



1.- INTRODUCCIÓN.

Algunos esperaban ésta segunda parte. Yo también, jeje, pero escribirla, no he tenido tiempo hasta ahora y retomo lo dicho para seguir con la incentiva del curso. Espero que éste tutorial empuje a algunos a desempaquetar sin problemas cualquier **packer** (compresor). Es hora de meternos de lleno.

2.-HERRAMIENTAS.

- [Import Reconstructor](#).
- [OllyDBG](#).
- [Lord PE](#). (opcional, pero recomendable, version **DELUXE**).
- [PE Tools](#).
- [PE Editor](#).

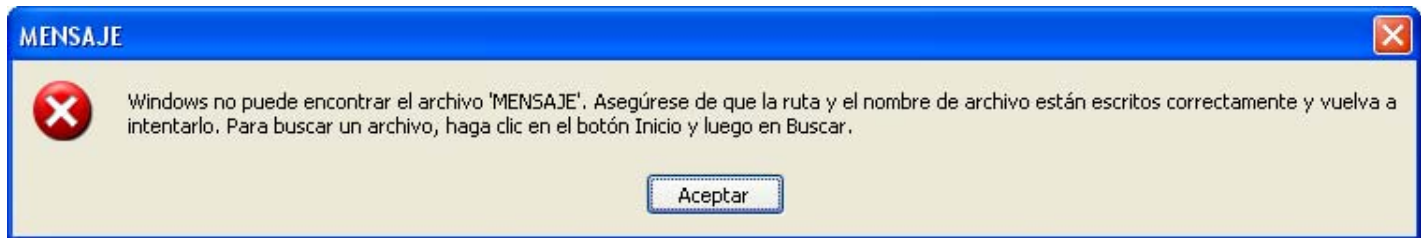
Nota: [Muchas herramientas hacen lo mismo, pero las pongo para que cada uno utilize la que más le guste.](#)

3.- EMPEZAMOS ☺.

Como no, un poquito de teoría. Tenemos que saber que es la **IAT** (Import **A**dress **T**able / Tabla de dirección de importaciones). La **IAT** es una tabla que contienen todos los programas en windows, y lo que contiene son las llamadas **API**. La wikipedia nos dice esto:

API (del inglés **A**pplication **P**rogramming **I**nterface - **I**nterfaz de **P**rogramación de **A**plicaciones) es el conjunto de **funciones** y **procedimientos** (o **métodos** si se refiere a programación orientada a objetos) que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción.

Para que os hagáis una idea, las dll de windows tan conocidas, "[kernel32.dll](#)", "[ntdll.dll](#)", "[user32.dll](#)", y un largo etcétera. Contienen éstas API, con lo que podemos hacer cualquier tipo de cosa, un ejemplo práctico es el típico mensaje de windows.



Está es la api llamada "[MessageBoxA](#)" creo que es el ejemplo más práctico. Bien, como esta api hay miles, cada una se encarga de hacer algo en concreto, así que, lo que contiene la [IAT](#) son todas esas direcciones a esas [API](#), que en cada máquina [cambian](#), es decir, como no todos tenemos las mismas versiones de la [kernel](#), ni de la [user32](#), ya que, unos usamos [SP2](#), otros no, otros [Windows Vista](#), otros [Windows 2000](#), etc... se necesita resolver esas direcciones, es decir, saber cuales son, y por ello se llevan a esa tabla, donde cada máquina (PC) las resuelve automáticamente. Eso es la [IAT](#), y por lo tanto lo que ocurre cuando tenemos el programa que queremos desempaquetar en el [OEP](#), es que cuando lo [dumpeamos](#) (volcar), es decir, copiar todos esos bytes descomprimidos en un nuevo [EJECUTABLE](#), nos guarda las direcciones de las [API](#) de [NUESTRA](#) máquina. Y nosotros queremos guardar esa tabla que resuelve las direcciones en [TODAS](#) las máquinas.

Teniendo el programa en el mismo punto del tute anterior, en el [OEP](#).

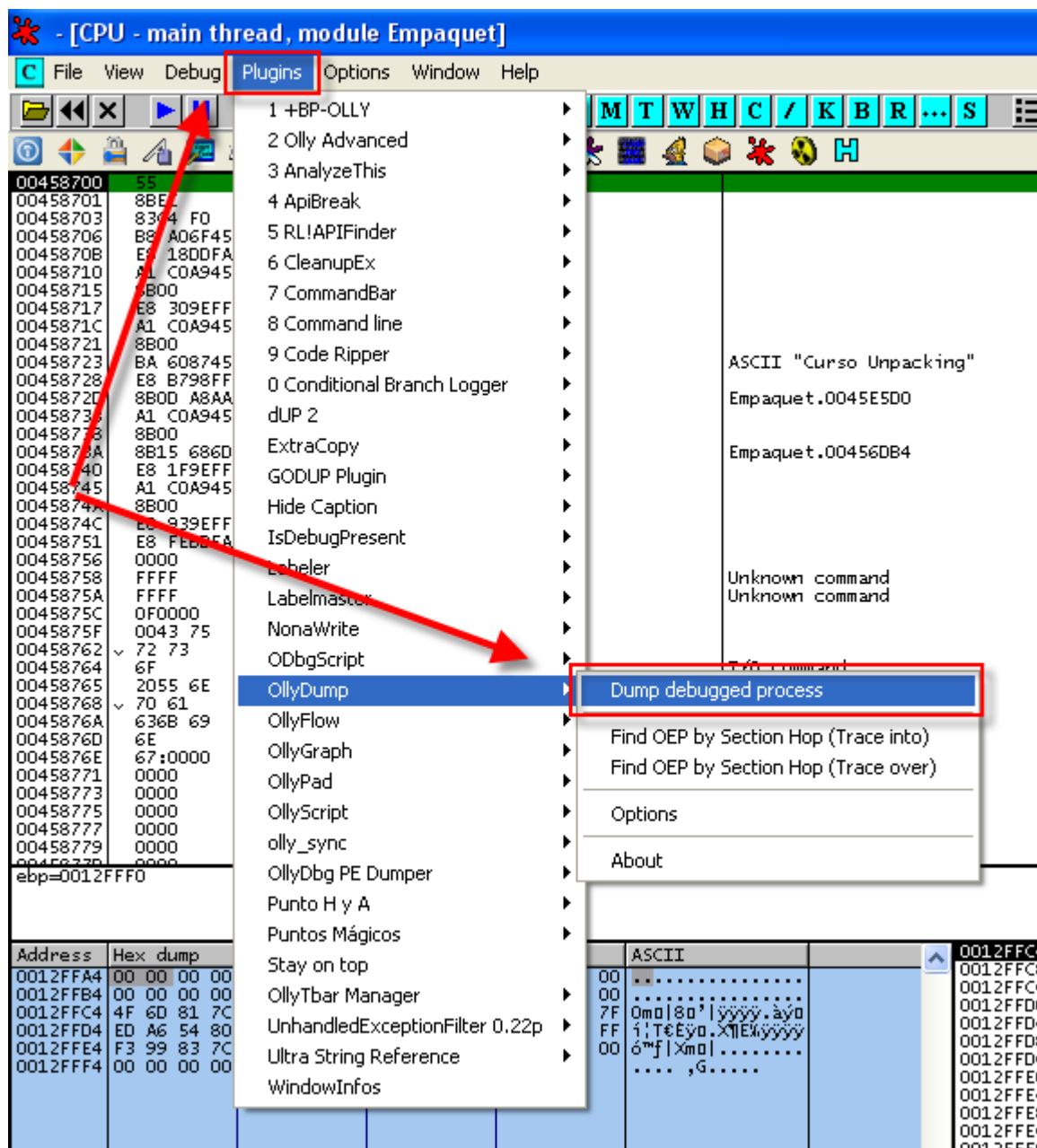
| | | | |
|----------|---------------|---------------------------|-------------|
| 00458700 | 55 | push ebp | |
| 00458701 | 8BEC | mov ebp,esp | |
| 00458703 | 83C4 F0 | add esp,-10 | |
| 00458706 | B8 406F4500 | mov eax,Empaquet.00456FA0 | |
| 00458708 | E8 180FAFF | call Empaquet.00406428 | |
| 00458710 | A1 C0A94500 | mov eax,ds:[45A9C0] | |
| 00458715 | 8B00 | mov eax,ds:[eax] | |
| 00458717 | E8 309EFFFF | call Empaquet.0045254C | |
| 0045871C | A1 C0A94500 | mov eax,ds:[45A9C0] | |
| 00458721 | 8B00 | mov eax,ds:[eax] | |
| 00458723 | BA 60874500 | mov edx,Empaquet.00458760 | |
| 00458728 | E8 B798FFFF | call Empaquet.00452564 | |
| 0045872D | 8B0D A8AA4500 | mov ecx,ds:[A8AA4500] | ASCII "Curs |
| 00458733 | A1 C0A94500 | mov ecx,ds:[45A9C0] | Empaquet.0C |
| 00458738 | 8B00 | mov ecx,ds:[eax] | Empaquet.0C |
| 0045873A | 8B15 686D4500 | mov edx,ds:[456D68] | |
| 00458740 | E8 1F9EFFFF | call Empaquet.00452564 | |
| 00458745 | A1 C0A94500 | mov eax,ds:[45A9C0] | |
| 0045874A | 8B00 | mov eax,ds:[eax] | |
| 0045874C | E8 939EFFFF | call Empaquet.004525E4 | |
| 00458751 | E8 FEBFAFF | call Empaquet.00404554 | |
| 00458756 | 0000 | add ds:[eax],al | |
| 00458758 | FFFF | ??? | Unknown cor |
| 0045875A | FFFF | ??? | Unknown cor |
| 0045875C | 0F0000 | sltd ds:[eax] | |

Lo primero que haremos será [dumpearlo](#) (volcarlo), ¿Por qué?, pues porque así ya tenemos guardado el programa descomprimido. Podemos hacerlo de 3 maneras.

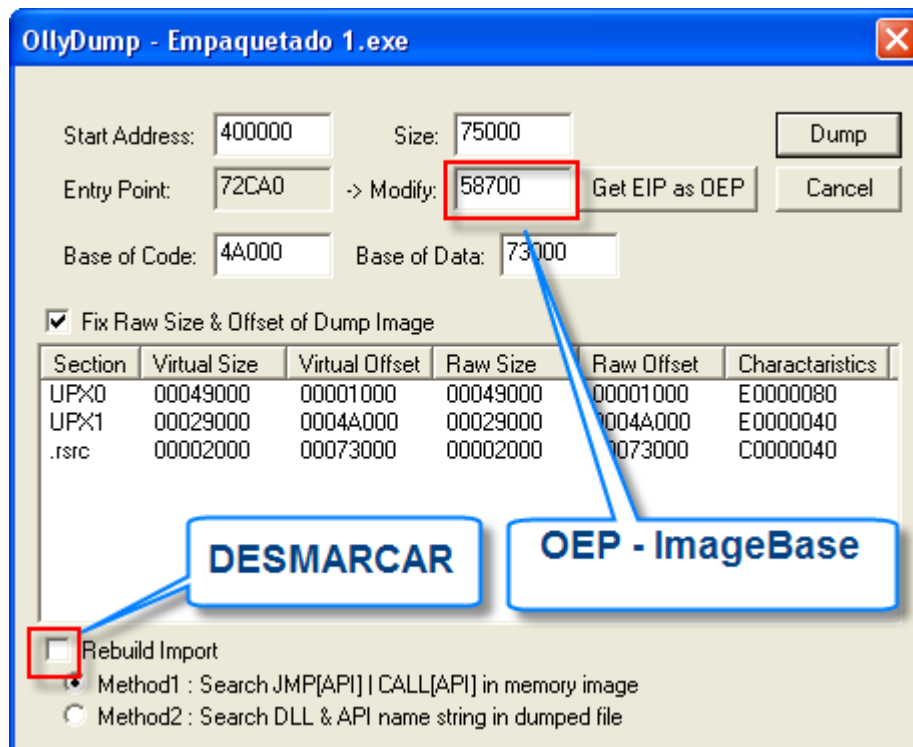
4.- DUMPEANDO

1ª – **OllyDump** (Tradicional, viene con mi [OllyDBG](#)).

En los plugins del [OllyDBG](#) encontramos esta maravillosa herramienta.



Una vez lo ejecutáis, veréis la ventana de dumpeado.



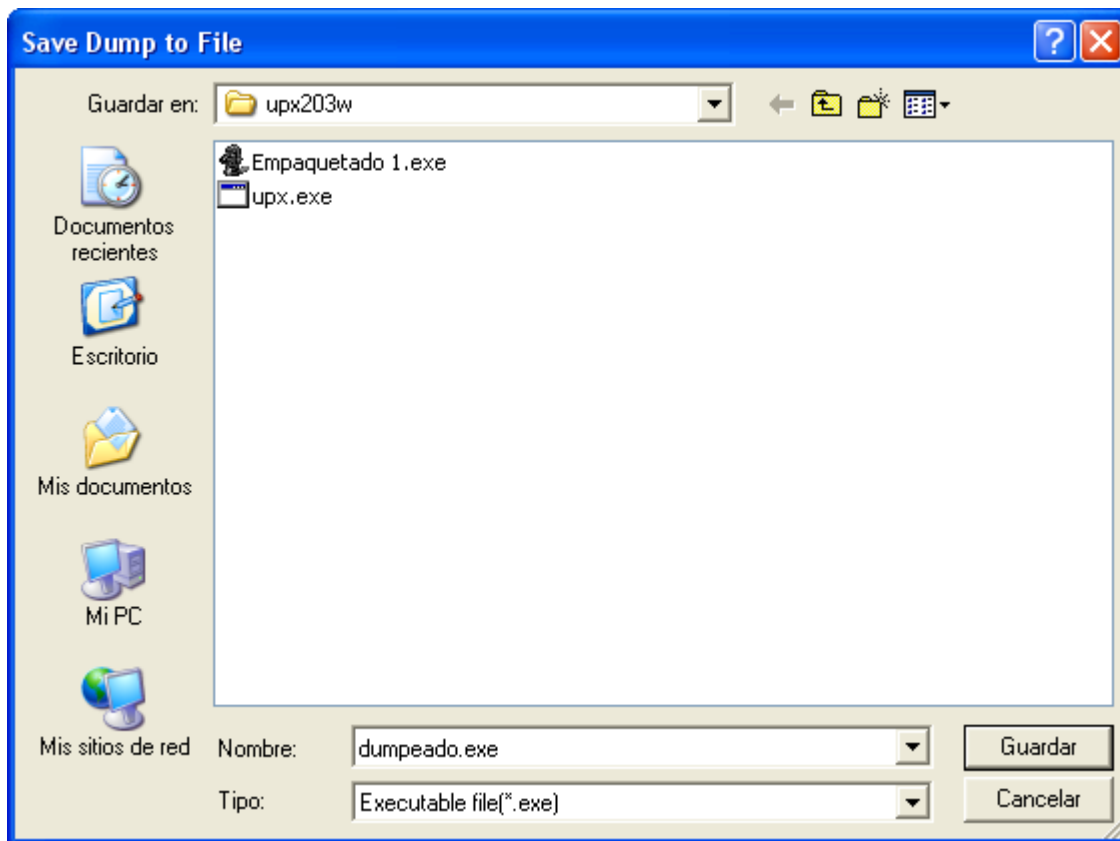
Donde pone **Entry Point** y luego **Modify**, estamos diciendole el nuevo EntryPoint, que es igual a la dirección en la que nos encontramos menos la ImageBase.

```
00458700  55  | push ebp
```

458700 es donde nos encontramos (OEP), sin embargo hay una propiedad de los ejecutables que se llama “Base de la Imagen (**ImageBase**)” que indica la dirección por donde comenzar, es decir, si le quitamos la base de la imagen tendríamos lo que conocemos como “**Offset**”, y si quisiésemos llegar hasta ahí desde un editor hexadecimal deberíamos quitarle la base de la imagen, pero en depuración se utiliza éste “número base” para asignar las direcciones.

Insisto en **DESMARCAR** la casilla de “**Rebuild Import**” porque nosotros la vamos a reconstruir así que no hace falta hacerla por ese método, que no siempre acierta.

Clickeamos a “**DUMP**” y guardamos el dumpeado.



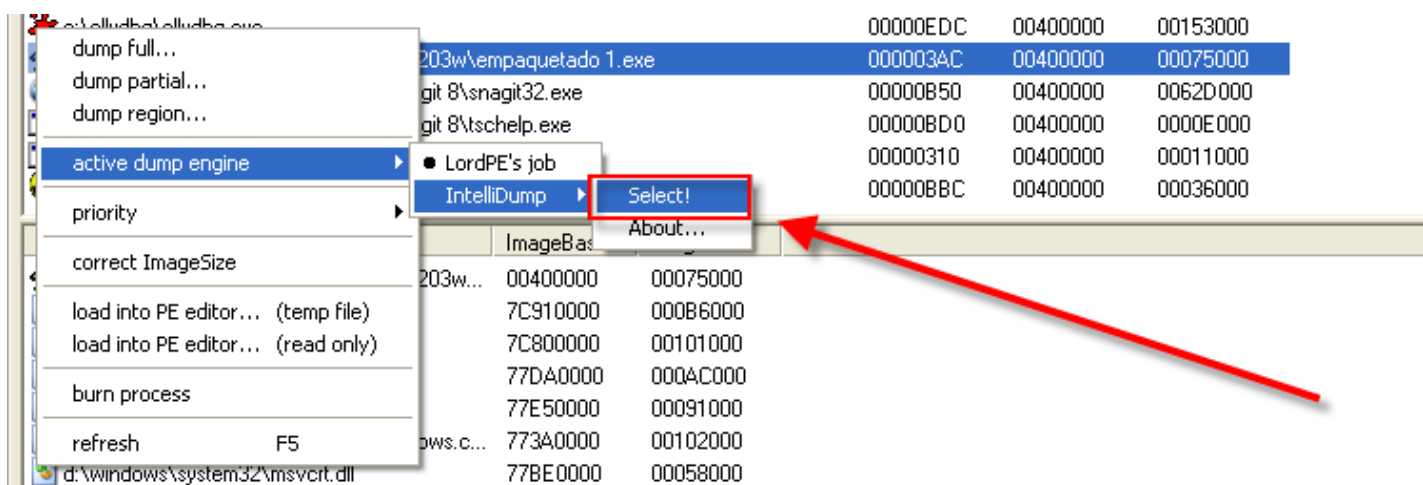
Y con esto tenemos el archivo volcado en “[dumpeado.exe](#)”.

2ª – [Lord PE](#) ([descarga](#)).

Aquí lo mismo abrimos el [Lord PE](#).

| Path | PID | ImageBase | ImageSize |
|--|----------|-----------|-----------|
| d:\archivos de programa\hp\digital imaging\bin\hpqtra08.exe | 00000724 | 00400000 | 0003F000 |
| d:\archivos de programa\kaspersky lab\kaspersky anti-virus 6.0\avp.exe | 00000364 | 00400000 | 00032000 |
| d:\windows\system32\svchost.exe | 0000036C | 01000000 | 00006000 |
| d:\windows\system32\svsmb32.exe | 00000354 | 00400000 | 0002C000 |
| d:\windows\system32\hpzipm12.exe | 0000052C | 00400000 | 00012000 |
| d:\windows\system32\svchost.exe | 00000718 | 01000000 | 00006000 |
| d:\windows\system32\wdfmgr.exe | 000006A8 | 01000000 | 0000C000 |
| d:\archivos de programa\vmware\vmware workstation\vmware-authd.exe | 000005D0 | 00400000 | 00038000 |
| d:\archivos de programa\archivos comunes\vmware\vmware virtual image editing\vm... | 00000758 | 00400000 | 00042000 |
| d:\windows\system32\vmnat.exe | 000007A8 | 00400000 | 00024000 |
| d:\windows\system32\vmnetdhcp.exe | 00000818 | 00400000 | 00024000 |
| d:\archivos de programa\pc connectivity solution\servicelayer.exe | 000009B4 | 00400000 | 0004C000 |
| d:\windows\system32\alg.exe | 00000E08 | 01000000 | 0000D000 |
| d:\archivos de programa\msn messenger\usnsvc.exe | 00000D48 | 00400000 | 00019000 |
| d:\windows\system32\wuauclt.exe | 00000F90 | 00400000 | 0001E000 |
| d:\archivos de programa\microsoft office\office12\outlook.exe | 00000264 | 30000000 | 00C3A000 |
| d:\archivos de programa\vmware\vmware workstation\vmware.exe | 00000544 | 00400000 | 00238000 |
| d:\archivos de programa\mozilla firefox\firefox.exe | 000006E4 | 00400000 | 0075E000 |
| d:\archivos de programa\vmware\vmware workstation\bin\vmware-vmx.exe | 00000FB0 | 00400000 | 00500000 |
| d:\archivos de programa\microsoft office\office12\winword.exe | 00000EF8 | 30000000 | 00057000 |
| c:\lollydbg\lollydbg.exe | 00000EDC | 00400000 | 00153000 |
| c:\herramientas cracking\patchers\upx203w\empaquetado 1.exe | 000003AC | 00400000 | 00075000 |
| c:\archivos de programa\techsmith\snagit 8\snagit32.exe | 00000B50 | 00400000 | 0062D000 |
| c:\archivos de programa\techsmith\snagit 8\tschelp.exe | 00000BD0 | 00400000 | 0000E000 |
| c:\archivos de programa\techsmith\snagit 8\snagpriv.exe | 00000310 | 00400000 | 00011000 |
| c:\herramientas cracking\lord pe\lordpe.exe | 00000BBC | 00400000 | 00036000 |

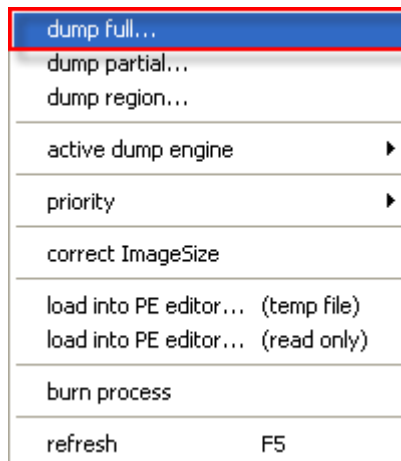
Y seleccionamos el “empaquetado 1.exe”, lo bueno que tiene Lord PE a diferencia de los demás es el **IntelliDump**. Para seleccionarlo seleccionamos el proceso, le damos al botón derecho y vamos a la opción.



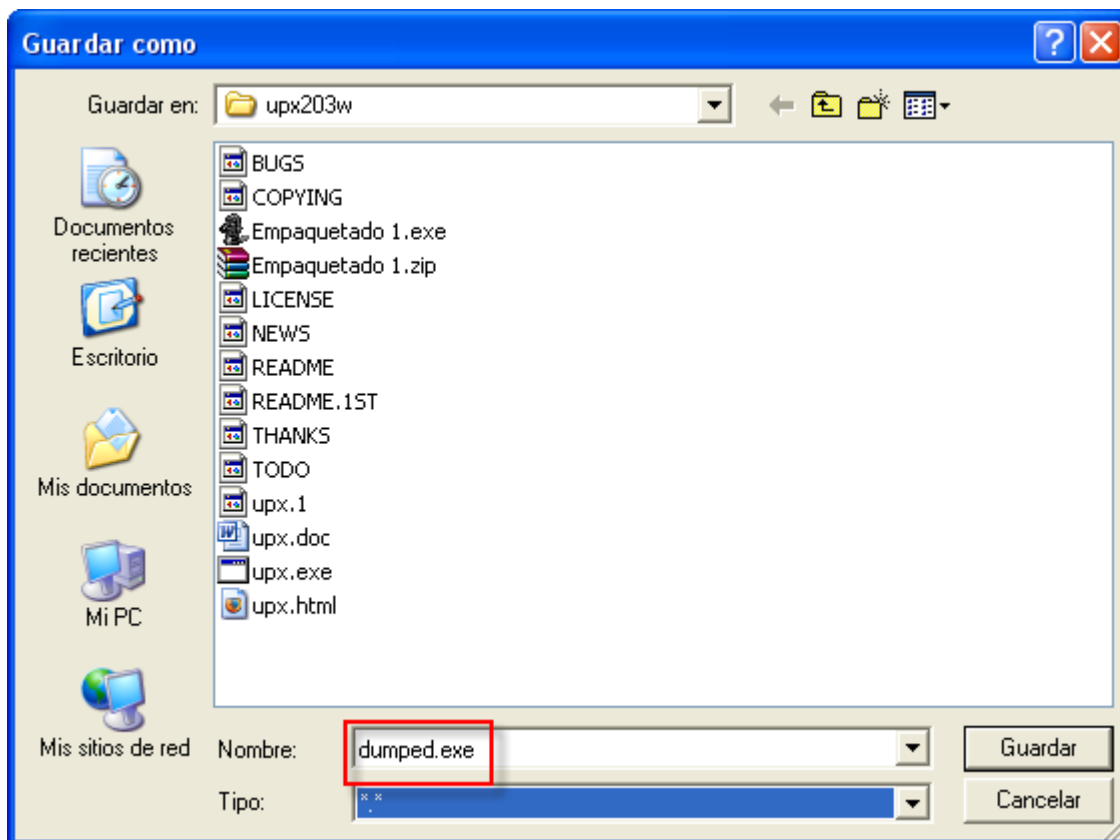
Realmente, aunque no sepáis exactamente que hace es mejor que os acostumbréis a utilizarlo, con packers más complicados. Ésta opción permite dumpear por regiones, y muchas veces nos salva de una, porque

dumpear con el plugin [OllyDump](#) o cualquier otro hace que no dumpeemos **TODAS** las secciones y por lo tanto quedan zonas sin volcar (que luego se pueden volcar a mano), pero te soluciona bastante. Repito, aunque no lo entendáis lo explico, así algún día tendréis la oportunidad de entenderlo ☺.

Una vez seleccionado le seleccionamos la opción “**DUMP FULL**”.



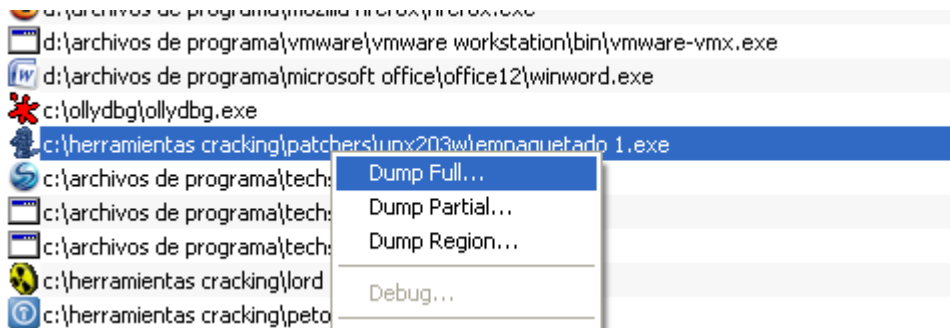
Y lo mismo.



Aquí también tendríamos el dumpeado.

3ª - PE Tools ([descarga](#)).

Es exactamente lo mismo que Lord PE (el motor es igual).



Y lo guardamos como “dumpeado.exe” o como queráis

Una vez tenemos el dumpeado guardado y listo. Es hora de arreglar la tabla con el [Import Reconstructor](#).

5.- REPARANDO LA IAT

Vamos a utilizar el programa que ya puse anteriormente, el [Import Reconstructor](#) éste lo que hace es buscar todas las llamadas a las API y apuntarlas a la tabla correctamente. Según la wikipedia inglesa extraje ésta explicación de la IAT.

The **IAT** is used as a lookup table when the application is calling a Windows API function. Because a compiled PE DLL/EXE cannot know in advance where the other DLLs it depends upon are located in memory, an indirect jump is required. As the dynamic linker loads modules and joins them together, it writes jump instructions into the IAT slots which point to the actual location of the destination function. Though this adds an extra jump over the cost of an intra-module call, the performance hit is mostly negligible and easily worth the flexibility of dynamic libraries. If the compiler knows ahead of time that a call will be inter-module (via a `dllimport` attribute) it can produce more optimised code that simply results in an indirect call opcode.

Bueno básicamente se refiere a que nos da las direcciones de las **API** reales en una tabla. Para poder reparar la tabla debemos de saber 3 datos:

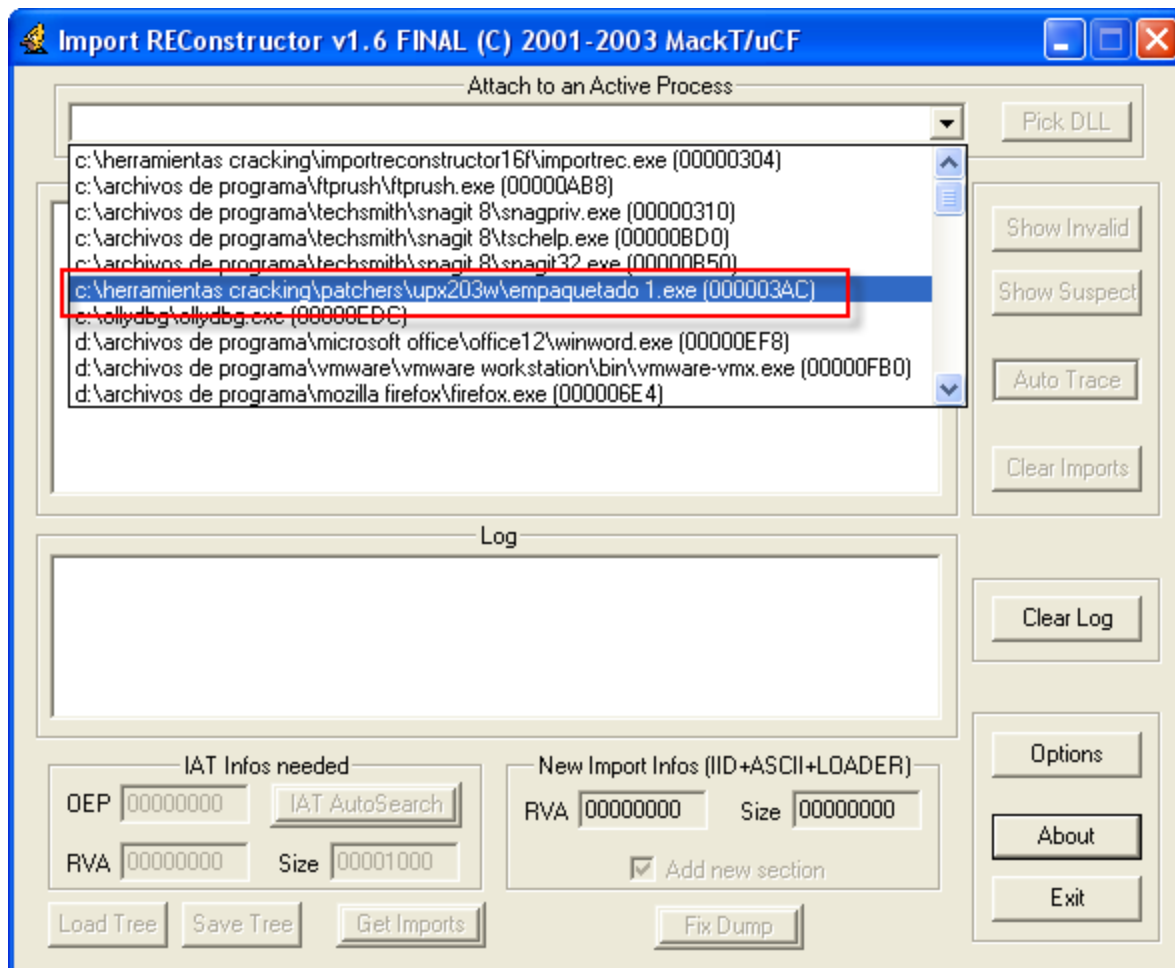
Offset del OEP – Original Entry Point (he hablado ya mucho sobre él) = 58700 (recordad que se le restaba 400000).

Inicio de la tabla.

Largo (tamaño) de la tabla.

He decidido no mostrar como localizar la tabla de importaciones, creo que se hace pesado éste tutorial para quienes no saben, y lo escribiré más adelante, ya que el mismo [Import Reconstructor](#) te lo resuelve automáticamente no lo mostraré en éste tutorial.

Abrimos el import reconstructor (con el programa parado en el OEP en OllyDBG).



Y seleccionamos ahí el proceso que queremos repararle la IAT. Una vez cargado vemos que nos pone una dirección de OEP (la que tenía anteriormente).



Y la debemos cambiar por la nueva (58700).



IAT Infos needed

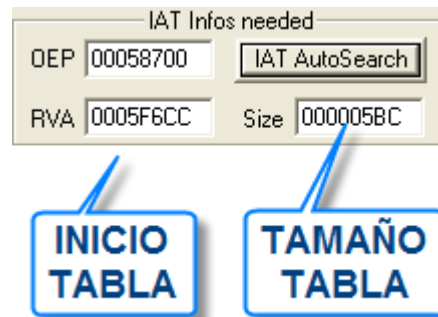
OEP: 00058700 IAT AutoSearch

RVA: 00000000 Size: 00001000

Y ahora le damos a **IAT AutoSearch** para que nos busque automáticamente la tabla de importaciones.



Dice que ha encontrado algo, que probemos a recibir las importaciones (**Get Imports**). Pero antes observad los valores que dije que pedirían.



IAT Infos needed

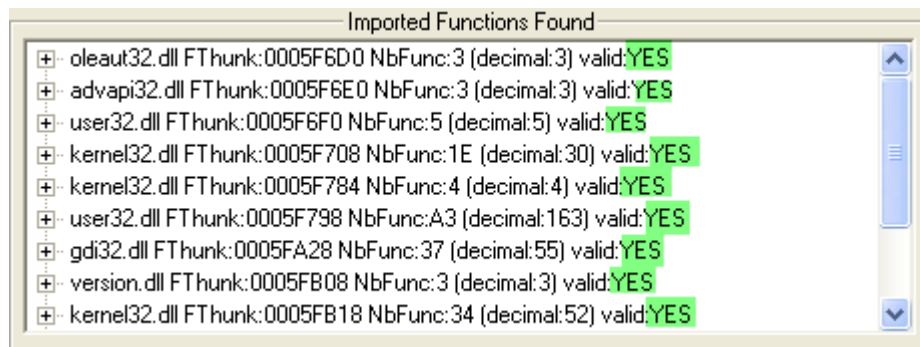
OEP: 00058700 IAT AutoSearch

RVA: 0005F6CC Size: 000005BC

INICIO TABLA

TAMAÑO TABLA

Así que el solito nos lo encontró, ahora démosle a **Get Imports** y recibiremos las direcciones.



Y si bajamos vemos que todas siguen correctas, cuando dice YES es que esta todo correcto. Ya está todo listo para reparar el dumpeado (**Fix Dump**). Importante, aunque siempre lo pone, no olvidarse.

New Import Infos (IID+ASCII+LOADER)

RVA 00000000 Size 0000193E

☒ Add new section

Tener marcada la casilla para poner la tabla de importaciones en una nueva sección. Y ya darle al botón de [Fix Dump](#), para arreglar nuestro dumpeado.

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

Attach to an Active Process

c:\herramientas\cracking\patchers\upx203w\empaquetado 1.exe (000003AC) Pick DLL

Imported Functions Found

- + kernel32.dll FTunk:0005F784 NbFunc: 4 (decimal:4) valid:YES
- + user32.dll FTunk:0005F798 NbFunc:A3 (decimal:163) valid:YES
- + gdi32.dll FTunk:0005FA28 NbFunc:37 (decimal:55) valid:YES
- + version.dll FTunk:0005FB08 NbFunc:3 (decimal:3) valid:YES
- + kernel32.dll FTunk:0005FB18 NbFunc:34 (decimal:52) valid:YES
- + advapi32.dll FTunk:0005FBEC NbFunc:4 (decimal:4) valid:YES
- + kernel32.dll FTunk:0005FC00 NbFunc:1 (decimal:1) valid:YES
- + oleaut32.dll FTunk:0005FC08 NbFunc:8 (decimal:8) valid:YES
- + comctl32.dll FTunk:0005FC2C NbFunc:16 (decimal:22) valid:YES

Show Invalid Show Suspect Auto Trace Clear Imports

Log

rva:0005FBCC forwarded from mod:ntdll.dll ord:0098 name:RtlEnterCriticalSection
rva:0005FBD0 forwarded from mod:ntdll.dll ord:0097 name:RtlDeleteCriticalSection

Current imports:
D (decimal:13) valid module(s) (added: +D (decimal:+13))
161 (decimal:353) imported function(s). (added: +161 (decimal:+353))

Clear Log

IAT Infos needed

OEP 00058700 IAT AutoSearch

RVA 0005F6CC Size 000005BC

New Import Infos (IID+ASCII+LOADER)

RVA 00000000 Size 0000193E

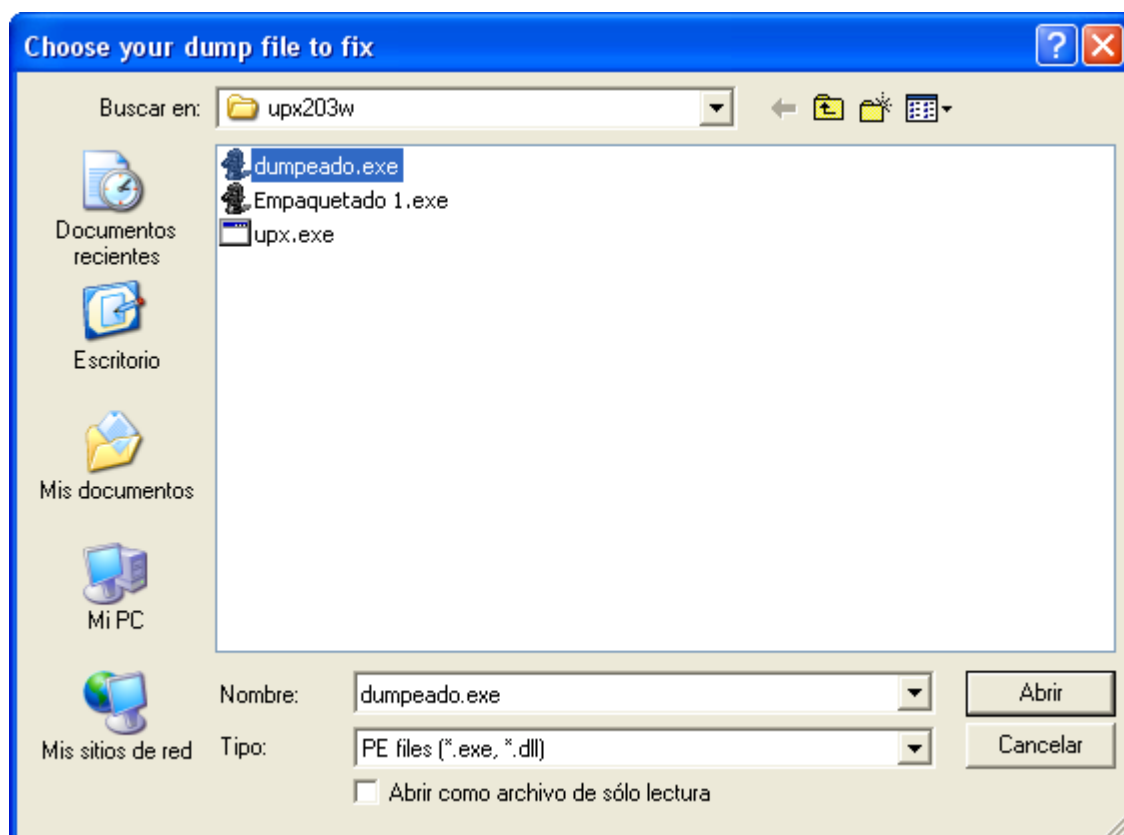
☒ Add new section

Load Tree Save Tree Get Imports

Fix Dump

Options About Exit

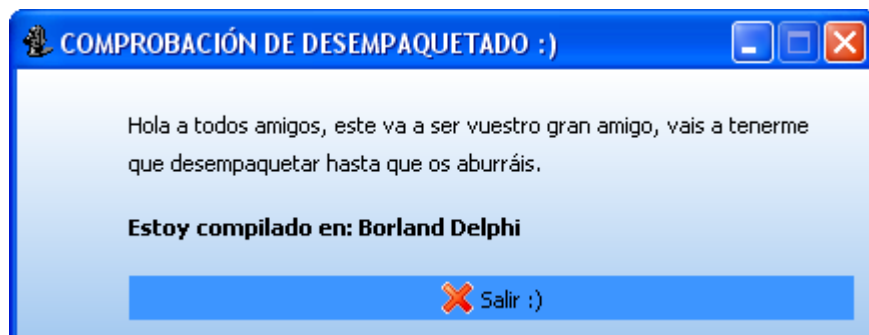
Y seleccionamos el dumpeado que hicimos.



Y como comprobaremos el LOG no falla.

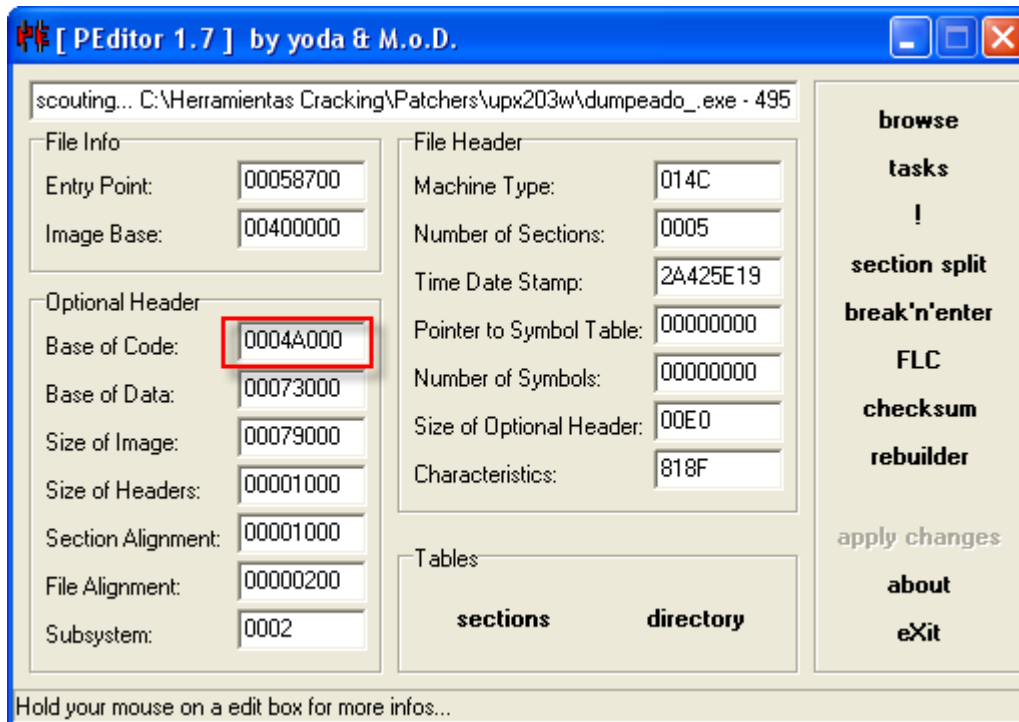
```
Fixing a dumped file...
D (decimal:13) module(s)
161 (decimal:353) imported function(s).
*** New section added successfully. RVA:00077000 SIZE:00002000
Image Import Descriptor size: 104; Total length: 193E
C:\Herramientas Cracking\Patchers\upx203w\dumpeado_.exe saved successfully.
```

Nos dice que está perfesto. Lo ejecutamos a ver y...



PERFECTO.

Por último nos queda restaura el Base of Code, normalmente es lo único que no se porque jeje, no se llega a poner a su valor original, así que abrimos el [PE Editor](#) (se puede hacer con Lord PE o PE Tools). Arrastramos el desempaketado hacia el [PE Editor](#).



Ahí vemos la base del código. Simplemente vamos a "[sections](#)" y vemos la dirección por la que empieza la primera sección.

| Section | Virtual Size | Virtual Offset | Raw Size | Raw Offset | Characteristics |
|---------|--------------|----------------|----------|------------|-----------------|
| UPX0 | 00049000 | 00001000 | 00049000 | 00001000 | E0000080 |
| UPX1 | 00029000 | 0004A000 | 00029000 | 0004A000 | E0000040 |
| .rsrc | 00002000 | 00073000 | 00002000 | 00073000 | C0000040 |
| .idata2 | 00002000 | 00075000 | 00001A00 | 00075000 | C0000040 |
| .mact | 00002000 | 00077000 | 00002000 | 00077000 | E0000060 |

Y entonces cambiamos el [base of code](#) y aplicamos cambios.

| Optional Header | |
|--------------------|----------|
| Base of Code: | 00001000 |
| Base of Data: | 00073000 |
| Size of Image: | 00079000 |
| Size of Headers: | 00001000 |
| Section Alignment: | 00001000 |
| File Alignment: | 00000200 |
| Subsystem: | 0002 |

Y listo. Ya esta completamente desempaquetado, abría que cambiar los nombres de las secciones y dejarlo como al principio pero por hoy ya vale jajaja.

ESPERO QUE HAYÁIS APRENDIDO ALGO. Cualquier duda, reclamación, insulto, etc... a n0debytes@yahoo.es.

Hasta el próximo tute ☺. Un saludo a todo www.indetectables.net.

// Shaddy // ESPAÑA 23 – JUNIO – 2007 //