



Introduction au format Portable Executable

Abstract

Ce document présente le format natif des fichiers exécutables dans l'environnement logiciel Microsoft Windows : le format Portable Executable.

La dernière version de ce document est accessible sur le Web à l'adresse :

<http://esl.epitech.net/~arnaud/introduction-au-format-portable-executable>

Pour toutes questions et commentaires concernant ce document, merci de contacter:

maillard.arnaud@gmail.com

Table des matières

Introduction.....	2
Vue d'ensemble du format	3
L'entête DOS	5
IMAGE_DOS_HEADER	5
DOS STUB	5
L'entête PE (IMAGE_NT_HEADERS)	8
IMAGE_FILE_HEADER	9
IMAGE_OPTIONAL_HEADER	13
Les entêtes de sections (IMAGE_SECTION_HEADER).....	20
Analyse d'un programme PE	25
Code source	25
Analyse	25
Les sections	29
Section de code	29
Section de donnée.....	29
Section des importations	30
Annexe I. Retrouver l'adresses des APIs	37
Fondamentaux: processus et threads	37
Importations et exportations: le module Kernel32	37
L'adresse de base de Kernel32.dll	38
Stack Pointer	39
PEB: Process Environment Block	40
Conclusion & ressources	42
Ressources	42

Introduction

Le format **PE**, pour **Portable Executable**, est un format de fichier binaire utilisé par les systèmes d'exploitation Microsoft Windows. Il est dérivé de la spécification **COFF (Common Object File Format)** et pose une structure formelle à laquelle doivent répondre:

- Les fichiers exécutables
- Les DLL (Dynamic-Link Library)
- Les pilotes de périphériques (drivers)

Microsoft a d'abord introduit ce format sur son système Windows NT 3.1. Par la suite, le format a progressivement évolué pour s'adapter aux architectures matérielles (avec les processeurs trente-deux puis soixante-quatre bits) et logicielles (avec le langage **.NET** par exemple).

Un fichier PE contient :

- Un certain nombre de définition de structures
- Des sections

Les structures définissent les informations nécessaires au *loader* Windows pour charger le module et préparer son exécution.

Une section est une entité logique définie par un début, une taille et des caractéristiques. On groupe dans une section l'ensemble des données devant être régies par un même schéma de protection et d'exécution de la mémoire (read/write/execute).

Avant de poursuivre, il est nécessaire de préciser quelques définitions:

- Espace d'adressage : il s'agit de la plage d'adresses mémoire à laquelle un processus peut accéder.
- Adresse virtuelle (*virtual address*, *VA*) : il s'agit de l'adresse mémoire d'une donnée, c'est à dire sa position dans l'espace d'adressage du processus.
- Adresse virtuelle relative (*relative virtual address*, *RVA*) : il s'agit de l'adresse d'une donnée une fois chargée en mémoire. Ces adresses sont relatives à une adresse de base, exprimée par le champ *ImageBase* du format, ce champ précisant à quelle adresse le programme sera chargé en mémoire.

Vue d'ensemble du format

Le format PE est une unité logique que l'on peut diviser en plusieurs parties, sous-ensembles qui vont constituer un programme : code, données, mécanismes de gestion interne, etc.

On peut compter quatre sous-ensembles généraux :

- L'entête DOS et le « DOS Stub » : cette portion du fichier est présente pour des raisons de compatibilité avec l'environnement logiciel MS-DOS (Microsoft Disk Operating System)
- L'entête PE : il s'agit de l'ensemble des structures d'entêtes, normalisées par la spécification PE.
- Les entêtes de sections : ici, on trouve les structures permettant la définition des sections, indiquant pour chacune un début, une limite (sa taille) et ses propriétés
- Les sections.

Entête DOS

- IMAGE_DOS_HEADER
- DOS Stub

Entête PE

- IMAGE_NT_HEADERS
 - IMAGE_FILE_HEADER
 - IMAGE_OPTIONAL_HEADER

(1) IMAGE_SECTION_HEADER

(2) IMAGE_SECTION_HEADER

(3) IMAGE_SECTION_HEADER

(N) IMAGE_SECTION_HEADER

Section 1

Section 2

Section 3

Section N

L'entête DOS

IMAGE_DOS_HEADER

Pour des raisons de compatibilité, la première structure de donnée que l'on retrouve dans un fichier au format PE est similaire à celle des fichiers exécutables de l'environnement Microsoft DOS.

La seule différence tient en l'ajout d'un champ, **e_lfanew**, qui représente un offset vers l'entête PE.

```
typedef struct _IMAGE_DOS_HEADER {  
    WORD    e_magic;           // Magic number  
    WORD    e_cblp;           // Bytes on last page of file  
    WORD    e_cp;             // Pages in file  
    WORD    e_crlc;           // Relocations  
    WORD    e_cparhdr;        // Size of header in paragraphs  
    WORD    e_minalloc;        // Minimum extra paragraphs needed  
    WORD    e_maxalloc;        // Maximum extra paragraphs needed  
    WORD    e_ss;             // Initial (relative) SS value  
    WORD    e_sp;             // Initial SP value  
    WORD    e_csum;           // Checksum  
    WORD    e_ip;             // Initial IP value  
    WORD    e_cs;             // Initial (relative) CS value  
    WORD    e_lfarlc;         // File address of relocation table  
    WORD    e_ovno;           // Overlay number  
    WORD    e_res[4];         // Reserved words  
    WORD    e_oemid;          // OEM identifier (for e_oeminfo)  
    WORD    e_oeminfo;        // OEM information; e_oemid specific  
    WORD    e_res2[10];        // Reserved words  
    LONG    e_lfanew;         // File address of new exe header  
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

A l'heure actuelle, seulement deux champs de cette structure ont un intérêt, **e_magic** et **e_lfanew**.

IMAGE_DOS_HEADER

Champ	Offset	Taille
e_magic	0x00	WORD
e_lfanew	0x3c	DWORD

e_magic

Signature d'un exécutable DOS.

```
#define IMAGE_DOS_SIGNATURE 0x4D5A // MZ
```

e_lfanew

Offset vers l'entête PE.

DOS STUB

A la suite de l'IMAGE_DOS_HEADER, on trouve quelquefois une région appelée le **DOS Stub**. Il s'agit d'une zone facultative destinée à prévenir l'utilisateur lorsqu'il tente d'exécuter un programme Windows à l'intérieur d'un environnement DOS.

Par exemple, sur le fichier 'c:\windows\notepad.exe', on trouve les données suivantes:

```
00000000  4D 5A          ASCII "MZ"          ; DOS EXE Signature  
00000002  9000          DW 0090            ; DOS_PartPag = 90 (144.)
```

```

00000004  0300      DW 0003      ; DOS_PageCnt = 3
00000006  0000      DW 0000      ; DOS_ReloCnt = 0
00000008  0400      DW 0004      ; DOS_HdrSize = 4
0000000A  0000      DW 0000      ; DOS_MinMem = 0
0000000C  FFFF      DW FFFF      ; DOS_MaxMem = FFFF (65535.)
0000000E  0000      DW 0000      ; DOS_ReloSS = 0
00000010  B800      DW 00B8      ; DOS_ExeSP = B8
00000012  0000      DW 0000      ; DOS_ChkSum = 0
00000014  0000      DW 0000      ; DOS_ExeIP = 0
00000016  0000      DW 0000      ; DOS_ReloCS = 0
00000018  4000      DW 0040      ; DOS_Tabloff = 40
0000001A  0000      DW 0000      ; DOS_Overlay = 0
0000001C  00        DB 00
0000001D  00        DB 00
0000001E  00        DB 00
0000001F  00        DB 00
00000020  00        DB 00
00000021  00        DB 00
00000022  00        DB 00
00000023  00        DB 00
00000024  00        DB 00
00000025  00        DB 00
00000026  00        DB 00
00000027  00        DB 00
00000028  00        DB 00
00000029  00        DB 00
0000002A  00        DB 00
0000002B  00        DB 00
0000002C  00        DB 00
0000002D  00        DB 00
0000002E  00        DB 00
0000002F  00        DB 00
00000030  00        DB 00
00000031  00        DB 00
00000032  00        DB 00
00000033  00        DB 00
00000034  00        DB 00
00000035  00        DB 00
00000036  00        DB 00
00000037  00        DB 00
00000038  00        DB 00
00000039  00        DB 00
0000003A  00        DB 00
0000003B  00        DB 00
0000003C  E0000000  DD 000000E0      ; Offset to PE signature

```

Après cette zone, on retrouve à l'offset 0x40 le DOS Stub:

```

00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  °.´.í!,Lí!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.

```

Une fois cette zone désassemblé, cela donne:

```

seg000:0000 start      proc near
seg000:0000      push    cs
seg000:0001      pop     ds
seg000:0002      assume  ds:seg000
seg000:0002      mov     dx, 0Eh
seg000:0005      mov     ah, 9
seg000:0007      int     21h      ; DOS - PRINT STRING
seg000:0007      " $"      ; DS:DX -> string terminated by
seg000:0009      mov     ax, 4C01h
seg000:000C      int     21h      ; DOS - 2+ - QUIT WITH EXIT CODE
seg000:000C      (EXIT)
seg000:000C start      endp      ; AL = exit code
seg000:000C

```

```

seg000:000C ;
-----
seg000:000E aThisProgramCan db 'This program cannot be run in DOS mode.',0Dh,0Dh,0Ah
seg000:000E                      db '$',0

```

Il s'agit ici d'un code 16 bits, utilisant les interruptions du système d'exploitation DOS afin d'afficher un message prévenant l'utilisateur que ce programme nécessite un environnement Windows pour fonctionner.

Le programme suivant ouvre un fichier, vérifie la signature validant un exécutable DOS et reporte l'offset vers l'entête PE:

```

/**
 ** peviewimagedosheader.c
 */

#include <windows.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    PIMAGE_DOS_HEADER pImageDosHeader;
    HANDLE hFile;
    HANDLE hMapObject;
    PUCHAR uFileMap;

    if (argc < 2)
        return (-1);

    if (!(hFile = CreateFile(argv[1], GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, 0)))
        return (-1);

    if (!(hMapObject = CreateFileMapping(hFile, NULL, PAGE_READONLY, 0, 0, NULL)))
        return (-1);

    if (!(uFileMap = MapViewOfFile(hMapObject, FILE_MAP_READ, 0, 0, 0)))
        return (-1);

    pImageDosHeader = (PIMAGE_DOS_HEADER) uFileMap ;
    if (pImageDosHeader->e_magic != IMAGE_DOS_SIGNATURE)
        return (-1);

    printf("e_magic:    0x%04X (%c%c)\n", pImageDosHeader->e_magic, *uFileMap,
*(uFileMap + 1));
    printf("e_lfanew:    0x%08X\n", pImageDosHeader->e_lfanew);

    return (0);
}

C:\>peviewimagedosheader.exe c:\WINDOWS\notepad.exe
e_magic:    0x5A4D (MZ)
e_lfanew:    0x000000E0

```

L'entête PE (IMAGE_NT_HEADERS)

L'entête PE est définie par une structure de type **IMAGE_NT_HEADERS**. Cette structure est en fait un conteneur englobant un DWORD de signature et les structures :

- **IMAGE_FILE_HEADER**
- **IMAGE_OPTIONAL_HEADER**

```
typedef struct _IMAGE_NT_HEADERS {  
    DWORD Signature;  
    IMAGE_FILE_HEADER FileHeader;  
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;  
} IMAGE_NT_HEADERS, *PIMAGE_NT_HEADERS;
```

IMAGE_NT_HEADERS

Champ	Offset	Taille
Signature	0x000	DWORD
FileHeader	0x004	sizeof(IMAGE_FILE_HEADER)
OptionalHeader	0x018	sizeof(IMAGE_OPTIONAL_HEADER)

Signature

Signature d'un fichier au format Portable Executable.

```
#define IMAGE_NT_SIGNATURE 0x00004550 // PE00
```

FileHeader

Structure de type IMAGE_FILE_HEADER.

Machine

Structure de type IMAGE_OPTIONAL_HEADER.

IMAGE_FILE_HEADER

A l'intérieur de l'IMAGE_NT_HEADERS, après la définition de l'élément *Signature* débute une structure de type IMAGE_FILE_HEADER.

```
typedef struct _IMAGE_FILE_HEADER {
    WORD    Machine;
    WORD    NumberOfSections;
    DWORD   TimeDateStamp;
    DWORD   PointerToSymbolTable;
    DWORD   NumberOfSymbols;
    WORD    SizeOfOptionalHeader;
    WORD    Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;

#define IMAGE_SIZEOF_FILE_HEADER 20
```

IMAGE_FILE_HEADER

Champ	Offset	Taille
Machine	0x00	WORD
NumberOfSections	0x02	WORD
TimeDateStamp	0x04	DWORD
PointerToSymbolTable	0x08	DWORD
NumberOfSymbols	0x0C	DWORD
SizeOfOptionalHeader	0x10	WORD
Characteristics	0x12	WORD

Machine

Type de machine pour lequel est destiné l'exécutable.

```
#define IMAGE_FILE_MACHINE_UNKNOWN 0
#define IMAGE_FILE_MACHINE_I386 0x014c // Intel 386.
#define IMAGE_FILE_MACHINE_R3000 0x0162 // MIPS little-endian,
0x160 big-endian
#define IMAGE_FILE_MACHINE_R4000 0x0166 // MIPS little-endian
#define IMAGE_FILE_MACHINE_R10000 0x0168 // MIPS little-endian
#define IMAGE_FILE_MACHINE_WCEMIPSV2 0x0169 // MIPS little-endian
WCE v2
#define IMAGE_FILE_MACHINE_ALPHA 0x0184 // Alpha AXP
#define IMAGE_FILE_MACHINE_SH3 0x01a2 // SH3 little-endian
#define IMAGE_FILE_MACHINE_SH3DSP 0x01a3
#define IMAGE_FILE_MACHINE_SH3E 0x01a4 // SH3E little-endian
#define IMAGE_FILE_MACHINE_SH4 0x01a6 // SH4 little-endian
#define IMAGE_FILE_MACHINE_SH5 0x01a8 // SH5
#define IMAGE_FILE_MACHINE_ARM 0x01c0 // ARM Little-Endian
#define IMAGE_FILE_MACHINE_THUMB 0x01c2
#define IMAGE_FILE_MACHINE_AM33 0x01d3
#define IMAGE_FILE_MACHINE_POWERPC 0x01f0 // IBM PowerPC Little-
Endian
#define IMAGE_FILE_MACHINE_POWERPCFP 0x01f1
#define IMAGE_FILE_MACHINE_IA64 0x0200 // Intel 64
#define IMAGE_FILE_MACHINE_MIPS16 0x0266 // MIPS
#define IMAGE_FILE_MACHINE_ALPHA64 0x0284 // ALPHA64
#define IMAGE_FILE_MACHINE_MIPSFPU 0x0366 // MIPS
#define IMAGE_FILE_MACHINE_MIPSFPU16 0x0466 // MIPS
#define IMAGE_FILE_MACHINE_AXP64 IMAGE_FILE_MACHINE_ALPHA64
#define IMAGE_FILE_MACHINE_TRICORE 0x0520 // Infineon
#define IMAGE_FILE_MACHINE_CEF 0x0CEF
#define IMAGE_FILE_MACHINE_EBC 0x0EBC // EFI Byte Code
#define IMAGE_FILE_MACHINE_AMD64 0x8664 // AMD64 (K8)
#define IMAGE_FILE_MACHINE_M32R 0x9041 // M32R little-endian
#define IMAGE_FILE_MACHINE_CEE 0xC0EE
```

NumberOfSections

Indique le nombre de sections présentes dans le fichier.

TimeDateStamp

Date et heure de création du fichier.

PointerToSymbolTable

Offset vers la table des symboles.

NumberOfSymbols

Nombre d'entrées dans la table des symboles.

SizeOfOptionalHeader

Taille de l'IMAGE_OPTIONAL_HEADER

Characteristics

Caractéristiques générales du fichier.

```
#define IMAGE_FILE_RELOCS_STRIPPED          0x0001 // Relocation info
stripped from file.
#define IMAGE_FILE_EXECUTABLE_IMAGE        0x0002 // File is executable
(i.e. no unresolved external references).
#define IMAGE_FILE_LINE_NUMS_STRIPPED      0x0004 // Line numbers
stripped from file.
#define IMAGE_FILE_LOCAL_SYMS_STRIPPED     0x0008 // Local symbols
stripped from file.
#define IMAGE_FILE_AGGRESSIVE_WS_TRIM      0x0010 // Aggressively trim
working set
#define IMAGE_FILE_LARGE_ADDRESS_AWARE     0x0020 // App can handle >2gb
addresses
#define IMAGE_FILE_BYTES_REVERSED_LO       0x0080 // Bytes of machine
word are reversed.
#define IMAGE_FILE_32BIT_MACHINE           0x0100 // 32 bit word machine.
#define IMAGE_FILE_DEBUG_STRIPPED          0x0200 // Debugging info
stripped from file in .DBG file
#define IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP 0x0400 // If Image is on
removable media, copy and run from the swap file.
#define IMAGE_FILE_NET_RUN_FROM_SWAP       0x0800 // If Image is on Net,
copy and run from the swap file.
#define IMAGE_FILE_SYSTEM                  0x1000 // System File.
#define IMAGE_FILE_DLL                     0x2000 // File is a DLL.
#define IMAGE_FILE_UP_SYSTEM_ONLY          0x4000 // File should only be
run on a UP machine
#define IMAGE_FILE_BYTES_REVERSED_HI       0x8000 // Bytes of machine
word are reversed.
```

Le programme suivant ouvre un fichier, vérifie les signatures DOS et PE puis affiche l'IMAGE_FILE_HEADER de l'exécutable passé en paramètres.

```
/**
 ** pevviewimagefileheader.c
 **/

#include <windows.h>
#include <stdio.h>

void viewImageFileCharacteristics(WORD);

int main(int argc, char **argv)
```

```

{
    PIMAGE_DOS_HEADER    pImageDosHeader;
    PIMAGE_NT_HEADERS    pImageNtHeaders;
    PIMAGE_FILE_HEADER    pImageFileHeader;
    HANDLE                hFile;
    HANDLE                hMapObject;
    PCHAR                 uFileMap;

    if (argc < 2)
        return (-1);

    if (!(hFile = CreateFile(argv[1], GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, 0)))
        return (-1);

    if (!(hMapObject = CreateFileMapping(hFile, NULL, PAGE_READONLY, 0, 0, NULL)))
        return (-1);

    if (!(uFileMap = MapViewOfFile(hMapObject, FILE_MAP_READ, 0, 0, 0)))
        return (-1);

    pImageDosHeader = (PIMAGE_DOS_HEADER) uFileMap ;
    if (pImageDosHeader->e_magic != IMAGE_DOS_SIGNATURE)
        return (-1);

    pImageNtHeaders = (PIMAGE_NT_HEADERS) ((PCHAR) uFileMap + pImageDosHeader->
    >e_lfanew);
    if (pImageNtHeaders->Signature != IMAGE_NT_SIGNATURE)
        return (-1);

    pImageFileHeader = (PIMAGE_FILE_HEADER) &(pImageNtHeaders->FileHeader);
    printf("Machine:                0x%04X", pImageFileHeader->Machine);
    if (pImageFileHeader->Machine == IMAGE_FILE_MACHINE_I386)
        ? printf(" (I386) \n")
        : printf(" (?) \n");
    printf("NumberOfSections:          0x%04X\n", pImageFileHeader->NumberOfSections);
    printf("TimeDateStamp:              0x%08X\n", pImageFileHeader->TimeDateStamp);
    printf("PointerToSymbolTable:        0x%08X\n", pImageFileHeader->
    >PointerToSymbolTable);
    printf("NumberOfSymbols:             0x%08X\n", pImageFileHeader->NumberOfSymbols);
    printf("SizeOfOptionalHeader:        0x%04X\n", pImageFileHeader->
    >SizeOfOptionalHeader);
    printf("Characteristics:              0x%04X", pImageFileHeader->Characteristics);
    viewImageFileCharacteristics(pImageFileHeader->Characteristics);

    return (0);
}

void    viewImageFileCharacteristics(WORD wCharacteristics)
{
    BYTE    szCharacteristics[100];

    memset(szCharacteristics, 0, 100);
    szCharacteristics[0] = '(';
    if (wCharacteristics & 0x0001)
        strcat(szCharacteristics, "RELOCS_STRIPPED|");
    if (wCharacteristics & 0x0002)
        strcat(szCharacteristics, "EXECUTABLE_IMAGE|");
    if (wCharacteristics & 0x0004)
        strcat(szCharacteristics, "LINE_NUMS_STRIPPED|");
    if (wCharacteristics & 0x0100)
        strcat(szCharacteristics, "32BIT_MACHINE|");
    if (wCharacteristics & 0x0200)
        strcat(szCharacteristics, "DEBUG_STRIPPED|");
    if (wCharacteristics & 0x1000)
        strcat(szCharacteristics, "FILE_SYSTEM|");
    if (wCharacteristics & 0x2000)
        strcat(szCharacteristics, "FILE_DLL|");

    szCharacteristics[strlen(szCharacteristics) - 1] = ')';
    szCharacteristics[strlen(szCharacteristics)] = '\0';
    printf(" %s\n", szCharacteristics);
}

```

```
C:\>peviewimagefileheader.exe c:\WINDOWS\notepad.exe
Machine:                0x014C (I386)
NumberOfSections:       0x0003
TimeDateStamp:          0x48025287
PointerToSymbolTable:   0x00000000
NumberOfSymbols:        0x00000000
SizeOfOptionalHeader:   0x00E0
Characteristics:         0x010F (RELOCS_STRIPPED|EXECUTABLE_IMAGE|LINE_NUMS_STRIPPED|
32BIT_MACHINE)
```

IMAGE_OPTIONAL_HEADER

```
#define IMAGE_NUMBEROF_DIRECTORY_ENTRIES 16

typedef struct _IMAGE_OPTIONAL_HEADER {
    USHORT    Magic;
    UCHAR     MajorLinkerVersion;
    UCHAR     MinorLinkerVersion;
    ULONG     SizeOfCode;
    ULONG     SizeOfInitializedData;
    ULONG     SizeOfUninitializedData;
    ULONG     AddressOfEntryPoint;
    ULONG     BaseOfCode;
    ULONG     BaseOfData;
    ULONG     ImageBase;
    ULONG     SectionAlignment;
    ULONG     FileAlignment;
    USHORT    MajorOperatingSystemVersion;
    USHORT    MinorOperatingSystemVersion;
    USHORT    MajorImageVersion;
    USHORT    MinorImageVersion;
    USHORT    MajorSubsystemVersion;
    USHORT    MinorSubsystemVersion;
    ULONG     Reserved1;
    ULONG     SizeOfImage;
    ULONG     SizeOfHeaders;
    ULONG     CheckSum;
    USHORT    Subsystem;
    USHORT    DllCharacteristics;
    ULONG     SizeOfStackReserve;
    ULONG     SizeOfStackCommit;
    ULONG     SizeOfHeapReserve;
    ULONG     SizeOfHeapCommit;
    ULONG     LoaderFlags;
    ULONG     NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;
```

IMAGE_OPTIONAL_HEADER

Champ	Offset	Taille
Magic	0x00	WORD
MajorLinkerVersion	0x02	BYTE
MinorLinkerVersion	0x03	BYTE
SizeOfCode	0x04	DWORD
SizeOfInitializedData	0x08	DWORD
SizeOfUninitializedData	0x0C	DWORD
AddressOfEntryPoint	0x10	DWORD
BaseOfCode	0x14	DWORD
BaseOfData	0x18	DWORD
ImageBase	0x1C	DWORD
SectionAlignment	0x20	DWORD
FileAlignment	0x24	DWORD
MajorOperatingSystemVersion	0x28	WORD
MinorOperatingSystemVersion	0x2A	WORD
MajorImageVersion	0x2C	WORD
MinorImageVersion	0x2E	WORD
MajorSubsystemVersion	0x30	WORD
MinorSubsystemVersion	0x32	WORD
Reserved1	0x34	DWORD
SizeOfImage	0x38	DWORD
SizeOfHeaders	0x3C	DWORD
CheckSum	0x40	DWORD

Champ	Offset	Taille
Subsystem	0x44	WORD
DllCharacteristics	0x46	WORD
SizeOfStackReserve	0x48	DWORD
SizeOfStackCommit	0x4C	DWORD
SizeOfHeapReserve	0x50	DWORD
SizeOfHeapCommit	0x54	DWORD
LoaderFlags	0x58	DWORD
NumberOfRvaAndSizes	0x5C	DWORD
DataDirectory	0x60	32 DWORD

Magic

Différencie une image PE 32 bits ou 64 bits

```
#define IMAGE_NT_OPTIONAL_HDR32_MAGIC    0x10b
#define IMAGE_NT_OPTIONAL_HDR64_MAGIC    0x20b
```

MajorLinkerVersion

Identifie la version majeure du linker utilisé lors de la création de l'exécutable.

MinorLinkerVersion

Identifie la version mineure du linker utilisé lors de la création de l'exécutable.

SizeOfCode

Taille cumulée des sections présentant du code. Habituellement, il s'agit de la taille de la section « .code ».

SizeOfInitializedData

Taille cumulée des sections présentant des données initialisées. Habituellement, il s'agit de la taille de la section « .data ».

SizeOfUninitializedData

Taille cumulée des sections présentant des données non initialisées. Habituellement, il s'agit de la taille de la section « .bss » ou « .rdata ».

AddressOfEntryPoint

Adresse virtuelle relative du point d'entrée du programme.

BaseOfCode

Adresse virtuelle relative du début du code.

BaseOfData

Adresse virtuelle relative du début des données.

ImageBase

Cette adresse indique au loader l'adresse de préférence à laquelle charger le fichier exécutable en mémoire. Ce champ est particulièrement important puisque les adresses virtuelles relatives le sont par rapport à cette adresse.

SectionAlignment

Alignement des sections chargées en mémoire. La taille de chacune des sections doit être un multiple de ce nombre. Le plus souvent 0x1000h, soit 4096 octets.

FileAlignment

Alignement des sections dans le fichier sur disque dur. La taille de chacune des sections doit être multiple de ce nombre. Le plus souvent 0x200, soit 512 octets.

MajorOperatingSystemVersion

Version majeure du système d'exploitation requis pour faire fonctionner l'exécutable.

MinorOperatingSystemVersion

Version mineure du système d'exploitation requis pour faire fonctionner l'exécutable.

MajorImageVersion

Version majeure de l'image. Ce champ n'est plus utilisé.

MinorImageVersion

Version mineure de l'image. Ce champ n'est plus utilisé.

MajorSubsystemVersion

Version majeure du sous système d'environnement requis pour faire fonctionner l'exécutable.

MinorSubsystemVersion

Version majeure du sous système d'environnement requis pour faire fonctionner l'exécutable.

Reserved1 -

SizeOfImage

Taille du fichier en mémoire. Ce champs doit être multiple de SectionAlignment.

SizeOfHeaders

Taille cumulée des entêtes (c'est à dire la taille des entêtes DOS et PE, mais aussi celle des sections). Ce champ doit être multiple de FileAlignment.

Checksum

Somme de contrôle de l'exécutable. Ce champ, rarement positionné, n'est pas utilisé.

Subsystem

Sous système d'environnement requis pour faire fonctionner l'exécutable.

```
#define IMAGE_SUBSYSTEM_UNKNOWN          0    // Unknown subsystem.
#define IMAGE_SUBSYSTEM_NATIVE          1    // Image doesn't require a
subsystem.
#define IMAGE_SUBSYSTEM_WINDOWS_GUI     2    // Image runs in the Windows GUI
subsystem.
#define IMAGE_SUBSYSTEM_WINDOWS_CUI     3    // Image runs in the Windows
character subsystem.
#define IMAGE_SUBSYSTEM_OS2_CUI         5    // image runs in the OS/2 character
subsystem.
#define IMAGE_SUBSYSTEM_POSIX_CUI       7    // image runs in the Posix character
subsystem.
#define IMAGE_SUBSYSTEM_NATIVE_WINDOWS  8    // image is a native Win9x driver.
#define IMAGE_SUBSYSTEM_WINDOWS_CE_GUI  9    // Image runs in the Windows CE
subsystem.
#define IMAGE_SUBSYSTEM_EFI_APPLICATION 10    //
#define IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER 11 //
#define IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER 12 //
#define IMAGE_SUBSYSTEM_EFI_ROM         13
#define IMAGE_SUBSYSTEM_XBOX            14
#define IMAGE_SUBSYSTEM_WINDOWS_BOOT_APPLICATION 16
```

DllCharacteristics

Caractéristiques supplémentaires pour le cas où le fichier est une DLL

SizeOfStackReserve

Taille maximale de la pile pour le programme

SizeOfStackCommit

Taille de la pile à allouer pour l'exécution du programme

SizeOfHeapReserve

Taille maximale du tas pour le programme

SizeOfHeapCommit

Taille du tas à allouer pour l'exécution du programme

LoaderFlags

Ce champ n'est plus utilisé.

NumberOfRvaAndSizes

Ce champ indique le nombre d'éléments du tableau DataDirectory.

DataDirectory

Il s'agit d'un Tableau de structures d'au minimum seize entrées de type IMAGE_DATA_DIRECTORY.

Chacun des des entrées dans cette table donne l'adresse virtuelle et la taille d'une section spécifique au format (que l'on appelle ici répertoire, chacun possédant sa propre structure interne). De fait, cet élément est appelé la table des répertoires.

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

#define IMAGE_NUMBEROF_DIRECTORY_ENTRIES    16

// Directory Entries

#define IMAGE_DIRECTORY_ENTRY_EXPORT        0 // Export Directory
#define IMAGE_DIRECTORY_ENTRY_IMPORT        1 // Import Directory
#define IMAGE_DIRECTORY_ENTRY_RESOURCE      2 // Resource Directory
#define IMAGE_DIRECTORY_ENTRY_EXCEPTION    3 // Exception Directory
#define IMAGE_DIRECTORY_ENTRY_SECURITY      4 // Security Directory
#define IMAGE_DIRECTORY_ENTRY_BASERELOC     5 // Base Relocation Table
#define IMAGE_DIRECTORY_ENTRY_DEBUG         6 // Debug Directory
// IMAGE_DIRECTORY_ENTRY_COPYRIGHT         7 // (X86 usage)
#define IMAGE_DIRECTORY_ENTRY_ARCHITECTURE  7 // Architecture Specific Data
#define IMAGE_DIRECTORY_ENTRY_GLOBALPTR     8 // RVA of GP
#define IMAGE_DIRECTORY_ENTRY_TLS           9 // TLS Directory
#define IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG  10 // Load Configuration Directory
#define IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT  11 // Bound Import Directory in headers
#define IMAGE_DIRECTORY_ENTRY_IAT          12 // Import Address Table
#define IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT  13 // Delay Load Import Descriptors
#define IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14 // COM Runtime descriptor
```

Le programme suivant ouvre un fichier, vérifie les signatures DOS et PE puis affiche l'IMAGE_OPTIONAL_HEADER de l'exécutable passé en paramètres.

```
/**
 ** peviewimageoptionalheader.c
 **/

#include <windows.h>
#include <stdio.h>
```



```

void piewOptionalHeaderDirectoryEntries(PIMAGE_DATA_DIRECTORY);
void viewOptionalHeaderSubsystem(WORD);

int main(int argc, char **argv)
{
    PIMAGE_DOS_HEADER      pImageDosHeader;
    PIMAGE_NT_HEADERS      pImageNtHeaders;
    PIMAGE_OPTIONAL_HEADER pImageOptionalHeader;
    PIMAGE_DATA_DIRECTORY  pImageDataDirectory;
    HANDLE                 hFile;
    HANDLE                 hMapObject;
    PUCHAR                  uFileMap;

    if (argc < 2)
        return (-1);

    if (!(hFile = CreateFile(argv[1], GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, 0)))
        return (-1);

    if (!(hMapObject = CreateFileMapping(hFile, NULL, PAGE_READONLY, 0, 0, NULL)))
        return (-1);

    if (!(uFileMap = MapViewOfFile(hMapObject, FILE_MAP_READ, 0, 0, 0)))
        return (-1);

    pImageDosHeader = (PIMAGE_DOS_HEADER) uFileMap ;
    if (pImageDosHeader->e_magic != IMAGE_DOS_SIGNATURE)
        return (-1);

    pImageNtHeaders = (PIMAGE_NT_HEADERS) ((PUCHAR) uFileMap + pImageDosHeader->e_lfanew);
    if (pImageNtHeaders->Signature != IMAGE_NT_SIGNATURE)
        return (-1);

    pImageOptionalHeader = (PIMAGE_OPTIONAL_HEADER) &(pImageNtHeaders->OptionalHeader);
    printf("Magic:                0x%04x", pImageOptionalHeader->Magic);
    ((pImageOptionalHeader->Magic == IMAGE_NT_OPTIONAL_HDR_MAGIC)
     ? printf(" (HDR32)\n")
     : printf(" (HDR64)\n"));
    printf("MajorLinkerVersion:        0x%02x\n", pImageOptionalHeader->MajorLinkerVersion);
    printf("MinorLinkerVersion:        0x%02x\n", pImageOptionalHeader->MinorLinkerVersion);
    printf("SizeOfCode:                0x%08x\n", pImageOptionalHeader->SizeOfCode);
    printf("SizeOfInitializedData:     0x%08x\n", pImageOptionalHeader->SizeOfInitializedData);
    printf("SizeOfUninitializedData:   0x%08x\n", pImageOptionalHeader->SizeOfUninitializedData);
    printf("AddressOfEntryPoint:       0x%08x\n", pImageOptionalHeader->AddressOfEntryPoint);
    printf("BaseOfCode:                0x%08x\n", pImageOptionalHeader->BaseOfCode);
    printf("BaseOfData:                0x%08x\n", pImageOptionalHeader->BaseOfData);
    printf("ImageBase:                 0x%08x\n", pImageOptionalHeader->ImageBase);
    printf("SectionAlignment:          0x%08x\n", pImageOptionalHeader->SectionAlignment);
    printf("FileAlignment:             0x%08x\n", pImageOptionalHeader->FileAlignment);
    printf("MajorOperatingSystemVersion: 0x%04x\n", pImageOptionalHeader->MajorOperatingSystemVersion);
    printf("MinorOperatingSystemVersion: 0x%04x\n", pImageOptionalHeader->MinorOperatingSystemVersion);
    printf("MajorImageVersion:         0x%04x\n", pImageOptionalHeader->MajorImageVersion);
    printf("MinorImageVersion:         0x%04x\n", pImageOptionalHeader->MinorImageVersion);
    printf("MajorSubsystemVersion:     0x%04x\n", pImageOptionalHeader->MajorSubsystemVersion);
    printf("MinorSubsystemVersion:     0x%04x\n", pImageOptionalHeader->MinorSubsystemVersion);
    printf("SizeOfImage:               0x%08x\n", pImageOptionalHeader->SizeOfImage);

```



```

        "EFI_ROM",
        "XBOX",
        "?",
        "WINDOWS_BOOT_APPLICATION",
    };

    printf(" (%s)\n", Subsystems[Subsystem]);
}

C:\>peviewimageoptionalheader.exe c:\WINDOWS\NOTEPAD.EXE
Magic:                0x010b (HDR32)
MajorLinkerVersion:   0x07
MinorLinkerVersion:   0x0a
SizeOfCode:           0x00007800
SizeOfInitializedData: 0x0000a600
SizeOfUninitializedData: 0x00000000
AddressOfEntryPoint:  0x0000739d
BaseOfCode:           0x00001000
BaseOfData:           0x00009000
ImageBase:            0x01000000
SectionAlignment:     0x00001000
FileAlignment:        0x00000200
MajorOperatingSystemVersion: 0x0005
MinorOperatingSystemVersion: 0x0001
MajorImageVersion:    0x0005
MinorImageVersion:    0x0001
MajorSubsystemVersion: 0x0004
MinorSubsystemVersion: 0x0000
SizeOfImage:          0x00014000
SizeOfHeaders:        0x00000400
Checksum:             0x00018700
Subsystem:            0x0002 (WINDOWS_GUI)
DllCharacteristics:   0x00008000
SizeOfStackReserve:   0x00040000
SizeOfStackCommit:    0x00011000
SizeOfHeapReserve:    0x00100000
SizeOfHeapCommit:     0x00001000
LoaderFlags:          0x00000000
NumberOfRvaAndSizes:  0x00000010

DIRECTORY ENTRIES   VirtualAddress   Size
IMPORT              0x00007604   0x000000c8
RESOURCE            0x0000b000   0x00008948
DEBUG               0x00001350   0x0000001c
LOAD_CONFIG         0x000018a8   0x00000040
BOUND_IMPORT        0x00000250   0x000000d0
IAT                 0x00001000   0x00000348

```

Les entêtes de sections (IMAGE_SECTION_HEADER)

Chaque entête de section définit une section sur le disque dur, c'est à dire la façon dont celle-ci doit être chargée par le loader.

```
#define IMAGE_SIZEOF_SHORT_NAME 8

typedef struct _IMAGE_SECTION_HEADER {
    UCHAR    Name[IMAGE_SIZEOF_SHORT_NAME];
    union {
        ULONG    PhysicalAddress;
        ULONG    VirtualSize;
    } Misc;
    ULONG    VirtualAddress;
    ULONG    SizeOfRawData;
    ULONG    PointerToRawData;
    ULONG    PointerToRelocations;
    ULONG    PointerToLinenumbers;
    USHORT   NumberOfRelocations;
    USHORT   NumberOfLinenumbers;
    ULONG    Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;

#define IMAGE_SIZEOF_SECTION_HEADER 40
```

IMAGE_SECTION_HEADER		40/0x28 octets
Champ	Offset	Taille
Name	0x000	BYTE(8)
VirtualSize	0x008	DWORD
VirtualAddress	0x00C	DWORD
SizeOfRawData	0x010	DWORD
PointerToRawData	0x014	DWORD
PointerToRelocations	0x018	DWORD
PointerToLineNumbers	0x01C	DWORD
NumberOfRelocations	0x020	WORD
NumberOfLineNumbers	0x022	WORD
Characteristics	0x024	DWORD

Name

Nom de la section.

VirtualSize

Taille de la section lors du chargement en mémoire

VirtualAddress

Adresse virtuelle relative de la section en mémoire. Ce chiffre doit être un multiple de la valeur contenue dans le champ SectionAlignment.

SizeOfRawData

Taille de la section (sur disque). Ce champ doit être aligné sur le FileAlignment

PointerToRawData

Offset du début de la section (sur disque). Ce champ doit être aligné sur le FileAlignment

PointerToRelocations

Adresse de la section dans la table de relocations

PointerToLineNumbers -

NumberOfRelocations

Nombre total des relocations

NumberOfLineNumbers -

Characteristics

Caractéristiques de la section. Elles définissent des attributs pour l'exécution. Par exemple si la section est exécutable, en lecture seule, si elle contient du code ou des données, etc.

```
#define IMAGE_SCN_CNT_CODE 0x00000020 // Section contains code.
#define IMAGE_SCN_CNT_INITIALIZED_DATA 0x00000040 // Section contains
initialized data.
#define IMAGE_SCN_CNT_UNINITIALIZED_DATA 0x00000080 // Section contains
uninitialized data.

#define IMAGE_SCN_LNK_OTHER 0x00000100 // Reserved.
#define IMAGE_SCN_LNK_INFO 0x00000200 // Section contains comments
or some other type of information.
// IMAGE_SCN_TYPE_OVER 0x00000400 // Reserved.
#define IMAGE_SCN_LNK_REMOVE 0x00000800 // Section contents will not
become part of image.
#define IMAGE_SCN_LNK_COMDAT 0x00001000 // Section contents comdat.
// 0x00002000 // Reserved.
// IMAGE_SCN_MEM_PROTECTED - Obsolete 0x00004000
#define IMAGE_SCN_NO_DEFER_SPEC_EXC 0x00004000 // Reset speculative
exceptions handling bits in the TLB entries for this section.
#define IMAGE_SCN_GPREL 0x00008000 // Section content can be
accessed relative to GP
#define IMAGE_SCN_MEM_FARDATA 0x00008000
// IMAGE_SCN_MEM_SYSHEAP - Obsolete 0x00010000
#define IMAGE_SCN_MEM_PURGEABLE 0x00020000
#define IMAGE_SCN_MEM_16BIT 0x00020000
#define IMAGE_SCN_MEM_LOCKED 0x00040000
#define IMAGE_SCN_MEM_PRELOAD 0x00080000

#define IMAGE_SCN_ALIGN_1BYTES 0x00100000 //
#define IMAGE_SCN_ALIGN_2BYTES 0x00200000 //
#define IMAGE_SCN_ALIGN_4BYTES 0x00300000 //
#define IMAGE_SCN_ALIGN_8BYTES 0x00400000 //
#define IMAGE_SCN_ALIGN_16BYTES 0x00500000 // Default alignment if no
others are specified.
#define IMAGE_SCN_ALIGN_32BYTES 0x00600000 //
#define IMAGE_SCN_ALIGN_64BYTES 0x00700000 //
#define IMAGE_SCN_ALIGN_128BYTES 0x00800000 //
#define IMAGE_SCN_ALIGN_256BYTES 0x00900000 //
#define IMAGE_SCN_ALIGN_512BYTES 0x00A00000 //
#define IMAGE_SCN_ALIGN_1024BYTES 0x00B00000 //
#define IMAGE_SCN_ALIGN_2048BYTES 0x00C00000 //
#define IMAGE_SCN_ALIGN_4096BYTES 0x00D00000 //
#define IMAGE_SCN_ALIGN_8192BYTES 0x00E00000 //
// Unused 0x00F00000
#define IMAGE_SCN_ALIGN_MASK 0x00F00000

#define IMAGE_SCN_LNK_NRELOC_OVFL 0x01000000 // Section contains extended
relocations.
#define IMAGE_SCN_MEM_DISCARDABLE 0x02000000 // Section can be discarded.
#define IMAGE_SCN_MEM_NOT_CACHED 0x04000000 // Section is not cachable.
#define IMAGE_SCN_MEM_NOT_PAGED 0x08000000 // Section is not pageable.
#define IMAGE_SCN_MEM_SHARED 0x10000000 // Section is shareable.
#define IMAGE_SCN_MEM_EXECUTE 0x20000000 // Section is executable.
#define IMAGE_SCN_MEM_READ 0x40000000 // Section is readable.
#define IMAGE_SCN_MEM_WRITE 0x80000000 // Section is writeable.
```

Le programme suivant liste les entêtes de sections d'un exécutable:

```

/**
** peviewimagesectionheaders.c
**/

#include <windows.h>
#include <stdio.h>

void viewImageSectionHeaderCharacteristics(DWORD);

int main(int argc, char **argv)
{
    PIMAGE_DOS_HEADER    pImageDosHeader;
    PIMAGE_NT_HEADERS    pImageNtHeaders;
    PIMAGE_SECTION_HEADER pImageSectionHeader;
    HANDLE               hFile;
    HANDLE               hMapObject;
    PUCHAR               uFileMap;
    DWORD                dwCount;

    if (argc < 2)
        return (-1);

    if (!(hFile = CreateFile(argv[1], GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, 0)))
        return (-1);

    if (!(hMapObject = CreateFileMapping(hFile, NULL, PAGE_READONLY, 0, 0, NULL)))
        return (-1);

    if (!(uFileMap = MapViewOfFile(hMapObject, FILE_MAP_READ, 0, 0, 0)))
        return (-1);

    pImageDosHeader = (PIMAGE_DOS_HEADER) uFileMap ;
    if (pImageDosHeader->e_magic != IMAGE_DOS_SIGNATURE)
        return (-1);

    pImageNtHeaders = (PIMAGE_NT_HEADERS) ((PUCHAR) uFileMap + pImageDosHeader->e_lfanew);
    if (pImageNtHeaders->Signature != IMAGE_NT_SIGNATURE)
        return (-1);

    pImageSectionHeader = (PIMAGE_SECTION_HEADER) ((DWORD) pImageNtHeaders + sizeof
    (IMAGE_NT_HEADERS));
    for (dwCount = 0; dwCount != pImageNtHeaders->FileHeader.NumberOfSections; dwCount++) {
        printf("Name:                %s\n", pImageSectionHeader->Name);
        printf("Misc:                %08X\n", pImageSectionHeader->Misc);
        printf("VirtualAddress:        %08X\n", pImageSectionHeader->VirtualAddress);
        printf("SizeOfRawData:         %08X\n", pImageSectionHeader->SizeOfRawData);
        printf("PointerToRawData:      %08X\n", pImageSectionHeader->PointerToRawData);
        printf("PointerToRawData:      %08X\n", pImageSectionHeader->PointerToRawData);
        printf("PointerToRelocations:  %08X\n", pImageSectionHeader->PointerToRelocations);
        printf("PointerToRelocations:  %08X\n", pImageSectionHeader->PointerToRelocations);
        printf("PointerToLinenumbers:  %08X\n", pImageSectionHeader->PointerToLinenumbers);
        printf("PointerToLinenumbers:  %08X\n", pImageSectionHeader->PointerToLinenumbers);
        printf("NumberOfRelocations:   %04X\n", pImageSectionHeader->NumberOfRelocations);
        printf("NumberOfRelocations:   %04X\n", pImageSectionHeader->NumberOfRelocations);
        printf("NumberOfLinenumbers:   %04X\n", pImageSectionHeader->NumberOfLinenumbers);
        printf("NumberOfLinenumbers:   %04X\n", pImageSectionHeader->NumberOfLinenumbers);
        printf("Characteristics:        %08X", pImageSectionHeader->Characteristics);
        viewImageSectionHeaderCharacteristics(pImageSectionHeader->Characteristics);
        printf("\n");
        pImageSectionHeader = (PIMAGE_SECTION_HEADER) ((DWORD) pImageSectionHeader +
        sizeof (IMAGE_SECTION_HEADER));
    }

    return (0);
}

void viewImageSectionHeaderCharacteristics(DWORD dwCharacteristics)
{

```

```

BYTE  szCharacteristics[100];

memset(szCharacteristics, 0, 100);
szCharacteristics[0] = '(';
if (dwCharacteristics & IMAGE_SCN_CNT_CODE)
    strcat(szCharacteristics, "CODE|");
if (dwCharacteristics & IMAGE_SCN_CNT_INITIALIZED_DATA)
    strcat(szCharacteristics, "INITIALIZED_DATA|");
if (dwCharacteristics & IMAGE_SCN_CNT_UNINITIALIZED_DATA)
    strcat(szCharacteristics, "UNINITIALIZED_DATA|");
if (dwCharacteristics & IMAGE_SCN_LNK_OTHER)
    strcat(szCharacteristics, "LNK_OTHER|");
if (dwCharacteristics & IMAGE_SCN_LNK_INFO)
    strcat(szCharacteristics, "LNK_INFO|");
if (dwCharacteristics & IMAGE_SCN_LNK_REMOVE)
    strcat(szCharacteristics, "LNK_REMOVE|");
if (dwCharacteristics & IMAGE_SCN_LNK_COMDAT)
    strcat(szCharacteristics, "LNK_COMDAT|");
if (dwCharacteristics & IMAGE_SCN_MEM_FARDATA)
    strcat(szCharacteristics, "MEM_FARDATA|");
if (dwCharacteristics & IMAGE_SCN_MEM_PURGEABLE)
    strcat(szCharacteristics, "MEM_PURGEABLE|");
if (dwCharacteristics & IMAGE_SCN_MEM_16BIT)
    strcat(szCharacteristics, "MEM_16BIT|");
if (dwCharacteristics & IMAGE_SCN_MEM_LOCKED)
    strcat(szCharacteristics, "MEM_LOCKED|");
if (dwCharacteristics & IMAGE_SCN_MEM_PRELOAD)
    strcat(szCharacteristics, "MEM_PRELOAD|");
if (dwCharacteristics & IMAGE_SCN_LNK_NRELOC_OVFL)
    strcat(szCharacteristics, "LNK_NRELOC_OVFL|");
if (dwCharacteristics & IMAGE_SCN_MEM_DISCARDABLE)
    strcat(szCharacteristics, "MEM_DISCARDABLE|");
if (dwCharacteristics & IMAGE_SCN_MEM_NOT_CACHED)
    strcat(szCharacteristics, "MEM_NOT_CACHED|");
if (dwCharacteristics & IMAGE_SCN_MEM_NOT_PAGED)
    strcat(szCharacteristics, "MEM_NOT_PAGED|");
if (dwCharacteristics & IMAGE_SCN_MEM_SHARED)
    strcat(szCharacteristics, "MEM_SHARED|");
if (dwCharacteristics & IMAGE_SCN_MEM_EXECUTE)
    strcat(szCharacteristics, "MEM_EXECUTE|");
if (dwCharacteristics & IMAGE_SCN_MEM_READ)
    strcat(szCharacteristics, "MEM_READ|");
if (dwCharacteristics & IMAGE_SCN_MEM_WRITE)
    strcat(szCharacteristics, "MEM_WRITE|");
szCharacteristics[strlen(szCharacteristics) - 1] = ')';
szCharacteristics[strlen(szCharacteristics)] = '\0';
printf(" %s\n", szCharacteristics);
}

```

```

C:\>peviewimagesectionheaders.exe c:\WINDOWS\notepad.exe
Name:                .text
Misc:                00007748
VirtualAddress:      00001000
SizeOfRawData:       00007800
PointerToRawData:    00000400
PointerToRelocations: 00000000
PointerToLinenumbers: 00000000
NumberOfRelocations: 0000
NumberOfLinenumbers: 0000
Characteristics:      60000020 (CODE|MEM_EXECUTE|MEM_READ)

Name:                .data
Misc:                00001BA8
VirtualAddress:      00009000
SizeOfRawData:       00000800
PointerToRawData:    00007C00
PointerToRelocations: 00000000
PointerToLinenumbers: 00000000
NumberOfRelocations: 0000
NumberOfLinenumbers: 0000
Characteristics:      C0000040 (INITIALIZED_DATA|MEM_READ|MEM_WRITE)

```

```
Name: .rsrc
Misc: 00008948
VirtualAddress: 0000B000
SizeOfRawData: 00008A00
PointerToRawData: 00008400
PointerToRelocations: 00000000
PointerToLinenumbers: 00000000
NumberOfRelocations: 0000
NumberOfLinenumbers: 0000
Characteristics: 40000040 (INITIALIZED_DATA|MEM_READ)
```


Analyse d'un programme PE

Dans cette partie, on se propose de détailler un à un les éléments d'un programme simple.

Code source

```
; donothing.asm

.386
.model flat, stdcall
option casemap :none

.data
    dd      012345678h

.code

start:
    ret

end      start
```

Analyse

Le code ci-dessus, une fois assemblé, génère un exécutable de 4096 octets. Une vue hexadécimale nous renseigne sur les données contenues à l'intérieur de celui-ci.

```
$ hexdump -C donothing.exe
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ.....|
00000010  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |.....@....|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 a8 00 00 00  |.....|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |.....!.L.!Th|
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.....|
00000080  5d 17 1d db 19 76 73 88  19 76 73 88 19 76 73 88  |]....vs..vs..vs.|
00000090  e5 56 61 88 18 76 73 88  52 69 63 68 19 76 73 88  |.Va..vs.Rich.vs.|
000000a0  00 00 00 00 00 00 00 00  50 45 00 00 4c 01 02 00  |.....PE..L...|
000000b0  fd c2 de 4a 00 00 00 00  00 00 00 00 e0 00 0f 01  |...J.....|
000000c0  0b 01 05 0c 00 02 00 00  00 02 00 00 00 00 00 00  |.....|
000000d0  00 10 00 00 00 10 00 00  00 20 00 00 00 00 40 00  |.....  ....@.|
000000e0  00 10 00 00 00 02 00 00  04 00 00 00 00 00 00 00  |.....|
000000f0  04 00 00 00 00 00 00 00  00 30 00 00 00 02 00 00  |.....0.....|
00000100  00 00 00 00 02 00 00 00  00 00 10 00 00 10 00 00  |.....|
00000110  00 00 10 00 00 10 00 00  00 00 00 00 10 00 00 00  |.....|
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
*
000001a0  2e 74 65 78 74 00 00 00  01 00 00 00 00 10 00 00  |.text.....|
000001b0  00 02 00 00 00 02 00 00  00 00 00 00 00 00 00 00  |.....|
000001c0  00 00 00 00 20 00 00 60  2e 64 61 74 61 00 00 00  |.... ..`.data...|
000001d0  04 00 00 00 00 20 00 00  00 02 00 00 00 04 00 00  |.....|
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 c0  |.....@....|
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
00000200  c3 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
```

```

00000210  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
00000400  78 56 34 12 00 00 00 00 00 00 00 00 00 00 00 00 |xV4.....|
00000410  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
00000600

```

La vue hexadécimale complète présente des suites continues de zéros. Si on obtient une suite d'octets aussi étendue pour un programme aussi simple, c'est parce que les processus d'assemblage et d'éditions des liens ont générés un exécutable suivant les règles usuelles des programmes Windows.

Il est tout à fait possible d'optimiser le PE produit, notamment en jouant sur certains champs pour éviter un padding aussi conséquent (la plupart des zéros contenu dans le listing original étant du remplissage, absolument inutile pour l'exécution du code).

Analysons les données en les reportant aux structures définies dans la spécification Portable Executable.

IMAGE_DOS_HEADER

```

00000000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 |MZ.....|
00000010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 |.....@.....|
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000030  00 00 00 00 00 00 00 00 00 00 00 00 a8 00 00 00 |.....|
00000040  0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 |.....!.L.!Th|
00000050  69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f |is program cann|
00000060  74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 |mode....$......|
00000080  5d 17 1d db 19 76 73 88 19 76 73 88 19 76 73 88 |]....vs..vs..vs.|
00000090  e5 56 61 88 18 76 73 88 52 69 63 68 19 76 73 88 |.Va..vs.Rich.vs.|
000000a0  00 00 00 00 00 00 00 00 50 45 00 00 4c 01 02 00 |.....PE..L...|

```

Champ	Offset	Taille	Valeur
e_magic	0x000	WORD	0x4d5a (ASCII 'MZ')
e_lfanew	0x03c	DWORD	0x000000a8

A l'offset 0x000000a8, on observe la présence de la signature PE. A la suite de celle-ci débute l'IMAGE_FILE_HEADER.

IMAGE_FILE_HEADER

```

000000a0  00 00 00 00 00 00 00 00 50 45 00 00 4c 01 02 00 |.....PE..L...|
000000b0  fd c2 de 4a 00 00 00 00 00 00 00 00 e0 00 0f 01 |...J.....|

```

20/0x14 octets

Champ	Offset	Taille	Valeur
Machine	0x000	WORD	0x014c
NumberOfSections	0x002	WORD	0x0002
TimeDateStamp	0x004	DWORD	0x4adec2fd
PointerToSymbolTable	0x008	DWORD	0x00000000
NumberOfSymbols	0x00C	DWORD	0x00000000
SizeOfOptionalHeader	0x010	WORD	0x000e
Characteristics	0x012	WORD	0x010f

A la lecture de ces données, on peut en déduire de ce fichier:

- Qu'il est à destination d'une architecture Intel 386.

- Qu'il comporte 2 sections.
- Qu'il s'agit d'un fichier dont les caractéristiques sont :
 - IMAGE_FILE_RELOCS_STRIPPED
 - IMAGE_FILE_LINE_NUMS_STRIPPED
 - IMAGE_FILE_EXECUTABLE_IMAGE
 - IMAGE_FILE_32BIT_MACHINE

IMAGE_OPTIONAL_HEADER

```

000000c0 0b 01 05 0c 00 02 00 00 00 02 00 00 00 00 00 00 | ..... |
000000d0 00 10 00 00 00 10 00 00 00 20 00 00 00 00 40 00 | .....@. |
000000e0 00 10 00 00 00 02 00 00 04 00 00 00 00 00 00 00 | ..... |
000000f0 04 00 00 00 00 00 00 00 00 30 00 00 00 02 00 00 | .....0..... |
00000100 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 | ..... |
00000110 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 | ..... |
00000120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ..... |

```

*

Champ	Offset	Taille	Valeur
Magic	0x00	WORD	0x010b
MajorLinkerVersion	0x02	BYTE	0x05
MinorLinkerVersion	0x03	BYTE	0x0c
SizeOfCode	0x04	DWORD	0x00000200
SizeOfInitializedData	0x08	DWORD	0x00000200
SizeOfUninitializedData	0x0C	DWORD	0x00000000
AddressOfEntryPoint	0x10	DWORD	0x00001000
BaseOfCode	0x14	DWORD	0x00001000
BaseOfData	0x18	DWORD	0x00002000
ImageBase	0x1C	DWORD	0x00400000
SectionAlignment	0x20	DWORD	0x00001000
FileAlignment	0x24	DWORD	0x00000200
MajorOperatingSystemVersion	0x28	WORD	0x0004
MinorOperatingSystemVersion	0x2A	WORD	0x0000
MajorImageVersion	0x2C	WORD	0x0000
MinorImageVersion	0x2E	WORD	0x0000
MajorSubsystemVersion	0x30	WORD	0x0004
MinorSubsystemVersion	0x32	WORD	0x0000
Reserved1	0x34	DWORD	0x00000000
SizeOfImage	0x38	DWORD	0x00003000
SizeOfHeaders	0x3C	DWORD	0x00000200
Checksum	0x40	DWORD	0x00000000
Subsystem	0x44	WORD	0x0002
DllCharacteristics	0x46	WORD	0x0000
SizeOfStackReserve	0x48	DWORD	0x00100000
SizeOfStackCommit	0x4C	DWORD	0x00001000
SizeOfHeapReserve	0x50	DWORD	0x00100000
SizeOfHeapCommit	0x54	DWORD	0x00001000
LoaderFlags	0x58	DWORD	0x00000000
NumberOfRvaAndSizes	0x5C	DWORD	0x00000010
DataDirectory[0-15].VirtualAddress	0x60	DWORD	0x00000000
DataDirectory[0-15].Size	0x64	DWORD	0x00000000

IMAGE_SECTION_HEADER

```
000001a0  2e 74 65 78 74 00 00 00 01 00 00 00 00 10 00 00 |.text.....|
000001b0  00 02 00 00 00 02 00 00 00 00 00 00 00 00 00 00 |.....|
000001c0  00 00 00 00 20 00 00 60 2e 64 61 74 61 00 00 00 |.... ``.data...|
```

SECTION 1 - « .text »

Champ	Offset	Taille	Valeur
Name	0x000	BYTE(8)	0x2e74657874 (ASCII «.text»)
VirtualSize	0x008	DWORD	0x00000001
VirtualAddress	0x00C	DWORD	0x00001000
SizeOfRawData	0x010	DWORD	0x00000200
PointerToRawData	0x014	DWORD	0x00000200
PointerToRelocations	0x018	DWORD	0x00000000
PointerToLineNumbers	0x01C	DWORD	0x00000000
NumberOfRelocations	0x020	WORD	0x0000
NumberOfLineNumbers	0x022	WORD	0x0000
Characteristics	0x024	DWORD	0x60000020

```
000001c0  00 00 00 00 20 00 00 60 2e 64 61 74 61 00 00 00 |.... ``.data...|
000001d0  04 00 00 00 00 20 00 00 00 02 00 00 00 04 00 00 |.....|
000001e0  00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 c0 |.....@...|
```

SECTION 2 - « .data »

Champ	Offset	Taille	Valeur
Name	0x000	BYTE(8)	0x2e64617461 (ASCII «.data »)
VirtualSize	0x008	DWORD	0x00000004
VirtualAddress	0x00C	DWORD	0x00002000
SizeOfRawData	0x010	DWORD	0x00000200
PointerToRawData	0x014	DWORD	0x00000400
PointerToRelocations	0x018	DWORD	0x00000000
PointerToLineNumbers	0x01C	DWORD	0x00000000
NumberOfRelocations	0x020	WORD	0x0000
NumberOfLineNumbers	0x022	WORD	0x0000
Characteristics	0x024	DWORD	0xc0000040

Les sections

Cette partie s'intéresse aux sections dans l'espace d'adressage du processus. Dans celui-ci, on trouvera:

- le module `donothing.exe`
- le module `kernel32.dll`
- le module `ntdll.dll`

Le module 'Kernel32.dll' constitue une implémentation de l'API Windows. C'est lui qui met à disposition la plupart des fonctions utiles pour la création de programmes simples, et c'est par lui que doivent transiter les programmes désireux de respecter les « règles » de la programmation Windows. Les procédures de 'Kernel32.dll' vont faire appel à des fonctions de 'ntdll.dll', comportant l'API dite « native » qui s'adresse au noyau.

Les fonctions de l'API Windows sont normalisées et documentées. A l'inverse, celles de l'API native sont parfois non documentées et susceptibles de changement.

Le module `donothing.exe` est organisé de la façon suivante:

- l'entête PE à l'adresse `0x00400000` (ImageBase)
- une section nommée « `.text` », sur disque à l'offset `0x200` (PointerToRawData) a été chargée en mémoire à l'adresse `0x00401000` (ImageBase + VirtualAddress)
- une section nommée « `.data` », sur disque à l'offset `0x400` (PointerToRawData) a été chargée en mémoire à l'adresse `0x00401000` (ImageBase + VirtualAddress)

Section de code

Dans cette section se trouve le code de l'application. Cette zone mémoire a les propriétés:

- `IMAGE_SCN_CNT_CODE`
- `IMAGE_SCN_MEM_EXECUTE`
- `IMAGE_SCN_MEM_READ`

```
00000200  c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000210  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
```

```
$ ndisasm -k 0,0x200 donothing.exe
00000000  skipping 0x200 bytes
00000200  c3                      ret
00000201  0000                    add [bx+si],al
*
```

Section de donnée

Dans cette section, on trouve des données de l'application. Cette zone mémoire a les propriétés:

- `IMAGE_SCN_CNT_INITIALIZED_DATA`
- `IMAGE_SCN_MEM_READ`
- `IMAGE_SCN_MEM_WRITE`

```

00000400  78 56 34 12 00 00 00 00 00 00 00 00 00 00 00 00 |xV4.....|
00000410  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|

```

Section des importations

Pour expliquer la notion d'importation et d'exportation des fonctions, nous allons avoir recours à un programme simple, affichant une boîte de dialogue (MessageBox) puis demandant sa propre terminaison (ExitProcess).

```

; msgbox.asm

.386
.model flat, stdcall
option casemap:none

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc

includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data
    szWndTitle    db    "[msgbox]",0
    szWndText     db    "msgbox",0

.code

start:

    push    MB_OK
    push    offset szWndTitle
    push    offset szWndText
    push    0
    call    MessageBoxA

    push    0
    call    ExitProcess

```

Le fait que ce programme appelle une fonction de l'API Windows a impliqué, pour le linkeur, la création d'une section supplémentaire et le renseignement d'une entrée dans le tableau **DataDirectory**.

Champ	Offset	Taille	Valeur
DataDirectory[0].VirtualAddress	0x60	DWORD	0x00000000
DataDirectory[0].Size	0x64	DWORD	0x00000000
DataDirectory[0].VirtualAddress	0x68	DWORD	0x00002010
DataDirectory[0].Size	0x6c	DWORD	0x0000003c

```

000001a0  00 00 00 00 00 00 00 00 00 2e 74 65 78 74 00 00 00 |.....text...|
000001b0  26 00 00 00 00 10 00 00 00 02 00 00 00 04 00 00 00 |&.....|
000001c0  00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60 00 |.....`|

```

SECTION 1 - « .text »

Champ	Offset	Taille	Valeur
Name	0x000	BYTE(8)	0x2e74657874 (ASCII «.text»)
VirtualSize	0x008	DWORD	0x00000026
VirtualAddress	0x00C	DWORD	0x00001000
SizeOfRawData	0x010	DWORD	0x00000200
PointerToRawData	0x014	DWORD	0x00000400
PointerToRelocations	0x018	DWORD	0x00000000
PointerToLineNumbers	0x01C	DWORD	0x00000000
NumberOfRelocations	0x020	WORD	0x0000
NumberOfLineNumbers	0x022	WORD	0x0000
Characteristics	0x024	DWORD	0x60000020

```
$ ndisasm -b 32 -k 0,0x400 msgbox.exe
```

```
00000000  skipping 0x400 bytes
00000400  6A00          push byte +0x0
00000402  6800304000    push dword 0x403000
00000407  6809304000    push dword 0x403009
0000040C  6A00          push byte +0x0
0000040E  E807000000    call 0x41a
00000413  6A00          push byte +0x0
00000415  E806000000    call 0x420
0000041A  FF2508204000  jmp near [0x402008]
00000420  FF2500204000  jmp near [0x402000]
```

```
000001d0  2e 72 64 61 74 61 00 00  92 00 00 00 00 20 00 00  |.rdata..... ..|
000001e0  00 02 00 00 00 06 00 00  00 00 00 00 00 00 00 00  |.....|
000001f0  00 00 00 00 40 00 00 40  2e 64 61 74 61 00 00 00  |....@..@.data...|
00000200  10 00 00 00 00 30 00 00  00 02 00 00 00 08 00 00  |.....0.....|
00000210  00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 c0  |.....@...|
00000220  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
```

SECTION 2 - « .rdata »

Champ	Offset	Taille	Valeur
Name	0x000	BYTE(8)	0x2e7264610000 (ASCII «.text»)
VirtualSize	0x008	DWORD	0x00000092
VirtualAddress	0x00C	DWORD	0x00002000
SizeOfRawData	0x010	DWORD	0x00000200
PointerToRawData	0x014	DWORD	0x00000600
PointerToRelocations	0x018	DWORD	0x00000000
PointerToLineNumbers	0x01C	DWORD	0x00000000
NumberOfRelocations	0x020	WORD	0x0000
NumberOfLineNumbers	0x022	WORD	0x0000
Characteristics	0x024	DWORD	0x40000040

```
$ hexdump -C -s 0x600 msgbox.exe
```

```
00000600  76 20 00 00 00 00 00 00  5c 20 00 00 00 00 00 00  |v .....\ .....|
00000610  54 20 00 00 00 00 00 00  00 00 00 00 6a 20 00 00  |T .....j ..|
00000620  08 20 00 00 4c 20 00 00  00 00 00 00 00 00 00 00  |. ..L .....|
00000630  84 20 00 00 00 20 00 00  00 00 00 00 00 00 00 00  |. . ... ..|
00000640  00 00 00 00 00 00 00 00  00 00 00 00 76 20 00 00  |.....v ..|
```

```

00000650  00 00 00 00 5c 20 00 00 00 00 00 00 b1 01 4d 65 |....\ .....Me|
00000660  73 73 61 67 65 42 6f 78 41 00 75 73 65 72 33 32 |ssageBoxA.user32|
00000670  2e 64 6c 6c 00 00 9b 00 45 78 69 74 50 72 6f 63 |.dll....ExitProc|
00000680  65 73 73 00 6b 65 72 6e 65 6c 33 32 2e 64 6c 6c |ess.kernel32.dll|
00000690  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|

```

*

```

000001f0  00 00 00 00 40 00 00 40 2e 64 61 74 61 00 00 00 |....@..@.data...|
00000200  10 00 00 00 00 30 00 00 00 02 00 00 00 08 00 00 |.....0.....|
00000210  00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 c0 |.....@...|

```

SECTION 3 - « .data »

Champ	Offset	Taille	Valeur
Name	0x000	BYTE(8)	0x2e64617461 (ASCII «.data »)
VirtualSize	0x008	DWORD	0x00000010
VirtualAddress	0x00C	DWORD	0x00003000
SizeOfRawData	0x010	DWORD	0x00000200
PointerToRawData	0x014	DWORD	0x00000800
PointerToRelocations	0x018	DWORD	0x00000000
PointerToLineNumbers	0x01C	DWORD	0x00000000
NumberOfRelocations	0x020	WORD	0x0000
NumberOfLineNumbers	0x022	WORD	0x0000
Characteristics	0x024	DWORD	0xc0000040

```

$ hexdump -C -s 0x800 msgbox.exe
00000800  5b 6d 73 67 62 6f 78 5d 00 6d 73 67 62 6f 78 00 |[msgbox].msgbox.|
00000810  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|

```

*

Un fichier exécutable sur un support de stockage – c’est à dire inactif par rapport au système – ne connaît pas l’adresse des APIs que le programme utilise. On peut bien sûr inscrire dans la section de code des appels directs aux APIs Windows (Absolute Call) mais il s’agit d’une méthode peu sûre. Microsoft ne garantit jamais qu’une fonction conserve son adresse mémoire au travers des Service Pack et autres versions du système d’exploitation. C’est même un cas de plus en plus rare, avec l’apparition de mécanismes de sécurité visant à faire émerger une répartition « au hasard » de l’espace mémoire utilisateur.

La table des importations sert à lier l’adresse d’une API Windows à son nom. Dans le fichier sur disque, cette table ne contient uniquement que des noms de fonction. C’est à l’exécution du programme, lorsque le loader charge le fichier en mémoire, qu’il inspecte la table et renseigne les structures de données correspondantes.

Ce mécanisme est étroitement lié à celui des « Sous-systèmes d’environnement » (Sub System Environment, SSE) et de la mise à disposition de fonctions par les Dynamic Link Library (DLL). Un SSE définit des primitives, des structures de données ; un ensemble de fonctions que les programmes peuvent utiliser. La plupart des applications sous Windows font appel au sous-système WINDOWS qui “donne” (exporte) ses fonctions par l’intermédiaire de DLL, comme Kernel32.dll. Par la suite, les fonctions d’un SSE vont faire appel au SubSystem “native”, dernier rempart avant le kernelland.

Lorsqu’un programme appelle une fonction, il s’agit d’un appel indirect, qui passe d’abord par la table des importations, comme on peut le constater lors du désassemblage de la section de code :


```

00000400  6A00                push byte +0x0
00000402  6800304000          push dword 0x403000
00000407  6809304000          push dword 0x403009
0000040C  6A00                push byte +0x0
0000040E  E807000000          call 0x41a
00000413  6A00                push byte +0x0
00000415  E806000000          call 0x420
0000041A  FF2508204000        jmp near [0x402008]
00000420  FF2500204000        jmp near [0x402000]

```

Les instructions « call 0x41a » et « call 0x420 » appellent non pas une adresse dans un module externe (MessageBox et ExitProcess) mais font référence à une autre instruction: des JMPs sur les adresses 0x402000 et 0x402008.

Par recoupement, on peut observer que ces adresses font partie de la section « .rdata », où se trouve des entrées de type IMAGE_IMPORT_DESCRIPTOR.

```

typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD Characteristics;
        DWORD OriginalFirstThunk;
    };
    DWORD TimeDateStamp;
    DWORD ForwarderChain;
    DWORD Name;
    DWORD FirstThunk;
} IMAGE_IMPORT_DESCRIPTOR, *PIMAGE_IMPORT_DESCRIPTOR;

```

IMAGE_IMPORT_DESCRIPTOR

Champ	Offset	Taille
OriginalFirstThunk	0x000	DWORD
TimeDateStamp	0x004	DWORD
ForwarderChain	0x008	DWORD
Name	0x00C	DWORD
FirstThunk	0x010	DWORD

OriginalFirstThunk

Adresse virtuelle relative d'un tableau IMAGE_THUNK_DATA

TimeDateStamp

Date de création

ForwarderChain -

Name

Adresse virtuelle d'une chaîne de caractères représentant le nom d'une DLL, suivi d'un zéro terminal

FirstThunk

Adresse virtuelle relative d'un tableau IMAGE_THUNK_DATA

Il existe autant d'IMAGE_IMPORT_DESCRIPTOR qu'il y a de DLL impliquées dans le programme, suivi d'un IMAGE_IMPORT_DESCRIPTOR dont tous les champs sont nuls.

Un IMAGE_THUNK_DATA est en fait un simple DWORD

```
typedef struct _IMAGE_THUNK_DATA32 {
    union {
        DWORD ForwarderString;
        DWORD Function;
        DWORD Ordinal;
        DWORD AddressOfData;
    } u1;
} IMAGE_THUNK_DATA32, *PIMAGE_THUNK_DATA32;
```

Dans la plupart des cas, ce champ va pointer sur une structure de type **IMAGE_IMPORT_BY_NAME**

```
typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD Hint;
    BYTE Name[1];
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

```
00000610  54 20 00 00 00 00 00 00 00 00 00 00 6a 20 00 00 |T .....j ..|
00000620  08 20 00 00 4c 20 00 00 00 00 00 00 00 00 00 00 |. ..L .....|
00000630  84 20 00 00 00 20 00 00 00 00 00 00 00 00 00 00 |. ... .....|
00000640  00 00 00 00 00 00 00 00 00 00 00 00 76 20 00 00 |.....v ..|
00000650  00 00 00 00 5c 20 00 00 00 00 00 00 b1 01 4d 65 |....\ .....Me|
00000660  73 73 61 67 65 42 6f 78 41 00 75 73 65 72 33 32 |ssageBoxA.user32|
00000670  2e 64 6c 6c 00 00 9b 00 45 78 69 74 50 72 6f 63 |.dll....ExitProc|
00000680  65 73 73 00 6b 65 72 6e 65 6c 33 32 2e 64 6c 6c |ess.kernel32.dll|
00000690  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
```

IMAGE_IMPORT_DESCRIPTOR .1

Champ	Offset	Taille	Valeur
OriginalFirstThunk	0x000	DWORD	0x00002054
TimeDateStamp	0x004	DWORD	0x00000000
ForwarderChain	0x008	DWORD	0x00000000
Name	0x00C	DWORD	0x0000206a
FirstThunk	0x010	DWORD	0x00002008

```
00000610  54 20 00 00 00 00 00 00 00 00 00 00 6a 20 00 00 |T .....j ..|
00000620  08 20 00 00 4c 20 00 00 00 00 00 00 00 00 00 00 |. ..L .....|
00000630  84 20 00 00 00 20 00 00 00 00 00 00 00 00 00 00 |. ... .....|
00000640  00 00 00 00 00 00 00 00 00 00 00 00 76 20 00 00 |.....v ..|
00000650  00 00 00 00 5c 20 00 00 00 00 00 00 b1 01 4d 65 |....\ .....Me|
00000660  73 73 61 67 65 42 6f 78 41 00 75 73 65 72 33 32 |ssageBoxA.user32|
00000670  2e 64 6c 6c 00 00 9b 00 45 78 69 74 50 72 6f 63 |.dll....ExitProc|
00000680  65 73 73 00 6b 65 72 6e 65 6c 33 32 2e 64 6c 6c |ess.kernel32.dll|
00000690  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
```

IMAGE_IMPORT_DESCRIPTOR .2

Champ	Offset	Taille	Valeur
OriginalFirstThunk	0x000	DWORD	0x0000204c
TimeDateStamp	0x004	DWORD	0x00000000
ForwarderChain	0x008	DWORD	0x00000000
Name	0x00C	DWORD	0x00002084
FirstThunk	0x010	DWORD	0x00002000

IMAGE_IMPORT_DESCRIPTOR .3

Champ	Offset	Taille	Valeur
OriginalFirstThunk	0x000	DWORD	0x00000000

Champ	Offset	Taille	Valeur
TimeStamp	0x004	DWORD	0x00000000
ForwarderChain	0x008	DWORD	0x00000000
Name	0x00C	DWORD	0x00000000
FirstThunk	0x010	DWORD	0x00000000

Le programme liste les APIs importés par un exécutable:

```

**
** peviewimports.c
**/

#include <windows.h>
#include <stdio.h>

DWORD RvaToOffset(PIMAGE_NT_HEADERS pImageNtHeaders, DWORD dwRva);

int main(int argc, char **argv)
{
    PIMAGE_DOS_HEADER pImageDosHeader;
    PIMAGE_NT_HEADERS pImageNtHeaders;
    PIMAGE_IMPORT_DESCRIPTOR pImageImportDescriptor;
    PIMAGE_IMPORT_BY_NAME pImageImportByName;
    DWORD dwCount;
    DWORD dwCount2;
    DWORD *Thunks;
    DWORD dwFileOffset;
    HANDLE hFile;
    HANDLE hMapObject;
    PCHAR uFileMap;

    if (argc < 2)
        return (-1);

    if (!(hFile = CreateFile(argv[1], GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, 0)))
        return (-1);

    if (!(hMapObject = CreateFileMapping(hFile, NULL, PAGE_READONLY, 0, 0, NULL)))
        return (-1);

    if (!(uFileMap = MapViewOfFile(hMapObject, FILE_MAP_READ, 0, 0, 0)))
        return (-1);

    pImageDosHeader = (PIMAGE_DOS_HEADER) uFileMap ;
    if (pImageDosHeader->e_magic != IMAGE_DOS_SIGNATURE)
        return (-1);

    pImageNtHeaders = (PIMAGE_NT_HEADERS) ((PCHAR) uFileMap + pImageDosHeader->e_lfanew);
    if (pImageNtHeaders->Signature != IMAGE_NT_SIGNATURE)
        return (-1);

    dwFileOffset = RvaToOffset(pImageNtHeaders, pImageNtHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress);
    pImageImportDescriptor = (PIMAGE_IMPORT_DESCRIPTOR) ((PCHAR) uFileMap + dwFileOffset);
    dwCount = 0;
    while (pImageImportDescriptor[dwCount].FirstThunk) {
        printf("Module Name: %s\n", ((PCHAR) uFileMap + RvaToOffset(pImageNtHeaders, pImageImportDescriptor[dwCount].Name)));
        Thunks = (DWORD *) ((PCHAR) uFileMap + RvaToOffset(pImageNtHeaders, pImageImportDescriptor[dwCount].OriginalFirstThunk));
        dwCount2 = 0;
        while (Thunks[dwCount2]) {

```

```

        pImageImportByName = (PIMAGE_IMPORT_BY_NAME) ((PUCHAR) uFileMap +
RvaToOffset(pImageNtHeaders, Thunks[dwCount2]));
        printf("Name: %s\n", pImageImportByName->Name);
        dwCount2++;
    }
    dwCount++;
}

return (0);
}

DWORD RvaToOffset(PIMAGE_NT_HEADERS pImageNtHeaders, DWORD dwRva)
{
    PIMAGE_SECTION_HEADER pImageSectionHeader;
    DWORD dwCount;
    DWORD dwFileOffset;

    pImageSectionHeader = IMAGE_FIRST_SECTION(pImageNtHeaders);
    dwFileOffset = dwRva;
    for (dwCount = 0; dwCount < pImageNtHeaders->FileHeader.NumberOfSections; dwCount++) {
        if (dwRva >= pImageSectionHeader[dwCount].VirtualAddress &&
            dwRva < (pImageSectionHeader[dwCount].VirtualAddress +
pImageSectionHeader[dwCount].SizeOfRawData)) {
            dwFileOffset -= pImageSectionHeader[dwCount].VirtualAddress;
            dwFileOffset += pImageSectionHeader[dwCount].PointerToRawData;
            return (dwFileOffset);
        }
    }
    return (0);
}

```

Annexe I. Retrouver l'adresses des APIs

Une problématique que l'on retrouve souvent lors de la manipulation du format PE est lié à l'espace d'adressage du processus. Particulièrement, du fait qu'un programme n'ait pas connaissance complète des adresses mémoires liés à son sous-système d'environnement.

Sous Windows, le mécanisme d'une table d'importations permet de demander au loader de récupérer ces adresses, à partir d'un nom de DLL et de celui d'une fonction. Les adresses étant calculés via des mécanismes de gestion interne, il est nécessaire d'utiliser différentes méthodes pour obtenir des informations fiables à leur sujet.

Cet annexe s'intéresse à l'organisation de la mémoire utilisateur lors du chargement d'un programme au format Portable Executable.

Fondamentaux: processus et threads

- ✓ Un **processus** est l'image en mémoire d'un fichier exécutable, associé à des mécanismes de gestion du système d'exploitation
- ✓ Un processus possède :
 - Une ou plusieurs **unités d'exécution** appelée(s) **thread**.
 - Un programme exécutable (code et données).
 - Un espace d'adressage **virtuel privé**.
- ✓ Un processus a au minimum un thread.
- ✓ Un processus peut être vu comme un espace de mémoire linéaire, d'une taille de **4GB** (2^{32}), allant de l'adresse 0x00000000 à 0xFFFFFFFF.
- ✓ Cet espace mémoire est privé, en théorie il est inaccessible par les autres processus.
- ✓ Cet espace se partage en 2GB pour le système et 2GB pour l'utilisateur.
- ✓ Le noyau s'occupe entièrement de faire l'opération de translation entre adressage virtuel et adressage réel.

- ✓ Un thread comprend :
 - Un compteur d'instructions
 - Un environnement, une pile en mode utilisateur, une pile en mode noyau.
 - Un ensemble de valeurs pour les registres (état du processeur)
 - Une zone privée de données
- ✓ Tous ces éléments sont rassemblés sous le nom de **contexte de thread**.
- ✓ Un thread ne peut appartenir qu'à un seul processus
- ✓ L'espace d'adressage est commun à tous les threads d'un même processus

Importations et exportations: le module Kernel32

Le problème peut se formuler de la façon suivante : par quel moyen pourrait-on appeler une API dont on ne connaît pas au préalable l'adresse ? La réponse la plus évidente consiste en l'utilisation des fonctions LoadLibrary et GetProcAddress.

```

/*
 * gewifa.c
 * Get win32 function address
 */

#include <windows.h>
#include <stdio.h>

int      main(int argc, char **argv)
{
    HMODULE    hLib;
    DWORD      FuncAddress;

    printf("Gewifa - Get win32 function address\n");
    if (argc < 3) {
        printf("%s <DLL Name> <Function Name>\n", argv[0]);
        return (-1);
    }
    if (!(hLib = LoadLibrary(argv[1]))) {
        printf("Error from LoadLibrary!\n");
        return (-1);
    }
    if (!(FuncAddress = (DWORD) GetProcAddress(hLib, argv[2]))) {
        printf("Error from GetProcAddress\n");
        return (-1);
    }
    printf("%s - %s [0x%08x]\n", argv[1], argv[2], (unsigned int)
FuncAddress);
    return (0);
}

$ gewifa kernel32.dll ExitProcess
Gewifa - Get win32 function address
kernel32.dll - ExitProcess [0x7c81cafa]

```

Cette solution est envisageable mais peu fiable. En effet, les adresses des fonctions de l'API Windows peuvent différer entre les versions du système d'exploitations, les services pack, et même parfois la langue utilisée.

Une solution fiable se situe au niveau des procédures d'importation et d'exportation des fonctions.

Sous Windows, il existe une DLL obligatoire pour tous les programmes, puisque c'est elle qui exporte la plupart des API qui leur sont nécessaires. Il s'agit du module « Kernel32.dll ». On trouvera toujours cette DLL présente dans l'espace mémoire du processus qui s'exécute.

Lors de l'exécution d'un fichier PE, le loader va charger en mémoire toutes les DLL que le programme utilise. Par exemple, si un programme utilise la fonction ExitProcess, contenue dans KERNEL32.DLL, tout le module sera mappée en mémoire.

Dans sa structure interne, une DLL est un fichier au format PE. On va donc pouvoir parser la table d'export du module KERNEL32 pour retrouver l'adresse de nos API. Mais ici un autre problème se pose, puisque ce qu'on va trouver dans l'export table, ce sont des adresses mémoires relatives à celle du module, information que nous ne connaissons pas.

Il va donc falloir d'abord retrouver l'adresse de base de KERNEL32, puis seulement ensuite trouver les renseignements utiles dans la table d'exportations des fonctions.

L'adresse de base de Kernel32.dll

Il existe plusieurs méthodes pour obtenir cette information. Ce document en exposera deux:

- Utilisation du pointeur de pile (ESP)
- Utilisation de la structure *Process Environment Block*

Stack Pointer

Le point d'entrée d'un programme étant appelée par une fonction de Kernel32.dll, on peut tirer profit de cette particularité. C'est à dire que l'applicatif Kernel32 effectue un CALL à l'adresse mémoire où a été mappe l'exécutable.

L'instruction CALL a la propriété d'empiler l'EIP courant avant de le faire pointer vers le code appelé. C'est grâce à cette propriété que nous allons pouvoir « retourner » dans le mapping mémoire de Kernel32, pour ensuite « revenir en arrière » dans la mémoire en recherchant les marqueurs MZ et PE, signatures d'un fichier exécutable.

```
; getkernelbase-esp.asm

.386
.model flat, stdcall
option casemap:none

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc

includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data

    WndTextOut1 db "Kernel32 base address: 0x"
    WndTextOut2 db 8 dup (66), 13, 10
    WndTextFmt db "%x"

.code

start:

    mov esi,[esp]
    and esi,0FFFF0000h

l1:
    sub esi,1000h
    cmp word ptr [esi],"ZM"
    jne l1

    mov eax,[esi+3Ch]
    cmp word ptr [esi+eax],"EP"
    jne exit

    push esi
    push offset WndTextFmt
    push offset WndTextOut2
    call wsprintfA

    push STD_OUTPUT_HANDLE
    call GetStdHandle

    push NULL
    push NULL
    push SIZEOF WndTextOut1 + SIZEOF WndTextOut2
```

```

        push    offset WndTextOut1
        push    eax
        call    WriteFile

exit:
        push    0
        call    ExitProcess

end     start

```

PEB: Process Environment Block

Pour chaque processus qui s'exécute sur une machine, le système d'exploitation va allouer une structure de donnée contenant des informations importantes sur le processus créé. Le PEB (Process Environment Block) nous renseigne par exemple sur l'état de la pile ou les handles ouverts.

Le PEB contient aussi trois listes chaînées. Ces listes concernent les modules mappés dans l'espace mémoire du processus. L'une d'entre elle décrit l'ordre d'initialisation de ces modules. On pourra remarquer que Kernel32 est toujours initialisé en seconde position. On va donc pouvoir parcourir la liste et en récupérer la deuxième entrée.

Sous WinDbg, cela donne:

0:001> lm start end module name 01000000 01014000 notepad (deferred) ... 7c800000 7c8f6000 kernel32 (deferred) 7c900000 7c9af000 ntdll (pdb symbols)
0:001> dd @fs:0x30 0038:00000030 7ffd4000 00000000 00000000 00000000
0:001> dd 0x7ffd4000 + 0xC 7ffd400c 001a1e90 00020000 00000000 000a0000
0:001> dd 0x1a1e90 + 0x1c 001a1eac 001a1f28 001a28b8 00000000 0006000b
0:001> dd 0x001a1f28 001a1f28 001a1fd0 001a1eac 7c900000 7c912c28
0:001> dd 0x001a1fd0 + 0x8 001a1fd8 7c800000 7c80b63e 000f6000 00420040

Séquence que l'on pourrait traduire ainsi:

```

0:001> lm
; getkernelbase-peb.asm

.386
.model flat, stdcall
option casemap:none
assume fs:nothing

```



```

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc

includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data

    WndTextOut1    db    "Kernel32 base address: 0x"
    WndTextOut2    db    8 dup (66), 13, 10
    WndTextFmt      db    "%x"

.code

start:

    xor     esi,esi
    mov     esi,fs:[030h]          ; pointer to PEB
    mov     esi,[esi + 0Ch]        ; PEB->Ldr
    mov     esi,[esi + 01Ch]       ; PEB-
>Ldr.InLoadOrderModuleList.Flink
    mov     esi,[esi]              ; second entry
    mov     esi,[esi + 08h]        ; kernel base address

    push    esi
    push    offset WndTextFmt
    push    offset WndTextOut2
    call    wsprintfA

    push    STD_OUTPUT_HANDLE
    call    GetStdHandle

    push    NULL
    push    NULL
    push    SIZEOF WndTextOut1 + SIZEOF WndTextOut2
    push    offset WndTextOut1
    push    eax
    call    WriteFile

exit:
    push    0
    call    ExitProcess

end        start

```

Conclusion & ressources

L'étude du format natif des exécutables sur un système d'exploitation donné revient à s'initier - en partie du moins - aux composants liés au chargement et à l'exécution des applications.

Pour poursuivre, le lecteur est invité à consulter les ressources ci-après.

Ressources

Microsoft:

Microsoft Portable Executable and Common Object File Format Specification

<http://www.microsoft.com/whdc/system/platform/firmware/pecoff.msp>

An In-Depth Look into the Win32 Portable Executable File Format,

Matt Pietrek, [http://msdn.microsoft.com/fr-fr/magazine/cc301805\(en-us\).aspx](http://msdn.microsoft.com/fr-fr/magazine/cc301805(en-us).aspx)

An In-Depth Look into the Win32 Portable Executable File Format, Part 2

Matt Pietrek, [http://msdn.microsoft.com/fr-fr/magazine/cc301808\(en-us\).aspx](http://msdn.microsoft.com/fr-fr/magazine/cc301808(en-us).aspx)

MASM32 (Microsoft assembler)

<http://www.masm32.com>

OllyDbg

<http://www.ollydbg.de>

Debugging tools for Windows

<http://www.microsoft.com/whdc/DevTools/Debugging/default.msp>

LordPE

<http://esl.epitech.net/~arnaud/lld/s/pe-tools/lordpe/>