Estudio de los Encabezados PE

Por SickTroen.

Parte 1

Introducción: Bueno, esto lo clasifico como una serie de tutoriales, que nos ayudara básicamente a saber, como editar manualmente, y como localizar los datos de nuestras estructuras dentro de estos archivos visualmente solo con un algún editor o visualizador hex.

Tratare de explicar lo necesario sin entrar mucho en detalles, estudiaremos las estructuras por "fuera", y si nos van a servir lo detallamos mas ;)

En esta primera parte estudiaremos todo acerca de los encabezados.
Luego iremos viendo como modificar manualmente algunos datos,
Agregar miembros a la import, export section, y pondremos algunos ejercicios y todo el tiempo ejemplos sobre todo lo que podamos hacer con nuestros ejecutables ;)

Cualquier sugerencia, idea, o quizás algún error en este Tutorial, me pueden contactar de 2 maneras:

Vía Web: www.CrackNFO.da.ru
Vía E-Mail: SickTroen@Sin-Nada.com.ar

Bibliografía:

- Microsoft Portable Executable and Common Object File Format Specification
- PE Tutorials by IcZelion
- Descabezando archivos ejecutables portables por nuMIT_or

Índice:

1.1 Introducción al Formato PE:

PE significa Portable Executable, es el formato utilizado por la mayoría de programas hechos para Win32 (Ej.: win95, winNT, etc.).
Como ya sabrán, los procesadores 8086, usan el Little Endian Byte Ordering, que consiste en que el byte de menos valor queda situado en la posición mas baja de la memoria y así el número queda invertido.

La estructura que posee que es la siguiente:

- Encabezado MZ EXE y DOS Stub:
  La cual contiene al principio del archivo las letras "MZ", que es un Standard bajo dos/windows, y os/2 también.
  Después nos encontraremos una pequeña estructura llamada DOS Stub, que en realidad es un programa ejecutable, que por ejemplo, si no estamos bajo windows nos dirá "This program cannot be run in DOS mode".

- Encabezado PE:
  Cuando ejecutamos un determinado programa, el sistema operativo carga el ejecutable, y si encuentra esta estructura, pasa por alto lo que seria el DOS Stub.
  Aquí vamos a encontrar información sobre en que tipo de maquinas corre el programa, que tipo de ejecutable es (Ej.: EXE, Librería) etc...

- Section Table:
  Básicamente nos dice información acerca de las secciones que tenemos en nuestro ejecutable.

- Secciones:

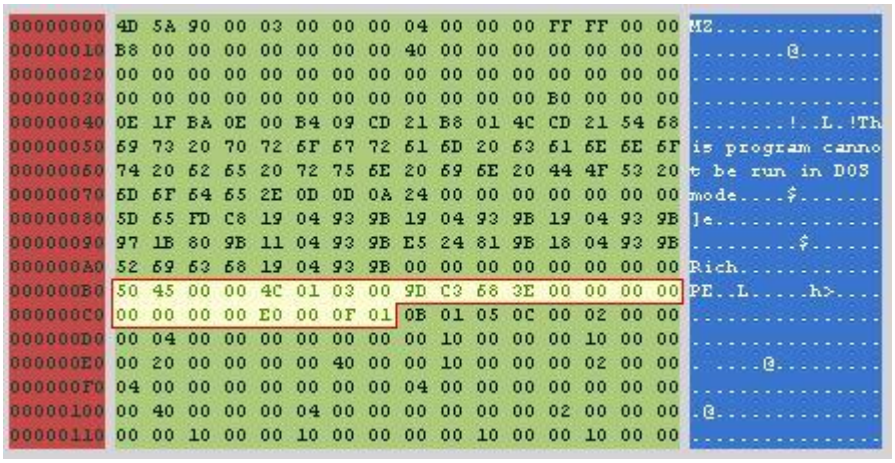
Por ultimo tenemos las secciones en si, que son las que contienen datos o códigos, lo que seria el cuerpo del archivo.

1.2 Explorando un ejecutable:

1.2.1 Encabezados PE.

Bueno, en este caso vamos a usar el “hi.exe” que trae este .zip.

Es un simple programa hecho en MASM, para que estudiemos de apoco como es el ejecutable a primera vista y como identificar sus estructuras, secciones, etc...



La estructura de este encabezado es la siguiente (usaremos todo el tiempo nuestro grafico):

```
IMAGE_NT_HEADERS STRUCT
  Signature dd ?
  FileHeader IMAGE_FILE_HEADER <>
  OptionalHeader IMAGE_OPTIONAL_HEADER32 <>
IMAGE_NT_HEADERS ENDS
```

Signature:

Lo que vemos aquí marcado en el dibujo, como se ve a simple vista empieza con las letras “PE” (50 45 00 00h).

FileHeader:

Offset	Tamaño	Nombre	Descripción
0	2	Machine	CPU Type: 386/486/Pentium/etc...
2	2	NumberOfSections	Cantidad de Secciones
4	4	TimeDateStamp	Fecha y Hora del archivo original
8	4	PointerToSymbolTable	Nos apunta hacia la tabla de símbolos.
12	4	NumberOfSymbols	Nos dice la cantidad de símbolos
16	2	SizeOfOptionalHeader	Tamaño del encabezado opcional
18	2	Characteristics	Flags que nos dicen los atributos del archivo.

Machine: Lo que vemos es 4C 01, esto nos dice para que tipo de cpu esta escrito.

Algunos Ejemplos: 01 4C = 80386
01 4D = 80486
01 4E = 80586

NumberOfSections: Luego nos encontramos con 03h, que esto nos dice el número de secciones que posee.

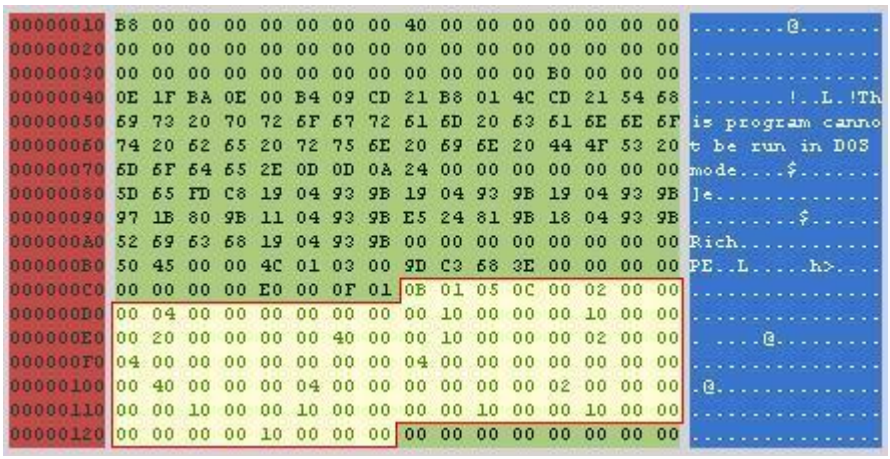
TimeDateStamp: Lo que sigue 9D C3 68 3E, es un Timestamp, no nos interesa mucho, pero tiene hora y fecha original del archivo.

PointerToSymbolTable y NumberOfSymbols: Los siguientes 8 ceros que vemos, son referentes a la tabla de símbolos, los primeros 4 bytes nos dicen en donde podemos encontrarla, y los 4 restantes nos dice la cantidad de símbolos que tenemos.

SizeOfOptionalHeader: Después vemos 0Eh, que es el tamaño del encabezado opcional + el data directories (224 bytes).

Characteristics: Y para terminar 010Fh, que nos dice sin entrar mucho en detalle que es ejecutable.

OptionalHeader: Es el ultimo miembro de la estructura, y contiene 31 datos, que son de la estructura lógica del archivo.



Posee campos Standartd , campos NT, y el data directories.
Analicemos que tenemos en nuestro archivo...

<u>Campos Starndart</u>:

- **Magic** 0B 01 ; esto nos dice que es un PE32, a diferencia del 0B 02 que es el PE32+ (estas limitan nuestra imagen hasta 4gb!).
- **MajorLinkerVersion** 05 ; Datos sobre la versión del linker.
- **MinorLinkerVersion** 0C ;
- **SizeOfCode** 0200h ; Nos dice el tamaño de la sección.
- **SizeOfInitializedData** 0400h ; Nos dice el tamaño de la sección de datos inicializados.
- **SizeOfUninitializedData** 00 00 00 00 ; Nos dice el tamaño de la sección de datos no inicializados.
- **AddressOfEntryPoint** 1000h; Es el RVA de la instrucción que va a ser ejecutada cuando ya este listo el programa para ser ejecutado.
- **BaseOfCode** 1000h ; Base de la sección de código.
- **BaseOfData** 2000h ; Base de la sección de datos.

Offset	Cabecera Opcional	Tamaño
18h	*Magic*	Word
1Ah	*MajorLinkerVersion*	Byte

1Bh	*MinorLinkerVersion*	Byte
1Ch	*SizeOfCode*	dword
20h	*SizeOfInitializedData*	dword
24h	*SizeOfUninitializedData*	dword
28h	*AddressOfEntryPoint*	dword
2Ch	*BaseOfCode*	dword
30h	*BaseOfData*	dword

Campos Adicionales NT:

- **ImageBase** 400000h ; Base de la Imagen - inicio de la imagen en la memoria virtual.
- **SectionAlignment** 1000h ; Alineamiento de las secciones para cuando son cargadas en memoria (potencia de 2 entre 512 y 256M)
- **FileAlignment** 200h ; Nos dice básicamente cada cuanto vamos a encontrar una sección (potencia de 2 entre 512 y 64k).
  Por Ej.: nosotros tenemos 02 00h (512) para cada sección, supongamos que nosotros tenemos X sección que comienza a los 400h, para ir a la siguiente ya sabemos donde encontrarla (400h + 200h), entonces vamos al 600h, y nos encontramos con la siguiente sección. Sabiendo esto nos será facilísimo encontrarlas manualmente ;)
  NOTA: si nuestra sección X que era 400h, solo usa 100h, el resto hasta llegar a 200h serán datos que no se van a usar, y que no están definidos.
- **MajorOperatingSystemVersion** 04h ;
- **MinorOperatingSystemVersion** 00 00 ; Estos nos dice entre que versión de OS puede ejecutarse el programa básicamente.
- **MajorImageVersion** 00 00
- **MinorImageVersion** 00 00
- **MajorSubsystemVersion** 04h ;
- **MinorSubsystemVersion** 00 00 ; Versión del subsistema Win32, por Ej.: si no tenemos una versión 4.0, los dialogs no tendrán aspecto 3D.
- **Reserved** 00 00 00 00 ;
- **SizeOfImage** 4000h ; Es la suma total de todas las encabezadoss y secciones (alineadas con SectionAlignment), la nuestra por Ej. Es 4000h, a cargar en memoria.
- **SizeOfHeaders** 400h ; Es la suma de todos los encabezados (Dos Stub + PE Headers + Section Headers).
- **CheckSum** 00 00 00 00 ; Suma de chequeo del archive.
- **Subsystem** 02h ; Subsystem de WinNT.

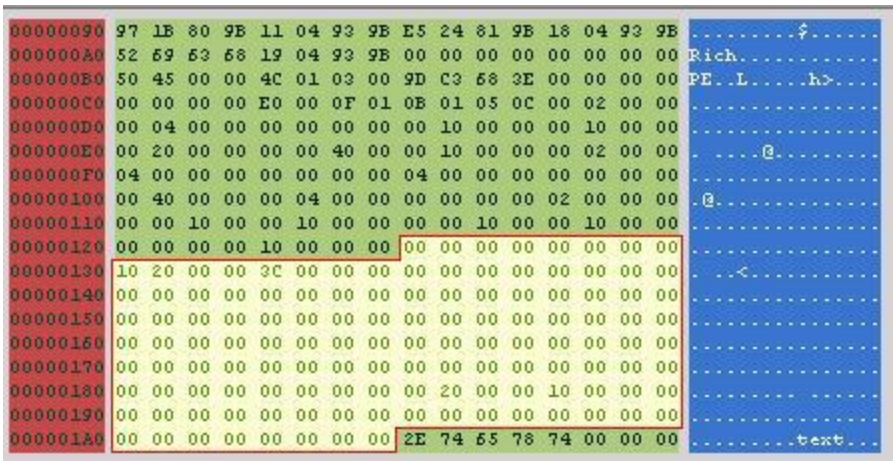
Algunos son: 00 00 : Subsistema Desconocido
00 01 : Usado para Drivers, y procesos Nativos.
00 02 : Windows GUI
00 03 : Consola de Windows

- **DLL Characteristics** 00 00 ; Flag especial para las DLL's
- **SizeOfStackReserve** 100000h ; Memoria reservada para la pila (stack)
- **SizeOfStackCommit** 1000h ; Memoria comprometida para la pila.
- **SizeOfHeapReserve** 100000h ; Tamaño a reservar para el montículo (o heap).
- **SizeOfHeapCommit** 1000h ; Memoria comprometida para el montículo o heap.
- **LoaderFlags** 00 00 00 00 ; Se usa?? ☺
- **NumberOfRvaAndSizes** 10h ; Recuerdan que el SizeOfOptionalHeader era la suma del optional header + el data directories?, bueno los que nos dice acá es el valor del data directories.
  Nuestro valor 10h es igual a 16d, que es el tamaño del directorio de datos, y sabiendo que el contenido de cada uno es de 8 bytes, lo multiplicamos(8 * 16 = 128d), y obtenemos el tamaño real.

offset	Campos Adicionales NT	Tamaño
34h	*ImageBase*	*DWORD*
38h	*SectionAlignment*	*DWORD*
3Ch	*FileAlignment*	*DWORD*

40h	*MajorOperatingSystemVersion*	*WORD*
42h	*MinorOperatingSystemVersion*	*WORD*
44h	*MajorImageVersion*	*WORD*
46h	*MinorImageVersion*	*WORD*
48h	*MajorSubsystemVersion*	*WORD*
4Ah	*MinorSubsystemVersion*	*WORD*
4Ch	*Reserved*	*WORD*
50h	*SizeOfImage*	*DWORD*
54h	*SizeOfHeaders*	*DWORD*
58h	*CheckSum*	*DWORD*
5Ch	*Subsystem*	*WORD*
5Eh	*DLL Characteristics*	*WORD*
60h	*SizeOfStackReserve*	*DWORD*
64h	*SizeOfStackCommit*	*DWORD*
68h	*SizeOfHeapReserve*	*DWORD*
6Ch	*SizeOfHeapCommit*	*DWORD*
70H	*LoaderFlags*	*DWORD*
74h	*NumberOfRvaAndSizes*	*DWORD*

Data Directories:



Esta estructura nos dice el RVA y el tamaño sobre los elementos de la tabla.
Los 8 bytes están divididos: 4 para el RVA, y 4 para el Tamaño.
Mas adelante cuando estemos sobre la Tabla de Secciones explicaremos como sacar la dirección “física” en nuestro archivo de estos elementos.

```
typedef struct _IMAGE_DATA_DIRECTORY {
    DWORD   RVA;
    DWORD   Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

- **Export Table** 00 00 00 00 00 00 00 00 ; Tabla de exportaciones.
- **Import Table** 10 20 00 00 3C 00 00 00 ; Tabla de importaciones.
- **Resource Table** 00 00 00 00 00 00 00 00 ; Tabla de recursos.
- **Exception Table** 00 00 00 00 00 00 00 00 ; Tabla de excepciones.
- **Certificate Table** 00 00 00 00 00 00 00 00 ; Tabla de Certificados.
- **Base Relocation Table** 00 00 00 00 00 00 00 00 ;
- **Debug** 00 00 00 00 00 00 00 00 ; Debug Information
- **Architecture** 00 00 00 00 00 00 00 00 ;
- **Global Ptr** 00 00 00 00 00 00 00 00 ; RVA del resultado a ser guardado, el tamaño debe ser 0.

- **TLS Table** 00 00 00 00 00 00 00 00 ; Thread Local Storage.
- **Load Config Table** 00 00 00 00 00 00 00 00 ;
- **Bound Import** 00 00 00 00 00 00 00 00 ;
- **IAT** 00 20 00 00 10 00 00 00; Import Address Table.
- **Delay Import Descriptor** 00 00 00 00 00 00 00 00;

- **COM+ Runtime Header** 00 00 00 00 00 00 00 00;
- **Reserved** 00 00 00 00 00 00 00 00;

1.2.3 Tabla de Secciones:

Creo que no hace falta aclarar que nos da información sobre nuestras secciones.. ;)

```
IMAGE_SIZEOF_SHORT_NAME equ 8
IMAGE_SECTION_HEADER STRUCT
  Name1 db IMAGE_SIZEOF_SHORT_NAME dup(?)  (8 bytes)
  union Misc
    PhysicalAddress dd ?
    VirtualSize dd ?
  ends
  VirtualAddress dd ?
  SizeOfRawData dd ?
  PointerToRawData dd ?
  PointerToRelocations dd ?
  PointerToLinenumbers dd ?
  NumberOfRelocations dw ?
  NumberOfLinenumbers dw ?
  Characteristics dd ?
IMAGE_SECTION_HEADER ENDS
```

```
00000180h: 00 00 00 00 00 00 00 00 00 20 00 00 10 00 00 00 ; ......... ......
00000190h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001a0h: 00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00 ; .........text...
000001b0h: 26 00 00 00 00 10 00 00 00 02 00 00 00 04 00 00 ; &...............
000001c0h: 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60 ; ............ ..`
000001d0h: 2E 72 64 61 74 61 00 00 92 00 00 00 00 20 00 00 ; .rdata..'.... ..
000001e0h: 00 02 00 00 00 06 00 00 00 00 00 00 00 00 00 00 ; ................
000001f0h: 00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00 ; ....@..@.data...
00000200h: 1B 00 00 00 00 30 00 00 00 02 00 00 00 08 00 00 ; .....0..........
00000210h: 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 C0 ; ............@..À
```

Inmediatamente después de que termina el Data Directories, nos encontramos con la tabla de secciones.
Cada entrada a esta sección consta de 40 bytes (28h),, así que pueden ver que empieza con un ".text", contamos 40 bytes (28h), y tenemos ".rdata", otros 40 bytes (28h), mas y tenemos .data ;)

Veamos nuestra estructura como ejemplo:

- **Name** 2E 74 65 78 74 00 00 00 ; Nombre de la sección, en nuestro caso .text
- **VirtualSize** 26 00 00 00 ; Tamaño que ocupa la sección para cuando se carge en memoria.

- **VirtualAddress** 00 10 00 00 ; Es el RVA de la sección, en nuestro caso 1000h
- **SizeOfRawData** 00 02 00 00 ; De aquí el OS lee la cantidad de bytes de esta sección para asignar en memoria.
- **PointerToRawData** 00 04 00 00 ; Nos dice en que dirección podemos ubicar los datos de esta sección.
- **PointerToRelocations** 00 00 00 00 ; Casi siempre esta en 0.
- **PointerToLinenumbers** 00 00 00 00 ;
- **NumberOfRelocations** 00 00 ;
- **NumberOfLinenumbers** 00 00 ;
- **Characteristics** 20 00 00 60 ;

Equivalencias:

· 000000020h __Código.
· 000000040h __Datos inicializados.
· 000000080h __Datos no inicializados.
· 040000000h __Sección cacheable.
· 080000000h __Sección paginable.
· 100000000h __Sección compartida.
· 200000000h __Ejecutable.
· 400000000h __Se puede leer.
· 800000000h __Se puede escribir en la sección.

En este caso la nuestra seria 60 00 00 20, ósea:

000000020h (Código).
\+ 200000000h (Ejecutable).
\+ 400000000h (Se puede leer).
= 600000020h

offset	Tabla de Secciones	Tamaño
00h	Name Nombre de la seccion	**QWord**
08h	VirtualSize: Tamaño en memoria	**DWord**
0Ch	VirtualAddress: Direccion base en memoria	**DWord**
10h	SizeOfRawData: Tamaño en el archivo	**DWord**
14h	PointerToRawData: Direccion base en el archivo	**DWord**
18h	PointerToRelocations	**DWord**
1Ch	PointerToLinenumbers	**DWord**
20h	NumberOfRelocations	**Word**
22h	NumberOfLinenumbers	**Word**
24h	Characteristics	**DWord**

Tamaño por cabecera de sección =

Como saber cual es la verdadera posición física en el ejecutable de una sección?

Para saber la dirección real en el ejecutable, tenemos que restar el RVA de la X tabla (por Ej. Import), a el RVA de la X Sección (en este caso .rdata), y a esto sumarle el PointerToRawData de la sección.

RVA .rdata 2000h
RVA I.T. 2010h (import table)
RAW .rdata 600h
IT RAW = 610h

Pruébenlo ustedes mismo y van a ver que van directo al grano ;)

Para la Próxima: Import y Export Sections, y quizás ya empezamos con algunas practicas #)

Saludos,
SickTroen.