Estudio de los Encabezados PE

Por SickTroen.

Parte 2

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

Índice:

Bueno esta es la 2da parte, y por ahora la ultima.

Nota sobre la sección .rsrc, que es la quedo fuera de estos tutos, les recomiendo que lean la 3era parte de los tutoriales de numit_or (Descabezando archivos ejecutables portables), que da una excelente explicación sobre esta sección.

Bibliografía:

- Microsoft Portable Executable and Common Object File Format Specification
- PE Tutorials by IcZelion
- Descabezando archivos ejecutables portables por nuMIT_or
- Peering Inside the PE: A Tour of the Win32 Portable Executable File Format by Matt Pietrek.

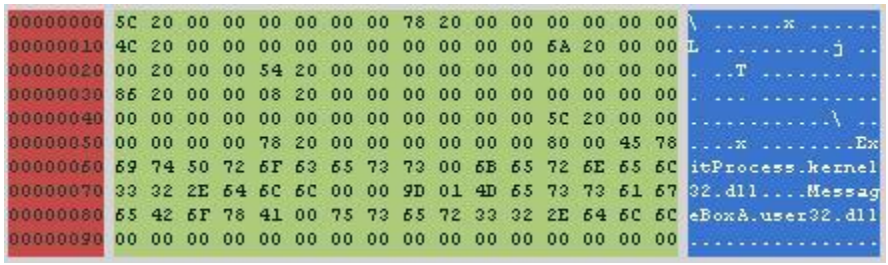
1.2.3 Algunos tipos de Secciones:

Esta pequeña tabla que sacamos del PE-COFF de MS, nos dice algunos tipos de secciones con las que nos podemos encontrar.
Son secciones genéricas, por llamarlas de alguna manera, también pueden encontrar otras secciones, como cuando por ejemplo el ejecutable esta comprimido, alguna sección tipo .upx.

Section Name	**Content**	**Characteristics**
.arch	Alpha architecture information	IMAGE_SCN_MEM_READ | IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_ALIGN_8BYTES | IMAGE_SCN_MEM_DISCARDABLE
.bss	Uninitialized data	IMAGE_SCN_CNT_UNINITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.data	Initialized data	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.edata	Export tables	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ
.idata	Import tables	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.pdata	Exception information	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ
.rdata	Read-only initialized data	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ
.reloc	Image relocations	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_DISCARDABLE
.rsrc	Resource directory	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.text	Executable code	IMAGE_SCN_CNT_CODE | IMAGE_SCN_MEM_EXECUTE | IIMAGE_SCN_MEM_READ
.tls	Thread-local storage	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.xdata	Exception information	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ

1.2.4 Tabla de Importaciones:

Podemos decir básicamente, que es una tabla donde nos dice que funciones usa el programa, que no son suyas, y desde que DLL se utilizan.




```
IMAGE_IMPORT_DESCRIPTOR STRUCT  (20 bytes)
  union
    Characteristics dd ?          ; (Original FirstThunk), nos apunta a un RVA,
    OriginalFirstThunk dd ?       ; donde nos dice donde encontrar el import X.
  ends
  TimeDateStamp dd ?              ; Generalmente = 0
  ForwarderChain dd ?             ; Nos habla de que la función que cargamos desde un dll
                                  utiliza "partes" de otro dll.
                                  Lamentablemente esto, no está documentado.
  Name1 dd ?                      ; RVA nombre del DLL
  FirstThunk dd ?                 ; Es lo mismo  que el OriginalFirstThunk, nada mas que
                                  apunta a otro array, pero que contiene lo mismo que el
                                  OriginalFirstThunk, es un duplicado.
IMAGE_IMPORT_DESCRIPTOR ENDS
```

En nuestro caso sabemos que nuestra Import Table esta en 610h, entonces aplicando la estructura que acabamos de ver:

```
OriginalFirstThunk  =  54 20          ; RVA = 2054
TimeDateStamp  =  0
ForwarderChain = 0
Name = 86 20                          ; RVA = 2086
FirstThunk = 08 20                    ; RVA = 2008
```

Antes de explicar bien algunas cositas, recuerden que siempre para calcular el RVA, es RVA – VirtualOffset de la sección + RawOffset de la sección. Por eso todas las "direcciones" que vamos a usar aquí por Ej.: son 6xx.

Nuestro OriginalFirstThunk apunta a 2054, nuestro Raw es 654h usando lo que acabo de explicar.
Dijimos que FirstThunk era una especie de duplicado, y su valor es 608h.
Miremos las dos Raws.., y que vemos?
El valor “78 20” en las dos direcciones, esto nos apunta directamente a la estructura “IMAGE_IMPORT_BY_NAME”.
Conclusión, estos dos firstthunks son punteros de una estructura.

También tenemos Name, que apuntan a 686h, y vemos que ahí tenemos el valor “user32.dll”.

Ahora si pasemos a la estructura que vimos recién:

```
IMAGE_IMPORT_BY_NAME STRUCT
  Hint dw ?                    ; Nos dice donde encontrar la función en el
                               Export Name Pointer Table, del su dll.
                               Algunos linkers directamente ponen 0.
                               Si el PE Loador no encuentra en esta dirección la función,
                               Lo busca por si mismo en la tabla que nombramos.

  Name1 db ?                   ; El nombre de la función.
IMAGE_IMPORT_BY_NAME ENDS
```

Ahora, si abrimos el Hi.exe con el Olly, vamos a ver que tenemos por ejemplo:

```
00401000 >/$  6A 00         PUSH 0                          ; /Style = MB_OK|MB_APPLMODAL
00401002  |.  68 00304000   PUSH hi.00403000                 ; |Title = "Hey!"
00401007  |.  68 05304000   PUSH hi.00403005                 ; |Text = "Win Assembly is Great"
0040100C  |.  6A 00         PUSH 0                          ; |hOwner = NULL
0040100E  |.  E8 0D000000   CALL <JMP.&user32.MessageBoxA>          ; CALL 00401020
```

y vemos que ese call es:

```
00401020   $- FF25 08204000 JMP DWORD PTR DS:[402008]   ; No se les hace conocido el 008?
```

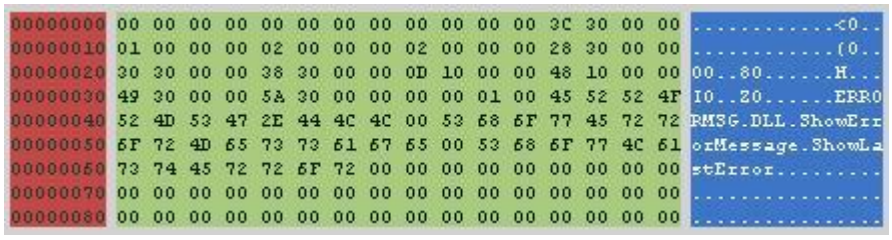
Limpiemos eso:
A esa dirección si le restamos el imagebase (00400000), también le sacamos el virtualoffset (2000), y le sumamos el rawoffset (600)
Nos quedará 608h verdad?.
Fíjense que este valor es el mismo que firstthunk.. ☺
Ahora ya saben el “porque de esas direcciones”.

1.2.5 Tabla de Exportaciones.

Cuando otro programa usa alguna función de por ejemplo algún DLL, puede "llamarlo" de 2 maneras: por el nombre, o por el "identificador ordinal".

```
typedef struct _IMAGE_EXPORT_DIRECTORY {
   DWORD   Characteristics;                ; Siempre esta en 0
   DWORD   TimeDateStamp;                  ;
   WORD    MajorVersion;                   ; También siempre en 0
   WORD    MinorVersion;
   DWORD   Name;                           ; RVA de un ascii string.
   DWORD   Base;                           ; Nº desde el cual se empiezan
                                           Enumerar las funciones.
                                           Casi siempre = 1.
   DWORD   NumberOfFunctions;              ; Nº de funciones exportadas.
   DWORD   NumberOfNames;                  ; Nº de funciones exportadas x nombre.
   DWORD   AddressOfFunctions;             ; RVA de la EAT.
   DWORD   AddressOfNames;                 ; RVA de la tabla de nombres.
   DWORD   AddressOfNameOrdinals;          ; RVA de la tabla de ordinales.
}
```

Analizamos el archivo adjunto a este .zip, ERRORMSG.DLL.



- **Characteristics** 00 00 00 00 ;
- **TimeDateStamp** 00 00 00 00 ;
- **MajorVersion** 00 00 ;
- **MinorVersion** 00 00 ;
- **Name** 3C 30 00 00 ; Apunta a ERRORMSG.DLL. (xx 3C)
- **Base** 01 00 00 00 ; Como dijimos casi siempre 1.
- **NumberOfFunctions** 02 00 00 00 ; Tenemos 2 funciones.
- **NumberOfNames** 02 00 00 00 ; En el 99% de los caso es igual al anterior.

- **AddressOfFunction** 28 30 00 00 ;

Vemos que su contenido tiene 2 rva's:

0D 10 00 00 ; 100Dh
48 10 00 00 ; 1048h

Abrimos por ejemplo el Win32Dasm, vamos a nuestras primeras líneas,
Y vemos antes de 0040100D: Exported fn(): ShowErrorMessage.

Ahora hagan lo mismo ustedes con el 2do RVA. ☺

- **AddressOfNames** 30 30 00 00 ;

Aquí también encontraremos 2 rva's:

49 30 00 00; 3049h
5A 30 00 00; 305Ah

Sabiendo que su contenido está en nuestra sección .edata, chequeamos y los 2 rvas apuntan a los nombres de las funciones.
Sitúense con su editor hex amigo, en la línea 85Ah, y verán que justo se posicionan sobre la "S" de ShowErrorMessage.

- **AddressOfNameOrdinals** 38 30 00 00 ; Solamente se usa para funciones exportadas solo por número.