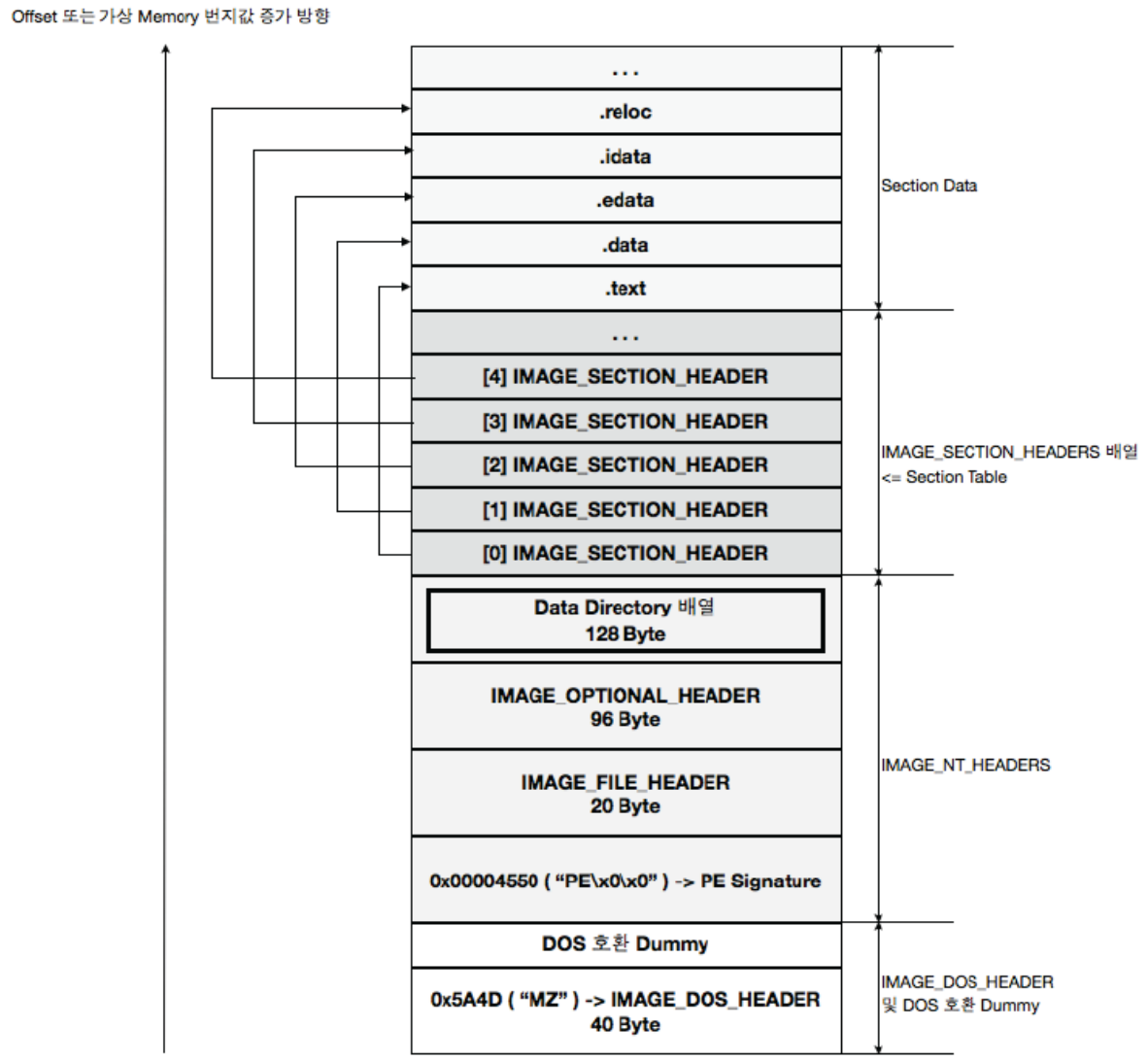
PE File Format

작성일	2011년 07월 06일
성명 or 아이디	koromoon
연락처	koromoon@naver.com

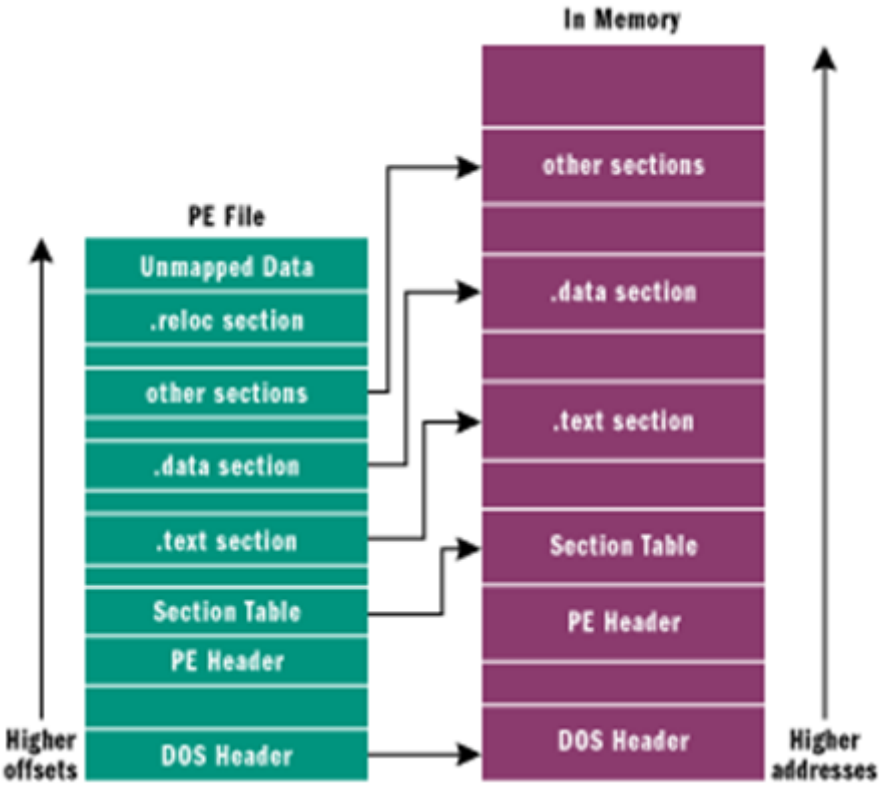
Sur3x5F

(1) PE File Format

근본적으로 윈도 OS 로더가 감춰진 실행 코드를 다루는 데 있어서 필수적인 정보를 은닉해 주는 자료구조
PE(Portable Excutable - 이식이 가능한 실행 형식이라는 이름을 붙임)
마이크로소프트의 윈도 3.1부터 지원되는 실행 파일의 형식
유닉스 COFF(Common Object file format)를 기반으로 나왔음.



[그림 - PE File Format의 전체 구조]



[그림 - 디스크상의 PE 구조와 메모리상의 PE 구조 비교]

PE File은 디스크에 파일 형태로 존재하지만 사실 그 내용은 메모리 상에서 돌아가는 프로세스의 이미지임. 즉, 메모리 상의 프로세스 이미지를 거의 그대로 파일 형태로 저장한 것이 PE File이라는 것임.
PE File의 전체적인 구조를 공부할 때는 "PE 파일은 디스크 상의 모습과 메모리 상의 모습이 거의 같다" 라는 것과 PE 파일의 구성 요소, 그리고 각 구성 요소의 시작점을 찾는 방법을 잘 알아두어야 함.

(2) PE File Format의 종류

실행 파일 계열 : EXE, SCR
라이브러리 계열 : DLL, OCX
드라이버 계열 : SYS
오브젝트 파일 계열 : OBJ (자체적으로 실행 X)

(3) PE File Format 이해를 위한 기본 지식

Sur3x5F

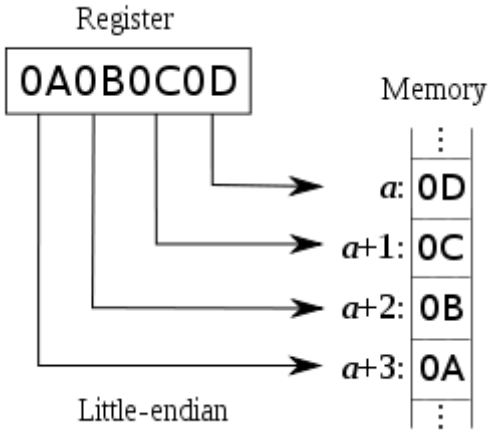
VA(Virtual Address)
프로세스 가상 메모리의 절대 주소

RVA(Relative Virtual Address)
상대적인 가상 주소를 말하며 메모리에 로드 시 기준이 되는 주소(Imagebase)로부터의 상대적인 주소
파일이 실행될 때 주소와 관련 있는 값(Entry Point, Import Table 등)들 RVA으로 표현
PE File(주로 DLL)이 프로세스 가상 메모리의 특정 위치로 로딩되는 순간 이미 그 위치에 다른 PE File(문맥상 DLL로 이해)이 로딩되어 있을 수 있음. 그럴 때 재배치(Relocation) 과정을 통해서 비어있는 다른 위치에 로딩되어야 하는데, 만약 PE Header 정보들이 VA(절대 주소)로 되어 있다면 정상적인 액세스가 이루어지지 않을 것임. 정보들이 RVA(상대 주소)로 되어 있다면 재배치(Relocation) 과정이 발생해도 기준 위치에 대한 상대 주소는 변하지 않기 때문에 아무런 문제없이 원하는 정보에 액세스할 수 있음.

VA와 RVA의 관계
RVA + ImageBase = VA

VAS(Virtual Address Space : 가상 주소 공간)와의 관계
프로세스는 자신의 구성원으로 4GB(64비트 시스템에서는 1024GB)의 가상 주소 공간을 가짐. 이 공간을 실제의 물리적인 기억 장치와 연결시켜 주는 것이 가상 메모리 관리자(Virtual Memory Manager)임. 여기서 말하는 물리적 기억장치는 RAM과 하드디스크 상의 특정 파일(pagefile.sys : 일명 페이징 파일)를 포함함.
페이징 파일(pagefile.sys)과 RAM 그리고 VAS는 가상 메모리 관리자에 의해 관리되면서 프로세스에서는 4GB의 선형 주소 공간을 가진 것처럼 착각하게 만듬. 실제로 존재하지는 않지만 프로세스에 속한 특정 스레드가 가상 주소 공간 내의 특정 번지를 액세스할 때는 가상 메모리 관리자를 통해 해당 번지의 페이지를 페이징 파일과 매핑시켜 줌.
가상 주소 공간의 특정 페이지에 접근한다는 것은 페이징 파일의 일부에 그 페이지가 매핑되어야 함을 의미함. 여기서 매핑은 가상 주소 공간과 페이징 파일 사이의 페이지 단위의 대응을 의미함.
하지만 PE File이 로드될 때 가상 메모리 관리자는 페이징 파일을 사용하지 않고 PE File 자체를 마치 페이징 파일처럼 가상 주소 공간에 그대로 매핑하게 됨. 이렇게 파일 자체가 페이징 파일의 역할을 대신하는 경우 메모리에 매핑된 파일(MMF : Memory Mapped File)이라고 하며 즉 PE File의 경우 MMF로 해당 파일을 연다고 볼 수 있음.
WinNT.H에 정의된 PE Header의 구조체를 보면 IMAGE라는 단어를 사용하고 있음. 여기서 "IMAGE"의 의미는 메모리에 매핑된 하드디스크 상의 PE File 자체를 말함.
정리하자면, PE 파일은 디스크 상의 모습과 메모리 상의 모습이 거의 같다.

Little Endian 표기법

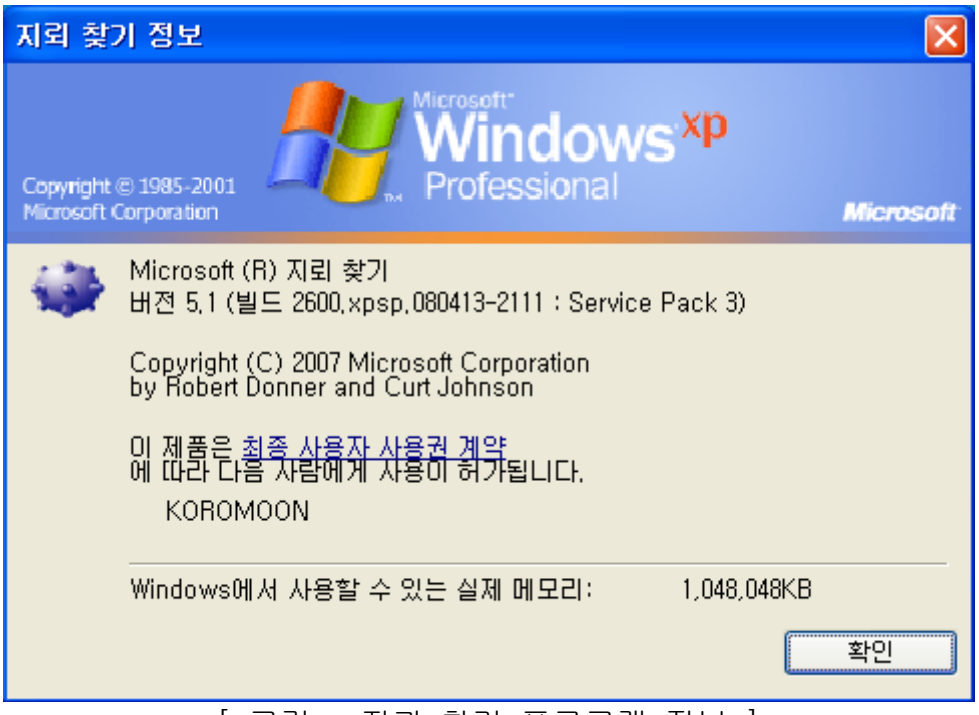


[그림 - Little Endian]

상위 바이트 값(0x0A)이 번지 수가 가장 큰 메모리(메모리 번지 : a + 3)상에 먼저 표시되는 방식

Intel계열 PC는 자료가 역순으로 저장함

(4) PE File Format 분석 준비



[그림 - 지뢰 찾기 프로그램 정보]

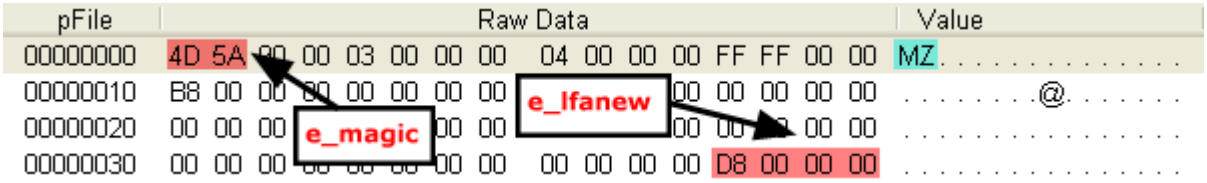
OS : Windows XP SP3
Program : 지뢰찾기 프로그램
분석 Program : PEview v0.9.9.0
기타 파일 : WinNT.h 파일 (PE File Format의 다양한 구조체 정보를 알아보기

위해서 필요함. 구글링으로 구해보자!)
학습 목표 : 지뢰찾기 프로그램에 대한 PE File Format을 분석해보자!

(5) IMAGE_DOS_HEADER

```
typedef struct _IMAGE_DOS_HEADER {
    WORD e_magic;          // IMAGE_DOS_Header 구조체의 Signature (MZ)
    WORD e_cblp;
    WORD e_cp;
    WORD e_crlc;
    WORD e_cparhdr;
    WORD e_minalloc;
    WORD e_maxalloc;
    WORD e_ss;
    WORD e_sp;
    WORD e_csum;
    WORD e_ip;
    WORD e_cs;
    WORD e_lfarlc;
    WORD e_ovno;
    WORD e_res[4];
    WORD e_oemid;
    WORD e_oeminfo;
    WORD e_res2[10];
    LONG e_lfanew;         // IMAGE_NT_HEADER 구조체의 시작 오프셋 값
} IMAGE_DOS_HEADER,*PIMAGE_DOS_HEADER;
```

[그림 - IMAGE_DOS_HEADER 구조체]



[그림 - PEview로 본 지뢰찾기 프로그램의 IMAGE_DOS_HEADER 구조체]

사이즈 : 64 바이트
e_masic과 e_lfanew만 기억하자!

e_masic	- IMAGE_DOS_Header 구조체의 Signature (MZ) - PE 로더가 이 값을 체크하여 맞다면 실행파일을 메모리에 로드하게 됨 - MZ는 Microsoft에서 DOS 실행파일을 설계한 Mark Zbikowski 라는 사람의 이니셜임
e_lfanew	- IMAGE_NT_HEADER 구조체의 시작 오프셋 값

(6) IMAGE_DOS_HEADER - DOS Stub Code

pFile	Raw Data	Value
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	E5 6A 12 49 A1 0B 7C 1A A1 0B 7C 1A A1 0B 7C 1A	.j.I..|...|...|.
00000090	A1 0B 7C 1A A0 0B 7C 1A 5B 28 3C 1A A0 0B 7C 1A	..|...|.[(<...|.
000000A0	A1 0B 7D 1A C9 0B 7C 1A 5B 28 65 1A B0 0B 7C 1A	..}...|.[(e...|.
000000B0	36 28 39 1A A0 0B 7C 1A 7B 28 60 1A B0 0B 7C 1A	6(9...|.{(`...|.
000000C0	5B 28 41 1A A0 0B 7C 1A 52 69 63 68 A1 0B 7C 1A	[(A...|.Rich..|.
000000D0	00 00 00 00 00 00 00 00	....

[그림 - PEview로 본 지뢰찾기 프로그램의 DOS Stub Code]

DOS Header 밑에 DOS Stub Code가 존재하며 크기가 일정치 않음.
OS가 PE File 형식을 알지 못할 때 실행되는 부분임.
("This program cannot be run in DOS mode." 라는 문자열이 출력하는 코드)
중요 요소가 아니므로 생략 가능함.

(7) IMAGE_NT_HEADERS

```
typedef struct _IMAGE_NT_HEADERS32 {        // 32비트
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32,*PIMAGE_NT_HEADERS32;
typedef struct _IMAGE_NT_HEADERS64 {        // 64비트
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER64 OptionalHeader;
} IMAGE_NT_HEADERS64,*PIMAGE_NT_HEADERS64;
```

[그림 - IMAGE_NT_HEADERS 구조체]

pFile	Raw Data	Value
000000D8	50 45 00 00 4C 01 03 00 75 84 7D 3B 00 00 00 00	PE..L...u.};....
000000E8	00 00 00 00 E0 00 0F 01 0B 01 07 00 00 3C 00 00	............<..
000000F8	00 94 01 00 00 00 00 00 21 3E 00 00 00 10 00 00	........!>......
00000108	00 50 00 00 00 00 00 01 00 10 00 00 00 02 00 00	.P..............
00000118	05 00 01 00 05 00 01 00 04 00 00 00 00 00 00 00	....
00000128	00 00 02 00 00 04 00 00 2D 06 02 00 02 00 00 80	........-.......
00000138	00 00 04 00 00 40 00 00 00 00 10 00 00 10 00 00	.....@..........
00000148	00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00	....
00000158	5C 41 00 00 B4 00 00 00 00 60 00 00 AC 8F 01 00	\A.....`......
00000168	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00000178	00 00 00 00 00 00 00 00 D0 11 00 00 1C 00 00 00	....
00000188	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00000198	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
000001A8	48 02 00 00 A8 00 00 00 00 10 00 00 B8 01 00 00	H...............
000001B8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
000001C8	00 00 00 00 00 00 00 00	....

[그림 - PEview로 본 지뢰찾기 프로그램의 IMAGE_NT_HEADERS 구조체]

Signature, IMAGE_FILE_HEADER 구조체, IMAGE_OPTION_HEADER 구조체로 구성

(8) IMAGE_NT_HEADERS - Signature

pFile	Raw Data	Value
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 D8 00 00 00	....
0000	4C CD 21 54 68	....!..L.!Th
0000	63 61 6E 6E 6F	is program canno
0000	20 44 4F 53 20	t be run in DOS
0000	00 00 00 00 00	mode....$.......
0000	1A A1 0B 7C 1A	.j.l..|...|...|.
00000090	A1 0B 7C 1A A0 0B 7C 1A 5B 28 3C 1A A0 0B 7C 1A	..|...|.[(<...|.
000000A0	A1 0B 7D 1A C9 0B 7C 1A 5B 28 65 1A B0 0B 7C 1A	..}...|.[(e...|.
000000B0	36 28 39 1A A0 0B 7C 1A 7B 28 60 1A B0 0B 7C 1A	6(9...|.{(`...|.
000000C0	5B 28 41 1A A0 0B 7C 1A 52 69 63 68 A1 0B 7C 1A	[(A...|.Rich..|.
000000D0	00 00 00 00 00 00 00 00 50 45 00 00 4C 01 03 00	........PE..L...
000000E0	75 84 7D 3B 00 00 00 00 00 00 00 00 E0 00 0F 01	u.};............

IMAGE_DOS_HEADER 구조체의 e_lfanew 필드 값은 IMAGE_NT_HEADERS 구조체의 Signature 필드 위치를 가리킴 (PE\00\00)

[그림 - IMAGE_NT_HEADERS 구조체의 Signature 필드 위치]

PE File 임을 나타내는 매직넘버
사이즈 : 4 바이트
항상 "PEWx00Wx00" (Hex -> 50 45 00 00)
PE File 임을 확인하기 위해서는 IMAGE_DOS_HEADER 구조체의 e_lfanew 필드가 가

리키는 오프셋만큼 파일 포인터를 이동시켜 그 오프셋부터 4바이트를 DWORD 형으로 읽어들여 그 값을 IMAGE_NT_HEADERS 구조체의 Signature 매크로 상수와 비교하면 됨.

(9) IMAGE_NT_HEADERS - IMAGE_FILE_HEADER

```
typedef struct _IMAGE_FILE_HEADER {
    WORD Machine;                   // CPU ID
    WORD NumberOfSections;          // 섹션 수
    DWORD TimeDateStamp;            // 파일을 만들어낸 시간
    DWORD PointerToSymbolTable;     // COFF 심벌의 파일 오프셋
    DWORD NumberOfSymbols;          // PointerToSymbolTable 필드가 가리키는
                                    // COFF 심벌 테이블 내에서의 심벌의 수
    WORD SizeOfOptionalHeader;      // IMAGE_OPTIONAL_HEADER 구조체의 바이트 수
    WORD Characteristics;           // 해당 PE File에 대한 특정 정보를 나타내는 플래그
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

[그림 - IMAGE_FILE_HEADER 구조체]

pFile	Data	Description	Value
000000DC	014C	Machine	IMAGE_FILE_MACHINE_I386
000000DE	0003	Number of Sections	
000000E0	3B7D8475	Time Date Stamp	2001/08/17 20:54:13 UTC
000000E4	00000000	Pointer to Symbol Table	
000000E8	00000000	Number of Symbols	
000000EC	00E0	Size of Optional Header	
000000EE	010F	Characteristics	
		0001	IMAGE_FILE_RELOCS_STRIPPED
		0002	IMAGE_FILE_EXECUTABLE_IMAGE
		0004	IMAGE_FILE_LINE_NUMS_STRIPPED
		0008	IMAGE_FILE_LOCAL_SYMS_STRIPPED
		0100	IMAGE_FILE_32BIT_MACHINE

[그림 - PEview로 본 지뢰찾기 프로그램의 IMAGE_FILE_HEADER 구조체]

사이즈 : 20 바이트

해당 필드들이 주요 필드이므로 전부 다 기억해두자!

Machine	- CPU ID - WinNT.H 파일에서 "#define IMAGE_FILE_MACHINE" 이름으로 검색 - 32비트 Intel x86 계열 : 0x014c 64비트 Intel x86 계열 : 0x0200 64비트 AMD 계열 : 0x8664
NumberOfSections	- 섹션의 수 - 정의된 섹션 개수보다 실제 섹션이 적다면 : 에러 정의된 섹션 개수보다 실제 섹션이 많다면 : 정의된

	개수만큼만 인식
TimeDateStamp	- 파일을 만들어낸 시간 (OBJ 파일 - 컴파일러, EXE & DLL - 링커) - TimeStomp 툴을 이용하여 시간 변경 가능
PointerToSymbolTable	- COFF 심벌의 파일 오프셋
NumberOfSymbols	- PointerToSymbolTable 필드가 가리키는 COFF 심벌 테이블 내에서의 심벌의 수
SizeOfOptionalHeader	- IMAGE_OPTIONAL_HEADER 구조체의 바이트 수 - OBJ 파일의 경우 0 - 32비트 PE File일 경우 : 0xE0 (224 바이트) 64비트 PE File일 경우 : 0xF0 (240 바이트)
Characteristics	- 해당 PE File에 대한 특정 정보를 나타내는 플래그 - 논리합(OR)으로 구성

(10) IMAGE_NT_HEADERS - IMAGE_FILE_HEADER - Characteristics 필드에 대한 주요 매크로

```
#define IMAGE_FILE_RELOCS_STRIPPED  1               // 현재 파일에 재배치 정보가 없음
#define IMAGE_FILE_EXECUTABLE_IMAGE 2               // 실행 파일 이미지를 나타냄
                                                    // (OBJ나 LIB 파일이 아님)
#define IMAGE_FILE_LINE_NUMS_STRIPPED   4           // 라인 정보가 없음
#define IMAGE_FILE_LOCAL_SYMS_STRIPPED  8           // 로컬 심벌이 없음
#define IMAGE_FILE_AGGRESIVE_WS_TRIM    16          // OS로 하여금 적극적으로 워킹셋을
                                                    // 정리할 수 있도록 함
#define IMAGE_FILE_LARGE_ADDRESS_AWARE  32          // 어플리케이션이 2G 이상의
                                                    // 가상 주소 번지 제어가능
#define IMAGE_FILE_BYTES_REVERSED_LO    128
#define IMAGE_FILE_32BIT_MACHINE    256             // 32비트 워드 머신을 필요로 함
#define IMAGE_FILE_DEBUG_STRIPPED   512             // 디버그 정보가 본 파일에는 없고
                                                    // .DBG 파일에 존재
#define IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP  1024    // PE 이미지가 이동 가능한 장치 위에
                                                    // 존재할 때 고정 디스크 상의 스왑
                                                    // 파일로 복사해서 실행함
#define IMAGE_FILE_NET_RUN_FROM_SWAP    2048        // PE 이미지가 네트워크 상에
                                                    // 존재할 때 고정 디스크 상의 스왑
                                                    // 파일로 복사해서 실행함
#define IMAGE_FILE_SYSTEM   4096                    // 시스템 파일
#define IMAGE_FILE_DLL  8192                        // 동적 링크 라이브러리 파일
#define IMAGE_FILE_UP_SYSTEM_ONLY   16384           // 하나의 프로세서만을 장착한
                                                    // 머신에서 실행됨
#define IMAGE_FILE_BYTES_REVERSED_HI    32768
```

[그림 - Characteristics 필드에 대한 주요 매크로]

(11) IMAGE_OPTIONAL_HEADER

```
} IMAGE_DATA_DIRECTORY,*PIMAGE_DATA_DIRECTORY;
typedef struct _IMAGE_OPTIONAL_HEADER32 {
    WORD Magic;                         // IMAGE_OPTIONAL_HEADER 구조체의 Signature
    BYTE MajorLinkerVersion;            // 본 파일을 만들어낸 링커의 버전
    BYTE MinorLinkerVersion;            // 본 파일을 만들어낸 링커의 버전
    DWORD SizeOfCode;                   // 모든 코드 섹션들의 사이즈를 합한 라운드업된 크기
    DWORD SizeOfInitializedData;        // 코드 섹션을 제외한 초기화된 데이터 섹션의 전체 크기
    DWORD SizeOfUninitializedData;      // 초기화되지 않은 데이터 섹션의 바이트 수
    DWORD AddressOfEntryPoint;          // 로더가 실행을 개시할 주소 (RVA)
    DWORD BaseOfCode;                   // 첫 번째 코드 섹션이 시작되는 RVA (보통 0x1000)
    DWORD BaseOfData;                   // 메모리에 로드될 때 데이터의 첫 번째 바이트의 RVA
    DWORD ImageBase;                    // 해당 PE File이 가상 주소 공간에 매핑될 때
                                        // 매핑시키고자 하는 메모리 상의 시작 주소
                                        // EXE : 0x00400000, DLL : 0x10000000, 참고로 변경 가능
    DWORD SectionAlignment;             // PE File이 메모리에 매핑될 때 각 섹션의 시작 주소는
                                        // 언제나 이 SectionAlignment 필드에서 지정된 값의
                                        // 배수가 되는 가상 주소가 되도록 보장
    DWORD FileAlignment;                // PE File 내에서 섹션들의 정렬 단위 (디스크 섹터 단위)
    WORD MajorOperatingSystemVersion;   // PE FILE을 실행하는 데 필요한 운영체제의 최대 버전
    WORD MinorOperatingSystemVersion;   // PE FILE을 실행하는 데 필요한 운영체제의 최소 버전
    WORD MajorImageVersion;             // 유저가 정의 가능한 필드
    WORD MinorImageVersion;             // 유저가 정의 가능한 필드
    WORD MajorSubsystemVersion;         // PE File을 실행하는 데 필요한 서브시스템의 최대 버전
    WORD MinorSubsystemVersion;         // PE File을 실행하는 데 필요한 서브시스템의 최소 버전
    DWORD Win32VersionValue;            // 거의 사용 X (보통 0)
    DWORD SizeOfImage;                  // ImageBase 필드가 가리키는 주소 값으로부터
                                        // 이 PE File의 마지막 섹션의 끝까지의 크기
                                        // SectionAlignment 필드 값의 배수
    DWORD SizeOfHeaders;                // MS-DOS 헤더, PE 헤더, 그리고 섹션 테이블들의 크기를
                                        // 모두 합친 바이트 수
                                        // 반드시 FileAlignment 필드 값의 배수
    DWORD CheckSum;                     // 이미지의 체크섬 값
    WORD Subsystem;                     // 본 실행 파일이 유저 인터페이스로 사용하는
                                        // 서브 시스템의 종류
    WORD DllCharacteristics;
    DWORD SizeOfStackReserve;           // 프로세스 생성 및 잠입 시에 스택 예약 크기
    DWORD SizeOfStackCommit;            // 프로세스 생성 및 잠입 시에 스택 커밋 크기
    DWORD SizeOfHeapReserve;            // 프로세스 생성 및 잠입 시에 힙 예약 크기
    DWORD SizeOfHeapCommit;             // 프로세스 생성 및 잠입 시에 힙 커밋 크기
    DWORD LoaderFlags;                  // 보통 0 (디버킹 지원과 관련)
    DWORD NumberOfRvaAndSizes;          // IMAGE_DATA_DIRECTORY 구조체 배열의 원소 개수
                                        // 항상 16개 (0x00000010)1
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER32,*PIMAGE_OPTIONAL_HEADER32;
```

[그림 - IMAGE_OPTIONAL_HEADER 구조체 (32비트용)]

pFile	Data	Description	Value
000000F0	010B	Magic	IMAGE_NT_OPTIONAL_HDR32_MAGIC
000000F2	07	Major Linker Version	
000000F3	00	Minor Linker Version	
000000F4	00003C00	Size of Code	
000000F8	00019400	Size of Initialized Data	
000000FC	00000000	Size of Uninitialized Data	
00000100	00003E21	Address of Entry Point	
00000104	00001000	Base of Code	
00000108	00005000	Base of Data	
0000010C	01000000	Image Base	
00000110	00001000	Section Alignment	
00000114	00000200	File Alignment	
00000118	0005	Major O/S Version	
0000011A	0001	Minor O/S Version	
0000011C	0005	Major Image Version	
0000011E	0001	Minor Image Version	
00000120	0004	Major Subsystem Version	
00000122	0000	Minor Subsystem Version	
00000124	00000000	Win32 Version Value	
00000128	00020000	Size of Image	
0000012C	00000400	Size of Headers	
00000130	0002062D	Checksum	
00000134	0002	Subsystem	IMAGE_SUBSYSTEM_WINDOWS_GUI
00000136	8000	DLL Characteristics	
	8000		IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER_AWARE
00000138	00040000	Size of Stack Reserve	
0000013C	00004000	Size of Stack Commit	
00000140	00100000	Size of Heap Reserve	
00000144	00001000	Size of Heap Commit	
00000148	00000000	Loader Flags	
0000014C	00000010	Number of Data Directories	

[그림 - PEview로 본 지뢰찾기 프로그램의 IMAGE_OPTIONAL_HEADER 구조체]

32비트 PE File일 경우 사이즈 : 224 바이트
64비트 PE File일 경우 사이즈 : 240 바이트 (첨부 자료 생략함)
참고로 IMAGE_ROM_OPTIONAL_HEADER 구조체에 대한 첨부 자료 생략했음

Magic	- IMAGE_OPTIONAL_HEADER 구조체의 Signature - 32비트 PE File : 0x010B 64비트 PE File : 0x020B ROM 이미지 : 0x0107 .NET PE File : 0x010B
MajorLinkerVersion	- 본 파일을 만들어낸 링커의 버전
MinorLinkerVersion	
SizeOfCode	- 모든 코드 섹션들의 사이즈를 합한 라운드업된 크기 - 섹션 중 IMAGE_SCN_CNT_CODE 속성을 가진 섹션들의 전체 크기
SizeOfInifializedData	- 코드 섹션을 제외한 초기화된 데이터 섹션의 전체 크기 - 참고로 초기화된 데이터 섹션은

	IMAGE_SCN_CNT_INITIALIZED_DATA 속성을 가짐
SizeOfUninitializedData	- 초기화되지 않은 데이터 섹션(일반적으로 .bss 섹션 또는 .textbss 섹션)의 바이트 수 - 참고로 초기화되지 않은 데이터 섹션은 IMAGE_SCN_CNT_UNINITIALIZED_DATA 속성을 가짐 - 일반적으로 링커는 초기화되지 않은 데이터를 일반 데이터 섹션에 병합시킬 수 있기 때문에 이 필드는 0임
AddressOfEntryPoint	- 로더가 실행을 개시할 주소 (RVA) - 프로그램이 처음으로 실행될 코드를 담고 있는 주소
BaseOfCode	- 첫 번째 코드 섹션이 시작되는 RVA - 코드 섹션은 전형적으로 PE 헤더 다음에, 그리고 데이터 섹션 바로 직전에 옴 - 마이크로소프트 링커가 만들어내는 EXE의 RVA는 보통 0x1000
BaseOfData	- 이론적으론 메모리에 로드될 때 데이터의 첫 번째 바이트의 RVA - MS의 링커 버전에 따라 서로 다름 - 64비트 PE File에서는 이 필드가 없음
ImageBase	- 해당 PE File이 가상 주소 공간에 매핑될 때 매핑시키고자 하는 메모리 상의 시작 주소 - EXE : 0x00400000 DLL : 0x10000000 - 링크 시 옵션 /BASE를 지정함으로써 변경 가능
SectionAlignment	- PE File이 메모리에 매핑될 때 각 섹션의 시작 주소는 언제나 이 SectionAlignment 필드에서 지정된 값의 배수가 되는 가상 주소가 되도록 보장 - 디폴트 값 : 인텔 기반 윈도우의 경우 메모리 페이지 크기가 4K이기 때문에 0x100 - FileAlignment 필드 값보다 크거나 같아야 함
FileAlignment	- PE File 내에서 섹션들의 정렬 단위 - 디스크 섹터 단위 - NTFS 경우 한 섹터는 디폴트로 4K이므로 보통 0x200 또는 0x100 - SectionAlignment 필드 값보다 커서는 안 됨
MajorOperatingSystemVersion	- PE FILE을 실행하는 데 필요한 운영체제의 최대 버전
MinorOperatingSystemVersion	- PE FILE을 실행하는 데 필요한 운영체제의 최소 버전

MajorImageVersion	- 유저가 정의 가능한 필드 - 링킹 시에 /VERSION 옵션을 사용해서 링크를 해주면 됨
MinorImageVersion	
MajorSubSystemVersion	- PE File을 실행하는 데 필요한 서브시스템의 최대 버전
MinorSubsystemVersion	- PE File을 실행하는 데 필요한 서브시스템의 최소 버전
Win32VersionValue	- 거의 사용 X (보통 0)
SizeOfImage	- ImageBase 필드가 가리키는 주소 값으로부터 이 PE File의 마지막 섹션의 끝까지의 크기 (로더가 해당 PE File을 메모리 상에 로드할 때 확보 및 예약해야 할 해당 PE File을 위한 충분한 크기) - 보통 PE File의 크기보다 큼 - 반드시 SectionAlignment 필드 값의 배수
SizeOfHeaders	- MS-DOS 헤더, PE 헤더, 그리고 섹션 테이블들의 크기를 모두 합친 바이트 수 - 반드시 FileAlignment 필드 값의 배수
CheckSum	- 이미지의 체크섬 값 - 본 PE File의 체크섬 값은 IMAGEHELP.DLL의 CheckSumMappedFile API를 통해서 얻을 수 있음 - 체크섬 값은 커널 모드 드라이버나 어떤 시스템 DLL의 경우 요구됨 (그 이외의 경우라면 보통 0)
Subsystem	- 본 실행 파일이 유저 인터페이스로 사용하는 서브시스템의 종류
DllCharacteristics	- 본 PE File이 DLL 파일이라는 전제 하에 어떤 상황에서 DLL 초기화 함수가 호출되어야 하는지를 지시하는 프래그로 추정
SizeOfStackReserve	- 프로세스 생성 및 잠입 시에 스택 예약 크기
SizeOfStackCommit	- 프로세스 생성 및 잠입 시에 스택 커밋 크기
SizeOfHeapReserve	- 프로세스 생성 및 잠입 시에 힙 예약 크기
SizeOfHeapCommit	- 프로세스 생성 및 잠입 시에 힙 커밋 크기
LoaderFlags	- 보통 0 (디버깅 지원과 관련)
NumberOfRvaAndSizes	- IMAGE_DATA_DIRECTORY 구조체 배열의 원소 개수 - 항상 16개 (0x00000010)

※ 음영이 어두운 필드들은 주요 필드이며 기억해두자!

(12) IMAGE_OPTIONAL_HEADER - IMAGE_DATA_DIRECTORY

00000150	00000000	RVA	EXPORT Table
00000154	00000000	Size	
00000158	0000415C	RVA	IMPORT Table
0000015C	000000B4	Size	
00000160	00006000	RVA	RESOURCE Table
00000164	00018FAC	Size	
00000168	00000000	RVA	EXCEPTION Table
0000016C	00000000	Size	
00000170	00000000	Offset	CERTIFICATE Table
00000174	00000000	Size	
00000178	00000000	RVA	BASE RELOCATION Table
0000017C	00000000	Size	
00000180	000011D0	RVA	DEBUG Directory
00000184	0000001C	Size	
00000188	00000000	RVA	Architecture Specific Data
0000018C	00000000	Size	
00000190	00000000	RVA	GLOBAL POINTER Register
00000194	00000000	Size	
00000198	00000000	RVA	TLS Table
0000019C	00000000	Size	
000001A0	00000000	RVA	LOAD CONFIGURATION Table
000001A4	00000000	Size	
000001A8	00000248	RVA	BOUND IMPORT Table
000001AC	000000A8	Size	
000001B0	00001000	RVA	IMPORT Address Table
000001B4	000001B8	Size	
000001B8	00000000	RVA	DELAY IMPORT Descriptors
000001BC	00000000	Size	
000001C0	00000000	RVA	CLI Header
000001C4	00000000	Size	
000001C8	00000000	RVA	
000001CC	00000000	Size	

[그림 - PEview에서 본 지뢰찾기 프로그램의 IMAGE_DATA_DIRECTORY 구조체 배열]

사이즈 : 128 바이트

IMAGE_DATA_DIRECTORY 구조체 배열과 관련 있는 IMAGE_OPTIONAL_HEADER 구조체의 NumberOfRvaAndSizes 필드 값은 0x00000010 이어야 함.

IMAGE_DATA_DIRECTORY 구조체 배열의 마지막 엔트리, 즉 인덱스 15에 해당하는 엔트리는 언제나 0으로 세팅됨.
(주의! 배열의 인덱스 0 ~ NumberOfRvaAndSizes-1 까지의 인덱스가 지정하는 필드가 모두 사용되는 것이 아니라 15개의 엔트리가 사용된다는 점)
각 엔트리가 나름대로의 의미를 가지고 있기 때문에 반드시 참조해야 하는 필드임. (특히 병합된 섹션과 관련된 정보가 해당 엔트리에 들어가기 때문)

VirtualAddress - IMAGE_DATA_DIRECTORY_ENTRY의 인덱스에 해당하는 미리 지정된 섹션 또는 블록의 정보에 대한 시작 주소
Size - IMAGE_DATA_DIRECTORY_ENTRY의 인덱스에 해당하는 미리 지정된 섹션 또는 블록의 정보에 대한 크기

IMAGE_DIRECTORY_ENTRY_EXPORT	- IMAGE_EXPORT_DIRECTORY 구조체의 시작 번지 - 익스포트(내보내기) 섹션의 시작 주소
IMAGE_DIRECTORY_ENTRY_IMPORT	- IMAGE_IMPORT_DESCRIPTOR 구조체 배열의 번지 - 임포트(가져오기) 섹션의 시작 주소
IMAGE_DIRECTORY_ENTRY_RESOURCE	- IMAGE_RESOURCE_DIRECTORY 구조체의 시작 번지 - 리소스 섹션의 시작 주소
IMAGE_DIRECTORY_ENTRY_EXCEPTION	- IMAGE_RUNTIME_FUNCTION_ENTRY 구조체 배열의 시작 번지 - x86 계열에서는 의미가 없음
IMAGE_DIRECTORY_ENTRY_SECURITY	- WIN_CERTIFICATE 구조체들의 리스트의 시작 번지 - 해당 리스트는 메모리 상에 매핑되지 않기 때문에 VirtualAddress 필드 값은 RVA가 아니라 파일 오프셋에 해당
IMAGE_DIRECTORY_ENTRY_BASERELOC	- 기준 재배치 정보를 가리킴 - 재배치란 로더가 실행 모듈을 원하는 위치, 즉 IMAGE_OPTION_HEADER의 ImageBase 필드에 지정된 가상 주소 공간의 주소에 위치시키지 못했을 때 코드 상의 포인터 연산과 관련된 주소를 다시 갱신해야 하는 경우를 말함. (EXE 경우 거의 없지만 DLL 경우 빈번하게 발생)
IMAGE_DIRECTORY_ENTRY_DEBUG	- IMAGE_DEBUG_DIRECTORY 구조체의 배열을 가리키는 번지
IMAGE_DIRECTORY_ENTRY_ARCHITECTURE	- IMAGE_ARCHITECTURE_HEADER 구조체의 배열에 대한 포인터 - x86 또는 IA-64 계열에서는 사용하지 않음
IMAGE_DIRECTORY_ENTRY_GLOBALPTR	- IMAGE_DATA_DIRECTORY 엔트리의 VirtualAddress 필드는 글로벌 포인터(GP)로 사용되는 RVA - x86 계열 : 사용 X IA-64 계열 : 사용 - Size 필드 : 사용 X
IMAGE_DIRECTORY_ENTRY_TLS	- 스레드 지역 저정소(TLS) 초기화 섹션에 대한 포인터

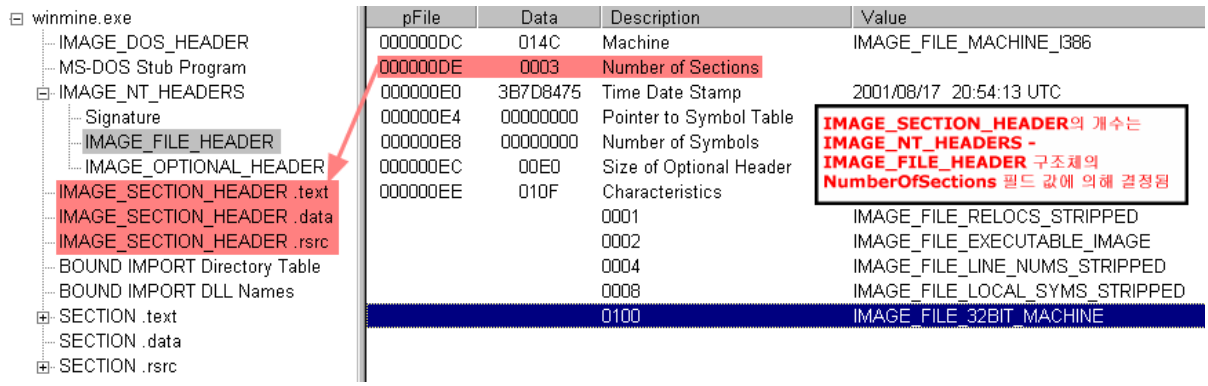
IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG	- IMAGE_LOAD_CONFIG_DIRECTORY 구조체에 대한 포인터
IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT	- IMAGE_BOUND_IMPORT_DESCRIPTOR 구조체의 배열을 가리키는 포인터 - DLL 바인딩과 관련된 정보
IMAGE_DIRECTORY_ENTRY_IAT	- 첫 번째 임포트 주소 테이블(IAT)의 시작 번지 - 임포트된 각각의 DLL에 대한 IAT는 메모리 상에서 연속적으로 나타남 - Size 필드 : 모든 IAT의 전체 크기
IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT	- ImgDelayDescr 구조체의 배열을 가리키는 지연 로딩 정보에 대한 포인터
IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR	- 최근 IMAGE_DIRECTORY_ENTRY_COM_HEADER라는 이름으로 변경 - .NET 응용 어플리케이션이나 DLL용 PE File을 위한 것으로 PE File 내의 .NET 정보에 대한 최상위 정보의 시작 번지를 가리킴 (IMAGE_COR20_HEADER)

(13) IMAGE_SECTION_HEADER

```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE Name[IMAGE_SIZEOF_SHORT_NAME]; // 섹션의 아스크 이름
    union {                             // OBJ 파일의 경우와 PE File의 경우에 따라 달라짐
        DWORD PhysicalAddress;          // PE File : 공용체의 VirtualSize 필드를 사용하며
                                        // 코드와 데이터의 실제 바이트 수를 담음
        DWORD VirtualSize;              // OBJ 파일 : 0
    } Misc;
    DWORD VirtualAddress;               // 메모리 상에서의 본 섹션의 시작 주소
    DWORD SizeOfRawData;                // Mics. VirtualSize 필드 값에 대한 파일 정렬
                                        // (Alignment) 값의 배수로 라운드업된 값
    DWORD PointerToRawData;             // 해당 섹션의 PE File 상에서 시작하는 실제 파일
                                        // 오프셋 값
    DWORD PointerToRelocations;         // 본 섹션을 위한 재배치 파일 오프셋
    DWORD PointerToLinenumbers;         // 본 섹션을 위한 COFF 스타일의 라인 번호를 위한
                                        // 파일 오프셋
    WORD NumberOfRelocations;           // PointerToRelocations 필드가 가리키는 IMAGE_
                                        // RELOCATION 구조체 배열의 원소의 개수
    WORD NumberOfLinenumbers;           // PointerToLinernumbers 필드가 가리키는 IMAGE_
                                        // LINENUMBER 구조체 배열의 원소의 개수
    DWORD Characteristics;              // 해당 섹션의 속성을 나타내는 플래그의 집합
} IMAGE_SECTION_HEADER,*PIMAGE_SECTION_HEADER;
```

[그림 - IMAGE_SECTION_HEADER 구조체]

사이즈 : 40 바이트



[그림 - PEview로 본 지뢰찾기 프로그램의 IMAGE_SECTION_HEADER의 개수 확인]

IMAGE_SECTION_HEADER의 개수는 IMAGE_NT_HEADERS - IMAGE_FILE_HEADER 구조체의 NumberOfSections 필드 값에 의해 결정됨

Name	- 섹션의 아스크 이름 - 8 바이트 크기의 이름 값이 저장됨 (NULL 제외) - 8 바이트 이상으로 지정했을 경우 링커는 8바이트 이후의 문자열은 잘라버린 후 이 필드에 값을 채움
PhysicalAddress or VirtualSize	- OBJ 파일의 경우와 PE File의 경우에 따라 달라짐 - PE File : 공용체의 VirtualSize 필드를 사용하며 코드와 데이터의 실제 바이트 수를 담음 - OBJ 파일 : 0
VirtualAddress	- 메모리 상에서의 본 섹션의 시작 주소 (매핑시켜야 할 가상 주소 공간 상의 RVA)
SizeOfRawData	- Mics. VirtualSize 필드 값에 대한 파일 정렬 (Alignment) 값의 배수로 라운드업된 값 - OBJ 파일 : 0
PointerToRawData	- 해당 섹션의 PE File 상에서 시작하는 실제 파일 오프셋 값 (IMAGE_OPTIONAL_HEADER 구조체의 FileAlignment 필드 값의 배수)
PointerToRelocations	- 본 섹션을 위한 재배치 파일 오프셋 (OBJ 파일에서만 사용, PE File일 경우 0)
PointerToLinenumbers	- 본 섹션을 위한 COFF 스타일의 라인 번호를 위한 파일 오프셋 - 0이 아닐 경우 IMAGE_LINENUMBER 구조체 배열의 시작을 나타냄 - COFF 라인 번호가 PE File에 첨부되었을 경우에만 사용
NumberOfRelocations	- PointerToRelocations 필드가 가리키는 IMAGE_RELOCATION 구조체 배열의 원소의 개수 - PE File : 0

PointerToLinenumbers	- PointerToLinernumbers 필드가 가리키는 IMAGE_LINENUMBER 구조체 배열의 원소의 개수 - COFF 라인 번호가 PE File에 첨부되었을 경우에만 사용
Characteristics	- 해당 섹션의 속성을 나타내는 플래그의 집합 - WinNT.h 파일에 IMAGE_SCN_xxx_xxx의 형태로 #define 문에 의해 정의됨 - 논리합(OR)으로 구성

※ 음영이 어두운 필드들은 주요 필드이며 기억해두자!

(14) IMAGE_SECTION_HEADER - Charcteristics 필드의 주요 플래그

WinNT.h 파일에 IMAGE_SCN_xxx_xxx의 형태로 #define 문에 의해 정의됨

```
#define IMAGE_SCN_TYPE_REG 0
#define IMAGE_SCN_TYPE_DSECT 1
//#define IMAGE_SCN_TYPE_NOLOAD 2
#define IMAGE_SCN_TYPE_GROUP 4
#define IMAGE_SCN_TYPE_NO_PAD 8
#define IMAGE_SCN_CNT_CODE 32                   // 실행 가능 플래그
                                                // IMAGE_SCN_MEM_EXECUTE 플래그와 함께 지정
#define IMAGE_SCN_CNT_INITIALIZED_DATA 64       // 섹션이 초기화된 데이터
                                                // 실행 가능 섹션과 .bss 섹션을 제외한
                                                // 거의 대부분의 섹션이 이 플래그를 가짐
#define IMAGE_SCN_CNT_UNINITIALIZED_DATA 128    // 섹션이 초기화되지 않은 데이터
#define IMAGE_SCN_LNK_OTHER 256
#define IMAGE_SCN_LNK_INFO 512
#define IMAGE_SCN_LNK_REMOVE 2048
#define IMAGE_SCN_LNK_COMDAT 4096
#define IMAGE_SCN_MEM_FARDATA 0x8000
#define IMAGE_SCN_MEM_PURGEABLE 0x20000
#define IMAGE_SCN_MEM_16BIT 0x20000
#define IMAGE_SCN_MEM_LOCKED  0x40000
#define IMAGE_SCN_MEM_PRELOAD 0x80000
#define IMAGE_SCN_ALIGN_1BYTES 0x100000
#define IMAGE_SCN_ALIGN_2BYTES 0x200000
#define IMAGE_SCN_ALIGN_4BYTES 0x300000
#define IMAGE_SCN_ALIGN_8BYTES 0x400000
#define IMAGE_SCN_ALIGN_16BYTES 0x500000
#define IMAGE_SCN_ALIGN_32BYTES 0x600000
#define IMAGE_SCN_ALIGN_64BYTES 0x700000
#define IMAGE_SCN_LNK_NRELOC_OVFL 0x1000000
#define IMAGE_SCN_MEM_DISCARDABLE 0x2000000     // 프로세스가 시작된 이후에 필요 없는 데이터
                                                // 재배치 테이블
#define IMAGE_SCN_MEM_NOT_CACHED 0x4000000
#define IMAGE_SCN_MEM_NOT_PAGED 0x8000000       // 페이지 아웃이 안되고 항상 메모리에 있어야
                                                // 함을 나타내는 플래그
                                                // (디바이스 드라이버의 섹션)
#define IMAGE_SCN_MEM_SHARED 0x10000000         // 공유 가능한 섹션
#define IMAGE_SCN_MEM_EXECUTE 0x20000000        // 실행 가능 섹션
                                                // (IMAGE_SCN_CNT_CODE 플래그와 함께 지정)
#define IMAGE_SCN_MEM_READ 0x40000000           // 읽기 가능 섹션
#define IMAGE_SCN_MEM_WRITE 0x80000000          // 쓰기 가능 섹션
```

[그림 - Charcteristics 필드의 주요 플래그]

(15) 섹션(Section)

섹션 헤더 이후의 영역에는 실제 실행 코드와 데이터 혹은 리소스와 같은 여러 가지 실제적인 정보들이 섹션 헤더에서 명시한 위치와 크기별로 포함됨.
실제적인 데이터는 로더에 의해 메모리에 배치되고 실행됨.

코드	.text	- 프로그램 실행을 위한 코드가 담고 있는 섹션
데이터	.data	- 초기화된 전역 변수들을 담고 있는 읽고 쓰기

		가능한 섹션
	.rdata	- 읽기 전용 데이터 섹션 (문자열 표현, C++/COM 가상 함수 테이블 등)
	.bss	- 초기화되지 않으느 전역 변수들을 위한 섹션
Import API 정보	.idata	- 임포트(Import)할 DLL과 그 API들에 대한 정보를 담고 있는 섹션 - IAT(Import Address Table)이 존재, - .rdata에 병합하는 추세
	.didat	- 지연 로딩(Delay-Loading) 임포트 데이터를 위한 섹션
Export API 정보	.edata	- 익스포트(Export)할 API에 대한 정보를 담고 있는 섹션 - 주로 DLL - .rdata 또는 .text 섹션과 병합되어 DLL에서는 보이지 않음.
리소스	.rsrc	- 다이얼로그, 아이콘, 커서 등의 윈도우 APP 리소스 관련 데이터들이 배치되는 섹션
재배치 정보	.reloc	- 실행 파일에 대한 기본 재배치 정보를 담고 있는 섹션
TLS	.tls	- __declspec(thread) 지시어와 함께 선언되는 스레드 지역 저장소(Thread Local Storage)를 위한 섹션. - 런타임이 필요로 하는 부가적인 변수나 __declspec(thread) 지시어에 의한 데이터의 초기 값을 포함
C++ 런타임	.crt	- C++ 런타임(CRT)을 지원하기 위해 추가된 섹션
Short	.sdata	- 전역 포인터에 상대적으로 주소 지정될 수 있는 읽고 쓰기 가능한 "Short" 데이터 섹션 - IA-64 같은 전역 포인터 레지스터를 사용하는 플랫폼을 위해 사용 - IA-64 상에서의 정규화 크기 전역 변수들은 이 섹션이 존재
	.srdata	- .sdata에 들어갈 수 있는 데이터들의 읽기 전용 섹션
예외 정보	.pdta	- 예외 정보를 담고 있는 섹션 - IMAGE_RUNTIME_FUNCTION_ENTRY 구조체의 배열을 가지고 CPU 플랫폼에 의존적임
디버깅	.debug$S	- OBJ 파일에만 존재 - 가변 길이 코드뷰 포맷 심벌 레코드의 스트림
	.debug$T	- OBJ 파일에만 존재 - 가변 길이 코드뷰 포맷 타입 레코드의 스트림
	.debug$P	- 미리 컴파일된 헤더를 사용했을 때 OBJ 파일에

		만 존재하는 섹션.
Directives	**.derective**	- BJ 파일에만 존재

(16) 참고 자료 및 사이트

WINDOWS 시스템 실행파일의 구조와 원리 (한빛미디어-이호동)
Windows 구조와 원리 - OS를 관통하는 프로그래밍의 원리 (한빛미디어-정덕영)
ReverseCore 사이트 (http://www.reversecore.com/)
기억의 파편님의 블로그 (http://www.sinwoong.co.kr/259)
PE Structure (Deok9)
An In-Depth Look into the Win32 Portable Executable File Format
(http://msdn.microsoft.com/ko-kr/magazine/cc301805(en-us).aspx)