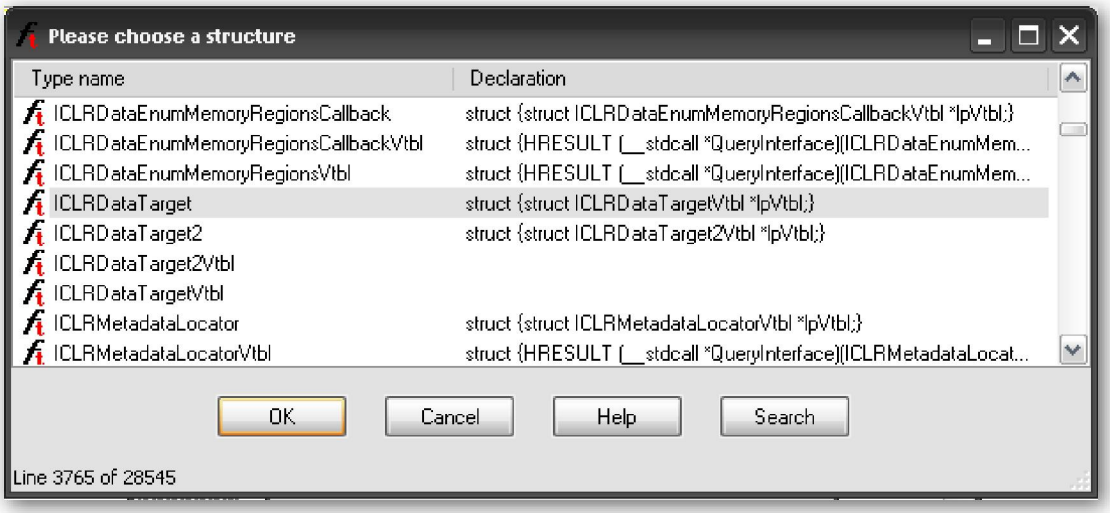
7.5.—Utilizar estructuras estandarizadas

Como ya habíamos mencionada anteriormente, IDA reconoce una gran cantidad de estructuras de datos asociadas con varias librerías y con funciones API. Cuando se crea inicialmente una base de datos, IDA intenta determinar el compilador y la plataforma asociada al binario y cargar la plantilla de estructura apropiada. En el momento en que IDA encuentra en el desensamblado alguna manipulación de la estructura actual, éste añade las definiciones de estructura apropiadas en la ventana **Structures**. Así pues, en la ventana **Structures** se representan el subconjunto de las estructuras conocidas del binario actual analizado. También, además de poder crear nuestras estructuras a medida, podemos añadir estructuras estándares a la ventana **Structures**, desde la lista de IDA de tipos de estructura conocidos.

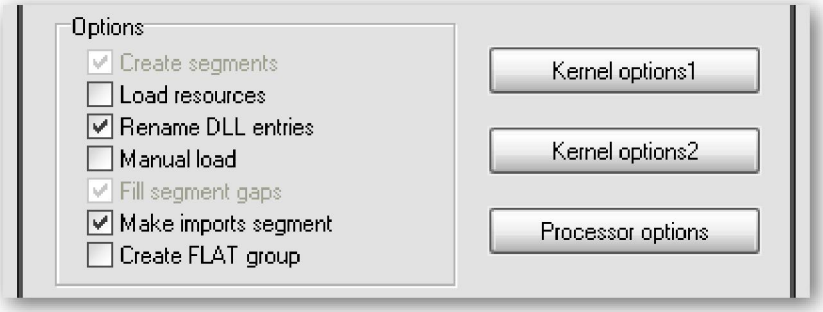
El proceso para poder añadir una nueva estructura se inicia pulsando la tecla **INSERT** en la ventana **Structures**. En, figura abajo, vemos el diálogo **Create Structure/Union,**

Uno de sus componentes es el botón **Add standard structure**. Si “clickeamos” dicho botón accedemos a la lista maestra de estructuras correspondientes al compilador actual, detectado durante la fase de análisis, y al formato de archivo. Esta lista de estructuras también contiene todas las estructuras que se han añadido a la base de datos debido al análisis de los encabezados C del archivo. El diálogo de selección, figura abajo, se utiliza para elegir una estructura y añadirla a la ventana **Structures**.



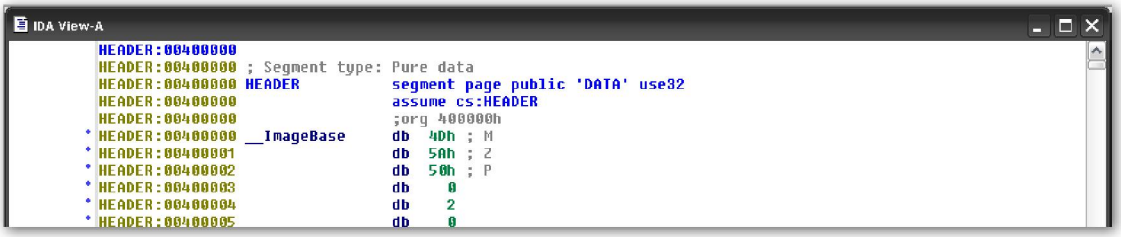
Podemos utilizar **Search** para localizar una estructura, introduciendo parte de su nombre. Si conocemos los primeros caracteres del nombre de una estructura, simplemente los tecleamos, en la ventanita que nos aparece al pulsar **Search**, y en la lista nos aparecerá la primera estructura que coincida con lo tecleado. Elegir una estructura es añadir la estructura y cualquier estructura anidada a ella, a la ventana **Structures**.

Como siempre un ejemplo vale por mil palabras, consideremos un caso en el cual queremos examinar los encabezados de archivo asociados a un binario Windows PE. Por defecto los encabezados del archivo no se graban en la base de datos cuando esta se crea por primera vez; sin embargo, los encabezados de archivo pueden cargarse si seleccionamos en la ventana **Load a new file** la opción **Manual load** durante la creación de la base de datos, figura abajo.



Cargar los encabezados de archivo nos asegura solamente que los bytes asociados con estos encabezados estarán presentes en la base de datos. En la mayoría de los casos, los encabezados no estarán formateados de ninguna forma ya que los programas no hacen ninguna referencia directa a sus propios encabezados. Por lo tanto no hay razón para analizar las plantillas de estructura que se aplican a los encabezados.

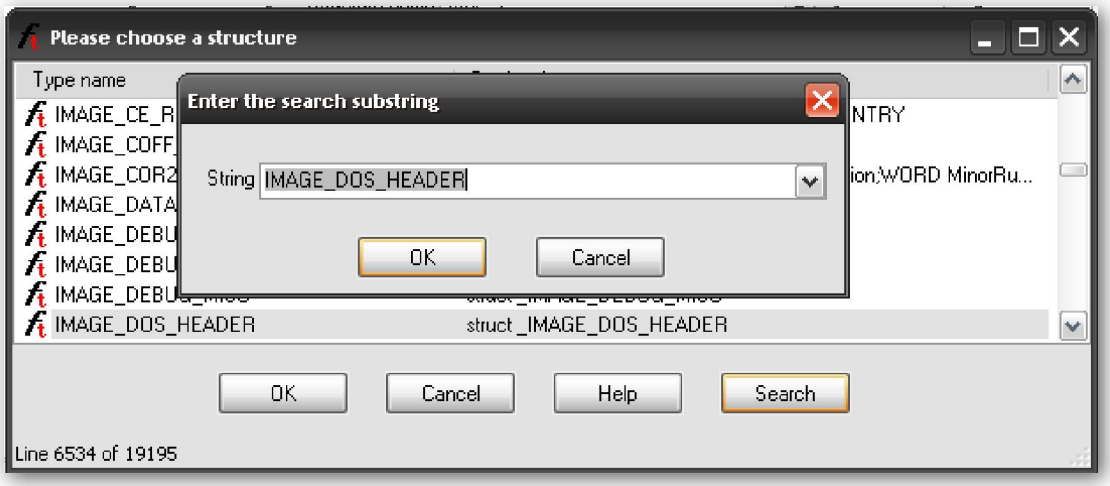
Como ya conocemos el formato que tiene un binario PE, sabemos que dicho archivo se inicia con un encabezado estructurado **MS-DOS** llamado **IMAGE_DOS_HEADER**. Un dato contenido en el **IMAGE_DOS_HEADER** apunta a la ubicación de una estructura **IMAGE_NT_HEADERS**, la cual detalla el esquema de memoria del binario PE. Si elegimos cargar los encabezados PE, del archivo CRACKME.EXEveremos en el desensamblado algo similar a lo siguiente.

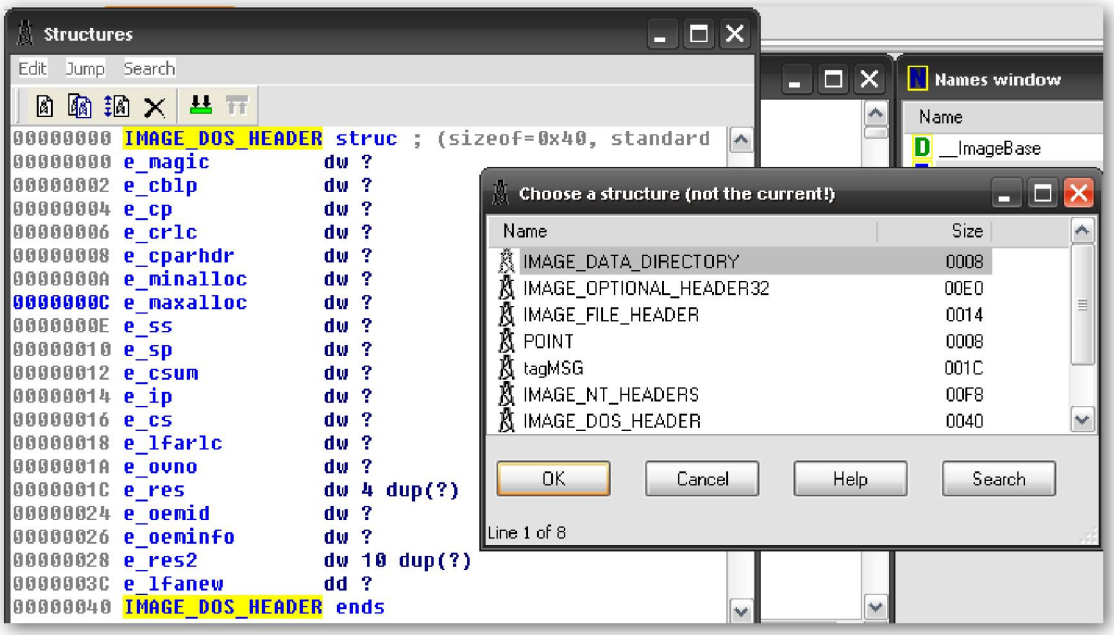


Los que estamos familiarizados con la estructura de un archivo PE, reconoceremos el valor mágico MS-DOS, **MZ** como los dos primeros bytes en el archivo.

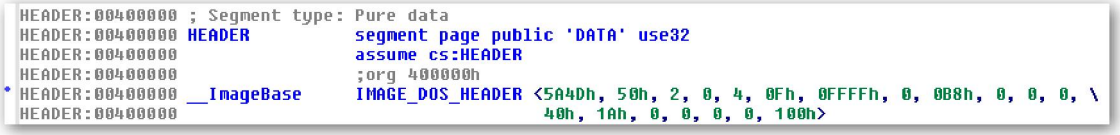
Para entender cada byte de este archivo tal como está formateado aquí, necesitarás consultar la documentación de referencia sobre archivos PE. Pero si utilizamos las plantillas de estructuras de IDA, este puede formatear los bytes como un **IMAGE_DOS_HEADER**, convirtiendo los datos en comprensibles y útiles. El primer paso para realizarlo, es añadir la plantilla de estructura **IMAGE_DOS_HEADER**

estandar, y ya que nos ponemos podemos añadir también la plantilla de estructura **HEADER_NT_HEADERS** a la base da datos de nuestro desensamblado.





El segundo paso, será convertir los bytes cargados en una estructura **IMAGE_DOS_HEADER**, seleccionamos los bytes en el desensamblado empezando desde la dirección **__ImageBase** hasta **00400040,** que es la longitud del header y utilizando la acción **Edit > Struct Var** o el atajo, **ALT-Q** nos quedará toda la estructura en una sola línea, ver figura siguiente.



Si expandimos la estructura, el resultado del nuevo formateado es el siguiente:

```
HEADER:00400000 ; Segment type: Pure data
HEADER:00400000 HEADER          segment page public 'DATA' use32
HEADER:00400000                 assume cs:HEADER
HEADER:00400000                 ;org 400000h
HEADER:00400000 __ImageBase     dw 5A4Dh                ; e_magic
HEADER:00400000                 dw 50h                  ; e_cblp
HEADER:00400000                 dw 2                    ; e_cp
HEADER:00400000                 dw 0                    ; e_crlc
HEADER:00400000                 dw 4                    ; e_cparhdr
HEADER:00400000                 dw 0Fh                  ; e_minalloc
HEADER:00400000                 dw 0FFFFh               ; e_maxalloc
HEADER:00400000                 dw 0                    ; e_ss
HEADER:00400000                 dw 0B8h                 ; e_sp
HEADER:00400000                 dw 0                    ; e_csum
HEADER:00400000                 dw 0                    ; e_ip
HEADER:00400000                 dw 0                    ; e_cs
HEADER:00400000                 dw 40h                  ; e_lfarlc
HEADER:00400000                 dw 1Ah                  ; e_ovno
HEADER:00400000                 dw 4 dup(0)             ; e_res
HEADER:00400000                 dw 0                    ; e_oemid
HEADER:00400000                 dw 0                    ; e_oeminfo
HEADER:00400000                 dw 0Ah dup(0)           ; e_res2
HEADER:00400000                 dd 100h                 ; e_lfanew
```

Como podemos ver, figura anterior, los primeros **64 (0x40)** bytes del archivo han sido volcados en una estructura de datos, con su tipo anotado en el desensamblado. Por supuesto si no tienes conocimientos del contenido de estos encabezados, estos datos actuales también te pueden ser irreconocibles. Para poder ver toda esta información hemos tenido que expandir la estructura, cuando expandimos la estructura en cada campo de ésta se realiza la anotación del nombre del campo correspondiente a la definición de la estructura. Como ya sabemos la acción de expandir una estructura, se realiza con la tecla más del teclado numérico.

Por desgracia, los campos de **IMAGE_DOS_HEADER** no poseen nombres significativos, por lo cual necesitaremos consultar una referencia al archivo PE, para recordar que, por ejemplo el campo **e_lfanew** indica el offset de ubicación, en donde encontraremos la estructura de **IMAGE_NT_HEADERS**. Si realizamos todos los pasos anteriores para crear una estructura **IMAGE_NT_HEADER** en la dirección **00400100**, **0x100** bytes más abajo de la base de datos según **e_lfanew**, nos encontramos con un formato de estructura tal como:

```
HEADER:00400100                 dd 4550h                ; Signature
HEADER:00400100                 dw 14Ch                 ; FileHeader.Machine
HEADER:00400100                 dw 6                    ; FileHeader.NumberOfSections
HEADER:00400100                 dd 0AD92429h            ; FileHeader.TimeDateStamp
HEADER:00400100                 dd 0                    ; FileHeader.PointerToSymbolTable
HEADER:00400100                 dd 0                    ; FileHeader.NumberOfSymbols
HEADER:00400100                 dw 0E0h                 ; FileHeader.SizeOfOptionalHeader
HEADER:00400100                 dw 818Eh                ; FileHeader.Characteristics
HEADER:00400100                 dw 10Bh                 ; OptionalHeader.Magic
HEADER:00400100                 db 2                    ; OptionalHeader.MajorLinkerVersion
HEADER:00400100                 db 19h                  ; OptionalHeader.MinorLinkerVersion
HEADER:00400100                 dd 600h                 ; OptionalHeader.SizeOfCode
HEADER:00400100                 dd 2200h                ; OptionalHeader.SizeOfInitializedData
HEADER:00400100                 dd 0                    ; OptionalHeader.SizeOfUninitializedData
HEADER:00400100                 dd 1000h                ; OptionalHeader.AddressOfEntryPoint
HEADER:00400100                 dd 1000h                ; OptionalHeader.BaseOfCode
HEADER:00400100                 dd 2000h                ; OptionalHeader.BaseOfData
HEADER:00400100                 dd 400000h              ; OptionalHeader.ImageBase
HEADER:00400100                 dd 1000h                ; OptionalHeader.SectionAlignment
HEADER:00400100                 dd 200h                 ; OptionalHeader.FileAlignment
HEADER:00400100                 dw 1                    ; OptionalHeader.MajorOperatingSystemVersion
HEADER:00400100                 dw 0                    ; OptionalHeader.MinorOperatingSystemVersion
HEADER:00400100                 dw 0                    ; OptionalHeader.MajorImageVersion
HEADER:00400100                 dw 0                    ; OptionalHeader.MinorImageVersion
HEADER:00400100                 dw 3                    ; OptionalHeader.MajorSubsystemVersion
HEADER:00400100                 dw 0Ah                  ; OptionalHeader.MinorSubsystemVersion
HEADER:00400100                 dd 0                    ; OptionalHeader.Win32VersionValue
HEADER:00400100                 dd 8000h                ; OptionalHeader.SizeOfImage
HEADER:00400100                 dd 400h                 ; OptionalHeader.SizeOfHeaders
HEADER:00400100                 dd 0                    ; OptionalHeader.CheckSum
HEADER:00400100                 dw 2                    ; OptionalHeader.Subsystem
HEADER:00400100                 dw 0                    ; OptionalHeader.DllCharacteristics
HEADER:00400100                 dd 100000h              ; OptionalHeader.SizeOfStackReserve
HEADER:00400100                 dd 2000h                ; OptionalHeader.SizeOfStackCommit
HEADER:00400100                 dd 100000h              ; OptionalHeader.SizeOfHeapReserve
HEADER:00400100                 dd 1000h                ; OptionalHeader.SizeOfHeapCommit
HEADER:00400100                 dd 0                    ; OptionalHeader.LoaderFlags
```

```
HEADER:00400100                 dd 10h                  ; OptionalHeader.NumberOfRvaAndSizes
HEADER:00400100                 dd 4000h                ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 46h                  ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 3000h                ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 670h                 ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 6000h                ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 1400h                ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 5000h                ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 0DCh                 ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.Size
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.VirtualAddress
HEADER:00400100                 dd 0                    ; OptionalHeader.DataDirectory.Size
```

Afortunadamente para nosotros, los nombres de los campos en este caso son muy significativos. Rápidamente podemos determinar que el archivo contiene 6 secciones,

```
HEADER:00400100                 dw 6                    ; FileHeader.NumberOfSections
```

que se cargará en memoria en la dirección virtual 00400000, etc.

```
HEADER:00400100                 dd 400000h              ; OptionalHeader.ImageBase
```

Para finalizar recordemos que para encoger las estructuras expandidas tenemos que utilizar la tecla menos del teclado numérico.

Performance Bigundill@