Sistemas Operativos

Ejecutables - II
PE

Sistemas Operativos

Versión: Mayo 2019

- ☑ Palabras Claves: Linux, Windows, PE, Ejecutable, ELF, Linking, Carga Dinámica, Proceso, Fuente

Generalidades

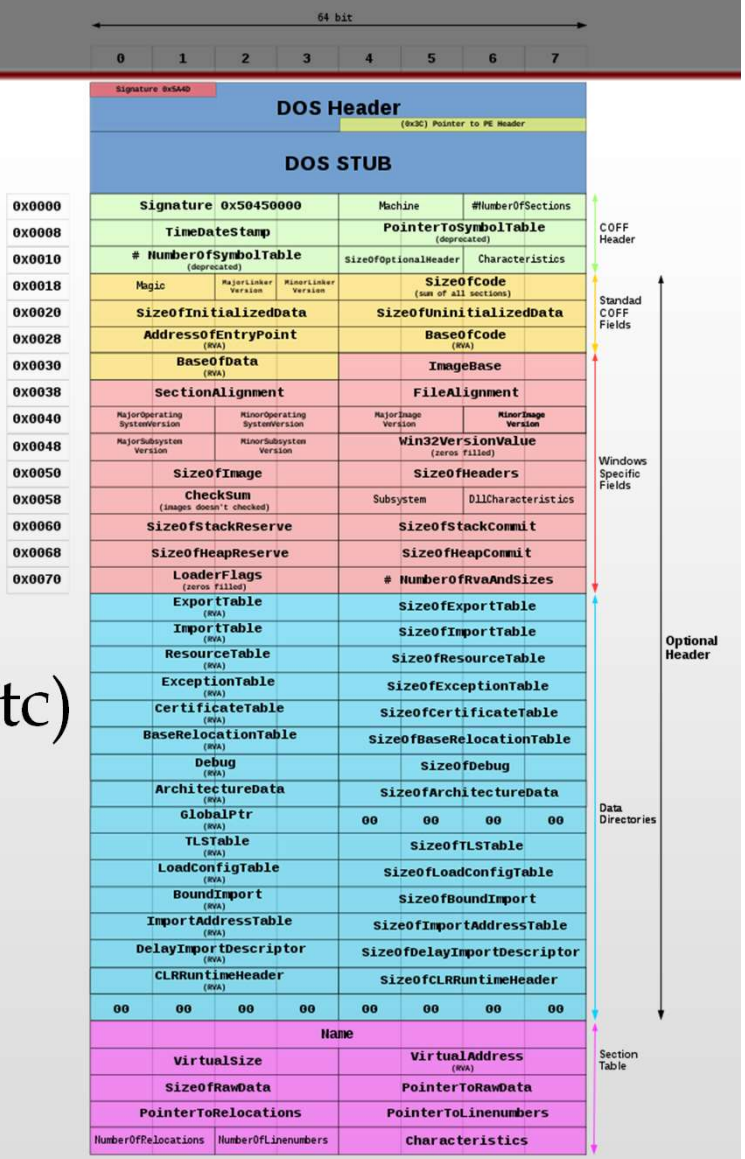
- ☑ PE : Portable Ejecutable
- ☑ Fue desarrollado por Microsoft junto con la especificación de Win32
- ☑ Derivan del formato COFF (Common Object File Format) de VAX/VMS (DEC - Digital Equipment Corporation)
- ☑ Es utilizado en las diferentes versiones de Windows: Desde Windows NT hasta las versiones actuales, incluso en WinCE, Xbox, .NET, etc.

Capacidades de PE

- ☑ dynamic linking
- ☑ dynamic loading
- ☑ Facilita la creación de librerías compartidas (shared libraries)
  - ✓ DLL, SYS, OBJ, OCX, CPL, etc
- ☑ La versión para 64 bits se la denomina PE32+
  - ✓ Varían campos de las estructuras (de 32 a 64 bits)

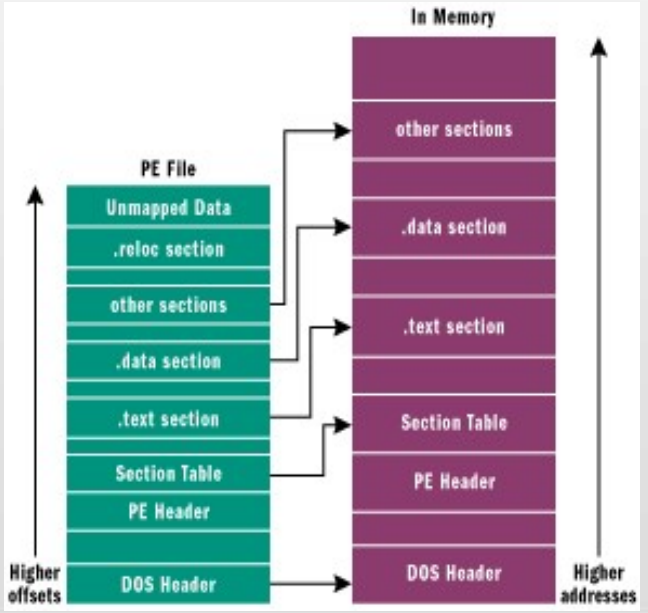
Su representación

- La información que contiene está representada por un conjunto de estructuras de datos
- Estas estructuras son las mismas que se utilizan para llevar el archivo a la memoria
- Su estructura (tipos, constantes, etc) se encuentran definidos en WINNT.H (Archivo de cabeceras/headers)
  https://docs.microsoft.com/en-us/windows/desktop/api/winnt/



El formato

- ☑ Está compuesto por un conjunto de headers y secciones que indican al Loader como cargar el archivo en la Memoria
- ☑ Cada sección contiene un conjunto de información diferente así como un conjunto de niveles de protección diferentes al ser cargadas en la memoria
  - ✓ Cada sección se almacena en una página. A través de ello podemos asignar atributos (R,W,X) ← se hace en tiempo de ejecución



Secciones

- ☑ División Lógica dentro del archivo que contiene código o datos
- ☑ Los datos no son solo variables del programa
  - ✓ Incluyen también información que indica como "trabajar" con el archivo
- ☑ Cada sección tiene:
  - ✓ Un nombre que describe su contenido.
    El nombre no es tenido en cuenta por el SO
  - ✓ Un tipo
  - ✓ Atributos (R,W,X)

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
      DWORD PhysicalAddress;
      DWORD VirtualSize;
  } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

Secciones (cont.)

- ☑ Cada sección posee dos direcciones:
  - ✓ Una en el archivo (donde se encuentra)
  - ✓ Una dirección virtual (donde se cargara en el espacio de usuario)

Donde comienza la sección dentro del archivo PE

```
  01 .text      VirtSize: 00074658  VirtAddr:  00001000
    raw data offs:   00000400  raw data size: 00074800
...
  02 .data      VirtSize: 000028CA  VirtAddr:  00076000
    raw data offs:   00074C00  raw data size: 00002400
```

Desplazamiento a partir del cual debe ser cargada en memoria

- ✓ Una vez cargado en la memoria el archivo PE, el comienzo de cada sección debe corresponderse con el comienzo de una página (facilita la definición de atributos)

RVAs

- RVA: Relative Virtual Address
- Cada archivo tiene un dirección "preferida" de carga en el espacio de direcciones (Virtual):
  - No es posible garantizar que el PE sea cargado en "esa dirección"
  - Para solucionar el inconveniente se utilizan RVAs con el fin de direccionar elementos del archivo

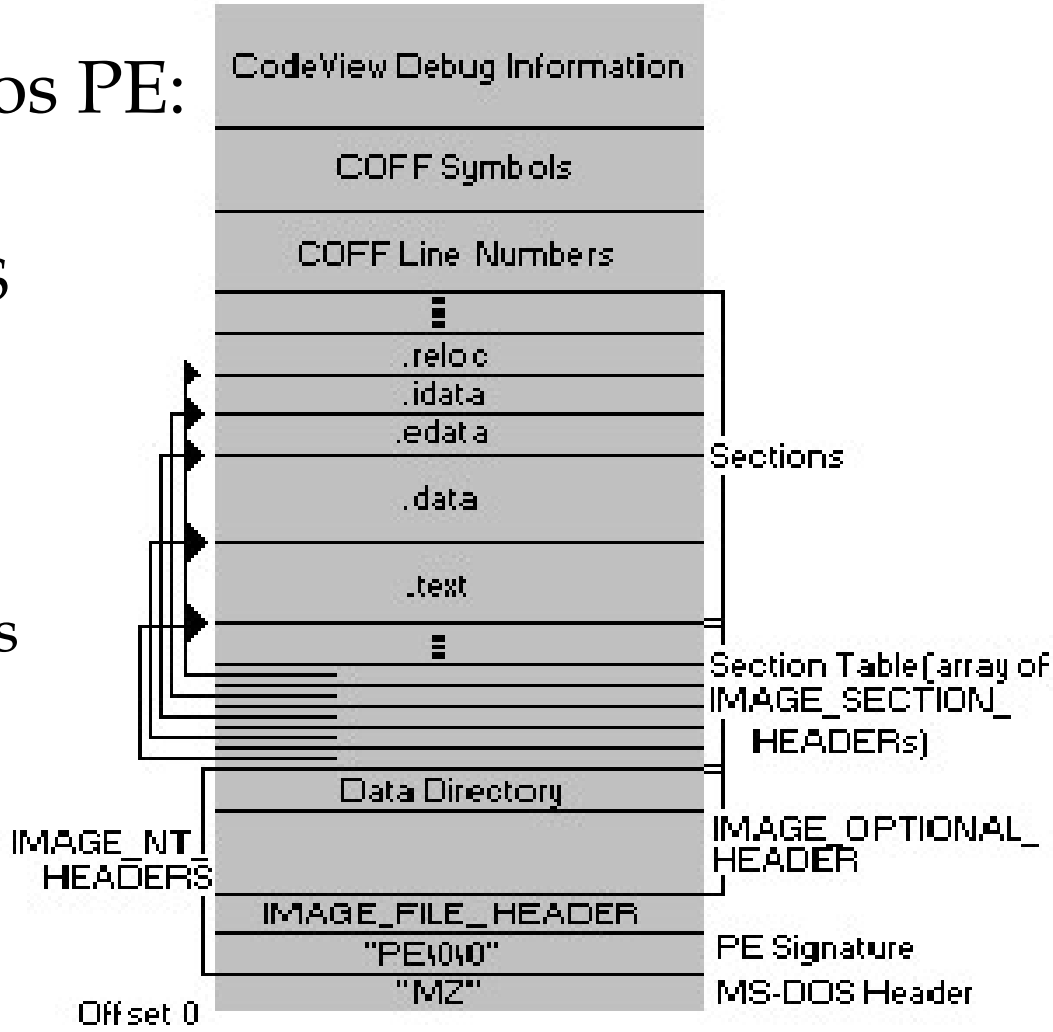
RVAs (cont.)

- Una RVA es un desplazamiento relativo en memoria a la dirección de inicio donde el contenido del archivo PE es cargado en la Memoria Virtual
- Por ejemplo:
  - Archivo cargado a partir de la dirección 0x400000
  - Una variable se encuentra en la dirección 0x401000
  - La RVA de la variable será 0x1000

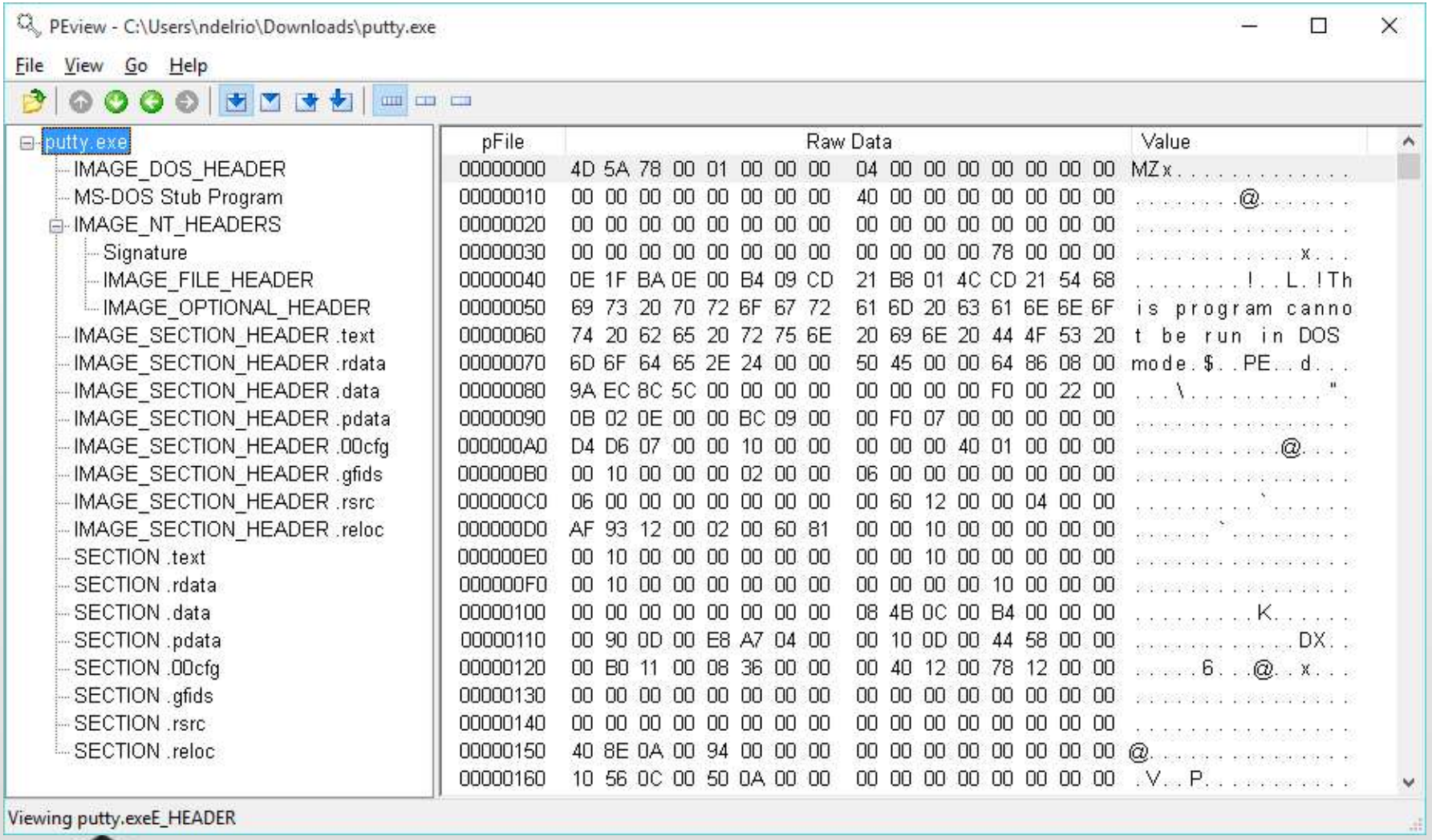
(target address) 0x401000 - (load address)0x400000 = (RVA)0x1000.

La estructura

- ☑ Formato de los archivos PE:
  - ☑ Header MS-DOS
  - ☑ IMAGE_NT_HEADERS
  - ☑ Section Table
  - ☑ Sections
- ☑ Las secciones y estructuras en 32 y 64 bits son similares



Análisis de archivos PE



Header MS-DOS

- Todo archivo PE comienza con un pequeño programa MS-DOS ejecutable:
  - Quedo por compatibilidad
  - Lo único que haces es imprimir:
    - This program cannot run in DOS mode
- Entre sus campos hay 2 importantes (distinguen que el archivo es un PE)
  - e_magic: 0x5A4D (En ASCII “MZ” Mark Zbikowski su creador)
  - e_lfanew: Desplazamiento dentro del archivo de la estructura IMAGE_NT_HEADERS

IMAGE_NT_HADERS

- ☑ Especifica los detalles del contenido del archivo.
- ☑ Existe una versión de 32 bits y otra de 64.

```
typedef struct _IMAGE_NT_HEADERS {
DWORD Signature;
IMAGE_FILE_HEADER FileHeader;
IMAGE_OPTIONAL_HEADER OptionalHeader;
}
```

- ✓ Signature: 0x00004550 (PE00 en ASCII)
- ✓ FileHeader: Lo analizamos a continuación
- ✓ OptionalHeader:Lo analizamos a continuación

IMAGE_NT_HADERS- FileHeader

- ☑ Tipo: IMAGE_FILE_HEADER
- ☑ Contiene Información básica sobre el archivo

Campo	Descripción

Machine	Tipo de CPU para la que se compilo el archivo. Por ejemplo:
	`IMAGE_FILE_MACHINE_I386  0x014c // Intel 386`
	`IMAGE_FILE_MACHINE_IA64  0x0200 // Intel 64`

NumberOfSections	Indica la cantidad de secciones en la tabla de secciones. La tabla de secciones se encuentra luego de IMAGE_NT_HEADERS
TimeDateStamp	Tiempo en el que el archivo fue creado.
PointerToSymbolTable	Desplazamiento a la tabla de símbolos COFF.
NumberOfSymbols	Cantidad de símbolos en la tabla de símbolos COFF.
SizeOfOptionalHeader	Tamaño del header de datos opcionales que se encuentra luego de la estructura IMAGE_FILE_HEADER
Characteristics	Un conjunto de flags (bits) que indican atributos en el archivo. Por ejemplo si es ejecutable o no, si es para 32 bits, etc

IMAGE_NT_HADERS- OptionalHeader

- ☑ Tipo: IMAGE_OPTIONAL_HEADER
- ☑ Divide su información en 3 grandes grupos
- ☑ Si bien su nombre dice que es opcional, el mismo está presente en la mayoría de los archivos
  - ✓ Puede no estar en las DLLs, si en los ejecutables

1 Campo	Descripción
Magic	Una firma indicando el tipo de header. Los valores mas usados son: IMAGE_NT_OPTIONAL_HDR32_MAGIC = 0x10b (PE32) IMAGE_NT_OPTIONAL_HDR64_MAGIC = 0x20b (PE32+)
MajorLinkerVersion	Versión mayor del compilador utilizado
MinorLinkerVersion	Versión menor del compilador utilizado
SizeOfCode	La suma del tamaño de todas las secciones de código
SizeOfInitializedData	La suma del tamaño de todas las secciones de datos inicializados
SizeOfUninitializedData	La suma del tamaño de todas las secciones de datos no inicializados
AddressOfEntryPoint	La RVA del primer byte de código en el archivo. Para las DLL este valor puede ser 0.
BaseOfCode	La RVA del primer byte de código cuando es cargado a memoria.
BaseOfData	La RVA del primer byte de los datos cuando estos son cargados en memoria. Este campo no esta presente en la versión de 64 bits (PE32+).

IMAGE_NT_HADERS- OptionalHeader (cont.)

2 Campo	Descripción
ImageBase	La dirección preferida de carga del archivo en la memoria. Si el cargador logra colocarlo en está dirección, no es necesario realizar una re-ubicación (veremos mas adelante está situación). Los valores por defecto son: para EXEs 0x400000 y para DLLs 0x10000000.
MajorOperatingSystemVersion	El número de versión mayor del Sistema Operativo Requerido.
MinorOperatingSystemVersion	El número de versión menor del Sistema Operativo Requerido.
SizeOfHeaders	La suma de los tamaños del header MS-DOS, el header PE y la tabla se secciones.

Subsystem	Un valor que representa el sub-sistema necesario. Algunos valor posibles son:
	`IMAGE_SUBSYSTEM_UNKNOWN, IMAGE_SUBSYSTEM_NATIVE,`
	`IMAGE_SUBSYSTEM_WINDOWS_GUI, IMAGE_SUBSYSTEM_WINDOWS_CUI,`
	`IMAGE_SUBSYSTEM_WINDOWS_CE_GUI, IMAGE_SUBSYSTEM_XBOX`

SizeOfStackReserve	El tamaño máximo reservado de la pila (stack) del thread inicial.
NumberOfRvaAndSizes	La cantidad de entradas que existen en el directorio de datos que continúa a este grupo de campos del IMAGE_OPTIONAL_HEADER. Actualmente contiene el valor 16.

IMAGE_NT_HADERS- OptionalHeader (cont.)

3 ☑ La sección final es un arreglo de 16 elementos del tipo IMAGE_DATA_DIRECTORY:

- ✓ Contiene direcciones a ubicaciones importantes del archivo (Funciones Importadas, Expotadas,etc)
- ✓ Su definición:

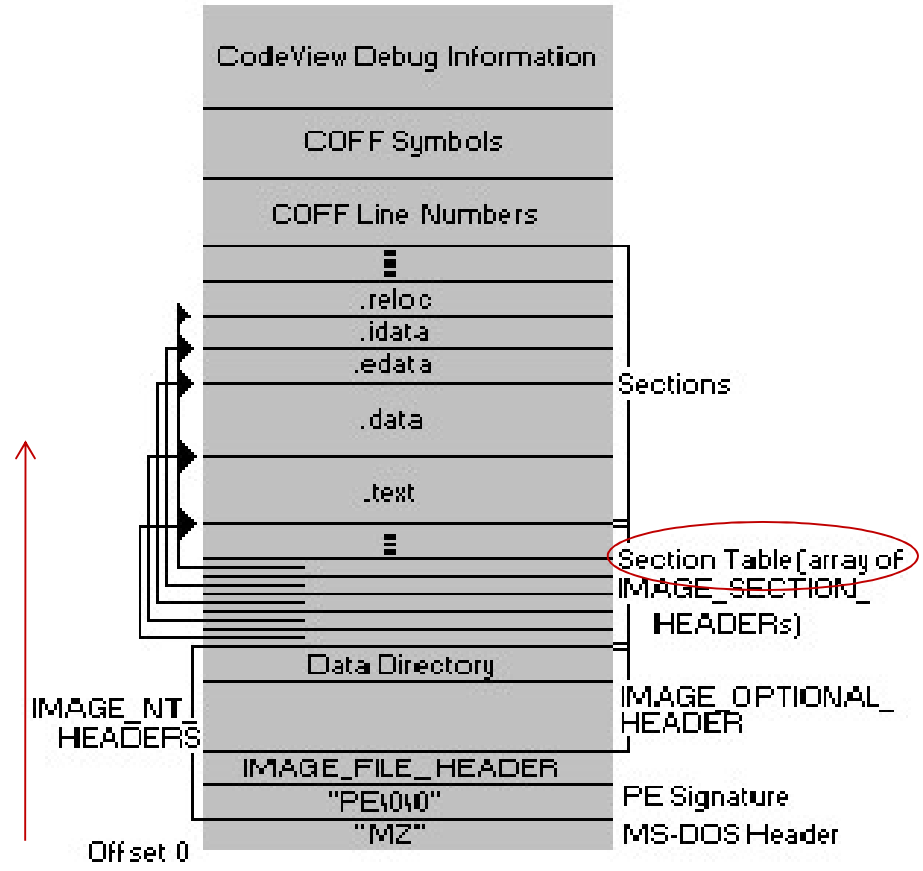
```
#define IMAGE_NUMBEROF_DIRECTORY_ENTRIES 16

IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
```

- ✓ El Data Directory

```
typedef struct _IMAGE_DATA_DIRECTORY {
 DWORD   VirtualAddress;     // RVA de los datos
 DWORD   Size;               // Tamaño de los datos
} IMAGE_DATA_DIRECTORY;
```

Section Table

- ☑ Es un arreglo de estructuras del tipo IMAGE_SECTION_HEADER
- ☑ Describe todas las secciones del archivo
  - ✓ Ubicación, tamaño, flags
- ☑ La cantidad de entradas se encuentra especificada en IMAGE_NT_HEADER (campo NumberOfSections)



Section Table (cont.)

☑ Campos de IMAGE_SECTION_HEADER

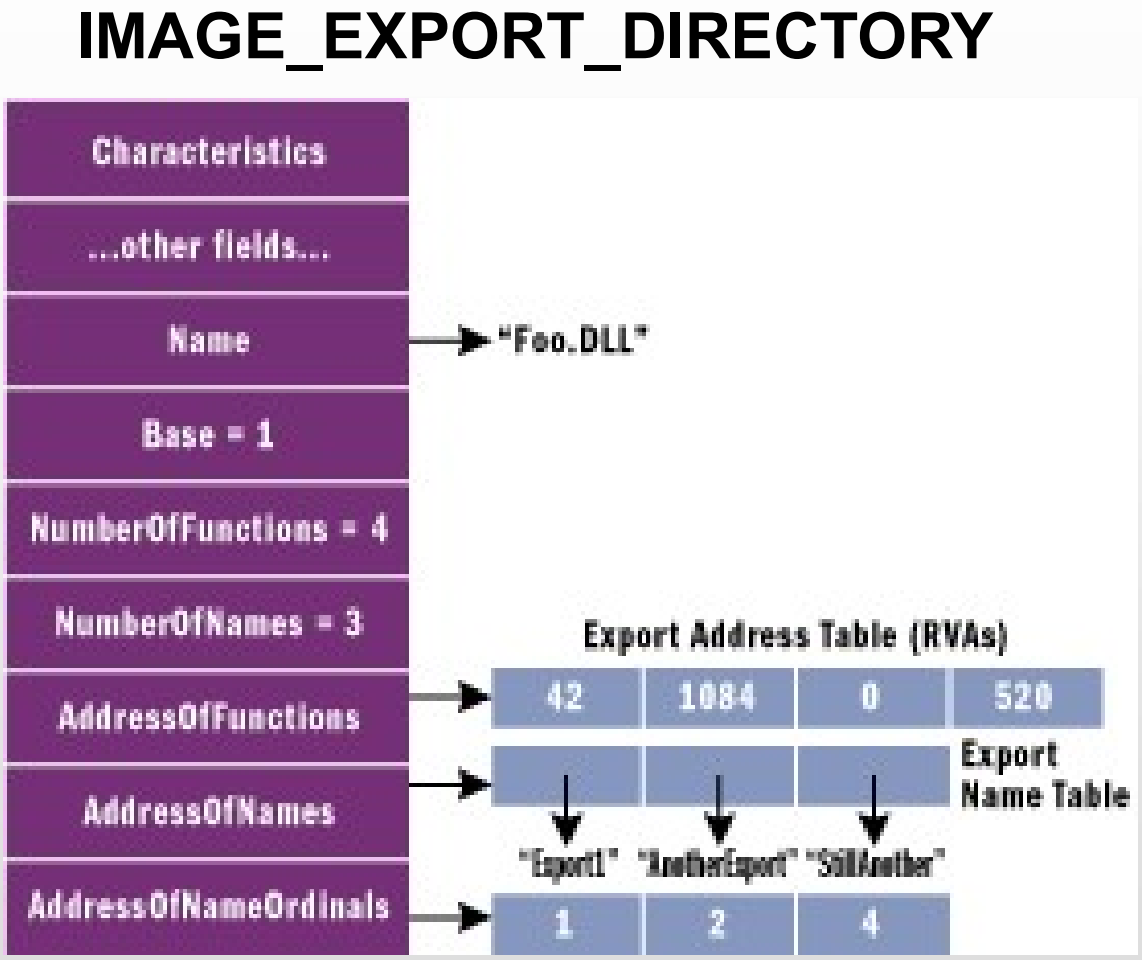
Campo	Descripción
Name[8]	Nombre de la sección.
VirtualSize	Indica el tamaño de sección.
VirtualAddress	Indica la RVA donde la sección comienza en la memoria.
SizeOfRawData	El tamaño (en bytes) de datos para la sección.
PointerToRawData	El desplazamiento en el archivo donde los datos para la sección se encuentran.
PointerToRelocations	El desplazamiento para las reubicaciones para la sección. Es 0 en los ejecutables.
PointerToLinenumbers	El desplazamiento en el archivo para los numeros de línea
NumberOfRelocations	La cantidad de reubicaciones a las que apunta el campo PointerToRelocations. Es 0 en los ejecutables.
NumberOfLinenumbers	La cantidad de entradas a las que apunta el campo NumberOfRelocations.
Characteristics	Conjunto de flags (unidos por un OR) que indican atributos para la sección. (Si es código, si se puede escribir, si se puede leer, etc).

Secciones Importantes - Exports

- Contiene información de códigos o datos que pueden ser exportados desde el archivo
  - Se trata de nombres de variables o funciones que pueden ser referenciadas desde otros archivos PE
- Los elementos que se exportan se denominan "Símbolos" y generalmente se los accede a través de su nombre
- Se mantiene
  - Un nombre del símbolo
  - Un número único que lo identifica
  - Dirección del símbolo dentro del archivo

Secciones Importantes - Exports (cont.)

- Para representar los símbolos expotados se usa un arreglo de estructuras:

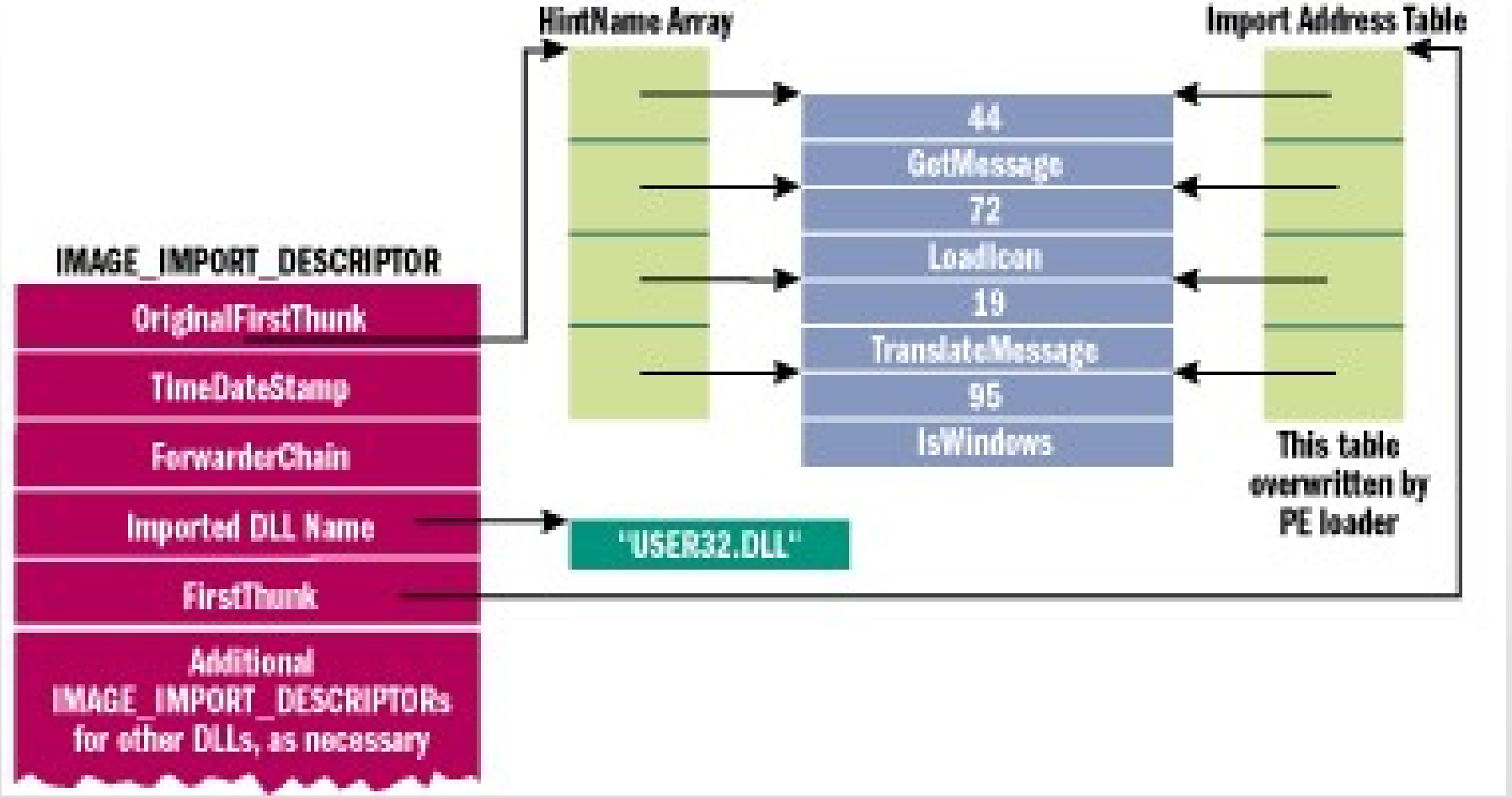


Secciones Importantes - Imports

- Símbolos importados desde otro archivo
  - Funciones, variables, etc.
- La sección es un arreglo de estructuras
  - IMAGE_IMPORT_DESCRIPTOR

Campo	Descripción
OriginalFirstThunk	Contiene la dirección RVA a la tabla de nombres importados (INT – Import Name Table). Esta tabla es un arreglo de estructuras IMAGE_THUNK_DATA. Cuando este valor es 0 se indica que es el final de la tabla de IMAGE_IMPORT_DESCRIPTORs.
TimeDateStamp	Este valor es cero si el ejecutable no se encuentra ligado a la DLL.
ForwarderChain	Es el índice a la primera función redireccionada.
Name	La dirección RVA al nombre del archivo importado.
FirstThunk	La dirección RVA de la tabla de direcciones importadas (IAT -Import Address Table). Esta tabla es un arreglo de estructuras IMAGE_THUNK_DATA.

Secciones Importantes - Imports (cont.)



Secciones Importantes - Resources

- Almacena información de iconos, bitmaps, ventanas de dialogo, etc.
  - Recursos
- Generalmente se almacenan en una sección llamada ".rsrc"
- Las estructuras utilizadas para representar los recursos permite armar una jerarquía similar a un sistema de archivos con directorios y archivos

Secciones Importantes - Base Relocatios

- ☑ Las direcciones (virtuales) que figuran en un archivo PE solo son validas si el archivo puede ser cargado en su "Dirección Preferida" (Se encuentra en la estructura IMAGE_FILE_HEADER)
- ☑ Esto no sucede con RVAs (recordemos que estas son siempre relativas)
- ☑ Esta sección contiene información para el loader
- ☑ Indica que direcciones deben ser modificadas cuando el archivo es cargado en una dirección que no es su "preferida"

Material Adicional

- http://www.codeproject.com/Articles/36928/Parse-a-PE-EXE-DLL-OCX-Files-and-New-Dependency-Wa
- http://msdn.microsoft.com/en-us/library/ms809762.aspx