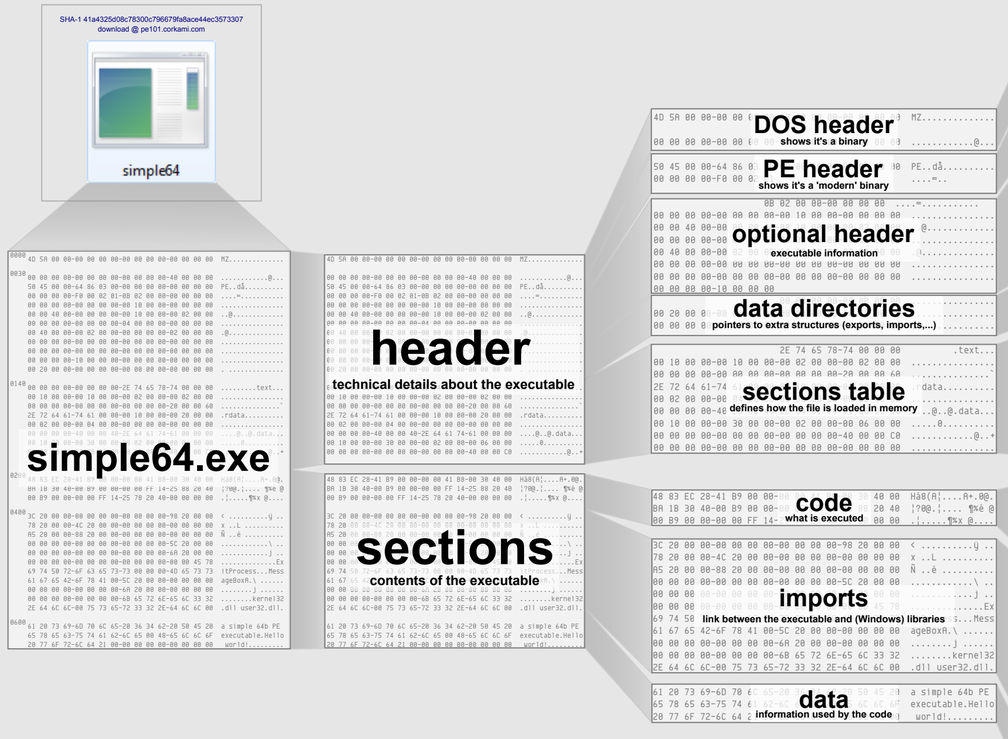
PE101 a windows executable walkthrough (64bits)

Ange Albertini
corkami.com

Dissected PE



Hexadecimal dump	ASCII dump	Fields	Values	Explanation
4D 5A 00 00-00 00 00 00-00 00 00 00-00 00 00 00	MZ.....	e_magic	'MZ'	constant signature
Offset:0x30 00 00 00 00-00 00 00 00-00 00 00 00-40 00 00 00	.....@...	e_lfanew	0x40	offset of the PE Header ❶
Offset:0x40 50 45 00 00 64 86 03 00 00 00 00 00 00 00 00 00	PE..d†.....	Signature	'PE', 0, 0	constant signature
00 00 00 00 F0 00 02 01...	.....à.	Machine	0x8664 [AMD 64]	processor: ARM/MIPS/Intel/...
		NumberOfSections	3	number of sections ❷
		SizeOfOptionalHeader	0xf0	relative offset of the section table ❷
		Characteristics	0x102 [32b EXE]	EXE/DLL/...
Offset:0x58 ...0B 02 00 00 00 00 00 00	.....	Magic	0x20b [64b]	32 bits/64 bits
00 00 00 00 00 00 00 00-10 10 00 00 00 00 00 00	.....	AddressOfEntryPoint	0x1000	where execution starts ❺
00 00 40 00 00 00 00 00-00 10 00 00 00 00 02 00	..@.....	ImageBase	0x400000	address where the file should be mapped in memory
00 00 00 00 00 00 00 00-04 00 00 00 00 00 00 00	.....	SectionAlignment	0x1000	where sections should start in memory ❷
00 40 00 00 00 02 00 00-00 00 00 00 02 00 00 00	.@.....	FileAlignment	0x200	where sections should start on file ❷
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00	.....	MajorSubsystemVersion	4 [NT 4 or later]	required version of Windows
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00	.....	SizeOfImage	0x4000	total memory space required
00 00 00 00 10 00 00 00	.....	SizeOfHeaders	0x200	total size of the headers ❸
		Subsystem	2 [GUI]	driver/graphical/command line/...
		NumberOfRvaAndSizes	16	number of 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* of the imports ❹
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00	.....			
Offset:0x148 2E 74 65 78-74 00 00 00	.text...			
00 10 00 00-00 10 00 00-00 02 00 00-00 02 00 00	.....			
00 00 00 00-00 00 00 00-00 00 00 00-20 00 00 60	.....`			
2E 72 64 61-74 61 00 00-00 10 00 00-00 20 00 00	.rdata.....			
00 02 00 00-00 04 00 00-00 00 00 00-00 00 00 00	.....			
00 00 00 00-40 00 00 40-2E 64 61 74-61 00 00 00	....@..@.data...			
00 10 00 00-00 30 00 00-00 02 00 00-00 06 00 00	.....0.....			
00 00 00 00-00 00 00 00-00 00 00 00-40 00 00 C0	.....@..+			

Sections table

Name	VirtualSize (RVA*)	VirtualAddress (RVA*)	SizeOfRawData (physical size)	PointerToRawData (physical 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

For each section, a SizeofRawData sized block is read from the file at PointerToRawData offset.
It will be loaded in memory at address ImageBase + VirtualAddress in a VirtualSize sized block, with specific characteristics.

Offset:0x200/RVA:0x401000	x86 assembly	Equivalent C code
48 83 EC 28-41 B9 00-00-00 00 41 B8-00 30 40 00	Hà8(A¦....A+.@.	sub rsp, 0x28 — // stack adjustment
BA 1B 30 40-00 B9 00-00-00 00 -FF 14-25 88 20 40	¦?0@.¦.... ¶%ê @	mov r9d, 0
00 B9 00 00-00 00 FF 14-25 78 20 40-00	.¦.....¶%x @.	mov r8d, 0x403000
		mov edx, 0x40301B
		mov ecx, 0
		call [0x402088] → MessageBox(0, "Hello World!","a simple PE executable", 0);
		mov ecx, 0
		call [0x402078] → ExitProcess(0);

Offset:0x400/RVA:0x402000	Imports structures	Consequences
3C 20 00 00-00 00 00 00-00 00 00 00-98 20 00 00	descriptors: 0x203C → 0x205c, 0 (INT*)	
78 20 00 00-4C 20 00 00-00 00 00 00-00 00 00 00	0x2098 → kernel32.dll → 0,ExitProcess (Hint,Name)	
A5 20 00 00-88 20 00 00-00 00 00 00-00 00 00 00	0x2078 → 0x205c, 0 (IAT*)	after loading,
00 00 00 00-00 00 00 00-00 00 00 00-5C 20 00 00	0x204C → 0x206a, 0 (INT*)	0x402078 will point to kernel32.dll´s ExitProcess
00 00 00 00-00 00 00 00-00 00 00 00-6A 20 00 00	0x20A5 → user32.dll → 0,MessageBoxA (Hint,Name)	0x402088 will point to user32.dll´s MessageBoxA
00 00 00 00-00 00 00 00-00 00 00 00-00 00 45 78	0x2088 → 0x206a, 0 (IAT*)	
69 74 50 72-6F 63 65 73-73 00 00 00-4D 65 73 73	00000	
61 67 65 42-6F 78 41 00-5C 20 00 00-00 00 00 00	All addresses here are RVAs.*	
00 00 00 00-00 00 00 00-6A 20 00 00-00 00 00 00		
00 00 00 00-00 00 00 00-6B 65 72 6E-65 6C 33 32		
2E 64 6C 6C-00 75 73 65-72 33 32 2E-64 6C 6C 00		

Offset:0x600/RVA:0x403000	Strings
61 20 73 69-6D 70 6C 65-20 36 34 62-20 50 45 20	a simple 64b PE executable\0
65 78 65 63-75 74 61 62-6C 65 00 48-65 6C 6C 6F	Hello world!\0
20 77 6F 72-6C 64 21 00	

This is the whole file, however, most PE files contain more elements. Explanations are simplified, for conciseness.

version 1 64b, 25th September 2012

Loading process

❶ Headers

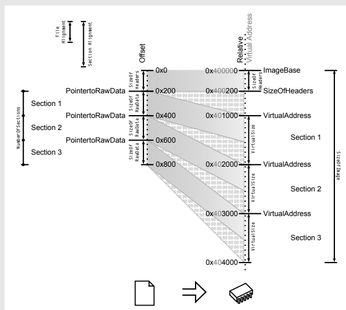
the *DOS Header* is parsed
the *PE Header* is parsed
(its offset is *DOS Header´s* e_lfanew)
the *Optional Header* is parsed
(it follows the *PE Header*)

❷ Sections table

Sections table is parsed
(it is located at: offset (*OptionalHeader*) + *SizeOfOptionalHeader*)
it contains *NumberOfSections* elements
it is checked for validity with alignments:
FileAlignments and *SectionAlignments*

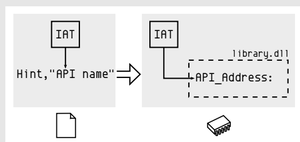
❸ Mapping

the file is mapped in memory according to:
the *ImageBase*
the *SizeOfHeaders*
the Sections table



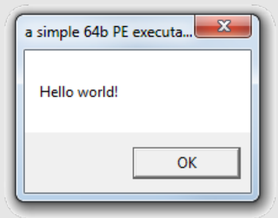
❹ Imports

DataDirectories are parsed
they follow the *OptionalHeader*
their number is *NumOfRVAAndSizes*
imports are always #2
Imports are parsed
each descriptor specifies a *DLLname*
this DLL is loaded in memory
IAT and *INT* are parsed simultaneously
for each API in *INT*
its address is written in the *IAT* entry



❺ Execution

Code is called at the *EntryPoint*
the calls of the code go via the IAT to the APIs



Notes

MZ HEADER *aka* DOS_HEADER
Starts with 'MZ' (initials of *Mark Zbikowski* MS-DOS developer)

PE HEADER *aka* IMAGE_FILE_HEADERS / COFF file header
Starts with 'PE' (Portable Executable)

OPTIONAL HEADER *aka* IMAGE_OPTIONAL_HEADER
Optional only for non-standard PEs but required for executables

RVA Relative Virtual Address
Address relative to ImageBase (at ImageBase, RVA = 0)
Almost all addresses of the headers are RVAs
In code, addresses are *not* relative.

INT Import Name Table
Null-terminated list of pointers to Hint, Name structures

IAT Import Address Table
Null-terminated list of pointers
On file it is a copy of the INT
After loading it points to the imported APIs

HINT
Index in the exports table of a DLL to be imported
Not required but provides a speed-up by reducing look-up