기술문서	08. 10. 30. 작성

PE 파일구조

작성자 : 한서대학교 H.I.S.L 동아리 진선호 sunho104@msn.com

2008. 10. 30

한서대학교 정보보호동아리
Since 2007 Hanseo University Infomation Security Lab.

◎ 목 차

1. 개 요

PE파일에 대해서 공부한 내용을 정리해 보았습니다. 아직 공부를 하는 학생이기에 부족한 면이 없지않아 있지만 PE파일구조에 관해 공부하는 분들에게 도움이 되었으면 하고 제작을 하였습니다.

최대한 쉽게 설명을 하기위해 노력했으며 더욱 깊게 공부하고 싶으신분들은 Windows시스템 실행파일의 구조와 원리(한빛미디어) 책을 꼭 한번 읽어보라고 권해드리고 싶습니다.

본문은 각종 서적과 인터넷 자료를 많이 카피 하였으며 어떤부분은 책과 거의 동일한 내용을 담았습니다. 아직 부족해서 재 해석할 능력이 안되기때문이라고 생각해 주시고 너그럽게 봐주셨으면 합니다. 문서에 틀린 부분이나 수정할 여지가 있으면 연락을 주시기 바랍니다.

2. PE(Portable Executable)이란?

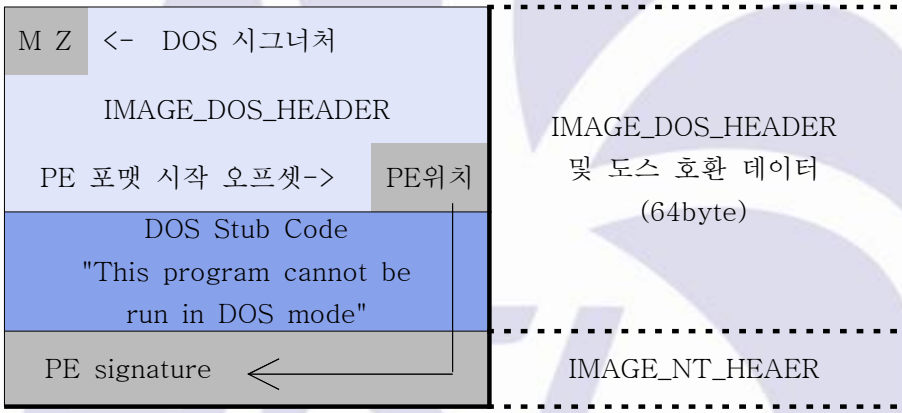
PE(Portable Executable File Format)란 Win32 운영체제라면 어디에서라도 잘 돌아가는 파일 포맷을 말한다. exe, obj, dll, sys 파일도 모두 PE 파일 구조를 가지고 있으며 Win32 Platform SDK의 WinNT.H 헤더 파일에 PE관련 구조체들이 선언되어 있다.

<PE 형식 구조>

3. IMAGE_DOS_HEADER

PE형식을 잠시 보면은 첫 2바이트를 차지하고 있는 0x4d, 0x5a('MZ')로 시작되는 것을 볼 수 있다. MZ는 DOS 개발자인 Mark Zbikowski라는 사람의 이니셜이며 DOS에서 실행시킬수 있는 파일을 의미한다. DOS용 프로그램과 Windows용 프로그램을 같이 사용하던 시절에 DOS에서도 Windows용 프로그램이 정상적으로 작동하는 프로그램을 제작하기 위해서 앞부분에 더미 DOS 프로그램을 덧붙였다.

그래서 DOS에서 윈도우 프로그램을 실행시켰을 때 "This program cannot be run in DOS mode"라며 더미 DOS 프로그램이 메시지를 출력하고 종료(DOS Stub Code)가 되는 것이다.



IMAGE_DOS_HEADER 마지막 4바이트에는 PE 포맷 시작 오프셋 값이 존재하고 있고 그 값만큼 파일 포인터를 이동시키면 'PE\0\0'을 만날 수 있으며 PE 포맷의 시작을 의미하는 시그너처이다.

IMAGE_DOS_HEADER에서 가장 중요한 부분은 처음 2byte를 차지하는 e_magic(MZ) 과 마지막 4byte를 차지하는 e_lfanew(PE) 외에는 크게 중요하지 않다. 아래 선언 부분에서 19개의 멤버가 정의되 있지만 0으로 채워도 아무런 문제를 잃으키지는 않는다.

```
typedef struct_IMAGE_DOS_HEADER{      // DOS .EXE header
    WORD e_magic;                     // Magic number (MZ)
    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];                  // Resverved words
    WORD e_lfanew;                    // File address of new exe header (PE)
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

<WinNT.H에 정의되 있는 IMAGE_DOS_HEADER 구조체>

4. IMAGE_NT_HEADER

1) IMAGE_FILE_HEADER

'PE\0\0'을 만나고 나면 바로 COFF(Common Objet File Format) 헤더부분이 나오는데 아래와 같은 구조를 가진다.

```
typedef struct_IMAGE_FILE_HEADER{
    WORD     Machine;                 // 타겟 머신 타입(예 Intel의 80386)
    WORD     NumberOfSections;        // 파일에 포함된 섹션 수
    DWORD    TimeDateStamp;           // 파일 제작일시
    DWORD    PointerToSymbolTable;    // 심볼 테이블 오프셋
    DWORD    NumberOfSymbols;;        // 심볼 테이블의 엔트리 수
    WORD     SizeOfOptionalHeader;    // 옵션 헤더의 크기
    WORD     Characteristics;         // 파일 속성 플래그
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

<WinNT.H에 정의되 있는 IMAGE_FILE_HEADER 구조체>

주석을 보면 알겠지만 FILE_HEADER부분은 기초적인 정보가 포함되어 있다. 타겟 머신 타입은 말 그대로 어떤 환경에서 작동되는가에 대한 정보이며 그뒤로 파일 섹션수, 파일 제작일시(1970년 1월 1일 09시를 기준으로 현재까지의 시점을 초단위로 표시한다), 심볼 테이블이 나오고 이는 디버그용 실행 파일에 포함된 함수명 등의 심볼을 저장하는 영역이다, 그 뒤로 심볼 테이블의 엔트리 수, 옵션 헤더의 크기, 파일속성 플래그까지의 구조를 가진다.

파일속성 플래그에 조금 설명을 붙이자면 EXE파일인지 DLL파일인지, 2GB이상의 메모리를 다룰수 있는가처럼 실행 파일에 대한 다양한 정보를 가지고 있다.

2) IMAGE_OPTIONAL_HEADER

옵션 헤더에는 주로 Windows가 실행 파일을 메모리에 로드할 때 필요한 정보가 들어있다. IMAGE_OPTIONAL_HEADER 는 총 224바이트로 구성되어 있고 구조체는 96바이트를 차지하는 36개의 기본 필드와 8바이트 크기의 IMAGE_DATA_DIRECTORY 구조체에 대한 엔트리 개수가 16개인 배열 128(=8*16)바이트로 구성되어 있다.

```
typedef struct _IMAGE_OPTIONAL_HEADER {
    // Standard fields.                          // 기본필드
    WORD    Magic;                               // OPTIONAL 시그니처
    BYTE    MajorLinkerVersion;                  // 링커버전
    BYTE    MinorLinkerVersion;                  // 링커버전
    DWORD   SizeOfCode;                          // 코드 섹션들의 사이즈의 합
    DWORD   SizeOfInitializedData;               // 코드 섹션 제외 초기화된 사이즈
    DWORD   SizeOfUninitializedData;             // 초기화 안된 데이터 섹션의 크기
    DWORD   AddressOfEntryPoint;                 // 엔트리 포인트 RVA
    DWORD   BaseOfCode;                          // 코드 섹션 RVA
    DWORD   BaseOfData;                          // 데이터 섹션 RVA

    // NT additional fields.
    DWORD   ImageBase;                           // 이미지 베이스
    DWORD   SectionAlignment;                    // 메모리 페이지크기(4K)
    DWORD   FileAlignment;                       // 섹션들의 정렬 단위
    WORD    MajorOperatingSystemVersion;         // 운영체제 최소 버전
    WORD    MinorOperatingSystemVersion;         // 운영체제 최소 버전
    WORD    MajorImageVersion;                   // 유저 정의 파일 버전
    WORD    MinorImageVersion;                   // 유저 정의 파일 버전
    WORD    MajorSubsystemVersion;               // 서브시스템의 최소 버전
    WORD    MinorSubsystemVersion;               // 서브시스템의 최소 버전
    DWORD   Win32VersionValue;                   // (사용 ×)
    DWORD   SizeOfImage;                         // 메모리상의 PE 파일의 총 사이즈
    DWORD   SizeOfHeaders;                       // 디스크 상에서의 헤더의 총 사이즈
    DWORD   CheckSum;                            // 체크섬 값
    WORD    Subsystem;                           // 서브 시스템
    WORD    DllCharacteristics;                  // DLL 초기화 함수(현재 사용 ×)
    DWORD   SizeOfStackReserve;                  // 스택 예약 크기
    DWORD   SizeOfStackCommit;                   // 스택 초기 수용 크기
    DWORD   SizeOfHeapReserve;                   // 기본 힙 예약 크기
    DWORD   SizeOfHeapCommit;                    // 기본 힙 초기 수용 크기
    DWORD   LoaderFlags;                         // (사용 ×)
    DWORD   NumberOfRvaAndSizes;                 // DIRECTORY 배열의 원소 개수
    IMAGE_DATA_DIRECTORY
DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

<WinNT.H에 정의되 있는 IMAGE_OPTIONAL_HEADER 구조체>

옵션 헤더는 참 많은 구조를 가지고 있다. 단순히 주석을 보면 이해할수 있는부분은 넘어가고 설명을 하겠다.

우선 기본 필드부터 살펴 보겠다. Magic값을 살펴보자 이 값은 32비트 PE, 64비트 PE, ROM 이미지, .NET PE의 값들을 가지고 있는데 IMAGE_OPTION_HEADER을 나타나는 시그너처로 사용된다.

AddressOfEntryPoint값은 코드 섹션(.text)의 시작점을 가르킨다. 이 주소는 RVA(Relative Virtual Address)로 되어있으며 프로그램이 로드된 후 프로세스의 메인 스레드 문맥의 EIP 레지스터가 가질수 있는 최초의 값이다. 이 엔트리 포인트 뒤로 코드섹션과 데이터 섹션의 RVA값이 지정된다.

다음에 봐야 할부분은 ImageBase다 PE가 가상 주소 공간에 매핑될 때 매핑시키고자 하는 메모리 상의 시작주소이다. 위에 AddressOfEntryPoint 값이 가지는 RVA 즉 상대적 가상 주소는 이 ImageBase값의 상대적 주소가 되는 것이다. EXE파일은 0x400000번지, DLL은 0x10000000이다.

메모리 매핑시 각 섹션의 시작주소는 언제나 SectionAlignment 필드에서 지정된 값의 배수가 되는 가상주소가 된다. 인텔 기반 윈도우의 경우 메모리 페이지 크기가 4K이기 때문에 0x1000을 디폴트 값으로 가지고 있다.

FileAlignment는 섹션들의 정렬 단위를 나타낸다. 각 섹션을 구성하는 바이너리 데이터들은 이 필드값의 배수로 시작된다. NTFS 경우 한섹터는 디폴트로 4K이다. 보통은 0x200, 0x100을 디폴트 값으로 가지고 있다.

특별한 필드 외에는 그냥 주석만 읽어도 이해가 되리라 본다. 그럼 다음에 볼값은 Subsystem를 살펴보자 이값은 명령행 프로그램인지 GUI(Graphical User Interface)프로그램인지에 관해 명시하도록 되어 있다. 명령행 프로그램은 '0x0003', GUI 프로그램은 '0x0002', 디바이스 드라이버처럼 별도의 서브 시스템이 필요 없는 경우는 '0x0001'값을 가진다.

3) IMAGE_DATA_DIRECTORY

위에서 설명을 하지 않았지만 NumberOfRvaAndSizes은 IMAGE_DATA_DIRECTORY 구조체 배열의 원소 개수를 의미하고 항상 16의 값을 가진다. 즉 IMAGE_DATA_DIRECTORY 구조체는 16개의 배열 엔트리를 가지고 있다.(마지막 엔트리 값은 0으로 설정되어 있기 때문에 실질적으론 15개를 가진다.)

데이터 디렉토리는 Export table, Import table 등 PE 파일에서 중요한 역할을 하는 개체들의 주소와 크기에 대한 정보를 가지고 있으니 WinNT.h에 정의된 값들을 살펴보자.

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

// 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
#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
```

<WinNT.H에 정의되 있는 Directory Entries>

중요하다고 생각 되는부분은 EXPORT, IMPORT, BASERELOC, TLS 정도가 되겠으며 눈여겨 봐둬야 할 것이다.

· IMAGE_DIRECTORY_ENTRY_EXPORT : EXPORT 테이블의 메모리상에서의 시작점과 크기에 대한 정보를 가지고 있다.

· IMAGE_DIRECTORY_ENTRY_IMPORT : 위의 EXPORT와 마찬가지로 IMPORT 테이블의 메모리상 시작점과 크기에 대한 정보가 들어있다.

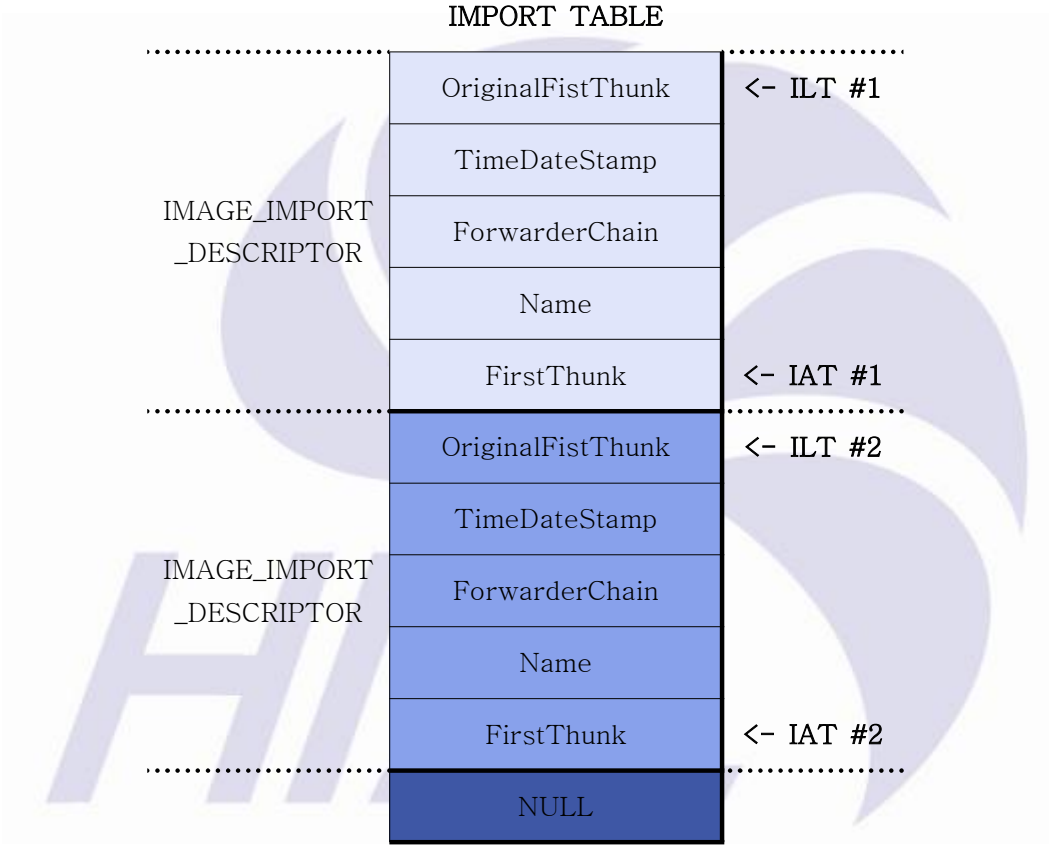
· IMAGE_DIRECTORY_ENTRY_BASERELOC : 기준 재배치(Base Relocation)정보를 가르킨다. ImageBase 필드에서 지정된 가상 주소 공간에 위치시키지 못했을 때 주소를 다시 갱신해 준다. 주로 DLL에서 재배치가 많이 이루어진다.

· IMAGE_DIRECTORY_ENTRY_TLS : 스레드 지역 저장소(Thread Local Storage) 초기화 섹션에 대한 포인터이다. 안티 리버싱 기법중 TLS Callback에서 사용되기도 한다.

5. Import Table

DLL파일은 함수를 Export 시키고 EXE파일이 Export된 함수를 Import 시킨다. PE파일의 섹션 테이블에서 .idata에는 사용하는 DLL에 대한 정보가 저장되어 있다.

Import Table의 각 IMAGE_IMPORT_DESCRIPTOR에는 임포트한 DLL 정보를 가지고 있으며 마지막을 알리기 위해 NULL값으로 채워져 있다.(Import Table의 개수는 Import한 DLL 개수 + 1)



그렇다면 IMAGE_IMPORT_DESCRIPTOR은 어떠한 구조를 가지고 있는지 WinNT.h에 정의되어있는 구조체를 살펴보자.

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD    Characteristics;
        DWORD    OriginalFirstThunk;
    };
    DWORD    TimeDateStamp;
    DWORD    ForwarderChain;
    DWORD    Name;
    DWORD    FirstThunk;
} IMAGE_IMPORT_DESCRIPTOR;
```

<WinNT.H에 정의되 있는 IMAGE_IMPORT_DESCRIPTOR 구조체>

· OriginalFirstThunk : ILT(Import Lookup Table)을 가리키는 RVA 값이다. ILT는 IMAGE_THUNK_DATA로 구성된 배열이다.

· Name : Import한 DLL의 이름을 가리키는 RVA 값

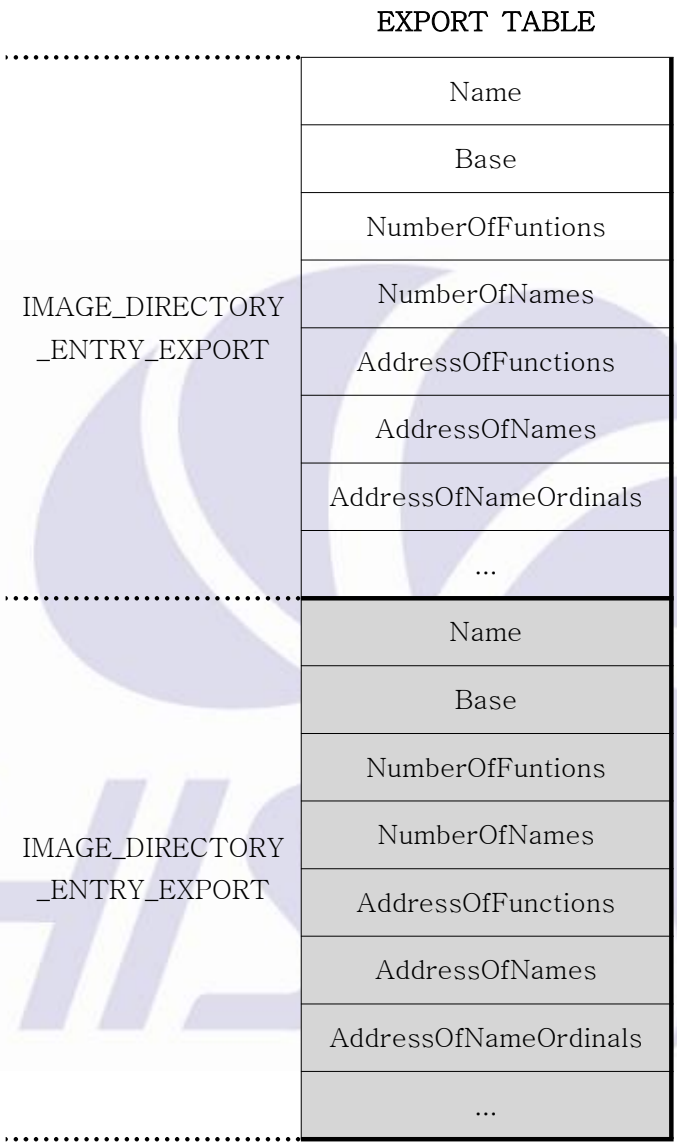
· FirstThunk : IAT(Import Address Table)의 RVA 주소값을 가지고 있다. IAT역시 ILT처럼 IMAGE_THUNK_DATA로 구성된 배열이다. 단 IAT같은경우는 Binding 전과 후 값이 달라지며 Binding 후에는 DLL의 Export한 함수의 실제 주소를 가르킨다.

```
typedef struct _IMAGE_THUNK_DATA32 {
    union {
        PBYTE  ForwarderString;
        PDWORD Function;
        DWORD Ordinal;
        PIMAGE_IMPORT_BY_NAME  AddressOfData;
    } u1;
} IMAGE_THUNK_DATA32;
```

<WinNT.H에 정의되 있는 IMAGE_THUNK_DATA 구조체>

6. Export Table

Import Table가 DLL로부터 함수목록을 가져온것이면 Export는 그 함수를 제공하기 위한 Table가 되겠다.



```
typedef struct _IMAGE_EXPORT_DIRECTORY {
    DWORD   Characteristics;
    DWORD   TimeDateStamp;
    WORD    MajorVersion;
    WORD    MinorVersion;
    DWORD   Name;
    DWORD   Base;
    DWORD   NumberOfFunctions;
    DWORD   NumberOfNames;
    DWORD   AddressOfFunctions;     // RVA from base of image
    DWORD   AddressOfNames;         // RVA from base of image
    DWORD   AddressOfNameOrdinals;  // RVA from base of image
} IMAGE_EXPORT_DIRECTORY, *PIMAGE_EXPORT_DIRECTORY;
```

<WinNT.H에 정의되 있는 IMAGE_THUNK_DATA 구조체>

· Name : DLL이름을 나타내는 ASCII문자열의 위치가 있는 RVA값

· Base : Export된 함수들에 대한 서수의 시작번호

· NumberOfFunction : AddressOfFunction가 가르키는 RVA 배열의 수

· NumberOfNames : AddressOfNames가 가르키는 RVA 배열의 수

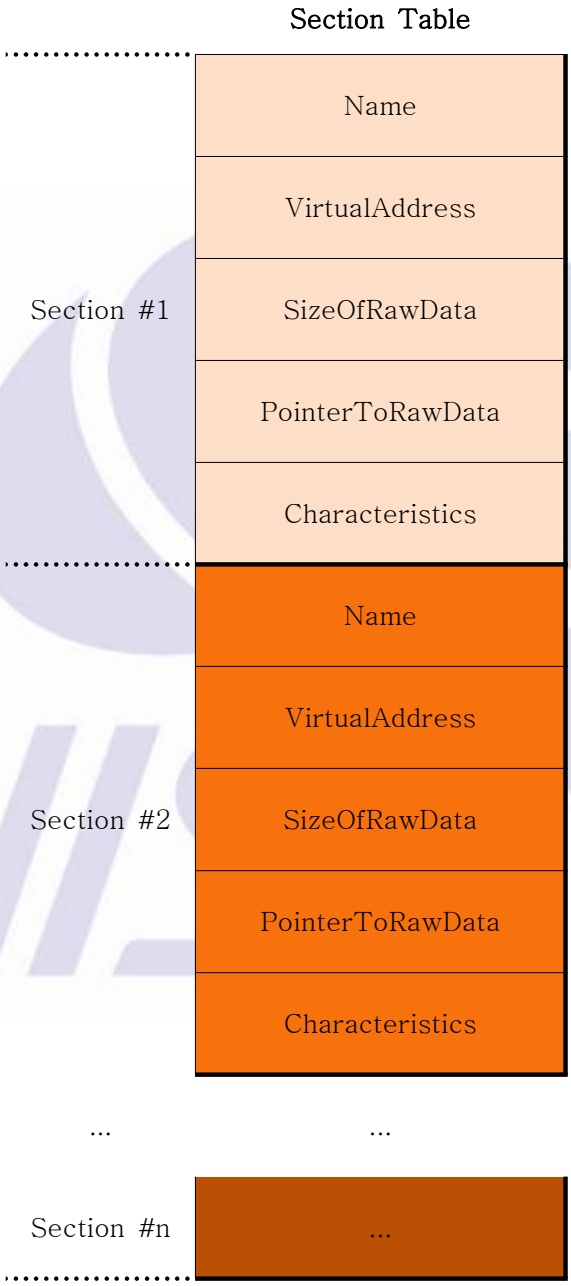
· AddressOfFunction : 함수의 실제 주소가 담긴 배열(RVA)

· AddressOfName : 함수의 심볼을 나타내는 문자열 배열(RVA)

· AddressOfNameOrdinals : 함수의 서수값의 배열 위치

7. IMAGE_SECTION_HEADER

PE 헤더 다음에 IMAGE_SECTION_HEADER의 배열이 온다. 40바이트의 크기를 가지고 있는 이 구조체는 각 SECTION의 정보를 담고 있는 테이블 형태로 구성되어 있다.



```
#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
```

<WinNT.H에 정의되 있는 IMAGE_SECTION_HEADER 구조체>

· Name : 섹션의 이름으로 8byte의 크기를 가지고 있다.

· VirtualAddress : 섹션이 로드될 가상 주소(RVA)

· SizeOfRawData : 파일상에서의 섹션의 사이즈(FileAlignment의 배수)

· PointerToRawData : 파일상에서의 섹션의 시작위치

· Characteristics : 섹션의 속성값

8. 맺음말

기술문서를 제작하면서 많이 부족한걸 느꼈습니다. 이것이 PE파일의 모든 것이라 생각하지 말아주셨으면 합니다. 다루지 못한 부분이 너무나 많기 때문입니다. 기회가 된다면 좀더 완성도가 높은 문서를 제작하고 싶습니다.

이 글을 읽는분에게 조금이나마 도움이 되었으면 하는 바램으로 이 글을 마칩니다. 제가 잘못 이해하고 있는부분이 있을수도 있으니 틀린 부분이 있으면 연락주시기 바랍니다.

◎ 참고 서적

1. Windows 시스템 실행파일의 구조와 원리 - 한빛미디어
2. API로 배우는 Windows의 구조 - 한빛미디어
3. 2008 KUCIS 교육 Reverse Engineering 교제

◎ 참고 사이트

1. http://zesrever.xstone.org/category/지식뽐뿌질%20II(연재물) - zesrever의 지식펌프
2. http://www.openrce.org/reference_library/files/reference/PE%20Format.pdf - OpenRCE