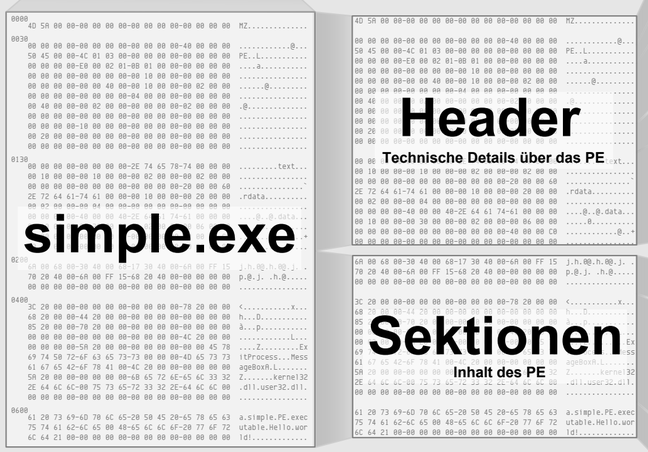
PE101 portable executable — Ein Überblick über Windows Executables

Ange Albertini
corkami.com

Zerlegtes PE

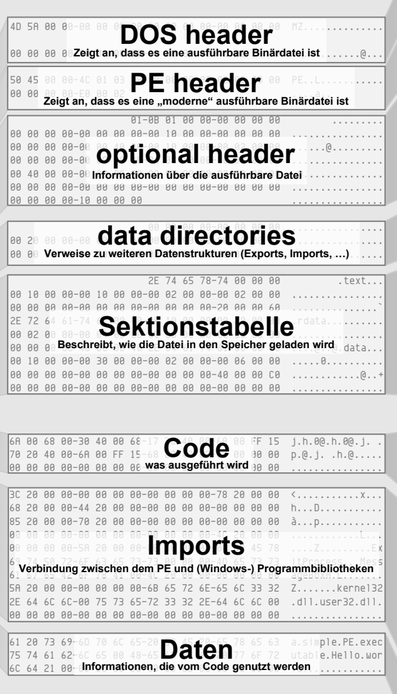
SHA-1 b7af4cb51ce38e43e030656eb2698fab408cf9cb
download @ pe101.corkami.com



simple.exe

Header
Technische Details über das PE

Sektionen
Inhalt des PE



DOS header
Zeigt an, dass es eine ausführbare Binärdatei ist

PE header
Zeigt an, dass es eine „moderne" ausführbare Binärdatei ist

optional header
Informationen über die ausführbare Datei

data directories
Verweise zu weiteren Datenstrukturen (Exports, Imports, ...)

Sektionstabelle
Beschreibt, wie die Datei in den Speicher geladen wird

Code
was ausgeführt wird

Imports
Verbindung zwischen dem PE und (Windows-) Programmbibliotheken

Daten
Informationen, die vom Code genutzt werden

Dump im Hexadezimalformat	ASCII Dump	Bezeichner/Felder	Werte	Erläuterung
4D 5A 00 00-00 00 00 00-00 00 00 00-00 00 00 00	MZ..............	e_magic	'MZ'	Konstante Signatur
Offset:0x30 00 00 00 00-00 00 00 00-00 00 00 00-40 00 00 00	............@...	e_lfanew	0x40	Offset zum PE Header ❶
Offset:0x40 50 45 00 00-4C 01 03 00-00 00 00 00-00 00 00 00	PE..L...........	Signature	'PE', 0, 0	Konstante Signatur
00 00 00 00-E0 00 02 01...	....a...	Machine	0x14c [intel 386]	Prozessortyp: ARM/MIPS/Intel/...
		NumberOfSections	3	Anzahl an Sektionen ❷
		SizeOfOptionalHeader	0xe0	Relatives Offset der Sektionstabelle ❷
		Characteristics	0x102 [32b EXE]	EXE/DLL/...
Offset:0x58 ...0B 01 00 00-00 00 00 00	....	Magic	0x10b [32b]	32 bits/64 bits
00 00 00 00-00 00 00 00-00 10 00 00-00 00 00 00	....	AddressOfEntryPoint	0x1000	Einstiegspunkt zur Ausführung des PE ❺
00 00 00 00-00 00 40 00-00 10 00 00-00 02 00 00	......@.........	ImageBase	0x400000	Adresse, an die die Datei in den Speicher geladen werden soll
00 00 00 00-00 00 00 00-04 00 00 00-00 00 00 00	....	SectionAlignment	0x1000	Ausrichtung, an der Sektionen im Speicher beginnen sollen ❷
00 40 00 00-00 02 00 00-00 00 00 00-02 00 00 00	.@..............	FileAlignment	0x200	Ausrichtung, an der Sektionen in der Datei beginnen sollen ❷
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00	....	MajorSubsystemVersion	4 [NT 4 oder höher]	Zur Ausführung benötigte Version von Windows
00 00 00 00-10 00 00 00...	....	SizeOfImage	0x4000	Gesamtmenge an benötigtem Arbeitsspeicher
		SizeOfHeaders	0x200	Gesamtgröße aller Header ❸
		Subsystem	2 [GUI]	Treiber / grafische Oberfläche / Kommandozeile
		NumberOfRvaAndSizes	16	Anzahl an Data Directories ❹
...00 00 00 00-00 00 00 00	....			
00 20 00 00-00 00 00 00-00 00 00 00-00 00 00 00	....	ImportsVA	0x2000	RVA*der Imports ❹
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00	....			

Offset:0x138

Name	VirtualSize	RVA* VirtualAddress	SizeOfRawData	Physikalische Größe PointerToRawData	Physikalisches Offset Characteristics
.text	0x1000	0x1000	0x200	0x200	CODE EXECUTE READ
.rdata	0x1000	0x2000	0x200	0x400	INITIALIZED READ
.data	0x1000	0x3000	0x200	0x600	DATA READ WRITE

Für jede Sektion wird ein Block der Größe SizeofRawData ab der Stelle PointerToRawData aus der Datei gelesen.
Er wird in den Speicher an die Adresse VirtualAddress in einem Block der Größe VirtualSize mit den vorgegebenen Charakteristiken abgelegt.

Offset:0x200/RVA:0x401000

```
6A 00 68 00-30 40 00 68-17 30 40 00-6A 00 FF 15   j.h.0@.h.0@.j. .
70 20 40 00-6A 00 FF 15-68 20 40 00               p.@.j. .h.@.
```

x86 Assembler-Instruktionen | Äquivalenter C-Code

```
push 0                                                          ;
push 0x403000                                                   ;
push 0x403017                                                   ;
push 0                                                          ;
call [0x402070]  →MessageBox(0, "Hello World!","a simple PE executable", 0);
push 0                                                          ;
call [0x402068]  →ExitProcess(0);
```

Offset:0x400/RVA:0x402000

```
3C 20 00 00-00 00 00 00-00 00 00 00-78 20 00 00   <...........x...
68 20 00 00-44 20 00 00-00 00 00 00-00 00 00 00   h...D...........
85 20 00 00-70 20 00 00-00 00 00 00-00 00 00 00   à...p...........
00 00 00 00-00 00 00 00-00 00 00 00-4C 20 00 00   ............L...
00 00 00 00-5A 20 00 00-00 00 00 00-00 00 45 78   ....Z.........Ex
69 74 50 72-6F 63 65 73-73 00 00 00-4D 65 73 73   itProcess...Mess
61 67 65 42-6F 78 41 00-4C 20 00 00-00 00 00 00   ageBoxA.L.......
5A 20 00 00-00 00 00 00-6B 65 72 6E-65 6C 33 32   Z.......kernel32
2E 64 6C 6C-00 75 73 65-72 33 32 2E-64 6C 6C 00   .dll.user32.dll.
```

Strukturen der Imports | Auswirkungen

Deskriptoren

```
0x203c → 0x204c, 0 (INT*)
0x2078 → kernel32.dll → 0, ExitProcess (Hint,Name)
0x2068 → 0x204c, 0 (IAT*)
0x2044 → 0x205a, 0 (INT*)
0x2085 → user32.dll → 0, MessageBoxA (Hint,Name)
0x2070 → 0x205a, 0 (IAT*)
00000
```

Alle Adressen hier sind RVAs.*

nach dem Ladevorgang zeigt
0x402068 auf ExitProcess in kernel32.dll
0x402070 auf MessageBoxA in user32.dll

Offset:0x600/RVA:0x403000

```
61 20 73 69-6D 70 6C 65-20 50 45 20-65 78 65 63   a.simple.PE.exec
75 74 61 62-6C 65 00 48-65 6C 6C 6F-20 77 6F 72   utable.Hello.wor
6C 64 21 00                                       ld!.
```

Zeichenketten

```
a simple PE executable\0
Hello world!\0
```

Dies ist die gesamte Datei. Die meisten PEs enthalten jedoch mehr Elemente. Erläuterungen sind wegen der Platzbeschränkungen vereinfacht.

version 1.0 DE, 23. Mai 2012

übersetzt von Daniel Plohmann

Ladevorgang

❶ Headers

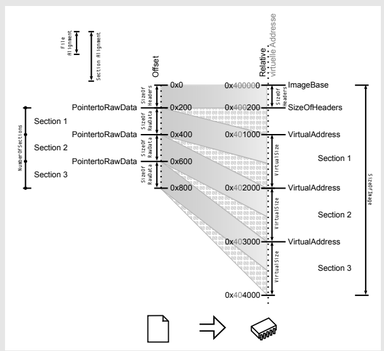
Der *DOS Header* wird geparst
Der *PE Header* wird geparst
(sein Offset ist e_lfanew aus dem *DOS Header*)
Der *Optional Header* wird geparst
(er folgt dem *PE Header*)

❷ Sektionstabelle

Sektionstabelle wird geparst
(befindet sich an der Speicherstelle: offset (*OptionalHeader*) + *SizeOfOptionalHeader*)
enthält *NumberOfSections* Elemente
wird auf Validität der Ausrichtungen geprüft:
FileAlignments und *SectionAlignments*

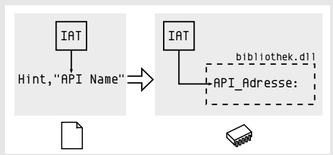
❸ Mapping

Die Datei wird in den Speicher geladen, abhängig von:
der *ImageBase*
der Größe von *SizeOfHeaders*
der Sektionstabelle



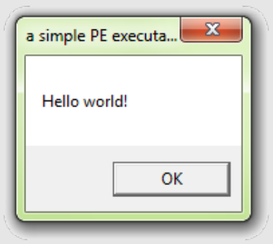
❹ Imports

Data Directories werden geparst
sie folgen dem Optional Header
ihre Anzahl ist *NumOfRVAAndSizes*
Imports sind immer an zweiter Stelle
Imports werden geparst
Jeder *Deskriptor* spezifiziert eine DLL
Die DLL wird in den Speicher geladen
IAT und *INT* werden gleichzeitig geparst
für jede gewünschte API-Funktion in der DLL
ihre Adresse wird in die *IAT* geschrieben



❺ Ausführung

Codeausführung beginnt an der Stelle *EntryPoint*
API-Funktionsaufrufe werden via IAT weitergeleitet



Bemerkungen

MZ HEADER *bzw.* DOS_HEADER
Beginnt mit „MZ" (Initialen von *Mark Zbikowski*, MS-DOS Entwickler)

PE HEADER *bzw.* IMAGE_FILE_HEADERS / COFF file header
Beginnt mit „PE" (Portable Executable)

OPTIONAL HEADER *bzw.* IMAGE_OPTIONAL_HEADER
Nur für nicht-standard PEs optional, aber erforderlich für ausführbare Dateien

RVA Relative virtuelle Adresse
Adresse relativ zur ImageBase (ImageBase hat RVA = 0)
Beinahe alle Adressen in den Headern sind RVAs
Im Code sind Adressen *nicht* relativ.

INT Import Name Table
Nullterminierte Liste von Verweisen zu Hint/Name Strukturen

IAT Import Address Table
Nullterminierte Liste von Verweisen
ist in der Datei eine Kopie der INT
Nach dem Ladevorgang zeigt sie auf die importierten API-Funktionen

HINT
Index in der Export Table einer zu importierenden DLL
Nicht erforderlich, erlaubt aber Geschwindigkeitsgewinn
durch Reduzierung der Namensauflösung