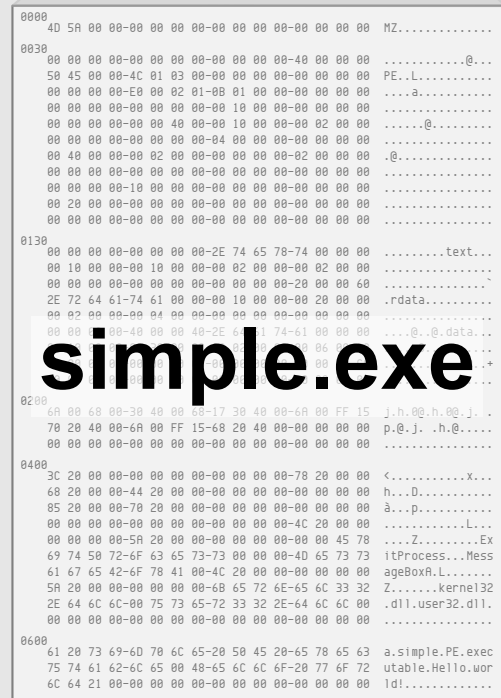


переведено Lyr1k

[illegible]

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-**данные** 6F 72 utable.Hello.world
6C 64 21 информация, которая необходима коду программы

Ассемблер x86		Эквивалентный код на C	
<pre> Смещение: 0x200(RVA: 0x410100) 6A 00 68 00-30 40 00 68-17 30 40 00-6A 00 FF 15 70 20 40 00-6A 00 FF 15 68 20 40 00 </pre>	<pre> j.h.@.h.@.j. . p.@.j. .h.@. </pre>	<pre> push 0 push 0x403000 push 0x403017 push 0 call [0x402070] push 0 call [0x402068] </pre>	<pre> MessageBox(0, "Hello world!", "a simple PE executable", 0); ExitProcess(0); </pre>

Смещение: 0x400(RVA):0x412000

3C	20	00	00	00	00	00	00	00	00	00	00	78	20	00	00
68	20	00	00	44	20	00	00	00	00	00	00	00	00	00	00
95	20	00	00	70	20	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	4C	20	00	00
00	00	00	00	5A	20	00	00	00	00	00	00	00	00	00	78
69	74	50	72	6F	63	65	73	73	00	00	00	4D	65	73	73
61	67	65	42	6F	78	41	00	4C	20	00	00	00	00	00	00
5A	20	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

Дескрипторы

0x203C	→	0x204C, 0 ^{INT*}	HintName 0,ExitProcess
0x2078	→	kernel32.dll	
0x2068	→	0x204C, 0 ^{IAT*}	HintName 0,MessageBoxA
0x2044	→	0x205a, 0 ^{INT*}	
0x2085	→	user32.dll	HintName 0,MessageBoxA
0x2070	→	0x205a, 0 ^{IAT*}	

Здесь все адреса записаны в RVA

Строки

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

после загрузки, 0x402068 будет указывать на ExitProcess в kernel32.dll
0x402070 будет указывать на MessageBoxA в user32.dll

Это полноценный PE-файл, однако, большинство PE-файлов содержат большее количество элементов. Для простоты объяснения опущены

версия 1.00 RU, 03.07.2013

1 Заголовки

2 Таблица секций


③ Проецирование

The diagram illustrates the structure of a file header and data sections. It shows a vertical stack of sections: Section 1, Section 2, and Section 3. Each section contains a 'PointerToRawData' and a 'VirtualAddress'. The 'PointerToRawData' values are 0x200, 0x400, and 0x600. The 'VirtualAddress' values are 0x0, 0x1000, 0x2000, 0x3000, and 0x4000. The diagram also shows a 'Section in Image' and a 'Section in Memory'.

4 Таблица импорта

The diagram illustrates the process of finding an API address. On the left, a box labeled 'IAT' points to the text 'Hint, "имя API"'. Below this is a document icon. An arrow points to the right, where a box labeled 'IAT' points to a dashed box containing 'library.dll' and 'API Адрес:'. Below this is a microchip icon.

5 Запуск



MZ HEADER он же DOS_HEADER
Начинается с "MZ" (инициалы MS DOS программиста Mark Zbikowski)

PE HEADER он же IMAGE_FILE_HEADERS / файловый заголовок COFF
Начинается с "PE" (Portable Executable, переносимый запускаемый файл)

OPTIONAL HEADER он же IMAGE_OPTIONAL_HEADER
Опциональный только для не стандартных PE-файлов, в других случаях является необходимым

INT Import Name Table

Список указателей на структуры «Hint, Имя функции» (заканчивается нулевым элементом)

IAT Import Address Table

Список указателей на структуры «Hint, Имя функции» (заканчивается нулевым элементом)

В файле - это копия таблицы INT

После загрузки она содержит адреса импортируемых API функций

HINT

Индекс в таблице экспорта импортируемой DLL

Не обязателен, но может ускорить процесс поиска имени функции в DLL