Reverse Engineering and Malware Analysis Fundamentals

PE Structures

Adam Podlosky
@apodlosky

DOS Header (Partial)

```
#define IMAGE_DOS_SIGNATURE  0x5A4D  // MZ

typedef struct {
    UINT16  e_magic;     // Magic value ‘MZ’
    UINT16  e_cblp;
    UINT16  e_cp;
    UINT16  e_crlc;
    UINT16  e_cparhdr;
    UINT16  e_maxalloc;
    UINT16  e_crlc;
    // ...  (fields omitted)
    INT32   e_lfanew;    // Offset of the PE header
} IMAGE_DOS_HEADER;
```

NT Headers

```
#define IMAGE_NT_SIGNATURE    0x00004550        // PE00

typedef struct {
    UINT32                    Signature;
    IMAGE_FILE_HEADER         FileHeader;       // Important offsets
    IMAGE_OPTIONAL_HEADER32   OptionalHeader;   // Mostly required
} IMAGE_NT_HEADERS32;

typedef struct {
    UINT32                    Signature;
    IMAGE_FILE_HEADER         FileHeader;
    IMAGE_OPTIONAL_HEADER64   OptionalHeader;
} IMAGE_NT_HEADERS64;
```

File Header

```
typedef struct {
    UINT16  Machine;                // The target architecture
    UINT16  NumberOfSections;       // Number of Section Headers
    UINT32  TimeDateStamp;          // Build timestamp (bound imports)
    UINT32  PointerToSymbolTable;
    UINT32  NumberOfSymbols;
    UINT16  SizeOfOptionalHeader;   // Locates section headers
    UINT16  Characteristics;        // Flags that describe the PE
} IMAGE_FILE_HEADER;
```

Optional Header (Partial)

```
typedef struct {
    UINT16   Magic;                  // 0x10B for PE32, 0x20B for PE32+ (64-bit)
    // ...   (versions omitted)
    UINT32   SizeOfCode;             // Size of code (.text) section
    // ...   (sizes omitted)
    UINT32   AddressOfEntryPoint;    // RVA to begin execution at once loaded
    UINT32   BaseOfCode;             // Loader ignores this value (RVA to code)
    UINTPTR  ImageBase;              // Preferred base virtual address to map at (32/64)
    UINT32   SectionAlignment;       // Alignment of sections in memory, page-size (4K usually)
    UINT32   FileAlignment;          // Alignment of PE on disk, usually 512
    // ...   (versions omitted)
    UINT32   SizeOfImage;
    UINT32   SizeOfHeaders;          // Size of all headers, including section header table
    UINT32   CheckSum;               // File checksum, only verified for drivers
    UINT16   Subsystem;              // Subsystem responsible to run this executable
    // ...   (sizes and flags omitted)
    UINT32   NumberOfRvaAndSizes;    // Determine size of DataDirectory[] (not always 16)
    IMAGE_DATA_DIRECTORY DataDirectory[16]; // Data directories used to locate sections
} IMAGE_OPTIONAL_HEADERxx; // 32/64 dependent
```

Data Directories

```
typedef struct {
    UINT32  VirtualAddress;  // RVA to an index-specific directory structure
    UINT32  Size;            // Size of the directory structure
} IMAGE_DATA_DIRECTORY;

#define IMAGE_DIRECTORY_ENTRY_EXPORT          0  // Export directory
#define IMAGE_DIRECTORY_ENTRY_IMPORT          1  // Import directory
#define IMAGE_DIRECTORY_ENTRY_RESOURCE        2  // Resources!
#define IMAGE_DIRECTORY_ENTRY_EXCEPTION       3  // Exception handlers
#define IMAGE_DIRECTORY_ENTRY_SECURITY        4  // Digital signature
#define IMAGE_DIRECTORY_ENTRY_BASERELOC       5  // Relocation tables
#define IMAGE_DIRECTORY_ENTRY_DEBUG           6  // Debugging information
#define IMAGE_DIRECTORY_ENTRY_ARCHITECTURE    7  // ?
#define IMAGE_DIRECTORY_ENTRY_GLOBALPTR       8  // Itanium only
#define IMAGE_DIRECTORY_ENTRY_TLS             9  // Thread local storage
#define IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG    10  //
#define IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT   11  // Bound import tables
#define IMAGE_DIRECTORY_ENTRY_IAT            12  // Import address table
#define IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT   13  // Delay import tables
#define IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14  // .NET
```

Section Header

```
typedef struct {
    CHAR    Name[8];                // Short name describing the section
    union {
       UINT32  PhysicalAddress;
       UINT32  VirtualSize;         // Size of section in memory
    } Misc;
    UINT32  VirtualAddress;         // RVA of section when mapped into memory
    UINT32  SizeOfRawData;          // Size of section on disk (aligned)
    UINT32  PointerToRawData;       // File offset of section
    UINT32  PointerToRelocations;
    UINT32  PointerToLinenumbers;
    UINT16  NumberOfRelocations;
    UINT16  NumberOfLinenumbers;
    UINT32  Characteristics;        // Flags and memory page permissions
} IMAGE_SECTION_HEADER;
```

Export Directory

```
typedef struct {
    UINT32  Characteristics;        // Not used
    UINT32  TimeDateStamp;
    UINT16  MajorVersion;
    UINT16  MinorVersion;
    UINT32  Name;                   // RVA to string - DLL name
    UINT32  Base;                   // Ordinal base value (usually 1)
    UINT32  NumberOfFunctions;      // Array size for Functions
    UINT32  NumberOfNames;          // Array size for Names/NameOrdinals
    UINT32  AddressOfFunctions;     // RVA to array of RVAs to addresses
    UINT32  AddressOfNames;         // RVA to array of RVAs to strings
    UINT32  AddressOfNameOrdinals;  // RVA to array of 16bit ordinals
} IMAGE_EXPORT_DIRECTORY;
```

Import Thunks

```
typedef struct {
    UINT16  Hint;                   // Possible index in Export Names table
    CHAR    Name[1];                // String of function/symbol
} IMAGE_IMPORT_BY_NAME;

typedef struct {
    union {
        UINT32 ForwarderString; // RVA to forwarder string
        UINT32 Function;        // RVA to function address
        UINT32 Ordinal;         // RVA to ordinal number (16bit)
        UINT32 AddressOfData;   // RVA to IMAGE_IMPORT_BY_NAME
    };
} IMAGE_THUNK_DATA32; // IMAGE_THUNK_DATA64 is union of 4x UINT64
```

Structure for Import Descriptor

```
typedef struct {
    union {
        UINT32  Characteristics;    // 0 for terminating last descriptor
        UINT32  OriginalFirstThunk; // RVA to original unbound INT
    };
    UINT32  TimeDateStamp;          // 0 if not bound, -1 if bound
    UINT32  ForwarderChain;         // -1 if no forwarders
    UINT32  Name;                   // RVA to string - DLL name
    UINT32  FirstThunk;             // RVA to IAT (if bound has addresses)
} IMAGE_IMPORT_DESCRIPTOR;
```

TLS Callback and TLS Directory

```
typedef VOID (NTAPI IMAGE_TLS_CALLBACK)(
    VOID *DllHandle,
    UINT32 Reason,
    VOID *Reserved);                // Callback function signature

typedef struct {
    UINT32  StartAddressOfRawData;
    UINT32  EndAddressOfRawData;
    UINT32  AddressOfIndex;         // RVA to TLS index
    UINT32  AddressOfCallBacks;     // RVA to IMAGE_TLS_CALLBACK functions
    UINT32  SizeOfZeroFill;
    UINT32  Characteristics;
} IMAGE_TLS_DIRECTORY32; // 64-bit similar, four addresses are UINT64
```

Enough with the Structures!!!

- Still 60+ structures and 200+ macros to cover…
  - See “Image Format” in the ***winnt.h*** header file
  - Included with the Microsoft Windows Platform SDK
- Microsoft PE and COFF Specification (73 pages)
- PE format visualizations:
  - Corkami’s PE Posters: 101 (PNG) and 102 (PDF)
  - Ero Carrera's PE File Format Graphs