Présentation du format PE

aaSSfxxx

27 janvier 2011

Table des matières

1 Introduction

1.1 Objectif de ce document

Le format PE est le format de fichier utilisé par tous les fichiers exécutables sous Windows, qu'il est important de connaître lorsqu'on veut créer un compilateur, un linker ou encore un virus. Le but de ce document est donc de présenter le format PE afin de le démystifier.

Nous aborderons tout d'abord l'architecture de l'en-tête PE, puis nous étudierons les différentes sections d'un exécutable. (à compléter)

1.2 Prérequis

Afin de suivre ce tutoriel, il est recommandé d'avoir des connaissances en C/C++ voire assembleur, ainsi qu'en numérotation hexadécimale. Il faut notamment avoir des notions sur la manipulations des fichiers (notamment connaître le fonctionnement des fonctions fopen, fseek et fread), sur les structures (afin de faciliter la lecture de données contenues dans l'exécutable) ainsi que sur les chaînes de caractères.

Je noterai les nombres hexadécimaux selon la notation 0x puis le nombre, par exemple 0x5F ou 0xAA. Les nombres en système décimal seront écrits "normalement", par exemple : 10 , 20 ou 995.

2 L'en-tête PE

2.1 L'en-tête MS-DOS

Tout exécutable Windows contient d'abord un en-tête DOS : en effet, un exécutable Windows est tout d'abord un exécutable MS-DOS (Windows ayant été pendant très longtemps été une surcouche graphique à MS-DOS, le format de fichier PE a conservé cette spécificité). Nous allons donc étudier en détail cet en-tête.

Cet en-tête peut être représenté par cette structure (que l'on retrouve dans le fichier d'include winnt.h) :

```
typedef struct _IMAGE_DOS_HEADER {
  WORD   e_magic;       /* 00: MZ Header signature */
  WORD   e_cblp;        /* 02: Bytes on last page of file */
  WORD   e_cp;          /* 04: Pages in file */
  WORD   e_crlc;        /* 06: Relocations */
  WORD   e_cparhdr;     /* 08: Size of header in paragraphs */
  WORD   e_minalloc;    /* 0a: Minimum extra paragraphs needed */
  WORD   e_maxalloc;    /* 0c: Maximum extra paragraphs needed */
  WORD   e_ss;          /* 0e: Initial (relative) SS value */
  WORD   e_sp;          /* 10: Initial SP value */
  WORD   e_csum;        /* 12: Checksum */
  WORD   e_ip;          /* 14: Initial IP value */
  WORD   e_cs;          /* 16: Initial (relative) CS value */
  WORD   e_lfarlc;      /* 18: File address of relocation table */
  WORD   e_ovno;        /* 1a: Overlay number */
  WORD   e_res[4];      /* 1c: Reserved words */
  WORD   e_oemid;       /* 24: OEM identifier (for e_oeminfo) */
  WORD   e_oeminfo;     /* 26: OEM information; e_oemid specific */
  WORD   e_res2[10];    /* 28: Reserved words */
  DWORD e_lfanew;       /* 3c: Offset to extended header */
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

La structure IMAGE_DOS_HEADER

Cependant le but de ce document n'est pas d'étudier le format MS-DOS, ainsi nous allons juste nous intéresser au champ e_lfanew à l'offset 0x3C du fichier, qui contient l'offset du début de l'en-tête PE.

Ce champ nous permet donc de savoir où se trouve notre en-tête PE, étant donné qu'après l'en-tête DOS, nous avons le code du programme DOS (bien souvent qui affiche un message qui ressemble à "This program can't run in DOS mode" pour une application Win32). Voici la structure de notre header de l'exécutable :

DOS header	code du programme DOS (DOS Stub)	PE Header

Le format PE est donc le coeur de notre exécutable, puisque c'est lui qui contient toutes les informations nécessaires pour l'exécution de cet exécutable. Nous allons donc voir en détail notre PE Header.

2.2 Le PE Header

2.2.1 Le File Header

Le PE Header démarre donc à l'offset indiqué par le champ e_lfanew. Pour simplifier la lecture de ce tutoriel, je noterai pe_base la valeur contenue dans le champ e_lfanew. Regardons de plus près le contenu de notre structure :

```
typedef struct _IMAGE_NT_HEADERS {
  DWORD Signature; /* "PE"\0\0 */    /* pe_base + 0x00 */
  IMAGE_FILE_HEADER FileHeader;    /* pe_base + 0x04 */
  IMAGE_OPTIONAL_HEADER32 OptionalHeader;    /* pe_base + 0x18 */
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

Comme nous pouvons le constater, notre PE header est consitué de plusieurs sections : une signature qui contiendra "PE" quoi qu'il arrive (plus précisément cette suite d'octets : 0x45 0x50 0x00 0x00). Ensuite vient un IMAGE_FILE_HEADER qui est représenté par cette structure :

```
 typedef struct _IMAGE_FILE_HEADER {
   WORD  Machine;    /* pe_base + 0x04 */
   WORD  NumberOfSections;    /* pe_base + 0x06 */
   DWORD TimeDateStamp;    /* pe_base + 0x08 */
   DWORD PointerToSymbolTable;    /* pe_base + 0x0c */
   DWORD NumberOfSymbols;    /* pe_base + 0x10 */
   WORD  SizeOfOptionalHeader;    /* pe_base + 0x14 */
   WORD  Characteristics;    /* pe_base + 0x16 */
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

Cette structure contient différentes informations, comme le type de processeur utilisé (représenté par le champ "Machine"), le nombre de sections du programme, la date et l'heure de la compilation du programme, la table des symboles si elle existes ou encore les caractéristiques. Ces caractéristiques permettent de savoir si un programme est exécutable ou non (s'il s'agit d'un .exe ou d'un .dll). Je vous renvoie à la MSDN pour plus de détails sur cette structure.

2.2.2 L'Optional Header

Après le File Header, vient l'Optional Header, qui, contrairement à son nom, n'est pas optionnel du tout, puisque c'est cette structure qui contient toutes les informations importantes pour l'exécutable, notamment son Entry Point, ou encore la table des sections de l'exécutable. Nous allons donc étudier plus en détail cet Optional Header.

L'optional header est (une fois de plus) représenté par cette énorme structure :

```
typedef struct _IMAGE_OPTIONAL_HEADER {
   /* Standard fields */

   WORD  Magic;   /* pe_base + 0x18 */
   BYTE  MajorLinkerVersion;   /* pe_base + 0x1A */
   BYTE  MinorLinkerVersion;   /* pe_base + 0x1B */
   DWORD SizeOfCode;   /* pe_base + 0x1C */
   DWORD SizeOfInitializedData;   /* pe_base + 0x20 */
   DWORD SizeOfUninitializedData;   /* pe_base + 0x24 */
   DWORD AddressOfEntryPoint;   /* pe_base + 0x28 */
   DWORD BaseOfCode;   /* pe_base + 0x2C */
   DWORD BaseOfData;   /* pe_base + 0x30 */

   /* NT additional fields */

   DWORD ImageBase;   /* pe_base + 0x34 */
   DWORD SectionAlignment;    /* pe_base + 0x38 */
   DWORD FileAlignment;   /* pe_base + 0x3C */
   WORD  MajorOperatingSystemVersion;   /* pe_base + 0x40 */
   WORD  MinorOperatingSystemVersion;   /* pe_base + 0x42 */
   WORD  MajorImageVersion;   /* pe_base + 0x44 */
   WORD  MinorImageVersion;   /* pe_base + 0x46 */
   WORD  MajorSubsystemVersion;   /* pe_base + 0x48 */
   WORD  MinorSubsystemVersion;   /* pe_base + 0x4A */
   DWORD Win32VersionValue;   /* pe_base + 0x4C */
   DWORD SizeOfImage;   /* pe_base + 0x50 */
   DWORD SizeOfHeaders;   /* pe_base + 0x54 */
   DWORD CheckSum;    /* pe_base + 0x58 */
   WORD  Subsystem;   /* pe_base + 0x5C */
   WORD  DllCharacteristics;   /* pe_base + 0x5E */
   DWORD SizeOfStackReserve;   /* pe_base + 0x60 */
   DWORD SizeOfStackCommit;   /* pe_base + 0x64 */
   DWORD SizeOfHeapReserve;   /* pe_base + 0x68 */
   DWORD SizeOfHeapCommit;   /* pe_base + 0x6C */
```

```
    DWORD LoaderFlags;    /* pe_base + 0x70 */
    DWORD NumberOfRvaAndSizes;    /* pe_base + 0x74 */
    IMAGE_DATA_DIRECTORY DataDirectory[16]; /* pe_base + 0x78 */
    /* pe_base + 0xF8 */
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

3 Sitographie

- http ://sandsprite.com/CodeStuff/Understanding_imports.html
- http ://hi.baidu.com/56hacker/blog/item/aa6993be23330b0d19d81f88.html
- http ://assembly.ifrance.com/pehader.htm
- http ://source.winehq.org/source/include/winnt.h