- PE 파일의 구조와 분석 -

(미 완 성)

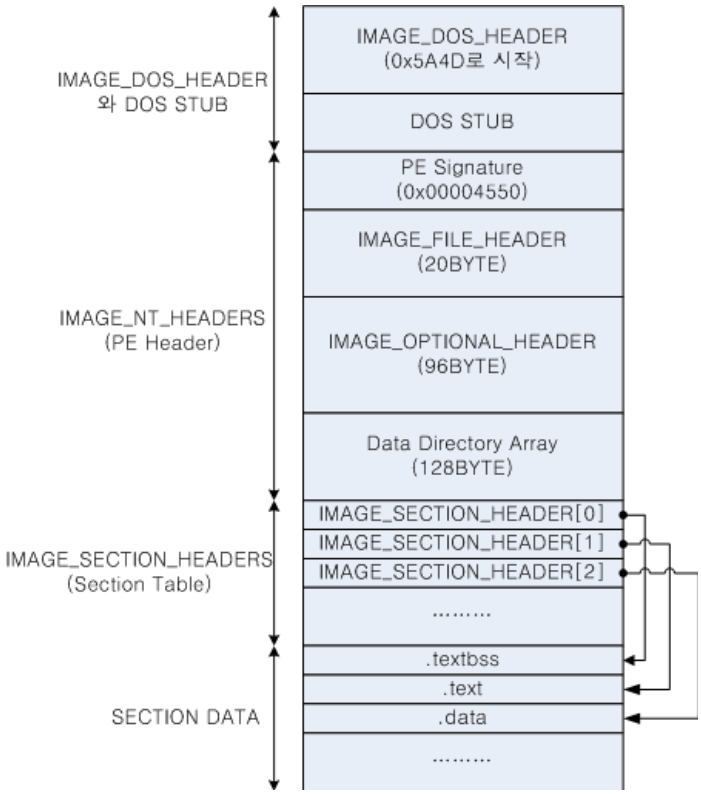
07. 01. 25
수원대학교 보안동아리 FLAG 26회 세미나
윤 석 언(SlaxCore@gmail.com)

악성코드분석, 패킹/언패킹, DLL Injection, 바이너리조작, 기타등등 윈도우즈 바이너리분석에 있어서 필수기본지식인 PE파일에 대하여 틈틈이 공부를 하면서 정리한것입니다.
'Windows 시스템 실행파일의 구조와 원리'라는 책을 기본으로 인터넷상의 PE File Format에 대한 자료들을 참고해가며 틈틈이 공부를 하고 있지만 참고 자료들마다 표현들을 제각각 다르게 해놓은게 해깔리는 부분들이 좀 있어서 제 나름대로 이해하기 쉽게 정리를 한 것이라는것을 말씀드립니다. 그리고 이것은 연구보고서가 아닌 공부한것에 대한 노트필기와 같은것이기 때문에 그냥 참고만 해주세용~!! 내용들은 공부하면서 보충이 될 때마다 업데이트 할 예정입니다. 히힛~

차 례

0. PE파일의 구조

- PE파일이란
  # Portable Excutable의 약자로서 이식 가능한 파일(프로그램)
  # Win32체제에서 공통적으로 사용가능한 기본적인 파일 형식
  # UNIX의 COFF(Common Object File Format)에서 갈라져 나옴.
  # 윈도우즈 바이너리를 분석하기 위한 기초적인 지식
- PE로더란
  # PE 파일 포맷을 읽어들여 실행시에 메모리로 로드하는 역할을 한다.
- Win32 PE파일의 형식
  # EXE, DLL, OCX, CPL, .NET응용프로그램
- PE파일포맷 전체적인 구조의 모습



1. IMAGE_DOS_HEADER 구조와 DOS STUB

- PE파일은 몇 십바이트의 MS-DOS 스텁으로 시작한다.
- MS-DOS스텁은 PE에서는 큰 의미가 없다.
- IMAGE_DOS_HEADER의 구조체(WinNT.H)

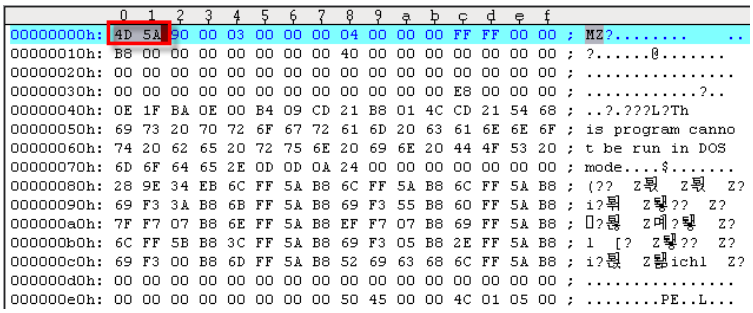
```
typedef struct _IMAGE_DOS_HEADER {      // DOS .EXE header
    WORD   e_magic;                     // Magic number
    WORD   e_cblp;                      // Bytes on last page of file
    WORD   e_cp;                        // Pages in file
    WORD   e_crlc;                      // Relocations
    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;
```

위 IMAGE_DOS_HEADER 구조체중 실제로 눈여겨 봐야 할것은 첫 번째 필드인 e_magic과 마지막 필드인 e_lfanew이다.

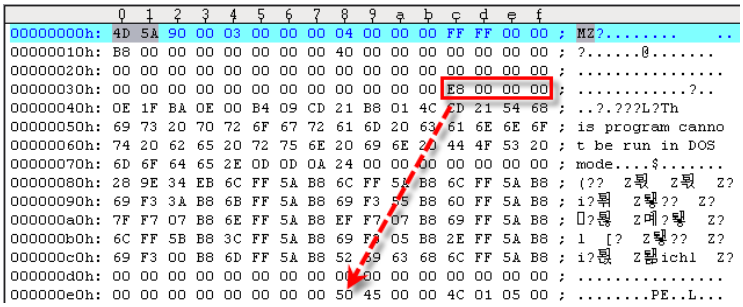
- 모든 Win32 PE파일은 IMAGE_DOS_HEADER로 시작한다.

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 ; MZ?.....  ..
00000010h: B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ; ?.....@.....
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 E8 00 00 00 ; ............?..
00000040h: 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ; ..?.???L?Th
00000050h: 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F ; is program canno
00000060h: 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 ; t be run in DOS
00000070h: 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 ; mode....$.......
00000080h: 28 9E 34 EB 6C FF 5A B8 6C FF 5A B8 6C FF 5A B8 ; (??  Z퉋  Z퉋  Z?
00000090h: 69 F3 3A B8 6B FF 5A B8 69 F3 55 B8 60 FF 5A B8 ; i?뒦  Z퉋??  Z?
000000a0h: 7F F7 07 B8 6E FF 5A B8 EF F7 07 B8 69 FF 5A B8 ; ?퉋  Z뎨?퉋  Z?
000000b0h: 6C FF 5B B8 3C FF 5A B8 69 F3 05 B8 2E FF 5A B8 ; l  [?  Z퉋??  Z?
000000c0h: 69 F3 00 B8 6D FF 5A B8 52 69 63 68 6C FF 5A B8 ; i?뒦  Z퉋ichl  Z?
000000d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000000e0h: 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ; ........PE..L...
```

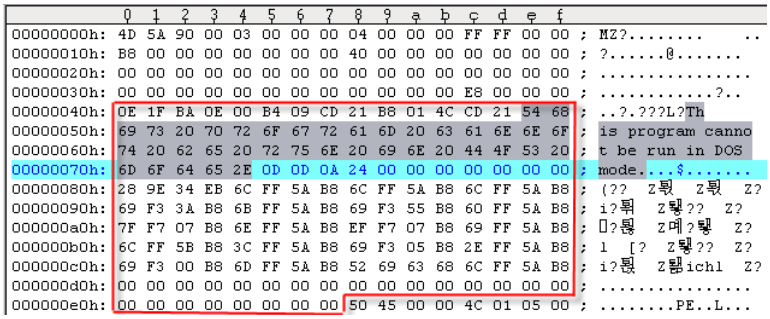
- e_magic은 ASCII "MZ"로 고정(0x5A4D) 되어 있다.



- e_lfanew 필드는 실제 PE파일의 시작이라고 할 수 있는 IMAGE_NT_HEADER의 오프셋값을 가진다.



- 위 그림을 보면 ₩x50₩x45(PE)가 오프셋 0x000000e8에서 시작함을 확인할 수 있다.
- PE파일을 열고 처음 IMAGE_DOS_HEADER를 읽어 들인 다음 e_magic이 "MZ"인지 확인한 후 e_lfanew필드값을 읽어서 이 값만큼 파일 포인터를 이동시키면서 본격적으로 PE파일포맷을 분석할 수 있다.
- DOS STUB



- 그러나 이 부분은 PE에선 의미가 없는 부분이다.(그냥 이런게 있다 라는 정도만 확인)

2. IMAGE_NT_HEADERS의 구조

- "PE₩x0₩x0" + IMAGE_FILE_HEADER + IMAGE_OPTIONAL_HEADER

```
000000e0h: 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ; ........PE..L...
000000f0h: 11 6A CB 41 00 00 00 00 00 00 00 00 E0 00 0F 01 ; .j?........?..
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□......
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 00 40 00 ; ..............@.
00000120h: 00 10 00 00 00 10 00 00 04 00 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
00000160h: 00 00 00 00 00 00 00 00 00 00 B0 02 00 50 00 00 ; .........?.P...
00000170h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000180h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000190h: 00 40 02 00 1C 00 00 00 00 00 00 00 00 00 00 00 ; .@..............
000001a0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001b0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001c0h: 50 B2 02 00 00 02 00 00 00 00 00 00 00 00 00 00 ; P?..............
000001d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001e0h: 2E 74 65 78 74 62 73 73 00 00 01 00 00 10 00 00 ; .textbss........
000001f0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000200h: 00 00 00 00 A0 00 00 E0 2E 74 65 78 74 00 00 00 ; ....?.?text...
```

- 실제 PE파일의 시작이다.(PE Header)
- PE에 관계된 많은 부분을 담고 있다.
- 구조체의 모습

```
typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

2.1 DWORD Signature

- PE파일임을 나타내는 시그네쳐로서 항상 "PE₩x00₩x00" 이다.
- IMAGE_DOS_HEADER구조체의 e_lfanew필드가 가리키는 오프셋으로부터 4바이트 값이다.

```
#define IMAGE_DOS_SIGNATURE          0x4D5A     // MZ
#define IMAGE_OS2_SIGNATURE          0x4E45     // NE
#define IMAGE_OS2_SIGNATURE_LE       0x4C45     // LE
#define IMAGE_NT_SIGNATURE           0x50450000 // PE00
```

```
000000e0h: 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ; ........PE..L...
000000f0h: 11 6A CB 41 00 00 00 00 00 00 00 00 E0 00 0F 01 ; .j?........?..
```

2.2 IMAGE_FILE_HEDER 구조

```
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;
```

- PE 시그네쳐 다음에 오는 20바이트로 구성.

```
000000e0h: 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ; ........PE..L...
000000f0h: 11 6A CB 41 00 00 00 00 00 00 00 00 E0 00 0F 01 ; .j?........?..
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□......
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 00 40 00 ; ..............@.
00000120h: 00 10 00 00 00 10 00 00 04 00 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
00000160h: 00 00 00 00 00 00 00 00 00 00 B0 02 00 50 00 00 ; .........?.P...
00000170h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000180h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000190h: 00 40 02 00 1C 00 00 00 00 00 00 00 00 00 00 00 ; .@..............
000001a0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001b0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001c0h: 50 B2 02 00 00 02 00 00 00 00 00 00 00 00 00 00 ; P?..............
000001d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001e0h: 2E 74 65 78 74 62 73 73 00 00 01 00 00 10 00 00 ; .textbss........
000001f0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000200h: 00 00 00 00 A0 00 00 E0 2E 74 65 78 74 00 00 00 ; ....?.?text...
```

2.2.1 WORD Machine

- 파일이 실행된 CPU ID를 나타냄.

```
#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_POWERPC       0x01F0 // IBM PowerPC Little-Endian
#define IMAGE_FILE_MACHINE_SH3           0x01a2 // SH3 little-endian
#define IMAGE_FILE_MACHINE_SH3E          0x01a4 // SH3E little-endian
#define IMAGE_FILE_MACHINE_SH4           0x01a6 // SH4 little-endian
#define IMAGE_FILE_MACHINE_ARM           0x01c0 // ARM Little-Endian
#define IMAGE_FILE_MACHINE_THUMB         0x01c2
#define IMAGE_FILE_MACHINE_IA64          0x0200 // Intel 64
#define IMAGE_FILE_MACHINE_MIPS16        0x0266 // MIPS
#define IMAGE_FILE_MACHINE_MIPSFPU       0x0366 // MIPS
#define IMAGE_FILE_MACHINE_MIPSFPU16     0x0466 // MIPS
#define IMAGE_FILE_MACHINE_ALPHA64       0x0284 // ALPHA64
#define IMAGE_FILE_MACHINE_AXP64         IMAGE_FILE_MACHINE_ALPHA64
```

```
000000E0h: 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ; ........PE..L...
000000F0h: 11 6A CB 41 00 00 00 00 00 00 00 00 E0 00 0F 01 ; .j?........?..
```

2.2.2 WORD NumberOfSections

- 파일의 섹션의 수
- IMAGE_SECTION_HEADER 구조체 배열의 원소의 개수와 해당 섹션의 개수를 나타냄.
- 섹션은 추가/삭제시 이값을 조작.

```
000000E0h: 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ; ........PE..L...
000000F0h: 11 6A CB 41 00 00 00 00 00 00 00 00 E0 00 0F 01 ; .j?........?..
```

2.2.3 DWORD TimeDateStamp

- 1970년 1월 1일 9시(GMT 시간 기준)으로부터 해당 파일을 만들어낸 시점까지의 시간을 초단위로 표현함.

```
000000E0h: 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ; ........PE..L...
000000F0h: 11 6A CB 41 00 00 00 00 00 00 00 00 E0 00 0F 01 ; .j?........?..
```

2.2.4 DWORD PointerToSymbolTable, DWORD NumberOfSymbols

- 디버그 정보를 가진 PE파일에서만 사용.
- PE파일내에서는 보기 힘들다.

```
000000E0h: 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ; ........PE..L...
000000F0h: 11 6A CB 41 00 00 00 00 00 00 00 00 E0 00 0F 01 ; .j?........?..
```

2.2.5 WORD SizeOfOptionalHeader

- IMAGE_OPTIONAL_HEADER 구조체의 바이트수.
- obj파일의 경우는 0, 실행파일의 경우는 sizeof(IMAGE_OPTIONAL_HEADER)

```
000000E0h: 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ; ........PE..L...
000000F0h: 11 6A CB 41 00 00 00 00 00 00 00 00 E0 00 0F 01 ; .j?........?..
```

2.2.6 WORD Characterstics

- 해당 PE파일에 대한 특정 정보를 나타내는 플래그

```
#define IMAGE_FILE_RELOCS_STRIPPED         0x0001 // Relocation info stripped from file.
#define IMAGE_FILE_EXECUTABLE_IMAGE        0x0002 // File is executable  (i.e. no unresolved externel 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_AGGRESIVE_WS_TRIM       0x0010 // Agressively 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.
```

```
000000E0h: 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ; ........PE..L...
000000F0h: 11 6A CB 41 00 00 00 00 00 00 00 00 E0 00 0F 01 ; .j?........?..
```

2.3 IMAGE_OPTIONAL_HEADER 구조

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

  WORD   Magic;
  BYTE   MajorLinkerVersion;
  BYTE   MinorLinkerVersion;
  DWORD  SizeOfCode;
  DWORD  SizeOfInitializedData;
  DWORD  SizeOfUninitializedData;
  DWORD  AddressOfEntryPoint;
  DWORD  BaseOfCode;
  DWORD  BaseOfData;

  //
  // NT additional fields.
  //

  DWORD  ImageBase;
  DWORD  SectionAlignment;
  DWORD  FileAlignment;
  WORD   MajorOperatingSystemVersion;
  WORD   MinorOperatingSystemVersion;
  WORD   MajorImageVersion;
  WORD   MinorImageVersion;
  WORD   MajorSubsystemVersion;
  WORD   MinorSubsystemVersion;
  DWORD  Win32VersionValue;
  DWORD  SizeOfImage;
  DWORD  SizeOfHeaders;
  DWORD  CheckSum;
  WORD   Subsystem;
  WORD   DllCharacteristics;
  DWORD  SizeOfStackReserve;
  DWORD  SizeOfStackCommit;
  DWORD  SizeOfHeapReserve;
  DWORD  SizeOfHeapCommit;
  DWORD  LoaderFlags;
  DWORD  NumberOfRvaAndSizes;
  IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

- IMAGE_FILE_HEADER에 다음 224바이트로 구성됨.(36개의 기본 필드(96바이트), 8바이트 IMAGE_DATA_DIRECTORY 구조체의 16개 배열(128바이트))

```
000000e0h: 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ; ........PE..L...
000000f0h: 11 6A CB 41 00 00 00 00 00 00 00 00 E0 00 0F 01 ; .j?........?..
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□......
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 00 40 00 ; ..............@.
00000120h: 00 10 00 00 00 10 00 00 00 04 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
00000160h: 00 00 00 00 00 00 00 00 00 B0 02 00 50 00 00 00 ; .........?.P...
00000170h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000180h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000190h: 00 40 02 00 1C 00 00 00 00 00 00 00 00 00 00 00 ; .@..............
000001a0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001b0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001c0h: 50 B2 02 00 00 02 00 00 00 00 00 00 00 00 00 00 ; P?..............
000001d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001e0h: 2E 74 65 78 74 62 73 73 00 00 01 00 00 10 00 00 ; .textbss........
000001f0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000200h: 00 00 00 00 A0 00 00 E0 2E 74 65 78 74 00 00 00 ; ....?.?text...
```

- 중요한 정보들을 많이 담고 있는 헤더.

2.3.1 IMAGE_OPTIONAL_HEADER의 기본필드(96바이트)


```
000000e0h: 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ; ........PE..L...
000000f0h: 11 6A CB 41 00 00 00 00 00 00 00 00 E0 00 0F 01 ; .j?........?..
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□......
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 00 40 00 ; ..............@.
00000120h: 00 10 00 00 00 10 00 00 00 04 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
00000160h: 00 00 00 00 00 00 00 00 00 B0 02 00 50 00 00 00 ; .........?.P...
00000170h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000180h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000190h: 00 40 02 00 1C 00 00 00 00 00 00 00 00 00 00 00 ; .@..............
000001a0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001b0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001c0h: 50 B2 02 00 00 02 00 00 00 00 00 00 00 00 00 00 ; P?..............
000001d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001e0h: 2E 74 65 78 74 62 73 73 00 00 01 00 00 10 00 00 ; .textbss........
000001f0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000200h: 00 00 00 00 A0 00 00 E0 2E 74 65 78 74 00 00 00 ; ....?.?text...
```


(1) WORD Magic (*)

- IMAGE_OPTIONAL_HEADER를 나타내는 시그네쳐
- 32비트 PE인 경우 0x010B, 64비트 PE인 경우 0x020B, ROM 이미지 파일인 경우 0x0107.

```
#define IMAGE_NT_OPTIONAL_HDR32_MAGIC      0x10b
#define IMAGE_NT_OPTIONAL_HDR64_MAGIC      0x20b
#define IMAGE_ROM_OPTIONAL_HDR_MAGIC       0x107
```

```
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□.....
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 00 40 00 ; ..............@.
00000120h: 00 10 00 00 00 10 00 00 04 00 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
```

〈 32비트 PE임을 나타냄 〉

(2) BYTE MajorLinkerVersion, BYTE MinorLinkerVersion

- 파일을 만들어낸 링커의 버전

```
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□.....
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 00 40 00 ; ..............@.
00000120h: 00 10 00 00 00 10 00 00 04 00 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
```

〈 링커버전 7.10 〉

(3) DWORD SizeOfCode (*)

- 모든 코드 섹션의 사이즈를 합한 크기
- 일반적으로 .text섹션의 바이트 수와 같다.

```
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□.....
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 00 40 00 ; ..............@.
00000120h: 00 10 00 00 00 10 00 00 04 00 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
```

〈 코드 섹션의 크기 : 0x00013000 〉

(4) DWORD SizeOfInitializedData

- 초기화된 데이터 섹션의 전체 크기를 나타냄(코드섹션 제외)

```
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□.....
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 00 40 00 ; ..............@.
00000120h: 00 10 00 00 00 10 00 00 04 00 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
```

〈 초기화된 데이터 섹션의 크기 : 0x00008000 〉

(5) DWORD SizeOfUnInitializedData

- SizeOfInitializedData와 반대 개념
- 초기화되지 않은 데이터 섹션의 바이트 수를 나타냄
- 보통 이 필드는 0이다.

```
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□.....
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 00 40 00 ; ..............@.
00000120h: 00 10 00 00 00 10 00 00 04 00 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
```

〈 초기화 되지 않은 데이터 섹션의 크기 〉

(6) DWORD AddressOfEntryPoint (*)

- 프로그램 코드의 시작 주소를 나타낸다.(프로그램 진입점)
- 메인 스레드가 최초로 실행시킬 프로그램 코드의 시작번지에 대한 상대 주소(RVA)값이다.
- 가리키고 있는 값은 .text 섹션의 특정번지에 위치한다.
- WinMain, DllMain같은 런타임 시작 루틴으로 점프하는 코드가 담겨 있다.
- 임의로 새로운 섹션을 만들고 이 필드의 값을 그 안의 값을 가리키게 할 수도 있다.

```
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□.....
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 40 00 ; .............@.
00000120h: 00 10 00 00 00 10 00 00 04 00 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
```

〈 메모리 상에서는 ImageBase의 필드 값(0x00400000)을 더한 0x0041142E가 된다. 〉

(7) DWORD BaseOfCode

- 코드섹션의 첫 번째 바이트에 대한 RVA를 의미
- 마이크로 소프트 링커로 만든 파일은 0x00001000

```
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□.....
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 40 00 ; .............@.
00000120h: 00 10 00 00 00 10 00 00 04 00 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
```

〈 코드의 시작은 0x004010000이 된다. 〉

(8) DWORD BaseOfData

- 데이터 섹션(.data)의 시작주소에 대한 RVA
- 64비트 PE에서는 나타나지 않음.

(9) DWORD ImageBase (*)

- PE파일이 메모리에 매핑될때 메모리상의 시작주소.
- 기본 BaseImage 값은 EXE : 0x00400000, DLL : 0x10000000

```
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□.....
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 40 00 ; .............@.
00000120h: 00 10 00 00 00 10 00 00 04 00 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
```

〈 메모리 상에서의 PE파일의 시작주소 : 0x00400000 〉

(10) DWORD SectionAlignment

- PE파일이 메모리에 매핑될때 각 섹션의 배치 간격(각 섹션의 시작주소는 이 필드값의 배수)

(11) DWORD FileAlignment

- PE파일 내에서의 섹션들의 정렬 단위.
- 파일 내 각각의 섹션을 구성하는 바이너리 데이터들의 시작 주소간의 간격이라고 보면 됨.

(12) WORD MajorOperationgSystemVersion, WORD MinorOperationgSystemVersion

- 해당 PE파일을 실행하는 데 필요한 운영체제의 최소 버전.

(13) WORD MajorImageVersion, WORD MinorImageVersion

- 유저가 임의로 정하는 EXE나 DLL의 버전

(14) WORD MajorSubsystemVersion, WORD MinorSubsystemVersion

- 해당 PE를 실행하는데 필요한 서브 시스템의 최소버전

(15) DWORD Win32VersionValue

- 보통 0으로 설정된다.

(16) DWORD SizeOfImage (*)

- 로더가 PE를 메모리상에 로드할 때 확보해야할 충분한 크기
- 보통 PE파일의 크기보다 크다.
  (만약 파일이 패킹이 되어 있다고 한다면 패킹루틴만큼의 파일 크기가 증가)
- 반드시 SectionAlignment 필드값의 배수가 되어야 함.

```
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□.....
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 00 40 00 ; ..............@.
00000120h: 00 10 00 00 00 10 00 00 04 00 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
```

〈 메모리에 확보된 PE 이미지의 크기 〉

(17) DWORD SizeOfHeader

- MS-DOS헤더, PE헤더, 섹션테이블의 크기를 합친 만큼의 바이트 수.
- 반드시 FileAlignment 필드값의 배수가 되어야 함.

```
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□.....
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 00 40 00 ; ..............@.
00000120h: 00 10 00 00 00 10 00 00 04 00 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
```

〈 모든 헤더의 크기 : 0x00001000(4K바이트) 〉

(18) DWORD CheckSum

- 이미지의 체크섬 값
- 커널 모드 드라이버나 어떤 시스템 DLL의 경우 요구 됨
- 그 외의 경우는 0(설정 안됨)이다.

(19) WORD Subsystem

- 해당 실행파일이 유저 인터페이스로 사용하는 서브시스템의 종류를 나타냄.

```
// Subsystem Values

#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.
```

(20) WORD DllCharacteristics

- 언제 DLL초기화함수(DllMain())가 호출되어야 하는지 지시하는 플래그

(21) DWORD SizeOfStackReserve, DWORD SizeOfStackCommit
DWORD SizeOfHeapReserve, DWORD SizeOfHeapCommit

- 프로세스는 메모리 공간상에 자신만의 스택과 힙을 별도로 갖는다.
- PE가 메모리에 로드될때 시스템이 디폴트 스택과 힙을 만들어주기 위해 참조하는 필드

(22) DWORD LoaderFlags

- 0으로 세트 된다.

(23) DWORD NumberOfRvaAndSizes

- IMAGE_DATA_DIRECTORY 구조체 배열의 원소 개수를 나타냄.
- 항상 16개이다.(0x00000010)

2.3.2 IMAGE_DATA_DIRECTORY 구조체 배열

IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES]

- 전체 128바이트로 이루어짐.

```
000000e0h: 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ; ........PE..L...
000000f0h: 11 6A CB 41 00 00 00 00 00 00 00 00 E0 00 0F 01 ; .j?........?..
00000100h: 0B 01 07 0A 00 30 01 00 00 80 00 00 00 00 00 00 ; .....0...□......
00000110h: 2E 14 01 00 00 10 00 00 00 10 00 00 00 00 40 00 ; ..............@.
00000120h: 00 10 00 00 00 10 00 00 04 00 00 00 00 00 00 00 ; ................
00000130h: 04 00 00 00 00 00 00 00 00 C0 02 00 00 10 00 00 ; .........?.....
00000140h: 00 00 00 00 02 00 00 00 00 00 00 10 00 00 10 00 00 ; ................
00000150h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ; ................
00000160h: 00 00 00 00 00 00 00 00 00 B0 02 00 50 00 00 00 ; .........?.P...
00000170h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000180h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000190h: 00 40 02 00 1C 00 00 00 00 00 00 00 00 00 00 00 ; .@..............
000001a0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001b0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001c0h: 50 B2 02 00 00 02 00 00 00 00 00 00 00 00 00 00 ; P?..............
000001d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001e0h: 2E 74 65 78 74 62 73 73 00 00 01 00 00 10 00 00 ; .textbss........
000001f0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000200h: 00 00 00 00 A0 00 00 E0 2E 74 65 78 74 00 00 00 ; ....?.?text...
```

- IMAGE_NUMBEROF_DIRECTORY_ENTRIES의 개수는 16으로 정의되어짐
- 배열의 마지막 원소는 항상 0으로 세팅된다.
- 마지막 원소는 항상 0이기 때문에 0~14(15개)의 원소만 사용이 된다.
- WinNT.H에 정의된 구조체의 모습

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

#define IMAGE_NUMBEROF_DIRECTORY_ENTRIES    16
```

- 배열 원소에 대한 정의

```
// 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
```

3. IMAGE_SECTION_HEADER

- Section Table이라고도 불린다.
- PE헤더(IMAGE_NT_HEADERS)다음 40바이트로 구성된 배열
- 섹션이 5개라고 하면 섹션헤더에 정의되어진 배열이 5개가 되고 마지막에 NULL로 채워진다.
- 섹션정보의 개수는 IMAGE_FILE_HEADER의 NumberOfSections멤버가 갖고 있다.
- WinNT.H에 정의된 구조체 모습

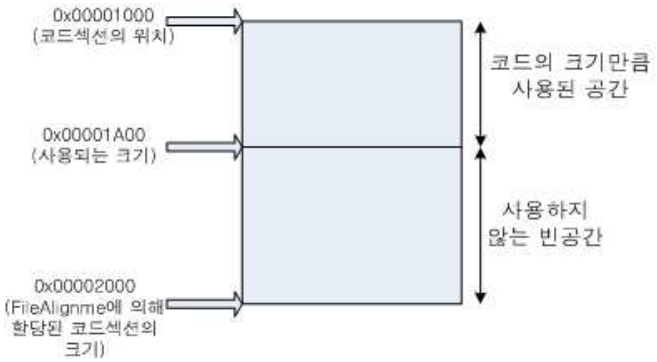
```
//
// Section header format.
//

#define IMAGE_SIZEOF_SHORT_NAME              8

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

#define IMAGE_SIZEOF_SECTION_HEADER          40
```

(1) BYTE Name[IMAGE_SIZEOF_SHORT_NAME]
- 섹션의 네임을 나타내는 멤버이다.
- 섹션의 이름을 나타내는 문자열을 저장한다.
- IMAGE_SIZEOF_SHORT_NAME은 8바이트 까지이며 NULL은 제외된다.
- .text나 .code같은 이름을 갖지만 이름으로 섹션의 성격까지 파악을 하면 안된다.
- 이 멤버의 값은 NULL일수도 있다.
- 섹션이름이 같다고 하여 같은 속성을 갖지는 않는다.
- 섹션의 속성을 파악하기 위해선 Characterstics멤버를 참조해야 한다.

(2) DWORD PhysicalAddress / VirtualSize
- Misc 유니온 구조체(공용체)의 멤버
- OBJ파일인 경우와 PE인 경우 의미가 달라짐
- PE의 경우 VirtualSize필드를 사용
- PE로더에 의해 메모리에 올려진 후에 해당 섹션이 얼마만큼의 크기를 갖는지에 대한 정보
- PhysicalAddress필드는 더 이상 의미가 없음

(3) DWORD VirtualAddress
- PE에서 해당 섹션이 가상주소공간에 매핑되었을때 RVA 값.
- OPTIONAL_HEADER의 SectionAlignment값의 배수값을 갖는다.
- ImageBase값을 기준으로 하는 RVA 값이다.

(4) DWORD SizeOfRawData (*)
- RawData상에서 해당섹션에 대한 실제 사용된 크기의 정보.
- 해당 섹션이 얼마만큼의 빈공간이 있는지 알아낼때 필요한 정보
- PE로더는 이 값을 통하여 섹션이 메모리에 얼마나 올라가야 하는지 안다.
- 옵셔널헤더의 FileAlignment값의 배수값을 갖는다.
- 코드섹션의 위치(PointerToRawData) : 0x00001000
  코드섹션에서 사용되는 크기(Misc.VitualSize) : 0x000000A00
  FileAlignment : 0x00001000



- SizeOfRawData : 코드의 크기만큼 실제 사용된 공간 + 사용되지 않는 빈공간
- RawData상에서 윈도우즈는 FileAlignment값에 영향을 받아 섹션의 크기를 할당하므로 해당 섹션의 크기에 관계없이 FileAlignment값의 배수만큼의 크기로 섹션공간을 할당한다.
- 이러한 섹션 관리법으로 인해 대개의 섹션들은 빈공간이 남아있다.(NULL로 채워짐)
- 빈공간에 임의의 코드를 작성함으로서 프로그램을 조작할 수 있다.
- 섹션을 직접 추가하거나 섹션테이블의 값을 조정하여 현재 존재하는 섹션의 크기를 늘릴수도 있다.

(5) DWORD PointerToRawData
- 해당 섹션의 PE파일상에서의 선두로부터의 오프셋.

- FileAlignment값의 배수값.

(6) DWORD PointerToRelocations
- 해당 섹션을 위한 재배치 파일 오프셋
- OBJ에서만 사용됨(실행파일에서는 0이된다.)
- OBJ파일에서 0이 아닐 경우 IMAGE_RELOCATION구조체 배열의 시작을 가리킴

(7) DWORD PointerToLinenumbers
- COFF(Common Object File Format) 라인번호가 PE에 첨부되었을 경우에만 사용.
- 값이 0이 아닐 경우 IMAGE_LINENUMBER구조체 배열의 시작을 가리킴.

(8) WORD NumberOfRelocations
- PointerToRelocations 필드가 가리키는 IMAGE_RELOCATION 구조체 배열의 원소의 개수
- 실행파일에서는 항상 0 이다.

(9) WORD NumberOfLinenymvers
- PointerToLinenumbers 필드가 가리키는 IMAGE_LINENUMBER 구조체 배열의 원소의 개수
- COFF 라인번호사 PE에 첨부되었을때만 사용.

(10) DWORD Characteristics
- 해당 섹션의 속성을 나타내는 플래그의 집합.
- WinNT.H에 IMAGE_SCN_XXX_XXX의 형태로 #define문에 의해 정의되어 있음.

```
#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.
```

- IMAGE_SCN_CNT_CODE 0x00000020
  # 코드로 채워진 섹션임을 나타냄
  # 보통 실행가능 플래그를 의미하는 IMAGE_SCN_MEM_EXECUTE 플래그와 함께 지정됨.
- IMAGE_SCN_CNT_INITIALIZED_DATA 0x00000040
  # 데이터가 초기화된 섹션
  # 실행가능 섹션과 .bbs 섹션을 제외한 거의 대부분의 섹션이 이 플래그를 갖는다.
- IMAGE_SCN_CNT_UNINITIALIZED_DATA
  # 데이터가 초기화되지 않은 섹션
  # 예를 들면 .bbs섹션

```
#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.
```

〈 해당 섹션의 메모리 페이지 속성을 나타내는 플래그의 집합 〉

- IMAGE_SCN_MEM_DISCARDABLE 0X02000000
  # 일단 메모리에 매핑되고 난 후 프로세스에게 더 이상 의미가 없는 섹션임을 나타냄
  # 가장 일반적인 폐기 가능 섹션은 .reloc(기본재배치)섹션이다.
- IMAGE_SCN_MEM_EXECUTE 0x20000000
  # 코드로서 실행될수 있는 섹션임을 나타냄
  # 보통 IMAGE_SCN_CNT_CODE 플래그와 함께 세트 됨.
- IMAGE_SCN_MEM_READ 0x40000000
  # 읽기 가능영역 섹션임을 나타냄.
  # 언제나 EXE파일 내의 대부분의 섹션들에 세트됨.

- IMAGE_SCN_MEM_WRITE 0x80000000
  # 쓰기 가능영역 섹션임을 나타냄.
  # 쓰기 가능 섹션의 전형적인 예 : .data .bbs

- 실제 예

```
000001e0h: 2E 74 65 78 74 62 73 73 00 00 01 00 00 10 00 00 ; .textbss........
000001f0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000200h: 00 00 00 00 A0 00 00 E0 2E 74 65 78 74 00 00 00 ; ....?.?text...
00000210h: A7 2D 01 00 00 10 01 00 00 30 01 00 00 10 00 00 ; ?.......0......
00000220h: 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60 ; ............ ..`
00000230h: 2E 72 64 61 74 61 00 00 9C 2E 00 00 00 40 02 00 ; .rdata..?...@..
00000240h: 00 30 00 00 00 40 01 00 00 00 00 00 00 00 00 00 ; .0...@..........
00000250h: 00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00 ; ....@..@.data...
00000260h: 4C 30 00 00 00 70 02 00 00 20 00 00 00 70 01 00 ; L0...p... ...p..
00000270h: 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 C0 ; ............@..?
00000280h: 2E 69 64 61 74 61 00 00 79 0B 00 00 00 B0 02 00 ; .idata..y....?.
00000290h: 00 10 00 00 00 90 01 00 00 00 00 00 00 00 00 00 ; .....?.........
000002a0h: 00 00 00 00 40 00 00 C0 00 00 00 00 00 00 00 00 ; ....@..?.......
```

필드	섹션 테이블				
Offset	0x000001e0	0x00000208	0x00000230	0x00000258	0x00000280
Name	.textbss	.text	.rdata	.data	.idata
VirtualSize	0x00010000	0x00012DA7	0x00002E90	0x0000304C	0x00000B79
VirtualAddress	0x00001000	0x00011000	0x00024000	0x00027000	0x0002B000
SizeOfRawData	0x00000000	0x00013000	0x00003000	0x00002000	0x00001000
PointerToRawData	0x00000000	0x00001000	0x00014000	0x00017000	0x00019000
PointerToRelocations	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
PointerToLinenumbers	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
NumberOfRelocations	0x0000	0x0000	0x0000	0x0000	0x0000
NumberOfLinenumbers	0x0000	0x0000	0x0000	0x0000	0x0000
Characteristics	0xE00000A0	0x60000002	0x40000040	0xC0000040	0xC0000040

- Characteristics값을 확인해보면...(예시)

```
# .textbss                                0xE00000A0
  ----------------------------------------
  IMAGE_SCN_MEM_EXECUTE                   0x20000000
  IMAGE_SCN_MEM_READ                      0x40000000
  IMAGE_SCN_MEM_WRITE                     0x80000000
  IMAGE_SCN_CNT_CODE                      0x00000020
  IMAGE_SCN_CNT_UNINITIALIZED_DATA        0x00000080
  ----------------------------------------
```

-〉 코드섹션과 초기화되지 않은 데이터가 있는 .bbs섹션이 혼합되어 있다.
-〉 메모리 속성으로는 실행, 읽기, 쓰기 속성
-〉 참고로 .bss(초기화되지 않은 데이터를 보관)섹션이 VC++7.0에서는 .textbss로 바뀐것이

고 디버깅모드에서만 생성됨(거의 사용안함)

- PE파일에서 해당 섹션의 실제 내용을 확인하려면..
  # PointerToRawData가 가리키는 위치로 파일 오프셋을 이동
  # 거기서부터 SizeOfRawData의 바이트 수만큼이 해당 섹션의 내용이 된다.
  # 이 섹션이 실제 메모리에 매핑되었을때의 RVA가 VirtualAddress가 된다.
  # 매핑후 섹션의 실제 시작포인터를 얻으려면 ImageBase + VirtualAddress
  # 매핑된 후 이 섹션이 차지하고 있는 메모리상의 크기는 VirtualSize에 명시된다.

4. 코드섹션