 25 99 80 7C	004D04D0	E8 D9 39 7E 1E C2 39 7E 53 5F 3D 7E B2 FF 39 7E
004D01A0	9E 97 80 7C 0C 34 83 7C 13 DC 81 7C 54 8A 83 7C	004D04E0	6C BF 39 7E 4D 3D 3A 7E 76 BD 39 7E 57 C2 39 7E
004D01B0	9B C4 81 7C 29 7D 83 7C 20 9F 80 7C DF BC 80 7C	004D04F0	EA F8 3A 7E C8 BD 39 7E C7 86 39 7E 9D 86 39 7E
004D01C0	11 9E 80 7C B7 27 81 7C 30 99 80 7C B8 92 80 7C	004D0500	6E CE 39 7E 41 BD 39 7E CE D6 39 7E 58 8F 39 7E
004D01D0	37 A4 80 7C ED 47 84 7C 30 8E 83 7C 07 51 83 7C	004D0510	EA C6 39 7E 64 5D 3D 7E 29 8C 3A 7E 9C 8F 39 7E
004D01E0	36 62 83 7C 75 63 83 7C 89 9E 80 7C F4 9A 80 7C	004D0520	7D 1A 3E 7E 37 02 38 7E 25 02 38 7E 5E 0F 38 7E
004D01F0	C6 2B 81 7C 08 0F 81 7C EE 1E 80 7C 61 0E 81 7C	004D0530	56 0D 3E 7E D4 09 39 7E 87 F7 39 7E 31 B6 39 7E
004D0200	A7 CC 80 7C 93 6F 81 7C 2D 2F 81 7C 68 CF 81 7C	004D0540	F9 07 39 7E A4 D8 39 7E 42 8C 39 7E 4E EB 39 7E
004D0210	87 DF 81 7C 4A 30 86 7C 49 2F 81 7C DD 04 92 7C	004D0550	AE B6 39 7E 2F B7 39 7E 65 C4 39 7E D4 B6 39 7E
004D0220	16 1E 80 7C 68 C0 80 7C EA CD 80 7C E9 74 92 7C	004D0560	EF FA 3A 7E 3C F4 3C 7E 78 8E 39 7E 00 00 00 00
004D0230	E1 B9 80 7C 66 2D 81 7C 61 9A 80 7C 00 1A 80 7C	004D0570	5B 4E B0 7E 00 00 00 00 90 53 F8 72 00 00 00 00
004D0240	C4 00 92 7C E5 17 80 7C 2D FF 91 7C FA A7 93 7C	004D0580	C0 2D A3 71 F4 2B A3 71 66 2B A3 71 39 96 A3 71
004D0250	1F AC 80 7C 65 1C 83 7C 76 E8 80 7C FC E7 80 7C	004D0590	8A 42 A3 71 5A 61 A3 71 19 45 A3 71 A1 3E A3 71
004D0260	BA 67 83 7C E7 36 81 7C 43 09 83 7C EE 2A 81 7C	004D05A0	6A 40 A3 71 4D 66 A3 71 91 38 A3 71 DC 94 A3 71
004D0270	C6 8D 80 7C DB 08 83 7C 40 97 80 7C D2 60 82 7C	004D05B0	04 4F A3 71 28 44 A3 71 00 00 00 00 B5 06 B8 4E
004D0280	D6 97 80 7C 3F 99 80 7C 57 9B 80 7C 18 C1 80 7C	004D05C0	4E 07 B8 4E B0 2C B8 4E 0A C2 BE 4E B3 71 B8 4E
004D0290	3F 29 83 7C 27 A0 80 7C 7A 97 83 7C 51 F8 85 7C	004D05D0	9A BC B8 4E 7D B1 BE 4E 73 2A B8 4E 7E 69 B8 4E
004D02A0	17 84 83 7C 94 0D 83 7C FB 98 80 7C 8A 97 80 7C	004D05E0	09 E1 B8 4E 8F C1 BE 4E B4 88 B8 4E 1D 72 B8 4E
004D02B0	76 97 80 7C 0E 18 80 7C 97 0D 81 7C 9E 0B 81 7C	004D05F0	5D 34 B8 4E E8 34 B8 4E 79 7A B8 4E 86 E3 C3 4E
004D02C0	51 26 81 7C A9 23 83 7C 04 23 83 7C 8E 20 83 7C	004D0600	05 04 BF 4E A5 98 BF 4E 9A 2A C4 4E A9 69 B8 4E
004D02D0	87 0A 81 7C 0E DE 80 7C 05 DE 80 7C 9D 99 80 7C	004D0610	00 00 00 00 FD 2C 4D 77 10 64 4D 77 00 00 00 00
004D02E0	C9 23 81 7C 01 4D 83 7C 50 97 80 7C AF 2D 81 7C		
004D02F0	D5 9B 80 7C 00 00 00 00 F0 10 33 76 9C 11 33 76		
004D0300	00 00 00 00 E5 79 0F 77 A7 4B 0F 77 20 49 0F 77		
004D0310	FF 68 0F 77 80 49 0F 77 80 48 0F 77 7E 4C 0F 77		
004D0320	68 4D 0F 77 05 4C 0F 77 00 00 00 00 FF 31 7A 7E		
004D0330	C8 71 75 7E 00 00 00 00 79 67 F4 77 8F 6D F4 77		
004D0340	97 6F F4 77 15 83 F4 77 00 00 00 00 F5 B5 39 7E		
004D0350	8C 0C 3A 7E 7D BC 39 7E 87 03 38 7E 27 BE 39 7E		
004D0360	62 07 3A 7E 3F B5 39 7E 1B C0 39 7E 69 D8 39 7E		

Incluso si elegimos en el Dump de Olly la vista:





Newbie Vs UnknowPacker - Revisitado - by El Cid

, veremos los nombres de las APIs:

00400000	770A6C07	ADVAPI32.RegCloseKey	004000CC	77EF94E8	GDI32.SetMapMode
00400004	00000000		00400000	77EF907F	GDI32.SetStretchBltMode
00400008	58C74834	COMCTL32.ImageList_LoadImageW	00400004	77EF5E08	GDI32.SetBkMode
0040000C	58C40205	COMCTL32.ImageList_Create	00400008	77EF8A70	GDI32.RestoreDC
00400010	58C40308	COMCTL32.ImageList_Destroy	0040000C	77EF8B36	GDI32.SaveDC
00400014	58C365CF	COMCTL32.InitCommonControls	0040000E	77EF61EF	GDI32.CreateBitmap
00400018	58C40FF1	COMCTL32.ImageList_Draw	004000E4	77EF5E29	GDI32.SetBkColor
0040001C	58C3C7F4	COMCTL32.ImageList_ReplaceIcon	004000E8	77EF5D77	GDI32.SetTextColor
00400020	58C41FF8	COMCTL32.ImageList_AddMasked	004000EC	77EF6AA1	GDI32.GetClipBox
00400024	58C72EC5	COMCTL32.ImageList_Add	004000F0	77EFA7B1	GDI32.SetDIBits
00400028	58C51226	COMCTL32.TrackMouseEvent	004000F4	77EF61C1	GDI32.GetStockObject
0040002C	58C72FF1	COMCTL32.ImageList_GetImageInfo	004000F8	77EFA8E2	GDI32.CreatePatternBrush
00400030	58C6229A	COMCTL32.ImageList_GetIcon	004000FC	77EF7874	GDI32.ExtSelectClipRgn
00400034	00000000		00400100	77F1D1EC	GDI32.ScaleWindowExtEx
00400038	77EF7786	GDI32.CreateRectRgn	00400104	77F06EB0	GDI32.SetWindowExtEx
0040003C	77EFB479	GDI32.GetPixel	00400108	77F1D108	GDI32.ScaleViewportExtEx
00400040	77EF7A00	GDI32.SelectClipRgn	0040010C	77F06F59	GDI32.SetViewportExtEx
00400044	77EF8803	GDI32.SetTextAlign	00400110	77EFC077	GDI32.OffsetViewportOrgEx
00400048	77EFA0FB	GDI32.MoveToEx	00400114	77EF7B4C	GDI32.SetViewportOrgEx
0040004C	77EFD09F	GDI32.LineTo	00400118	77F06B36	GDI32.Escape
00400050	77EFD749	GDI32.GetRgnBox	0040011C	77EF821B	GDI32.RectVisible
00400054	77EFA237	GDI32.CreatePolygonRgn	00400120	77F26017	GDI32.PtVisible
00400058	77EFE0E3	GDI32.Polyline	00400124	00000000	
0040005C	77EFAA56	GDI32.IntersectClipRect	00400128	7C91FE30	ntdll.RtlSetLastWin32Error
00400060	77EFAAF3	GDI32.GetDIBits	0040012C	7C91FE21	ntdll.RtlGetLastWin32Error
00400064	77EF8494	GDI32.GetTextCharsetInfo	00400130	7C801D77	kernel32.LoadLibraryA
00400068	77EF8EF7	GDI32.GetTextColor	00400134	7C814EFA	kernel32.GetSystemDirectoryA
0040006C	77EFE095	GDI32.Polygon	00400138	7C821373	kernel32.GetWindowsDirectoryA
00400070	77EF8EA3	GDI32.GetBkColor	0040013C	7C80B4DF	kernel32.GetModuleFileNameA
00400074	77EFE671	GDI32.DPtoLP	00400140	7C80B6B1	kernel32.GetModuleHandleA
00400078	77EF8E34	GDI32.GetMapMode	00400144	7C80D087	kernel32.CompareStringA
0040007C	77EF9C2D	GDI32.CombineRgn	00400148	7C80ABEE	kernel32.FreeLibrary
00400080	77EF90A4	GDI32.SetRectRgn	0040014C	7C80FC3F	kernel32.GlobalFree
00400084	77EF85E3	GDI32.PatBlt	00400150	7C80E0E7	kernel32.FindClose
00400088	77EF827C	GDI32.CreateRectRgnIndirect	00400154	7C802520	kernel32.WaitForSingleObject
0040008C	77EF700A	GDI32.CreateCompatibleBitmap	00400158	7C81CE13	kernel32.TerminateThread
00400090	77EF5FE0	GDI32.CreateCompatibleDC	0040015C	7C810647	kernel32.CreateThread
00400094	77EF6F79	GDI32.BitBlt	00400160	7C80FD3D	kernel32.GlobalAlloc
00400098	77EF6E5F	GDI32.DeleteDC	00400164	7C80FF29	kernel32.GlobalLock
0040009C	77EF5B70	GDI32.SelectObject	00400168	7C80FE92	kernel32.GlobalUnlock
004000A0	77F0C3C6	GDI32.SetDIBColorTable	0040016C	7C8111EA	kernel32.GetVersion
004000A4	77EFA075	GDI32.GetDIBColorTable	00400170	7C9110E0	ntdll.RtlLeaveCriticalSection
004000A8	77EF6BFA	GDI32.DeleteObject	00400174	7C911000	ntdll.RtlEnterCriticalSection
004000AC	77EFBAFA	GDI32.StretchBlt	00400178	7C802442	kernel32.Sleep
004000B0	77EF9AA1	GDI32.CreateDIBSection	0040017C	7C809FC5	kernel32.LoadResource
004000B4	77EFA036	GDI32.CreatePen	00400180	7C80CCA7	kernel32.SetHandleCount
004000B8	77EFA01E	GDI32.Rectangle	00400184	7C80BC79	kernel32.SizeOfResource
004000BC	77EFB4FF	GDI32.SetPixel	00400188	7C92137A	ntdll.RtlDeleteCriticalSection
004000C0	77EF61A5	GDI32.CreateSolidBrush	0040018C	7C809F01	kernel32.InitializeCriticalSection
004000C4	77EF5A6A	GDI32.GetDeviceCaps	00400190	7C812A19	kernel32.RaiseException
004000C8	77EF9034	GDI32.ExcludeClipRect	00400194	7C80A425	kernel32.GetThreadLocale
004000CC	77EF94E8	GDI32.SetMapMode			

, y resto de entradas que no aparecen.

¡Qué gusto ver todas las entradas correctas y con las direcciones reales de las APIs!

Buf, ha costado pero lo hemos conseguido.

12. Dumpeado y reparación final

En este momento sólo queda dumpear y arreglar el volcado con ImpRect.

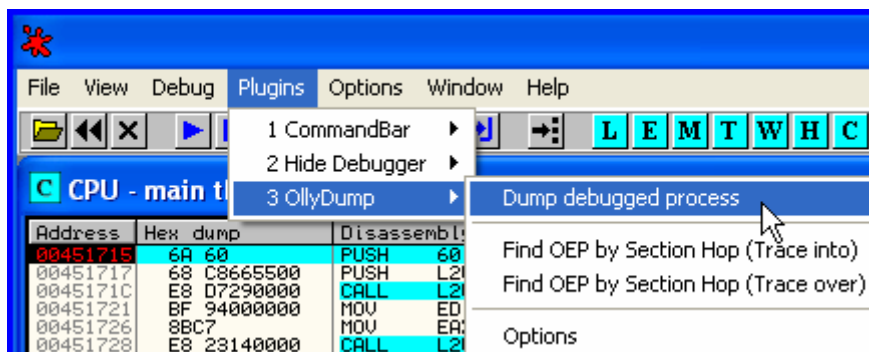
Yo, antes de hacer nada lo que hago en estos casos, es copiarme la IAT completa, marcándola con el ratón y haciendo Binary Copy. Después me la llevo a cualquier editor hexadecimal o simplemente a Word. Ésto lo hago como medida de seguridad porque si en el momento de dumpear se produce algún problema, ya la tengo salvada y la puedo regenerar rápidamente.

Pues bien, si lo habeis hecho es el momento de dumpear.

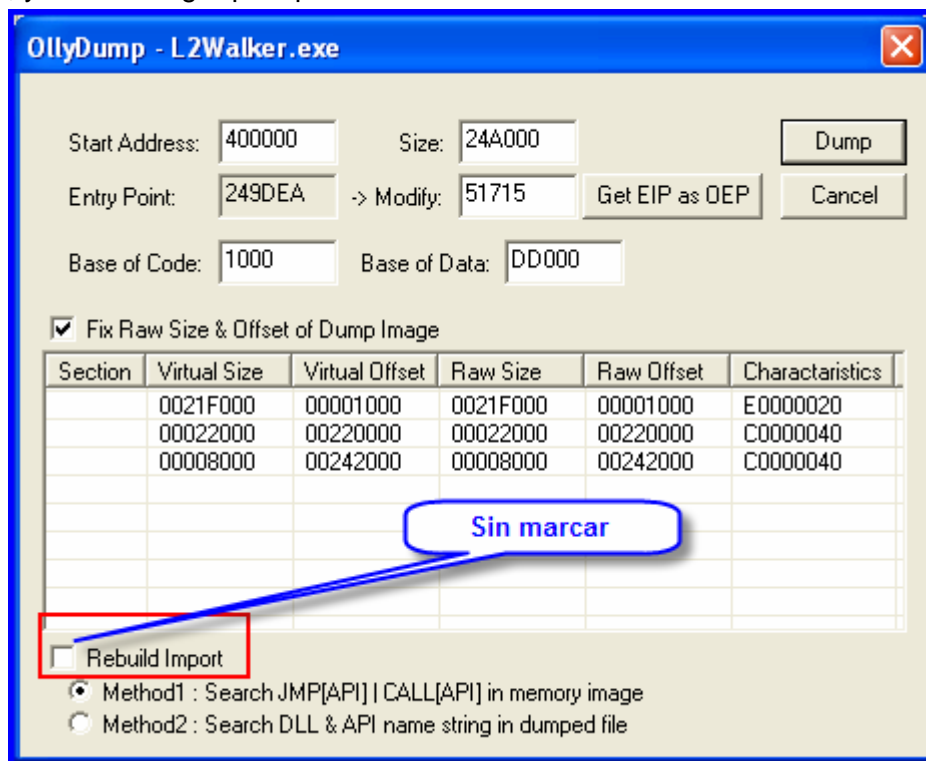


Newbie Vs UnknowPacker - Revisitado - by El Cid

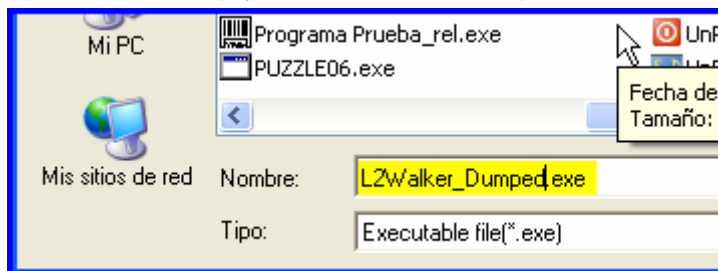
Hacemos:



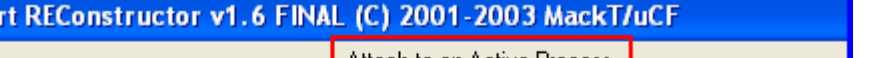
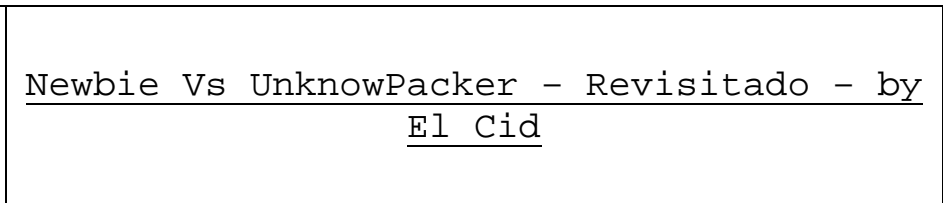
, y en el diálogo que aparece rellenamos los datos así:



, pulsamos Dump y nombramos el dumpeado como se ve en la figura:



, y pulsamos Aceptar.



Import REConstructor v1.6 FINAL (C) 2001-2003 MackT/uCF

Attach to an Active Process

- d:\ollydbg\tools auxiliares\import rec\importrec.exe (0000011C)
- d:\ollydbg\programas de trabajo\l2walker.exe (00000DF0)
- c:\archivos de programa\internet explorer\iexplore.exe (00000688)
- c:\archivos de programa\microsoft office\office11\winword.exe (000007CC)
- d:\ollydbg\parcheado 4.exe (000002CC)

IAT Infos needed

OEP 00051715 IAT AutoSearch

RVA 000DD000 Size 61C

Load Tree Save Tree Get Imports

The screenshot displays the PE Explorer application interface. The 'Imported Functions Found' list is highlighted with a red rectangle, showing a table of imported functions with columns for the function name, RVA, and size. The 'Log' window below it shows the current imports and the 'Fix Dump' button is highlighted with a red rectangle. The 'IAT Infos needed' section shows the 'IAT AutoSearch' button and the 'New Import Infos (IID+ASCII+LOADER)' section shows the 'Add new section' checkbox and the 'Fix Dump' button.

Function Name	RVA	Size
advapi32.dll FTunk:000DD000 NbFunc:1 (decimal:1) valid:YES		
comctl32.dll FTunk:000DD008 NbFunc:8 (decimal:11) valid:YES		
gdi32.dll FTunk:000DD038 NbFunc:38 (decimal:59) valid:YES		
kernel32.dll FTunk:000DD128 NbFunc:73 (decimal:115) valid:YES		
msimg32.dll FTunk:000DD2F8 NbFunc:2 (decimal:2) valid:YES		
oleaut32.dll FTunk:000DD304 NbFunc:9 (decimal:9) valid:YES		
shell32.dll FTunk:000DD32C NbFunc:2 (decimal:2) valid:YES		
shlwapi.dll FTunk:000DD338 NbFunc:4 (decimal:4) valid:YES		
user32.dll FTunk:000DD34C NbFunc:88 (decimal:136) valid:YES		

Log

rva:000DD248 forwarded from mod:ntdll.dll ord:01BE name:RtlFreeHeap
rva:000DD24C forwarded from mod:ntdll.dll ord:0209 name:RtlUnwind

Current imports:
E (decimal:14) valid module(s) (added: +E (decimal:+14))
17A (decimal:378) imported function(s). (added: +17A (decimal:+378))

IAT Infos needed

OEP 00051715 IAT AutoSearch

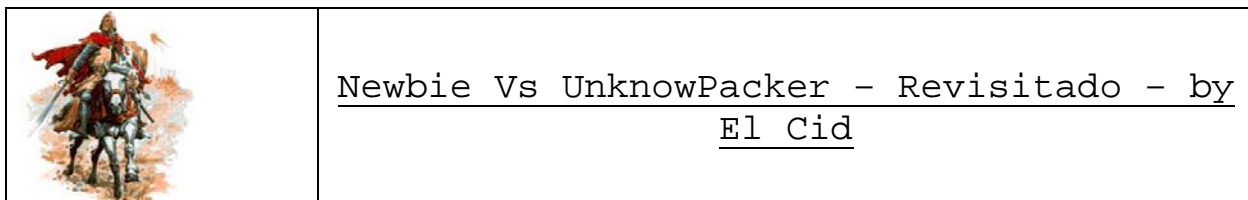
RVA 000DD000 Size 0000061C

New Import Infos (IID+ASCII+LOADER)

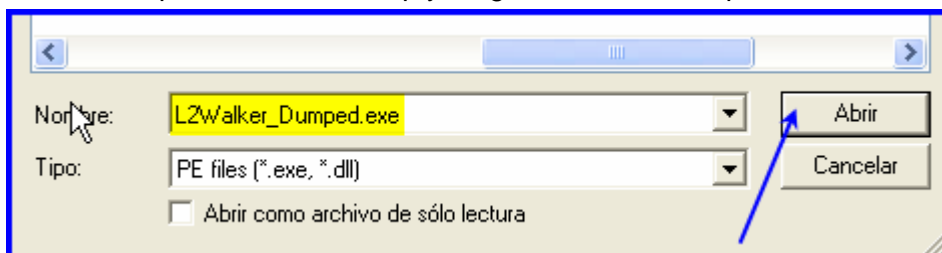
RVA 00000000 Size 00001AE6

☒ Add new section

Fix Dump

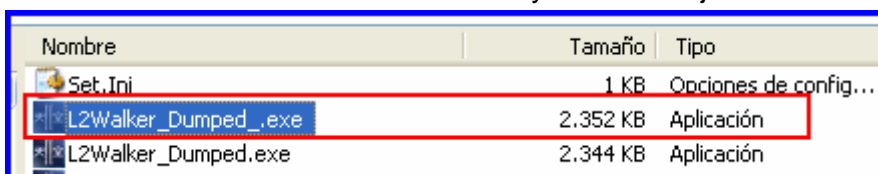


Finalmente pulsamos Fix Dump y elegimos el volcado que hemos salvado antes:

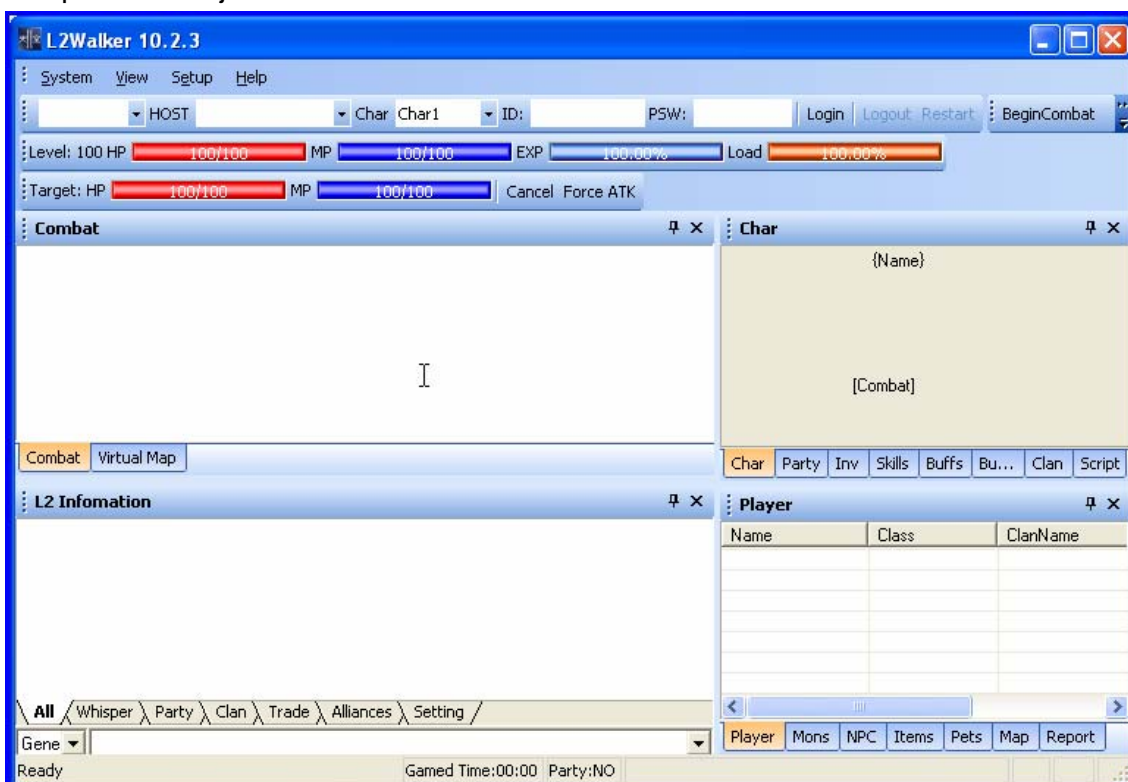


, y damos Aceptar (Abrir).

Se crea el fichero con el mismo nombre y un trazo bajo:




No queda sino ejecutarlo a ver si funciona. Hacemos doble click en dicho nombre:



, y funciona perfectamente.

Cerramos Import REConstructor, cerramos Olly y hemos acabado.

	<p><u>Newbie Vs UnknowPacker – Revisitado – by</u> <u>El Cid</u></p>
--	--

13. Notas finales

Unos comentarios finales.

- Ciertamente han salido una cantidad de páginas enorme, si hubiera pensado que saldrían tantas no sé si hubiera empezado a escribir el tuto. Sin embargo tengo la sensación de que realmente no han sido tantos los problemas a resolver. La idea era muy sencilla, lo que pasa que a veces cuesta más explicarlo que hacerlo.
Reparar la IAT me costó unas horas y sin embargo escribir el tuto han sido 3 ó 4 días.
- Soy consciente de que a muchos de vosotros el ritmo de avance os habrá resultado muy lento. Pido disculpas a quien se haya impacientado leyendo cosas que le hayan parecido obvias o requetesabidas, pero creo que siempre hay quien tiene menos conocimientos y le pueden venir bien unas explicaciones adicionales o un ritmo más lento. Siempre suelo pensar en mis propios comienzos que fueron duros y por éso a veces voy despacio.
- Gracias a todos los que os hayais molestado en leer hasta aquí. Espero que os hayan resultado útiles estas páginas y por descontado son bienvenidos comentarios constructivos acerca del tuto.

Un saludo a toda la lista de CracksLatinos

23 Noviembre 2010

El Cid



Newbie Vs UnknowPacker - Revisitado - by El Cid

ANEXO 1

IAT original completa, con entradas buenas y malas

00400000	07 6C DA 77 00 00 00 00 34 48 C7 58 05 02 C4 58	00400340	97 6F F4 77 15 83 F4 77 00 00 00 00 00 00 B4 00
00400010	08 03 C4 58 CF 65 C3 58 F1 DF C4 58 F4 C7 C3 58	00400350	20 00 B4 00 40 00 B4 00 60 00 B4 00 80 00 B4 00
00400020	F8 1F C4 58 C5 2E C7 58 26 12 C5 58 F1 2F C7 58	00400360	A0 00 B4 00 C0 00 B4 00 E0 00 B4 00 00 01 B4 00
00400030	9A 22 C6 58 00 00 00 00 86 77 EF 77 79 B4 EF 77	00400370	20 01 B4 00 40 01 B4 00 60 01 B4 00 80 01 B4 00
00400040	A0 7A EF 77 D3 8B EF 77 F8 AD EF 77 F7 D9 EF 77	00400380	A0 01 B4 00 C0 01 B4 00 E0 01 B4 00 00 02 B4 00
00400050	49 D7 EF 77 37 A2 EF 77 E3 E0 EF 77 56 6A EF 77	00400390	20 02 B4 00 40 02 B4 00 60 02 B4 00 80 02 B4 00
00400060	F3 AA EF 77 94 84 EF 77 F7 8E EF 77 05 EE EF 77	004003A0	A0 02 B4 00 C0 02 B4 00 E0 02 B4 00 00 03 B4 00
00400070	A3 8E EF 77 71 E6 EF 77 34 8E EF 77 20 9C EF 77	004003B0	20 03 B4 00 40 03 B4 00 60 03 B4 00 80 03 B4 00
00400080	A4 9D EF 77 E3 85 EF 77 7C 82 EF 77 0A 70 EF 77	004003C0	A0 03 B4 00 C0 03 B4 00 E0 03 B4 00 00 04 B4 00
00400090	E0 5F EF 77 79 6F EF 77 5F 6E EF 77 70 5B EF 77	004003D0	20 04 B4 00 40 04 B4 00 60 04 B4 00 80 04 B4 00
004000A0	C6 C3 F0 77 75 AC EF 77 FA 68 EF 77 FA BA EF 77	004003E0	A0 04 B4 00 C0 04 B4 00 E0 04 B4 00 00 05 B4 00
004000B0	A1 9A EF 77 36 AD EF 77 1E EA EF 77 FF BA EF 77	004003F0	20 05 B4 00 40 05 B4 00 60 05 B4 00 80 05 B4 00
004000C0	A5 61 EF 77 6A 5A EF 77 34 90 EF 77 E3 94 EF 77	00400400	A0 05 B4 00 C0 05 B4 00 E0 05 B4 00 00 06 B4 00
004000D0	7F 90 EF 77 DB 5E EF 77 70 8A EF 77 36 8B EF 77	00400410	20 06 B4 00 40 06 B4 00 60 06 B4 00 80 06 B4 00
004000E0	EF 61 EF 77 29 5E EF 77 77 5D EF 77 A1 6A EF 77	00400420	A0 06 B4 00 C0 06 B4 00 E0 06 B4 00 00 07 B4 00
004000F0	B1 A7 EF 77 C1 61 EF 77 E2 A8 EF 77 74 78 EF 77	00400430	20 07 B4 00 40 07 B4 00 60 07 B4 00 80 07 B4 00
00400100	EC D1 F1 77 B0 6E F0 77 0B D1 F1 77 59 6F F0 77	00400440	A0 07 B4 00 C0 07 B4 00 E0 07 B4 00 00 08 B4 00
00400110	77 C0 EF 77 4C 7B EF 77 36 68 F0 77 1B 82 EF 77	00400450	20 08 B4 00 40 08 B4 00 60 08 B4 00 80 08 B4 00
00400120	17 60 F2 77 00 00 00 00 00 00 B3 00 20 00 B3 00	00400460	A0 08 B4 00 C0 08 B4 00 E0 08 B4 00 00 09 B4 00
00400130	40 00 B3 00 60 00 B3 00 80 00 B3 00 A0 00 B3 00	00400470	20 09 B4 00 40 09 B4 00 60 09 B4 00 80 09 B4 00
00400140	C0 00 B3 00 E0 00 B3 00 00 01 B3 00 20 01 B3 00	00400480	A0 09 B4 00 C0 09 B4 00 E0 09 B4 00 00 0A B4 00
00400150	40 01 B3 00 60 01 B3 00 80 01 B3 00 A0 01 B3 00	00400490	20 0A B4 00 40 0A B4 00 60 0A B4 00 80 0A B4 00
00400160	C0 01 B3 00 E0 01 B3 00 00 02 B3 00 20 02 B3 00	004004A0	A0 0A B4 00 C0 0A B4 00 E0 0A B4 00 00 0B B4 00
00400170	40 02 B3 00 60 02 B3 00 80 02 B3 00 A0 02 B3 00	004004B0	20 0B B4 00 40 0B B4 00 60 0B B4 00 80 0B B4 00
00400180	C0 02 B3 00 E0 02 B3 00 00 03 B3 00 20 03 B3 00	004004C0	A0 0B B4 00 C0 0B B4 00 E0 0B B4 00 00 0C B4 00
00400190	40 03 B3 00 60 03 B3 00 80 03 B3 00 A0 03 B3 00	004004D0	20 0C B4 00 40 0C B4 00 60 0C B4 00 80 0C B4 00
004001A0	C0 03 B3 00 E0 03 B3 00 00 04 B3 00 20 04 B3 00	004004E0	A0 0C B4 00 C0 0C B4 00 E0 0C B4 00 00 0D B4 00
004001B0	40 04 B3 00 60 04 B3 00 80 04 B3 00 A0 04 B3 00	004004F0	20 0D B4 00 40 0D B4 00 60 0D B4 00 80 0D B4 00
004001C0	C0 04 B3 00 E0 04 B3 00 00 05 B3 00 20 05 B3 00	00400500	A0 0D B4 00 C0 0D B4 00 E0 0D B4 00 00 0E B4 00
004001D0	40 05 B3 00 60 05 B3 00 80 05 B3 00 A0 05 B3 00	00400510	20 0E B4 00 40 0E B4 00 60 0E B4 00 80 0E B4 00
004001E0	C0 05 B3 00 E0 05 B3 00 00 06 B3 00 20 06 B3 00	00400520	A0 0E B4 00 C0 0E B4 00 E0 0E B4 00 00 0F B4 00
004001F0	40 06 B3 00 60 06 B3 00 80 06 B3 00 A0 06 B3 00	00400530	20 0F B4 00 40 0F B4 00 60 0F B4 00 80 0F B4 00
00400200	C0 06 B3 00 E0 06 B3 00 00 07 B3 00 20 07 B3 00	00400540	A0 0F B4 00 C0 0F B4 00 E0 0F B4 00 00 10 B4 00
00400210	40 07 B3 00 60 07 B3 00 80 07 B3 00 A0 07 B3 00	00400550	20 10 B4 00 40 10 B4 00 60 10 B4 00 80 10 B4 00
00400220	C0 07 B3 00 E0 07 B3 00 00 08 B3 00 20 08 B3 00	00400560	A0 10 B4 00 C0 10 B4 00 E0 10 B4 00 00 00 00 00
00400230	40 08 B3 00 60 08 B3 00 80 08 B3 00 A0 08 B3 00	00400570	5B 4E B0 76 00 00 00 00 90 53 F8 72 00 00 00 00
00400240	C0 08 B3 00 E0 08 B3 00 00 09 B3 00 20 09 B3 00	00400580	00 00 BE 00 20 00 BE 00 40 00 BE 00 60 00 BE 00
00400250	40 09 B3 00 60 09 B3 00 80 09 B3 00 A0 09 B3 00	00400590	80 00 BE 00 A0 00 BE 00 C0 00 BE 00 E0 00 BE 00
00400260	C0 09 B3 00 E0 09 B3 00 00 0A B3 00 20 0A B3 00	004005A0	00 01 BE 00 20 01 BE 00 40 01 BE 00 60 01 BE 00
00400270	40 0A B3 00 60 0A B3 00 80 0A B3 00 A0 0A B3 00	004005B0	80 01 BE 00 A0 01 BE 00 C0 01 BE 00 E0 01 BE 00
00400280	C0 0A B3 00 E0 0A B3 00 00 0B B3 00 20 0B B3 00	004005C0	4E 07 BE 4E B0 2C BE 4E 0A C2 BE 4E B3 71 BA 4E
00400290	40 0B B3 00 60 0B B3 00 80 0B B3 00 A0 0B B3 00	004005D0	9A BC BE 4E 7D B1 BE 4E 73 2A BB 4E 7E 69 BA 4E
004002A0	C0 0B B3 00 E0 0B B3 00 00 0C B3 00 20 0C B3 00	004005E0	09 E1 BA 4E 8F C1 BE 4E B4 8B BB 4E 1D 72 BA 4E
004002B0	40 0C B3 00 60 0C B3 00 80 0C B3 00 A0 0C B3 00	004005F0	5D 34 BB 4E E8 34 BB 4E 79 7A BB 4E 86 E3 C3 4E
004002C0	C0 0C B3 00 E0 0C B3 00 00 0D B3 00 20 0D B3 00	00400600	05 04 BF 4E A5 98 BF 4E 9A 2A C4 4E A9 69 BA 4E
004002D0	40 0D B3 00 60 0D B3 00 80 0D B3 00 A0 0D B3 00	00400610	00 00 00 00 FD 2C 4D 77 10 64 4D 77 00 00 00 00
004002E0	C0 0D B3 00 E0 0D B3 00 00 0E B3 00 20 0E B3 00	00400620	18 25 55 00 28 D6 4D 00 05 00 00 00 00 00 00 00
004002F0	40 0E B3 00 60 0E B3 00 80 0E B3 00 A0 0E B3 00		
00400300	00 00 00 00 E5 79 0F 77 A7 48 0F 77 20 49 0F 77		
00400310	FF 68 0F 77 80 49 0F 77 80 48 0F 77 7E 4C 0F 77		
00400320	6B 4D 0F 77 05 4C 0F 77 00 00 00 00 FF 31 7A 7E		
00400330	C8 71 75 7E 00 00 00 00 79 67 F4 77 8F 6D F4 77		
00400340	97 6F F4 77 15 83 F4 77 00 00 00 00 00 00 B4 00		



Newbie Vs UnknowPacker - Revisitado - by
El Cid