09/09/2025

Malicious Code Analysis

Fangtian Zhong
CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

Overview

01 Optional Header

02 Section Headers

Part One

01

Optional Header

2025-8-23

Optional Header

This header contains information about the file's preferred load address, the size of the image, and the size of the stack required to run the file.

Optional Header

- there are two versions of the Optional Header, one for 32-bit executables and one for 64-bit executables.
  - The size of the structure itself (or the number of members defined within the structure): IMAGE_OPTIONAL_HEADER32 has 31 members while IMAGE_OPTIONAL_HEADER64 only has 30 members, that additional member in the 32-bit version is a DWORD named BaseOfData which holds an RVA of the beginning of the data section.
  - The data type of some of the members: The following 5 members of the Optional Header structure are defined as DWORD in the 32-bit version and as ULONGLONG in the 64-bit version:

    ImageBase
    SizeOfStackReserve
    SizeOfStackCommit
    SizeOfHeapReserve
    SizeOfHeapCommit

Optional Header-64 bit

```
typedef struct _IMAGE_OPTIONAL_HEADER64 {
    WORD        Magic;
    BYTE        MajorLinkerVersion;
    BYTE        MinorLinkerVersion;
    DWORD       SizeOfCode;
    DWORD       SizeOfInitializedData;
    DWORD       SizeOfUninitializedData;
    DWORD       AddressOfEntryPoint;
    DWORD       BaseOfCode;
    ULONGLONG   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;
    ULONGLONG   SizeOfStackReserve;
    ULONGLONG   SizeOfStackCommit;
    ULONGLONG   SizeOfHeapReserve;
    ULONGLONG   SizeOfHeapCommit;
    DWORD       LoaderFlags;
    DWORD       NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY
DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER64,
*PIMAGE_OPTIONAL_HEADER64;
```

Fields

Magic: Microsoft documentation describes this field as an integer that identifies the state of the image, the documentation mentions three common values:

- **0x10B:** Identifies the image as a PE32 executable.
- **0x20B:** Identifies the image as a PE64 executable.
- **0x107:** Identifies the image as a ROM image.

The value of this field is what determines whether the executable is 32-bit or 64-bit.

Fields

- **MajorLinkerVersion and MinorLinkerVersion:** The linker major and minor version numbers.
- **SizeOfCode:** This field holds the size of the code (.text) section, or the sum of all code sections if there are multiple sections.
- **SizeOfInitializedData:** This field holds the size of the initialized data (.data) section, or the sum of all initialized data sections if there are multiple sections.

Fields

- **SizeOfUninitializedData**: This field holds the size of the uninitialized data (.bss) section, or the sum of all uninitialized data sections if there are multiple sections.
- **AddressOfEntryPoint:** A pointer to the entry point function, relative to the image base address. The documentation states that for program images this relative address points to the starting address and for device drivers it points to initialization function. For DLLs an entry point is optional, and in the case of entry point absence the AddressOfEntryPoint field is set to 0.
- **BaseOfCode:** A pointer to the beginning of the code section, relative to the image base.

Fields

- **BaseOfData (PE32 Only):** A pointer to the beginning of the data section, relative to the image base..
- **ImageBase:** The preferred address of the first byte of the image when it is loaded in memory. this value must be a multiple of 64K. Due to memory protections like ASLR, and a lot of other reasons, the address specified by this field is almost never used, in this case the PE loader chooses an unused memory range to load the image into, after loading the image into that address the loader goes into a process called the relocating where it fixes the constant addresses within the image to work with the new image base, there's a special section that holds information about places that will need fixing if relocation is needed, that section is called the relocation section (.reloc), more on that in the upcoming posts.

Fields

- **SectionAlignment:** The alignment of sections loaded in memory, in bytes. Sections are aligned in memory boundaries that are multiples of this value. The documentation states that this value defaults to the page size for the architecture and it can’t be less than the value of FileAlignment.
- **FileAlignment:** The alignment of the raw data of sections in the image file, in bytes. if the size of the actual data in a section is less than the FileAlignment value, the rest of the chunk gets padded with zeroes to keep the alignment boundaries. The documentation states that this value should be a power of 2 between 512 and 64K, and if the value of SectionAlignment is less than the architecture’s page size then the sizes of FileAlignment and SectionAlignment must match.

Fields

- **MajorOperatingSystemVersion, MinorOperatingSystemVersion, MajorImageVersion, MinorImageVersion, MajorSubsystemVersion and MinorSubsystemVersion:** These members of the structure specify the major version number of the required operating system, the minor version number of the required operating system, the major version number of the image, the minor version number of the image, the major version number of the subsystem and the minor version number of the subsystem respectively.
- **Win32VersionValue:** A reserved field that the documentation says should be set to 0.
- **SizeOfImage:** The size of the image file (in bytes), including all headers. It gets rounded up to a multiple of SectionAlignment because this value is used when loading the image into memory.

Fields

- **SizeOfHeaders:** The combined size of the e_lfanew member of IMAGE_DOS_HEADER, NT Headers, and section headers, rounded up to a multiple of FileAlignment.
- **CheckSum:** A checksum of the image file, it’s used to validate the image at load time.
- **Subsystem:** This field specifies the Windows subsystem (if any) that is required to run the image.

Subsystem

Constant	Value	Description
IMAGE_SUBSYSTEM_UNKNOWN	0	An unknown subsystem
IMAGE_SUBSYSTEM_NATIVE	2	Device drivers and native Windows processes
IMAGE_SUBSYSTEM_WINDOWS_GUI	3	The Windows graphical user interface (GUI) subsystem
IMAGE_SUBSYSTEM_WINDOWS_CUI	4	The Windows character subsystem
IMAGE_SUBSYSTEM_OS2_CUI	5	The OS/2 character subsystem
IMAGE_SUBSYSTEM_POSIX_CUI	7	The Posix character subsystem
IMAGE_SUBSYSTEM_NATIVE_WINDOWS	8	Native Win9x driver
IMAGE_SUBSYSTEM_WINDOWS_CE_GUI	9	Windows CE
IMAGE_SUBSYSTEM_EFI_APPLICATION	10	An Extensible Firmware Interface (EFI) application
IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER	12	An EFI driver with run-time services
IMAGE_SUBSYSTEM_EFI_ROM	13	An EFI ROM image
IMAGE_SUBSYSTEM_XBOX	14	XBOX
IMAGE_SUBSYSTEM_WINDOWS_BOOT_APPLICATION	16	Windows boot application.

Fields

- **DLLCharacteristics:** This field defines some characteristics of the executable image file, like if it's NX compatible and if it can be relocated at run time. It exists within normal executable image files and it defines characteristics that can apply to normal executable files.
- **SizeOfStackReserve, SizeOfStackCommit, SizeOfHeapReserve and SizeOfHeapCommit:** These fields specify the size of the stack to reserve, the size of the stack to commit, the size of the local heap space to reserve and the size of the local heap space to commit respectively.

DLLCharacteristics

Constant	Value	Description
	0x0001	Reserved,must be zero.
	0x0002	Reserved,must be zero.
	0x0004	Reserved,must be zero.
	0x0008	Reserved, must be zero.
IMAGE_DLLCHARACTERISTICS_HIGH_ENTROPY_VA	0x0020	Image can handle a high entropy 64-bit virtual address space.
IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE	0x0040	DLL can be relocated at load time.
IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY	0x0080	Code Integrity checks are enforced.
IMAGE_DLLCHARACTERISTICS_NX_COMPAT	0x0100	Image is NX compatible.
IMAGE_DLLCHARACTERISTICS_NO_ISOLATION	0x0200	Isolation aware,but do not isolate the image.

DLLCharacteristics

Constant	Value	Description
IMAGE_DLLCHARACTERISTICS_NO_SEH	0x0400	Does not use structured exception (SE) handling. No SE handler may be called in this image.
IMAGE_DLLCHARACTERISTICS_NO_BIND	0x0800	Do not bind the image.
IMAGE_DLLCHARACTERISTICS_APPCONTAINER	0x1000	Image must execute in an AppContainer.
IMAGE_DLLCHARACTERISTICS_WDM_DRIVER	0×2000	A WDM driver.
IMAGE_DLLCHARACTERISTICS_GUARD_CF	0x4000	Image supports Control Flow Guard.
IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER_AWARE	0x8000	Terminal Server aware.

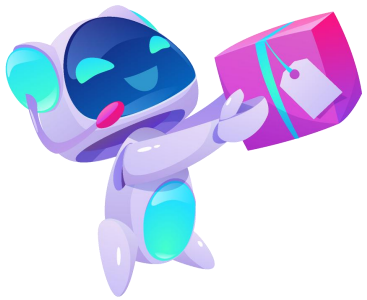
Fields

- **LoaderFlags:** A reserved field that the documentation says should be set to 0.
- **NumberOfRvaAndSizes:** Size of the DataDirectory array.
- **DataDirectory:** An array of IMAGE_DATA_DIRECTORY structures. We will talk about this in the next post.

Optional Header

Disasm: [.pdata] to [.rsrc] | General | DOS Hdr | Rich Hdr | File Hdr | Optional Hdr | Section Hdrs

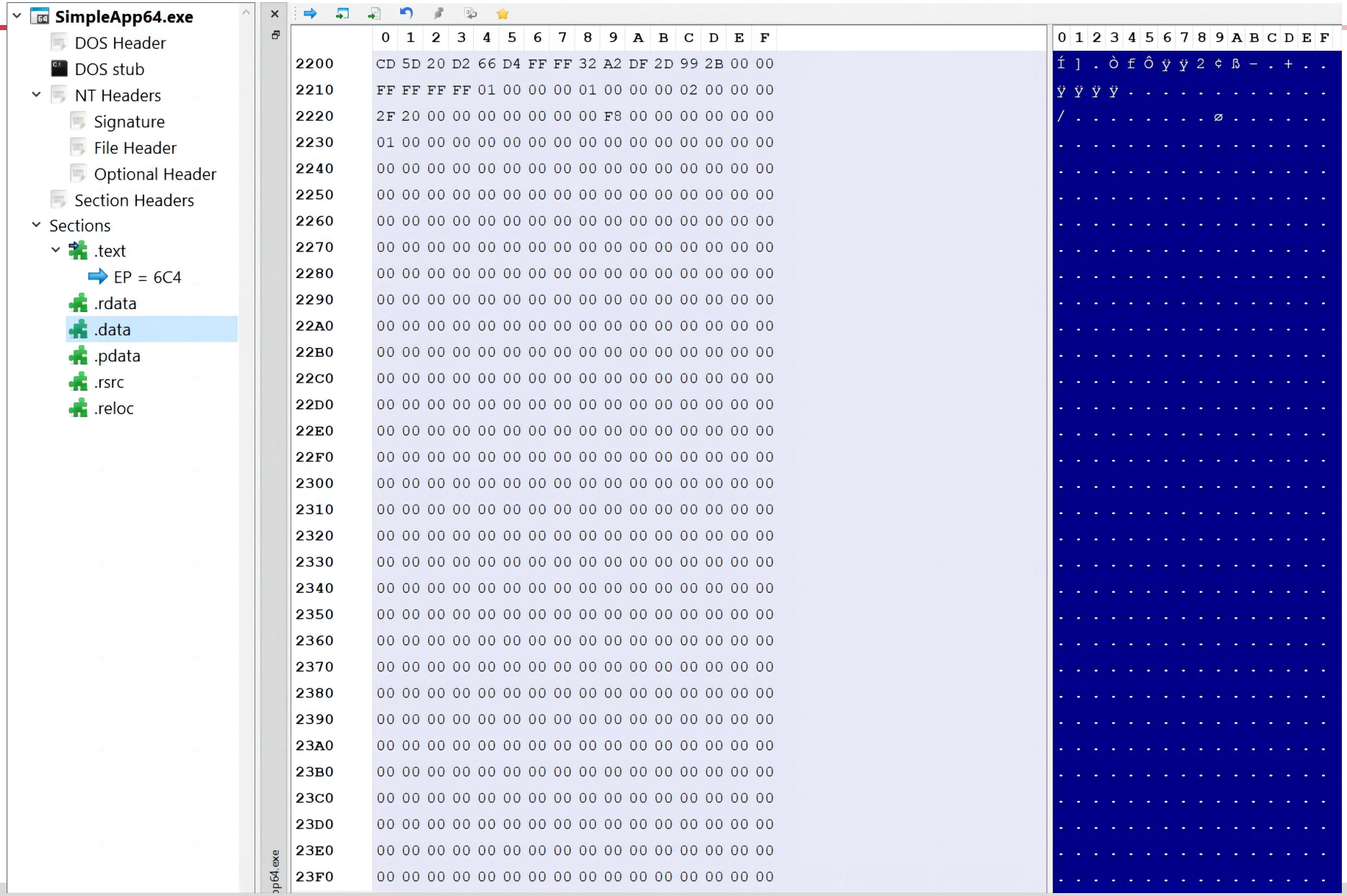
Offset	Name	Value	Value
118	Magic	20B	NT64
11A	Linker Ver. (Major)	E	
11B	Linker Ver. (Minor)	1A	
11C	Size of Code	E00	
120	Size of Initialized Data	1E00	
124	Size of Uninitialized Data	0	
128	Entry Point	12C4	
12C	Base of Code	1000	
130	Image Base	140000000	
138	Section Alignment	1000	
13C	File Alignment	200	
140	OS Ver. (Major)	6	Windows Vista / Server 2008
142	OS Ver. (Minor)	0	
144	Image Ver. (Major)	0	
146	Image Ver. (Minor)	0	
148	Subsystem Ver. (Major)	6	
14A	Subsystem Ver. Minor)	0	
14C	Win32 Version Value	0	
150	Size of Image	7000	
154	Size of Headers	400	
158	Checksum	0	
15C	Subsystem	3	Windows console
15E	DLL Characteristics	8160	
		40	DLL can move
		100	Image is NX compatible
		8000	TerminalServer aware
160	Size of Stack Reserve	100000	
168	Size of Stack Commit	1000	
170	Size of Heap Reserve	100000	
178	Size of Heap Commit	1000	
180	Loader Flags	0	
184	Number of RVAs and Sizes	10	
	Data Directory	Address	Size



Fields

- We can talk about some of these fields, first one being the Magic field at the start of the header, it has the value 0x20B meaning that this is a PE64 executable.
- We can see that the entry point RVA is 0x12C4 and the code section start RVA is 0x1000, it follows the alignment defined by the SectionAlignment field which has the value of 0x1000.
- File alignment is set to 0x200, and we can verify this by looking at any of the sections, for example the data section:

Optional Header



Part Two

02

Section Headers

Section Headers

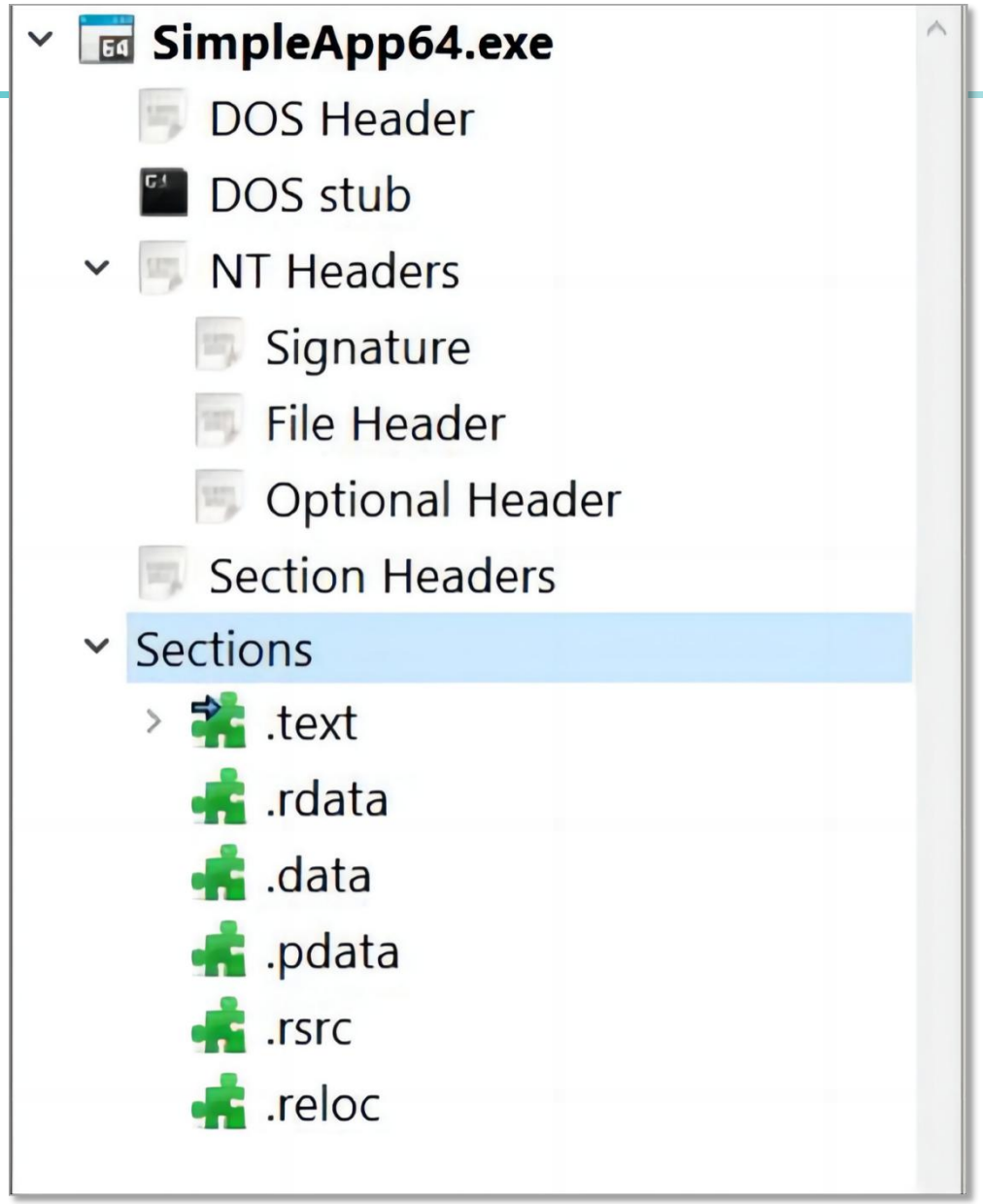
This section contains information about each section of the file, including the section's name, size, and location in the file.

Sections

- Sections are the containers of the actual data of the executable file, they occupy the rest of the PE file after the headers, precisely after the section headers.

.text: Contains the executable code of the program.
.data: Contains the initialized data.
.bss: Contains uninitialized data.
.rdata: Contains read-only initialized data.
.edata: Contains the export tables.
.idata: Contains the import tables.
.reloc: Contains image relocation information.
.rsrc: Contains resources used by the program, these include images, icons or even embedded binaries.
.tls: (Thread Local Storage), provides storage for every executing thread of the program.

Sections


SimpleApp64.exe
DOS Header
DOS stub
NT Headers
Signature
File Header
Optional Header
Section Headers
Sections
.text
.rdata
.data
.pdata
.rsrc
.reloc


?

Section Headers

```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE    Name[IMAGE_SIZEOF_SHORT_NAME];
    union {
            DWORD   PhysicalAddress;
            DWORD   VirtualSize;
    } Misc;
    DWORD   VirtualAddress;
    DWORD   SizeOfRawData;
    DWORD   PointerToRawData;
    DWORD   PointerToRelocations;
    DWORD   PointerToLinenumbers;
    WORD    NumberOfRelocations;
    WORD    NumberOfLinenumbers;
    DWORD   Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

Fields

- Name: First field of the Section Header, a byte array of the size IMAGE_SIZEOF_SHORT_NAME that holds the name of the section. IMAGE_SIZEOF_SHORT_NAME has the value of 8 meaning that a section name can’t be longer than 8 characters. For longer names the official documentation mentions a work-around by filling this field with an offset in the string table, however executable images do not use a string table so this limitation of 8 characters holds for executable images.

Fields

- **PhysicalAddress or VirtualSize:** A union defines multiple names for the same thing, this field contains the total size of the section when it's loaded in memory.
- **VirtualAddress:** The address of the first byte of the section when loaded into memory, relative to the image base, and for object files it holds the address of the first byte of the section before relocation is applied.
- **SizeOfRawData:** The size of the initialized data on disk, in bytes. It must be a multiple of IMAGE_OPTIONAL_HEADER.FileAlignment. SizeOfRawData and VirtualSize can be different.

Fields

- **PointerToRawData:** A pointer to the first page of the section within the file, for executable images it must be a multiple of IMAGE_OPTIONAL_HEADER.FileAlignment.
- **PointerToRelocations:** A file pointer to the beginning of relocation entries for the section. It’s set to 0 for executable files.
- **PointerToLineNumbers:** A file pointer to the beginning of COFF line-number entries for the section. It’s set to 0 because COFF debugging information is deprecated.
- **NumberOfRelocations:** The number of relocation entries for the section, it’s set to 0 for executable images.

Fields

- **NumberOfLinenumbers:** The number of COFF line-number entries for the section, it's set to 0 because COFF debugging information is deprecated.
- **Characteristics:** Flags that describe the characteristics of the section. These characteristics are things like if the section contains executable code, contains initialized/uninitialized data, can be shared in memory.

Characteristics

Flag	Value	Description
	0x00000000	Reserved for future use.
	0x00000001	Reserved for future use.
	0x00000002	Reserved for future use.
	0x00000004	Reserved for future use.
IMAGE_SCN_TYPE_NO_PAD	0x00000008	The section should not be padded to the next boundary. This flag is obsolete and is replaced by IMAGE_SCN_ALIGN_1BYTES. This is valid only for object files.
	0x00000010	Reserved for future use.
IMAGE_SCN_CNT_CODE	0x00000020	The section contains executable code.
IMAGE_SCN_CNT_INITIALIZED_DATA	0x00000040	The section contains initialized data.
IMAGE_SCN_CNT_UNINITIALIZED_DATA	0x00000080	The section contains uninitialized data.
IMAGE_SCN_LNK_OTHER	0x00000100	Reserved for future use.

Characteristics

Flag	Value	Description
IMAGE_SCN_LNK_INFO	0x00000200	The section contains comments or other information. The .drectve section has this type. This is valid for object files only.
	0x00000400	Reserved for future use.
IMAGE_SCN_LNK_REMOVE	0x00000800	The section will not become part of the image. This is valid only for object files.
IMAGE_SCN_LNK_COMDAT	0x00001000	The section contains COMDAT data.For more information,see COMDAT Sections (Object Only). This is valid only for object files.
IMAGE_SCN_GPREL	0x00008000	The section contains data referenced through the global pointer (GP).
IMAGE_SCN_MEM_PURGEABLE	0x00020000	Reserved for future use.
IMAGE_SCN_MEM_16BIT	0x00020000	Reserved for future use.
IMAGE_SCN_MEM_LOCKED	0x00040000	Reserved for future use.
IMAGE_SCN_MEM_PRELOAD	0x00080000	Reserved for future use.

Characteristics

Flag	Value	Description
IMAGE_SCN_ALIGN_1BYTES	0x00100000	Align data on a 1-byte boundary.Valid only for object files.
IMAGE_SCN_ALIGN_2BYTES	0x00200000	Align data on a 2-byte boundary.Valid only for object files.
IMAGE_SCN_ALIGN_4BYTES	0x00300000	Align data on a 4-byte boundary.Valid only for object files.
IMAGE_SCN_ALIGN_8BYTES	0x00400000	Align data on an 8-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_16BYTES	0x00500000	Align data on a 16-byte boundary.Valid only for object files.
IMAGE_SCN_ALIGN_32BYTES	0x00600000	Align data on a 32-byte boundary.Valid only for object files.
IMAGE_SCN_ALIGN_64BYTES	0x00700000	Align data on a 64-byte boundary.Valid only for object files.
IMAGE_SCN_ALIGN_128BYTES	0x00080000	Align data on a 128-byte boundary. Valid only for object files.

Characteristics

Flag	Value	Description
IMAGE_SCN_ALIGN_512BYTES	0x00A00000	Align data on a 512-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_1024BYTES	0x00B00000	Align data on a 1024-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_2048BYTES	0x00C00000	Align data on a 2048-byte boundary.Valid only for object files.
IMAGE_SCN_ALIGN_4096BYTES	0x00D00000	Align data on a 4096-byte boundary.Valid only for object files.
IMAGE_SCN_ALIGN_8192BYTES	0x00E00000	Align data on an 8192-byte boundary. Valid only for object files.
IMAGE_SCN_LNK_NRELOC_OVFL	0x01000000	The section contains extended relocations.
IMAGE_SCN_MEM_DISCARDABLE	0x02000000	The section can be discarded as needed.
IMAGE_SCN_MEM_NOT_CACHED	0x04000000	The section cannot be cached.
IMAGE_SCN_MEM_NOT_PAGED	0x08000000	The section is not pageable.
IMAGE_SCN_MEM_SHARED	0x10000000	The section can be shared in memory.

Characteristics

Flag	Value	Description
IMAGE_SCN_MEM_SHARED	0x10000000	The section can be shared in memory.
IMAGE_SCN_MEM_EXECUTE	0x20000000	The section can be executed as code.
IMAGE_SCN_MEM_READ	0x40000000	The section can be read.
IMAGE_SCN_MEM_WRITE	0x80000000	The section can be written to.

Reminder

- SizeOfRawData and VirtualSize can be different, and this can happen for multiple of reasons.
- SizeOfRawData must be a multiple of IMAGE_OPTIONAL_HEADER.FileAlignment, so if the section size is less than that value the rest gets padded and SizeOfRawData gets rounded to the nearest multiple of IMAGE_OPTIONAL_HEADER.FileAlignment.
- However when the section is loaded into memory it doesn’t follow that alignment and only the actual size of the section is occupied. In this case SizeOfRawData will be greater than VirtualSize.

Reminder

- If the section contains uninitialized data, these data won’t be accounted for on disk, but when the section gets mapped into memory, the section will expand to reserve memory space for when the uninitialized data gets later initialized and used.
- This means that the section on disk will occupy less than it will do in memory, in this case VirtualSize will be greater than SizeOfRawData.

Section Headers

Disasm: .text | General | DOS Hdr | Rich Hdr | File Hdr | Optional Hdr | Section Hdrs | Imports | Resources | Ex

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
.text	400	E00	1000	D2C	60000020	0	0	0
>	1200	^	1D2C	^	r-x			
.rdata	1200	1000	2000	E3C	40000040	0	0	0
>	2200	^	2E3C	^	r--			
.data	2200	200	3000	638	C0000040	0	0	0
>	2400	^	3638	^	rw-			
.pdata	2400	200	4000	168	40000040	0	0	0
>	2600	^	4168	^	r--			
.rsrc	2600	200	5000	1E0	40000040	0	0	0
>	2800	^	51E0	^	r--			
.reloc	2800	200	6000	28	42000040	0	0	0
>	2A00	^	6028	^	r--			

Example

- For example if we take the .text section, it has a raw address of 0x400 and a raw size of 0xE00, if we add them together we get 0x1200 which is displayed as the section end on disk.
- Similarly we can do the same with virtual size and address, virtual address is 0x1000 and virtual size is 0xD2C, if we add them together we get 0x1D2C.

MONTANA STATE UNIVERSITY

THE END

Fangtian Zhong

CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

09/09/2025