2023.10.10

Malicious Code Analysis

Fangtian Zhong
CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

Overview

01 DOS Header

02 NT Header

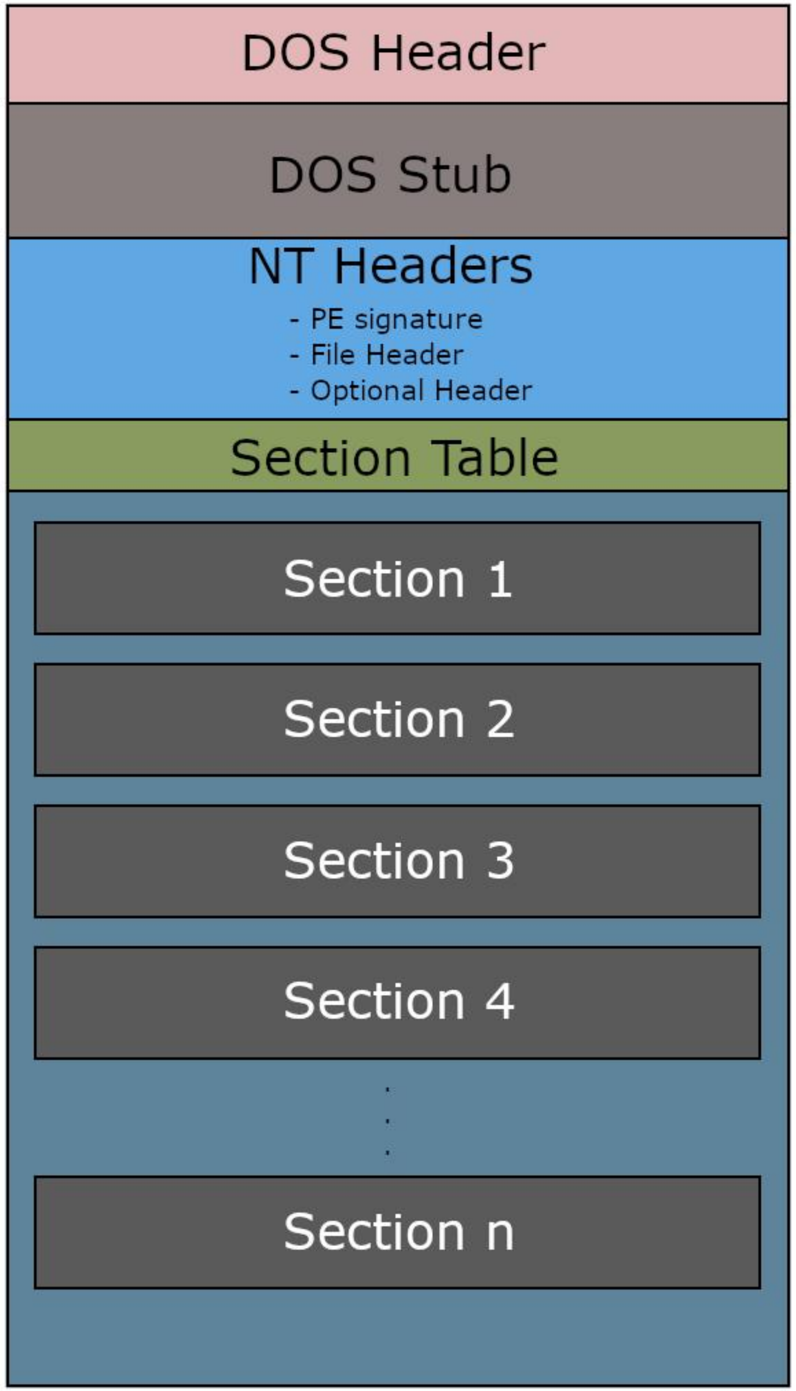
Part One

01

DOS Header

2025-8-22

PE



DOS Header

DOS Stub

NT Headers
- PE signature
- File Header
- Optional Header

Section Table

Section 1

Section 2

Section 3

Section 4

.
.
.

Section n

DOS Header

The DOS header (also called the MS-DOS header) is a 64-byte-long structure that exists at the start of the PE file.

DOS Header

```
typedef struct _IMAGE_DOS_HEADER {      // DOS .EXE header
   WORD   e_magic;                     // Magic number
   WORD   e_cblp;                      // Bytes on last page of file
   WORD   e_cp;                        // Pages in file
   WORD   e_crlc;                      // Relocations
   WORD   e_cparhdr;                   // Size of header in paragraphs
   WORD   e_minalloc;                  // Minimum extra paragraphs needed
   WORD   e_maxalloc;                  // Maximum extra paragraphs needed
   WORD   e_ss;                        // Initial (relative) SS value
   WORD   e_sp;                        // Initial SP value
   WORD   e_csum;                      // Checksum
   WORD   e_ip;                        // Initial IP value
   WORD   e_cs;                        // Initial (relative) CS value
   WORD   e_lfarlc;                    // File address of relocation table
   WORD   e_ovno;                      // Overlay number
   WORD   e_res[4];                    // Reserved words
   WORD   e_oemid;                     // OEM identifier (for e_oeminfo)
   WORD   e_oeminfo;                   // OEM information; e_oemid specific
   WORD   e_res2[10];                  // Reserved words
   LONG   e_lfanew;                    // File address of new exe header
 } IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

Fields

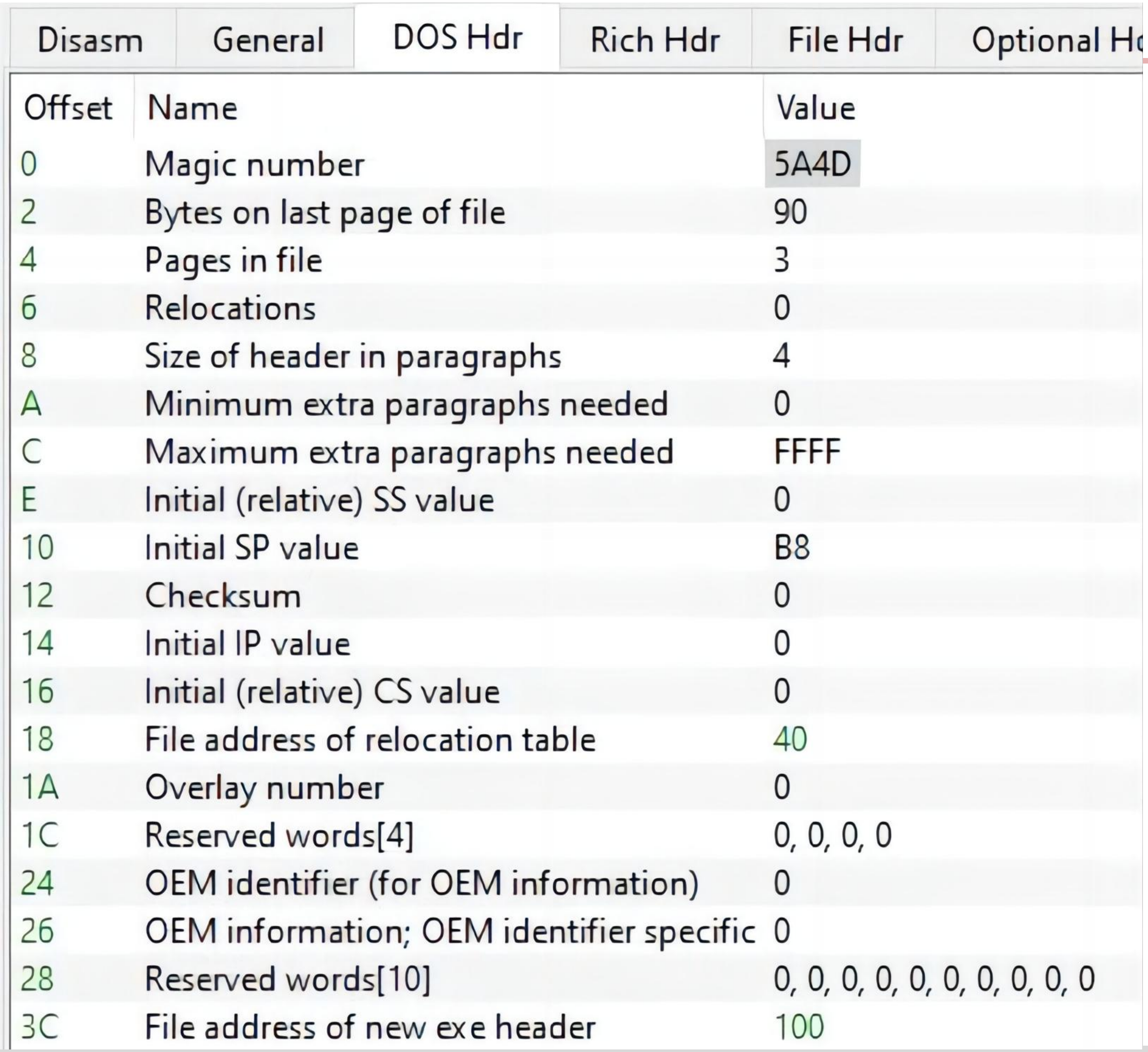
e_magic

❑ This is the first member of the DOS Header, it’s a WORD so it occupies 2 bytes, it’s usually called the magic number. It has a fixed value of 0x5A4D or MZ in ASCII, and it serves as a signature that marks the file as an MS-DOS executable.

e_lfanew

❑ This is the last member of the DOS header structure, it’s located at offset 0x3C into the DOS header and it holds an offset to the start of the NT headers. This member is important to the PE loader on Windows systems because it tells the loader where to look for the file header.

Examples



Offset	Name	Value
0	Magic number	5A4D
2	Bytes on last page of file	90
4	Pages in file	3
6	Relocations	0
8	Size of header in paragraphs	4
A	Minimum extra paragraphs needed	0
C	Maximum extra paragraphs needed	FFFF
E	Initial (relative) SS value	0
10	Initial SP value	B8
12	Checksum	0
14	Initial IP value	0
16	Initial (relative) CS value	0
18	File address of relocation table	40
1A	Overlay number	0
1C	Reserved words[4]	0, 0, 0, 0
24	OEM identifier (for OEM information)	0
26	OEM information; OEM identifier specific	0
28	Reserved words[10]	0, 0, 0, 0, 0, 0, 0, 0, 0, 0
3C	File address of new exe header	100

Explanations

- As you can see, the first member of the header is the magic number with the fixed value we talked about which was 5A4D.
- The last member of the header (at offset 0x3C) is given the name “File address of new exe header”, it has the value 100, we can follow to that offset and we’ll find the start of the NT headers as expected:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100	50	45	00	00	64	86	06	00	03	DE	59	61	00	00	00	00
110	00	00	00	00	F0	00	22	00	0B	02	0E	1A	00	0E	00	00

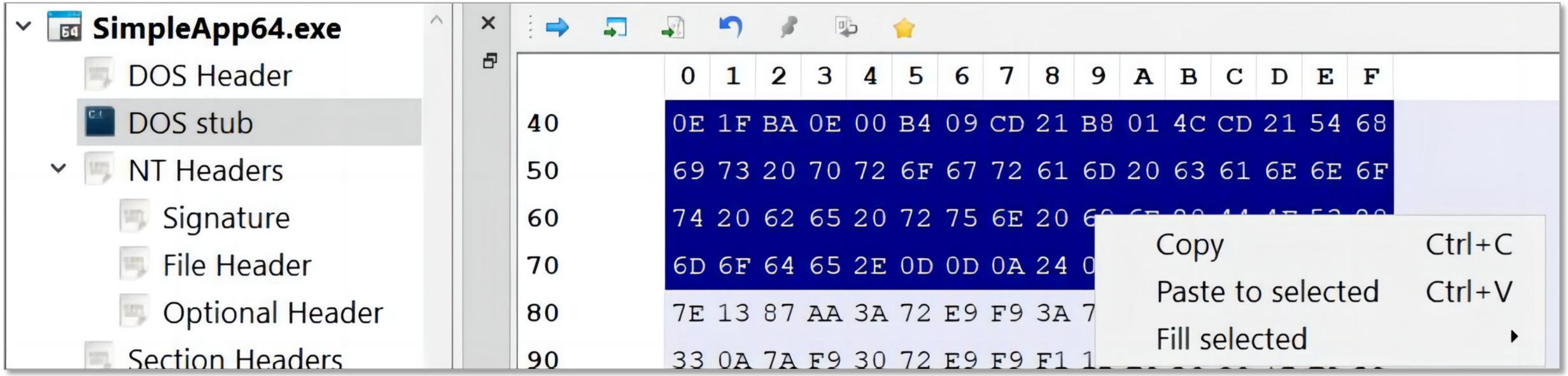
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
P	E	.	.	d	.	.	.	.	&	Y	a	.	.	.	.
.	.	.	.	a	.	”	.	.	.	.	.	.	.	.	.

DOS Stub

- This is a small segment of code at the beginning of the file that is executed by the MS-DOS operating system to display an error message if the file is not compatible with the operating system.
- This is what gets executed when the program is loaded in MS-DOS, the default error message is “This program cannot be run in DOS mode.”, however this message can be changed by the user during compile time.

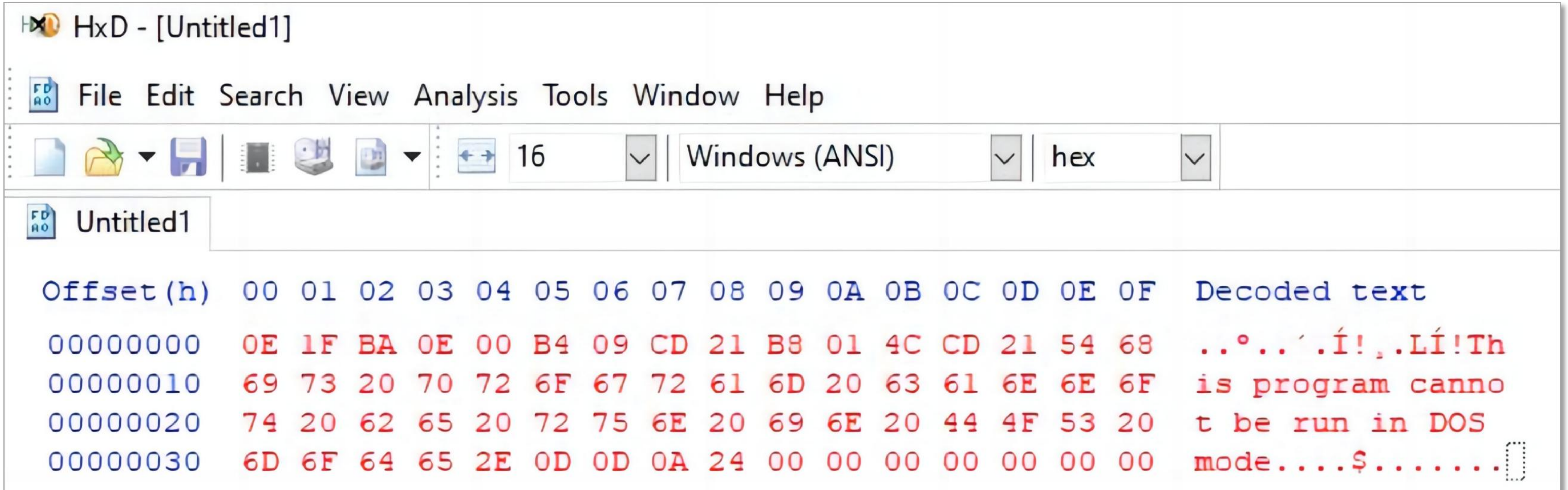
DOS Stub

- To be able to disassemble the machine code of the DOS stub, I copied the code of the stub from PE-bear, then I created a new file with the stub contents using a hex editor (HxD).



```
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
40  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68
50  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F
60  74 20 62 65 20 72 75 6E 20
70  6D 6F 64 65 2E 0D 0D 0A 24
80  7E 13 87 AA 3A 72 E9 F9 3A
90  33 0A 7A F9 30 72 E9 F9 F1
```

DOS Stub



HxD - [Untitled1]

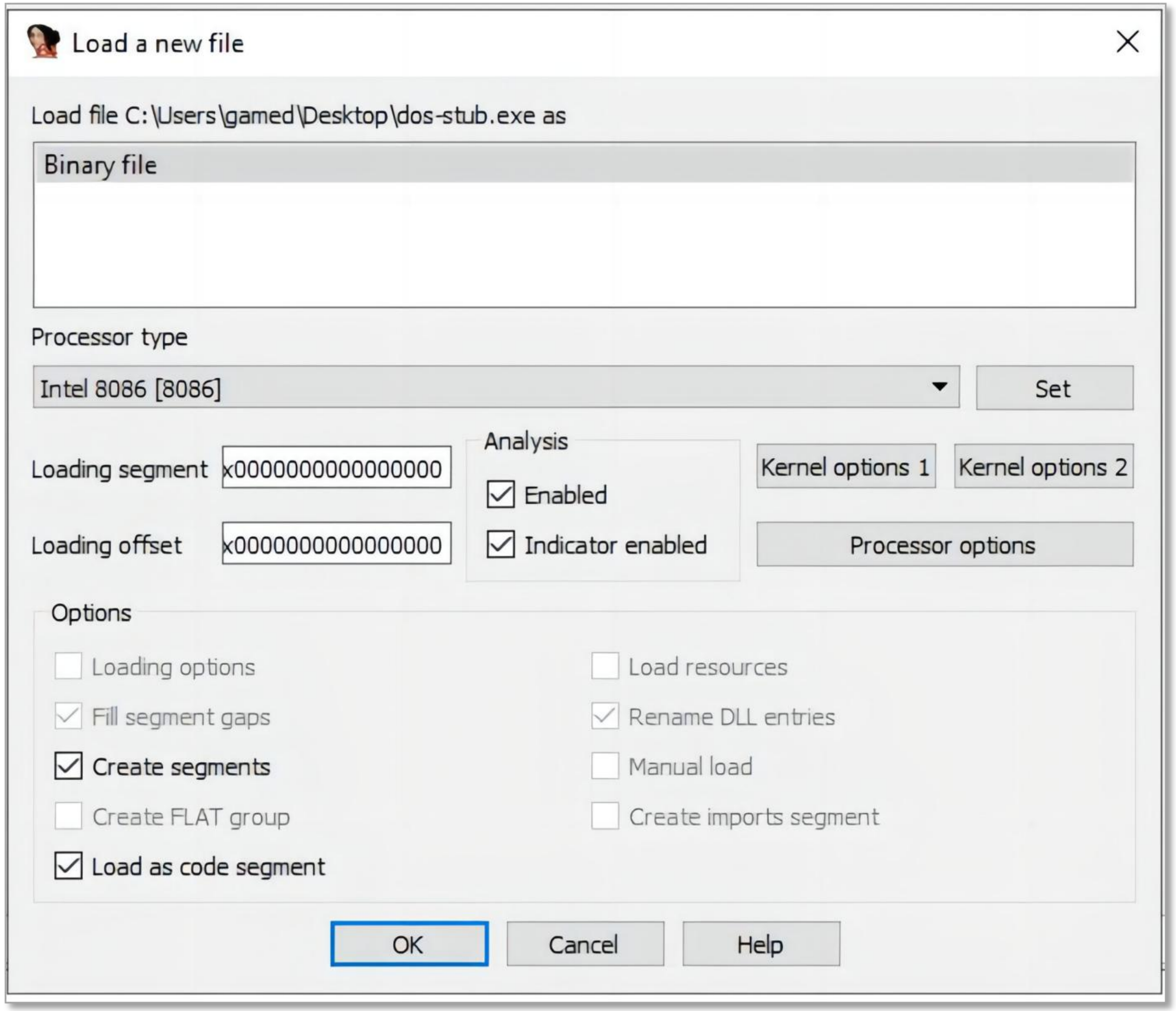
16 | Windows (ANSI) | hex

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000010  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000020  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000030  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
```

Assembly

- After that I used IDA to disassemble the executable, MS-DOS programs are 16-bit programs, so I chose the intel 8086 processor type and the 16-bit disassembly mode.



Load a new file

Load file C:\Users\gamed\Desktop\dos-stub.exe as

Binary file

Processor type: Intel 8086 [8086]

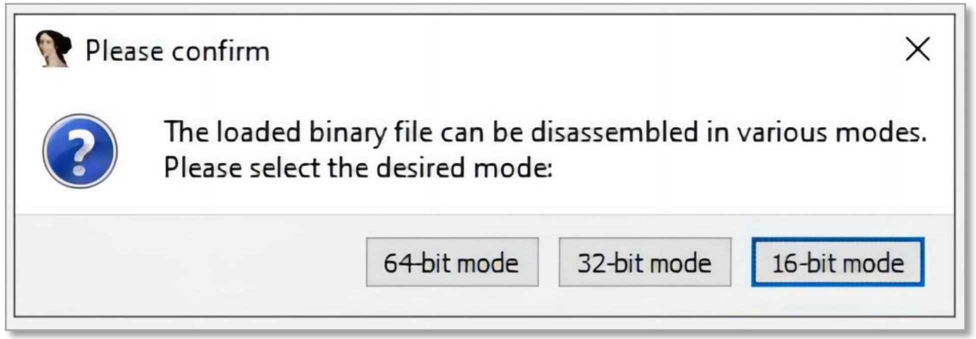
Loading segment: x0000000000000000

Loading offset: x0000000000000000

Analysis
- ☑ Enabled
- ☑ Indicator enabled

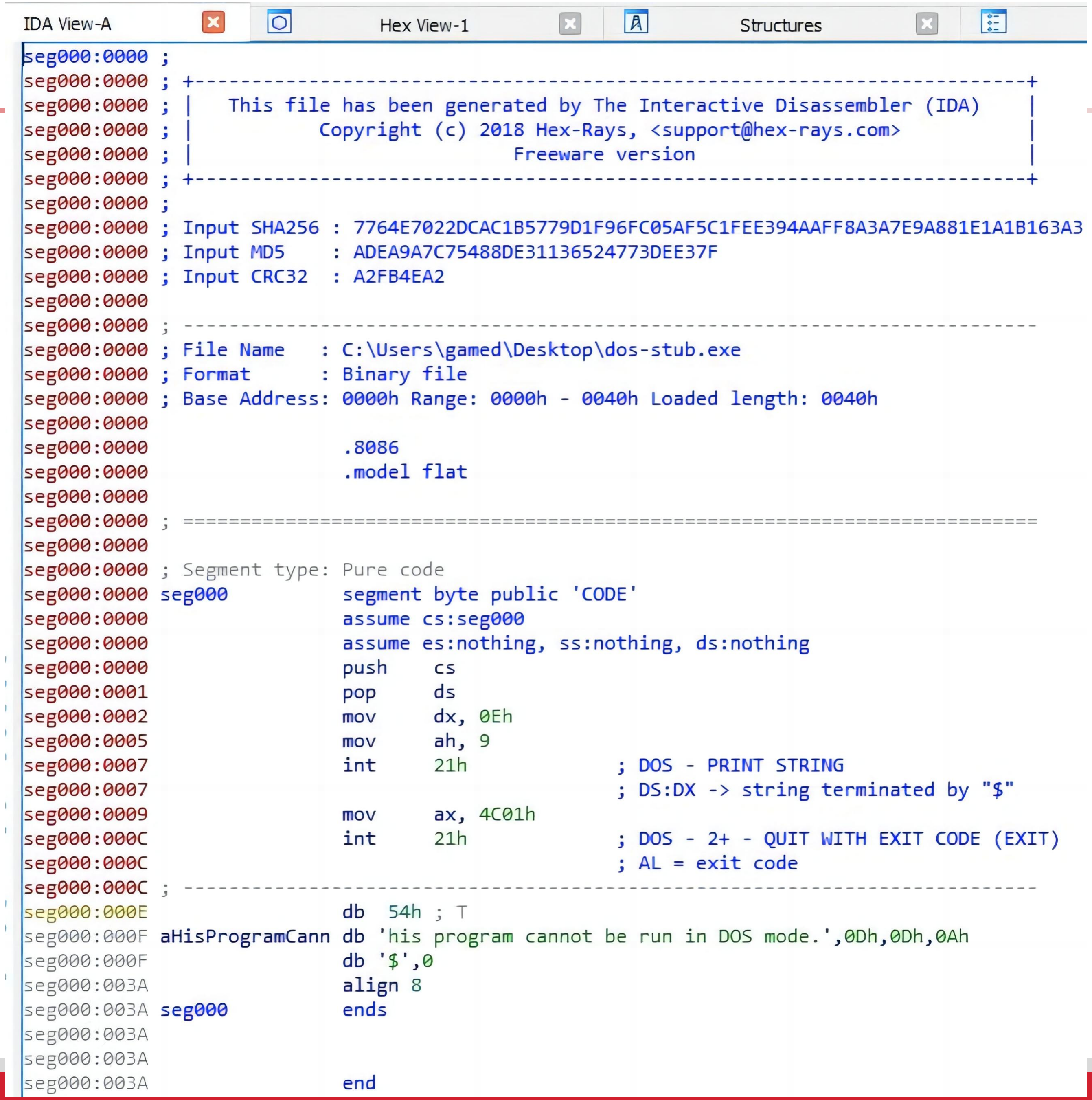
Options
- ☐ Loading options
- ☑ Fill segment gaps
- ☑ Create segments
- ☐ Create FLAT group
- ☑ Load as code segment
- ☐ Load resources
- ☑ Rename DLL entries
- ☐ Manual load
- ☐ Create imports segment

Assembly



Please confirm

The loaded binary file can be disassembled in various modes.
Please select the desired mode:



```
seg000:0000 ;
seg000:0000 ; +-------------------------------------------------------------------------+
seg000:0000 ; |   This file has been generated by The Interactive Disassembler (IDA)    |
seg000:0000 ; |           Copyright (c) 2018 Hex-Rays, <support@hex-rays.com>           |
seg000:0000 ; |                            Freeware version                             |
seg000:0000 ; +-------------------------------------------------------------------------+
seg000:0000 ;
seg000:0000 ; Input SHA256 : 7764E7022DCAC1B5779D1F96FC05AF5C1FEE394AAFF8A3A7E9A881E1A1B163A3
seg000:0000 ; Input MD5    : ADEA9A7C75488DE31136524773DEE37F
seg000:0000 ; Input CRC32  : A2FB4EA2
seg000:0000
seg000:0000 ; ---------------------------------------------------------------------------
seg000:0000 ; File Name   : C:\Users\gamed\Desktop\dos-stub.exe
seg000:0000 ; Format      : Binary file
seg000:0000 ; Base Address: 0000h Range: 0000h - 0040h Loaded length: 0040h
seg000:0000
seg000:0000                 .8086
seg000:0000                 .model flat
seg000:0000
seg000:0000 ; ===========================================================================
seg000:0000
seg000:0000 ; Segment type: Pure code
seg000:0000 seg000          segment byte public 'CODE'
seg000:0000                 assume cs:seg000
seg000:0000                 assume es:nothing, ss:nothing, ds:nothing
seg000:0000                 push    cs
seg000:0001                 pop     ds
seg000:0002                 mov     dx, 0Eh
seg000:0005                 mov     ah, 9
seg000:0007                 int     21h             ; DOS - PRINT STRING
seg000:0007                                         ; DS:DX -> string terminated by "$"
seg000:0009                 mov     ax, 4C01h
seg000:000C                 int     21h             ; DOS - 2+ - QUIT WITH EXIT CODE (EXIT)
seg000:000C                                         ; AL = exit code
seg000:000C ; ---------------------------------------------------------------------------
seg000:000E                 db  54h ; T
seg000:000F aHisProgramCann db 'his program cannot be run in DOS mode.',0Dh,0Dh,0Ah
seg000:000F                 db '$',0
seg000:003A                 align 8
seg000:003A seg000          ends
seg000:003A
seg000:003A
seg000:003A                 end
```

Assembly

seg000:0000	push	cs
seg000:0001	pop	ds

- First line pushes the value of cs onto the stack and the second line pops that value from the top of stack into ds. This is just a way of setting the value of the data segment to the same value as the code segment.

Assembly

```
seg000:0002        mov    dx, 0Eh
seg000:0005        mov    ah, 9
seg000:0007        int    21h          ; DOS - PRINT STRING
seg000:0007                           ; DS:DX -> string terminated by "$"
```

- These three lines are responsible for printing the error message, first line sets dx to the address of the string “This program cannot be run in DOS mode.” (0xe), second line sets ah to 9 and the last line invokes interrupt 21h.
- Interrupt 21h is a DOS interrupt (API call) that can do a lot of things, it takes a parameter that determines what function to execute and that parameter is passed in the ah register. We see here that the value 9 is given to the interrupt, 9 is the code of the function that prints a string to the screen, that function takes a parameter which is the address of the string to print, that parameter is passed in the dx register as we can see in the code.

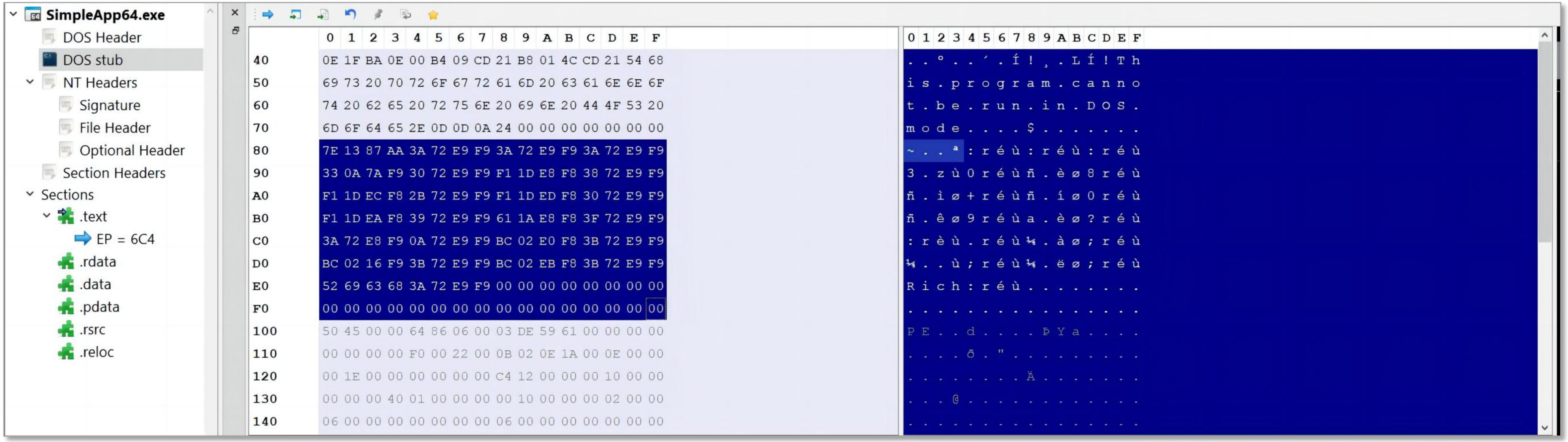
Assembly

```
seg000:0009                mov    ax, 4C01h
seg000:000C                int    21h           ; DOS - 2+ - QUIT WITH EXIT CODE (EXIT)
seg000:000C                                     ; AL = exit code
```

- The last three lines of the program are again an interrupt 21h call. There's a mov instruction that puts 0X4C01 into ax. This sets al to 0x01 and ah to 0x4c.
- 0x4c is the function code of the function that exits with an error code, it takes the error code from al, which in this case is 1.
- In summary, all the DOS stub is doing is print the error message then exit with code 1.

Rich Header

- We've seen the DOS Header and the DOS Stub, however there's still a chunk of data we haven't talked about lying between the DOS Stub and the start of the NT Headers.



```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    0123456789ABCDEF
40   0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68   ..°..´.Í!¸.LÍ!Th
50   69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F   is.program.canno
60   74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20   t.be.run.in.DOS.
70   6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00   mode....$.......
80   7E 13 87 AA 3A 72 E9 F9 3A 72 E9 F9 3A 72 E9 F9   ~..ª:réù:réù:réù
90   33 0A 7A F9 30 72 E9 F9 F1 1D E8 F8 38 72 E9 F9   3.zù0réùñ.èø8réù
A0   F1 1D EC F8 2B 72 E9 F9 F1 1D ED F8 30 72 E9 F9   ñ.ìø+réùñ.íø0réù
B0   F1 1D EA F8 39 72 E9 F9 61 1A E8 F8 3F 72 E9 F9   ñ.êø9réùa.èø?réù
C0   3A 72 E8 F9 0A 72 E9 F9 BC 02 E0 F8 3B 72 E9 F9   :rèù.réù¼.àø;réù
D0   BC 02 16 F9 3B 72 E9 F9 BC 02 EB F8 3B 72 E9 F9   ¼..ù;réù¼.ëø;réù
E0   52 69 63 68 3A 72 E9 F9 00 00 00 00 00 00 00 00   Rich:réù........
F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
100  50 45 00 00 64 86 06 00 03 DE 59 61 00 00 00 00   PE..d....ÞYa....
110  00 00 00 00 F0 00 22 00 0B 02 0E 1A 00 0E 00 00   ....ð.".........
120  00 1E 00 00 00 00 00 00 C4 12 00 00 00 10 00 00   ........Ä.......
130  00 00 00 40 01 00 00 00 00 10 00 00 00 02 00 00   ...@............
140  06 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00   ................
```

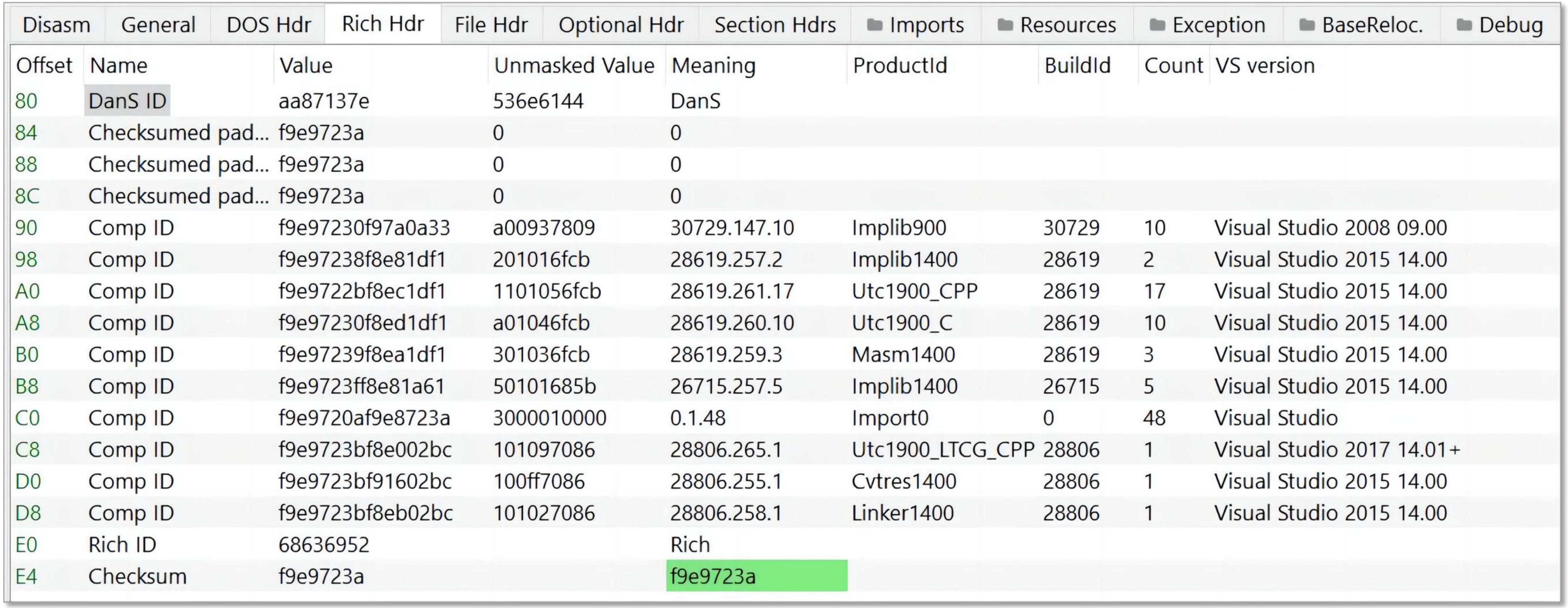
Rich Header

- That’s only present in executables built using the Microsoft Visual Studio toolset.
- This structure holds some metadata about the tools used to build the executable like their names or types and their specific versions and build numbers.
- The Rich Header consists of a chunk of XORed data followed by a signature (Rich) and a 32-bit checksum value that is the XOR key.

Rich Header

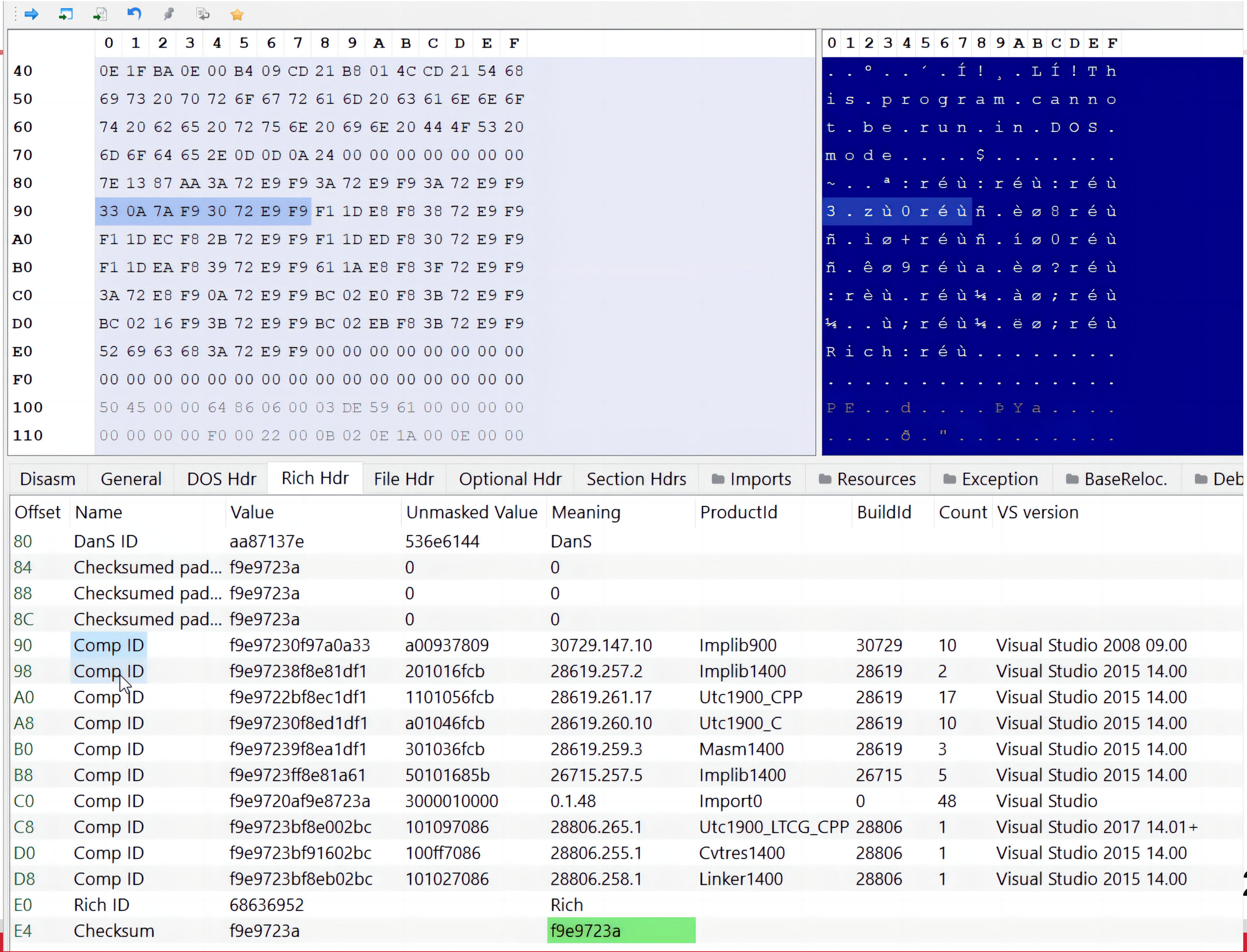
- The encrypted data consists of a DWORD signature DanS, 3 zeroed-out DWORDs for padding, then pairs of DWORDS each pair representing an entry, and each entry holds a tool name, its build number and the number of times it's been used.
- In each DWORD pair the first pair holds the type ID or the product ID in the high WORD and the build ID in the low WORD, the second pair holds the use count.

Rich Header



Offset	Name	Value	Unmasked Value	Meaning	ProductId	BuildId	Count	VS version
80	DanS ID	aa87137e	536e6144	DanS				
84	Checksumed pad...	f9e9723a	0	0				
88	Checksumed pad...	f9e9723a	0	0				
8C	Checksumed pad...	f9e9723a	0	0				
90	Comp ID	f9e97230f97a0a33	a00937809	30729.147.10	Implib900	30729	10	Visual Studio 2008 09.00
98	Comp ID	f9e97238f8e81df1	201016fcb	28619.257.2	Implib1400	28619	2	Visual Studio 2015 14.00
A0	Comp ID	f9e9722bf8ec1df1	1101056fcb	28619.261.17	Utc1900_CPP	28619	17	Visual Studio 2015 14.00
A8	Comp ID	f9e97230f8ed1df1	a01046fcb	28619.260.10	Utc1900_C	28619	10	Visual Studio 2015 14.00
B0	Comp ID	f9e97239f8ea1df1	301036fcb	28619.259.3	Masm1400	28619	3	Visual Studio 2015 14.00
B8	Comp ID	f9e9723ff8e81a61	50101685b	26715.257.5	Implib1400	26715	5	Visual Studio 2015 14.00
C0	Comp ID	f9e9720af9e8723a	3000010000	0.1.48	Import0	0	48	Visual Studio
C8	Comp ID	f9e9723bf8e002bc	101097086	28806.265.1	Utc1900_LTCG_CPP	28806	1	Visual Studio 2017 14.01+
D0	Comp ID	f9e9723bf91602bc	100ff7086	28806.255.1	Cvtres1400	28806	1	Visual Studio 2015 14.00
D8	Comp ID	f9e9723bf8eb02bc	101027086	28806.258.1	Linker1400	28806	1	Visual Studio 2015 14.00
E0	Rich ID	68636952		Rich				
E4	Checksum	f9e9723a		f9e9723a				

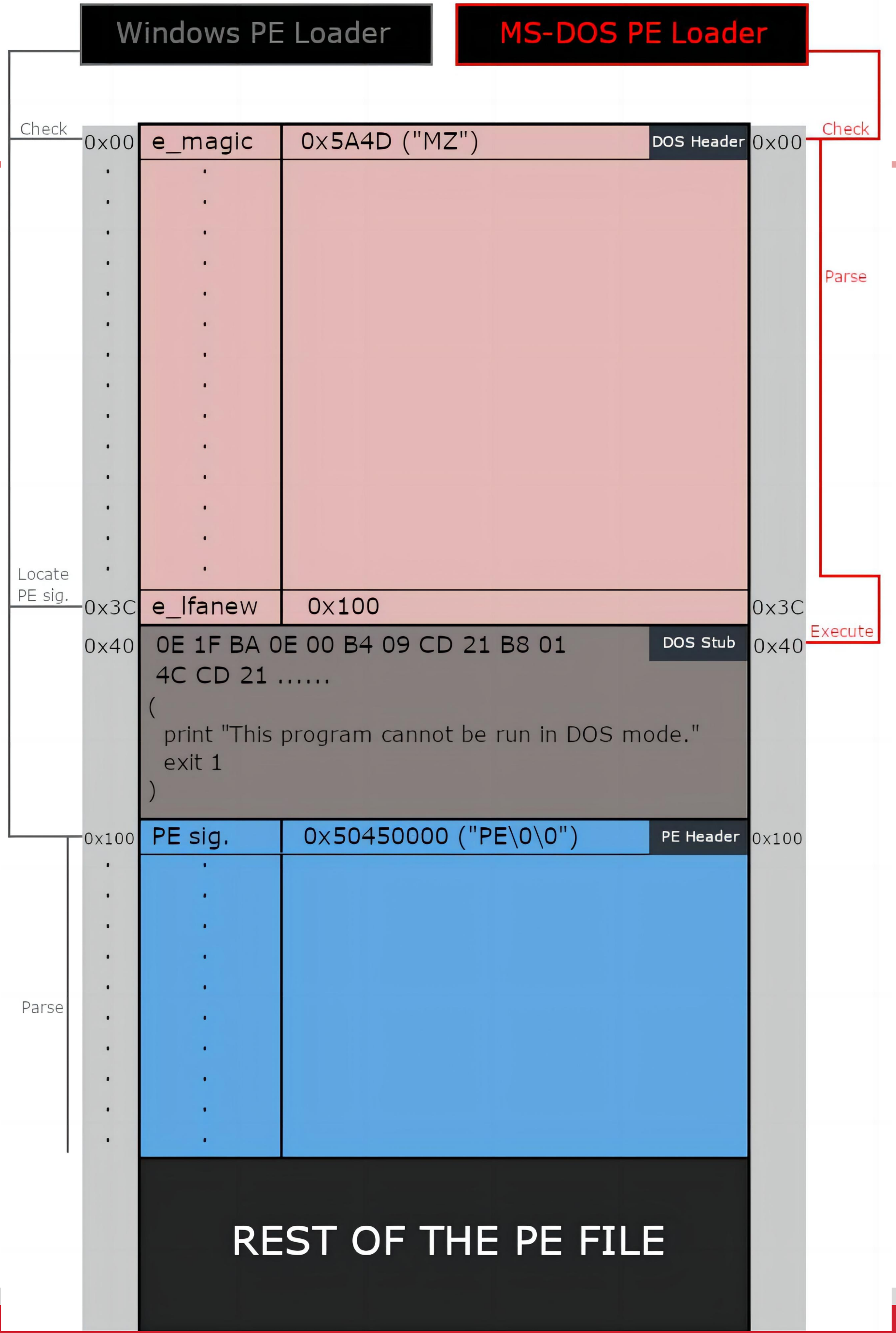
Rich Header



```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F     0 1 2 3 4 5 6 7 8 9 A B C D E F
40    0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68    . . º . . ´ . Í ! ¸ . L Í ! T h
50    69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F    i s . p r o g r a m . c a n n o
60    74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20    t . b e . r u n . i n . D O S .
70    6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00    m o d e . . . . $ . . . . . . .
80    7E 13 87 AA 3A 72 E9 F9 3A 72 E9 F9 3A 72 E9 F9    ~ . . ª : r é ù : r é ù : r é ù
90    33 0A 7A F9 30 72 E9 F9 F1 1D E8 F8 38 72 E9 F9    3 . z ù 0 r é ù ñ . è ø 8 r é ù
A0    F1 1D EC F8 2B 72 E9 F9 F1 1D ED F8 30 72 E9 F9    ñ . ì ø + r é ù ñ . í ø 0 r é ù
B0    F1 1D EA F8 39 72 E9 F9 61 1A E8 F8 3F 72 E9 F9    ñ . ê ø 9 r é ù a . è ø ? r é ù
C0    3A 72 E8 F9 0A 72 E9 F9 BC 02 E0 F8 3B 72 E9 F9    : r è ù . r é ù ¼ . à ø ; r é ù
D0    BC 02 16 F9 3B 72 E9 F9 BC 02 EB F8 3B 72 E9 F9    ¼ . . ù ; r é ù ¼ . ë ø ; r é ù
E0    52 69 63 68 3A 72 E9 F9 00 00 00 00 00 00 00 00    R i c h : r é ù . . . . . . . .
F0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    . . . . . . . . . . . . . . . .
100   50 45 00 00 64 86 06 00 03 DE 59 61 00 00 00 00    P E . . d . . . . Þ Y a . . . .
110   00 00 00 00 F0 00 22 00 0B 02 0E 1A 00 0E 00 00    . . . . ð . " . . . . . . . . .
```

Offset	Name	Value	Unmasked Value	Meaning	ProductId	BuildId	Count	VS version
80	DanS ID	aa87137e	536e6144	DanS				
84	Checksumed pad...	f9e9723a	0	0				
88	Checksumed pad...	f9e9723a	0	0				
8C	Checksumed pad...	f9e9723a	0	0				
90	Comp ID	f9e97230f97a0a33	a00937809	30729.147.10	Implib900	30729	10	Visual Studio 2008 09.00
98	Comp ID	f9e97238f8e81df1	201016fcb	28619.257.2	Implib1400	28619	2	Visual Studio 2015 14.00
A0	Comp ID	f9e9722bf8ec1df1	1101056fcb	28619.261.17	Utc1900_CPP	28619	17	Visual Studio 2015 14.00
A8	Comp ID	f9e97230f8ed1df1	a01046fcb	28619.260.10	Utc1900_C	28619	10	Visual Studio 2015 14.00
B0	Comp ID	f9e97239f8ea1df1	301036fcb	28619.259.3	Masm1400	28619	3	Visual Studio 2015 14.00
B8	Comp ID	f9e9723ff8e81a61	50101685b	26715.257.5	Implib1400	26715	5	Visual Studio 2015 14.00
C0	Comp ID	f9e9720af9e8723a	3000010000	0.1.48	Import0	0	48	Visual Studio
C8	Comp ID	f9e9723bf8e002bc	101097086	28806.265.1	Utc1900_LTCG_CPP	28806	1	Visual Studio 2017 14.01+
D0	Comp ID	f9e9723bf91602bc	100ff7086	28806.255.1	Cvtres1400	28806	1	Visual Studio 2015 14.00
D8	Comp ID	f9e9723bf8eb02bc	101027086	28806.258.1	Linker1400	28806	1	Visual Studio 2015 14.00
E0	Rich ID	68636952		Rich				
E4	Checksum	f9e9723a		f9e9723a				

PE Structure



Windows PE Loader

MS-DOS PE Loader

Check

Locate PE sig.

Parse

Check

Parse

Execute

Offset	Field	Value	Section	Offset
0x00	e_magic	0x5A4D ("MZ")	DOS Header	0x00
0x3C	e_lfanew	0x100		0x3C
0x40			DOS Stub	0x40
0x100	PE sig.	0x50450000 ("PE\0\0")	PE Header	0x100

```
 0E 1F BA 0E 00 B4 09 CD 21 B8 01
 4C CD 21 ......
(
  print "This program cannot be run in DOS mode."
  exit 1
)
```

REST OF THE PE FILE

Part Two

02

NT Header

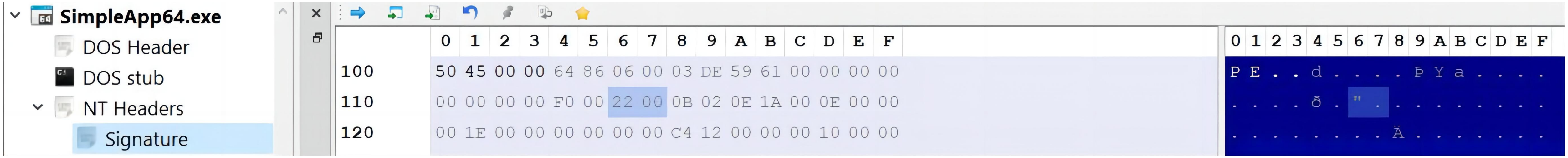
NT Header

```
typedef struct _IMAGE_NT_HEADERS64 {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER64 OptionalHeader;
} IMAGE_NT_HEADERS64, *PIMAGE_NT_HEADERS64;

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

Signature

- This is a 4-byte signature that identifies the file as a PE file.
- It always has a fixed value of 0x50450000 which translates to PE\0\0 in ASCII.

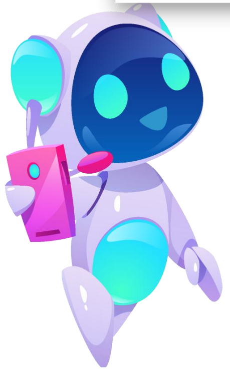


```
SimpleApp64.exe
    DOS Header
    DOS stub
    NT Headers
        Signature
```

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    0123456789ABCDEF
100  50 45 00 00 64 86 06 00 03 DE 59 61 00 00 00 00   PE..d....ÞYa....
110  00 00 00 00 F0 00 22 00 0B 02 0E 1A 00 0E 00 00   ....ð.".........
120  00 1E 00 00 00 00 00 00 C4 12 00 00 00 10 00 00   ........Ä.......
```

COFF (Common Object File Format) Header

- This header contains information about the file's format, including the size of the code and data sections, the number of sections, and the size of the optional header.



COFF Header

```
typedef struct _IMAGE_FILE_HEADER {
    WORD    Machine;
    WORD    NumberOfSections;
    DWORD   TimeDateStamp;
    DWORD   PointerToSymbolTable;
    DWORD   NumberOfSymbols;
    WORD    SizeOfOptionalHeader;
    WORD    Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

Fields

- **Machine:** This is a number that indicates the type of machine (CPU Architecture) the executable is targeting, this field can have a lot of values, but we’re only interested in two of them, 0x8864 for AMD64 and 0x14c for i386.

Constant	Value	Description
IMAGE_FILE_MACHINE_UNKNOWN	0x0	The content of this field is assumed to be applicable to any machine type
IMAGE_FILE_MACHINE_ALPHA	0x184	Alpha AXP, 32-bit address space
IMAGE_FILE_MACHINE_ALPHA64	0x284	Alpha 64, 64-bit address space
IMAGE_FILE_MACHINE_AM33	0x1d3	Matsushita AM33
IMAGE_FILE_MACHINE_AMD64	0x8664	x64
IMAGE_FILE_MACHINE_ARM	0x1c0	ARM little endian
IMAGE_FILE_MACHINE_ARM64	0xaa64	ARM64 little endian
IMAGE_FILE_MACHINE_ARMNT	0x1cd	ARM Thumb-2 little endian
IMAGE_FILE_MACHINE_AXP64	0x284	AXP 64 (Same as Alpha 64)
IMAGE_FILE_MACHINE_EBC	0xebc	EFI byte code
IMAGE_FILE_MACHINE_I386	0x14c	Intel 386 or later processors and compatible processors

Fields

- **Machine:** This is a number that indicates the type of machine (CPU Architecture) the executable is targeting, this field can have a lot of values, but we're only interested in two of them, 0x8864 for AMD64 and 0x14c for i386.

Constant	Value	Description
IMAGE_FILE_MACHINE_IA64	0x200	Intel Itanium processor family
IMAGE_FILE_MACHINE_LOONGARCH32	0x6232	LoongArch 32-bit processor family
IMAGE_FILE_MACHINE_LOONGARCH64	0x6264	LoongArch 64-bit processor family
IMAGE_FILE_MACHINE_M32R	0x9041	Mitsubishi M32R little endian
IMAGE_FILE_MACHINE_MIPS16	0x266	MIPS16
IMAGE_FILE_MACHINE_MIPSFPU	0x366	MIPS with FPU
IMAGE_FILE_MACHINE_MIPSFPU16	0x466	MIPS 16 with FPU
IMAGE_FILE_MACHINE_POWERPC	0x1f0	Power PC little endian
IMAGE_FILE_MACHINE_POWERPCFP	0x1f1	Power PC with floating point support
IMAGE_FILE_MACHINE_R4000	0x166	MIPS little endian
IMAGE_FILE_MACHINE_RISCV32	0x5032	RISC-V 32-bit address space
IMAGE_FILE_MACHINE_RISCV64	0x5064	RISC-V 64-bit address space
IMAGE_FILE_MACHINE_RISCV128	0x5128	RISC-V 128-bit address space

Fields

- **Machine:** This is a number that indicates the type of machine (CPU Architecture) the executable is targeting, this field can have a lot of values, but we’re only interested in two of them, 0x8864 for AMD64 and 0x14c for i386.

Constant	Value	Description
IMAGE_FILE_MACHINE_SH3	0x1a2	Hitachi SH3
IMAGE_FILE_MACHINE_SH3DSP	0x1a3	Hitachi SH3 DSP
IMAGE_FILE_MACHINE_SH4	0x1a6	Hitachi SH4
IMAGE_FILE_MACHINE_SH5	0x1a8	Hitachi SH5
IMAGE_FILE_MACHINE_THUMB	0x1c2	Thumb
IMAGE_FILE_MACHINE_WCEMIPSV2	0x169	MIPS little-endian WCE v2

Fields

- NumberOfSections: This field holds the number of sections (or the number of section headers. the size of the section table.)
- TimeDateStamp: A unix timestamp that indicates when the file was created.
- PointerToSymbolTable and NumberOfSymbols: These two fields hold the file offset to the COFF symbol table and the number of entries in that symbol table, however they get set to 0 which means that no COFF symbol table is present, this is done because the COFF debugging information is deprecated.

Fields

- SizeOfOptionalHeader: The size of the Optional Header.
- Characteristics: A flag that indicates the attributes of the file, these attributes can be things like the file being executable, the file being a system file and not a user program, and a lot of other things.

Characteristics

Flag	Value	Description
IMAGE_FILE_RELOCS_STRIPPED	0x0001	Image only, Windows CE, and Microsoft Windows NT and later. This indicates that the file does not contain base relocations and must therefore be loaded at its preferred base address. If the base address is not available, the loader reports an error. The default behavior of the linker is to strip base relocations from executable (EXE) files.
IMAGE_FILE_EXECUTABLE_IMAGE	0x0002	Image only.This indicates that the image file is valid and can be run. If this flag is not set, it indicates a linker error.
IMAGE_FILE_LINE_NUMS_STRIPPED	0x0004	COFF line numbers have been removed.This flag is deprecated and should be zero.
IMAGE_FILE_LOCAL_SYMS_STRIPPED	0x0008	COFF symbol table entries for local symbols have been removed. This flag is deprecated and should be zero.

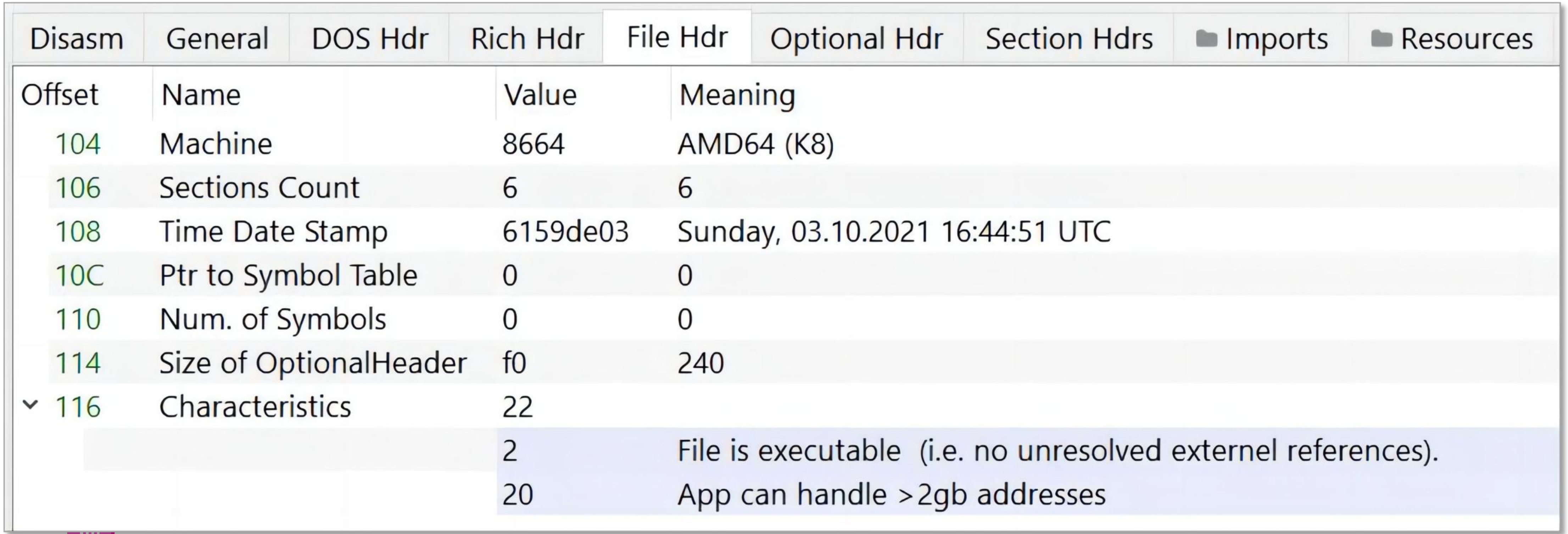
Characteristics

Constant	Value	Description
IMAGE_FILE_AGGRESSIVE_WS_TRIM	0x0010	Obsolete.Aggressively trim working set.This flag is deprecated for Windows 2000 and later and must be zero.
IMAGE_FILE_LARGE_ADDRESS_AWARE	0x0020	Application can handle>2-GB addresses.
	0x0040	This flag is reserved for future use.
IMAGE_FILE_BYTES_REVERSED_LO	0x0080	Little endian: the least significant bit (LSB)precedes the most significant bit (MSB) in memory. This flag is deprecated and should be zero.
IMAGE_FILE_32BIT_MACHINE	0x0100	Machine is based on a 32-bit-word architecture.
IMAGE_FILE_DEBUG_STRIPPED	0x0200	Debugging information is removed from the image file.
IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP	0x0400	If the image is on removable media,fully load it and copy it to the swap file.
IMAGE_FILE_NET_RUN_FROM_SWAP	0x0800	If the image is on network media,fully load it and copy it to the swap file.

Characteristics

Constant	Value	Description
IMAGE_FILE_SYSTEM	0x1000	The image file is a system file,not a user program.
IMAGE_FILE_DLL	0x2000	The image file is a dynamic-link library (DLL). Such files are considered executable files for almost all purposes, although they cannot be directly run.
IMAGE_FILE_UP_SYSTEM_ONLY	0x4000	The file should be run only on a uniprocessor machine.
IMAGE_FILE_BYTES_REVERSED_HI	0x8000	Big endian:the MSB precedes the LSB in memory. This flag is deprecated and should be zero.

COFF Header



Offset	Name	Value	Meaning
104	Machine	8664	AMD64 (K8)
106	Sections Count	6	6
108	Time Date Stamp	6159de03	Sunday, 03.10.2021 16:44:51 UTC
10C	Ptr to Symbol Table	0	0
110	Num. of Symbols	0	0
114	Size of OptionalHeader	f0	240
116	Characteristics	22	
		2	File is executable (i.e. no unresolved externel references).
		20	App can handle >2gb addresses

MONTANA STATE UNIVERSITY

THE END

Fangtian Zhong

CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

2025.09.04