

PE File Format

SCP 부원 신재형

PE File Format이란?

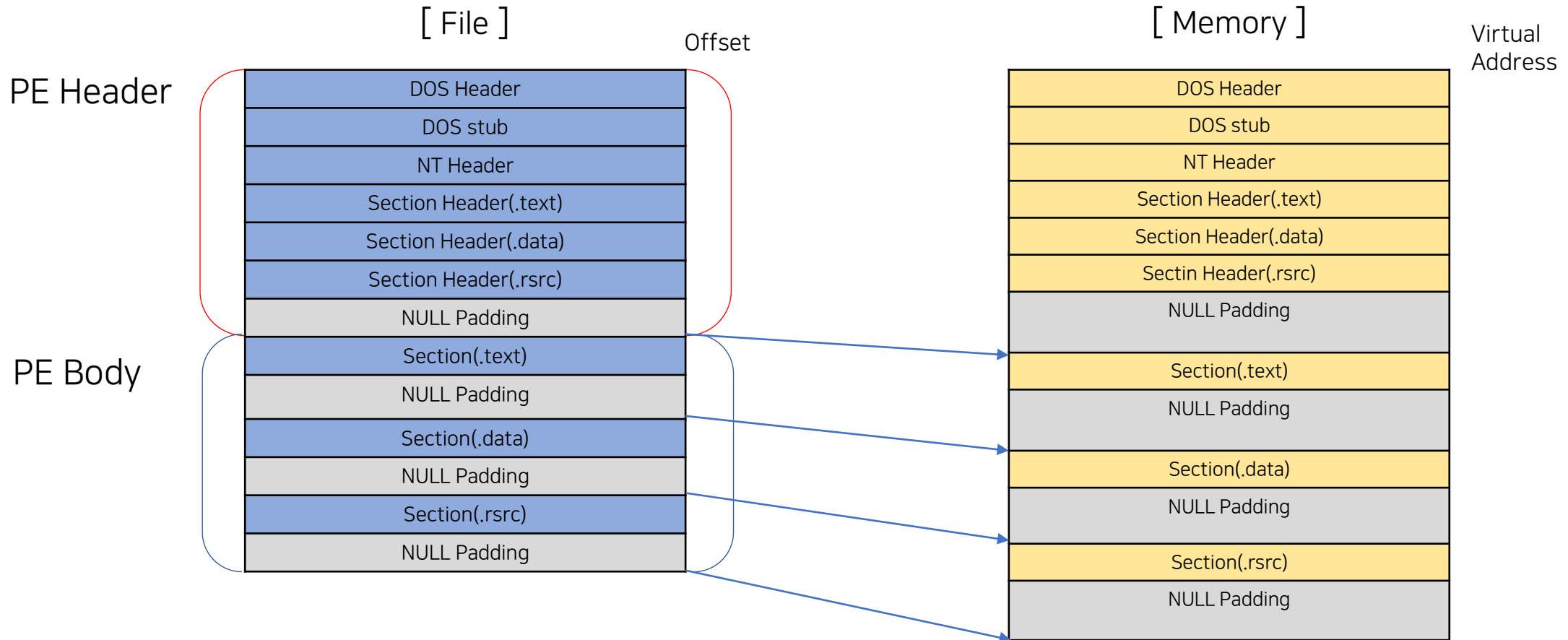
실행 파일 계열 : EXE, SCR

라이브러리 계열 : DLL, OCX

드라이버 계열 : SYS

오브젝트 파일 계열 : OBJ → 실행 불가능!

PE File의 기본 구조



VA & RVA

VA(Virtual Address) : 프로세스 가상 메모리의 절대 주소

RVA(Relative Virtual Address) : 기준 위치(ImageBase)에서부터의 상대 주소

$$RVA + ImageBase = VA$$

PE Header

IMAGE_DOS_HEADER

```
typedef struct _IMAGE_DOS_HEADER {  
    WORD e_magic; // DOS signature : 4D5A ("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; // offset of NT header (가변적)  
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

크기 : 40h
값 변경 X

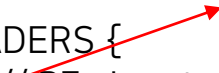
PE Header

NT_HEADER

크기 : F8h

```
typedef struct _IMAGE_NT_HEADERS {  
    DWORD Signature; // PE signature : 50450000h ("PE"00)  
    IMAGE_FILE_HEADER FileHeader;  
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;  
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

변경 불가



PE Header

NT_HEADER - File Header

```
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;
```

32bit : 14Ch
64bit : 200h

CPU 별로 고유한 값

섹션(code, data, rsrc, ...) 의 개수 > 0

IMAGE_OPTIONAL_HEADER32 구조체의 크기

파일의 속성을 나타내는 값

NumberOfSections > 실제 섹션 : Error

NumberOfSections < 실제 섹션 : 정의된 만큼 인식

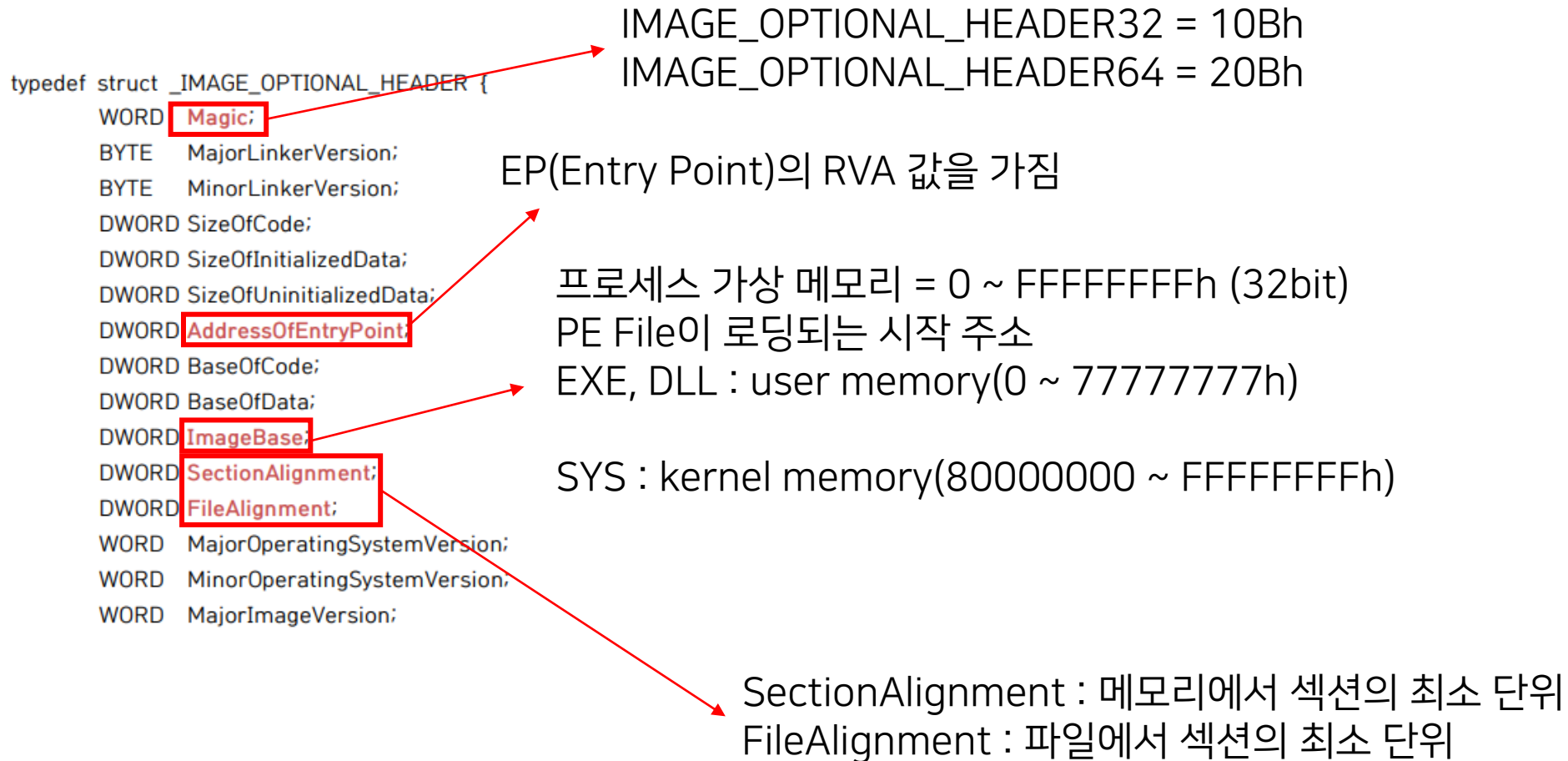
PE Header

NT_HEADER – Optional Header

```
typedef struct _IMAGE_OPTIONAL_HEADER {  
    WORD    Magic;  
    BYTE    MajorLinkerVersion;  
    BYTE    MinorLinkerVersion;  
    DWORD   SizeOfCode;  
    DWORD   SizeOfInitializedData;  
    DWORD   SizeOfUninitializedData;  
    DWORD   AddressOfEntryPoint;  
    DWORD   BaseOfCode;  
    DWORD   BaseOfData;  
    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;
```

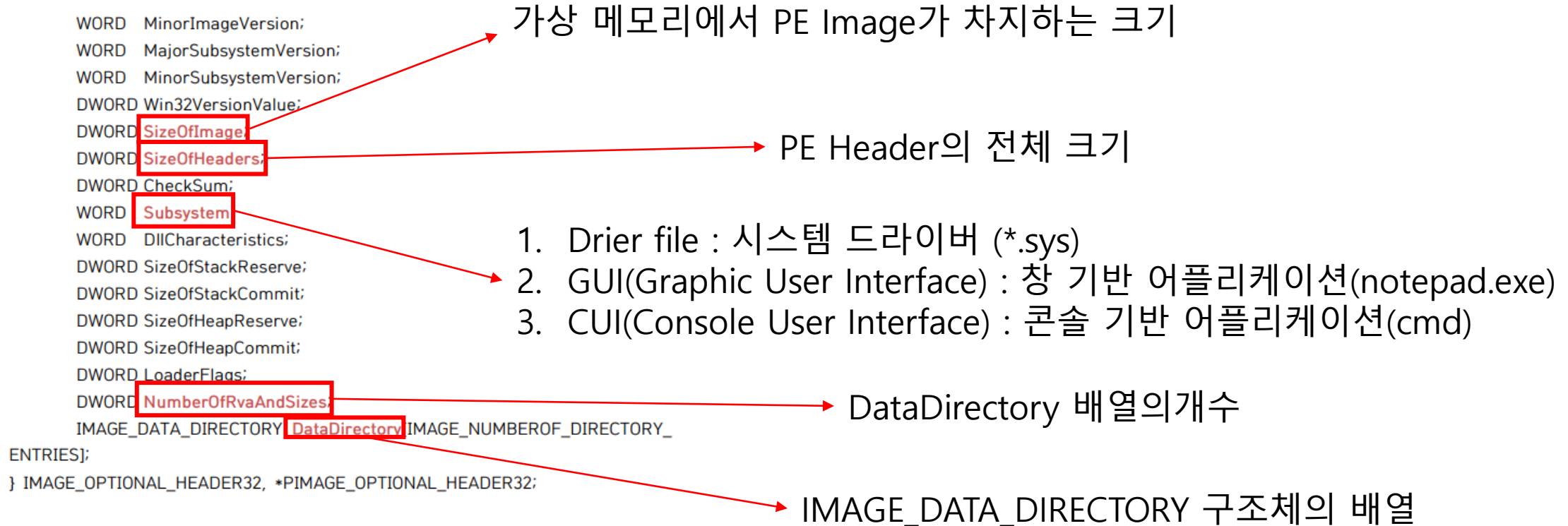

PE Header

NT_HEADER – Optional Header



PE Header

NT_HEADER - Optional Header



PE Header

Section Header

Section Header : 각 Section의 속성을 정의한 것

Why?

프로그램의 안정성을 위해 비슷한 성격의 자료를 모아두어 속성을 다르게 설정

-code : 실행, 읽기 권한

-data : 비실행, 읽기, 쓰기 권한

-resource : 비실행, 읽기 권한

PE Header

IMAGE_SECTION_HEADER

```
typedef struct _IMAGE_SECTION_HEADER {  
    BYTE Name[IMAGE_SIZEOF_SHORT_NAME];  
    union {  
        DWORD PhysicalAddress;  
        DWORD VirtualSize; // 메모리에서 섹션이 차지하는 크기  
    } Misc;  
    DWORD VirtualAddress; // 메모리에서 섹션의 시작 주소 (RVA)  
    DWORD SizeOfRawData; // 파일에서 섹션이 차지하는 크기  
    DWORD PointerToRawData; // 파일에서 섹션의 시작 위치  
    DWORD PointerToRelocations;  
    DWORD PointerToLinenumbers;  
    WORD NumberOfRelocations;  
    WORD NumberOfLinenumbers;  
    DWORD Characteristics; // 섹션의 속성  
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

VirtualAddress와 PointerToRawData
→ SectionAlignment(메모리), FileAlignment(파일)
에 맞게 결정

VirtualSize ≠ SizeOfRawData

RVA to RAW

RVA to RAW : PE File이 메모리에 로딩되었을 때 각 섹션에서 메모리의 주소(RVA)와 같은 파일 오프셋을 매핑하는 것

1. RVA가 속해있는 섹션을 찾는다
2. 비례식을 이용해서 파일 오프셋(RAW)를 계산한다.

$$\text{RAW} - \text{PointerToRawData} = \text{RVA} - \text{VirtualAddress}$$

$$\text{RAW} = \text{RVA} - \text{VirtualAddress} + \text{PointerToRawData}$$

RAW : 파일 오프셋

PointerToRawData : 파일에서 섹션의 시작 위치

RVA : 메모리의 주소

VirtualAddress : 메모리에서 섹션의 시작 위치

IAT(Import Address Table)

DLL

IAT : 프로그램이 어떤 라이브러리에서 어떤 함수를 사용하고 있는지를 기술한 테이블

DLL(Dynamic Linked Library) : 동적 연결 라이브러리

DLL 로딩 방식

1. Explicit Linking : 프로그램 내에서 사용되는 순간에 로딩하고 사용이 끝나면 메모리에서 해제 시키는 방법
2. Implicit Linking : 프로그램이 시작할 때 같이 로딩되어 프로그램이 종료할 때 메모리에서 해제되는 방법

IAT(Import Address Table)

IMAGE_IMPORT_DESCRIPTOR

IMAGE_IMPORT_DESCRIPTOR : 어떤 라이브러리를 Import하고 있는지

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD Characteristics;
        DWORD OriginalFirstThunk; // INT(Import Name Table) 주소(RVA)
    };
    DWORD TimeDateStamp;
    DWORD ForwarderChain;
    DWORD Name; // Library 이름 문자열 주소 (RVA)
    DWORD FirstThunk; // IAT (Import Address Table) 주소 (RVA)
} IMAGE_IMPORT_DESCRIPTOR;

typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD Hint; // ordinal
    BYTE Name[1]; // function name string
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

배열 형식으로 존재



-INT, IAT : long type(4byte)배열이고 NULL로 끝남

-INT와 IAT의 크기는 같아야 함

-INT에서 각 원소의 값은
IMAGE_IMPORT_BY_NAME 구조체 주소 값을
가짐

EAT(Export Address Table)

Windows Library : 다른 프로그램에서 불러 사용할 수 있도록 관련 함수들을 모아 놓은 파일

대표적 라이브러리 : Win32 API

대표적 Library File : kernel32.dll

EAT : 라이브러리 파일에서 제공하는 함수를 다른 프로그램에서 가져다 사용할 수 있도록 해주는 메커니즘

EAT는 IAT와 마찬가지로 PE 파일내에 특정 구조체(IMAGE_EXPORT_DIRECTORY)에 정보를 저장하고 있고 이 구조체는 PE 파일에 하나만 존재

EAT(Export Address Table)

IMAGE_EXPORT_DIRECTORY

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

1.NumberOfFunctions : 실제 export 함수 개수

2.NumberOfNames : export 함수중에서 이름을 가지는 함수 개수

3.AddressOfFunctions : export 함수들의 시작 위치 배열의 주소

4.AddressOfNames : 함수 이름 배열의 주소

5.AddressOfOrdinals : ordinal 배열의 주소

Q & A