Part III - Windows PE File Format Basics

Swapnil Pathak
Amit Malik

www.SecurityXploded.com

Disclaimer

Acknowledgement

- Special thanks to **null** & **Garage4Hackers** community for their extended support and cooperation.
- Thanks to all the trainers who have devoted their precious time and countless hours to make it happen.

Reversing & Malware Analysis Training

This presentation is part of our **Reverse Engineering & Malware Analysis** Training program. Currently it is delivered only during our local meet for FREE of cost.

For complete details of this course, visit our Security Training page.

Who am I #1

Amit Malik (sometimes DouBle_Zer0,DZZ)

- Member SecurityXploded
- Security Researcher @ McAfee Labs
- RE, Exploit Analysis/Development, Malware Analysis
- Email: m.amit30@gmail.com

Who am I #2

Swapnil Pathak

- Member SecurityXploded
- Security Researcher @ McAfee Labs
- RE, Malware Analysis, Network Security
- Email: swapnilpathak101@gmail.com

Course Q&A

- Keep yourself up to date with latest security news
  - http://www.securityphresh.com

- For Q&A, join our mailing list.
  - http://groups.google.com/group/securityxploded

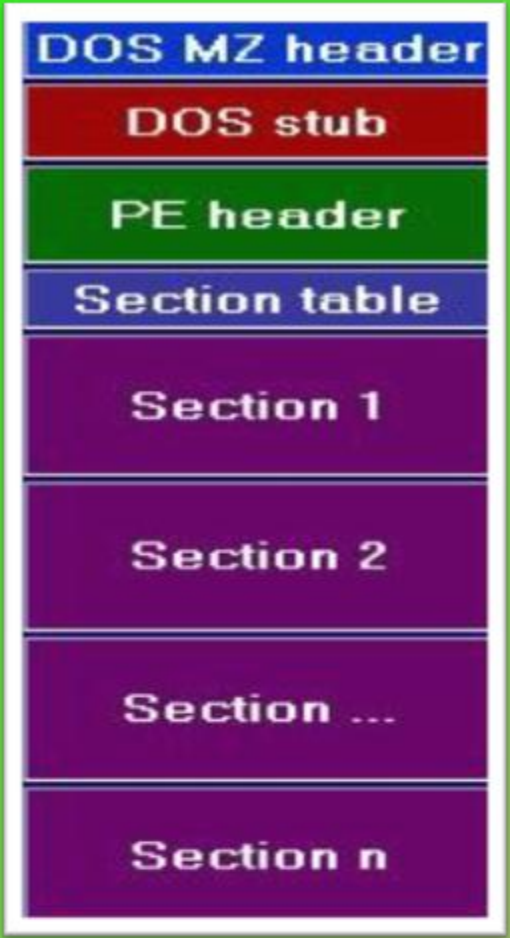
PE File Format

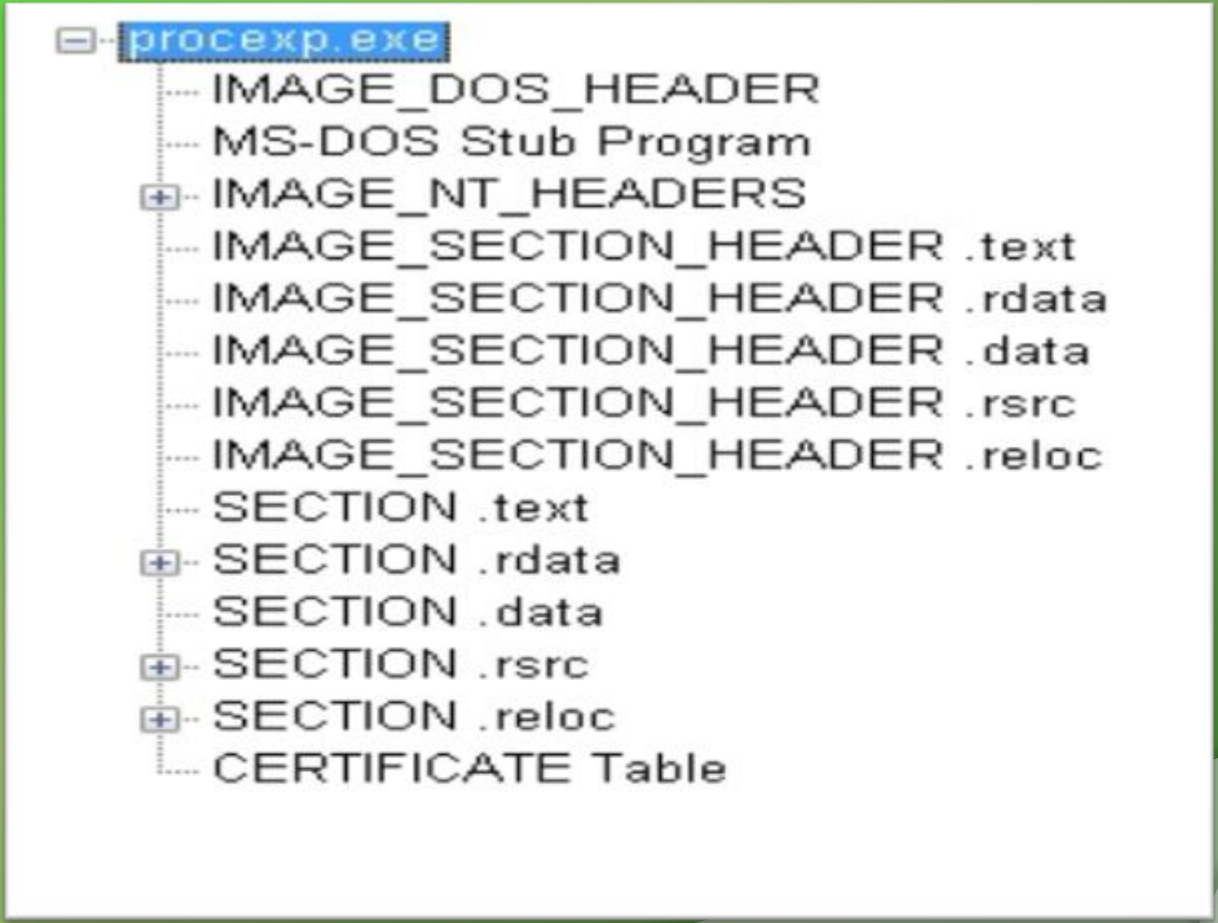
- PE – Portable Executable
- PE is the native Win32 file format.
- 32-bit DLL, COM, OCX, Control Panel Applets(.CPL), .NET, NT kernel mode drivers are all PE File Format.

Why PE File Format

- How windows loader loads the executable in memory.
- How loader build the import and export table for a module in memory
- From where to start the execution or Address of entry point
- Answer of the question "how binary compiled on a version of windows works on another version of windows?"
- Where should attacker attack ☺
- Also today's malwares are generally encrypted, packed. In order to rebuild the original binary we need to know how the binary is structured.

Basic Structure





```
⊟ procexp.exe
   ├ IMAGE_DOS_HEADER
   ├ MS-DOS Stub Program
   ⊞ IMAGE_NT_HEADERS
   ├ IMAGE_SECTION_HEADER .text
   ├ IMAGE_SECTION_HEADER .rdata
   ├ IMAGE_SECTION_HEADER .data
   ├ IMAGE_SECTION_HEADER .rsrc
   ├ IMAGE_SECTION_HEADER .reloc
   ├ SECTION .text
   ⊞ SECTION .rdata
   ├ SECTION .data
   ⊞ SECTION .rsrc
   ⊞ SECTION .reloc
   └ CERTIFICATE Table
```

Basic Structure Cont.

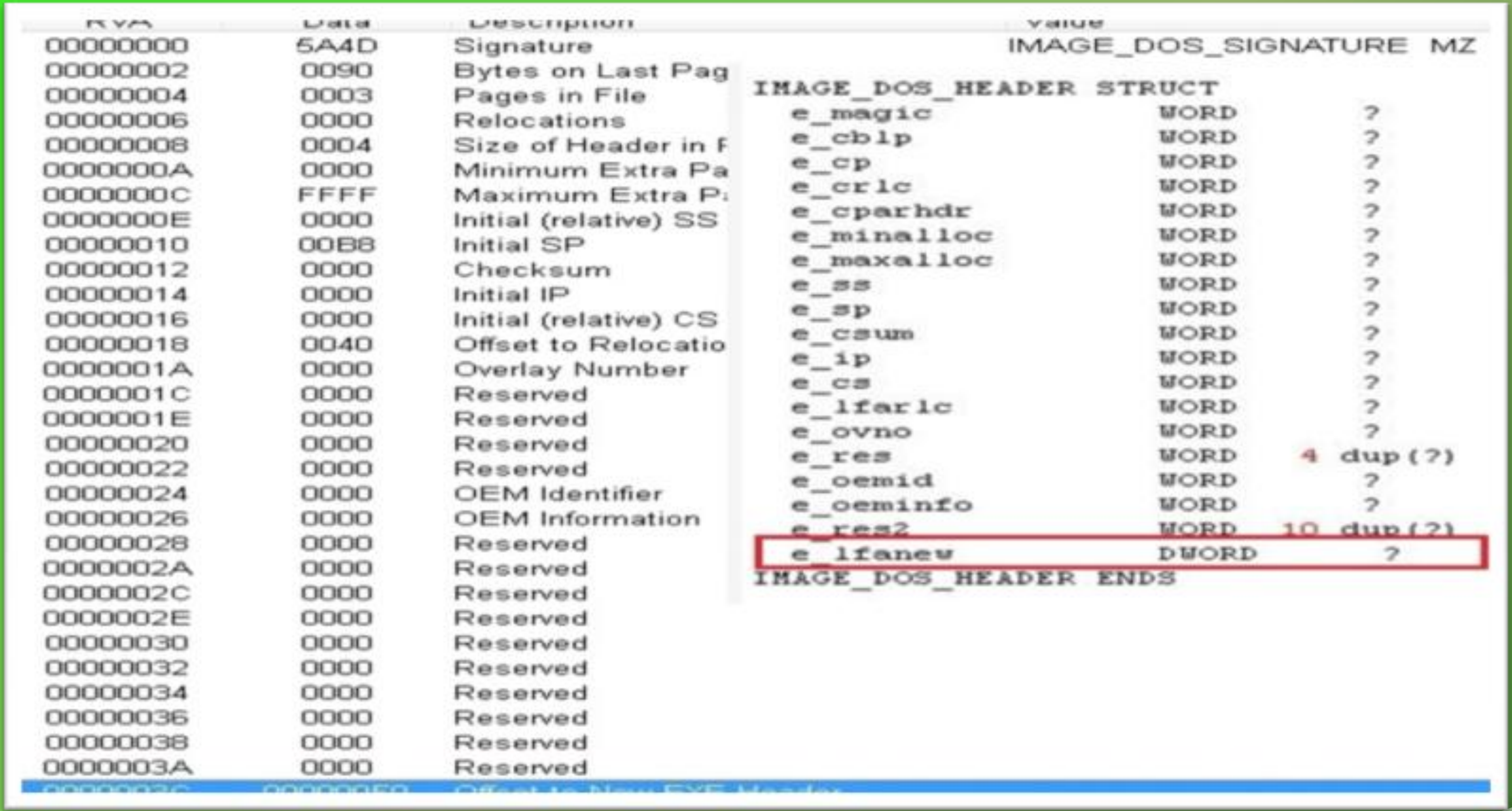
- Most common sections found in executable are
  - Executable Code section (.text , CODE)
  - Data Sections (.data, .rdata, .bss, DATA)
  - Resources section (.rsrc)
  - Export Section (.edata)
  - Import Section (.idata)
  - Debug Information Section (.debug)

Headers – DOS Header

- All PE files start with DOS header
- First 64 bytes of the file.
- Run program in DOS.
- Runs the DOS stub
- Usually the string

  “This program must be run under Microsoft Windows”
- **e_lfanew** is the pointer to PE or NT header
- Structure defined in windows.inc or winnt.h

Header- DOS header cont.



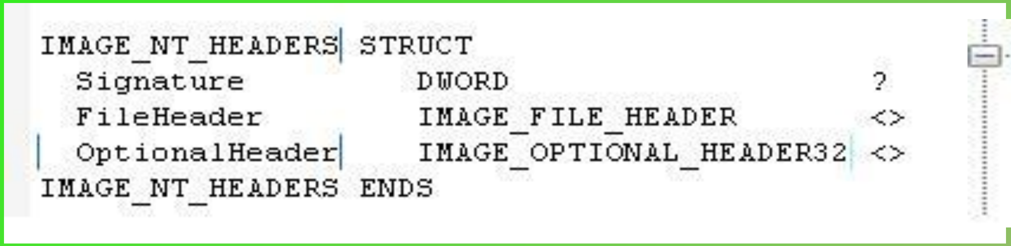
RVA	Data	Description	Value
00000000	5A4D	Signature	IMAGE_DOS_SIGNATURE MZ
00000002	0090	Bytes on Last Pag	
00000004	0003	Pages in File	
00000006	0000	Relocations	
00000008	0004	Size of Header in F	
0000000A	0000	Minimum Extra Pa	
0000000C	FFFF	Maximum Extra P:	
0000000E	0000	Initial (relative) SS	
00000010	00B8	Initial SP	
00000012	0000	Checksum	
00000014	0000	Initial IP	
00000016	0000	Initial (relative) CS	
00000018	0040	Offset to Relocatio	
0000001A	0000	Overlay Number	
0000001C	0000	Reserved	
0000001E	0000	Reserved	
00000020	0000	Reserved	
00000022	0000	Reserved	
00000024	0000	OEM Identifier	
00000026	0000	OEM Information	
00000028	0000	Reserved	
0000002A	0000	Reserved	
0000002C	0000	Reserved	
0000002E	0000	Reserved	
00000030	0000	Reserved	
00000032	0000	Reserved	
00000034	0000	Reserved	
00000036	0000	Reserved	
00000038	0000	Reserved	
0000003A	0000	Reserved	
0000003C	000000E0	Offset to New EXE Header	

```
IMAGE_DOS_HEADER STRUCT
  e_magic           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             WORD    4 dup(?)
  e_oemid           WORD      ?
  e_oeminfo         WORD      ?
  e_res2            WORD   10 dup(?)
  e_lfanew          DWORD     ?
IMAGE_DOS_HEADER ENDS
```

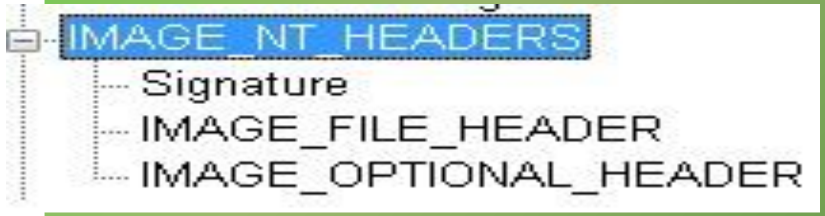
e_magic = 4D, 5A (MZ)

e_lfanew is a DWORD which contains the offset of the PE header

Headers – PE header



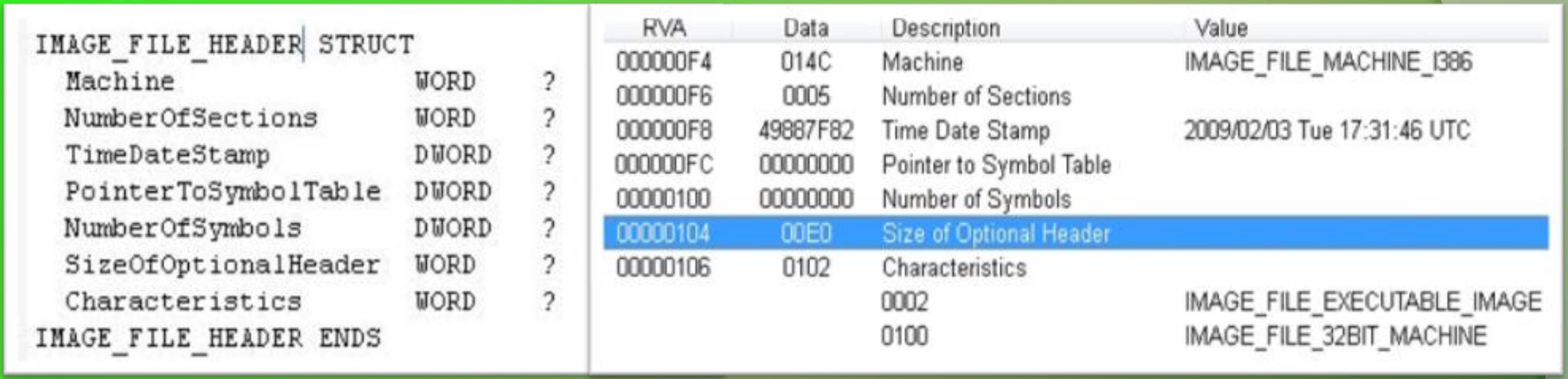
```
IMAGE_NT_HEADERS STRUCT
  Signature              DWORD                   ?
  FileHeader             IMAGE_FILE_HEADER       <>
  OptionalHeader         IMAGE_OPTIONAL_HEADER32 <>
IMAGE_NT_HEADERS ENDS
```



```
IMAGE_NT_HEADERS
  Signature
  IMAGE_FILE_HEADER
  IMAGE_OPTIONAL_HEADER
```

- Begins with signature (DWORD) 50h, 45h, 00h, 00h
- Letters “PE” followed by two terminating zeros
- File Header- 20 Bytes – contains info about physical layout and properties of the file
- Optional Header- 224 Bytes – contains info about the logical layout of the PE file – size given by member of File header

Headers – PE –> File header

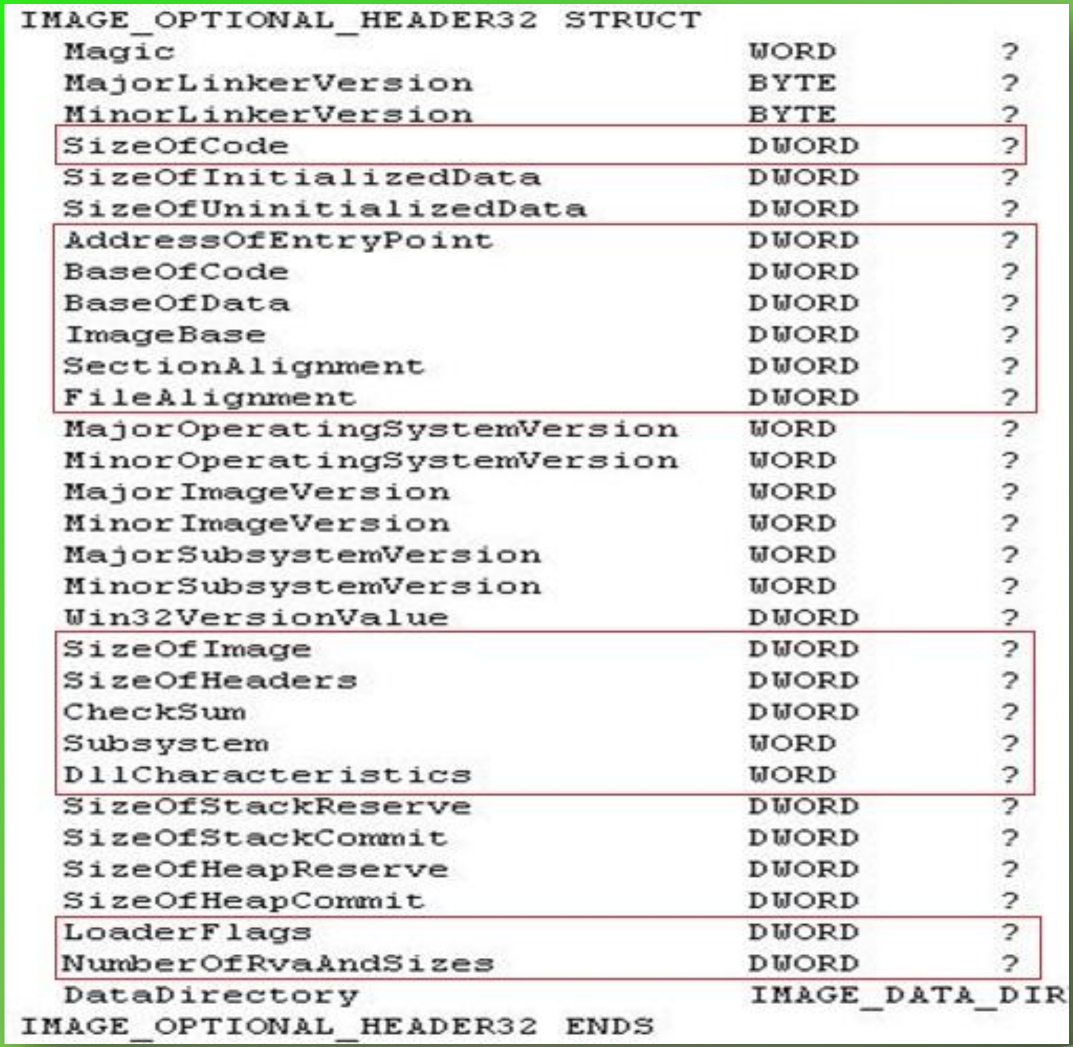


```
IMAGE_FILE_HEADER STRUCT
  Machine                 WORD    ?
  NumberOfSections        WORD    ?
  TimeDateStamp           DWORD   ?
  PointerToSymbolTable    DWORD   ?
  NumberOfSymbols         DWORD   ?
  SizeOfOptionalHeader    WORD    ?
  Characteristics         WORD    ?
IMAGE_FILE_HEADER ENDS
```

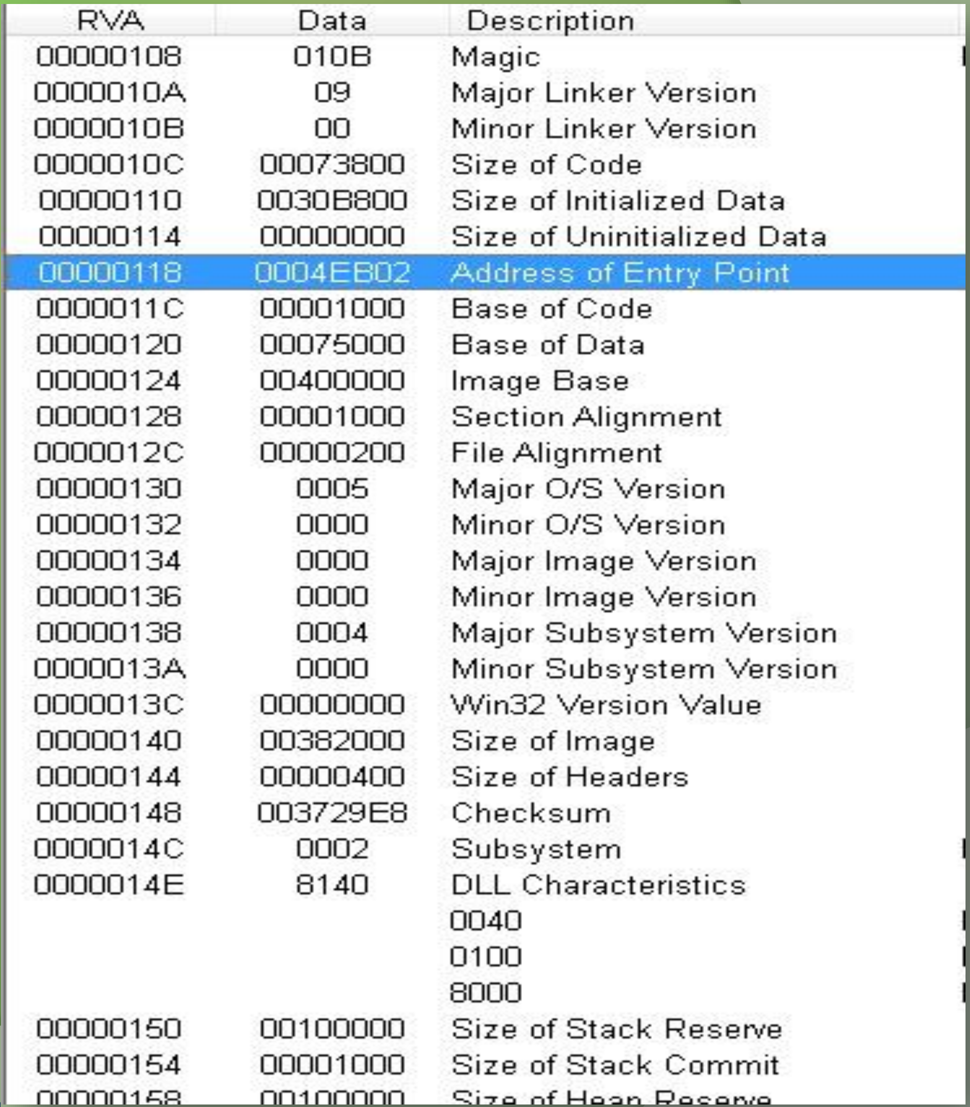
RVA	Data	Description	Value
000000F4	014C	Machine	IMAGE_FILE_MACHINE_I386
000000F6	0005	Number of Sections	
000000F8	49887F82	Time Date Stamp	2009/02/03 Tue 17:31:46 UTC
000000FC	00000000	Pointer to Symbol Table	
00000100	00000000	Number of Symbols	
00000104	00E0	Size of Optional Header	
00000106	0102	Characteristics	
		0002	IMAGE_FILE_EXECUTABLE_IMAGE
		0100	IMAGE_FILE_32BIT_MACHINE

- Machine
- NumberOfSections
- SizeOfOptionalHeader
- Characteristics

Header – PE –> Optional Header



```
IMAGE_OPTIONAL_HEADER32 STRUCT
  Magic                             WORD        ?
  MajorLinkerVersion                BYTE        ?
  MinorLinkerVersion                BYTE        ?
  SizeOfCode                        DWORD       ?
  SizeOfInitializedData             DWORD       ?
  SizeOfUninitializedData           DWORD       ?
  AddressOfEntryPoint               DWORD       ?
  BaseOfCode                        DWORD       ?
  BaseOfData                        DWORD       ?
  ImageBase                         DWORD       ?
  SectionAlignment                  DWORD       ?
  FileAlignment                     DWORD       ?
  MajorOperatingSystemVersion       WORD        ?
  MinorOperatingSystemVersion       WORD        ?
  MajorImageVersion                 WORD        ?
  MinorImageVersion                 WORD        ?
  MajorSubsystemVersion             WORD        ?
  MinorSubsystemVersion             WORD        ?
  Win32VersionValue                 DWORD       ?
  SizeOfImage                       DWORD       ?
  SizeOfHeaders                     DWORD       ?
  CheckSum                          DWORD       ?
  Subsystem                         WORD        ?
  DllCharacteristics                WORD        ?
  SizeOfStackReserve                DWORD       ?
  SizeOfStackCommit                 DWORD       ?
  SizeOfHeapReserve                 DWORD       ?
  SizeOfHeapCommit                  DWORD       ?
  LoaderFlags                       DWORD       ?
  NumberOfRvaAndSizes               DWORD       ?
  DataDirectory                     IMAGE_DATA_DIR
IMAGE_OPTIONAL_HEADER32 ENDS
```

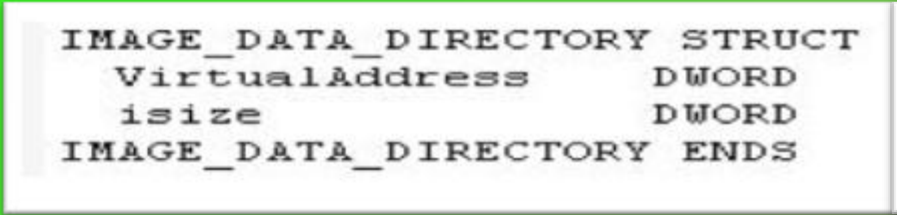


RVA	Data	Description
00000108	010B	Magic
0000010A	09	Major Linker Version
0000010B	00	Minor Linker Version
0000010C	00073800	Size of Code
00000110	0030B800	Size of Initialized Data
00000114	00000000	Size of Uninitialized Data
00000118	0004EB02	Address of Entry Point
0000011C	00001000	Base of Code
00000120	00075000	Base of Data
00000124	00400000	Image Base
00000128	00001000	Section Alignment
0000012C	00000200	File Alignment
00000130	0005	Major O/S Version
00000132	0000	Minor O/S Version
00000134	0000	Major Image Version
00000136	0000	Minor Image Version
00000138	0004	Major Subsystem Version
0000013A	0000	Minor Subsystem Version
0000013C	00000000	Win32 Version Value
00000140	00382000	Size of Image
00000144	00000400	Size of Headers
00000148	003729E8	Checksum
0000014C	0002	Subsystem
0000014E	8140	DLL Characteristics
		0040
		0100
		8000
00000150	00100000	Size of Stack Reserve
00000154	00001000	Size of Stack Commit
00000158	00100000	Size of Heap Reserve

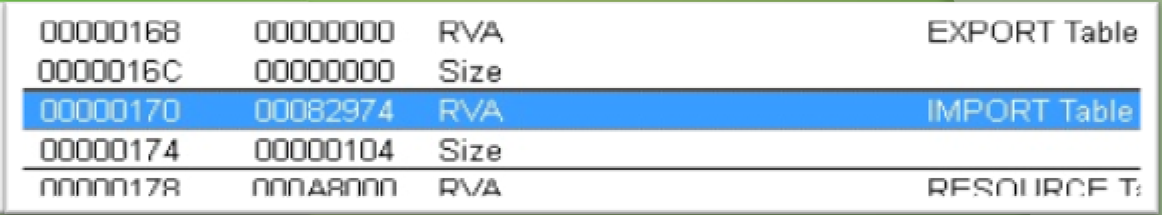
Optional Header Cont.

- AddressOfEntryPoint
- ImageBase
- SectionAlignment
- FileAlignment
- SizeOfImage
- SizeOfHeaders
- Subsystem
- DataDirectory

Header – PE –> Optional –> Data Directory



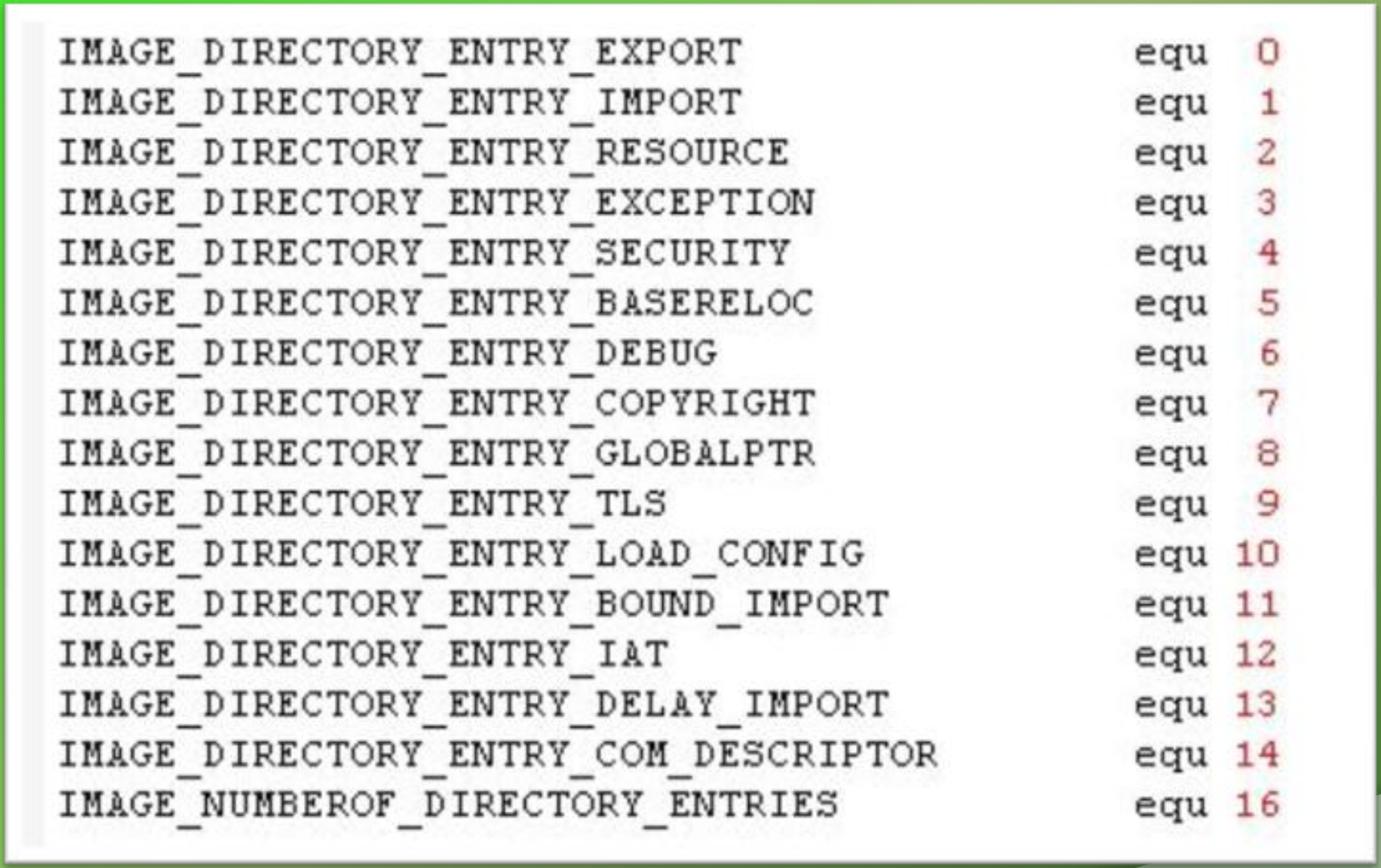
```
IMAGE_DATA_DIRECTORY STRUCT
  VirtualAddress       DWORD
  isize                DWORD
IMAGE_DATA_DIRECTORY ENDS
```



```
00000168    00000000    RVA     EXPORT Table
0000016C    00000000    Size
00000170    00082974    RVA     IMPORT Table
00000174    00000104    Size
00000178    000A8000    RVA     RESOURCE T
```

- Last 128 bytes of OptionalHeader
- Array of 16 Image_Data_Directory structures
- Each relating to an important data structure like the Import Table
- Members
- Virtual Address : RVA of the data structure
- iSize : size in bytes of the data structure

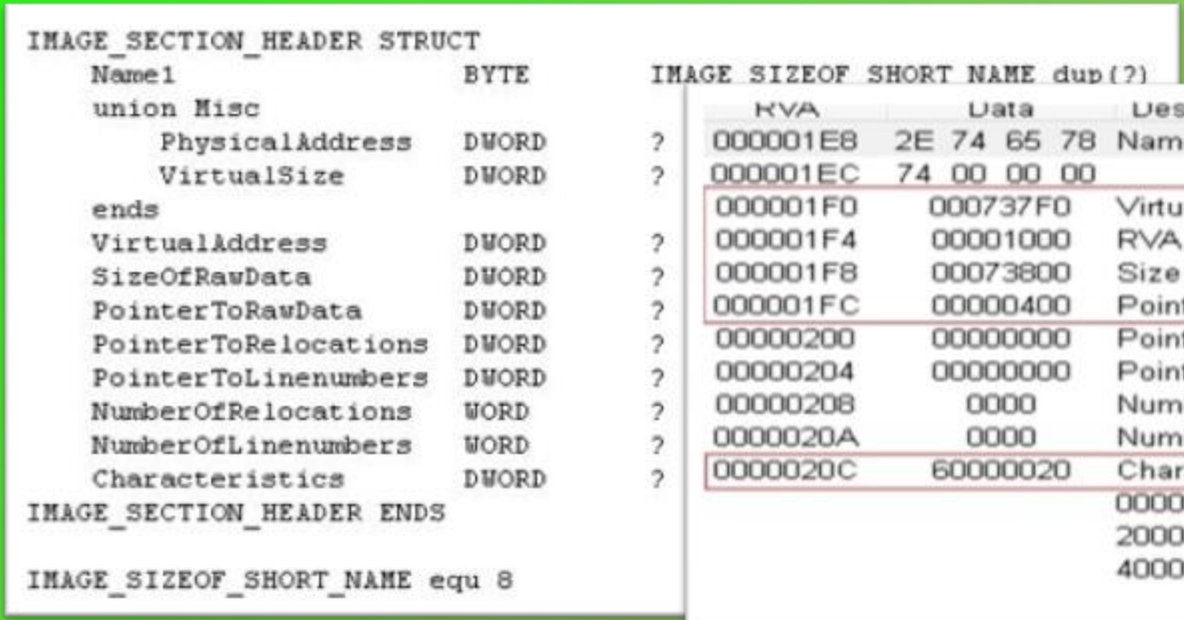
Data Directories



```
IMAGE_DIRECTORY_ENTRY_EXPORT                equ  0
IMAGE_DIRECTORY_ENTRY_IMPORT                equ  1
IMAGE_DIRECTORY_ENTRY_RESOURCE              equ  2
IMAGE_DIRECTORY_ENTRY_EXCEPTION             equ  3
IMAGE_DIRECTORY_ENTRY_SECURITY              equ  4
IMAGE_DIRECTORY_ENTRY_BASERELOC             equ  5
IMAGE_DIRECTORY_ENTRY_DEBUG                 equ  6
IMAGE_DIRECTORY_ENTRY_COPYRIGHT             equ  7
IMAGE_DIRECTORY_ENTRY_GLOBALPTR             equ  8
IMAGE_DIRECTORY_ENTRY_TLS                   equ  9
IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG           equ 10
IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT          equ 11
IMAGE_DIRECTORY_ENTRY_IAT                   equ 12
IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT          equ 13
IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR        equ 14
IMAGE_NUMBEROF_DIRECTORY_ENTRIES            equ 16
```

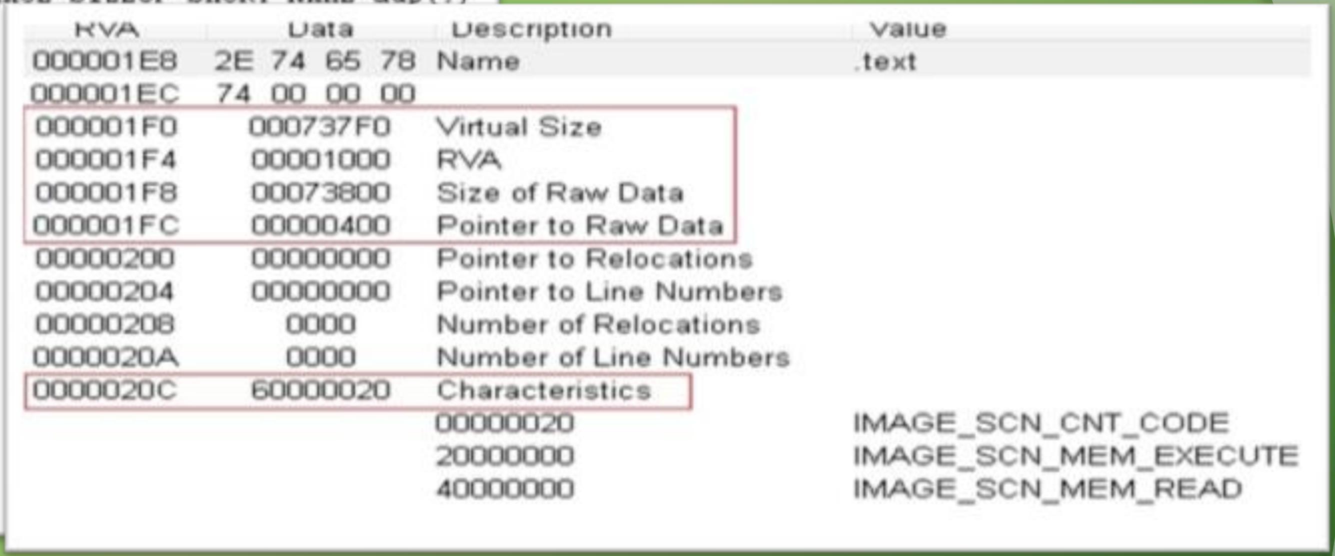
- IMAGE_DIRECTORY_ENTRY_EXPORT
- IMAGE_DIRECTORY_ENTRY_IMPORT
- IMAGE_DIRECTORY_ENTRY_RESOURCE
- IMAGE_DIRECTORY_ENTRY_TLS
- IMAGE_DIRECTORY_ENTRY_IAT

Headers - Section Header



```
IMAGE_SECTION_HEADER STRUCT
    Name1                BYTE        IMAGE_SIZEOF_SHORT_NAME dup(?)
    union Misc
        PhysicalAddress  DWORD       ?
        VirtualSize      DWORD       ?
    ends
    VirtualAddress       DWORD       ?
    SizeOfRawData        DWORD       ?
    PointerToRawData     DWORD       ?
    PointerToRelocations DWORD       ?
    PointerToLinenumbers DWORD       ?
    NumberOfRelocations  WORD        ?
    NumberOfLinenumbers  WORD        ?
    Characteristics      DWORD       ?
IMAGE_SECTION_HEADER ENDS

IMAGE_SIZEOF_SHORT_NAME equ 8
```



RVA	Data	Description	Value
000001E8	2E 74 65 78	Name	.text
000001EC	74 00 00 00		
000001F0	000737F0	Virtual Size	
000001F4	00001000	RVA	
000001F8	00073800	Size of Raw Data	
000001FC	00000400	Pointer to Raw Data	
00000200	00000000	Pointer to Relocations	
00000204	00000000	Pointer to Line Numbers	
00000208	0000	Number of Relocations	
0000020A	0000	Number of Line Numbers	
0000020C	60000020	Characteristics	
		00000020	IMAGE_SCN_CNT_CODE
		20000000	IMAGE_SCN_MEM_EXECUTE
		40000000	IMAGE_SCN_MEM_READ

- Array of IMAGE_SECTION_HEADER
- Equal to the numberofsections – FileHeader member.
- Each structure size = 40 bytes

Section Header cont.

- Name – Virtually can be anything in text
- VirtualSize – Size of section in memory
- VirtualAddress – section entry offset in memory (RVA)
- SizeOfRawData – Size of section on disk
- PointerToRawData – section entry offset on disk
- Characteristics – Type of section (execuatble, data etc.)
- Section Alignment and File Alignment are two important values from optional header that control the entry point of next section.

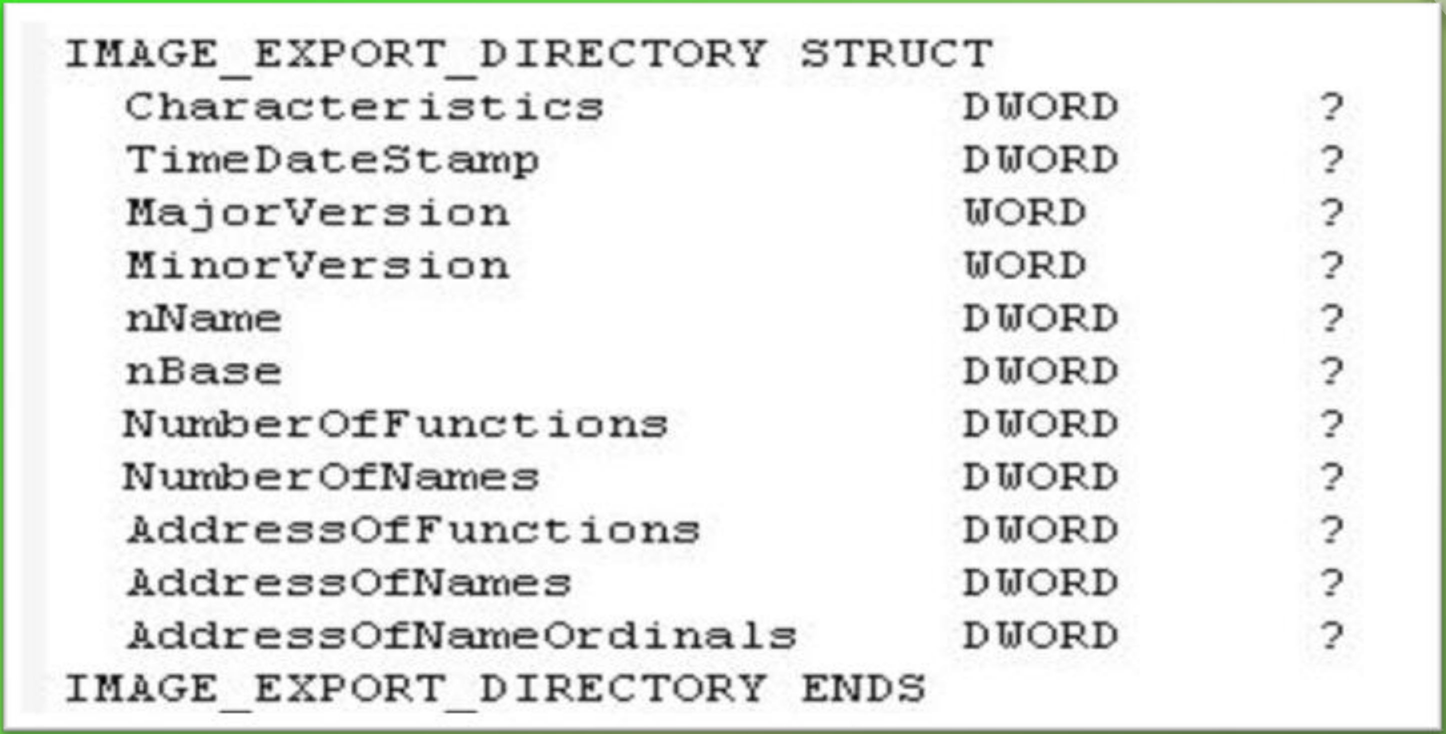
- The structure of PE file on disk is exactly the same as when it is loaded into memory.
- The windows loader maps the required sections in memory.
- When sections are loaded into memory they are aligned to fit 4KB memory pages (Section Alignment), each section starting on a new page.

Type of PE file sections

- Executable code
- Data
- Resources
- Export section
- Import section
- Thread Local Storage (TLS)
- Base Relocations (reloc.)

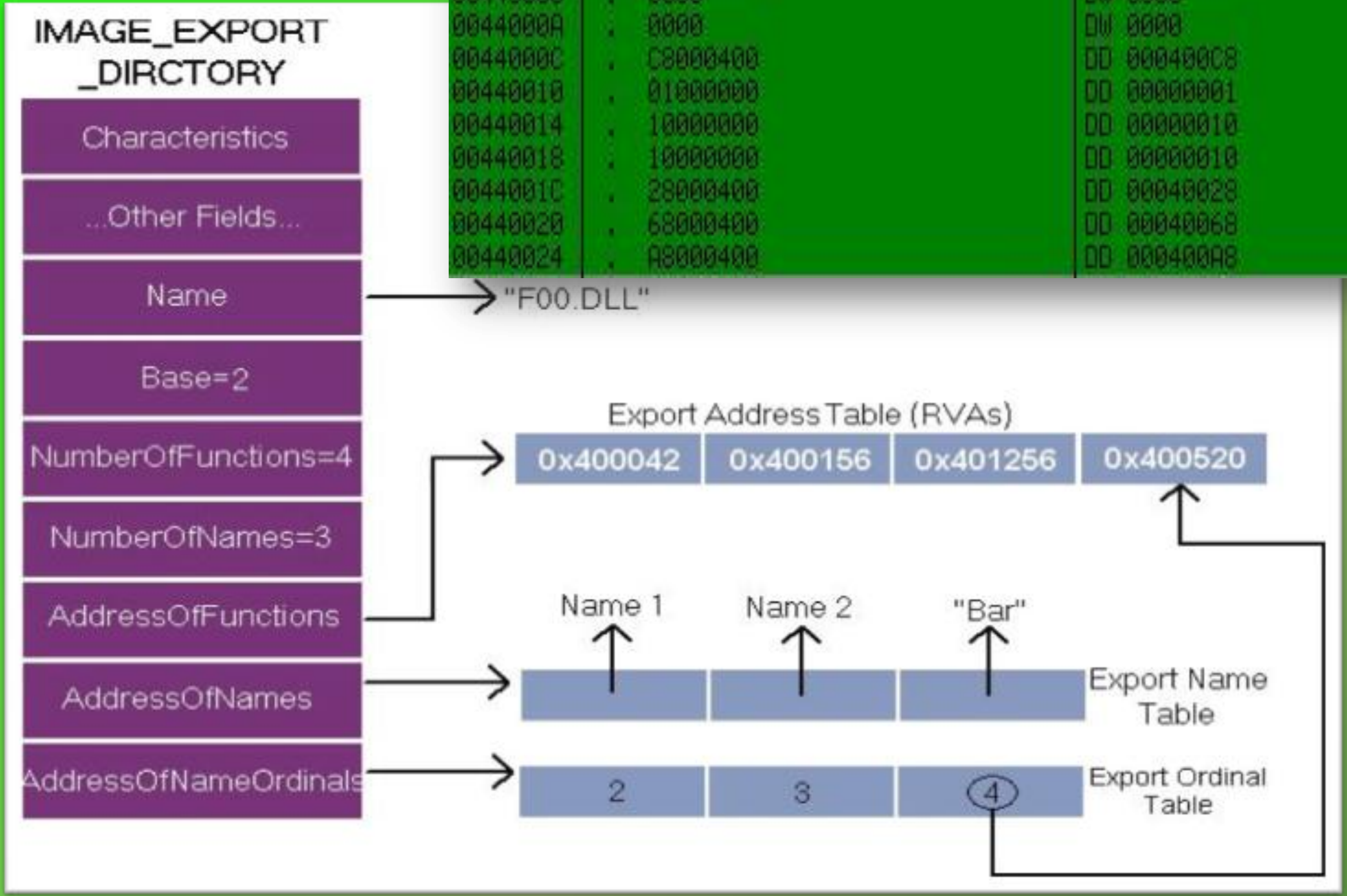
Export Section

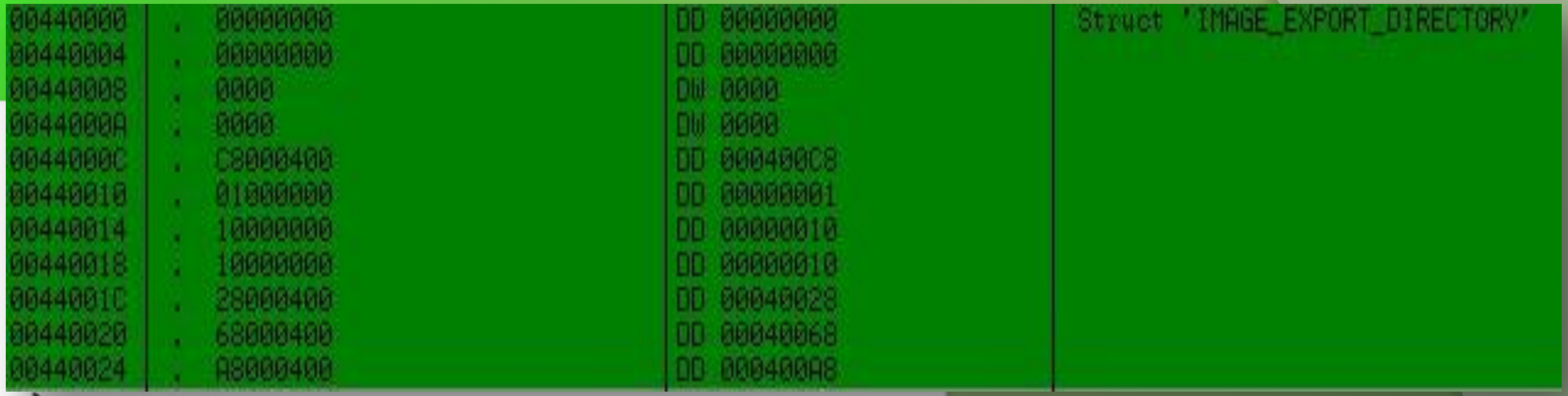
- Relevant to DLLs
- Export functions in two ways
  - By name
  - By ordinal only
- Ordinal – 16 bit value that uniquely defines a function in particular DLL



```
IMAGE_EXPORT_DIRECTORY STRUCT
  Characteristics              DWORD     ?
  TimeDateStamp                DWORD     ?
  MajorVersion                 WORD      ?
  MinorVersion                 WORD      ?
  nName                        DWORD     ?
  nBase                        DWORD     ?
  NumberOfFunctions            DWORD     ?
  NumberOfNames                DWORD     ?
  AddressOfFunctions           DWORD     ?
  AddressOfNames               DWORD     ?
  AddressOfNameOrdinals        DWORD     ?
IMAGE_EXPORT_DIRECTORY ENDS
```

nName
nBase
NumberOfFunctions
NumberOfNames
AddressOfFunctions
AddressOfNames
AddressOfNameOrdinals





```
00440000 | . 00000000          DD 00000000      Struct 'IMAGE_EXPORT_DIRECTORY'
00440004 | . 00000000          DD 00000000
00440008 | . 0000              DW 0000
0044000A | . 0000              DW 0000
0044000C | . C8000400          DD 000400C8
00440010 | . 01000000          DD 00000001
00440014 | . 10000000          DD 00000010
00440018 | . 10000000          DD 00000010
0044001C | . 28000400          DD 00040028
00440020 | . 68000400          DD 00040068
00440024 | . A8000400          DD 000400A8
```

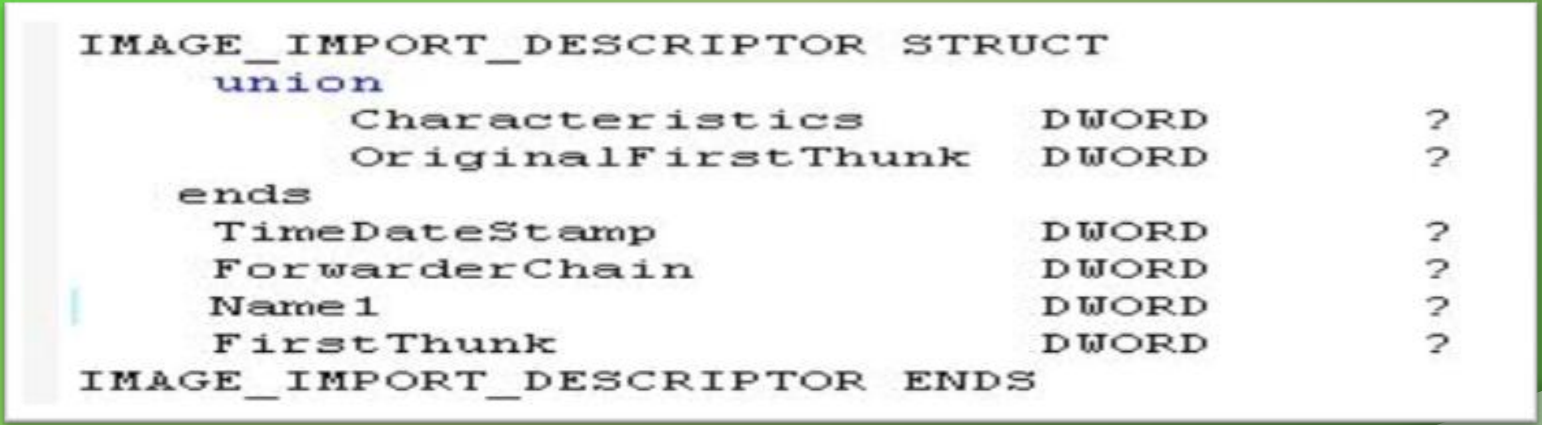
Export by Ordinal only
Export Forwarding

Import Section

- Contains information about all functions imported by executable from DLLs
- Loader maps all the DLLs used by the application into its address space
- Finds the addresses of all the imported functions and makes them available to the executable being loaded.

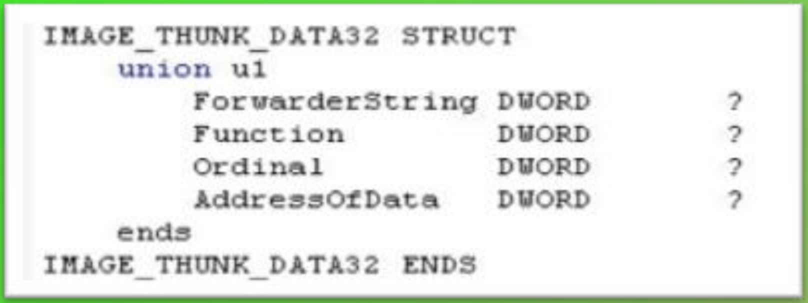
Import Directory

- 20 byte structure IMAGE_IMPORT_DESCRIPTOR
- Number of structures = Number of DLLs imported
- Last structure filed with zeros

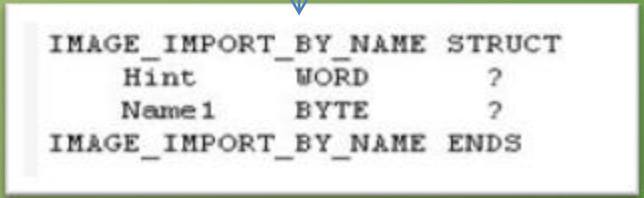


```
IMAGE_IMPORT_DESCRIPTOR STRUCT
    union
        Characteristics         DWORD       ?
        OriginalFirstThunk      DWORD       ?
   ends
    TimeDateStamp               DWORD       ?
    ForwarderChain              DWORD       ?
    Name1                       DWORD       ?
    FirstThunk                  DWORD       ?
IMAGE_IMPORT_DESCRIPTOR ENDS
```

- OriginalFirstThunk
- Name1
- FirstThunk



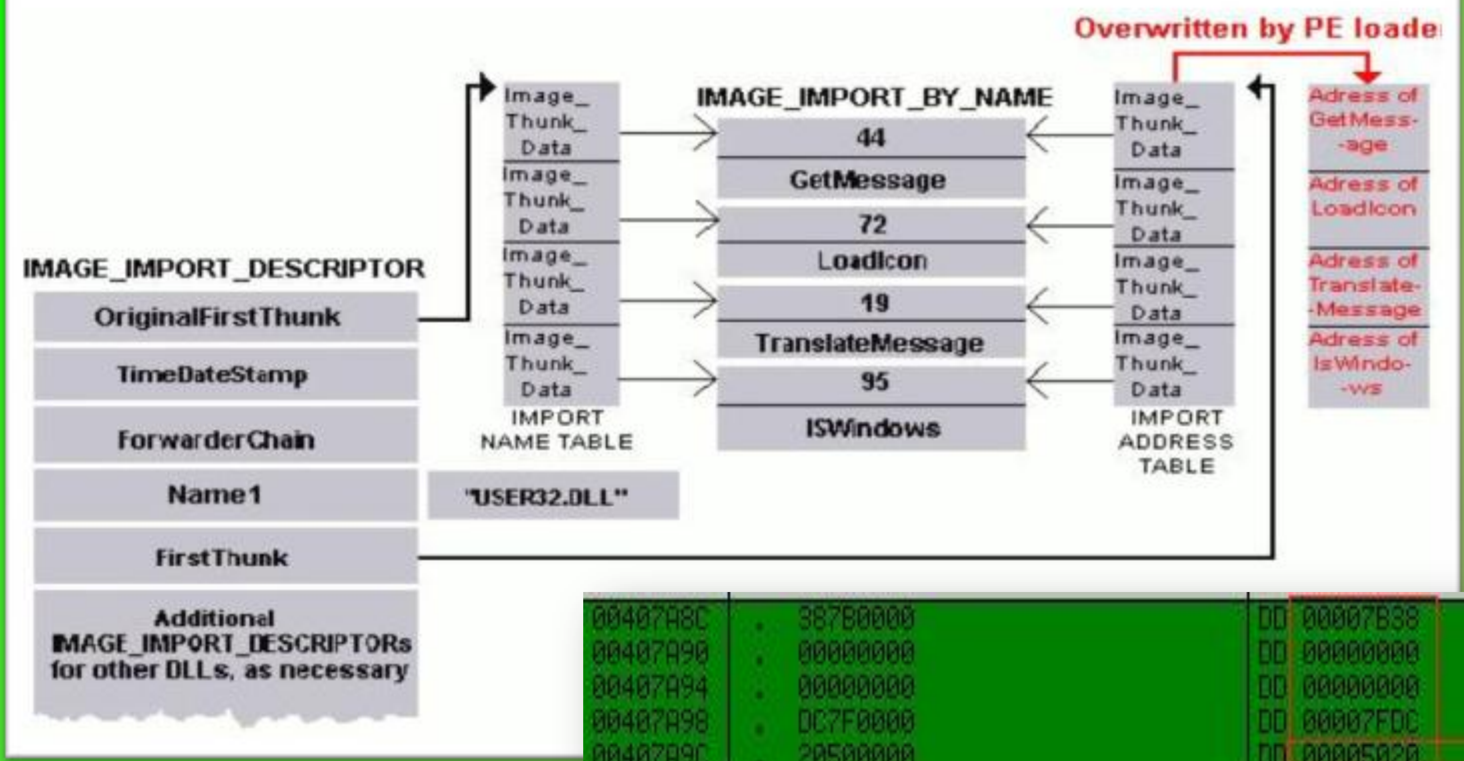
```
IMAGE_THUNK_DATA32 STRUCT
    union u1
        ForwarderString DWORD        ?
        Function        DWORD        ?
        Ordinal         DWORD        ?
        AddressOfData   DWORD        ?
    ends
IMAGE_THUNK_DATA32 ENDS
```

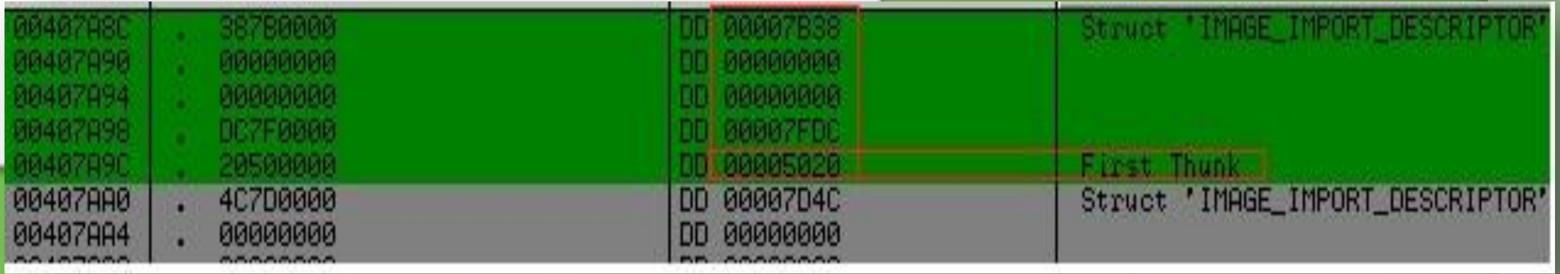


```
IMAGE_IMPORT_BY_NAME STRUCT
    Hint         WORD        ?
    Name1        BYTE        ?
IMAGE_IMPORT_BY_NAME ENDS
```

Hint
Name1

- Each IMAGE_THUNK_DATA str corresponds to one imported function from the dll.
- Arrays pointed by OriginalFirstThunk and FirstThunk run parallelly.
- OriginalFirstThunk – Import Name Table – Never modified
- FirstThunk – Import Address Table – Contain actual function addresses





```
00407A8C  .  387B0000   DD 00007B38   Struct 'IMAGE_IMPORT_DESCRIPTOR'
00407A90  .  00000000   DD 00000000
00407A94  .  00000000   DD 00000000
00407A98  .  DC7F0000   DD 00007FDC
00407A9C  .  20500000   DD 00005020   First Thunk
00407AA0  .  4C7D0000   DD 00007D4C   Struct 'IMAGE_IMPORT_DESCRIPTOR'
00407AA4  .  00000000   DD 00000000
```

Functions exported by ordinal only

- No IMAGE_IMPORT_BY_NAME structure

-IMAGE_THUNK_DATA contains the ordinal of the function

-MSB used to identify the same

- MSB is set, rest 31 bits are treated as an ordinal value.

-Bound Imports

DEMO

Reference

- Complete Reference Guide for Reversing & Malware Analysis Training

PE file format test

- Write a program in “C” or “ASM” that will modify the Address of Entry point of an Executable (.exe) file with any random address.
- Write a program in “C” or “ASM” that will add a new section into an executable (.exe)

  - For hints shoot us an email ☺

Thank You !

www.SecurityXploded.com