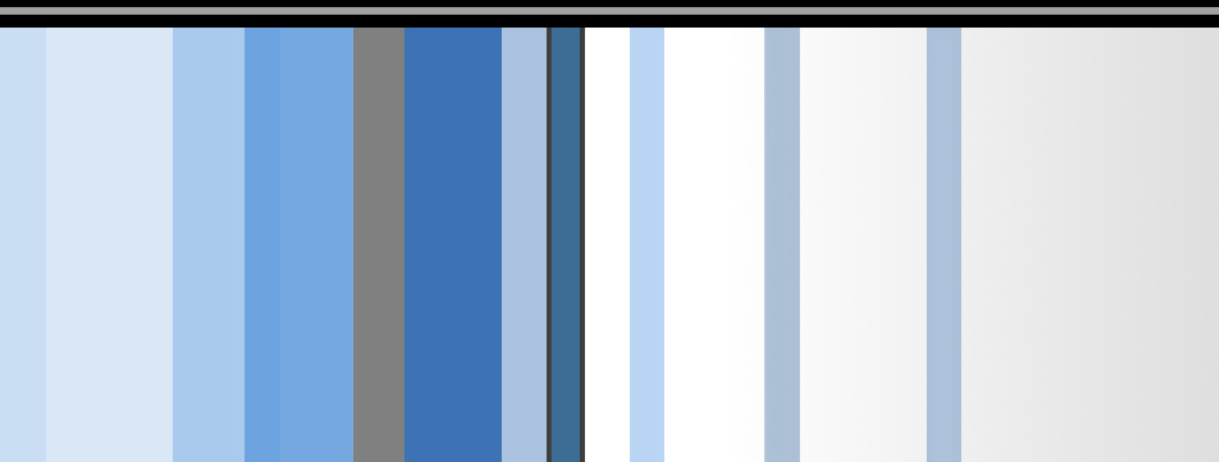


CSC 497/583 Advanced Topics in Computer Security

Modern Malware Analysis

Static Analysis, PE Format

Si Chen (schen@wcupa.edu)



Course Outline

- Static Analysis
 - Cryptographic Hash
 - Anti-Virus Scanning
 - Strings
 - PE file
 - Packer and Cryptor
- PE Format

Static Analysis

Fingerprinting the Malware -- Cryptographic Hash



Fingerprinting the Malware



- Fingerprinting involves generating the **cryptographic hash** values for the suspect binary based on its file content.
- Same cryptographic hashing algorithms:
 - MD5
 - SHA1
 - SHA256
- **Why not just use the file name?**
 - **Ineffective**, same malware sample can use different filenames, cryptographic hash is **calculated based on the file content**.
- File hash is frequently used as an indicator to share with other security researchers to help them identify the sample.

Tools and Python code

```
import hashlib
import sys

filename = sys.argv[1]

content = open(filename, "rb").read()

print(hashlib.md5(content).hexdigest())
print(hashlib.sha1(content).hexdigest())
print(hashlib.sha256(content).hexdigest())
```

■ Finding Strings ^[1]

- A string in a program is a sequence of characters such as “the.”
- A program contains strings if it prints a message, connects to a URL, or copies a file to a specific location.
- Searching through the strings can be **a simple way to get hints about the functionality of a program.**
 - For example, if the program accesses a URL, then you will see the URL accessed stored as a string in the program.
- You can use the **Strings** program to search an executable for strings, which are typically stored in either ASCII or Unicode format.

Static analysis (myhack.dll)

```
C:\Work>strings.exe myhack.dll_
```

```
modf
ldexp
_cabs
_hypot
fmod
frexp
_y0
_y1
_yn
_logb
_nextafter
index.html
http://www.naver.com/index.html
<myhack.dll> Injection!!! -- CSC 497/583 -- Si Chen
QI\
QI\
QI\
QI\
```

```
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
    HANDLE hThread = NULL;

    g_hMod = (HMODULE)hinstDLL;

    switch( fdwReason )
    {
        case DLL_PROCESS_ATTACH :
            OutputDebugString(L"<myhack.dll> Injection!!! -- CSC 497/583 -- Dr. Chen");
            hThread = CreateThread(NULL, 0, ThreadProc, NULL, 0, NULL);
            CloseHandle(hThread);
            break;
    }

    return TRUE;
}
```

Static analysis (myhack.dll)

```
4%5
7.787K7R7^7v7{7
7g8n8
9&9R9
9%.:. :g:r:e<
>&>+>6>A>U>
?l?
0G0^0i0q0!0
1A1^1
2"2+363
5<535c5
6"6j6
7<7
7.8
9:9I9I9s9
: #:A:h:}:
; > ; ; H ; a ; r ; ! ;
<"<></<J<Q<
=U>
1o2M3t3
6k6
7^7>7
8>83808J8Z8
;3;s;
<!<'<+<1<5<?<R<L<v<
```

Sometimes the strings detected by the Strings program are not actual strings.

strings in Linux and flare-floss



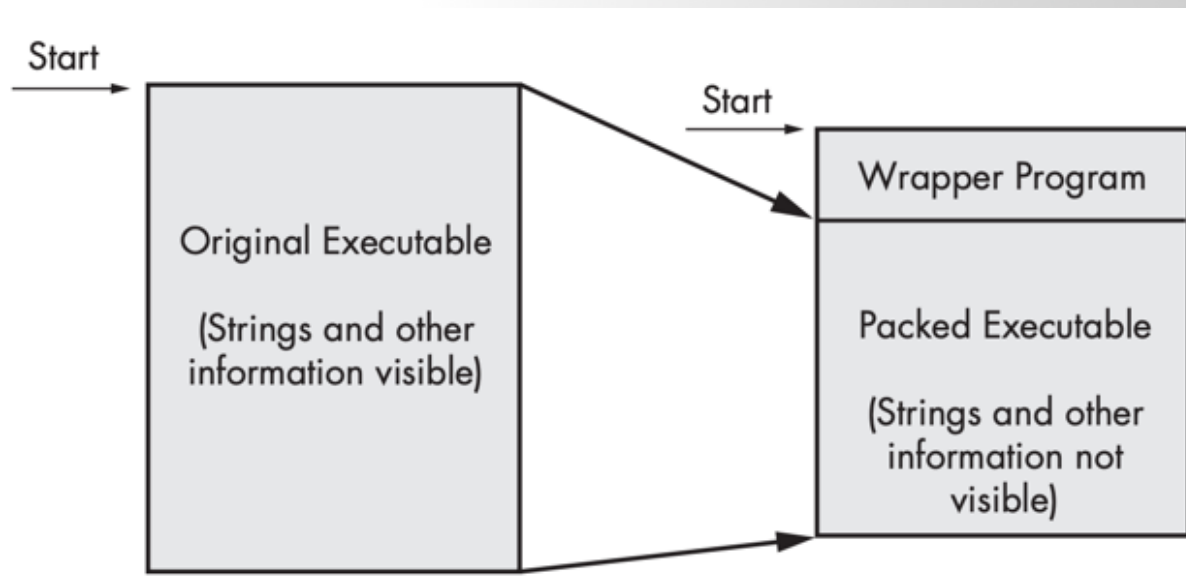
▪ FireEye Labs Obfuscated String Solver

- Many malware authors **evade heuristic detections** by obfuscating only key portions of an executable
 - These portions are strings and resources used to configure domains, files, and other artifacts of an infection
- The FireEye Labs Obfuscated String Solver (FLOSS) uses advanced static analysis techniques to **automatically deobfuscate strings** from malware binaries.

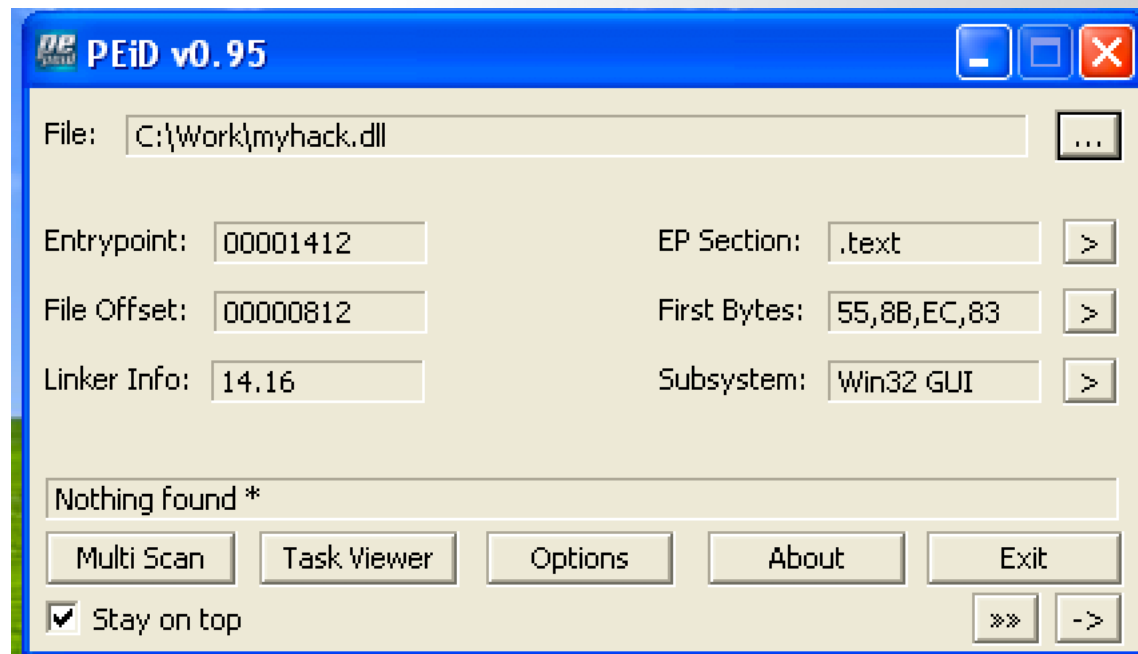
```
root@localhost ~# ./floss a99c01d5748b1bfd203fc1763e6612e8
FLOSS static ASCII strings
!This program cannot be run in DOS mode.
Rich
.text
.rdata
.data
.rsrc
SPWV
uNSW
j0Xf
RPSW
90t0
j Xf
PPPPP
Y__^[
9csm
u)jAXf;
u+9u
8csm
uTVWhA7@
PPPPP
<v*v
^SSSSS
tAVWP
Y[_ ^
PPPPP
8"u8
t j\Yf
t$9U
QQSVWh
i@i ^V
```


Packed and Obfuscated Malware

- Malware writers often use **packing or obfuscation** to make their files more difficult to detect or analyze.
- **Obfuscated** programs are ones whose execution the malware author has attempted to hide.
- **Packed** programs are a subset of obfuscated programs in which the malicious program is compressed and cannot be analyzed.
- Both techniques will severely limit your attempts to statically analyze the malware.



Packed and Obfuscated Malware



Packers and Cryptos

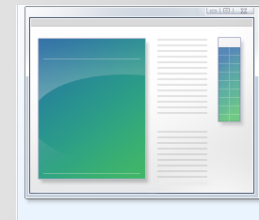
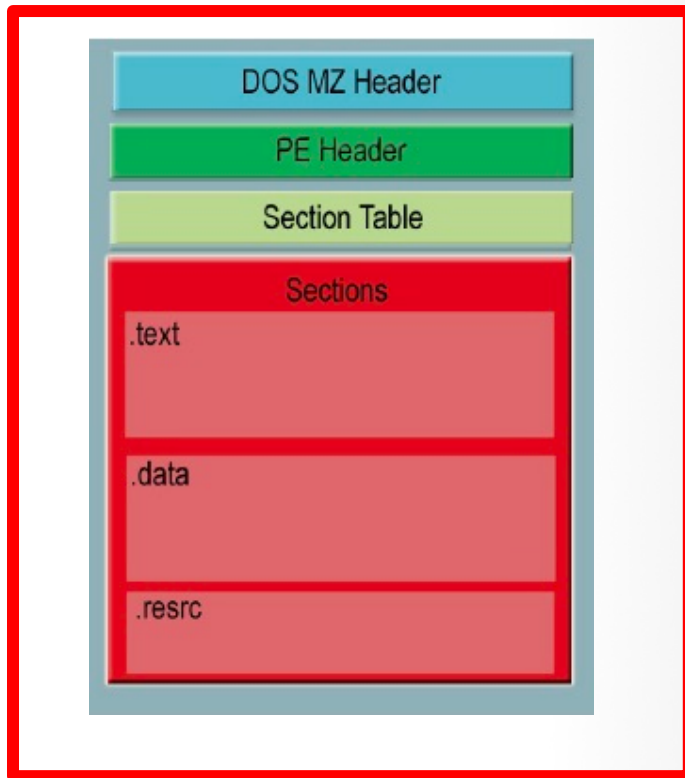
```
→ ~ upx -o myhack_packed.dll myhack.dll
      Ultimate Packer for eXecutables
      Copyright (C) 1996 - 2018
UPX 3.95      Markus Oberhumer, Laszlo Molnar & John Reiser      Aug 26th 2018

      File size      Ratio      Format      Name
      -----
      75264 ->      39424      52.38%      win32/pe      myhack_packed.dll

Packed 1 file.
```


Portable Executable (PE) file

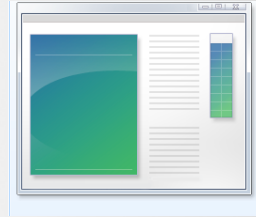
- A Portable Executable (**PE**) **file** is the standard binary **file** format for an **Executable (.exe) or DLL** under Windows NT, Windows 95, and Win32.



Portable Executable (PE) file

- PE formatted files include:

- .exe, .scr (executable)
- .dll, .ocx, .cpl, drv (library)
- .sys, .vxd (driver files)
- .obj (objective file)



- All PE formatted files can be executed, except obj file.

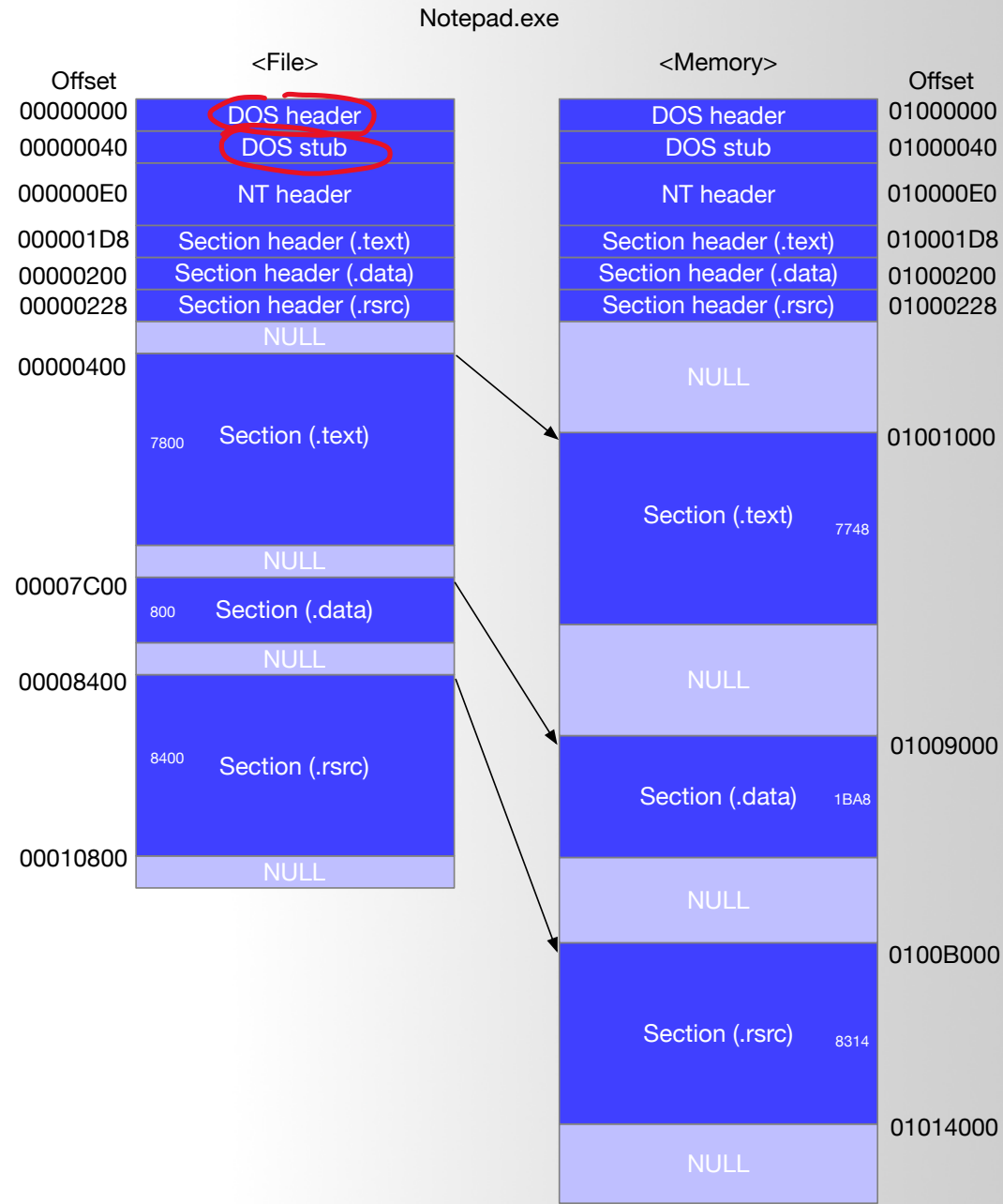
- .exe, .scr can be directly executed inside Shell (explorer.exe)
- others can be executed by other program/service

- **PE refers to 32 bit** executable file, or **PE32**. **64 bit** executable file is named as **PE+ or PE32+**. (Note that it is not PE64).

PE Example – Notepad.exe

00000000	4D 5A 90 00 03 00 00 00	04 00 00 00 FF FF 00 00	MZÉ..... ..
00000010	B8 00 00 00 00 00 00 00	40 00 00 00 00 00 00 00	7.....@.....
00000020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000030	00 00 00 00 00 00 00 00	00 00 00 00 E8 00 00 00Φ...
00000040	0E 1F BA 0E 00 B4 09 CD	21 B8 01 4C CD 21 54 68!=!7.L=!Th
00000050	69 73 20 70 72 6F 67 72	61 6D 20 63 61 6E 6E 6F	is.program.canno
00000060	74 20 62 65 20 72 75 6E	20 69 6E 20 44 4F 53 20	t.be.run.in.DOS.
00000070	6D 6F 64 65 2E 0D 0D 0A	24 00 00 00 00 00 00 00	mode....\$......
00000080	A5 6D 16 9B E1 0C 78 C8	E1 0C 78 C8 E1 0C 78 C8	Ñm.¢ß.xℒß.xℒß.xℒ
00000090	1B 2F 38 C8 E0 0C 78 C8	E1 0C 78 C8 E0 0C 78 C8	./8ℒα.xℒß.xℒα.xℒ
000000A0	1B 2F 61 C8 F2 0C 78 C8	E1 0C 79 C8 23 0C 78 C8	./aℒ≥.xℒß.yℒ#.xℒ
000000B0	76 2F 3D C8 E0 0C 78 C8	3B 2F 64 C8 F2 0C 78 C8	v/=ℒα.xℒ;/dℒ≥.xℒ
000000C0	1B 2F 45 C8 E0 0C 78 C8	52 69 63 68 E1 0C 78 C8	./Eℒα.xℒRichß.xℒ
000000D0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000E0	00 00 00 00 00 00 00 00	50 45 00 00 4C 01 03 00PE..L...
000000F0	0D 84 7D 3B 00 00 00 00	00 00 00 00 E0 00 0F 01	.ä};.....α...
00000100	0B 01 07 00 00 6E 00 00	00 A6 00 00 00 00 00 00n... ^a
00000110	E0 6A 00 00 00 10 00 00	00 80 00 00 00 00 00 01	αj.....Ç.....
00000120	00 10 00 00 00 02 00 00	05 00 01 00 05 00 01 00
00000130	04 00 00 00 00 00 00 00	00 30 01 00 00 04 00 00θ.....
00000140	55 D8 01 00 02 00 00 80	00 00 04 00 00 10 01 00	U†.....Ç.....
00000150	00 00 10 00 00 10 00 00	00 00 00 00 10 00 00 00
00000160	00 00 00 00 00 00 00 00	20 6D 00 00 C8 00 00 00m..ℒ...
00000170	00 A0 00 00 48 89 00 00	00 00 00 00 00 00 00 00	.á..Hë.....
00000180	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000190	40 13 00 00 1C 00 00 00	00 00 00 00 00 00 00 00	@.....

Load PE file (Notepad.exe) into Memory



VA & RVA

- VA (Virtual Address): The address is called a “VA” because **Windows creates a distinct VA space for each process, independent of physical memory**. For almost all purposes, a VA should be considered just an address. A VA is not as predictable as an RVA because the loader might not load the image at its preferred location.
- RVA (Relative Virtual Address): The address of an item after it is loaded into memory, with the base address of the image file subtracted from it. The RVA of an item almost always differs from its position within the file on disk (file pointer).

$$\text{RVA} + \text{ImageBase} = \text{VA}$$

In 32bit Windows OS, each process has 4GB virtual memory which means the range of VA is: **00000000 - FFFFFFFF**

DOS Header

```
struct DOS_Header
{
    // short is 2 bytes, long is 4 bytes
    char signature[2] = { 'M', 'Z' };
    short lastsize;
    short nblocks;
    short nreloc;
    short hdrsize;
    short minalloc;
    short maxalloc;
    void *ss; // 2 byte value
    void *sp; // 2 byte value
    short checksum;
    void *ip; // 2 byte value
    void *cs; // 2 byte value
    short relocpos;
    short noverlay;
    short reserved1[4];
    short oem_id;
    short oem_info;
    short reserved2[10];
    long e_lfanew; // Offset to the 'PE\0\0' signature relative to the beginning of the file
}
```

The first 2 letters are **always** the letters "MZ", the initials of Mark Zbikowski, who created the first linker for DOS. To some people, the first few bytes in a file that determine the type of file are called the "**magic number**,"

~~2075~~ 124

DOS Header

```
long e_lfanew;
```

long \rightarrow 32 bit $\rightarrow 32/8 = 4$ Bytes

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	Mz.....yy..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	000.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	E0	00	00	00à...

E0 00 00 00

value for e_lfanew \rightarrow ?

Handwritten notes:
E0 00 00 00
0x000000E0
224

8 bits = 1 Byte

32bits = $32 / 8 = 4$ Bytes = 0x00000000 – 0xFFFFFFFF

64bits = 8 Bytes

Byte Order

Little endian

- IA-32 processors use "little endian" as their byte order. This means that the bytes of a word are numbered starting from the least significant byte and that the least significant bit starts of a word starts in the least significant byte.

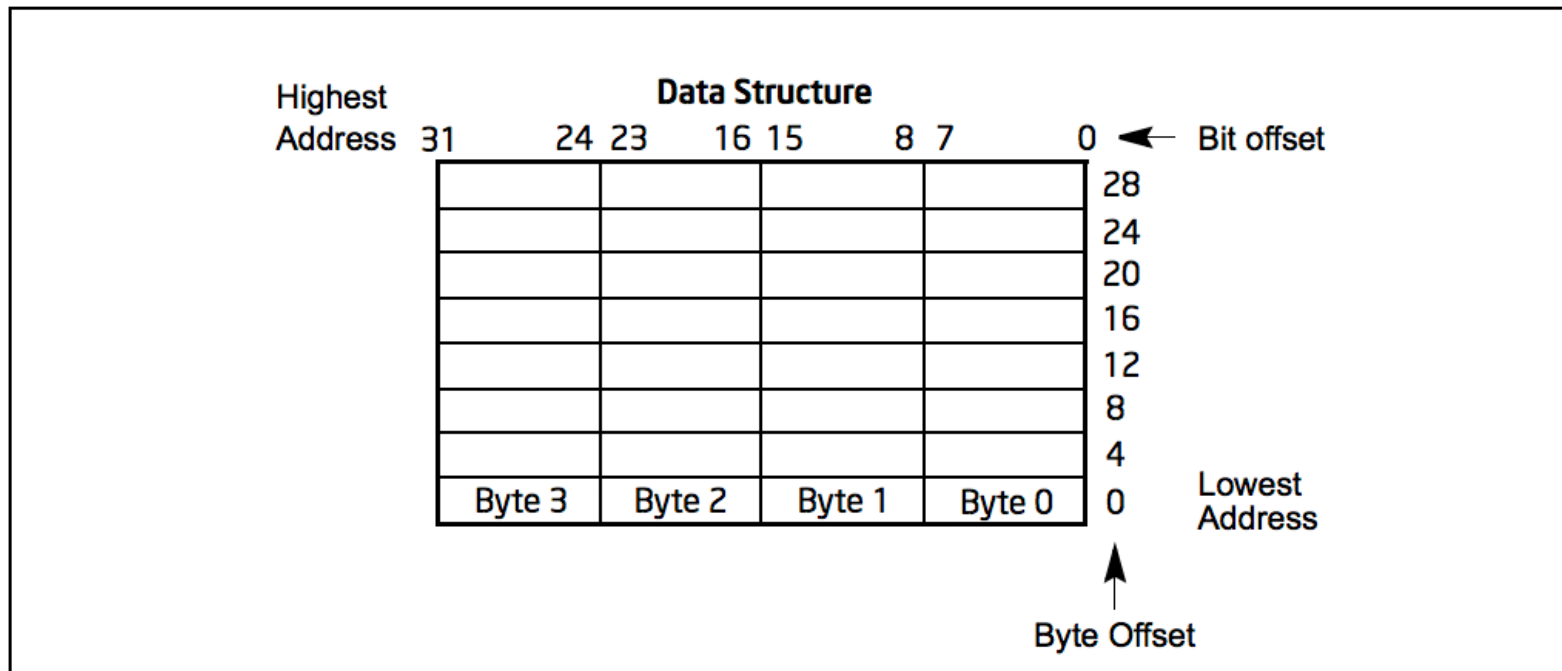


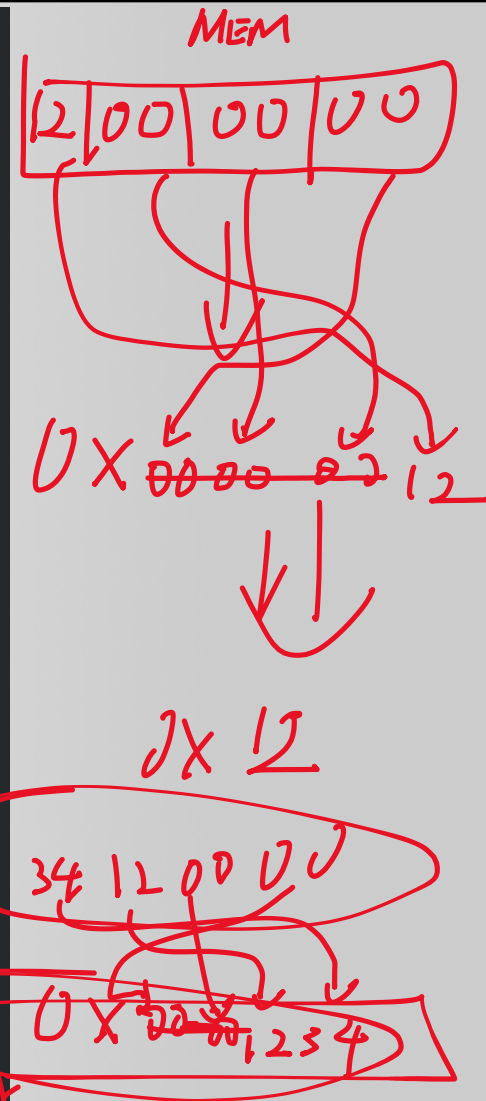
Figure 1-1. Bit and Byte Order

Byte Order

	Low address				High address			
Address	0	1	2	3	4	5	6	7
Little-endian	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Big-endian	Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
Memory content	0x11	0x22	0x33	0x44	0x55	0x66	0x77	0x88
64 bit value on Little-endian				64 bit value on Big-endian				
0x8877665544332211				0x1122334455667788				

LittleEndian.exe

```
1  #include "windows.h"
2
3  BYTE b = 0x12;
4  WORD w = 0x1234;
5  DWORD dw = 0x12345678;
6  char str[] = "abcde";
7
8
9  int main(int argc, char *argv[])
10 {
11     BYTE lb = b;
12     WORD lw = w;
13     DWORD ldw = dw;
14     char *lstr = str;
15
16
17     return 0;
18 }
```







LittleEndian.exe

OllyDbg - LittleEndian.exe - [CPU - main thread, module LittleEn]

File View Debug Plugins Options Window Help

LEMTWHC / KBR...S

00401000 \$ 55 PUSH EBP
 00401001 . 8BEC MOV EBP,ESP
 00401003 . 83EC 10 SUB ESP,10
 00401006 . A0 40AC4000 MOV AL, BYTE PTR DS:[40AC40] 
 0040100B . 8845 F3 MOV BYTE PTR SS:[EBP-DI],AL
 0040100E . 66:8B0D 44AC40 MOV CX, WORD PTR DS:[40AC44] 
 00401015 . 66:894D F4 MOV WORD PTR SS:[EBP-CI],CX
 00401019 . 8B15 48AC4000 MOV EDX, DWORD PTR DS:[40AC48] 
 0040101F . 8955 F8 MOV DWORD PTR SS:[EBP-8],EDX
 00401022 . C745 FC 4CAC40 MOV DWORD PTR SS:[EBP-4], LittleEn.0040AC4C 
 00401029 . 33C0 XOR EAX,EAX
 0040102B . 8BE5 MOV ESP,EBP
 0040102D . 5D POP EBP
 0040102E . C3 RETN
 0040102F \$ 3B0D 04A04000 CMP ECX, DWORD PTR DS:[40A004]
 00401035 . 75 02 JNZ SHORT LittleEn.00401039
 00401037 . F3: PREFIX REP:
 00401038 . C3 RETN
 00401039 > E9 85010000 JMP LittleEn.004011C3
 0040103E \$ 8BFF MOV EDI,EDI

Local call from 0040115D

Registers (FPU)

EAX 00000000
 ECX 0012FFB0
 EDX 7C90E514 ntdll.KiFastSystemCallRet
 EBX 7FFDD000
 ESP 0012FFC4
 EBP 0012FFF0
 ESI FFFFFFFF
 EDI 7C910228 ntdll.7C910228
 EIP 004011B9 LittleEn.<ModuleEntryPoint>

C 0 ES 0023 32bit 0(FFFFFFFF)
 P 1 CS 001B 32bit 0(FFFFFFFF)
 A 0 SS 0023 32bit 0(FFFFFFFF)
 Z 1 DS 0023 32bit 0(FFFFFFFF)
 S 0 FS 003B 32bit 7FFDF000(FFF)
 T 0 GS 0000 NULL
 D 0
 O 0 LastErr ERROR_SUCCESS (00000000)
 EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
 ST0 empty -UNORM BDEC 01050104 00300031

Address	Hex dump	ASCII
00401000	55 8B EC 83 EC 10 A0 40	Uï~â~â@
00401008	AC 40 00 88 45 F3 66 8B	%@.âEÿfi
00401010	0D 44 AC 40 00 66 89 4D	.D%@.fâM
00401018	F4 8B 15 48 AC 40 00 89	fiSH%@.ë
00401020	55 F8 C7 45 FC 4C AC 40	U° E"L%@
00401028	00 33 C0 8B E5 5D C3 3B	.3liσ t;
00401030	0D 04 A0 40 00 75 02 F3	.÷â@.u0<
00401038	C3 E9 85 01 00 00 8B FF	t0â@..i
00401040	55 8B EC 83 3D 68 AC 40	Uï~â=h%@
00401048	00 02 74 05 E8 AB 07 00	.0t+â%..

0012FFC4 7C81776F RETURN to kernel32.7C81776F
 0012FFC8 7C910228 ntdll.7C910228
 0012FFCC FFFFFFFF
 0012FFD0 7FFDD000
 0012FFD4 8054B838
 0012FFD8 0012FFC8
 0012FFDC 8661C180
 0012FFE0 FFFFFFFF End of SEH chain
 0012FFE4 7C839A90 SE handler
 0012FFE8 7C817778 kernel32.7C817778
 0012FFEC 00000000


Program entry point

start Shared Documents WBOXSVR\Windows... WindowsVM on Vbox... OllyDbg - LittleEndian... C:\Documents and Se...


Paused 9:19 PM

LittleEndian.exe


```
1 #include "windows.h"
2
3 BYTE b = 0x12;
4 WORD w = 0x1234;
5 DWORD dw = 0x12345678;
6 char str[] = "abcde";
7
8
9 int main(int argc, char *argv[])
10 {
11     BYTE lb = b;
12     WORD lw = w;
13     DWORD ldw = dw;
14     char *lstr = str;
15
16
17     return 0;
18 }
```




Address	Hex dump	ASCII
0040AC40	12 00 00 00 34 12 00 00	t...4t..
0040AC48	78 56 34 12 61 62 63 64	xV4tabcd
0040AC50	65 00 00 00 00 00 00 00	e.....
0040AC58	00 00 00 00 00 00 00 00
0040AC60	00 00 00 00 00 00 00 00
0040AC68	00 00 00 00 00 00 00 00
0040AC70	00 00 00 00 00 00 00 00
0040AC78	00 00 00 00 00 00 00 00
0040AC80	00 00 00 00 00 00 00 00
0040AC88	00 00 00 00 00 00 00 00



Address	Hex dump	ASCII
0040AC44	34 12 00 00 78 56 34 12	4t...xV4t
0040AC48	61 62 63 64 65 00 00 00	abcde...
0040AC54	00 00 00 00 00 00 00 00
0040AC5C	00 00 00 00 00 00 00 00
0040AC64	00 00 00 00 00 00 00 00
0040AC6C	00 00 00 00 00 00 00 00
0040AC74	00 00 00 00 00 00 00 00
0040AC7C	00 00 00 00 00 00 00 00
0040AC84	00 00 00 00 00 00 00 00
0040AC8C	00 00 00 00 00 00 00 00



Address	Hex dump	ASCII
0040AC48	78 56 34 12 61 62 63 64	xV4tabcd
0040AC50	65 00 00 00 00 00 00 00	e.....
0040AC58	00 00 00 00 00 00 00 00
0040AC60	00 00 00 00 00 00 00 00
0040AC68	00 00 00 00 00 00 00 00
0040AC70	00 00 00 00 00 00 00 00
0040AC78	00 00 00 00 00 00 00 00
0040AC80	00 00 00 00 00 00 00 00
0040AC88	00 00 00 00 00 00 00 00
0040AC90	00 00 00 00 00 00 00 00



Address	Hex dump	ASCII
0040AC4C	61 62 63 64 65 00 00 00	abcde...
0040AC54	00 00 00 00 00 00 00 00
0040AC5C	00 00 00 00 00 00 00 00
0040AC64	00 00 00 00 00 00 00 00
0040AC6C	00 00 00 00 00 00 00 00
0040AC74	00 00 00 00 00 00 00 00
0040AC7C	00 00 00 00 00 00 00 00
0040AC84	00 00 00 00 00 00 00 00
0040AC8C	00 00 00 00 00 00 00 00
0040AC94	00 00 00 00 00 00 00 00

DOS Header

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....yy..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	E0	00	00	00à...

e_lfanew → 000000E0

DOS stub

```
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  [].°...'.í!_.Lí!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.....
00000080  EC 85 5B A1 A8 E4 35 F2 A8 E4 35 F2 A8 E4 35 F2  i...;`ä5ò`ä5ò`ä5ò
00000090  6B EB 3A F2 A9 E4 35 F2 6B EB 55 F2 A9 E4 35 F2  kë:ò@ä5òkëUò@ä5ò
000000A0  6B EB 68 F2 BB E4 35 F2 A8 E4 34 F2 63 E4 35 F2  këhò»ä5ò`ä4òcä5ò
000000B0  6B EB 6B F2 A9 E4 35 F2 6B EB 6A F2 BF E4 35 F2  këkò@ä5òkëjòçä5ò
000000C0  6B EB 6F F2 A9 E4 35 F2 52 69 63 68 A8 E4 35 F2  këoò@ä5òRich`ä5ò
000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
```

<https://virtualconsoles.com/online-emulators/dos/>

```
C:\>notepad.exe
This program cannot be run in DOS mode.
```

IMAGE_NT_HEADERS32 structure

12/04/2018 • 2 minutes to read

Represents the PE header format.

Syntax

C++

 Copy

```
typedef struct _IMAGE_NT_HEADERS {  
    DWORD      Signature;  
    IMAGE_FILE_HEADER      FileHeader;  
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;  
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

Members

Signature

A 4-byte signature identifying the file as a PE image. The bytes are "PE\0\0".

FileHeader

An [IMAGE_FILE_HEADER](#) structure that specifies the file header.

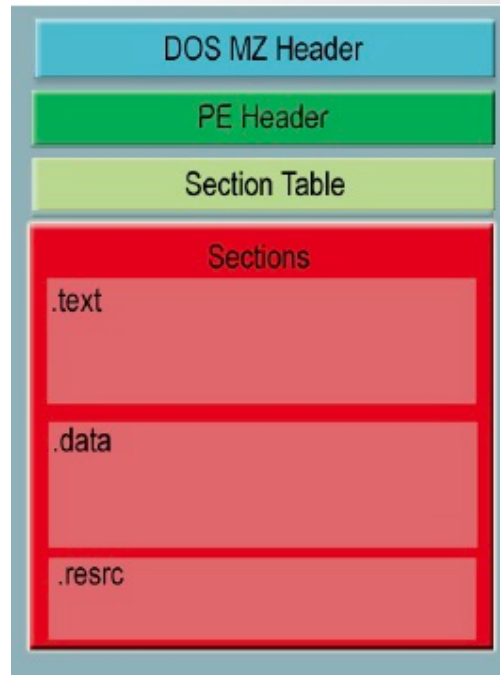
OptionalHeader

An [IMAGE_OPTIONAL_HEADER](#) structure that specifies the optional file header.

NT Header

NOTEPAD.EXE																	Decoded text
Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
000000E0	50	45	00	00	4C	01	03	00	A3	C3	B0	4A	00	00	00	00	PE..L...ËÃ°J....
000000F0	00	00	00	00	E0	00	0F	01	0B	01	07	0A	00	78	00	00à.....x..
00000100	00	A6	00	00	00	00	00	00	9D	73	00	00	00	10	00	00s.....
00000110	00	90	00	00	00	00	00	01	00	10	00	00	00	02	00	00
00000120	05	00	01	00	05	00	01	00	04	00	00	00	00	00	00	00
00000130	00	40	01	00	00	04	00	00	33	30	01	00	02	00	00	80	.@.....30.....€
00000140	00	00	04	00	00	10	01	00	00	00	10	00	00	10	00	00
00000150	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	00
00000160	04	76	00	00	C8	00	00	00	00	B0	00	00	58	89	00	00	.v..Ê....°..X%..
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000180	00	00	00	00	00	00	00	00	50	13	00	00	1C	00	00	00P.....
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001A0	00	00	00	00	00	00	00	00	A8	18	00	00	40	00	00	00"....@....
000001B0	00	00	00	00	00	00	00	00	00	10	00	00	48	03	00	00H....
000001C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001D0	00	00	00	00	00	00	00	00	2E	74	65	78	74	00	00	00text...

Section Header



Name	Privilege
.code	Executable, read
.data	Non-Executable, read/write
.resource	Non-Executable, read

IMAGE_SECTION_HEADER structure

12/04/2018 • 4 minutes to read

Represents the image section header format.

Syntax

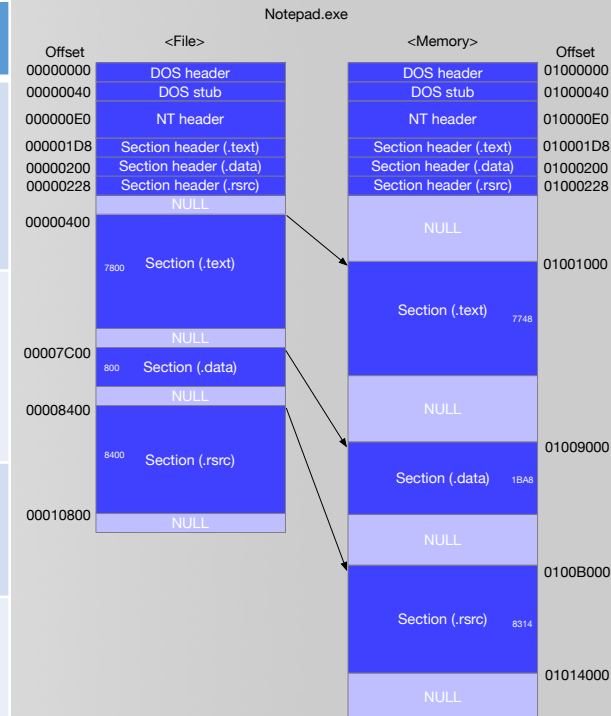
C++

 Copy

```
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;
```

Section Header

Members	Meaning
VirtualSize	The total size of the section when loaded into memory, in bytes.
VirtualAddress	The address of the first byte of the section when loaded into memory (RVA)
SizeOfRaw Data	The size of the section data on disk , in bytes.
PointerToRawData	The address of the first byte of the section on disk.
Characteristics	The characteristics of the image.



https://docs.microsoft.com/en-us/windows/desktop/api/winnt/ns-winnt-_image_section_header

Section Header

000001D0	00 00 00 00 00 00 00 00	2E 74 65 78 74 00 00 00text...
000001E0	48 77 00 00 00 10 00 00	00 78 00 00 00 04 00 00	Hw.....x.....
000001F0	00 00 00 00 00 00 00 00	00 00 00 00 20 00 00 60`
00000200	2E 64 61 74 61 00 00 00	A8 1B 00 00 00 90 00 00	.data..."
00000210	00 08 00 00 00 7C 00 00	00 00 00 00 00 00 00 00
00000220	00 00 00 00 40 00 00 C0	2E 72 73 72 63 00 00 00@..À.rsrc...
00000230	58 89 00 00 00 B0 00 00	00 8A 00 00 00 84 00 00	X%...°...Š...//..
00000240	00 00 00 00 00 00 00 00	00 00 00 00 40 00 00 40@..@

Inspecting PE Header Information in Linux

```
1 import pefile
2 import sys
3
4 malware_file = sys.argv[1]
5 pe = pefile.PE(malware_file)
6 for section in pe.sections:
7     print "Name: %s VirtualSize: %s VirtualAddr: %s SizeofRawData: %s PointerToRawData: %s" %
8         (section.Name, hex(section.Misc_VirtualSize), hex(section.VirtualAddress), section.SizeOfRawData, section.PointerToRawData)
```

```
root@localhost ~# python display_sections.py a99c01d5748b1bfd203fc1763e6612e8
```

```
Name: .text VirtualSize: 0x7378 VirtualAddr: 0x1000 SizeofRawData: 29696 PointerToRawData: 1024
Name: .rdata VirtualSize: 0x261c VirtualAddr: 0x9000 SizeofRawData: 10240 PointerToRawData: 30720
Name: .data VirtualSize: 0x2cac VirtualAddr: 0xc000 SizeofRawData: 3584 PointerToRawData: 40960
Name: .rsrc VirtualSize: 0x1b4 VirtualAddr: 0xf000 SizeofRawData: 512 PointerToRawData: 44544
```

Inspecting PE Header Information

PEview - C:\WINDOWS\notepad.exe

File View Go Help

NOTEPAD.EXE

- IMAGE_DOS_HEADER
- MS-DOS Stub Program
- IMAGE_NT_HEADERS
 - Signature
 - IMAGE_FILE_HEADER
 - IMAGE_OPTIONAL_HEADER
 - IMAGE_SECTION_HEADER**
 - IMAGE_SECTION_HEADER
 - IMAGE_SECTION_HEADER
- SECTION .text
 - IMPORT Address Table
 - IMAGE_DEBUG_DIRECTORY
 - IMAGE_LOAD_CONFIG_DIRECTORY
 - IMAGE_DEBUG_TYPES
 - IMPORT Directory Table
 - IMPORT Name Table
 - IMPORT Hints/Name Table
- SECTION .data
- SECTION .rsrc
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY

pFile	Data	Description	Value
000001D8	2E 74 65 78	Name	.text
000001DC	74 00 00 00		
000001E0	00007748	Virtual Size	
000001E4	00001000	RVA	
000001E8	00007800	Size of Raw Data	
000001EC	00000400	Pointer to Raw Data	
000001F0	00000000	Pointer to Relocations	
000001F4	00000000	Pointer to Line Numbers	
000001F8	0000	Number of Relocations	
000001FA	0000	Number of Line Numbers	
000001FC	60000020	Characteristics	
			IMAGE_SCN_CNT_CODE
			IMAGE_SCN_MEM_EXECUTE
			IMAGE_SCN_MEM_READ

Viewing IMAGE_SECTION_HEADER .text

Examining PE Section Table and Sections

- <https://hub.docker.com/r/remnux/pescanner/>

Q & A

