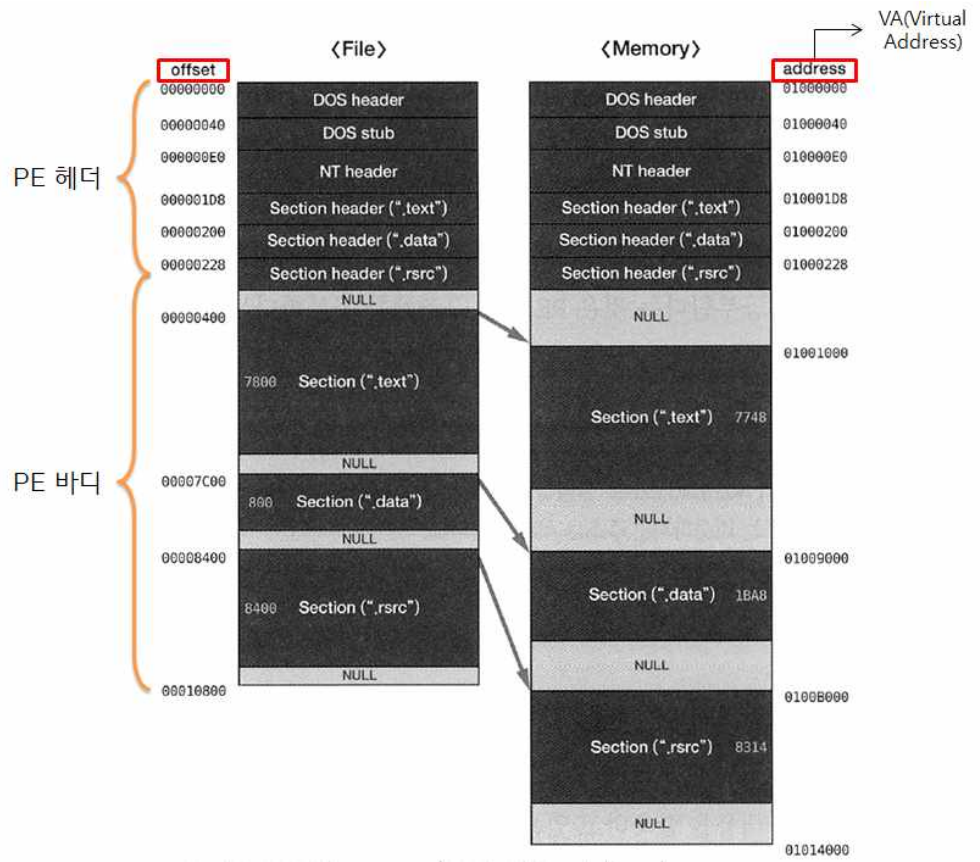
psnyeo88@gmail.com
rosnot

PE(Portable Executable) File Format

1. PE File
 Windows 운영체제에서 사용되는 실행 파일 형식
 32비트 : PE(32) / 64비트 : PE(32)+

2. PE 파일 종류
 실행 계열 - EXE, SCR
 드라이버 계열 - SYS, VXD
 라이브러리 계열 - DLL, OCX, CPL, DRV
 오브젝트 파일 계열 - OBJ (실행 불가능)

3. 기본 구조
 notepad.exe 파일이 메모리에 적재될 때의 모습 (Section의 크기, 위치 등이 달라짐)



 섹션 헤더 : 각 Section에 대한 파일/메모리에서의 크기, 위치 속성 등이 정의
 NULL padding : 파일/메모리에서 섹션의 시작 위치를 각 파일/메모리의 최소 기본 단위의 배수에 해당하는 위치로 만들어주기 위해

VA & RVA

VA(Virtual Address) : 프로세스 가상 메모리의 절대주소
RVA(Relative Virtual Address) : 기준 위치(ImageBase)에서부터의 상대주소
VA = ImageBase + RVA

PE 헤더 내의 정보는 RVA 형태로 된 것이 많음
why? PE 파일(주로 DLL)이 프로세스 가상 메모리의 특정 위치에 로딩되는 순간 이미 그 위치에 다른 PE 파일(DLL)이 로딩되어 있을 수 있음. 이때 **재배치**과정을 통해서 비어 있는 다른 위치에 로딩되어야 함. (VA로 되어 있으면 엑세스에 문제 발생할 것)

---

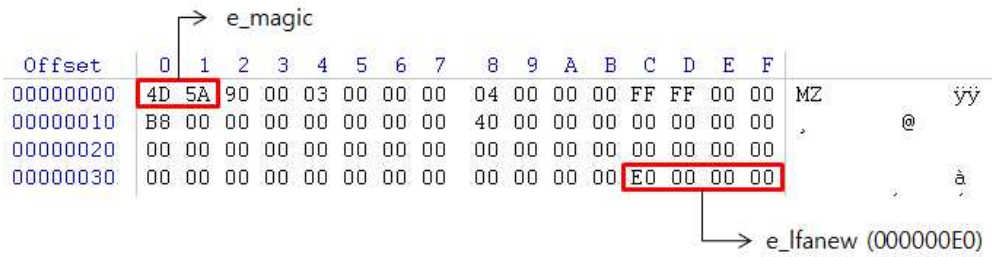
4. PE 헤더

4.1. DOS Header

DOS 파일에 대한 하위 호환성을 위해 DOS EXE Header를 확장시킨 IMAGE_DOS_HEADER 구조체가 존재

IMAGE_DOS_HEADER 구조체 (크기 40h)

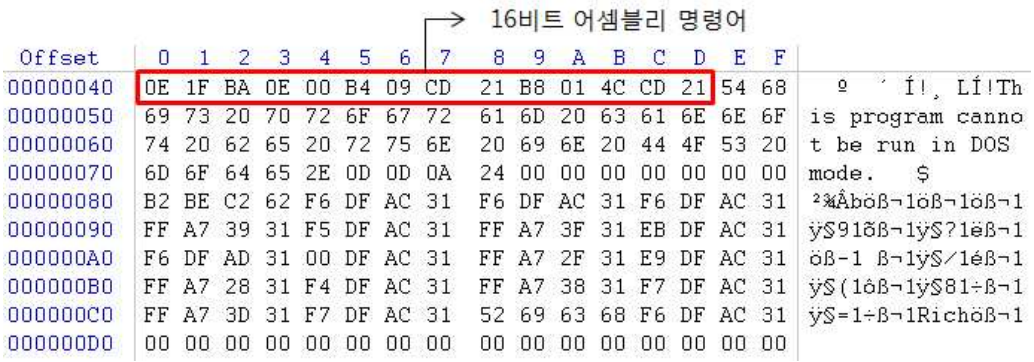
```
typedef struct _IMAGE_DOS_HEADER {
        WORD   e_magic;         // DOS signature : 4D5A ("MZ")
        WORD   e_cblp;
        WORD   e_cp;
        WORD   e_crlc;
        WORD   e_cparhdr;
        WORD   e_minalloc;
        WORD   e_maxalloc;
        WORD   e_ss;
        WORD   e_sp;
        WORD   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;
```



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ ÿÿ
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	E0	00	00	00	à

4.2. DOS Stub

존재 여부는 옵션이며 크기도 일정하지 않음 (없어도 파일 실행에 문제없음)
코드와 데이터의 혼합으로 이루어져 있음



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	º ´ Í! ¸ LÍ!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	B2	BE	C2	62	F6	DF	AC	31	F6	DF	AC	31	F6	DF	AC	31	²¾Âböß¬1öß¬1öß¬1
00000090	FF	A7	39	31	F5	DF	AC	31	FF	A7	3F	31	EB	DF	AC	31	ÿ§91õß¬1ÿ§?1ëß¬1
000000A0	F6	DF	AD	31	00	DF	AC	31	FF	A7	2F	31	E9	DF	AC	31	öß-1 ß¬1ÿ§/1éß¬1
000000B0	FF	A7	28	31	F4	DF	AC	31	FF	A7	38	31	F7	DF	AC	31	ÿ§(1ôß¬1ÿ§81÷ß¬1
000000C0	FF	A7	3D	31	F7	DF	AC	31	52	69	63	68	F6	DF	AC	31	ÿ§=1÷ß¬1Richöß¬1
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Offset 40~4D는 16비트 어셈블리 명령어로 32비트 OS에서는 실행되지 않음(DOS환경이나, DOS용 디버거를 이용하면 실행가능)

```
0D1E:0000 0E        PUSH    CS
0D1E:0001 1F        POP     DS
0D1E:0002 BA0E00    MOV     DX,000E   ; DX = 0E : "This program cannot be
                                            run in DOS mode"
0D1E:0005 B409      MOV     AH,09
0D1E:0007 CD21      INT     21        ; AH = 09 : WriteString()
0D1E:0009 B8014C    MOV     AX,4C01
0D1E:000C CD21      INT     21        ; AX = 4C01 : Exit()
```

화면에 문자열("This program cannot be run in DOS mode.")을 출력하고 종료되는 코드
→ 하나의 실행파일에 DOS와 Windows에서 모두 실행 가능한 파일을 만들 수도 있음

4.3. NT Header

IMAGE_NT_HEADERS 구조체 (크기 F8h)

```
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;
```

4.4. NT Header - File Header

IMAGE_File_HEADER 구조체 (크기 14h)

```
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;
```

#1. Machine : CPU별로 고유한 값

Machin 넘버의 값 (winnt.h)

```
#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_SH3              0x01a2  // SH3 little-endian
#define IMAGE_FILE_MACHINE_SH3DSP           0x01a3
#define IMAGE_FILE_MACHINE_SH3E             0x01a4  // SH3E little-endian
#define IMAGE_FILE_MACHINE_SH4              0x01a6  // SH4 little-endian
#define IMAGE_FILE_MACHINE_SH5              0x01a8  // SH5
#define IMAGE_FILE_MACHINE_ARM              0x01c0  // ARM Little-Endian
#define IMAGE_FILE_MACHINE_THUMB            0x01c2  // ARM Thumb/Thumb-2 Little-Endian
#define IMAGE_FILE_MACHINE_ARMNT            0x01c4  // ARM Thumb-2 Little-Endian
#define IMAGE_FILE_MACHINE_AM33             0x01d3
#define IMAGE_FILE_MACHINE_POWERPC          0x01F0  // IBM PowerPC Little-Endian
#define IMAGE_FILE_MACHINE_POWERPCFP        0x01f1
#define IMAGE_FILE_MACHINE_IA64             0x0200  // Intel 64
#define IMAGE_FILE_MACHINE_MIPS16           0x0266  // MIPS
#define IMAGE_FILE_MACHINE_ALPHA64          0x0284  // ALPHA64
#define IMAGE_FILE_MACHINE_MIPSFPU          0x0366  // MIPS
#define IMAGE_FILE_MACHINE_MIPSFPU16        0x0466  // MIPS
#define IMAGE_FILE_MACHINE_AXP64            IMAGE_FILE_MACHINE_ALPHA64
#define IMAGE_FILE_MACHINE_TRICORE          0x0520  // Infineon
#define IMAGE_FILE_MACHINE_CEF              0x0CEF
#define IMAGE_FILE_MACHINE_EBC              0x0EBC  // EFI Byte Code
#define IMAGE_FILE_MACHINE_AMD64            0x8664  // AMD64 (K8)
#define IMAGE_FILE_MACHINE_M32R             0x9041  // M32R little-endian
#define IMAGE_FILE_MACHINE_CEE              0xC0EE
```

#2. NumberOfSections : 섹션(code, data, rsrc, ...)의 개수(>0, 정의된 개수=실제 개수)

#3. SizeOfOptionalHeader : IMAGE_OPTIONAL_HEADER32 구조체의 크기 (C언어의 구조체이기 때문에 이미 크기가 결정되어 있으나 PE 로더는 이 값을 보고 구조체의 크기를 인식함) PE32+ 파일은 IMAGE_OPTIONAL_HEADER64 구조체를 사용하고 크기가 달라서

#4. Characteristics : 파일의 속성을 나타내는 값 (bit OR)

Characteristics 값 (winnt.h)

```
#define IMAGE_FILE_RELOCS_STRIPPED          0x0001  // Relocation info stripped
from file.
#define IMAGE_FILE_EXECUTABLE_IMAGE         0x0002  // File is executable  (i.e. no
unresolved external references).
#define IMAGE_FILE_LINE_NUMS_STRIPPED       0x0004  // Line nunbers stripped
from file.
#define IMAGE_FILE_LOCAL_SYMS_STRIPPED      0x0008  // Local symbols stripped
from file.
#define IMAGE_FILE_AGGRESIVE_WS_TRIM        0x0010  // Aggressively 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.
```

4.5. NT Header - Optional Header

IMAGE_OPTIONAL_HEADER32 구조체 (크기 E0h)

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

#define IMAGE_NUMBEROF_DIRECTORY_ENTRIES        16
```

```
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;
```

#1. Magic
IMAGE_OPTIONAL_HEADER32 : 10B / IMAGE_OPTIONAL_HEADER64 : 20B

#2. AddressOfEntryPoint : EP의 RVA값 (프로그램에서 최초로 실행되는 코드의 주소)

#3. ImageBase : PE파일이 로딩되는 시작 주소

EXE, DLL 파일은 0~7FFFFFFF (user memory 영역) 범위에 로딩
SYS 파일은 80000000~FFFFFFFF (kernel memory 영역) 범위에 로딩
일반적으로 EXE : 00400000, DLL : 10000000 (VB/VC++/Delphi)
PE로더는 PE파일을 실행시키기 위해 프로세스를 생성하고 파일을 메모리에 로딩한 후 EIP 레지스터 값을 **ImageBase + AddressOfEntryPoint** 값으로 세팅

#4. SectionAlignment, FileAlignment
FileAlignment : 파일에서 섹션의 최소단위
SectionAlignment : 메모리에서 섹션의 최소단위

#5. SizeOfImage : 가상 메모리에서 PE Image가 차지하는 크기

#6. SizeOfHeaders : PE 헤더 전체의 크기 (FileAlignment의 배수)
첫 번째 섹션은 파일 시작에서 SizeOfHeaders 옵셋만큼 떨어진 위치에 존재

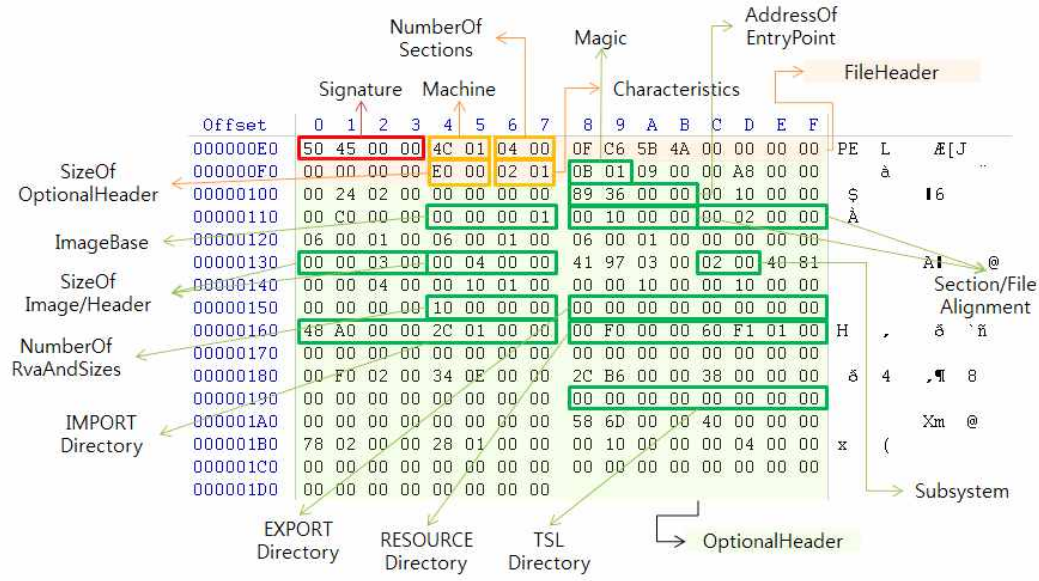
#7. Subsystem

값	의미	비고
1	Driver file	시스템 드라이버 (예: ntfs, sys)
2	GUI(Graphic User Interface) 파일	창 기반 애플리케이션 (예: notepad.exe)
3	CUI(Console User Interface) 파일	콘솔 기반 애플리케이션 (예: cmd.exe.)

#8. NumberOfRvaAndSizes : DataDirectory 배열의 개수 (구조체 정의에 배열의 개수가 16으로 명시되어 있지만 PE 로더는 이 값을 보고 인식, 16이 아닐 수도 있다는 의미)

#9. DataDirectory : IMAGE_DATA_DIRECTORY 구조체의 배열

```
DataDirectory[0] = EXPORT Directory
DataDirectory[1] = IMPORT Directory
DataDirectory[2] = RESOURCE Directory
DataDirectory[3] = EXCEPTION Directory
DataDirectory[4] = SECURITY Directory
DataDirectory[5] = BASERELOC Directory
DataDirectory[6] = DEBUG Directory
DataDirectory[7] = COPYRIGHT Directory
DataDirectory[8] = GLOBALPTR Directory
DataDirectory[9] = TLS Directory
DataDirectory[A] = LOAD_CONFIG Directory
DataDirectory[B] = BOUND_IMPORT Directory
DataDirectory[C] = IAT Directory
DataDirectory[D] = DELAY_IMPORT Directory
DataDirectory[E] = COM_DESCRIPTOR Directory
DataDirectory[F] = Reserved Directory
```



4.6. IMAGE_SECTION_HEADER

섹션 헤더 : 각 섹션의 속성을 정의 (file/memory에서의 시작 위치, 크기, 엑세스권한 등)

why? 프로그램의 안정성을 위해 비슷한 성격의 자료를 모아두어 속성을 다르게 설정

→ code : 실행, 읽기 / data : 비실행, 읽기, 쓰기 / resource : 비실행, 읽기

IMAGE_SECTION_HEADER 구조체 (크기 28h)

```
#define IMAGE_SIZEOF_SHORT_NAME     8

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 (bit OR)
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

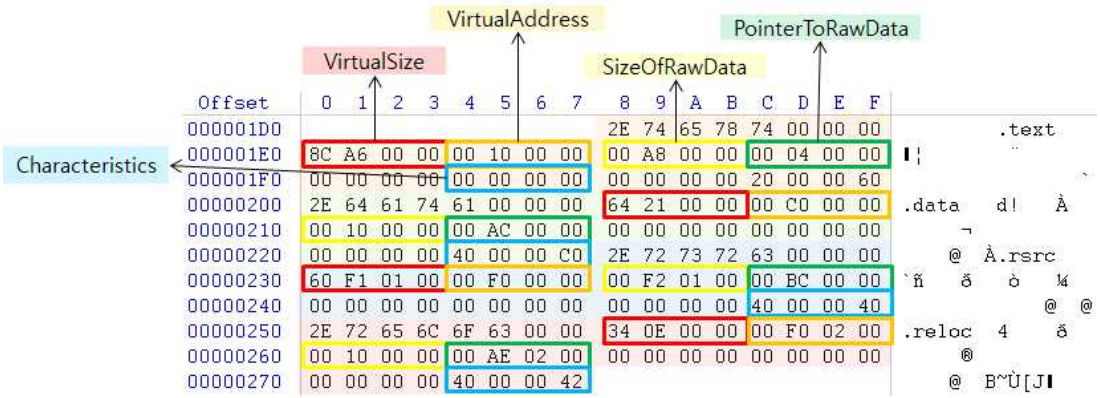
VirtualAddress/PointerToRawData : SectionAlignment/FileAlignment에 맞게 결정

VirtualSize ≠ SizeOfRawData → 파일에서의 섹션 크기 ≠ 메모리에 로딩된 섹션 크기

Characteristics 값 (winnt.h)

```
#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.
#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 writable.
```

Name : NULL로 끝나지 않음, ASCII 값이 안와도 됨



4.7. RVA to RAW

PE파일이 메모리에 로딩되었을 때 각 섹션에서 메모리의 주소(RVA)와 파일 오프셋(RAW) 매핑

RAW - PointerToRawData = RVA - VirtualAddress

파일 오프셋 - 파일에서 섹션의 시작 위치 = 메모리의 주소 - 메모리에서 섹션의 시작 위치

5. IAT (Import Address Table)

프로그램이 어떤 라이브러리에서 어떤 함수를 사용하고 있는지 기술한 테이블

5.1. DLL (Dynamic Linked Library)

16비트 DOS : 라이브러리에서 해당 함수의 binary 코드를 프로그램에 그대로 삽입

32비트 WIN (멀티태스킹 지원) : 프로그램마다 동일한 라이브러리를 포함 → 메모리 낭비

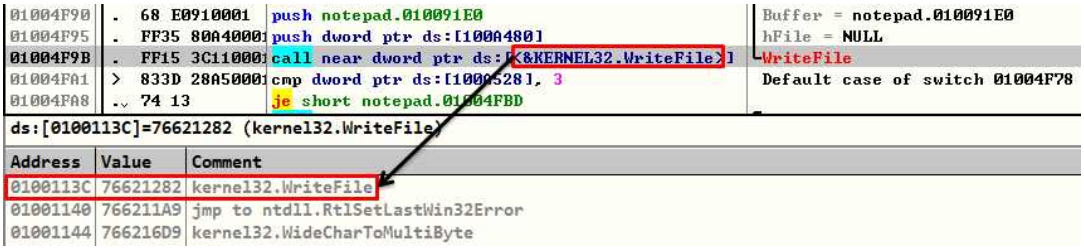
- 프로그램에 포함시키지 말고 별도의 파일로 구성하여 필요할 때 불러 사용
- 한 번 로딩된 DLL의 코드, 리소스는 Memory Mapping 기술로 여러 프로세스에서 공유
- 라이브러리가 업데이트되었을 때 해당 DLL 파일만 교체

Explicit Linking : 프로그램에서 사용되는 순간 로딩하고 사용이 끝나면 메모리에서 해제

Implicit Linking : 프로그램 시작할 때 같이 로딩되어 종료할 때 메모리에서 해제 (IAT)

* API 호출 방식

파일이 실행되는 순가 PE 로더가 0100113C의 위치에 WriteFile의 주소를 입력



CALL 76621282라고 하지 않는 이유?

- 프로그램을 컴파일하는 순간, 어떤 환경에서 실행될지 알 수 없고, 환경에 따라 DLL의 버전과 함수의 위치(주소)가 달라짐
- DLL Relocation (ImageBase에 이미 다른 DLL이 있을 때)

5.2. IMAGE_IMPORT_DESCRIPTOR : 어떤 라이브러리를 Import하고 있는지

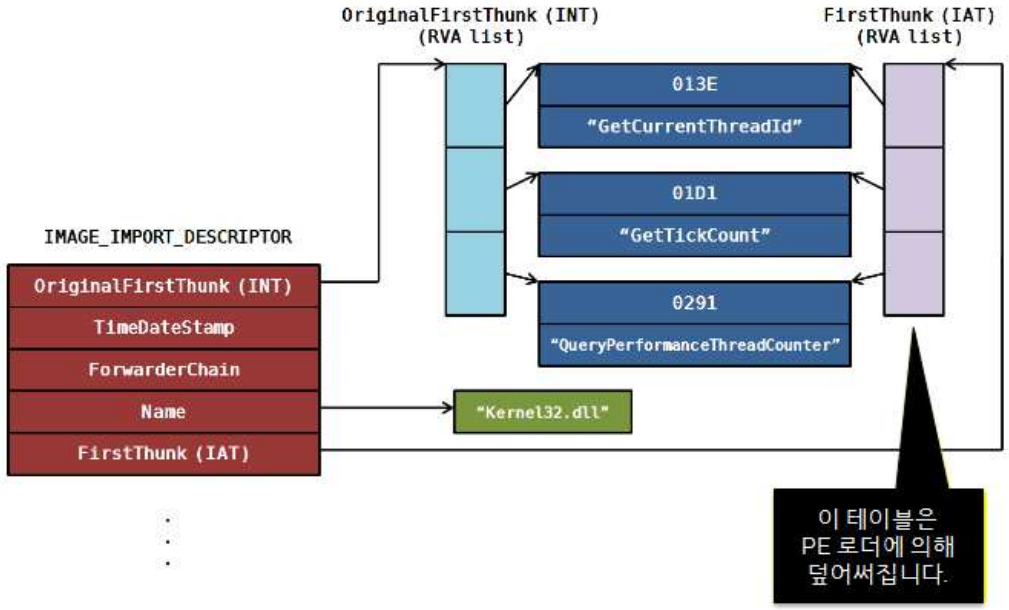
라이브러리 개수만큼 구조체의 배열 형식으로 존재, 배열의 마지막은 NULL

IMAGE_IMPORT_DESCRIPTOR 구조체 (크기 14h)

```
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;
```

- INT와 IAT는 long type(4바이트 자료형)배열이고 NULL로 끝남
- INT와 IAT의 크기는 같아야함
- INT에서 각 원소의 값은 IMAGE_IMPORT_BY_NAME 구조체 포인터

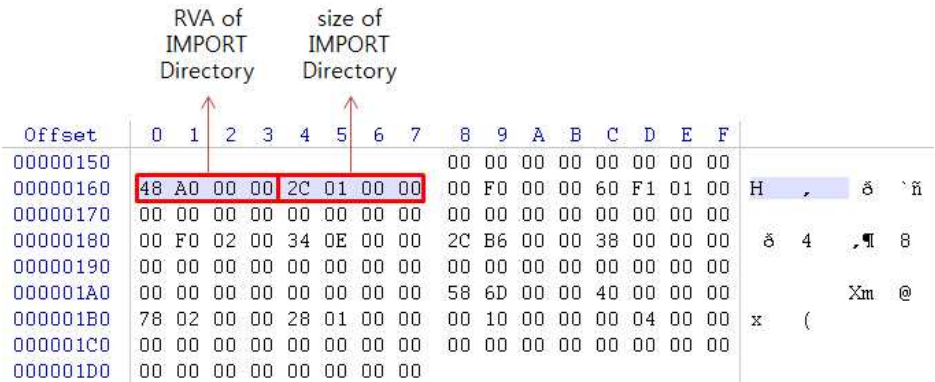


PE 로더가 임포트 함수 주소를 IAT에 입력하는 순서

1. IID의 Name 멤버를 읽어서 라이브러리의 이름 문자열("kernel32.dll")을 얻음
2. 해당 라이브러리를 로딩 → LoadLibrary("kernel32.dll")
3. IID의 OriginalFirstThunk 멤버를 읽어서 INT 주소를 얻음
4. INT에서 배열의 값을 하나씩 읽어 해당 IMAGE_IMPORT_BY_NAME 주소를 얻음
5. IMAGE_IMPORT_BY_NAME의 Hint(ordinal) 또는 Name 항목을 이용하여 해당 함수 시작 주소를 얻음 → GetProcAddress("GetCurrentThreadid")
6. IID의 FirstThunk(IAT) 멤버를 읽어서 IAT 주소를 얻음
7. 해당 IAT 배열 값에 위에서 구한 함수 주소를 입력
8. INT가 끝날 때까지(NULL을 만날 때까지) 위 4~7 과정을 반복

5.3. notepad.exe에서의 IAT

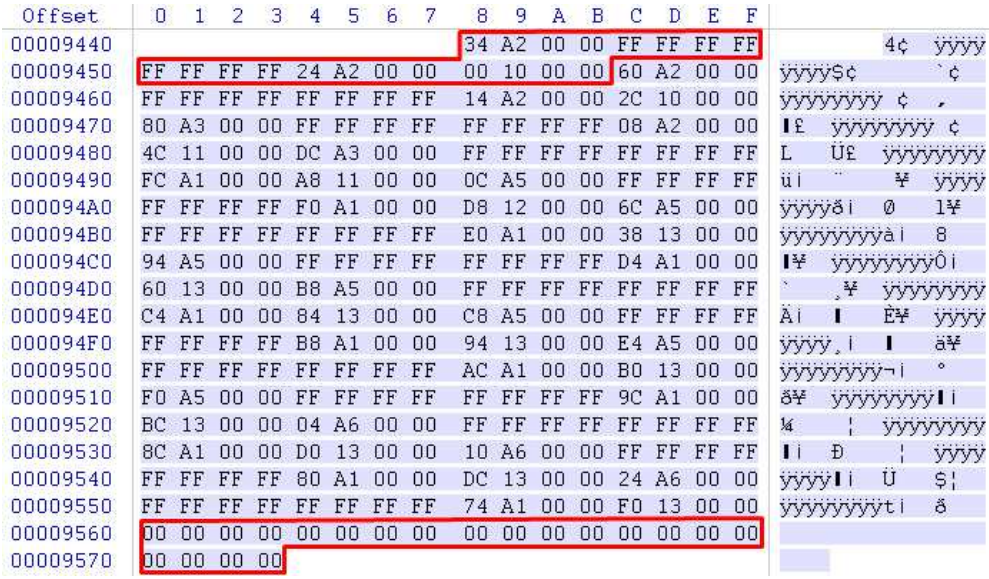
IMAGE_OPTIONAL_HEADER32.DataDirectory[1] = IMPORT Directory



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000150									00	00	00	00	00	00	00	00
00000160	48	A0	00	00	2C	01	00	00	00	F0	00	00	60	F1	01	00
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000180	00	F0	02	00	34	0E	00	00	2C	B6	00	00	38	00	00	00
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001A0	00	00	00	00	00	00	00	00	58	6D	00	00	40	00	00	00
000001B0	78	02	00	00	28	01	00	00	00	10	00	00	00	04	00	00
000001C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001D0	00	00	00	00	00	00	00	00								

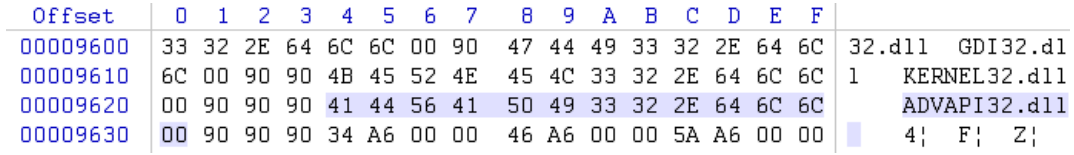
RAW = RVA - VirutalAddress + PointerToRawData
= A048 - 1000 + 400
= 9448 (~ 9574)

표시된 부분이 전부 IID구조체 배열이고 박스로 되어 있는 부분이 첫 번째 원소와 마지막 원소(NULL)



첫 번째 원소의 IMAGE_IMPORT_DESCRIPTOR 구조체			
File Offset	**Member**	**RVA**	**RAW**
9448	OriginalFirstThunk (INT)	0000A234	00009634
944C	TimeDateStamp	FFFFFFFF	-
9450	ForwarderChain	FFFFFFFF	-
9454	Name	0000A224	00009624
9458	FirstThunk (IAT)	00001000	00000400

① 라이브러리 이름 (Name)
Name 멤버 RVA:A224 → RAW:9624를 따라가면 "ADVAPI32.dll"임을 알 수 있음



② OriginalFirstThunk - INT(Import Name Table)
OriginalFirstThunk 멤버 RVA:A234 → RAW:9634를 따라가면 아래와 같음

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00009630	00	90	90	90	34	A6	00	00	46	A6	00	00	5A	A6	00	00	4¦ F¦ Z¦
00009640	68	A6	00	00	78	A6	00	00	88	A6	00	00	98	A6	00	00	h¦ x¦ ▌¦ ▌¦
00009650	AE	A6	00	00	C4	A6	00	00	D4	A6	00	00	00	00	00	00	®¦ Ä¦ Ô¦

주소 값 하나하나가 각각의 IMAGE_IMPORT_BY_NAME 구조체를 가리킴(마지막은 NULL)

③ IMAGE_IMPORT_BY_NAME

위에서 찾은 첫 번째 주소 RVA:A634 → RAW:9A34를 따라가면 Ordinal(027E)와 Name ("RegSetValueExW")를 찾을 수 있음

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00009A20	00	00	00	00	E6	B5	00	00	FE	B5	00	00	1A	B6	00	00	æµ þµ ¶
00009A30	00	00	00	00	7E	02	52	65	67	53	65	74	56	61	6C	75	~ RegSetValu
00009A40	65	45	78	57	00	00	6E	02	52	65	67	51	75	65	72	79	eExW n RegQuery

④ FirstThunk - IAT(Import Address Table)

FirstThunk 멤버 RVA:1000 → RAW:400을 따라가면 아래와 같음

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000400	82	1C	C7	77	D5	BC	C7	77	D4	BE	C7	77	C0	1C	C7	77	▌ ÇwÕ¼ÇwÔ¾ÇwÀ Çw
00000410	C4	BE	C7	77	E4	BE	C7	77	61	9A	C7	77	25	D2	C6	77	Ä¾Çwä¾Çwa▌Çw%ÒÆw
00000420	0D	D2	C6	77	F5	D1	C6	77	00	00	00	00	F9	29	E1	77	ÒÆwõÑÆw ù)áw

"RegSetValueExW" 함수의 주소는 77C71C82에 쓰임

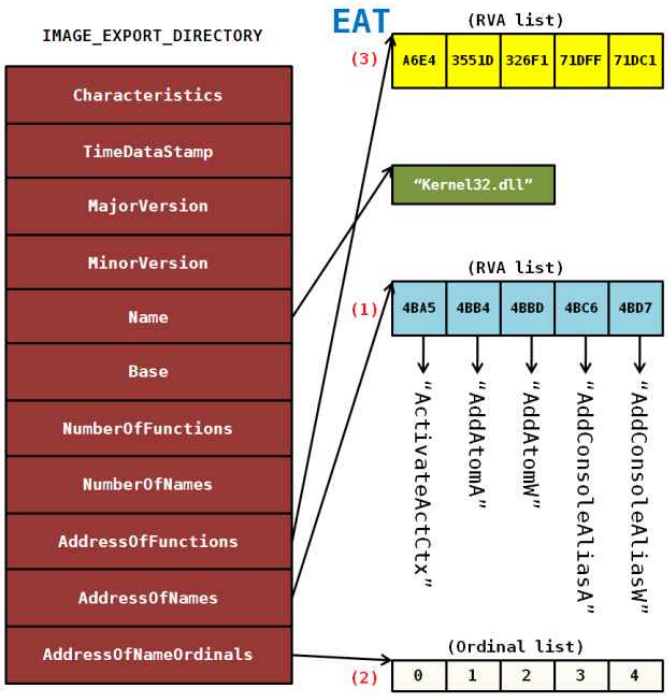
6. EAT (Export Address Table)

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

6.1. IMAGE_EXPORT_DIRECTORY

IMAGE_EXPORT_DIRECTORY 구조체 (크기 28h)

```
typedef struct _IMAGE_EXPORT_DIRECTORY {
        DWORD Characteristics;
        DWORD TimeDateStamp;
        WORD  MajorVersion;
        WORD  MinorVersion;
        DWORD Name;                 // 라이브러리 파일 이름의 주소
        DWORD Base;
        DWORD NumberOfFunctions;    // 실제 Export 함수 개수
        DWORD NumberOFNames;        // Export 함수 중 이름이 있는 함수 개수
        DWORD AddressOfFunctions;   // Export 함수 주소 배열
        DWORD AddressOFNames;       // 함수 이름 주소 배열
        DWORD AddressOfNameOrdinals;// Ordinal 주소 배열
} IMAGE_EXPORT_DIRECTORY, *PIMAGE_EXPORT_DIRECTORY;
```

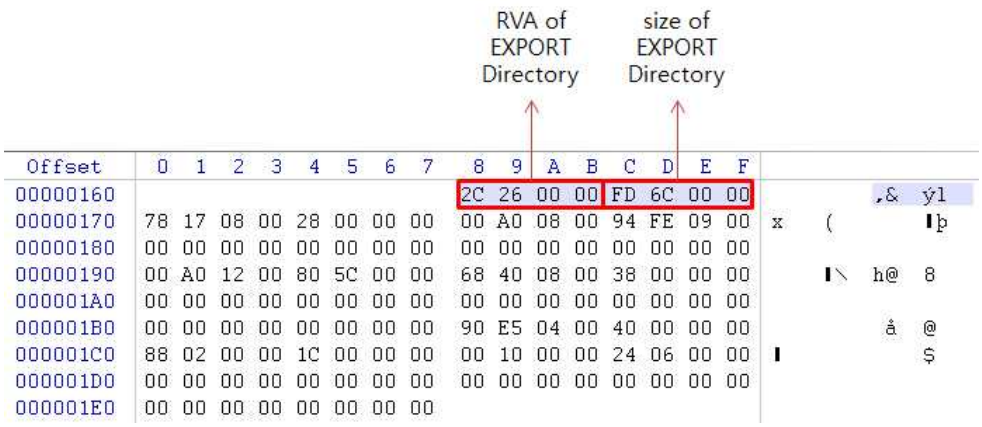


GetProcAddress() 동작 원리

1. AddressOfNames 멤버를 이용해 '함수 이름 배열'로 감
2. '함수 이름 배열'에서 문자열 비교(strcmp)를 통해 원하는 함수 이름을 찾음
3. AddressOfNameOrdinals 멤버를 이용해 'ordinal 배열'로 감
4. 'ordinal 배열'에서 name_index로 해당 ordinal 값을 찾음
5. AddressOfFunctions 멤버를 이용해 '함수 주소 배열(EAT)'로 감
6. '함수 주소 배열(EAT)'에서 ordinal_index로 원하는 함수의 시작 주소를 얻음

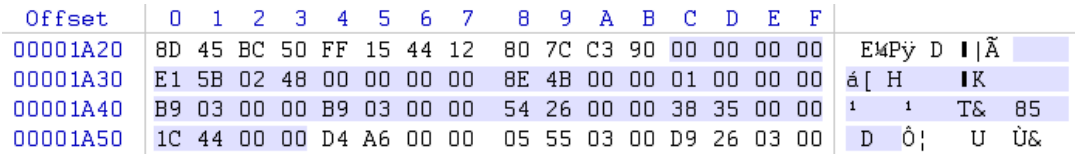
6.2. kernel32.dll에서의 EAT - AddAtomW 함수 주소 찾기

IMAGE_OPTIONAL_HEADER32.DataDirectory[0] = EXPORT Directory



RAW = RVA - VirutalAddress + PointerToRawData
= 262C - 1000 + 400
= 1A2C (~ 8729)

표시된 부분이 IMAGE_EXPORT_DIRECTORY 구조체



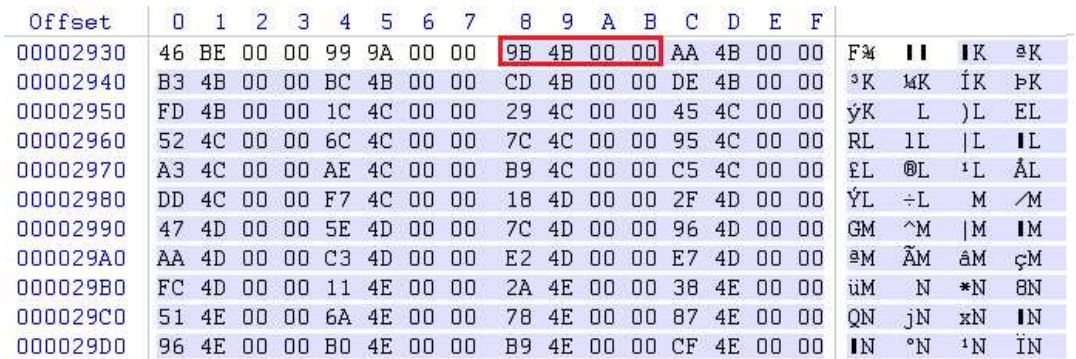
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00001A20	8D	45	BC	50	FF	15	44	12	80	7C	C3	90	00	00	00	00
00001A30	E1	5B	02	48	00	00	00	00	8E	4B	00	00	01	00	00	00
00001A40	B9	03	00	00	B9	03	00	00	54	26	00	00	38	35	00	00
00001A50	1C	44	00	00	D4	A6	00	00	05	55	03	00	D9	26	03	00

IMAGE_EXPORT_DIRECTORY 구조체

File Offset	Member	Value	RAW
1A2C	Characteristics	00000000	-
1A30	TimeDateStamp	48025BE1	-
1A34	MajorVersion	0000	-
1A36	MinorVersion	0000	-
1A38	Name	00004B8E	3F8E
1A3C	Base	00000001	-
1A40	NumberOfFunctions	000003B9	-
1A44	NumberOfNames	000003B9	-
1A48	AddressOfFunctions	00002654	1A54
1A4C	AddressOfNames	00003538	2938
1A50	AddressOFNameOrdinals	0000441C	381C

① 함수 이름 배열

AddressOfNames 멤버 RVA:3538 → RAW:2938을 따라가면 함수 이름 주소 배열이 나타남. 개수는 NumberOfNames인 3B9개



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00002930	46	BE	00	00	99	9A	00	00	9B	4B	00	00	AA	4B	00	00
00002940	B3	4B	00	00	BC	4B	00	00	CD	4B	00	00	DE	4B	00	00
00002950	FD	4B	00	00	1C	4C	00	00	29	4C	00	00	45	4C	00	00
00002960	52	4C	00	00	6C	4C	00	00	7C	4C	00	00	95	4C	00	00
00002970	A3	4C	00	00	AE	4C	00	00	B9	4C	00	00	C5	4C	00	00
00002980	DD	4C	00	00	F7	4C	00	00	18	4D	00	00	2F	4D	00	00
00002990	47	4D	00	00	5E	4D	00	00	7C	4D	00	00	96	4D	00	00
000029A0	AA	4D	00	00	C3	4D	00	00	E2	4D	00	00	E7	4D	00	00
000029B0	FC	4D	00	00	11	4E	00	00	2A	4E	00	00	38	4E	00	00
000029C0	51	4E	00	00	6A	4E	00	00	78	4E	00	00	87	4E	00	00
000029D0	96	4E	00	00	B0	4E	00	00	B9	4E	00	00	CF	4E	00	00

② 원하는 함수 이름 찾기

찾은 함수 이름 주소 배열의 처음부터 RVA:4B9B → RAW:3F9B 순서대로 따라가면 "AddAtomW"는 3번째(name_index=2) 있는 것을 알 수 있음

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00003F80	B2	03	B3	03	B4	03	B5	03	B6	03	B7	03	B8	03	4B	45	² ³ ´ µ ¶ · ¸ KE
00003F90	52	4E	45	4C	33	32	2E	64	6C	6C	00	41	63	74	69	76	RNEL32.dll Activ
00003FA0	61	74	65	41	63	74	43	74	78	00	41	64	64	41	74	6F	ateActCtx AddAto
00003FB0	6D	41	00	41	64	64	41	74	6F	6D	57	00	41	64	64	43	mA AddAtomW AddC
00003FC0	6F	6E	73	6F	6C	65	41	6C	69	61	73	41	00	41	64	64	onsoleAliasA Add
00003FD0	43	6F	6E	73	6F	6C	65	41	6C	69	61	73	57	00	41	64	ConsoleAliasW Ad
00003FE0	64	4C	6F	63	61	6C	41	6C	74	65	72	6E	61	74	65	43	dLocalAlternateC
00003FF0	6F	6D	70	75	74	65	72	4E	61	6D	65	41	00	41	64	64	omputerNameA Add
00004000	4C	6F	63	61	6C	41	6C	74	65	72	6E	61	74	65	43	6F	LocalAlternateCo

③ Ordinal 배열

AddressOfNameOrdinals 멤버 RVA:441C → RAW:381C를 따라가면 ordinal 배열이 나타나고 ordinal_index = 2라는 것을 알 수 있음

AddressOfNameOrdinals[name_index] = ordinal

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00003810	DB	8F	00	00	E3	8F	00	00	EC	8F	00	00	00	00	01	00	Û ã ì
00003820	02	00	03	00	04	00	05	00	06	00	07	00	08	00	09	00	
00003830	0A	00	0B	00	0C	00	0D	00	0E	00	0F	00	10	00	11	00	
00003840	12	00	13	00	14	00	15	00	16	00	17	00	18	00	19	00	
00003850	1A	00	1B	00	1C	00	1D	00	1E	00	1F	00	20	00	21	00	!
00003860	22	00	23	00	24	00	25	00	26	00	27	00	28	00	29	00	" # $ % & ' ( )

④ 함수 주소 배열 - EAT

AddressOfFunctions 멤버 RVA:2654 → RAW:1A54를 따라가면 EAT가 나타나고 RVA=000326D9라는 것을 알 수 있음

AddressOfFunctions[ordinal] = RVA

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00001A50	1C	44	00	00	D4	A6	00	00	05	55	03	00	D9	26	03	00	D Ô¦ U Ù&
00001A60	DF	1C	07	00	A1	1C	07	00	82	93	05	00	66	92	05	00	ß ¡ ▮▮ f'
00001A70	F9	BE	02	00	F5	8F	00	00	31	23	07	00	1A	F6	05	00	ù¾ õ 1# ö
00001A80	67	59	03	00	42	E4	02	00	19	25	07	00	CA	71	05	00	gY Bä % Êq
00001A90	B0	62	05	00	25	78	05	00	67	68	01	00	E6	CD	06	00	°b %x gh æÍ
00001AA0	6A	CE	06	00	A1	CC	06	00	1F	CC	06	00	5D	65	01	00	jÎ ¡Ì Ì]e
00001AB0	75	B3	02	00	D3	74	01	00	72	83	03	00	B0	51	01	00	u³ Ót r▮ °Q
00001AC0	A3	95	01	00	8F	7A	03	00	9B	0C	07	00	F8	0A	07	00	£▮ z ▮ ø
00001AD0	24	C0	02	00	CD	BE	06	00	9F	BE	06	00	FF	BE	06	00	$À Í¾ ▮¾ ÿ¾