FORMATO PORTABLE EXECUTABLE BAJO WINDOWS®

Edición en español por The Swash

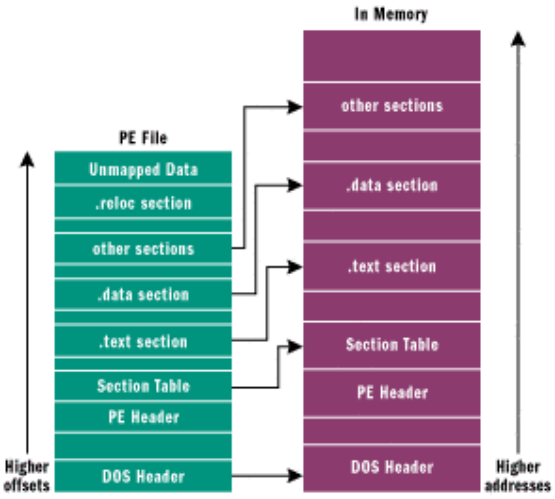
TODOS LOS DERECHOS RESERVADOS - 2011

CONTENIDO

ESTRUCTURA DE UN FICHERO EJECUTABLE

ELEMENTO	DESCRIPCIÓN
DOS HEADER	Cabecera de archivos DOS.
DOS STUB	Mensaje en sistemas DOS.
NT HEADER	Inicio del archivo ejecutable.
• PE Signatura	Signatura PE\x0\x0
FILE HEADER	Información base del archivo.
OPTIONAL HEADER	Información adicional (Necesaria en ejecutables).
DATA DIRECTORY	IMAGE DATA DIRECTORY
SECTION HEADER	Información de las secciones del ejecutable
CUERPO DEL ARCHIVO	Contenido de cada sección.
END OF FILE	Final del archivo

Los archivos ejecutables originalmente nombrados Portable Executable (A partir de ahora llamados con la abreviatura PE) son los denominados programas en Windows. La idea base de estos archivos fue formarlos de manera estructurada para mayor flexibilidad. El archivo se divide en cabeceras y cuerpo. Las cabeceras proveen información al loader de Windows para poder ejecutarlos. Posterior a estas cabeceras se encuentra el cuerpo del archivo la cual tendrá el contenido de cada una de las secciones.



DOS HEADER:

Cabecera obsoleta incluida en los archivos tanto DOS como PE, en los PE tiene solo 2 campos útiles:

- **e_magic**: Este campo contiene las siglas 'MZ' que todo archivo ejecutable debe tener, estás siglas son en honor a Mark Zbikowsky uno de los principales diseñadores de MS-DOS. Si un archivo ejecutable no tiene estas siglas no será ejecutado, podemos decir que actúa como una firma de validación.

- **l_fanew**: Dirección física o puntero hacia el NT HEADER el cual inicia por la firma PE\x0\x0. El valor de este campo suele encontrarse en el desplazamiento 0x3C.

DOS STUB:

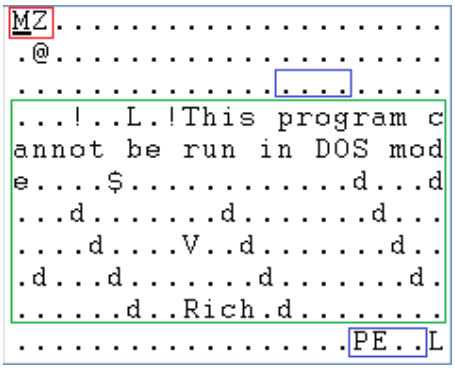
El DOS STUB es una aplicación válida para MS-DOS la cual imprimirá en pantalla el mensaje "This program cannot be run in DOS mode".

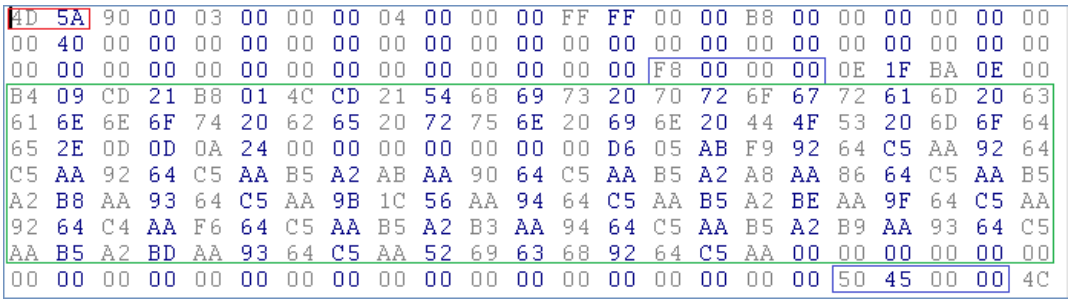
```
C:\UFO>UFO-SFX.EXE
This program cannot be run in DOS mode.
```

NT HEADER:

La cabecera NT inicia en el valor apuntado por el campo l_fanew, justo en el inicio nos encontramos con una firma que nos ubica y es la cadena formada por los caracteres PE\x0\x0 y posterior a este ya podemos encontrar la estructura del IMAGE_FILE_HEADER.

El NT HEADER está presente tanto en archivos ejecutables como librerías de enlace dinámico. En caso de no encontrar la firma PE\x0\x0 el archivo no abría de ejecutarse y lo mismo pasa con la firma MZ.





DIRECCIONES FÍSICAS, VIRTUALES RELATIVAS Y VIRTUALES

Es muy importante relacionarnos con estos 3 conceptos los cuales son necesarios para entender muchos campos de la estructura de un archivo ejecutable.

DIRECCIÓN FÍSICA O DESPLAZAMIENTO (OFFSET):

Corresponde a un valor que se encuentra en el rango del tamaño físico del archivo, no podrá haber desplazamiento menor a 0 o mayor al tamaño del archivo.

DIRECCIÓN VIRTUAL RELATIVA:

Como un archivo en memoria es distinto a un archivo en disco las direcciones se manejan de manera distinta, cada sección establece una Dirección virtual y desde ahí se copiará el contenido de cada sección. Para obtener una direccion Virtual relativa o abreviadamente RVA de un Offset debemos recurrir al siguiente proceso:

$$RVA = (Offset - PointerToRawData) + VirtualAddress$$

Para hacer lo contrario, es decir obtener un Offset a partir de un RVA:

$$Offset = (RVA - VirtualAddress) + PointerToRawData$$

Hay que tener en cuenta que los valores:

- VirtualAddress
- PointerToRawData

Son valores pertenecientes a la sección en la cual se encuentre dicho valor, pero como sabemos ¿a qué sección pertenece?

Sencillamente buscaremos en el rango de las secciones, si buscamos un RVA deberemos buscar en los rangos VirtualAddress – VirtualSize y si buscamos un Offset buscamos en los rangos PointerToRawData – SizeOfRawData.

DIRECCIONES VIRTUALES:

Las direcciones virtuales son aquellas direcciones que hacen referencia directa a la posición del archivo cargado en memoria, en el caso vendrían siendo la base donde se ha cargado en memoria que puede ser el valor proporcionado por el campo ImageBase (IMAGE OPTIONAL HEADER) o que puede variar ya que el loader de Windows en caso de que esta dirección esté ocupada reasignará otra y reconstruirá el modulo del ejecutable ahí con información del reloc directory.

Entonces con esto entendemos que una dirección virtual se diferenciará o no por la presencia del ImageBase, si lo quitamos pasa a ser un RVA y si queremos convertir un Offset a dirección virtual deberemos pasarlo a RVA y sumar el ImageBase.

IMAGE FILE HADER

Esta cabecera contiene información general acerca del archivo entre sus campos tenemos:

CAMPO	TAMAÑO
Machine	WORD
NumberOfSections	WORD
TimeDateStamp	DWORD
PointerToSymbolTable	DWORD
NumberOfSymbols	DWORD
SizeOfOptionalHeader	DWORD
Characteristics	DWORD

Machine: Este campo define el tipo de arquitectura del computador u emulador en el cual se podrá ejecutar el programa. Este puede ser:

CONSTANTE	VALOR	EQUIVALENCIA
IMAGE_FILE_MACHINE_I386	0x014C	X86
IMAGE_FILE_MACHINE_IA63	0x0200	Intel Itanium
IMAGE_FILE_MACHINE_AMD64	0x8664	X64

NumberOfSections: Este campo tiene un tamaño de 2 bytes y contiene el número de secciones que tienen el ejecutable sin exceptuar ninguna teniendo en cuenta que el límite que impone el loader es de 96 secciones o por lo menos solo esa cantidad como máximo cargará.

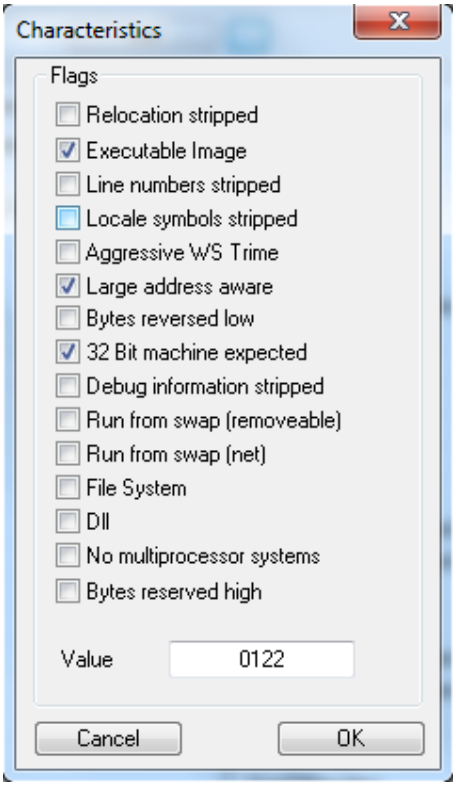
Address	Hex dump	Data	Comment
00400086	0400	DW 0004	NumberOfSections = 4

TimeDateStamp: Este campo tiene un tamaño de 4 bytes e indica el número de segundos a partir de 1 de Enero de 1970. Estos segundos nos indicarán la fecha de creación del archivo.

PointerToSymbolTable & NumberOfSymbols: Estos campos son empleados por los archivos .obj o COFF FILES, en los archivos ejecutables predeterminadamente estarán en valor 0.

SizeOfOptionalHeader: Este campo tiene un tamaño de 2 bytes y contendrá el valor del tamaño del IMAGE OPTIONAL HEADER.

Characteristics: Este campo indica los distintos atributos del archivo mediante distintos valores. Cuando el archivo presenta varias características el valor final se obtiene mediante la suma de cada característica:



Valores de las características:

CONSTANTE	VALOR
IMAGE_FILE_RELOCS_STRIPPED	0x0001
IMAGE_FILE_EXECUTABLE_IMAGE	0x0002
IMAGE_FILE_LINE_NUMS_STRIPPED	0x0004
IMAGE_FILE_LOCAL_SYMS_STRIPPED	0x0008
IMAGE_FILE_AGGRESSIVE_WS_TRIM	0x0010
IMAGE_FILE_LARGE_ADDRESS_AWARE	0x0020
Unknown	0x0040
IMAGE_FILE_BYTES_REVERSED_LO	0x0080
IMAGE_FILE_32BIT_MACHINE	0x0100
IMAGE_FILE_DEBUG_STRIPPED	0x0200
IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP	0x040
IMAGE_FILE_NET_RUN_FROM_SWAP	0x080
IMAGE_FILE_SYSTEM	0x100
IMAGE_FILE_DLL	0x200

Con esto terminamos la parte del IMAGE FILE HEADER y continuaremos con el IMAGE OPTIONAL HEADER.

IMAGE OPTIONAL HEADER

CAMPO	TAMAÑO
Magic	WORD
MajorLinkerVersion	BYTE
MinorLinkerVersion	BYTE
SizeOfCode	DWORD
SizeOfInitializedData	DWORD
SizeOfUninitializedData	DWORD
AddressOfEntryPoint	DWORD
BaseOfCode	DWORD
BaseOfData	DWORD
ImageBase	DWORD
SectionAlignment	DWORD
FileAlignment	DWORD
MajorOperatingSystemVersion	WORD
MinorOperatingSystemVersion	WORD
MajorImageVersion	WORD
MinorImageVersion	WORD
MajorSubsystemVersion	WORD
MinorSubsystemVersion	WORD
Win32VersionValue	DWORD
SizeOfImage	DWORD
SizeOfHeaders	DWORD
CheckSum	DWORD
Subsystem	WORD
DllCharacteristics	WORD
SizeOfStackReserve	DWORD
SizeOfStackCommit	DWORD
SizeOfHeapReserve	DWORD
SizeOfHeapCommit	DWORD
LoaderFlags	DWORD
NumberOfRvaAndSizes	DWORD

El IMAGE OPTIONAL HEADER proporciona información adicional al loader para que cargue correctamente el ejecutable. Esta información en los archivos COFF es opcional pero en los archivos ejecutables es obligatoria. El tamaño de esta estructura no es fijo, lo define el campo SizeOfOptionalHeader (IMAGE FILE HEADER).
Miremos cada uno de los campos de esta estructura:

Magic: (No confundir con e_magic).
Este campo tiene un tamaño de 2 bytes y determina si el ejecutable es para sistemas x86 o x64. Las constantes que lo determinan son:

Magic number	PE format
0x10b	PE32 (x86)
0x20b	PE32+ (x64)

MajorLinkerVersion:

Este campo tiene un tamaño de 2 bytes y proporciona la versión más alta del enlazador.

MinorLinkerVersion:

Este campo tiene un tamaño de 2 bytes y proporciona la versión más baja del enlazador.

SizeOfCode, SizeOfInitializedData & SizeOfUninitializedData:

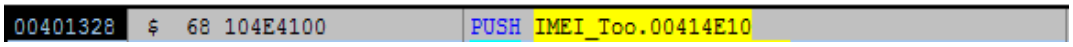
Estos tres campos se ven relacionados en cuanto a cómo obtener su valor y el único cambio es la procedencia de cada uno. Al igual que el ejecutable, sus secciones también presentan características y para relacionarlas con estos tres campos las características serían:

CONSTANTE	VALOR
IMAGE_SCN_CNT_CODE	0x00000020
IMAGE_SCN_CNT_INITIALIZED_DATA	0x00000040
IMAGE_SCN_CNT_UNINITIALIZED_DATA	0x00000080

Estos campos son acumuladores de la suma del SizeOfRawData de las secciones que presenten su característica. Por ejemplo tengo 2 secciones que tienen IMAGE_SCN_CNT_CODE entonces el campo SizeOfCode tendrá como valor la suma del SizeOfRawData de las 2 secciones.

AddressOfEntryPoint:

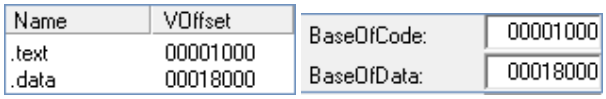
Este campo tiene un tamaño de 4 bytes y nos proporciona la dirección virtual relativa (RVA) hacia la primera instrucción que ejecutará el programa, en conclusión es el punto de partida de ejecución del programa.



Como vemos en memoria tiene el mismo valor sumado al ImageBase que es 0x00400000.

BaseOfCode & BaseOfData:

Estos dos campos proporcionan un RVA el cual indica o corresponde al inicio de las secciones de código y datos respectivamente.



ImageBase:

Este campo tiene un tamaño de 4 bytes y proporciona la dirección de preferencia donde se cargará el ejecutable. Este campo debe ser múltiplo de 0x10000. Es muy común ver en DLL este campo con valor 0x10000000 y en ejecutables 0x00400000.

00400000	0000100	IMEI_Too 00400000 (itself)		PE header

SectionAlignment:

Este campo tiene un tamaño de 4 bytes y nos proporciona un valor el cual será el alineamiento en bytes de las secciones del ejecutable en memoria. Este valor debe ser mayor o igual al FileAlignment y por lo general equivale al valor de una página en memoria: 4096 bytes.

Ejemplo de alineación:

Necesito alinear el VirtualAddress de la sección posterior a la siguiente:

Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	00016ED8	00001000	00017000	60000020

El siguiente VirtualAddress deberá ser igual al VirtualAddress + VirtualSize Alineado al SectionAlignment equivalente a 0x1000, entonces no podremos dejar el valor 0x17ED8 si no que alineamos y tendremos 0x18000.

FileAlignment:

Este campo tiene un tamaño de 4 bytes y nos proporciona un valor de alineamiento para los datos físicos de las secciones. Es un requisito que una sección física tenga como tamaño mínimo este valor y cualquier valor superior deberá ser alineado al mismo. Este valor puede ser potencia de 2 y su valor estará entre 0x200 y 0x10000.

Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	00016ED8	00001000	00017000	60000020
.data	00018000	00000AAC	00018000	00001000	C0000040
.rsrc	00019000	00000934	00019000	00001000	40000040

Como vemos aquí todos los PointerToRawData y SizeOfRawData están alineados al FileAlignment que equivale a 0x200.

MajorOperatingSystemVersion:

Este campo tiene un tamaño de 2 bytes y nos proporciona la versión principal del sistema operativo requerido.

MinorOperatingSystemVersion:

Este campo tiene un tamaño de 2 bytes y nos proporciona la versión mínima del sistema operativo requerido.

MajorImageVersion:

Este campo tiene un tamaño de 2 bytes y nos proporciona la versión principal del ejecutable.

MinorImageVersion:

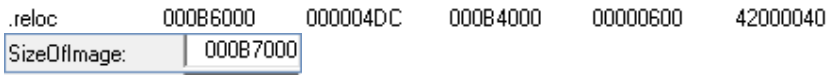
Este campo tiene un tamaño de 2 bytes y nos proporciona la versión menor del ejecutable.

MajorSubsystemVersion:
Este campo tiene un tamaño de 2 bytes y nos proporcióna la versión principal del subsistema requerida.

MinorSubsystemVersion:
Este campo tiene un tamaño de 2 bytes y nos proporcióna la versión menor del subsistema requerida.

Win32VersionValue:
Reservado por el sistema.

SizeOfImage:
Este campo tiene un tamaño de 4 bytes y proporciona el tamaño a reservar en memoria para cargar el ejecutable. Se puede obtener sumando el VirtualAddress de la última sección + VirtualSize redondeado a múltiplos del SectionAlignment.



SizeOfHeaders:
Es el tamaño de todas las cabeceras juntas:

- DOS HEADER
- DOS STUB
- FILE HEADER
- OPTIONAL HEADER
- DATA DIRECTORY
- SECTION HEADER

Redondeado al valor del FileAlignment.

CheckSum:
Este es un campo tiene un tamaño de 4 bytes y nos proporciona el resultado de un algoritmo de suma de comprobación del archivo proporcionado por la función CheckSumMappedFile o MapFileAndCheckSum de la librería Imagehlp.dll

Subsystem:
Este campo tiene un tamaño de 2 bytes y nos dice el subsistema requerido para ejecutar el programa, por ejemplo bajo consola, interface grafica, etc. Los valores definidos son:

CONSTANTE	VALOR
IMAGE_SUBSYSTEM_UNKNOWN	0
IMAGE_SUBSYSTEM_NATIVE	1
IMAGE_SUBSYSTEM_WINDOWS_GUI	2
IMAGE_SUBSYSTEM_WINDOWS_CUI	3
IMAGE_SUBSYSTEM_OS2_CUI	5

IMAGE_SUBSYSTEM_POSIX_CUI	7
IMAGE_SUBSYSTEM_WINDOWS_CE_GUI	9
IMAGE_SUBSYSTEM_EFI_APPLICATION	10
IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER	11
IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER	12
IMAGE_SUBSYSTEM_EFI_ROM	13
IMAGE_SUBSYSTEM_XBOX	14
IMAGE_SUBSYSTEM_WINDOWS_BOOT_APPLICATION	16

DllCharacteristics:

Este campo tiene un tamaño de 2 bytes y en caso de que la aplicación sea una DLL, este campo presentará las características y al igual que las características de las secciones o del ejecutable cuando hay más de una se suman los valores de todas las empleadas.

CONSTANTE	VALOR
-	0x0001
-	0x0002
-	0x0004
-	0x0008
IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE	0x0040
IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY	0x0080
IMAGE_DLLCHARACTERISTICS_NX_COMPAT	0x0100
IMAGE_DLLCHARACTERISTICS_NO_ISOLATION	0x0200
IMAGE_DLLCHARACTERISTICS_NO_SEH	0x0400
IMAGE_DLLCHARACTERISTICS_NO_BIND	0x0800
-	0x1000
IMAGE_DLLCHARACTERISTICS_WDM_DRIVER	0x2000
-	0x4000
IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER_AWARE	0x8000

SizeOfStackReserve:

Este campo tiene un tamaño de 4 bytes y contiene el número de bytes a reservar para la pila (Stack). Solo el valor del SizeOfStackCommit se ve comprometido el resto se encuentra disponible una página a la vez hasta que el tamaño reservado sea alcanzado.

SizeOfStackCommit:

Este campo tiene un tamaño de 4 bytes y es la parte de la memoria reservada para la pila que se verá comprometida.

SizeOfHeapReserve:

Heap o montículo es una región de memoria que se utilizará para satisfacer las peticiones de asignación de memoria. Este campo tiene un tamaño de 4 bytes y su valor expresa el tamaño a reservar para el montículo local (región de memoria en el espacio de direcciones de la aplicación).

SizeOfHeapCommit:
Este campo tiene un tamaño de 4 bytes y su valor expresa el tamaño de la memoria del montículo a comprometer.

LoaderFlags:
Este campo tiene un tamaño de 4 bytes y es obsoleto actualmente.
NumberOfRvaAndSizes:
Este campo tiene un tamaño de 4 bytes y su valor proporciona la cantidad de entradas del DATA DIRECTORY. A base de este valor podríamos obtener fácilmente el tamaño del DATA DIRECTORY operando así:

$$SizeOfDataDirectory = NumberOfRvaAndSizes * 8$$

Este es el último campo del OPTIONAL HEADER ya con esto hemos finalizado esta cabecera.

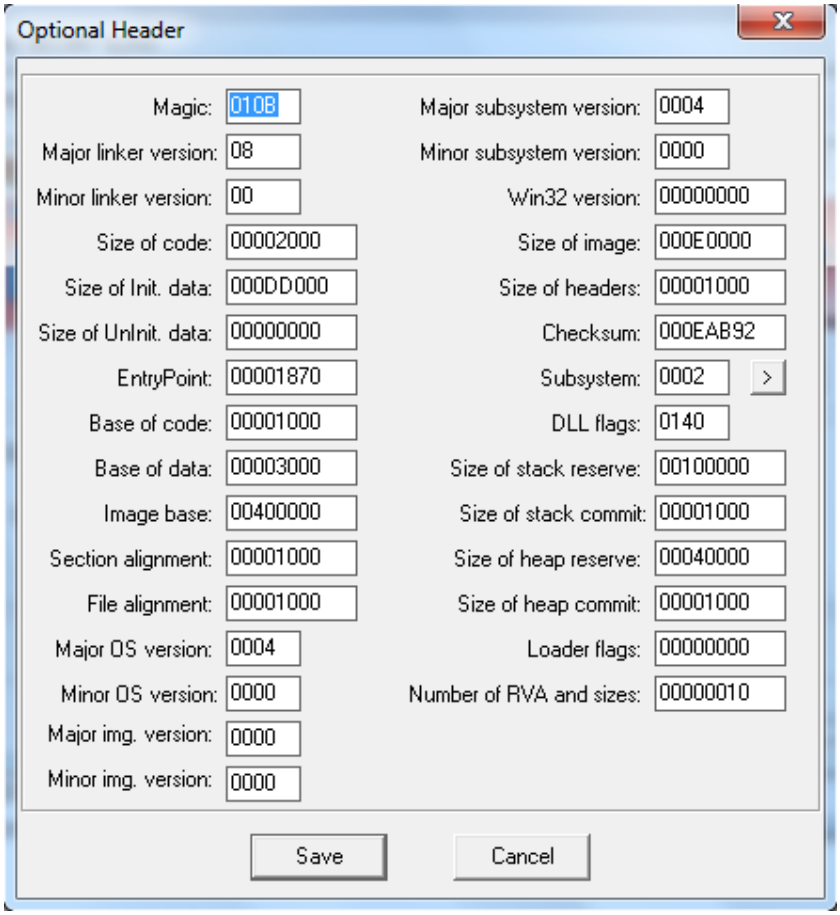


IMAGE DATA DIRECTORY

CAMPO	TAMAÑO
VirtualAddress	DWORD
Size	DWORD

Esta cabecera se encuentra ubicada justo después del IMAGE OPTIONAL HEADER y podemos decir que es una matriz de NumberOfRvaAndSizes cantidad de entradas, en sí su estructura es muy pequeña. Está compuesto por 2 valores cada 1 de 4 bytes, su estructura presenta 2 campos y son:

VirtualAddress:
Este campo tiene un tamaño de 4 bytes y su valor es una dirección virtual relativa de donde encontraremos el directorio actual en el cuerpo del archivo.

Size:
Este campo tiene un tamaño de 4 bytes y su valor expresa el tamaño del bloque del IMAGE DATA DIRECTORY apuntado por el VirtualAddress.

Estos campos se repiten para la cantidad de entradas proporcionadas por el NumberOfRvaAndSizes y son seguidas sin ningún tipo de separador. Este número generalmente suele ser 0x10.

Directory Name	RVA	Size
Export Table	00000000	00000000
Import Table	00003794	000000C8
Resource Table	00005000	000D9848
Exception Table	00000000	00000000
Certificate Table	000E0000	00001BD8
Relocation Table	000DF000	00000234
Debug Data	00003180	0000001C
Architecture	00000000	00000000
GlobalPtr	00000000	00000000
TlsTable	00000000	00000000
LoadConfig	00003670	00000040
Bound Import	00000000	00000000
IAT	00003000	00000164
Delay Import	00000000	00000000
CLR Runtime Header	00000000	00000000
Reserverd	00000000	00000000

Cada una de estas entradas obligatoriamente apuntará a un lugar del archivo el cual ocupara un bloque cuyo tamaño es proporcionado por el valor del campo Size. Estos bloques a su vez pueden dividirse en más bloques y cada entrada tiene su estructura.

ENTRADAS:

SIZE (bytes)	ENTRY
8	Export table
8	Import table
8	Resource table
8	Exception table
8	Certificate table
8	Base relocation table
8	Debugging information starting
8	Architecture-specific data
8	Global pointer register
8	Thread local storage (TLS)
8	Load configuration table
8	Bound import table
8	Import address table
8	Delay import descriptor
8	The CLR header
8	Reserved

Una vez terminado el último campo de esta serie de entradas sigue el IMAGE SECTION HEADER. Creo que aquí finalizo, después de los encabezados de sección explicaré la mayoría de estos campos.

IMAGE SECTION HEADER

CAMPO	TAMAÑO
Name	BYTE * 8
VirtualSize	DWORD
VirtualAddress	DWORD
SizeOfRawData	DWORD
PointerToRawData	DWORD
PointerToRelocations	DWORD
PointerToLineNumbers	DWORD
NumberOfRelocations	WORD
NumberOfLineNumbers	WORD
Characteristics	DWORD

Como les había dicho desde un comienzo un ejecutable se divide en encabezados y cuerpo del archivo el cual se ve a su vez dividido en secciones. Este encabezado es que el nos brinda información acerca de esas secciones, el número de secciones de un archivo como ya miramos es proporcionado por el campo NumberOfSections (FILE HEADER). El encabezado de secciones es una matriz de NumberOfSections estructuras cada una de una sección distinta. Esta estructura tiene un tamaño de 0x28 bytes.

NAME:

Este campo tiene un tamaño de 8 bytes, esta serie de bytes contendrán una cadena terminada con un carácter nulo (0x00) en caso de que la cadena sea menor a 8 caracteres. Si la cadena tiene exactamente 8 caracteres no se usa un carácter nulo al final. Hay que tener en cuenta que este campo no puede tener más de 8 bytes.

Name	Virtual Offset	Virtual Size	Raw Offset	Raw Size	Flags
EP .text	00001000	00016ED8	00001000	00017000	60000020
.data	00018000	00000AAC	00018000	00001000	C0000040
.rsrc	00019000	00000934	00019000	00001000	40000040

VirtualSize:

Este campo tiene un tamaño de 4 bytes y su valor corresponde al tamaño que ocupara la sección una vez cargada en memoria. Si este valor es mayor al SizeOfRawData los bytes sobrantes serán rellenados por caracteres nulos (0x0).

VirtualAddress:

Este campo tiene un tamaño de 4 bytes y su valor corresponde a la dirección virtual relativa donde se cargarán los datos de la sección es decir apunta al primer byte de la sección en memoria. Tiene como obligación ser múltiplo del SectionAlignment.

SizeOfRawData:
Este campo tiene un tamaño de 4 bytes y su valor hace referencia al tamaño de la información inicializada en disco correspondiente a la sección. Si la sección solamente tuviese información No inicializada este campo tendrá el valor 0, además debe ser múltiplo del FileAlignment.

PointerToRawData:
Este campo tiene un tamaño de 4 bytes y su valor corresponde al desplazamiento del archivo donde se encuentra el primer byte de la sección en disco. Este valor debe ser múltiplo del FileAlignment y en caso de que la sección únicamente tuviese información no inicializada el valor de este campo sería 0.

Name	VOffset	VSize	ROffset	RSize	Flags
.data	00001000	0000001C	00000400	00000200	C0000040

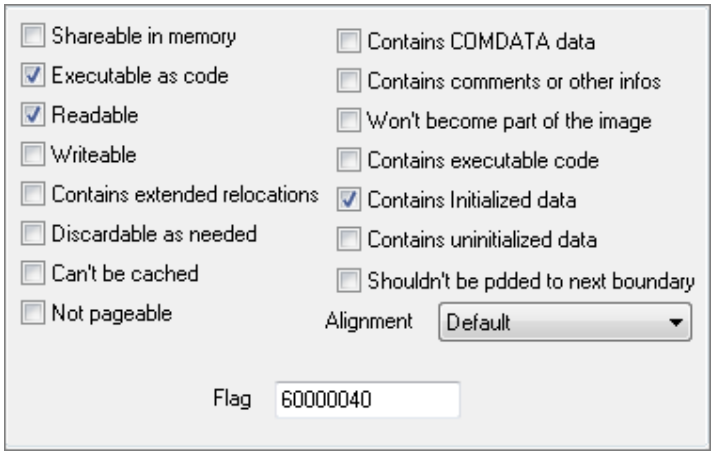
PointerToRelocations:
Este campo tiene un tamaño de 4 bytes y su valor apunta al comienzo de las entradas de reubicaciones de la sección. En archivos ejecutables este valor por defecto es 0, ya que esto lo hace el Reloc directory.

PointerToLineNumbers:
Este campo tiene un tamaño de 4 bytes y su valor solo aplica para archivos COFF, en archivos ejecutables su valor es 0.

NumberOfRelocations:
Este campo tiene un tamaño de 2 bytes y su valor corresponde al número de entradas de las reubicaciones en la sección. En archivos ejecutables su valor es 0.

NumberOfLineNumbers:
Este campo tiene un tamaño de 2 bytes y su valor solo aplica para COFF, en archivos ejecutables su valor es 0.

Characteristics:
Este campo tiene un tamaño de 4 bytes y su valor expresa las distintas características que describen la sección. Aquí también aplica múltiples características de manera que se suma el valor de cada característica presente.

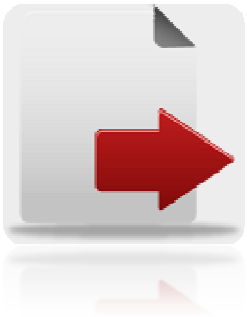


Cada característica tiene un valor individual aquí les dejo la tabla con las características principales:

CARACTERISTICA	VALOR
IMAGE_SCN_CNT_CODE	0x00000020
IMAGE_SCN_CNT_INITIALIZED_DATA	0x0000040
IMAGE_SCN_CNT_UNINITIALIZED_DATA	0x00000080
IMAGE_SCN_MEM_SHARED	0x10000000
IMAGE_SCN_MEM_EXECUTE	0x20000000
IMAGE_SCN_MEM_READ	0x40000000
IMAGE_SCN_MEM_WRITE	0x80000000
IMAGE_SCN_MEM_NOT_PAGED	0x08000000
IMAGE_SCN_GPREL	0x00008000
IMAGE_SCN_MEM_DISCARDABLE	0x02000000
IMAGE_SCN_MEM_NOT_CACHED	0x04000000

Con esto terminamos los encabezados de sección y ahora pasaremos a explicar cada entrada del IMAGE DATA DIRECTORY.

EXPORT DIRECTORY



El export directory está ubicado por la dirección que contiene el RVA del primer campo del IMAGE DATA DIRECTORY. Este directorio contiene información acerca de las funciones a las cuales mediante enlace dinámico pueden acceder otros archivos ejecutables y también archivos DLL. Generalmente quienes contienen directorio de exportación son las librerías (DLL) las cuales a su vez también pueden importar funciones de otras librerías.

El export directory viene distribuido en las siguientes tablas:

TABLA	DESCRIPCIÓN
EXPORT DIRECTORY TABLE	Esta tabla contiene información general acerca del directorio de exportaciones como número de entradas de cada directorio, etc.
EXPORT ADDRESS TABLE	Esta tabla contiene una serie de RVA de las funciones exportadas.
NAME POINTER TABLE	Esta tabla contiene una serie de punteros a los nombres de las funciones exportadas organizadas en forma ascendente.
ORDINAL TABLE	Esta tabla contiene una serie de ordinales de corresponden a cada miembro de la tabla de exportación de nombres, su correspondencia es de acuerdo a su posición.
EXPORT NAME TABLE	Esta tabla contiene una serie de cadenas terminadas con un carácter nulo las cuales corresponden a cada función exportada.

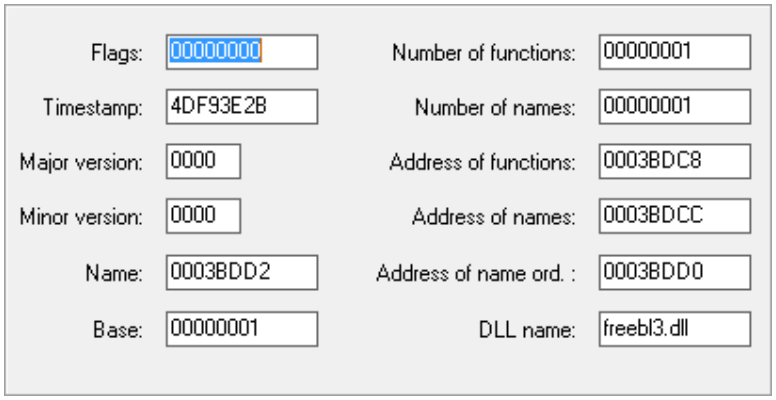
EXPORT DIRECTORY TABLE

Esta tabla presenta la siguiente estructura:

CAMPO	TAMAÑO
ExportFlags	DWORD
TimeDateStamp	DWORD
MajorVersion	WORD
MinorVersion	WORD
NameRVA	DWORD
OrdinalBase	DWORD
NumberOfFunctions	DWORD
NumberOfNamePointers	DWORD
ExportAddressTableRVA	DWORD
NamePointerRVA	DWORD
OrdinalTableRVA	DWORD

CAMPO	DESCRIPCIÓN
ExportFlags	Este campo tiene un tamaño de 4 bytes y aparece como reservado y su valor debe ser 0.
TimeDateStamp	Este campo tiene un tamaño de 4 bytes y contiene la hora y fecha en que los datos de exportación se han creado.
MajorVersion	Este campo tiene un tamaño de 2 bytes y presenta el número de versión menor. El mayor y menor número de versión pueden ser establecidos por el usuario.
MinorVersion	Este campo tiene un tamaño de 2 bytes y contiene el número de menor versión.
NameRVA	Este campo tiene un tamaño de 4 bytes y contiene una dirección RVA que apunta a la cadena que contiene el nombre de la librería. Esta dirección es relativa al ImageBase.
OrdinalBase	Este campo tiene un tamaño de 4 bytes y contiene el primer ordinal desde el cual se inician las exportaciones. Este campo muestra donde inicia el número ordinal para la tabla de exportación de direcciones.
AddressTableEntries	Este campo tiene un tamaño de 4 bytes y contiene el número de entradas en la tabla de exportación de direcciones.
NumberOfNamePointers	Este campo tiene un tamaño de 4 bytes y contiene el número de entradas en la tabla de punteros de nombres. Este valor debe ser igual al número de entradas en la tabla de ordinales.
ExportAddressTableRVA	Este campo tiene un tamaño de 4 bytes y contiene la dirección RVA de la tabla de exportación de direcciones. Esta dirección es relativa al ImageBase.
NamePointerRVA	Este campo tiene un tamaño de 4 bytes y contiene la dirección RVA de la tabla de punteros de nombres. El tamaño de esta tabla es proporcionado por el campo NumberOfNamePointers.
OrdinalTableRVA	Este campo tiene un tamaño de 4 bytes y contiene la dirección RVA de la tabla de ordinales. Esta dirección es relativa al ImageBase.

EXPORT DIRECTORY TABLE



EXPORT ADDRESS TABLE

Esta tabla contiene una serie de direcciones RVA correspondientes a cada función exportada, el índice para ubicarlas es el número ordinal.

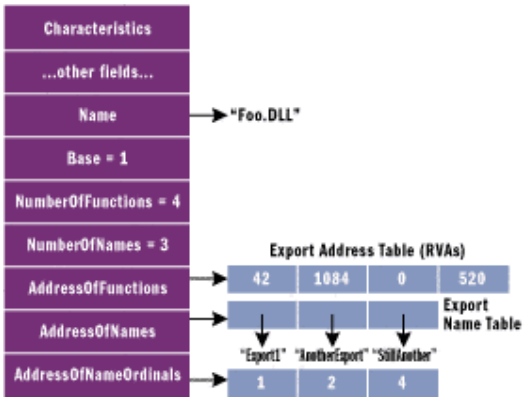
Cada entrada en la tabla de exportación de direcciones es un campo que usa 1 de 2 formatos en la siguiente tabla. Si la dirección especificada no se encuentra dentro de la sección de exportación (definida por la dirección y la longitud en el IMAGE DATA DIRECTORY) el campo es una dirección RVA de

exportación, que es una dirección actual en el código o en datos, de otra manera este campo es un ForwarderRVA que nombra una función en otra librería.

CAMPO	DESCRIPCIÓN	OFFSET
ExportRVA	Este campo tiene un tamaño de 4 bytes y contiene la dirección de la función exportada cuando es cargado en memoria. Esta dirección es un RVA relativa a la ImageBase.	0
ForwarderRVA	Este campo tiene un tamaño de 4 bytes y ocupa el mismo lugar que el anterior. Su dirección es un puntero a una cadena terminada en carácter nulo, esta cadena debe estar dentro de la sección de exportación. Esta cadena proporciona el nombre de la librería y el nombre de la función exportada (por ejemplo "MYDLL.expfunc") o el nombre de la librería y el número de ordinal de exportación (por ejemplo "MYDLL.#27).	0

Para comprender bien esta parte, que no les niego a mí también me ha dado drama entenderla voy a explicarla con mis palabras.
Cuando puse OFFSET 0 me referí a que el mismo campo puede ser bien ExportRVA o ForwarderRVA y para distinguirlo debemos saber si ese RVA está en el rango Export Directory Address (Proporcionado por el IMAGE DATA DIRECTORY) + Export Directory Size (Proporcionado igual que el anterior).
Si se encuentra fuera de este rango es un ExportRVA al cual sumado el ImageBase en el cual se carga la librería obtendríamos la dirección de la función en memoria. Ahora si se encuentra en el rango será un ForwarderRVA y apuntará a una cadena como: NTDLL.RtlAllocateHeap. Este sistema es una implementación que trae Windows donde puede exportar funciones de otras librerías por redirección generalmente para trabajo en modo usuario con funciones nativas.

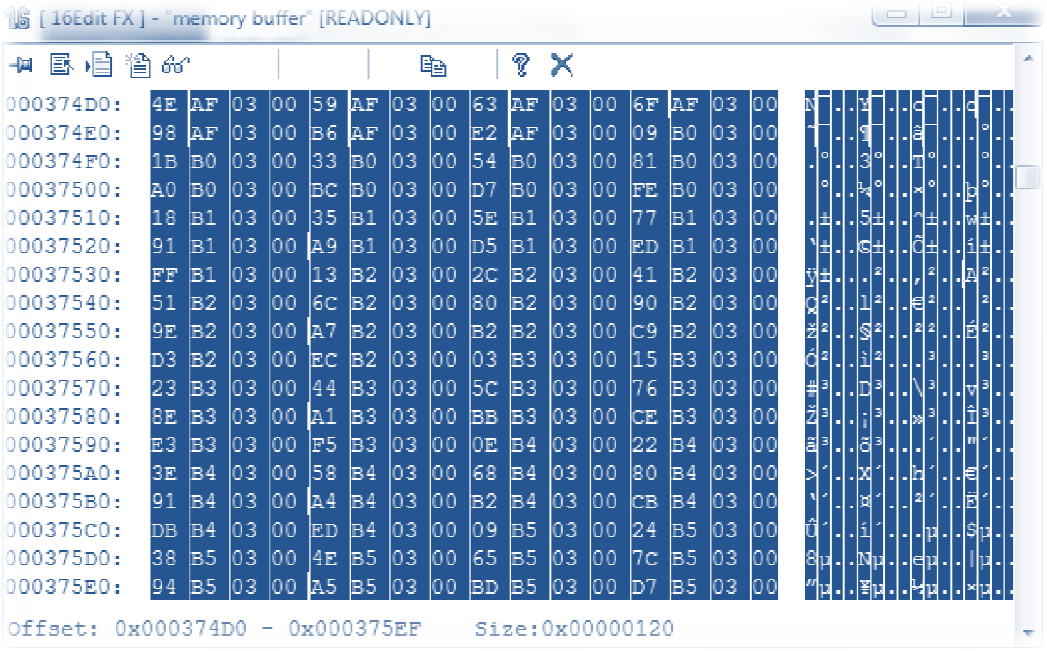
Hay un grafico de la MSDN que explica muy bien la lectura de las direcciones de las funciones exportadas donde el ordinal que es ubicado de acuerdo a la posición de la cadena de la función es el índice en la tabla de direcciones para obtener la misma:



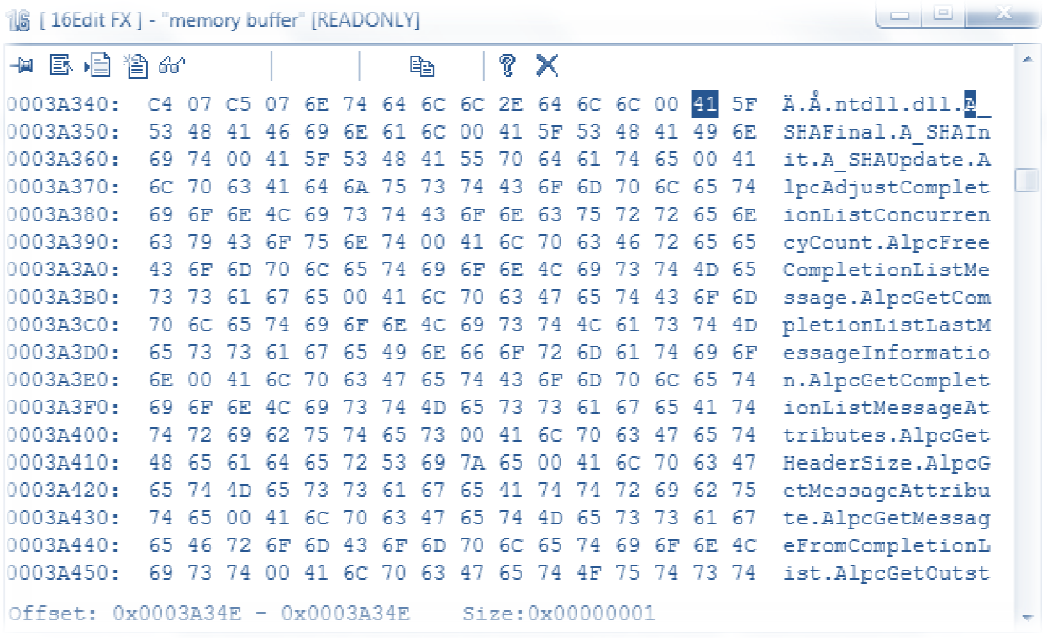
NAME POINTER TABLE

Esta tabla contiene una serie de direcciones RVA que apuntan a las cadenas de los nombres de cada función exportada. Es un puntero de 4 bytes y la dirección es relativa al ImageBase. Una exportación por nombre está definido solo si su puntero se encuentra en esta tabla.

NAME POINTER TABLE IMAGEN



EXPORT NAME TABLE:

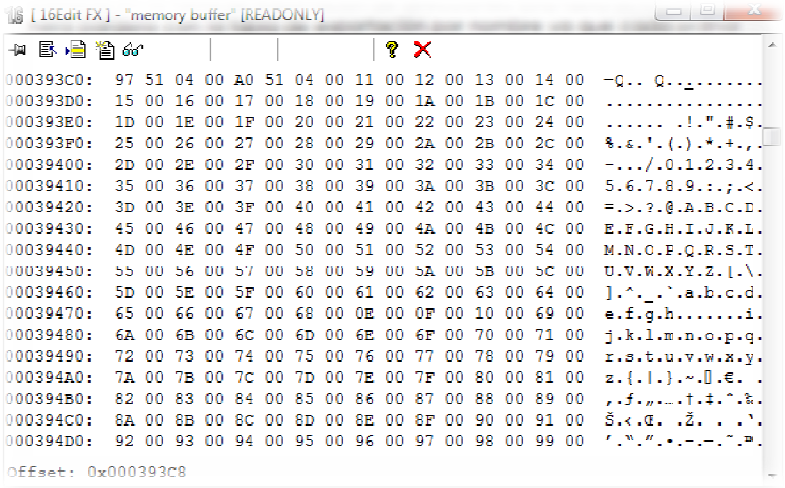


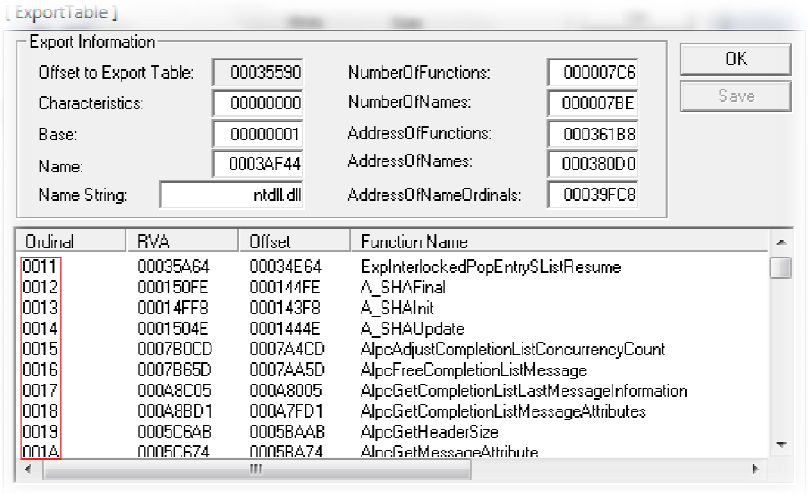
ORDINAL TABLE

Esta tabla contiene una serie de valores de 16 bits (WORD-2 Bytes) los cuales serán los índices de cada función en la tabla de exportación de direcciones.
El valor del OrdinalBase deberá restarse a cada ordinal para obtener el índice correcto en la tabla de exportación de direcciones. Esta tabla actúa de manera paralela con la tabla de exportación por nombre ya que cada ordinal corresponderá a una cadena, todo basado en la posición de cada cadena.

EXPORT NAME TABLE

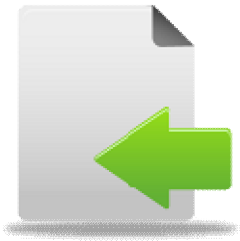
Esta tabla contiene una serie de cadenas correspondientes a lo apuntado por la tabla de apuntadores a nombres. Estas cadenas son públicas para que otros ejecutables puedan llamarlas o ubicarlas sin ningún inconveniente.
Como cada cadena viene en ASCII por lo cual la única forma de separarlos y obtener su tamaño es usando como delimitador y final de cada cadena el carácter nulo (0x00).



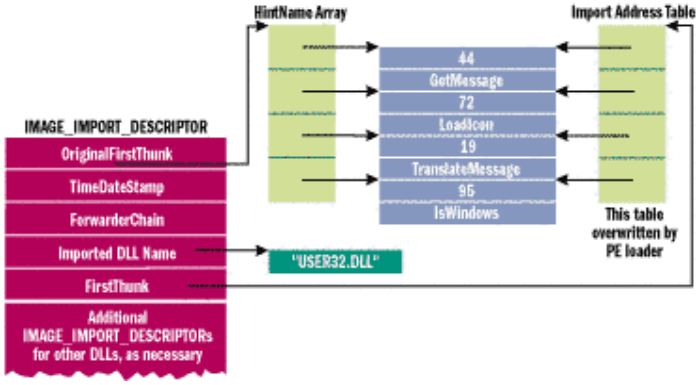


Con esto terminamos el directorio de exportación.

IMPORT DIRECTORY



El import directory se encuentra en la mayoría de archivos de imagen (DLL & EXE), generalmente el archivo que tiene importaciones es decir enlaza funciones de librerías bien del sistema operativo o propias suele tener entre su lista de secciones ".idata" o ".rdata".
Este directorio se divide de acuerdo a la cantidad de librerías de las cuales se haya importado funciones y presente más o menos la siguiente estructura:



La estructura básica para cada DLL lo conocemos como IMPORT_DESCRIPTOR, cada DLL tiene el suyo y la estructura general de este es:

CAMPO	DESCRIPCIÓN	TAMAÑO
OriginalFirstThunk	Este campo tiene una dirección RVA que apunta a una serie de direcciones de las cuales cada una nos dirá si la importación se realiza mediante el nombre o el número ordinal.	DWORD
TimeDateStamp	Este campo generalmente se encuentra a 0 y cuando el el IMPORT_DESCRIPTOR corresponde a una importación obligada se pone a 0xFFFFFFFF.	DWORD
ForwarderChain	Este campo es obsoleto se utilizaba en versiones anteriores de unión y refería al primer expedidor de API. Este campo ya no se utiliza y comúnmente se encuentra a 0 o -1.	DWORD
NameRVA	Este campo contiene una dirección RVA que apunta hacia la cadena que contiene el nombre de la DLL. Esta dirección es relativa al ImageBase.	DWORD
FirstThunk	Este campo es muy similar al OriginalFirstThunk con la diferencia que sobre lo que apunta este campo se escriben las direcciones de las funciones importadas.	DWORD

IMPORTACIONES POR ORDINAL:

Como les había comentado el contenido apuntado por los campos OriginalFirstThunk y FirstThunk determinan como se importarán las funciones y aquí explicaré como se importan por número ordinal.
Deberemos tener en cuenta la siguiente tabla para conocer como se importan:

CAMPO	BITS	DESCRIPCIÓN
OrdinalNameFlag	1	Si este bit está establecido con valor verdadero (1) la importación deberá hacerse por número ordinal, de lo contrario se hará por nombre. Para no confundirlos la manera más fácil de distinguir es basándose en que si la dirección apuntada comienza en base a 0x80000000 se importará por ordinal (el primer bit determina esa base).
OrdinalNumber	16-32	Este campo con un tamaño de 16 bits es decir ocupa los últimos 2 bytes para almacenar el número ordinal con el cual se hará la importación.
NameTableRVA	1-32	Este campo se utilizará únicamente si el campo OrdinalNameFlag es 0. Este campo apuntará a otra estructura en la cual se importará por nombre.

Ahora si importamos por número ordinal ¿cómo obtendría la dirección de la función importada el loader de Windows?
Si prestaron atención en la parte de las exportaciones les mencione en que la tabla de exportación de direcciones se emplea un índice el cual viene siendo el ordinal. Pues bien el campo OrdinalNumber nos provee ese número y con él podemos obtener la dirección de la función a importar de la librería correspondiente.

IMPORTACIONES POR NOMBRE:

Ya sabiendo cómo se determina una importación por nombre lo que se hace aquí es sencillo. La dirección que lo determinaba apuntaba a la siguiente estructura:

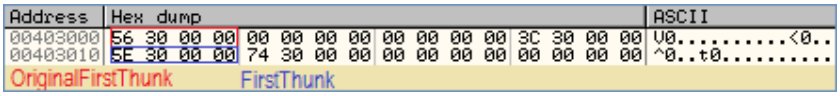
CAMPO	TAMAÑO	DESCRIPCIÓN
Hint	WORD	Éste campo contiene el índice de la tabla de exportación de punteros de nombres. Este campo es una opción adicional a la búsqueda binaria por nombre.
Name	VARIABLE	Éste campo contiene una cadena terminada con un carácter nulo (obligatoriamente) mediante la cual se buscará de manera binaria la dirección en el directorio de exportaciones de la librería correspondiente. Este campo es sensible a mayúsculas.

Con conocer esta estructura nos basta ya que el Loader de Windows hará la tarea de cargar la dirección de todas las funciones para que nosotros podamos usarlas sin problemas.

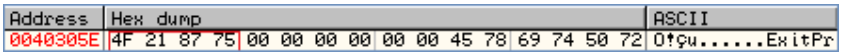
DIFERENCIANDO ORIGINALFIRSTTHUNK Y FIRSTTHUNK:

He visto tantos tutoriales que de estos 2 campos solo te dicen que el segundo es un duplicado del primero, pero ninguno se enfoca en que por algo estarán duplicados. Además hay que tener en cuenta muchos aspectos como por ejemplo los archivos ejecutables de Borland no cuentan con OriginalFirstThunk, todo lo hacen mediante el FirstThunk. Esto debido a que el campo Original-FirstThunk no es obligatorio y puede no existir, y en diferencia el campo FirstT-hunk es obligatorio y ambos deben apuntar al mismo valor.

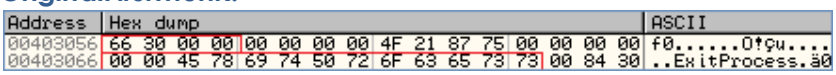
El punto de su diferencia está cuando el ejecutable está cargado en memoria y creo que mirándolo entenderán la diferencia.



FirstThunk:



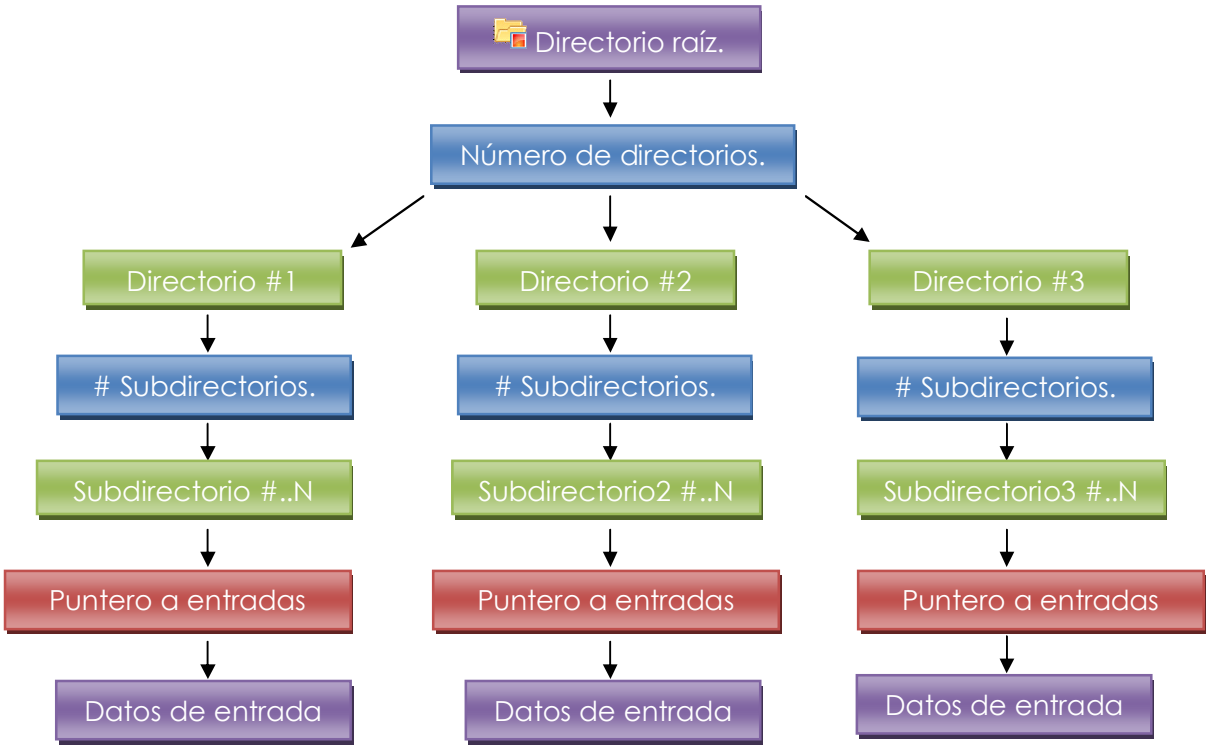
OriginalFirstThunk:



Se supone que ambos deberían apuntar a lo mismo, pero si miramos el FirstT-hunk tiene una dirección marcada por Olly en rojo, es porque ahí se ha escrito información nueva y esa dirección que ustedes ven 0x7587214F es la dirección de la función y en cambio OriginalFirstThunk tiene un RVA el cual apunta a la cadena con el nombre de la función. Cabe decir que en disco si son iguales más no en memoria.

RESOURCE DIRECTORY

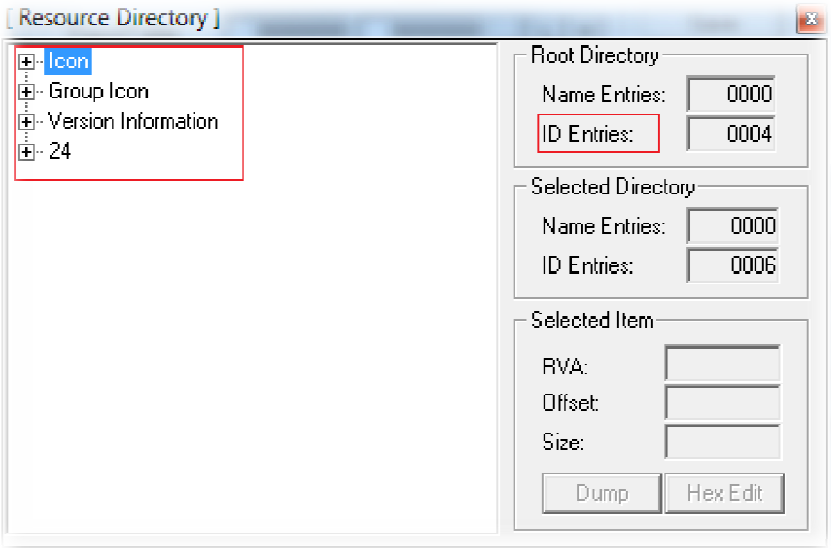
El resource directory el tercer directorio ubicado en el IMAGE DATA DIRECTORY, su principal utilidad es dar información de los recursos que tiene el programa como iconos, imágenes, información de la versión, etc. Casi siempre (por no decir siempre) esta información se encuentra en una sección nombrada ".rsrc".
El directorio de recursos presenta una estructura base explicada por el siguiente grafico:



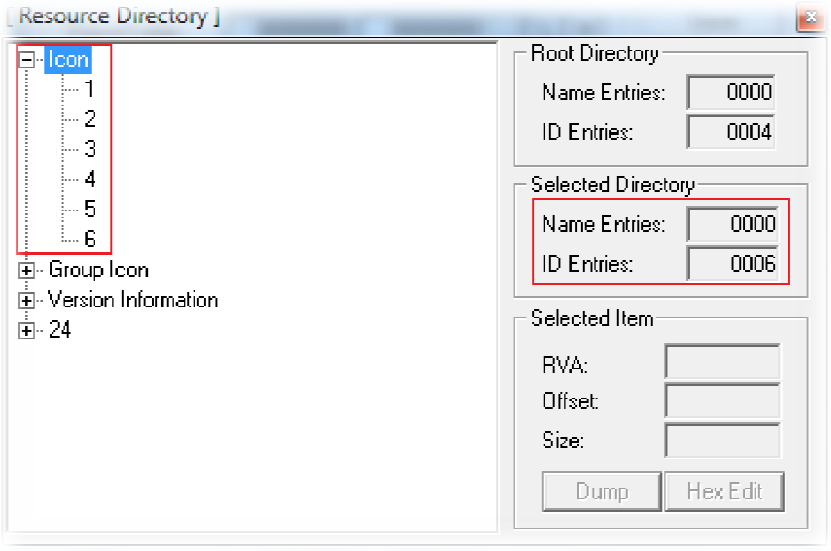
A continuación daré una explicación de cómo se organizan los datos y luego entramos en detalles.

EXPLICACIÓN:

Podemos decir que la base de los recursos son sus directorios, ya que ayuda a mantener un buen orden y proporcional facilidad de acceso. Existe un directorio raíz el cual contiene la cantidad de subdirectorios y a la vez sus tipos y direcciones:



Cada directorio tendrá un subdirectorio, la cantidad de subdirectorios viene determinado por 2 campos de la estructura de directorios, estos campos son Name Entries e ID Entries.

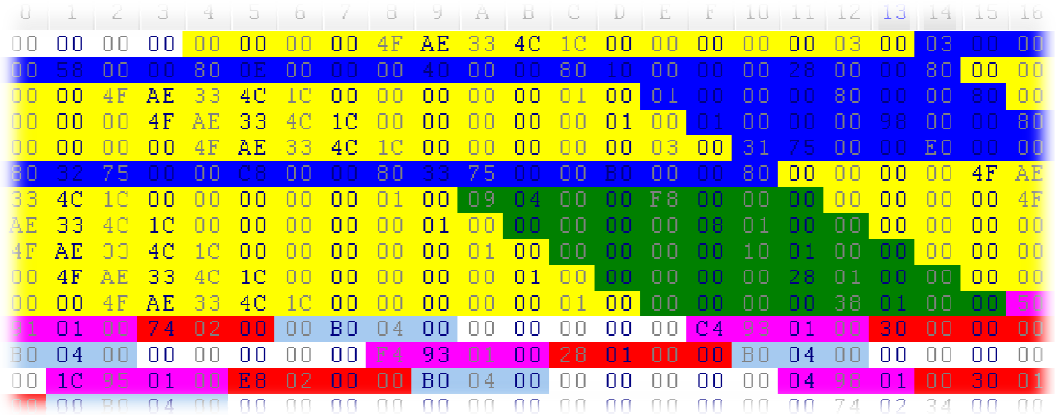


Ahora miremos la estructura de los directorios para que comprendamos mejor.

ESTRUCTURA DE DIRECTORIO DE TABLA DE RECURSOS:

CAMPO	TAMAÑO	DESCRIPCION
Characteristics	4	Caracteristicas del recurso, este campo es reservado para uso futuro, actualmente es 0.
TimeDateStamp	4	El tiempo en que los datos del recurso fueron creados por el compilador de recursos.
MajorVersion	2	El número mayor de versión, establecido por el usuario.
MinorVersion	2	El número menor de versión, establecido por el usuario.
NumberOfNameEntries	2	El número de entradas de directorio inmediatamente seguido de una tabla de cadenas para identificar tipo, lenguaje y nombre.
NumberOfIDEntries	2	El número de entradas de directorio inmediatamente después de las entradas de nombre que utilizan identificadores numéricos para tipo nombre y lenguaje.

Este grafico presenta más o menos la estructura de un directorio de recursos, señalando cada uno de sus campos:



En este grafico podemos ver:

- **Color amarillo**: Directorios
- **Color azul**: Tipo y desplazamiento en base del inicio del directorio de recursos.
- **Color verde**: Tipo y desplazamiento final a los datos de entrada del recurso.
- **Color violeta**: Dirección virtual relativo del inicio de la información del recurso
- **Color rojo**: Tamaño del recurso.

La parte de color azul se conoce como Entradas en el directorio de recursos y está compuesto por 2 campos, la estructura que la conforma es la siguiente:

CAMPO	TAMAÑO	DESPLAZAMIENTO	DESCRIPCIÓN
NameRVA	4	0	Este campo se emplea cuando el campo NumberOfNameEntries es mayor a 0. Se reconoce porque el primer bit es 1 es decir la dirección es en base a 0x80000000. Y descartando la base se suma el resto a el inicio del directorio de recursos y tendremos la cadena en UNICODE.
IntegerID	4	0	Este campo se emplea en base al NumberOfIDEntries y su valor especifica el tipo de recurso, por ejemplo Icon = 0x03.
DataEntryRVA	4	4	Cuando el primer bit de este campo es 0 el campo tiene un desplazamiento inmediato en base del inicio del directorio de recursos para encontrar los datos de entrada del recurso.
SubdirectoryRVA	4	4	Cuando el primer bit de este campo es 0 el campo apunta a otro directorio un nivel hacia abajo.

Las cadenas UNICODE presentan la siguiente estructura:

CAMPO	TAMAÑO	DESCRIPCIÓN
Length	2	Este campo nos proporciona el tamaño de la cadena en UNICODE sin contar los bytes nulos intermedios.
UNICODE	Variable	Este campo contiene la información, la cadena en sí.

Me basaré en la grafica para dar un último ejemplo.
El primer campo amarillo viene siendo el directorio raíz y una vez terminado y empezada la parte azul tenemos aplicamos la estructura de entradas en el directorio de recurso, tomando los primeros 8 bytes correspondientes a 0x00000003 y 0x80000058. Aquí se aplica IntegerID correspondiente a la constante Icon y un desplazamiento que la identificamos por la base 0x80000000 Ahora tomamos la base del directorio y sumamos 0x58 y nos lleva al directorio de la quinta línea y volvemos a repetir lo mismo, ahora nos lleva a Base + 0xE0
El directorio de la línea 10 faltando 2 bytes para pasar a la 11, que ya nos lleva a direcciones verdes es decir desplazamiento inmediato en este caso nos lleva al RVA 0x00019804 y justo en ese inicio se aplica la estructura dato de entrada en recurso:

CAMPO	TAMAÑO	DESCRIPCIÓN
DataRVA	4	La dirección virtual relativa de la unidad del recurso.
Size	4	Tamaño de la unidad del recurso. (RVA + Size = Recurso completo).
CodePageRVA	4	El código de página que es usado para decodificar el punto de valor con la información del recurso, típico en páginas UNICODE.
Reserved	4	Campo reservado establecido a 0.

Creo que con esto terminamos, espero hayan comprendido.

RELOCATION DIRECTORY

Si recordamos conceptos como ImageBase que era la dirección base donde se cargaba el ejecutable en memoria podremos comprender la finalidad del reloc directory. Existen casos en que el ejecutable no se carga en el ImageBase original si no en otra dirección debido a que quizá la dirección del ImageBase este ocupada, pero un archivo ejecutable especialmente en su IMAGE DATA DIRECTORY incluye muchas direcciones físicas al igual que en saltos a las importaciones, etc. Si estamos en otra base distinta a la proporcionada por el ImageBase el ejecutable no funcionará puesto a que no podría llegar a información vital. Y aquí es donde entran este directorio, pues proporciona el mecanismo de por decirlo reubicar estas direcciones estáticas para que tenga acceso a esa información si ningún inconveniente. Cabe tener en cuenta que este mecanismo solo se aplica cuando nos hemos cargado en una dirección base distinta al ImageBase.

EXPLICACIÓN DEL FUNCIONAMIENTO

Este mecanismo funciona de una única manera, supongamos que tenemos los siguientes datos:

DATO	VALOR
ImageBase original	0x00400000
Direcciones físicas	0x00403000; 0x00403010
ImageBase actual	0x00800000

Nos cargamos en memoria y el programa ejecutará un mensaje mediante el API **MessageBoxA** la cual necesita una cadena para el título y para el mensaje a mostrar, pues bien estas 2 direcciones físicas en la sección de datos contienen estas cadenas. Si nos cargamos en 0x00800000 las cadenas no se abran cargado en las correspondientes direcciones físicas. Para corregir esto este en teoría se haría lo siguiente:

Obtenemos el desplazamiento restando a nuestro ImageBase actual 0x00800000 el ImageBase original 0x00400000 y tendríamos un valor 0x00400000 al cual conocemos como Delta. Ahora para arreglar estas direcciones simplemente deberíamos sumar el delta a las direcciones físicas.

OPERACIÓN	RESULTADO
0x00403000 + 0x0040000	0x00803000
0x00403010 + 0x0040000	0x00803010

Según esta tabla ya se han actualizado correctamente las direcciones y por lo tanto el programa funcionaría de manera correcta. Básicamente esto es lo que sucede en las reubicaciones de ejecutables. El directorio de reubicación contiene las direcciones a reubicar y el tipo de reubicación.
Las reubicaciones vienen divididas en bloques y cada bloque representa las reubicaciones de 1 página de memoria (4096 bytes).

BASE RELOCATION BLOCK

CAMPO	TAMAÑO	DESPLAZAMIENTO	DESCRIPCIÓN
PageRVA	4	0	Este campo proporciona la dirección virtual relativa sobre la cual se aplicarán las reubicaciones del bloque actual.
BlockSize	4	4	Este campo proporciona el tamaño del bloque actual es decir el bloque de reubicaciones de una página en memoria. Se incluyen los campos PageRVA y BlockSize.

Ahora para saber el número de entradas del bloque lo haríamos con la siguiente operación:

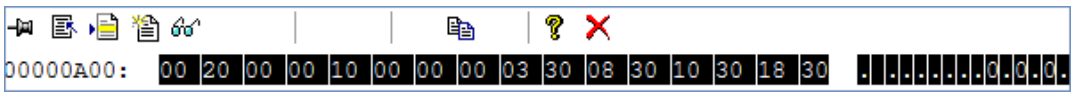
$$Número\ de\ entradas = (BlockSize - 8)/2$$

Restamos 8 para quitar los 2 campos BlockSize y PageRVA y divido entre 2 porque cada campo tiene en tamaño de un WORD (2 bytes).

Ahora cada una de las entradas utiliza la siguiente estructura ya que cada entrada especifica tipo y RVA de la reubicación.

CAMPO	TAMAÑO	DESCRIPCIÓN
Type	4 bits	Este campo proporciona el tipo de reubicación a aplicar, uno de los más comunes es REL_BASED_HIGHLOW cuyo valor es 0x03 y dice que la reubicación se aplica a los 32 bits de la dirección a reubicar. Así como esta hay más valores que los dejo debajo de esta tabla.
Offset	12 bits	Este campo nos proporciona el desplazamiento con respecto al valor del PageRVA sobre el cual será aplicada la reubicación.

Ahora miremos un ejemplo y detallemos.



Podemos detallar los valores que veremos en la siguiente tab

Según la fórmula anterior tenemos un total de 4 entradas y son:

ENTRADA	VALOR	TYPE	OFFSET	RVA
#1	0x3003	0x3	0x003	0x2003
#2	0x3008	0x3	0x008	0x2008
#3	0x3010	0x3	0x010	0x2010
#4	0x3018	0x3	0x018	0x2018

TIPOS DE REUBICACIÓN COMUNES

TIPO	VALOR	DESCRIPCIÓN
IMAGE_REL_BASED_ABSOLUTE	0x0	La base de la reubicación se omite. Este tipo puede ser usado para rellenar un bloque.
IMAGE_REL_BASED_HIGH	1	La reubicación de la base agrega los 16 bits más altos de la diferencia a los 16 bits del campo a reubicar.
IMAGE_REL_BASED_LOW	2	La reubicación de la base agrega los 16 bits más bajos de la diferencia a los 16 bits del campo a reubicar.
IMAGE_REL_BASED_HIGHLOW	3	La reubicación de la base aplica los 32 bits de la diferencia a los 32 bits del campo a reubicar.

Creo que aquí termino con esta parte, espero hayan comprendido.

DEBUG DIRECTORY

El debug directory contiene información que brinda ayuda a los distintos debuggers (depuradores) a la hora de depurar el archivo. El directorio consiste en una serie de directorios de depuración. La existencia de un directorio de depuración no depende de que exista una sección .debug ya que esta información puede estar sin ningún problema en otra sección del archivo.
Cada entrada de directorio de depuración identifica la ubicación y el tamaño de un bloque de información de depuración. El RVA especificado puede ser cero si la información de depuración no está cubierto por un encabezado de sección (es decir, que reside en el archivo de imagen y no se asignan al espacio de direcciones en tiempo de ejecución). Si está asignado, el RVA es su dirección.
Cada entrada en el directorio de depuración utiliza el siguiente formato:

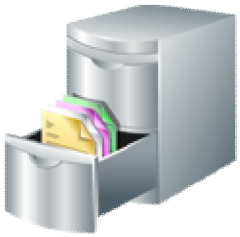
CAMPO	TAMAÑO	DESPLAZAMIENTO	DESCRIPCIÓN
Characteristics	4	0	Campo reservado, debe tener un valor de 0.
TimeDateStamp	4	4	Este campo contiene tiempo y fecha de la creación del directorio de depuración.
MajorVersion	2	8	Este campo contiene el número mayor de la versión de la información del formato de depuración.
MajorVersion	2	10	Este campo contiene el número menor de la versión de la información del formato de depuración.
Type	4	12	Contiene el formato de la información del formato de depuración. Permite soporte a múltiples depuradores.
SizeOfData	4	16	Este campo contiene el tamaño de la información de depuración, sin incluir su mismo directorio de depuración.
AddressOfRawData	4	20	Este campo contiene la dirección RVA de la información de depuración cuando el ejecutable es cargado en memoria.
PointerToRawData	4	24	Este campo contiene el puntero en el archivo a la información de depuración.

Este directorio no tiene mucha información y por decirlo aunque tiene utilidad no es imprescindible ni la más importante así que terminamos con los tipos de depuración.

CONSTANTE	VALOR	DESCRIPCIÓN
IMAGE_DEBUG_TYPE_UNKNOWN	0	Un valor desconocido que es ignorado por todas las herramientas.
IMAGE_DEBUG_TYPE_COFF	1	Información de depuración para archivos COFF(Objetos).
IMAGE_DEBUG_TYPE_CODEVIEW	2	Información de depuración de Visual C++.
IMAGE_DEBUG_TYPE_FPO	3	La información del marco de omisión de punteros. Esta información indica al depurador como interpretar los marcos de pila estándar, que utilizan el registro EBP para propósito que no sea puntero.
IMAGE_DEBUG_TYPE_MISC	4	La localización del archivo DBG.
IMAGE_DEBUG_TYPE_EXCEPTION	5	Una copia de la sección .pdata.
IMAGE_DEBUG_TYPE_FIXUP	6	Reservado.
IMAGE_DEBUG_TYPE_OMAP_TO_SRC	7	La asignación de un RVA en la imagen de un RVA en la imagen de origen.
IMAGE_DEBUG_TYPE_OMAP_FROM_SRC	8	La asignación de un RVA en la imagen de origen a un RVA en la imagen.
IMAGE_DEBUG_TYPE_BORLAND	9	Reservado para Borland.
IMAGE_DEBUG_TYPE_RESERVED10	10	Reservado.
IMAGE_DEBUG_TYPE_CLSID	11	Reservado.

Creo que no hay más información que dar ya que es un tema que ya se amplía metiéndonos de lleno a depuración y no es el fin de este documento.

TLS DIRECTORY (thread local storage)



TLS (thread local storage) es una clase especial de almacenamiento que soporta Windows. Esta contiene información de cada hilo (thread) y por lo tanto se ejecutaran posteriores a su creación y antes del Entrypoint porque está información tiene que ver con la ejecución del hilo.
De aquí podemos derivar Thread local storage Callbacks, los cuales son una serie de direcciones virtuales que apuntan a funciones a ser llamados por el loader después de la creación del Thread. La finalidad de los TLS Callbacks es principalmente la inicialización de objetos y por eso provee el soporte de funciones.

INFORMACIÓN:

En mis pruebas de este mecanismo me he dado cuenta de ciertos detalles que son importantes tenerlos en cuenta. Este mecanismo se ejecuta como acción de funciones en ntdll.dll antes de que siquiera se ejecute la primera instrucción del Entrypoint. Con permisos respectivos desde una función de TLS Callbacks se puede editar la memoria principal del programa (Uso común por compresores y packers). También para entonces se han cargado las direcciones de las importaciones y como es obvio pero no está demás decirlo la PEB también ha sido asignada al registro FS con su correspondiente desplazamiento. Cabe decir que no se puede almacenar resultados en registros para luego utilizarlos después del Entrypoint, ya que ntdll.dll al llamar a la parte de los TLS Callbacks deja una dirección de retorno, para una vez ejecutadas todas las funciones seguir preparando el archivo para ser ejecutado, por lo tanto los registros quedarán establecidos por ntdll.dll.

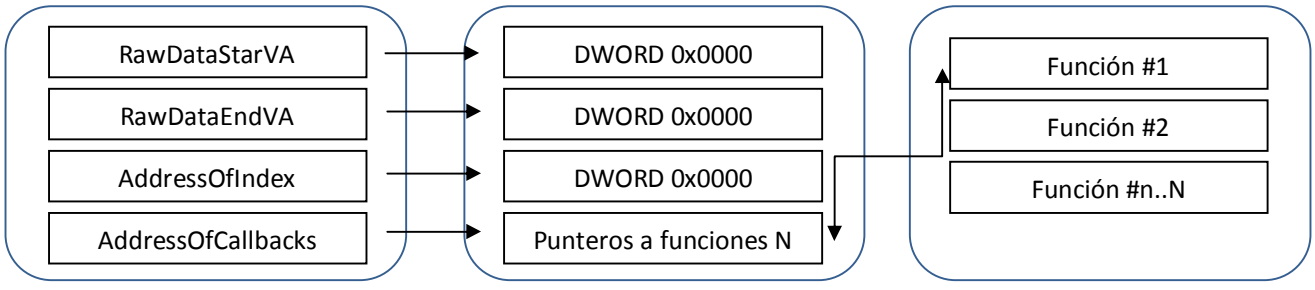
Es importante decir que las direcciones que se trabajan en el directorio son todas Virtuales estáticas, por lo cual se emplean tal cual están y en caso de cargarse en otra base en memoria deben ser reubicadas.

ESTRUCTURA DEL DIRECTORIO TLS

CAMPO	TAMAÑO	DESPLAZAMIENTO	DESCRIPCIÓN
RawDataStarVA	4	0	La dirección de inicio de la plantilla TLS. La plantilla es un bloque que es utilizado para inicializar la información TLS. El sistema copia todo de estos datos cada vez que un hilo es creado.
RawDataEndVA	4	4	La dirección del último byte de la información del TLS excepto para el relleno de ceros.
AddressOfIndex	4	8	El lugar que recibe el índice del TLS que el loader asigna. Esta ubicación es una sección de datos ordinaria por lo que se puede dar un nombre simbólico que sea accesible al programa.
AddressOfCallbacks	4	12	Puntero a la serie de funciones TLS. El fin de la matriz se identifica por que la última entrada tiene valor nulo.
SizeOfZeroFill	4	16	El tamaño en bytes de la plantilla.
Characteristics	4	20	Reservado para posible uso futuro en características de TLS.

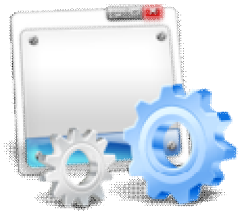
La documentación oficial no lo especifica bien pero tal parece los datos apuntados por estos valores forman una plantilla como se mencionó y los 2 primeros campos apuntan a los 2 primeros valores y seguidos de estos se encuentra el índice de TLS y posteriormente la matriz de direcciones de funciones TLS.

Un punto a tener en cuenta es que el campo AddressOfCallbacks es un puntero a las direcciones de las funciones, es decir una dirección virtual a la serie de punteros a cada función, y están todas seguidas sin ningún separador. AddressOfIndex también es una dirección virtual que apunta a un espacio ubicado antes de comenzar los AddressOfCallbacks y ambos deben estar organizados en plantilla como indiqué anteriormente.



Con esto terminamos la parte de TLS, espero hayan comprendido.

DIRECTORIO LOAD CONFIG



Este directorio describe varias opciones por que por su longitud se dificulta escribirlas en el IMAGE OPTIONAL HEADER o IMAGE FILE HEADER. Las actuales versiones del enlazador (Linker) de Microsoft desde la versión de Windows XP utilizan una nueva estructura para sistemas basados en la arquitectura 32-bit x86 y posteriores que incluyen la tecnología de SEH(Structured Exception Handling) reservados.

Este provee una lista de structured exception handlers (controladores estructurados de excepciones) que el sistema utilizará durante el envió de una excepción. Si la dirección del handler (controlador) reside en un rango de direcciones virtuales y es marcado como SEH-aware (es decir, IMAGE_DLLCHARACTERISTICS_NO_SEH se encuentra en DllCharacteristics del IMAGE OPTIONAL HEADER) entonces el handler (controlador) debe estar en la lista de handlers seguros y conocidos para la imagen (archivo en memoria). De lo contrario el sistema operativo terminará la aplicación. Esto ayuda a prevenir técnicas de explotación del sistema de excepciones para tomarse el control del sistema operativo. El enlazador de Microsoft automáticamente provee una estructura de carga de configuración por defecto que incluye la información reservada SEH.

Como miramos en la información oficial esta estructura mayoritariamente va enfocado a la información del manejo estructurado de excepciones.

LOAD CONFIGURATION STRUCTURE

TAMAÑO	CAMPO	DESCRIPCIÓN
DWORD	Characteristics	Banderas que indican las características del archivo, actualmente no usado.
DWORD	TimeDateStamp	Contiene la marca de fecha y hora. El valor se representa en el número de segundos transcurridos desde la medianoche (00-00-00), 1 de enero de 1970, Tiempo Universal Coordinado, de acuerdo con el reloj del sistema. La marca de tiempo se pueden imprimir utilizando (CRT).
WORD	MajorVersion	Número de versión mayor.
WORD	MinorVersion	Número de versión menor.
DWORD	GlobalFlagsClear	Indicadores globales que indican el control del sistema.
DWORD	GlobalFlagsSet	Indicadores globales establecidos que indican el control del sistema.

DWORD	CriticalSectionDefaultTimeout	El valor por defecto la sección crítica de tiempo de espera.
DWORD	DeCommitFreeBlockThreshold	Memoria que debe ser liberada antes de que se devuelva al sistema, en bytes.
DWORD	DeCommitTotalFreeThreshold	Cantidad total de memoria libre, en bytes.
DWORD	LockPrefixTable	[Solamente x86]La dirección virtual de una lista de direcciones donde se utiliza el prefijo LOCK, para que puedan ser sustituidos por NOP en las máquinas de un solo procesador. // VA
DWORD	MaximumAllocationSize	Tamaño máximo de asignación, en bytes.
DWORD	VirtualMemoryThreshold	Tamaño máximo de memoria virtual, en bytes.
DWORD	ProcessHeapFlags	Banderas del process heap (montículo del proceso) que corresponden al primer argumento de la función HeapCreate. Estas banderas aplican al process heap que es creado durante el inicio del proceso.
DWORD	ProcessAffinityMask	Al establecer este campo a un valor distinto de cero es como llamar a SetProcessAffinityMask con este valor durante el inicio del proceso.
WORD	CSDVersion	El identificador de la versión del Service Pack.
WORD	Reserved1	Campo reservado, su valor debe ser 0.
DWORD	EditList	Reservado para uso del sistema. // VA
DWORD	SecurityCookie	Un puntero a una cookie que se utiliza en Visual C++ o la implementación de servicios generales. // VA
DWORD	SEHandlerTable	[Solamente x86]Dirección virtual a una tabla ordenada de RVA de cada válida, único manejador estructurado de excepciones en la imagen. // VA
DWORD	SEHandlerCount	El conteo de los controladores únicos en la tabla.

De este directorio no tenemos mucho que decir, solo que tal parece está más orientado a información del manejo de excepciones. Conociendo esta estructura terminamos.

DELAY IMPORT DIRECTORY

El delay import directory ofrece una serie de tablas las cuales implementan un mecanismo uniforma de retraso para aplicaciones mediante la cual se cargará una librería (DLL) en memoria hasta la primera llamada de una función suya. Es común que su información se encuentre en la sección ".idata", aunque su existencia no depende de ello.
Este directorio también presenta una estructura y es la siguiente:

CAMPO	TAMAÑO	DESCRIPCIÓN
Attributes	4	Hasta el momento no hay banderas de atributos definidos. El enlazador establece este campo con valor 0. Este campo puede ser utilizado para ampliar el registro indicando la presencia de nuevos campos o puede ser usado para indicar el comportamiento del retraso o descarga de funciones de ayuda.
Name	4	La dirección virtual relativa de la librería (DLL) cargada. Este nombre reside en la sección de datos con permiso de únicamente lectura.
ModuleHandle	4	La dirección virtual relativa de el controlador de modulo(module handle)(en la sección de datos del archivo) de la librería a ser cargada con retraso. Se utiliza para el almacenamiento de la rutina que se suministra para la gestión de carga retrasada.
DelayImportAddressTable	4	La dirección virtual relativa de la tabla que contiene las direcciones de importación retrasada. Cada dirección de esta tabla será reemplazada con la dirección real de cada función.
DelayImportNameTable	4	La dirección virtual relativa de la tabla que contiene los nombres de las importaciones retrasadas que necesitan ser cargadas. Se comporta de manera idéntica a las de import directory.
DelayBoundIATandTimeStamp	4	La dirección virtual relativa de la tabla de importaciones retrasadas obligadas, solo si existe.
DelayUnloadImportAddressTable	4	La dirección virtual relativa de la tabla de descarga de importaciones retrasadas (unload delay-load Address table), solo si existe.

		En caso de existir esta será una copia idéntica de DelayImportAddressTable. Tiene como principal objeto la descarga de las direcciones de importación.
TimeDateStamp	4	La marca de tiempo en que la DLL se ha obligado.

Este mecanismo se parece mucho al de importación pero el proceso de carga de funciones se realiza a la primera llamada de una función de las librerías importadas por retraso. Internet Explorer implanta ese mecanismo y mirándolo me di cuenta de que al llamar una función de este directorio se carga la librería en memoria y carga las funciones de la librería.

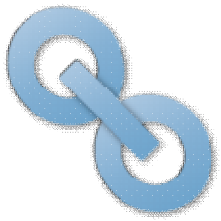
He puesto un breakpoint justo en la primera dirección del delay import address table que corresponde a una función de IEFRAME.DLL y sin iniciar el ejecutable esta librería no se encuentra en memoria. Al iniciar el programa en el debugger el para en esta instrucción:

010A37CA . FF15 00800A01 CALL DWORD PTR DS:[10A8000] ; iexplore.010A3667

Cuya dirección 0x10a8000 fue donde puse el breakpoint, ahora ejecuto esa llamada y una vez hecho eso carga la dirección de la función en 0x10A8000 y también carga la librería IEFRAME.DLL en memoria.

Este es el mecanismo de carga retrasada, muy parecido al import directory pero con diferencias. Espero hayan entendido, paso al último tema.

BOUND IMPORT DIRECTORY



Cuando un ejecutable utiliza este mecanismo lo que hace es guardar en el directorio Import Address Table las direcciones virtuales de las funciones de la librería obligada, es decir no guarda los RVA si no ya las direcciones reales, este proceso ayuda como optimización ya que con esto el ejecutable no necesita carga cada función si no simplemente cargar la librería y comprobar que las direcciones que ya tiene no estén obsoletas, si no lo están el programa evita la parte de carga de funciones, de lo contrario el programa cuenta con la información de los nombres de las importaciones y la librería lo cual le permitiría corregir esto y cargar las direcciones que están obsoletas.
Este vendría siendo un excelente mecanismo de optimización siempre y cuando se tenga en cuenta que la DLL que utilice este mecanismo no cambie de un sistema a otro, como es el caso de MSVBVM60.DLL, la librería que utilizan todos los ejecutables generados por VISUAL BASIC 6.0. Cada ejecutable utiliza este mecanismo y le permite mayor velocidad al iniciar. Ahora si la información no concuerda y tiene que cargar las funciones no habrá mucha diferencia que como sucede en el import directory. El paso clave de este método es que el loader compruebe correctamente si la información que tiene obligada el ejecutable concuerda con la librería que tiene el sistema.

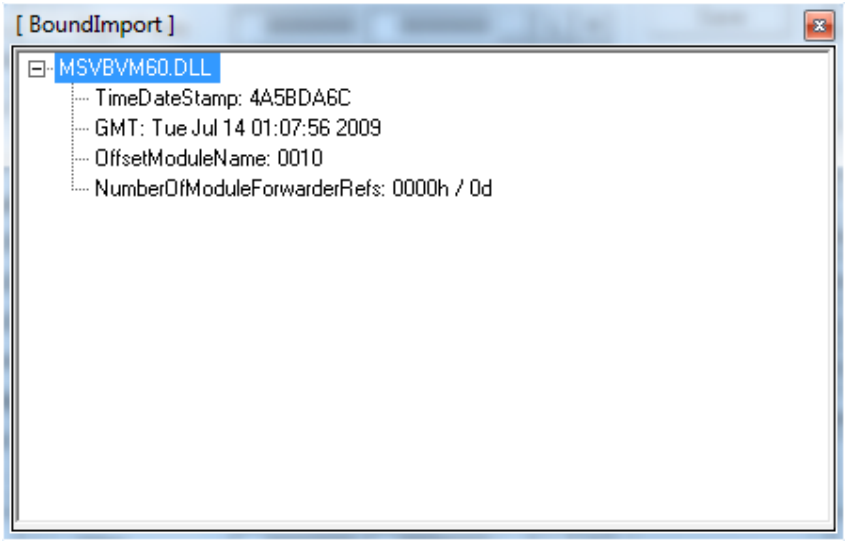
Este directory utiliza la siguiente estructura que sirve como base de información para comprobar si las direcciones son obsoletas:

IMAGE BOUND IMPORT DESCRIPTOR

CAMPO	TAMAÑO	DESCRIPCIÓN
TimeDateStamp	4	Contiene la hora y fecha de importación de la librería.
OffsetModuleName	2	Contiene el desplazamiento(no RVA) hacia el nombre de la librería.
NumberOfModulesForwarderRefs	Contiene el número de estructuras IMAGE_BOUND_FORWARED_REF	

	que es idéntico a esta estructura excepto porque el último campo (NumberOfModulesForwarderRefs) es reservado.

Explicándolo de manera simple un BOUND IMPORT DESCRIPTOR para cada librería importada es una matriz simple. Sin embargo cuando se enlaza una API de otra librería la validez de la DLL debe ser comprobado también. Por lo tanto las estructuras BOUND IMPORT DESCRIPTOR se intercalan con BOUND FORWARDER REF.
Un ejemplo, si usted ha enlazado HeapAlloc en KERNEL32.DLL que será remitido de RtlAllocateHeap. En el archivo ejecutable debería haber un BOUND IMPORT DESCRIPTOR para KERNEL32.DLL y un BOUND FORWARDER REF para NTDLL.DLL, inmediatamente seguidos estarían las otras estructuras de las demás importaciones.



DIRECCIONES REALES DE LAS API

```
07 05 A2 72 86 93 A3 72 F9 09 A3 72 EE 6A A4 72
37 05 A2 72 44 C2 A0 72 31 68 A4 72 74 8B A4 72
9B 6A A2 72 29 19 A2 72 62 72 A4 72 BA 02 A3 72
C3 9F A1 72 B7 70 A2 72 41 09 A3 72 76 6A A2 72
E5 76 A2 72 3A C3 A0 72 74 A2 A1 72 40 73 A4 72
6E 02 A3 72 40 39 A4 72 FE C1 A1 72 48 4A A2 72
F1 9F A1 72 9D 49 A2 72 32 D1 A1 72 06 03 A3 72
66 5E A3 72 08 A0 A1 72 06 04 A3 72 73 0D A2 72
```

Con esto terminamos y pasemos a las referencias.

REFERENCIAS

Este documento ha sido escrito utilizando como principal referencia la información oficial de Microsoft, su documento llamado PE/COFF el cual podemos denominarlo la biblia del formato portable executable.
La mayoría de los conceptos provienen de ahí, trate de darles forma y explicarlos lo más entendible posible.
Otra referencia que tome fue el documento escrito por Matt Pietrek llamado An In-Depth Look into the Win32 Portable Executable File Format y la información que saque fue la del BOUND IMPORT DIRECTORY, la cual no se encuentra documentada en la información oficial de Microsoft.

PE/COFF:
http://msdn.microsoft.com/en-us/windows/hardware/gg463119

An In-Depth Look into the Win32 Portable Executable File Format:
http://msdn.microsoft.com/en-us/magazine/cc301805.aspx

DESPEDIDA

Creo que aquí vengo a terminar el documento, me da mucho orgullo saber que pude realizarlo a pesar de que tardé más de lo que pensé pero traté de hacerlo lo mejor posible, teniendo en cuenta la poca información que hay clara acerca del tema en español.

Desde pequeño me enseñaron que siempre hay que ser agradecido y este tutorial se lo quiero dedicar a 3 personas en especial:

- Shaddy: Si no fuera por ti jamás existiera este documento ni tampoco yo conocería acerca del tema. Sos un gran tipo muchas gracias por toda tu ayuda.
- Hacker_Zero: Hey mi yo del futuro :P muchísimas gracias por cada opinión por cada aliento de ánimo, la ayuda y la gran amistad que me brindaste.
- Psymera: Se que no querías que publicase este documento pero discúlpame el fin que tenía era ayudar a muchos y creeme Hacker_Zero tiene razón quien quiera aprender que pase el trabajo leyéndolo, los que quieren todo fácil no lo van a encontrar aquí sin esforzarse.

Pero también quiero agradecer de manera enorme a todo el foro de h-sec, la lista de CracksLatinos y elhacker.net ya que de todos ellos he aprendido mucho.

Además agradezco sus consejos y opiniones a:

- Guan de dio (CracksLatinos) - RDGMax
- Khrane - Lord RNA
- Arkangel
- YST
- Queta
- Thor
- Karcrack
- Karmany
- Steve10120
- 79137913
- Trompetin17
- Tenacrk

Una disculpa a los que se me quedan y muchas gracias a todos y espero disfruten el documento, me despido.

Cualquier duda crítica, consulta o sugerencia será bien recibida.
MSN: the_swash@hotmail.es
Gtalk: jcuastumal@gmail.com
http://h-sec.org