# PE File Format

* PE (Portable Executable) 파일은 Windows 운영체제에서 사용되는 실행 파일 형식입니다.
  + 기존 UNIX에서 사용되는 COFF (Common Object File Format)를 기반으로 Microsoft에서 만들었습니다.
* PE 파일은 32비트 형태의 실행 파일을 의미하며, PE32라는 용어를 사용하기도 합니다.
  + 64비트 형태의 실행 파일은 PE+또는 PE32+라고 부르며 PE(또는 PE32) 파일의 확장 형태입니다. ( PE64가 아닙니다. )

# PE File Format

* PE 파일의 종류
  + 실행 계열 : EXE, SCR
  + 라이브러리 계열 : DLL, OCX, CPL, DRV
  + 드라이버 계열 : SYS, VXD
  + 오브젝트 파일 계열 : OBJ
* OBJ (오브젝트) 파일을 제외하면 모든 것은 실행 가능한 파일입니다.
  + DLL, SYS 파일 등은 셸 (Explorer.exe) 에서 직접 실행할 수는 없지만, 다른 형태의 방법 ( 디버거, 서비스, 기타 )을 이용하여 실행이 가능한 파일입니다

# 기본 구조

* DOS header ~ Section header : PE Header ( PE 헤더 )
* 그 밑의 Section들 : PE Body ( PE 바디 )
  + 파일에서는 offset으로, 메모리에서는 VA (Virtual Address, 절대주소)로 위치를 표현합니다.
  + 파일이 메모리에 로딩되면 모양이 달라집니다. ( Section의 크기, 위치 등 )
    - 파일의 내용은 보통 코드(.text), 데이터(.data), 리소스(.rsrc) 섹션에 나뉘어서 저장됩니다.
* 섹션 헤더 ( Section header [ PE header ] )
  + 섹션 헤더에는 각 Section에 대한 파일/메모리에서의 크기, 위치, 속성 등이 정의되어 있습니다.
* PE 헤더의 끝부분과 각 섹션의 끝에는 NULL padding이라고 불리는 영역이 존재합니다.
  + 컴퓨터에서 파일, 메모리, 네트워크 패킷 등을 처리할 때 효율을 높이기 위해 최소 기본 단위 개념을 사용하는데, PE 파일에도 같은 개념이 적용된 것입니다.
  + 파일/메모리에서 섹션의 시작 위치는 각 파일/메모리의 최소 기본 단위의 배수에 해당하는 위치여야 하고, 빈 공간은 NULL로 채워버립니다.

# VA & RVA

* VA (Virtual Address)는 프로세스 가상 메모리의 절대주소를 말합니다
* RVA (Relative Virtual Address)는 어느 기준 위치(ImageBase)에서부터의 상대주소를 말합니다.
  + RVA + ImageBase = VA
* PE 헤더 내의 정보는 RVA 형태로 된 것이 많습니다.
  + 그 이유는 PE 파일(주로 DLL)이 프로세스 가상 메모리의 특정 위치에 로딩되는 순간, 이미 그 위치에 다른 PE 파일(DLL)이 로딩되어 있을 수 있습니다.
    - 그럴 때 재배지(Relocation) 과정을 통해서 비어 있는 다른 위치에 로딩되어야 하는데, 만약 PE 헤더 정보들이 VA(Virtual Address, 절대주소)로 되어 있다면 정상적인 Access가 이루어지지 않을 것입니다.
    - 그러므로 정보를 RVA(Relative Virtual Address, 상대주소)로 해두면 Relocation이 발생해도 기준 위치에 대한 상대주소가 변하지 않기 때문에 아무런 문제 없이 원하는 정보에 Access할 수 있는 것입니다.

# PE 헤더

* DOS 헤더
  + Microsoft는 PE File Format을 만들 때 동시에 널리 사용되던 DOS 파일에 대한 하위 호환성을 고려해서 만들었습니다.
  + 그 결과로 PE 헤더의 제이 앞부분에는 기존 DOS EXE Header를 확장시킨 IMAGE\_DOS\_HEADER 구조체가 존재합니다.

Typedef struct \_IMAGE\_DOS\_HEADER

{

WORD e\_magic; // DOS signature : 4D5A ( “MZ “ )

…

LONG e\_lfanew; // offset to NT header

} IMAGE\_DOS\_HEADER, \*PIMAGE\_DOS\_HEADER;

* + IMAGE\_DOS\_HEADER 구조체의 크기는 40입니다.
    - e\_magic : DOS signature (4D5A => ASCII 값 “MZ”)
    - e\_lfanew : NT 헤더의 옵셋(offset)을 표시(파일에 따라 가변적인 값을 가짐)
  + 모든 PE 파일에는 시작 부분(e\_magic)에 DOS signature (“MZ”)가 존재하고,
  + e\_lfanew 값이 가리키는 위치에 NT header 구조체가 존재해야 합니다.
    - NT header 구조체의 이름은 IMAGE\_NT\_HEADERS 입니다.
* DOS Stub
  + DOS Header 밑에는 DOS Stub이 존재합니다. DOS Stub의 존재 여부는 옵션이며 크기도 일정하지 않습니다.
    - Notepad 기준, 40 ~ 4D 영역 ( 16비트 어셈블리 명령 )이며 Windows OS에서는 이쪽 명령어가 실행되지 않습니다. (PE 파일로 인식하기 때문 )
      * 만약 DOS 환경에서나, DOS 용 디버거를 이용해서 실행하면 저 코드가 실행됩니다.
      * CMD : …/notepad.exe -u
* NT Header
  + NT Header 구조체 IMAGE\_NT\_HEADERS

Typedef struct \_IMAGE\_NT\_HEADERS

{

DWORD Signature; // PE Signature : 50450000 (“PE”00)

IMAGE\_FILE\_HEADER FileHeader;

IMAGE\_OPTIONAL\_HEADER32 OptionalHeader;

} IMAGE\_NT\_HEADERS32, \*PIMAGE\_NT\_HEADERS32;

* + IMGAE\_NT\_HEADERS 구조체의 크기는 F8입니다.
  + NT Header – File Header
    - 파일의 개략적인 속성을 나타내는 IMAGE\_FILE\_HEADER 구조체입니다.

Typedef struct \_IMAGE\_FILE\_HEADER

{

WORD Machine;

WORD NumberOfSections;

…

WORD SizeOfOptionalHeader;

WORD Characteristics;

} IMAGE\_FILE\_HEADER, \*PIMAGE\_FILE\_HEADER;

* + - IMAGE\_FILE\_HEADER 구조체에서는 4가지 맴버(Machine, NumberOfSection, SizeofOptionalHeader, Characteristics)가 중요합니다.
    - Machine – IMAGE\_FILE\_HEADER – NT HEADER
      * Machine 넘버는 CPU 별로 고유한 값이며, 32비트 Intel x86 호환 칩은 14C의 값을 가집니다.
      * winnt.h 파일에는 정의된 Machine 넘버의 값들이 있습니다.
    - NumberOfSections – IMAGE\_FILE\_HEADER – NT HEADER
      * PE 파일은 코드, 데이터, 리소스 등이 각각의 섹션에 나뉘어서 저장되는데, NumberOfSection는 바로 섹션의 개수를 나타냅니다.
      * 그런데 이 값은 반드시 0보다 커야하며, 또 정의된 섹션 개수와 실제 섹션이 다르면 실행 에러가 발생합니다.
    - SizeOfOptionalHeader – IMAGE\_NT\_HEADER – NT\_HEADER
      * IMAGE\_NT\_HEADERS 구조체의 마지막 맴버는 IMAGE\_OPTIONAL\_HEADER32 구조체입니다.
      * 이 맴버는 IMAGE\_OPTIOANL\_HEADER32 구조체의 크기를 나타냅니다.
      * IMAGE\_OPTIONAL\_HEADER32는 C언어의 구조체이기 때문에 이미 그 크기가 결정되어 있습니다만, Windows의 PE 로더는 IMAGE\_FILE\_HEADER의 SizeOfOptionalHeader 값을 보고 IMAGE\_OPTIONAL\_HEADER32 구조체의 크기를 인식합니다.
        + PE32+ (64비트) 형태의 파일인 경우에는 IMAGE\_OPTIONAL\_HEADER32 구조체 대신 IMAGE\_OPTIONAL\_HEADER64 구조체를 사용합니다.

IMAGE\_DOS\_HEADER의 e\_lfanew 맴버와 IMAGE\_FILE\_HEADER의 SizeOfOptionalHeader맴버 때문에 일반적인(상식적인) PE 파일 형식을 벗어나는 일명 “꽈배기” PE 파일(PE Patch)을 만들 수 있습니다.

* + - Characteristics – IMAGE\_NT\_HEADER – NT\_HEADER
      * 파일의 속성을 나타내는 값으로, 실행이 가능한 형태인지 (executable or not) 혹은 DLL 파일인지 등의 정보들이 bit OR 형식으로 조합됩니다.
      * Winnt.h 파일에 정의된 Characteristics 값이 있습니다.
        + (0002h, 2000h의 값이 중요합니다.)

#define IMAGE\_FILE\_EXECUTABLE\_IMAGE 0x0002 // File is executable

#define IMAGE\_FILE\_LINE\_NUMS\_STRIPPED 0x0004 // Line Numbers

* + - * + Characteristics 값에 0002h가 없는 경우는 \*.obj와 같은 object 파일 및 resource DLL 같은 파일을 들 수 있습니다.
  + NT Header – Optional Header
    - PE 헤더 구조체 중에서 가장 크기가 큰 IMAGE\_OPTIONAL\_HEADER32입니다.

Typedef struct \_IMAGE\_DATA\_DIRECTORY

{

DWORD VirtualAddress;

DWORD Size;

} IMAGE\_DATA\_DIRECTORY, \*PIMAGE\_DATA\_DIRECTORY;

#define IMAGE\_NUMBEROF\_DIRECTORY\_ENTRIES 16

Typedef struct \_IMAGE\_DIRECTORY\_ENTRIES 16

Typedef struct \_IMAGE\_OPTIONAL\_HEADER

{

WORD Magic;

…

DWORD AddressOfEntryPoint

…

DWORD ImageBase

…

DWORD SectionAlignment

DOWRD FileAlignment

…

DWORD SizeOfImage

DWORD SizeOfHeader

WORD Subsystem

DWORD NumberOfRvaAndSizes;

IMAGE\_DATA\_DIRECTORY DataDirectory[IMAGE\_NUMBEROF\_DIECTORY\_ENTRIES];

} IMAGE\_OPTIONAL\_HEADER32, \*PIMAGE\_OPTIONAL\_HEADER32;

* + Magic
    - Magic 넘버는 IMAGE\_OPTIONAL\_HEADER32 구조체인 경우 10B, IMAGE\_OPTIONAL\_HEADER64 구조체인 경우 20B 값을 가집니다.
  + AddressOfEntryPoint
    - AddressOfEntryPoint는 EP (Entry Point)의 RVA(Relative Virtual Address) 값을 가지고 있습니다. 이 값은 프로그램에서 최초로 실행되는 코드의 시작 주소로, 매우 중요한 값입니다.
  + ImageBase
    - 프로세스 가상 메모리는 0~FFFFFFFF (32비트의 경우) 범위입니다. ImageBase는 이렇게 광할한 메모리에서 PE 파일이 로딩되는 시작 주소를 나타냅니다.
    - EXE, DLL 파일은 user memory 영역인 0~7FFFFFFF 범위에 로딩되고, SYS 파일은 kernel memory 영역인 80000000~FFFFFFFF 범위에 로딩됩니다.
    - 일반적으로 개발도구(VB/VC++/Delphi)들이 만들어내는 EXE 파일의 Image Base 값은 00400000이고, DLL 파일의 ImageBase 값은 10000000입니다.
    - PE 로더는 PE 파일을 실행시키기 위해 프로세스를 생성하고 파일을 메모리를 로딩한 후 EIP 레지스터 값을 ImageBase + AddressOfEntry Point 값으로 세팅합니다.
  + SectionAlignment, FileAlignment
    - PE 파일의 Body 부분은 섹션(Section)으로 나뉘어져 있습니다.
    - 파일에서 섹션의 최소단위를 나타내는 것이 FileAlignment이고,
    - 메모리에서 섹션의 최소단위를 나타내는 것이 SectionAlignment입니다.
      * 하나의 파일에서 FileAlignment와 SectionAlignment의 값은 같을 수도 있고 다를 수도 있습니다.
    - 파일/메모리의 섹션 크기는 반드시 각각 FileAlignment/SectionAlignment의 배수가 되어야 합니다.
  + SizeOfImage
    - SizeOfImage는 PE 파일이 메모리에 로딩되었을 때 가상 메모리에서 PE Image가 차지하는 크기를 나타냅니다. 일반적으로 파일의 크기와 메모리에 로딩된 크기는 다릅니다.
  + SizeOfHeader
    - SizeOfHeader는 PE 헤더의 전체 크기를 나타냅니다. 이 값 역시 FileAlignment의 배수여야 합니다. 파일 시작에서 SizeOfHeader 옵셋만큼 떨어진 위치에 첫 번째 섹션이 위치합니다.
  + Subsystem
    - 이 Subsystem의 값을 보고 시스템 드라이버 파일(\*.sys)인지, 일반 실행 파일(\*.exe, \*.dll)인지 구분할 수 있습니다.
      * 1 – Driver file – 시스템 드라이브
      * 2 – GUI 파일 – 창 기반 애플리케이션
      * 3 – CUI 파일 – 콘솔 기반 애플리케이션
  + NumberOfRvaAndSizes
    - NumberOfRvaAndSizes는 IMAGE\_OPTIONAL\_HEADER32 구조체의 마지막 맴버인 DataDirectory 배열의 개수를 나타냅니다. 구조체 정의에 분명히 배열 개수가 IMAGE\_NUMBEROF\_DIRECTORY\_ENTRIES (16)이라고 명시되어 있지만, PE 로더는 NumberOfRvaAndSizes의 값을 보고 배열의 크기를 인식합니다.
    - 즉, 16이 아닐수도 있다는 뜻입니다.
  + DataDirectory
    - DataDirectory는 IMAGE\_DATA\_DIRECTORY 구조체의 배열로, 배열의 각 항목마다 정의된 값을 가집니다

DataDirectory[0] = EXPORT Directory

DataDirectory[1]= IMPORT Directory

DataDirectory[2] = RESOURCE Directory

…

DataDirectory[9] = TLS Directory

…

DataDirectory[F] = Reserved Directory

* + - EXPORT, IMPORT, RESOUCE, TLS 가 제일 중요함