Taller de Formato PE by Ferchu

Este taller se tratara el Formato PE en archivos ejecutables, solo se tocaran los conceptos importantes, tratando de que se entiendan los conceptos mas fundamentalos que abarcan a los archivos con este formato. Voy a tratar de usar un lenguaje para que todos entiendan, pero cuando sea necesario, y a medida que el taller avance, se va recurrir a un lenguaje mas tecnico.

Antes de empezar, voy a mencionar algunas convenciones (con las cuales me explico) y conceptos minimos sobre la memoria que se deberian manejar para entender lo que luego se expone.

A lo largo de este texto, todos los numeros que comienzen con 0x... son numero en base hexadecimal y deben ser interpretados como tales.

Cuando hablamos de memoria virtual, nos referimos al bloque de memoria que nos asigna el Sistema Operativo en la Ram para trabajar con nuestro programa. La direccion virtual no hace referencia a la direccion real de la memoria ram, solo a un bloque que el programa esta usando de ella. En win32 los programas pueden manejar direcciones de memoria de 0 a 2^{32}, esto NO quiere decir que en memoria se cargue 4 gb, solo que el programa puede manejar direcciones de ese tamaño (0x00000000 - 0xFFFFFFFF).

Cuando hablamos de direcciones, no siempre nos estamos refiriendo a la memoria, sino tambien a la posicion de los datos en el archivo.

Para pasar en limpio, cuando hablemos de direcciones en memoria, seran "direcciones virtuales", y cuando hablemos de direcciones en el archivo, seran "direcciones fisicas".

Ahora si, empezemos...

Introduccion

Un archivo ejectuable esta estructurado de la siguiente forma:

Cabecera DOS
Stub Dos
Cabecera PE
Tabla de secciones
1º sección
2º sección
3º sección

Nº sección
Datos extra

- Cabecera DOS: Es el comienzo del archivo, especifica las caracteristicas necesarias para ejecutar el programa cuando el Sistema Operativo es DOS. De lo contrario salta a la cabecera PE.
- Stub Dos: Codigo que se ejecuta si el entorno es DOS. Por ej: El tipico cartelito "This program cannot be run in DOS mode"
- Cabecera PE: Estructura que contiene diversa informacion sobre el archivo ejectuable. (por ahora eso solo decimos.)
- Tabla de secciones: Como es obvio, contiene el listado de las secciones del ejectuable, mas adelante se explicara su estructura.
- Los datos extra, serian aquellos que no esten contemplados por la estructura del ejectuable, es decir, no pertenescan a ninguna sección. Para que entiendan mejor, por ej: como muchos servers de troyanos que utilizan el final del archivo para guardar la configuracion del mismo. Estos solo son apilados al final y quien entiende la estructura de esos datos es el programa.

Bueno la Cabecera DOS y el Stub Dos, no nos interesan mucho a nosotros. Lo que si realmente importa es la Cabecera PE, pero....donde encuentra la cabecera PE en un ejecutable?? Esta siempre en un mismo lugar??. La respuesta es NO, puede estar en cualquier posicion. Para saber donde esta, hay que ir al offset PE que esta e_lfanew (miembro de la cabecera DOS) que es la pos 0x3c del archivo, ahi se encuentra un entero (4 bytes) que dice la direccion fisica donde comienza la cabecera PE.

A continuacion se describe, Los miembros mas importantes del estructura PE.

Los primeros 2 bytes de la cabecera, corresponden a las letras "PE", asi a simple vista, con un editor hexadecimal, mirando el offset PE y yendo a la posicion, rapidamente encontramos el comienzo de la cabecera.

Machine (offset + 0x04)

Corresponde a la arquitectura para la cual esta "preparado" el archivo. Esto se indica por medio de Flags por lo cual con un editor solo veremos numeros, pero con un debugger como olly u otros programas podemos ver a cual arquitectura corresponde el valor (o buscando en la msdn), igual esto no es muy importante para nosotros.

NumberOfSections (offset + 0x06)

En este WORD (2 bytes) se indica la cantidad de secciones que contiene el archivo, En el caso de que agreguemos una sección este sera uno de los tantos valores a modificar.

SizeOfOptionalHeader (offset + 0x14)

Es un WORD (2 bytes) que nos dice el tamaño de la cabecera opcional, en la mayoria

de los archivos esta cabecera existe, y aca esta

Characteristics (offset + 0x16)

Contiene informacion sobre las caracteristicas de la imagen (si es ejecutable, si es 32 bits, etc). No es muy importante para nosotros, asi que solo lo mencionamos.

SizeOfCode (offset + 0x1C)

Es un entero (4 bytes) que contiene la suma de los tamaños de todas las secciones de "codigo".

SizeOfInitializedData (offset + 0x20)

Es un entero (4 bytes) que contiene la suma de los tamaños de todas las secciones de que tiene datos inicializados.

SizeOfUninitializedData (offset + 0x24)

Es un entero (4 bytes) que contiene la suma de los tamaños de todas las secciones de que tiene datos NO inicializados.

AddressOfEntryPoint (offset + 0x28)

Es un entero (4 bytes) que contiene el offset virtual donde comienza a ejecutar codigo. La direccion virtual seria este OEP + ImageBase.

BaseOfCode (offset + 0x2C)

Es un entero (4 bytes) que contiene el offset virtual donde se va a alojar la sección de codigo. Para tener la direccion virtual hay que sumarle la ImageBase.

BaseOfData (offset + 0x30)

Es un entero (4 bytes) que contiene el offset virtual donde se va a alojar la sección de datos. Para tener la direccion virtual hay que sumarle la ImageBase.

ImageBase (offset + 0x34)

Esta es la ImageBase de la que hablabamos en los 3 Itemos anteriores, Es un entero que contiene la direccion Virtual sobre la cual se va a cargar las secciones en memoria. Por eso de dice que es la Base, por que todos los offset estan haciendo referencia a una direccion virtual empezando desde esa posicion.

SectionAlignment (offset + 0x38)

Al alojar memoria, se hace con un criterio, se reserva memoria de a "bloques", estos bloques deben ser multipos de este valor, generalmente es 0x1000. En pocas palabras cuando se reserva memoria se lo hace de a bloques de 0x1000 bytes. No importa si a la sección a cargar le va a sobrar espacio, se redonde a ese numero, siempre para arriba

claro.

FileAlignment (offset + 0x3C)

Igual que el Item anterior, solo que aca se habla sobre los bloques en el archivo, los datos estan agrupados por bloques de un tamaño "FileAlignment", y al cargarlos en memoria el SO respeta esta alineacion. Esto es muy importante a considerar cuando vamos a agregar una nueva sección, de tener en cuenta de que nuestros datos esten alineados en el archivo, de lo contrario no se van a cargar Correctamente en memoria. A modo comentario, si lo prueban veran que carga en la memoria datos anteriores al comienzo de dicha sección.

SizeOfImage (offset + 0x50)

Es un entero (4 bytes) que nos dice el tamaño de memoria virtual total que utiliza el archivo para luego volcar las secciones, es decir, la cantidad de memoria virtual que utilizara el archivo. Es otro de los valores que vamos a modificar al agregunar una sección.

NumberOfRvaAndSizes (offset + 0x74)

Contiene el numero de "bloques de tablas" que hay en la cabecera opcional.

El segundo "bloque" contiene informacion sobre donde se aloja en memoria virtual la Import Table, y el tamaño de ella. Es el unico que nos importa por ahora.
Este Bloque, si existe, esta en el offset 0x80 de la cabecera. Digo si existe por que si NumberOfRvaAndSizes es 0, ese "bloque" no queda definido.

Al final de estos bloques, comienza a especificarse las secciones mediante una estructura de datos (IMAGE_SECTION_HEADER) que tiene un tamaño de 0x28 bytes, al final de esta comienza la proxima y asi hasta la ultima.

Con un editor hexadecimal se ven facilmente el nombre de la sección, x ej: ".text", ".data", ".rcsc", "idata", etc.

A continuacion se describen los miembros mas importantes de la estructura (IMAGE_SECTION_HEADER)

Ahora el offset es diferente al anterior, este corresponde a la direccion donde comienza la estructura que define a la sección.

Name (offset + 0x00) (8 bytes)

Nombre corto de la sección, por ej ".code", ".data", el nombre es solo para identificar a la sección, no describre si esta es de codigo, de datos, etc, eso lo hace el miembro Characteristics de la estructura. (ver mas abajo).

VirtualSize (offset + 0x08) (4 bytes)

Es el tamaño del bloque de la memoria virtual donde se cargara la sección. Es decir se

reserva un bloque de memoria de este tamaño, la direccion donde se lo reserva lo dice el siguiente miembro. El valor es siempre mayor o igual a el de SizeOfRawData.

VirtualAddress (offset + 0x0C) (4 bytes)

Es la direccion virtual que se reserva para cargar la sección.

SizeOfRawData (offset + 0x10) (4 bytes)

Tamaño que ocupa la sección en el archivo.

PointerToRawData (offset + 0x14) (4 bytes)

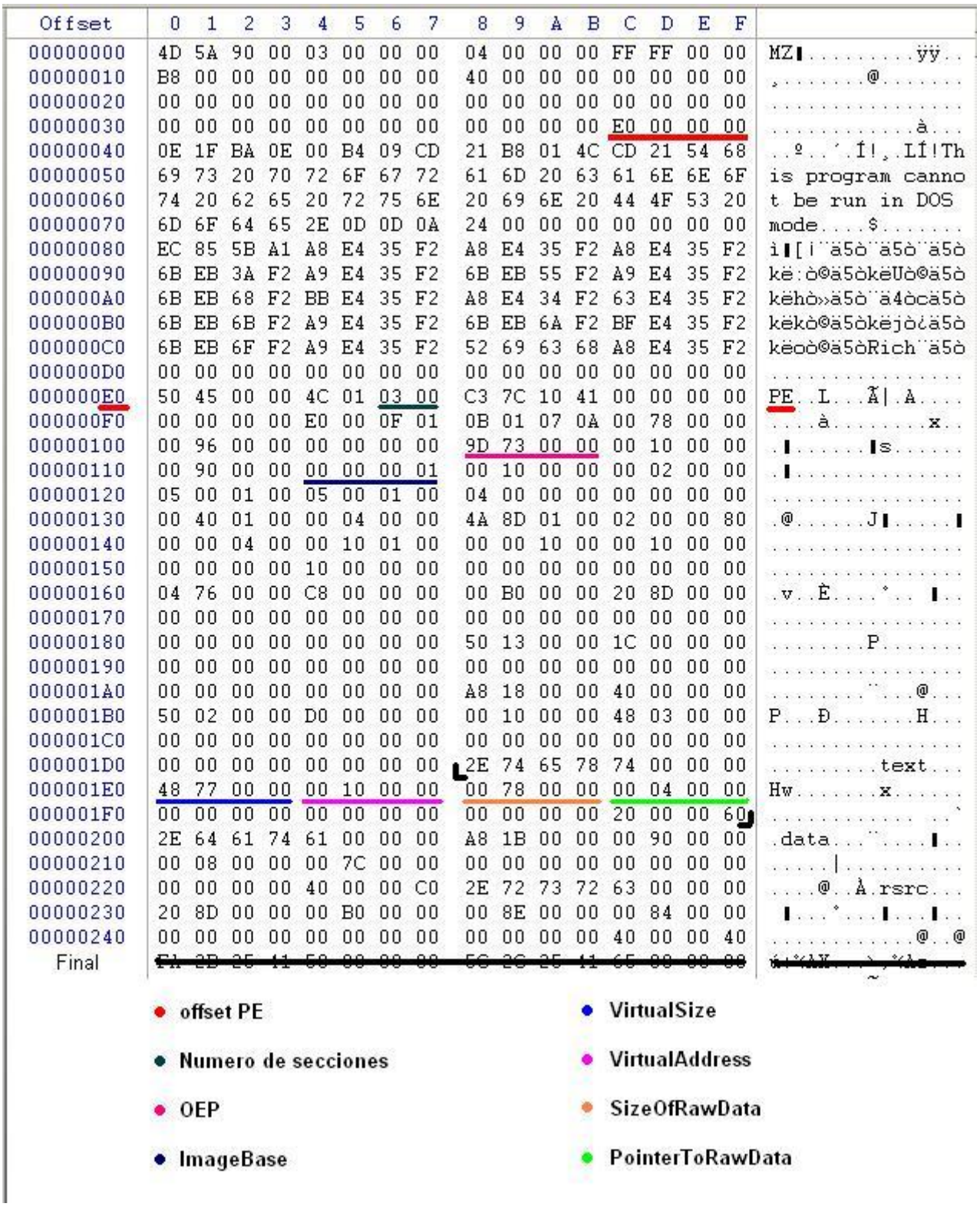
Direccion fisica que indica donde comienzan el contenido de la sección EN el archivo. Recordar que esta direccion debe ser multiplo de FileAlignment, que generalmente es 0x200.

Characteristics (offset + 0x24) (4 bytes)

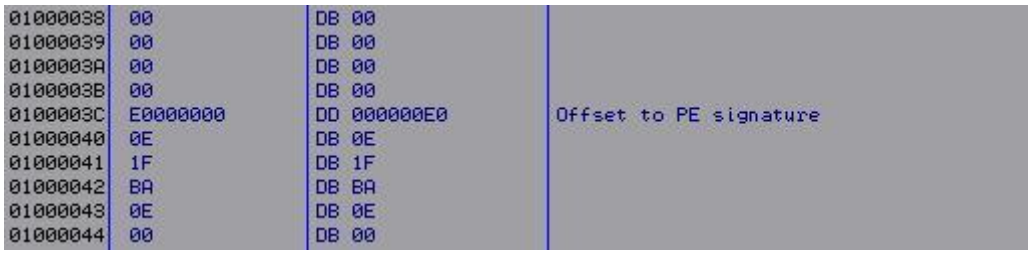
En este miembro se indican las caracteristicas de la sección mediante flags. Si en la sección esta PERMITIDO escribir, leer, executar, si la sección ES de codigo, datos inicializados, datos no inicializados, etc.
Para saber que corresponde a cada flag, lo mejor es mirar la msdn, no tiene mucho sentido explicar que es esto.

http://msdn2.microsoft.com/en-us/library/ms680341.aspx

A continuacion, para que se entienda donde esta hubicado cada miembro, vemos de 2 modos diferentes, al notepad.exe, la primera con un editor hexadecimal, para apreciar mejor los valores y ver como se almazenan, y luego con un debugger, para ver mas organizada la estructura dividida por miembros, ya que el debugger reconocer cada uno, y nos indica cuales son. De esta manera podran comparar e ir familiarizando con la ubicacion de los datos que luego vamos a modificar.



En la cabecera DOS, lo unico que nos interesa conocer es la posicion donde empieza la cabecera PE. Asi que en el olly seria esto:



```
01000038  00            DB 00
01000039  00            DB 00
0100003A  00            DB 00
0100003B  00            DB 00
0100003C  E0000000      DD 000000E0          Offset to PE signature
01000040  0E            DB 0E
01000041  1F            DB 1F
01000042  BA            DB BA
01000043  0E            DB 0E
01000044  00            DB 00
```

La Cabecera PE:

Address	Hex dump	Data	Comment
010000DF	00	DB 00	
010000E0	50 45 00 00	ASCII "PE"	PE signature (PE)
010000E4	4C01	DW 014C	Machine = IMAGE_FILE_MACHINE_I386
010000E6	0300	DW 0003	NumberOfSections = 3
010000E8	C37C1041	DD 41107CC3	TimeDateStamp = 41107CC3
010000EC	00000000	DD 00000000	PointerToSymbolTable = 0
010000F0	00000000	DD 00000000	NumberOfSymbols = 0
010000F4	E000	DW 00E0	SizeOfOptionalHeader = E0 (224.)
010000F6	0F01	DW 010F	Characteristics = EXECUTABLE_IMAGE!32BIT_MACHINE
010000F8	0B01	DW 010B	MagicNumber = PE32
010000FA	07	DB 07	MajorLinkerVersion = 7
010000FB	0A	DB 0A	MinorLinkerVersion = A (10.)
010000FC	00780000	DD 00007800	SizeOfCode = 7800 (30720.)
01000100	00960000	DD 00009600	SizeOfInitializedData = 9600 (38400.)
01000104	00000000	DD 00000000	SizeOfUninitializedData = 0
01000108	9D730000	DD 0000739D	AddressOfEntryPoint = 739D
0100010C	00100000	DD 00001000	BaseOfCode = 1000
01000110	00900000	DD 00009000	BaseOfData = 9000
01000114	00000001	DD 01000000	ImageBase = 1000000
01000118	00100000	DD 00001000	SectionAlignment = 1000
0100011C	00020000	DD 00000200	FileAlignment = 200
01000120	0500	DW 0005	MajorOSVersion = 5
01000122	0100	DW 0001	MinorOSVersion = 1
01000124	0500	DW 0005	MajorImageVersion = 5
01000126	0100	DW 0001	MinorImageVersion = 1
01000128	0400	DW 0004	MajorSubsystemVersion = 4
0100012A	0000	DW 0000	MinorSubsystemVersion = 0
0100012C	00000000	DD 00000000	Reserved
01000130	00400100	DD 00014000	SizeOfImage = 14000 (81920.)
01000134	00040000	DD 00000400	SizeOfHeaders = 400 (1024.)
01000138	4A8D0100	DD 00018D4A	CheckSum = 18D4A
0100013C	0200	DW 0002	Subsystem = IMAGE_SUBSYSTEM_WINDOWS_GUI
0100013E	0080	DW 8000	DLLCharacteristics = 8000
01000140	00000400	DD 00040000	SizeOfStackReserve = 40000 (262144.)
01000144	00100100	DD 00011000	SizeOfStackCommit = 11000 (69632.)
01000148	00001000	DD 00100000	SizeOfHeapReserve = 100000 (1048576.)
0100014C	00100000	DD 00001000	SizeOfHeapCommit = 1000 (4096.)
01000150	00000000	DD 00000000	LoaderFlags = 0
01000154	10000000	DD 00000010	NumberOfRvaAndSizes = 10 (16.)
01000158	00000000	DD 00000000	Export Table address = 0
0100015C	00000000	DD 00000000	Export Table size = 0
01000160	04760000	DD 00007604	Import Table address = 7604
01000164	C8000000	DD 000000C8	Import Table size = C8 (200.)
01000168	00B00000	DD 0000B000	Resource Table address = B000
0100016C	208D0000	DD 00008D20	Resource Table size = 8D20 (36128.)
01000170	00000000	DD 00000000	Exception Table address = 0
01000174	00000000	DD 00000000	Exception Table size = 0
01000178	00000000	DD 00000000	Certificate File pointer = 0
0100017C	00000000	DD 00000000	Certificate Table size = 0
01000180	00000000	DD 00000000	Relocation Table address = 0
01000184	00000000	DD 00000000	Relocation Table size = 0
01000188	50130000	DD 00001350	Debug Data address = 1350
0100018C	1C000000	DD 0000001C	Debug Data size = 1C (28.)
01000190	00000000	DD 00000000	Architecture Data address = 0
01000194	00000000	DD 00000000	Architecture Data size = 0
01000198	00000000	DD 00000000	Global Ptr address = 0
0100019C	00000000	DD 00000000	Must be 0
010001A0	00000000	DD 00000000	TLS Table address = 0
010001A4	00000000	DD 00000000	TLS Table size = 0
010001A8	A8180000	DD 000018A8	Load Config Table address = 18A8
010001AC	40000000	DD 00000040	Load Config Table size = 40 (64.)
010001B0	50020000	DD 00000250	Bound Import Table address = 250
010001B4	D0000000	DD 000000D0	Bound Import Table size = D0 (208.)
010001B8	00100000	DD 00001000	Import Address Table address = 1000
010001BC	48030000	DD 00000348	Import Address Table size = 348 (840.)
010001C0	00000000	DD 00000000	Delay Import Descriptor address = 0
010001C4	00000000	DD 00000000	Delay Import Descriptor size = 0
010001C8	00000000	DD 00000000	COM+ Runtime Header address = 0
010001CC	00000000	DD 00000000	Import Address Table size = 0
010001D0	00000000	DD 00000000	Reserved
010001D4	00000000	DD 00000000	Reserved
010001D8	2E 74 65 78 74	ASCII ".text"	SECTION
010001E0	48770000	DD 00007748	VirtualSize = 7748 (30536.)

La lista de Secciones:

Address	Hex dump	Data	Comment
010001D4	00000000	DD 00000000	Reserved
010001D8	2E 74 65 78 74	ASCII ".text"	SECTION
010001E0	48770000	DD 00007748	VirtualSize = 7748 (30536.)
010001E4	00100000	DD 00001000	VirtualAddress = 1000
010001E8	00780000	DD 00007800	SizeOfRawData = 7800 (30720.)
010001EC	00040000	DD 00000400	PointerToRawData = 400
010001F0	00000000	DD 00000000	PointerToRelocations = 0
010001F4	00000000	DD 00000000	PointerToLineNumbers = 0
010001F8	0000	DW 0000	NumberOfRelocations = 0
010001FA	0000	DW 0000	NumberOfLineNumbers = 0
010001FC	20000060	DD 60000020	Characteristics = CODE|EXECUTE|READ
01000200	2E 64 61 74 61	ASCII ".data"	SECTION
01000208	A81B0000	DD 00001BA8	VirtualSize = 1BA8 (7080.)
0100020C	00900000	DD 00009000	VirtualAddress = 9000
01000210	00080000	DD 00000800	SizeOfRawData = 800 (2048.)
01000214	007C0000	DD 00007C00	PointerToRawData = 7C00
01000218	00000000	DD 00000000	PointerToRelocations = 0
0100021C	00000000	DD 00000000	PointerToLineNumbers = 0
01000220	0000	DW 0000	NumberOfRelocations = 0
01000222	0000	DW 0000	NumberOfLineNumbers = 0
01000224	400000C0	DD C0000040	Characteristics = INITIALIZED_DATA|READ|WRITE
01000228	2E 72 73 72 63	ASCII ".rsrc"	SECTION
01000230	208D0000	DD 00008D20	VirtualSize = 8D20 (36128.)
01000234	00B00000	DD 0000B000	VirtualAddress = B000
01000238	008E0000	DD 00008E00	SizeOfRawData = 8E00 (36352.)
0100023C	00840000	DD 00008400	PointerToRawData = 8400
01000240	00000000	DD 00000000	PointerToRelocations = 0
01000244	00000000	DD 00000000	PointerToLineNumbers = 0
01000248	0000	DW 0000	NumberOfRelocations = 0
0100024A	0000	DW 0000	NumberOfLineNumbers = 0
0100024C	40000040	DD 40000040	Characteristics = INITIALIZED_DATA|READ
01000250	FA	DB FA	
01000251	2B	DB 2B	
01000252	25	DB 25	
01000253	41	DB 41	
01000254	58	DB 58	

Capitulo II: Agregando una sección, sin morir en el intento.

(17/04/08)

Como dice el titulo, en este capitulo vamos a agregar una sección, hacer esto no es muy dificil cuando se lo sabe hacer, pero para uno que empieza hay que tener cuidado, asi que vamos a ver en la practica que hay muchas formas de terminar en un error, y que el programa se cierre quedandonos sin entender el porque.

Esto es debido a que el Sistema Operativo, "valida" la cabecera PE, y sus secciones, es decir que cualquier descuido que tengamos al agregar la sección se reflejara en un error en la ejecucion del archivo.

Bueno para empezar vamos a agregar una sección de datos, esto es algo sin sentido, no tiene mucha utilidad, pero nos sirve para practicar como agregar la sección correctamente, sin tener errores en la "validacion".

Los pasos a seguir serian:

1) Analizar y Calcular los valores para la nueva sección (VirtualAddress, VirtualSize, PointerToRawData,SizeOfRawData, Characteristics).

2) Agregar al final de la lista de secciones la nueva sección.

3) Modificar los valores en la cabecera PE para que admita la nueva sección.

4) Agregar la sección donde acordamos que va a estar ubicada.

Bueno empezemos analizando que valores irian para cada uno.

PointerToRawData que era??, donde comienza la sección en el archivo. Y donde la vamos a poner?? en donde tengamos lugar libre, y para no complicar las cosas, eso es al final del archivo. Entonces el valor va coincidir con el tamaño del archivo.

SizeOfRawData, Esto sera el tamaño de los datos que va a contener la sección, hasta ahora no especifique que datos eran, pero para hacer las cosas simples vamos a poner una frase de 0x50 bytes de longuitud.

VirtualAddress, este seria el offset en memoria virtual donde se carga la sección, para saber que valor poner, tenemos q saber que valores hay libres, ya que si ponemos un valor cualquiera x ej 0x3000 este puede coincidir con el de otra sección, y eso provocaria un error al cargarse el archivo. Para hacerlo prollijo, hallamos el valor de la siguiente direccion virtual disponible, esto se puede hacer de 2 maneras.

Hallar la sección que tiene la VirtualAddress mas grande, sumarle su VirtualSize y redondear al multiplo de SectionAlignment (0x1000).

La otra forma, seria mirar SizeOfImage, y redondear al multiplo de SectionAlignment (que comunmente es 0x1000).

VirtualSize era el tamaño en memoria que ocupa la sección, como restriccion tenemos que debe ser mayor o igual que SizeOfRawData, osea que como valor podemos poner el tamaño, asi que nos queda 0x50.

Ahora vamos a aplicar todo lo dicho al archivo notepad.exe.

Definimos una frase aleatoria para usar como dato para el ejemplo:

"Hola yo soy la sección de prueba, y ocupo exactamente la cantidad de 0x50 bytes." (realmente ocupa eso la frase jeje)

El tamaño del notepad.exe, al menos en mi xp, es de 0x11200 bytes (70.144), en esa

posicion vamos a poner nuestra frase.

Y el valor de SizeOfImage es de 0x14000, como ya esta redondeado, usamos este.

En Characteristics vamos a definir la sección como de datos inicializados, y solo de lectura.

Entonces los datos quedarian asi:

Name: ".Ferchu" (yo pongo este, ustedes el nombre que quieran no mas grande que 8)
PointerToRawData: 0x11200
SizeOfRawData: 0x50
VirtualAddress: 0x14000
VirtualSize : 0x50
Characteristics : 0x40000040 (Ver msdn)

Luego agarramos el editor hexa, abrimos el notepad.exe vamos hasta donde esta la ultima sección, mui contento a agregar nuestra sección a la lista, pero al ubicar la posicion donde iria, nos encontramos con que ahi hay datos, usara esos datos el programa o sera basura??. Entonces abrimos el notepad con el olly, y si revisamos y miramos bien la cabecera PE, vemos que hay una tabla llamada "Bound Import Table" que hace referencia a esa posicion, es decir que al cargarse el archivo el SO va a ir a buscar esa tabla. Pero sera verdad todo esto?, entonces vamos a sacarnos la duda, modificamos alguno de sos valores, guardamos y vemos si pasa algo. Si hacemos todo eso vemos un lindo mensaje de error.

Como hacemos? que otra posiblidad hay?, una seria mover la tabla hacia otro lugar y modificar el puntero de la cabecera PE hacia la nueva posicion, asi tenemos lugar para añadir nuestros datos, pero como el archivo es de windows, ni en **** vamos a encontrar un espacio vacio disponible para copiar esa tabla. Luego pensamos, sera importante esa tabla?? ya probamos que si la modificamos tira error, pero y si la desvinculamos, si eliminamos el puntero q hace referencia a ella y hacemos como que no existe, pasara algo?, efectivamente si ponemos el valor de Bound Import Table address en 0, y hacemos lo mismo con el de size, guardamos y probamos el notepad vemos que arranca perfectamente.

Si no entendieron con exactitud lo que se dijo en los 2 parrafos anteriores, no se preocupen que no es muy importante, solo es para explicar el motivo por el cual se hace eso. En pocas palabras, ponemos en cero el puntero de esa tabla para que el SO piense que no existe, y No la vaya a buscar cuando se ejecuta el archivo, eso nos da la libertad de modificar los datos que antes no se podian sin que ocurra un error, asi que ahora si a agregar la sección a el final de la lista.

Los datos que vamosa escribir seran estos:

Nombre	Valor	Tamaño
Name:	.Ferchu	(8 bytes)

VirtualSize:	0x00000050	(4 bytes)
VirtualAddress:	0x00014000	(4 bytes)
SizeOfRawData:	0x00000050	(4 bytes)
PointerToRawData:	0x00011200	(4 bytes)
PointerToRelocations:	No interesa, todo a cero.	(4 bytes)
PointerToLinenumbers:	No interesa, todo a cero.	(4 bytes)
NumberOfRelocations:	No interesa, todo a cero.	(2 bytes)
NumberOfLinenumbers:	No interesa, todo a cero.	(2 bytes)
Characteristics:	0x40000040 (Ver msdn)	(4 bytes)

Esta vez si los datos estan en orden de aparicion, asi que solo hay que tener en cuenta al escribirlos, que van al revez (de atras hacia adelante), como se muestra en la foto.

La ultima sección termina en 0x24f, asi que la nuestra se empieza a definir en la posicion 0x250 del archivo, como se ve en la foto.

```
000001D0  00 00 00 00 00 00 00 00  2E 74 65 78 74 00 00 00  .........text...
000001E0  48 77 00 00 00 10 00 00  00 78 00 00 00 04 00 00  Hw.......x......
000001F0  00 00 00 00 00 00 00 00  00 00 00 00 20 00 00 60  ............ ..`
00000200  2E 64 61 74 61 00 00 00  A8 1B 00 00 00 90 00 00  .data...¨....▌..
00000210  00 08 00 00 00 7C 00 00  00 00 00 00 00 00 00 00  .....|..........
00000220  00 00 00 00 40 00 00 C0  2E 72 73 72 63 00 00 00  ....@..À.rsrc...
00000230  20 8D 00 00 00 B0 00 00  00 8E 00 00 00 84 00 00   ▌...°...▌...▌..
00000240  00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 40  ............@..@
00000250  2E 46 65 72 63 68 75 00  40 00 00 00 00 40 01 00  .Ferchu.@....@..
00000260  40 00 00 00 00 12 01 00  00 00 00 00 00 00 00 00  @...............
00000270  00 00 00 00 40 00 00 40  0C 2B 25 41 96 00 00 00  ....@..@.+%A▌...
00000280  9B 2C 25 41 A3 00 01 00  9A 2C 25 41 B0 00 00 00  ▌,%A£...▌,%A°...
00000290  FA 2B 25 41 BA 00 00 00  7F 2C 25 41 C4 00 00 00  ú+%Aº...▌,%AÄ...
000002A0  00 00 00 00 00 00 00 00  63 6F 6D 64 6C 67 33 32  ........comdlg32
000002B0  2E 64 6C 6C 00 53 48 45  4C 4C 33 32 2E 64 6C 6C  .dll.SHELL32.dll
000002C0  00 57 49 4E 53 50 4F 4F  4C 2E 44 52 56 00 43 4F  .WINSPOOL.DRV.CO
000002D0  4D 43 54 4C 33 32 2E 64  6C 6C 00 6D 73 76 63 72  MCTL32.dll.msvcr
```

Lo de azul es lo modificado.

Luego sigue modificar los valores en la cabecera PE. Se puede lograr que se cargue la sección solo modificando 2, el de NumberOfSections, y SizeOfImage. El primero, como seguramente piensan, se incrementa en 1. y al segundo se le suma el tamaño virtual de la nueva sección.

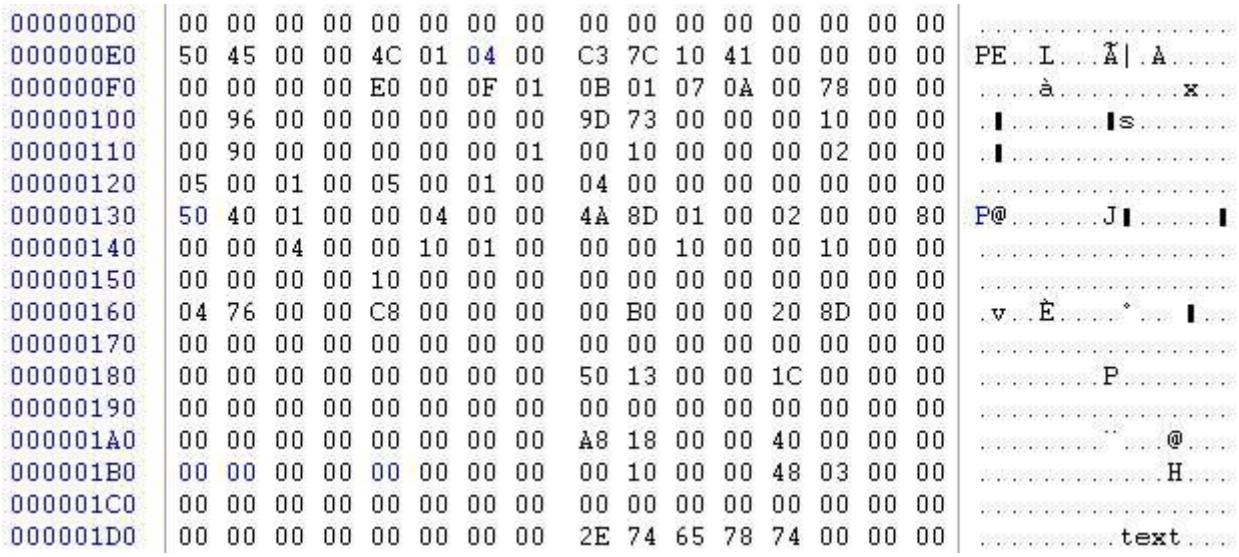
NumberOfSections: 0x0003 -----> 0x0004
SizeOfImage: 0x00014000 -----> 0x00014050

Bound Import Table address: 0x250 -----> 0x0
Bound Import Table size: 0xD0 -----> 0x0

Los valores de "Bound Import Table" lo modificamos por lo dicho anteriormente, estan ubicados en el archivo en el offset PE + 0xD0 y offset PE + 0xD4, Los ponemos a 0 y listo.

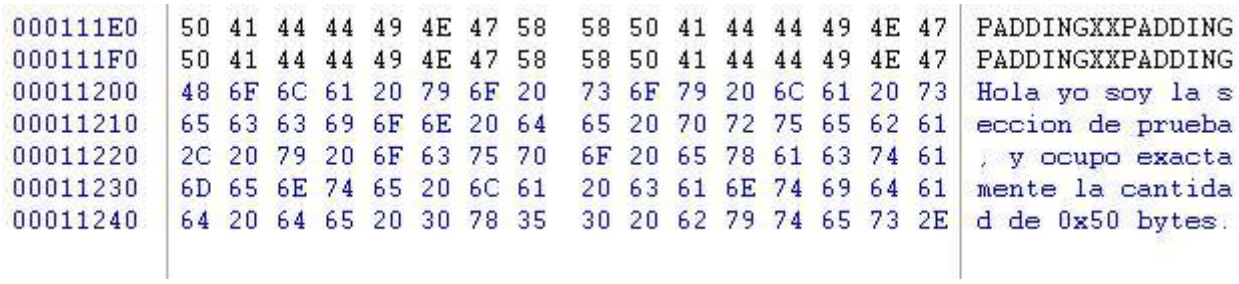
SizeOfData no la incluimos por que no es necesario modificarla, pero seguro que si no la modificamos, la suma de las secciones de datos inicializados es diferente a el valor de esta, y puede que un AV detecte que nuestra sección fue agregada a mano por no coincidir los valores, pero para nostros eso nos interesa.

Supuestamente conocen ya la posicion donde se encuentran esos valores, asi q solo hay q cambinar 2 numer con el editor hexa. Y quedaria asi:



000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000E0	50	45	00	00	4C	01	04	00	C3	7C	10	41	00	00	00	00	PE..L...Ã|.A....
000000F0	00	00	00	00	E0	00	0F	01	0B	01	07	0A	00	78	00	00	....à.........x..
00000100	00	96	00	00	00	00	00	00	9D	73	00	00	00	10	00	00	.▌.....▌s.....
00000110	00	90	00	00	00	00	00	01	00	10	00	00	00	02	00	00	.▌..............
00000120	05	00	01	00	05	00	01	00	04	00	00	00	00	00	00	00	.....
00000130	50	40	01	00	00	04	00	00	4A	8D	01	00	02	00	00	80	P@......J▌.....▌
00000140	00	00	04	00	00	10	01	00	00	00	10	00	00	10	00	00	.....
00000150	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	00	.....
00000160	04	76	00	00	C8	00	00	00	00	B0	00	00	20	8D	00	00	.v..È....°.. ▌..
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000180	00	00	00	00	00	00	00	00	50	13	00	00	1C	00	00	00	.....P.....
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001A0	00	00	00	00	00	00	00	00	A8	18	00	00	40	00	00	00	.....¨...@...
000001B0	00	00	00	00	00	00	00	00	00	10	00	00	48	03	00	00	.....H...
000001C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001D0	00	00	00	00	00	00	00	00	2E	74	65	78	74	00	00	00	.....text...

Y finalmente agregamos nuestra frase "Hola yo soy la sección de prueba, y ocupo exactamente la cantidad de 0x50 bytes." al final del archivo, como los datos de la nueva sección.



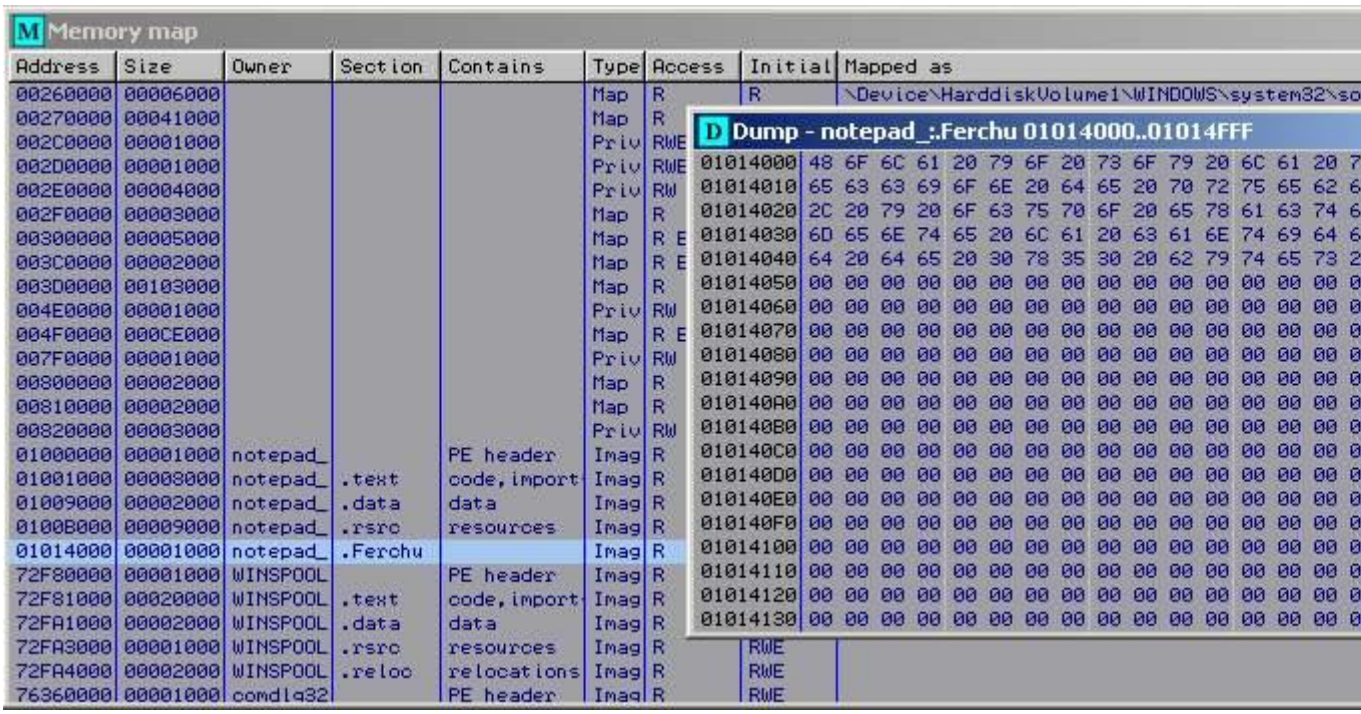
000111E0	50	41	44	44	49	4E	47	58	58	50	41	44	44	49	4E	47	PADDINGXXPADDING
000111F0	50	41	44	44	49	4E	47	58	58	50	41	44	44	49	4E	47	PADDINGXXPADDING
00011200	48	6F	6C	61	20	79	6F	20	73	6F	79	20	6C	61	20	73	Hola yo soy la s
00011210	65	63	63	69	6F	6E	20	64	65	20	70	72	75	65	62	61	eccion de prueba
00011220	2C	20	79	20	6F	63	75	70	6F	20	65	78	61	63	74	61	, y ocupo exacta
00011230	6D	65	6E	74	65	20	6C	61	20	63	61	6E	74	69	64	61	mente la cantida
00011240	64	20	64	65	20	30	78	35	30	20	62	79	74	65	73	2E	d de 0x50 bytes.

Y listo.

Ahora solo queda ver si hicimos las cosas bien, para eso usamos el olly y abrimos el notepad.exe (o el nombre que usaron) que acabamos de modificar. Si al abrirlo nos sale algun cartelito de de error, de seguro algun valor pusimos mal, en caso de que no haya

ninguna advertencia procedemos a ver si la sección fue cargada en memoria, para eso apretamos en la M que esta en la barra y buscamos la sección por el nombre que le dimos, si la encontramos hacemos 2 click y deberia aparecer los datos de la sección, osea la frase.

A mi me queda algo asi:



Y eso es todo, logramos que cargue la sección correctamente. Como se dijo al principio esto es de practica, agregar solo la sección asi no tiene mucha utilidad, en el proximo capitulo se hara algo mas funcional pero ya no se explicara tan detalladamente, por eso es que ahora se explica bien para que no se pierdan, luego iremos mas rapido.

Fin del segundo capitulo.

Nota: no pregunten cosas como x ej, como manejar un editor hexa, en el post, para eso abran un tema aparte asi no mesclamos las cosas.

Capitulo III: Agregando una sección con codigo.

23/04/08

En este capitulo vamos a hacer una infeccion simple solo para demostrar el concepto, si bien se puede infectar un ejecutable con codigo sobre una sección ya existente, nosotros lo vamos a hacer en una sección nueva, para utilizar lo aprendio hasta ahora.

La infeccion mas simple que se puede hacer, es que el programa antes de ejecutarce normalmente muestre un cuadro de mensaje (MessageBox), con algun texto de alerta que nosotros elijamos, esto no afectara en nada al funcionamiento del programa, tampoco es nada maligno, ya que esas cosas no se tocaran en este taller.

Se supone que ya conocen como agregar una sección, asi que no vamos a perder mucho tiempo en eso, solo se explicara detalladamente las cosas nuevas.

El metodo para agregar la sección es igual que en el capitulo anterior, pero con la diferencia que ahora en vez de desvincular la "Bound Import Table", lo que vamos a hacer es un truquito que consiste en bajar la cabecera PE 0x28 bytes, para tener lugar para definir nuestra sección sin pisar la tabla. Al bajar la cabecera pisamos datos que no son importantes. Luego solo modificamos en offset PE, y listo, agregamos la sección sin desvincular la tabla.

Los datos a modificar en la cabecera entonces son:

Nombre	Valor	Tamaño
offset PE:	0xE0 - 0x28 = 0xB8	(4 bytes)
NumberOfSections:	0x0004 (+1)	(2 bytes)
AddressOfEntryPoint:	0x00014031	(4 bytes)
SizeOfImagebase:	0x00014050	(4 bytes)

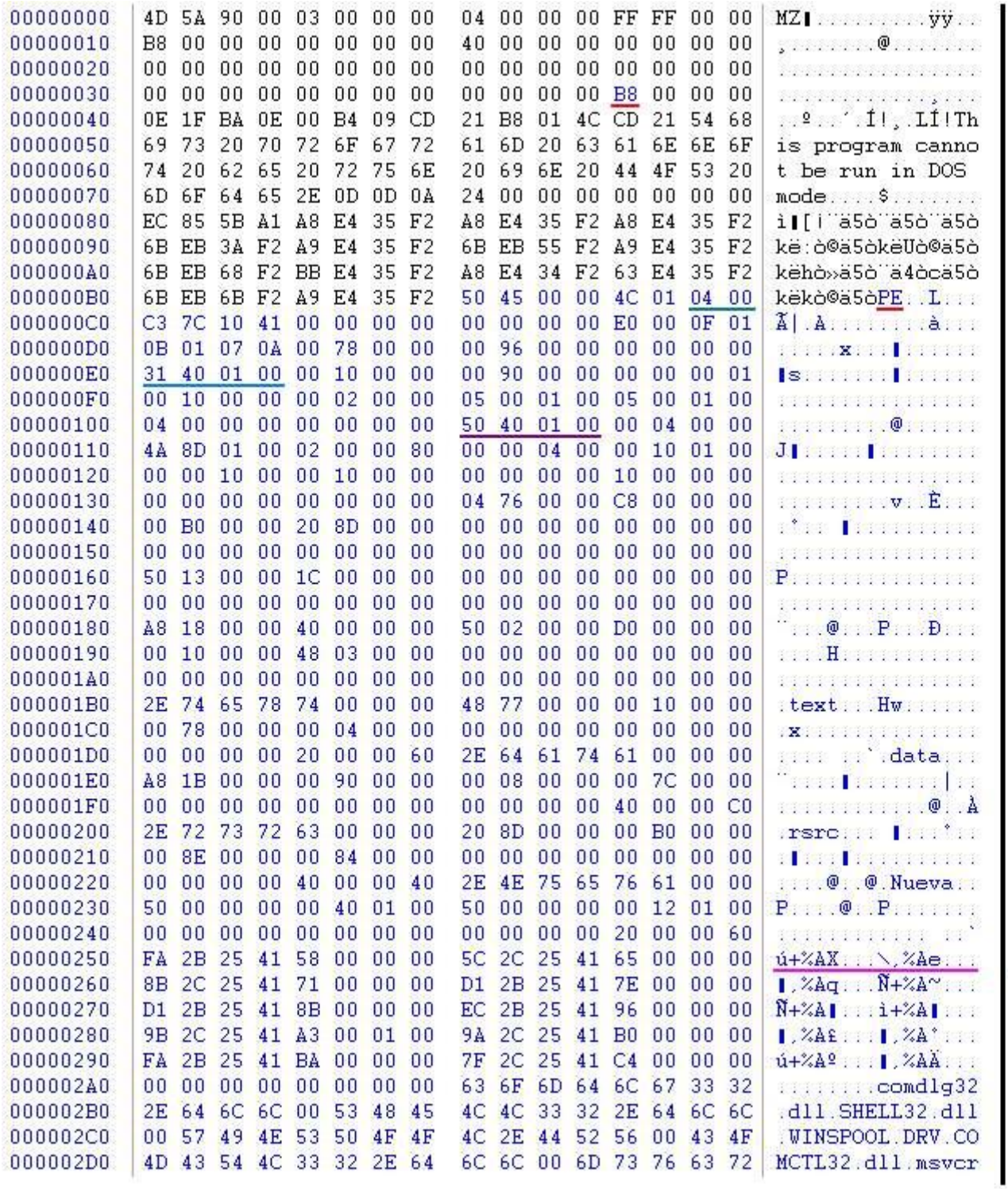
Como notaran, ahora modificamos el OEP, con el de nuestro codigo (como ya tengo el codigo echo ya se que empiza en esa dire), y debemos anotar el OEP original del programa para hacer un salto luego de que nuestro codigo finalize, para que comienze el programa normalmente.

sección nueva:

Nombre	Valor	Tamaño
Name:	.Nueva	(8 bytes)
VirtualSize:	0x00000050	(4 bytes)
VirtualAddress:	0x00014000	(4 bytes)
SizeOfRawData:	0x00000050	(4 bytes)

PointerToRawData:	0x00011200	(4 bytes)
PointerToRelocations:	No interesa, todo a cero.	(4 bytes)
PointerToLinenumbers:	No interesa, todo a cero.	(4 bytes)
NumberOfRelocations:	No interesa, todo a cero.	(2 bytes)
NumberOfLinenumbers:	No interesa, todo a cero.	(2 bytes)
Characteristics:	0x60000020 (Ver msdn)	(4 bytes)

Y como una Imagen vale mas que mil palabras...



Modificado

Se puede apreciar en la comparacion la cabecera PE desplazada 0x28 bytes hacia abajo (considerese abajo, como desplazamiento hacia menor direccion que la actual), y se ve la "Bound Import Table" intacta (la que en el otro capitulo eliminamos/sobreescribimos).

Ahora necesitamos crear el codigo que se va a ejecutar en la nueva sección, para hacerlo

simple vamos a utilizar la direccion harcoded, de la funcion MessageBoxA. No les voy a enseñar como crear un codigo ni nada por el estilo ya que eso no se relaciona con el taller, hay muchas formas de crear un codigo en asm portable, sin basarse en direcciones hardcoded, pero para hacer algo bien simple y ya que no necesitamos portabilidad, utilizamos la direccion de la funcion.

Entonces este sera el codigo en asm que ira en la sección nueva, es bastante simple, para que lo pueda entender cualquiera, por desgracia yo solo aprendi a usar el registro "eax", jajaj, asi que solo use ese.

Código:

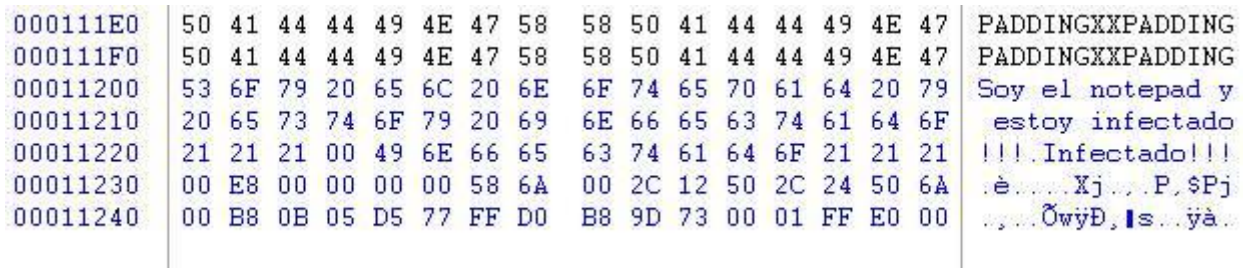
```
codigo                                  OPcodes

Cadena titulo                           "Soy el notepad y estoy
infectado!!!\0"
Cadena msg                              "Infectado!!!\0"
call [siguiente instruccion]            E8 00 00 00 00
pop eax                                 58
push 0                                  6A 00
sub eax, 0x12                           2C 12
push eax                                50
sub eax, 0x24                           2C 24
push eax                                50
push 0                                  6A 00
mov eax, 0x77D5050B                     B8 0B 05 D5 77       // B8 +
direccion de MessageBoxA
call eax                                FF D0
mov eax, 0x010739D                      B8 9D 73 00 01       // B8 +
entry point
jmp eax                                 FF E0
```

El codigo empieza con las cadenas de texto para el titulo y el mensaje, luego viene un call a la siguiente instruccion, es un pequeño truquito, la instruccion call antes de ir a la subrutina mete la direccion de su siguiente instruccion en la pila y luego salta hacia la direccion, pero en este caso la utilizamos para obtener una referencia sobre que direccion estamos parados, asi luego poder obtener la direccion de las cadenas de texto. Luego se meten a la pila los argumentos para el messagebox, se guarda la direccion de la funcion en eax, y se llama a la funcion, despues se guarda en entry point en eax, para saltar hacia alli y que comienze el programa como si nada hubiera pasado.

Para los que se estan preguntando por q no hice jmp [direccion] directamente, bueno la respuesta es simple, tanto los jmps como los calls tiene en los opcodes la distancia entre la direccion de la instruccion y la direccion a saltar, y para no estar haciendo cuentas, es mas facil asi, ademas de que necesitamos la direccion de esa instruccion. Y sino, creanme que es mas facil asi jeje.

Nota: La direccion de MessageBoxA esta harcoded, asi que deberan buscar la que corresponde a su windows, puede que coincida, como que no. pero lo mejor es verificarlo.

Ahora escribimos esos opcodes al final del archivo, y nos queda algo asi:



```
000111E0  50 41 44 44 49 4E 47 58  58 50 41 44 44 49 4E 47  PADDINGXXPADDING
000111F0  50 41 44 44 49 4E 47 58  58 50 41 44 44 49 4E 47  PADDINGXXPADDING
00011200  53 6F 79 20 65 6C 20 6E  6F 74 65 70 61 64 20 79  Soy el notepad y
00011210  20 65 73 74 6F 79 20 69  6E 66 65 63 74 61 64 6F   estoy infectado
00011220  21 21 21 00 49 6E 66 65  63 74 61 64 6F 21 21 21  !!!.Infectado!!!
00011230  00 E8 00 00 00 00 58 6A  00 2C 12 50 2C 24 50 6A  .è....Xj.,.P,$Pj
00011240  00 B8 0B 05 D5 77 FF D0  B8 9D 73 00 01 FF E0 00  .¸..ÕwÿÐ¸.s..ÿà.
```

Y listo, ya tenemos nuestro notepad infectado. Lo ejectuamos, vemos nuestro mensajito, le damos aceptar y luego se ejecutara todo normalmente.

No me responsabilizo de sus acciones, ni de la forma que utilizan lo aprendido en este taller.

http://foro.elhacker.net/analisis_y_diseno_de_malware/abril_negro_2008_taller_de_formato_pe_by_ferchu-t208278.0.html#ixzz1A51HufLh