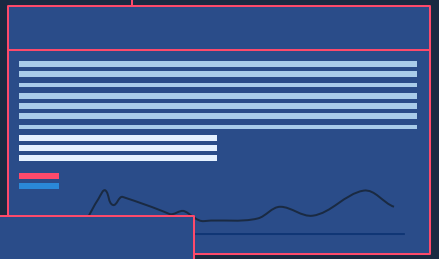
PE & Windows Basics



Malware Deconstructed:
Episode 1

Table of Content

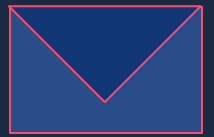
01 Preface

02 Preparing our environment

03 PE structure

04 Common Windows Security Concepts

05 What now?

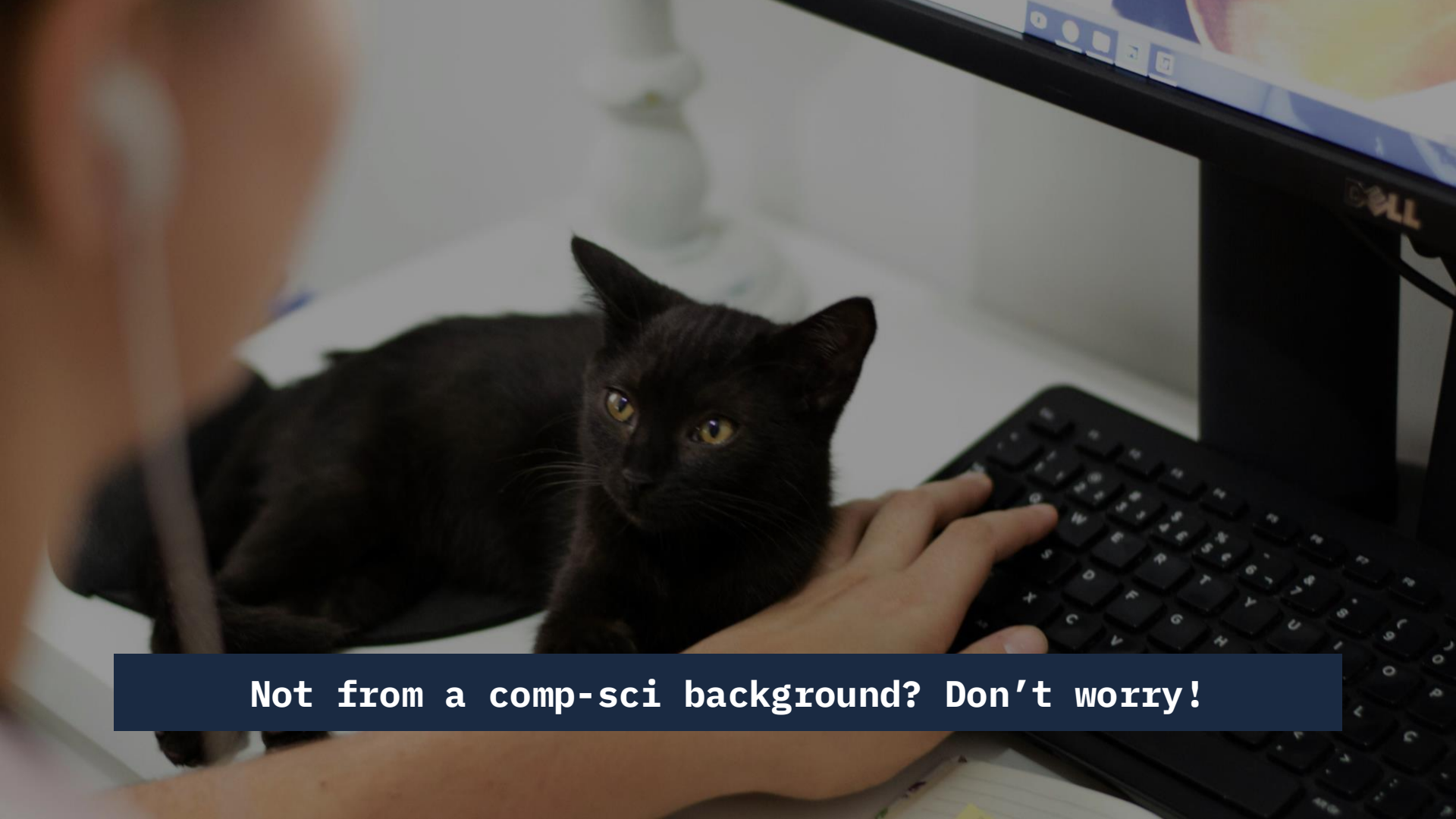


01 Preface

whoami

Azaka / Still

- they/them
- VTuber
- Taiwanese
- Full-time threat intelligence researcher for 3+ years


Not from a comp-sci background? Don’t worry!

02

Preparing the environment

Prepare a Safe Environment

Virtual Machine / Sandbox

- Free
  - Hyper-V
    - Only recommended if child partition is Windows 10 or above due to Generation 2 compatibility
  - Qemu
    - Relatively difficult to get into but highly flexible
  - VirtualBox
- Paid
  - VMware Workstation/ESXi
    - Perpetual licenses are being transitioned into subscription-based models

Prepare a Safe Environment

OS for Guests/Child Partitions

- Something not too old or too new
  - Preferably something at least 2 years old
  - Older Windows 10 builds
    - e.g. 1809, 1903, etc.
  - Windows 7
  - Older builds of Debian/CentOS
- Software
  - Office 2016 or below without updates
  - Analysis tools

Prepare a Safe Environment

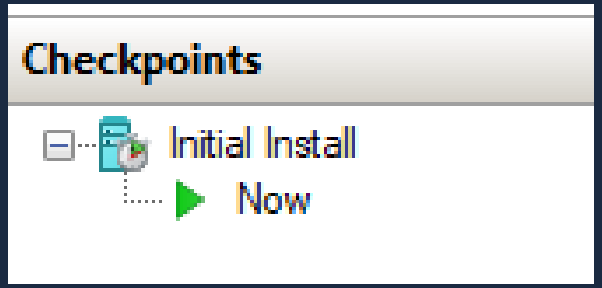
No network if you're not confident!

- DO NOT connect the child OS to network of any kind unless you are prepared.
  - The networking should have sufficient quarantining.
  - The child should NOT be able to…
    - Get your real WAN IP address
    - Access any other devices on the intranet
- Unless you are 100% confident that you are sufficiently tunneled and quarantined, DO NOT let it go online.

Prepare a Safe Environment

Make a Checkpoint

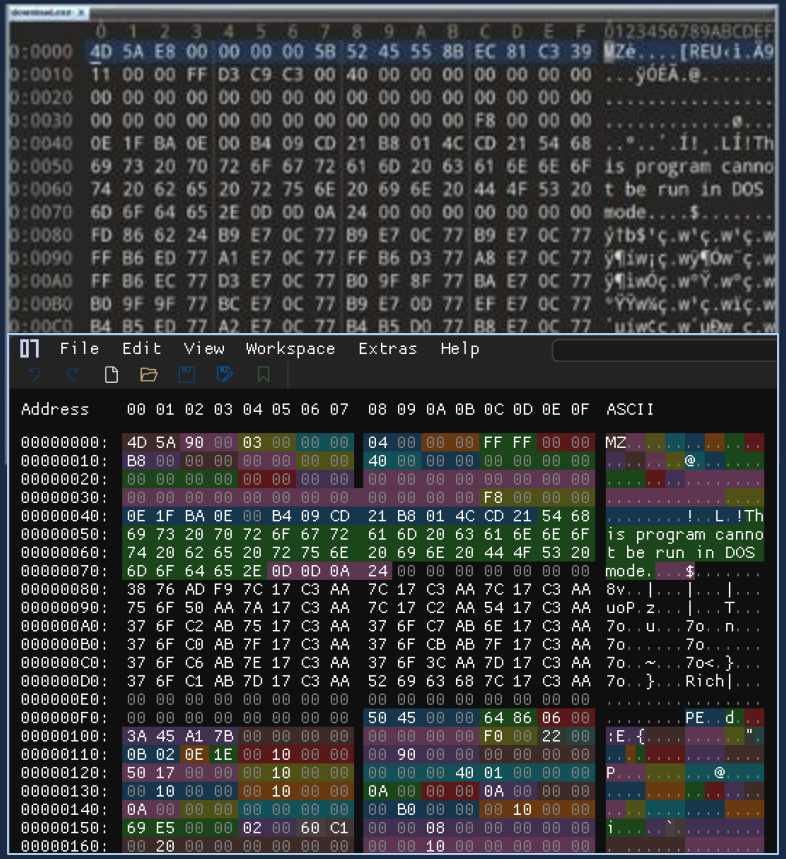
- AFTER you've installed all the necessary software, shutdown and make a checkpoint!
- A checkpoint or restore point allows you to clear your VM to a known clean state.



Other Software

Hex Editor

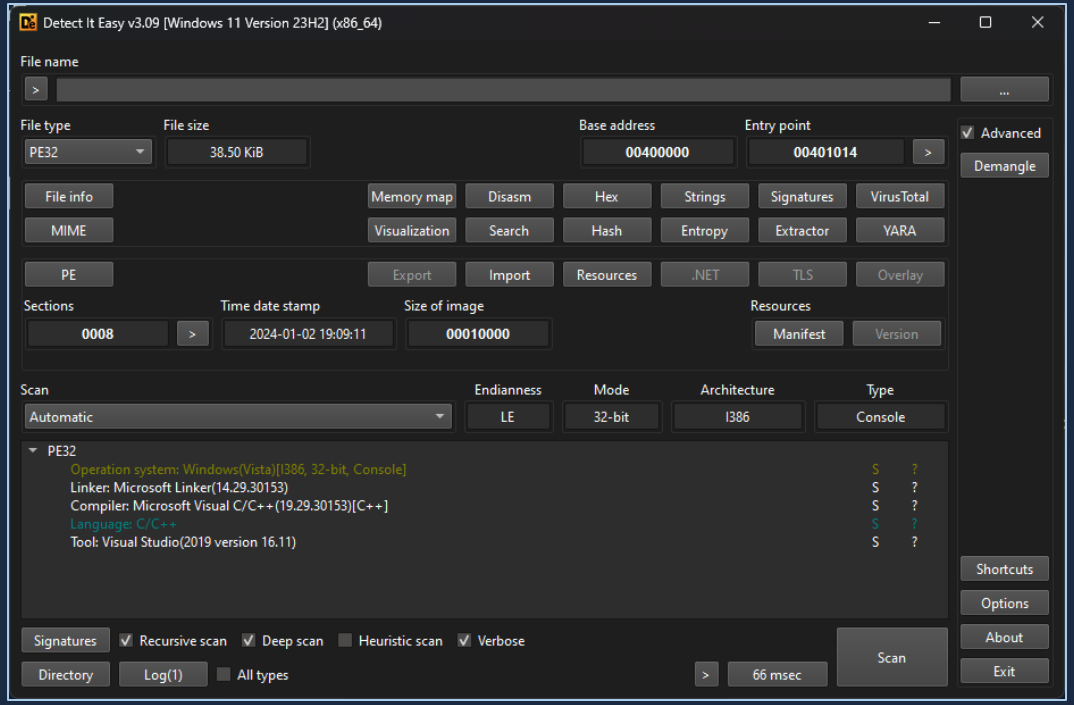
- Personal recommendations
  - 010 Editor (Proprietary)
  - ImHex (FOSS)



Other Software

Detect it Easy

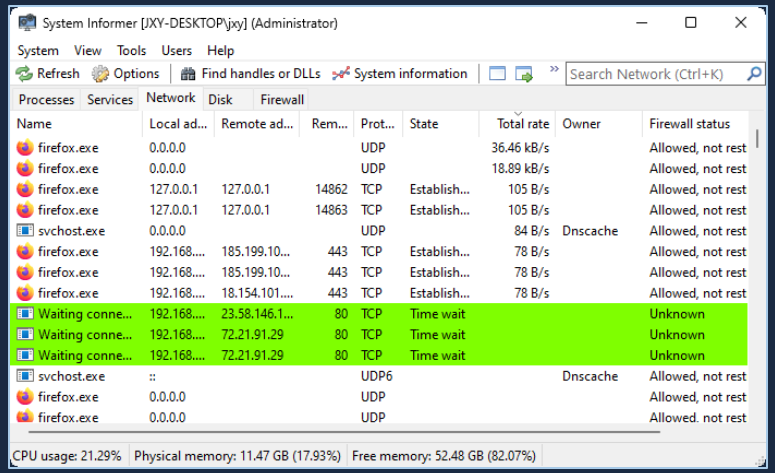
- Detection swiss-knife



Other Software

System Informer

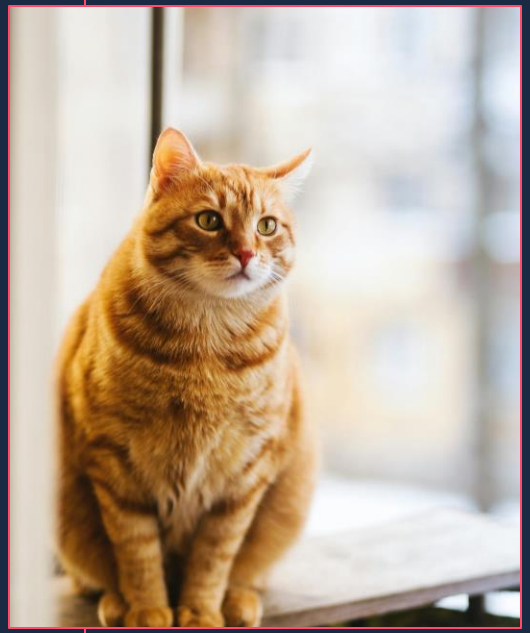
- Previously known as Process Hacker
- Task Manager on steroids



Materials to today’s course, including this slide

will be on the GitHub repository

(Still34/malware-deconstructed-lab)



03

Understanding PE Structure

What's This PE Stuff Anyways?

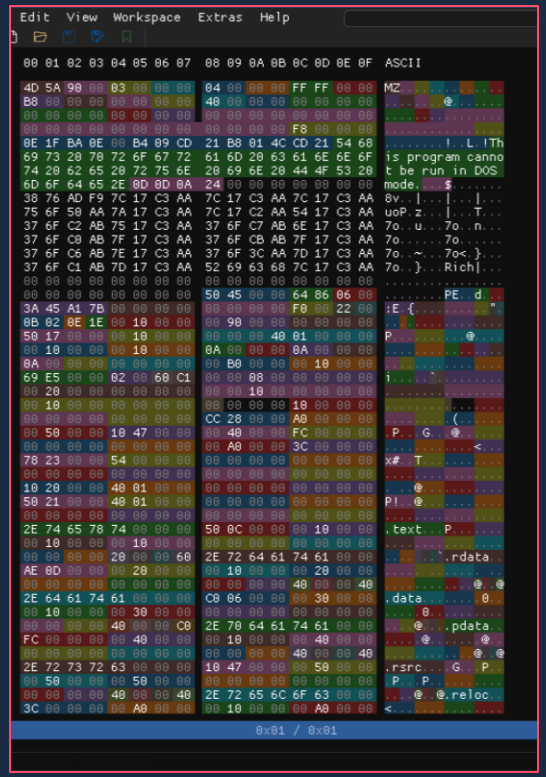
- Enables standardized software loading and execution
- Standard executable format for Windows
  - executable files (EXE)
  - dynamic link libraries (DLL)
  - drivers (SYS)
- Tells Windows
  - what the application is
  - how it should behave
  - where the code is
  - what resources are used, etc.

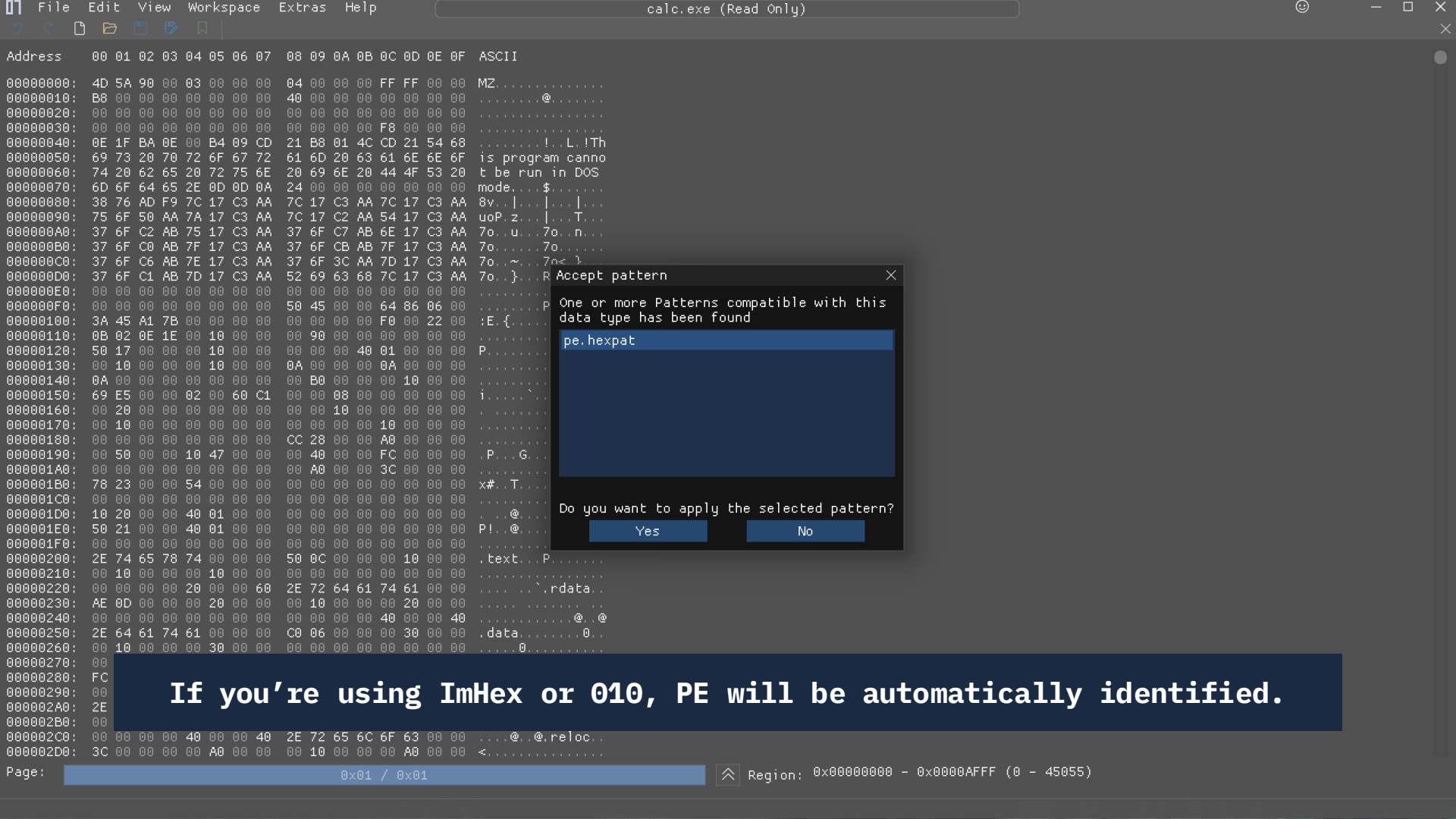
Workshop #1

Open C:\Windows\System32\calc.exe

(or any .EXE files under the directory)

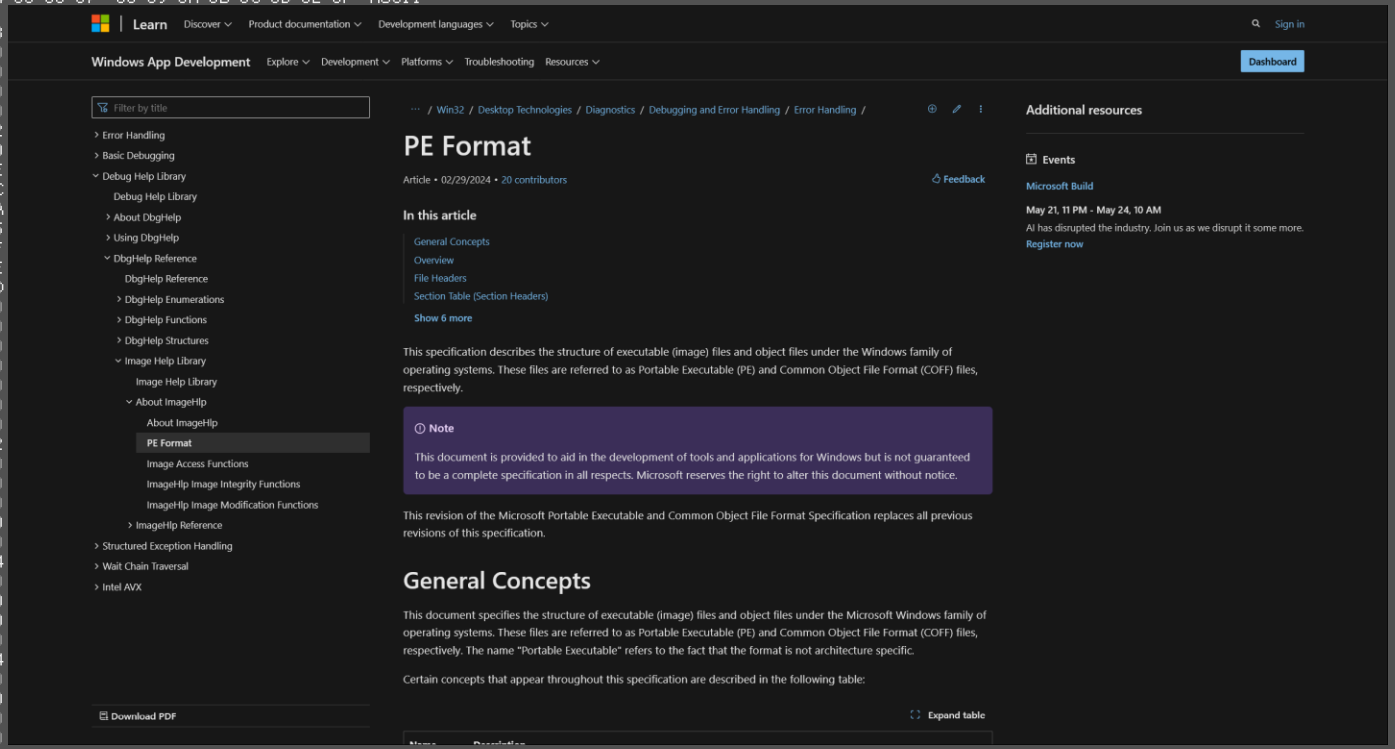
on your machine with a hex editor of your choice




File Edit View Workspace Extras Help
calc.exe (Read Only)
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
00000000: 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ..............
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 F8 00 00 00 ....
00000040: 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ....!..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: 38 76 AD F9 7C 17 C3 AA 7C 17 C3 AA 7C 17 C3 AA 8v..|...|...|...
00000090: 75 6F 50 AA 7A 17 C3 AA 7C 17 C2 AA 54 17 C3 AA uoP.z...|...T...
000000A0: 37 6F C2 AB 75 17 C3 AA 37 6F C7 AB 6E 17 C3 AA 7o..u...7o..n...
000000B0: 37 6F C0 AB 7F 17 C3 AA 37 6F CB AB 7F 17 C3 AA 7o......7o......
000000C0: 37 6F C6 AB 7E 17 C3 AA 37 6F 3C AA 7D 17 C3 AA 7o..~...7o<.}...
000000D0: 37 6F C1 AB 7D 17 C3 AA 52 69 63 68 7C 17 C3 AA 7o..}...Rich|...
000000E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000F0: 00 00 00 00 00 00 00 00 50 45 00 00 64 86 06 00 ....PE..d...
00000100: 3A 45 A1 7B 00 00 00 00 00 00 00 00 F0 00 22 00 :E.{..........".
00000110: 0B 02 0E 1E 00 10 00 00 00 90 00 00 00 00 00 00 ....
00000120: 50 17 00 00 00 10 00 00 00 00 00 40 01 00 00 00 P..........@....
00000130: 00 10 00 00 00 10 00 00 0A 00 00 00 0A 00 00 00 ....
00000140: 0A 00 00 00 00 00 00 00 00 B0 00 00 00 10 00 00 ....
00000150: 69 E5 00 00 02 00 60 C1 00 00 08 00 00 00 00 00 i.....`.........
00000160: 00 20 00 00 00 00 00 00 00 00 10 00 00 00 00 00 . ...
00000170: 00 10 00 00 00 00 00 00 00 00 00 00 10 00 00 00 ....
00000180: 00 00 00 00 00 00 00 00 CC 28 00 00 A0 00 00 00 ....(......
00000190: 00 50 00 00 10 47 00 00 00 40 00 00 FC 00 00 00 .P...G...@......
000001A0: 00 00 00 00 00 00 00 00 00 A0 00 00 3C 00 00 00 ....<...
000001B0: 78 23 00 00 54 00 00 00 00 00 00 00 00 00 00 00 x#..T...........
000001C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000001D0: 10 20 00 00 40 01 00 00 00 00 00 00 00 00 00 00 . ..@...........
000001E0: 50 21 00 00 40 01 00 00 00 00 00 00 00 00 00 00 P!..@...........
000001F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000200: 2E 74 65 78 74 00 00 00 50 0C 00 00 00 10 00 00 .text...P.......
00000210: 00 10 00 00 00 10 00 00 00 00 00 00 00 00 00 00 ....
00000220: 00 00 00 00 20 00 00 60 2E 72 64 61 74 61 00 00 .... ..`.rdata..
00000230: AE 0D 00 00 00 20 00 00 00 10 00 00 00 20 00 00 ....
00000240: 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 40 ....@..@
00000250: 2E 64 61 74 61 00 00 00 C0 06 00 00 00 30 00 00 .data........0..
00000260: 00 10 00 00 00 30 00 00 00 00 00 00 00 00 00 00 .....0..........
00000270: 00
00000280: FC
00000290: 00
000002A0: 2E
000002B0: 00
000002C0: 00 00 00 00 40 00 00 40 2E 72 65 6C 6F 63 00 00 ....@..@.reloc..
000002D0: 3C 00 00 00 00 A0 00 00 00 10 00 00 00 A0 00 00 <...............
Accept pattern
One or more Patterns compatible with this data type has been found
pe.hexpat
Do you want to apply the selected pattern?
Yes
No
If you’re using ImHex or 010, PE will be automatically identified.
Page: 0x01 / 0x01
Region: 0x00000000 - 0x0000AFFF (0 - 45055)

File Edit View Workspace Extras Help

calc.exe (Read Only)

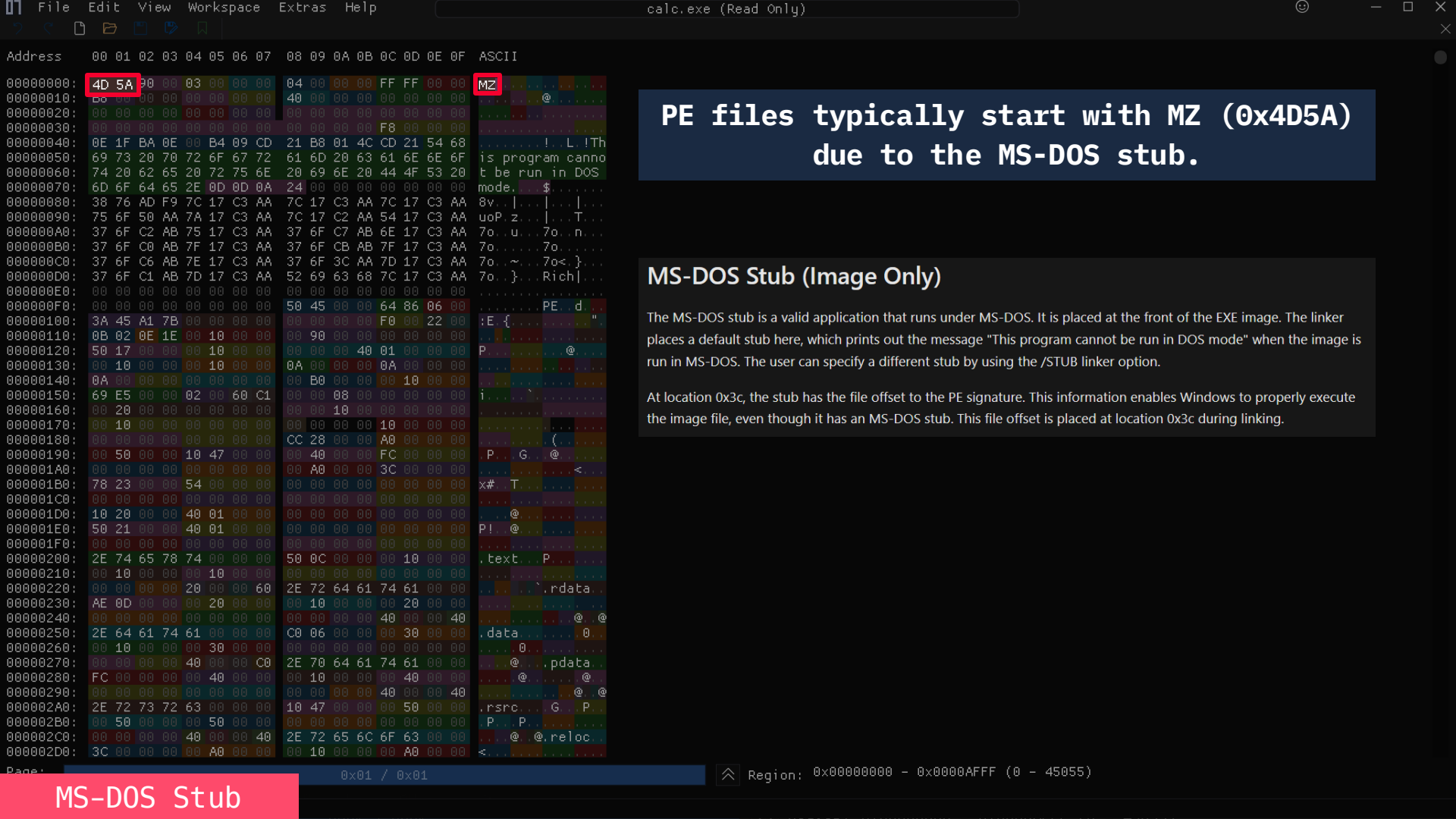



```
Address    00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ASCII
00000000:  4D 5A 90 00 03
00000010:  B8 00 00 00 00
00000020:  00 00 00 00 00
00000030:  00 00 00 00 00
00000040:  0E 1F BA 0E 00
00000050:  69 73 20 70 72
00000060:  74 20 62 65 20
00000070:  6D 6F 64 65 2E
00000080:  38 76 AD F9 7C
00000090:  75 6F 50 AA 7A
000000A0:  37 6F C2 AB 75
000000B0:  37 6F C0 AB 7F
000000C0:  37 6F C6 AB 7E
000000D0:  37 6F C1 AB 7D
000000E0:  00 00 00 00 00
000000F0:  00 00 00 00 00
00000100:  3A 45 A1 7B 00
00000110:  0B 02 0E 1E 00
00000120:  50 17 00 00 00
00000130:  00 10 00 00 00
00000140:  0A 00 00 00 00
00000150:  69 E5 00 00 02
00000160:  00 20 00 00 00
00000170:  00 10 00 00 00
00000180:  00 00 00 00 00
00000190:  00 50 00 00 10
000001A0:  00 00 00 00 00
000001B0:  78 23 00 00 54
000001C0:  00 00 00 00 00
000001D0:  10 20 00 00 40
000001E0:  50 21 00 00 40
000001F0:  00 00 00 00 00
00000200:  2E 74 65 78 74
00000210:  00 10 00 00 00
00000220:  00 00 00 00 20
00000230:  AE 0D 00 00 00
00000240:  00 00 00 00 00
00000250:  2E 64 61 74 61 00 00 00  C0 06 00 00 00 30 00 00  .data.......0..
00000260:  00 10 00 00 00 30 00 00  00 00 00 00 00 00 00 00  .....0..........
00000270:  00
00000280:  FC
00000290:  00
000002A0:  2E
000002B0:  00
000002C0:  00 00 00 00 40 00 00 40  2E 72 65 6C 6F 63 00 00  ....@..@.reloc..
000002D0:  3C 00 00 00 00 A0 00 00  00 10 00 00 00 A0 00 00  <...............
```

Let’s try to break this down by hand instead.

Page: 0x01 / 0x01

Region: 0x00000000 - 0x0000AFFF (0 - 45055)



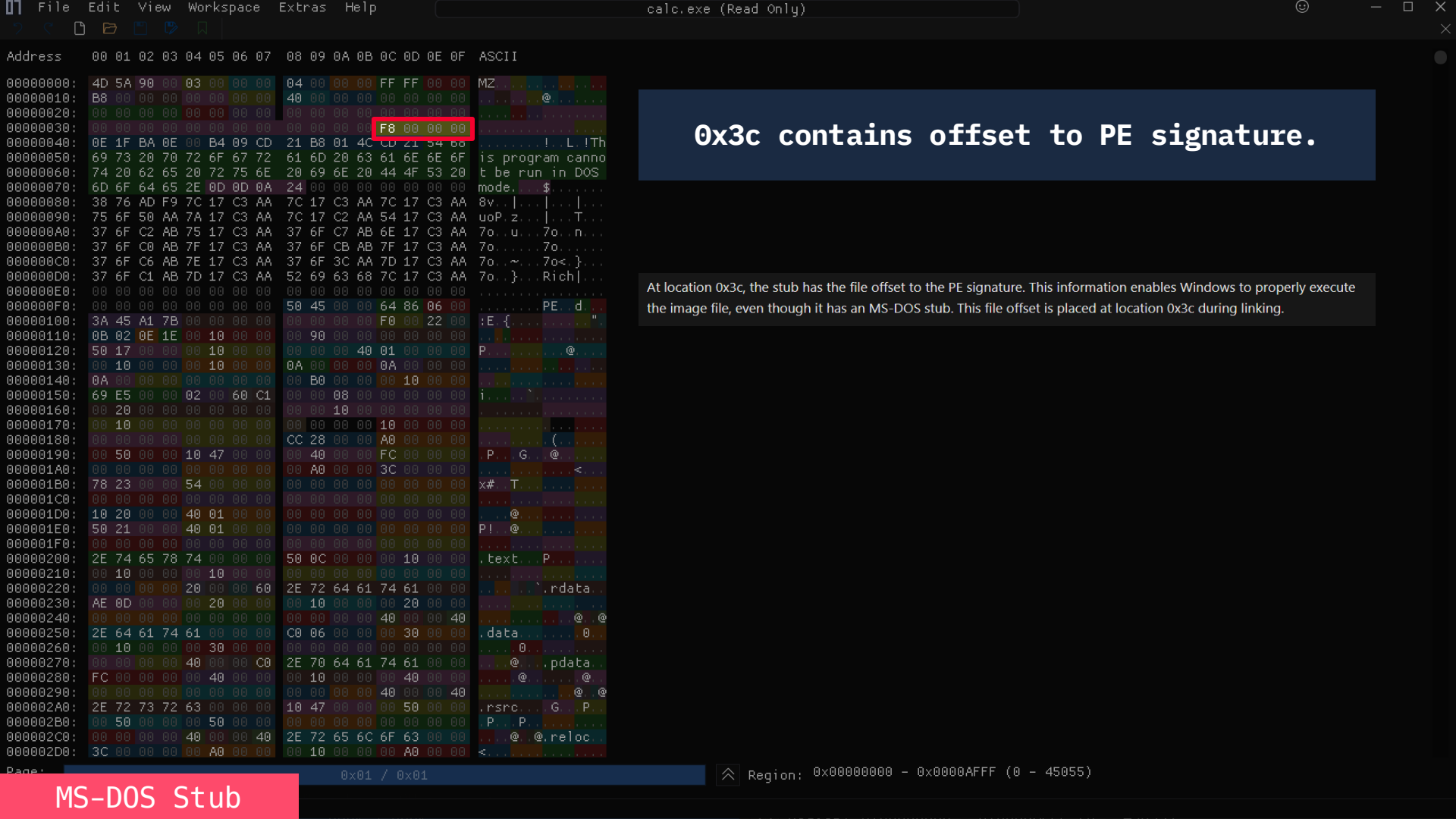
PE files typically start with MZ (0x4D5A) due to the MS-DOS stub.

MS-DOS Stub (Image Only)

The MS-DOS stub is a valid application that runs under MS-DOS. It is placed at the front of the EXE image. The linker places a default stub here, which prints out the message "This program cannot be run in DOS mode" when the image is run in MS-DOS. The user can specify a different stub by using the /STUB linker option.

At location 0x3c, the stub has the file offset to the PE signature. This information enables Windows to properly execute the image file, even though it has an MS-DOS stub. This file offset is placed at location 0x3c during linking.

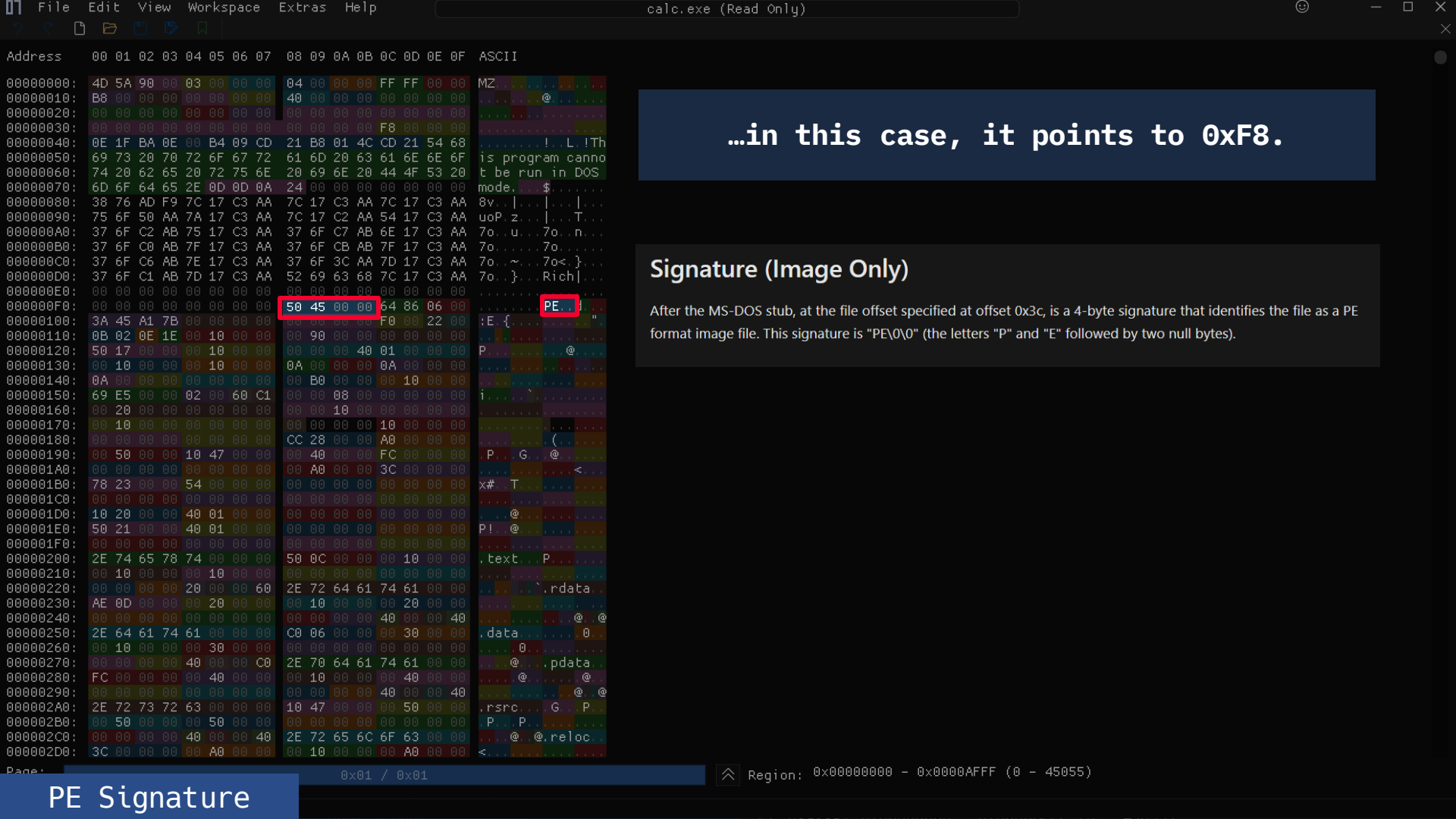
MS-DOS Stub

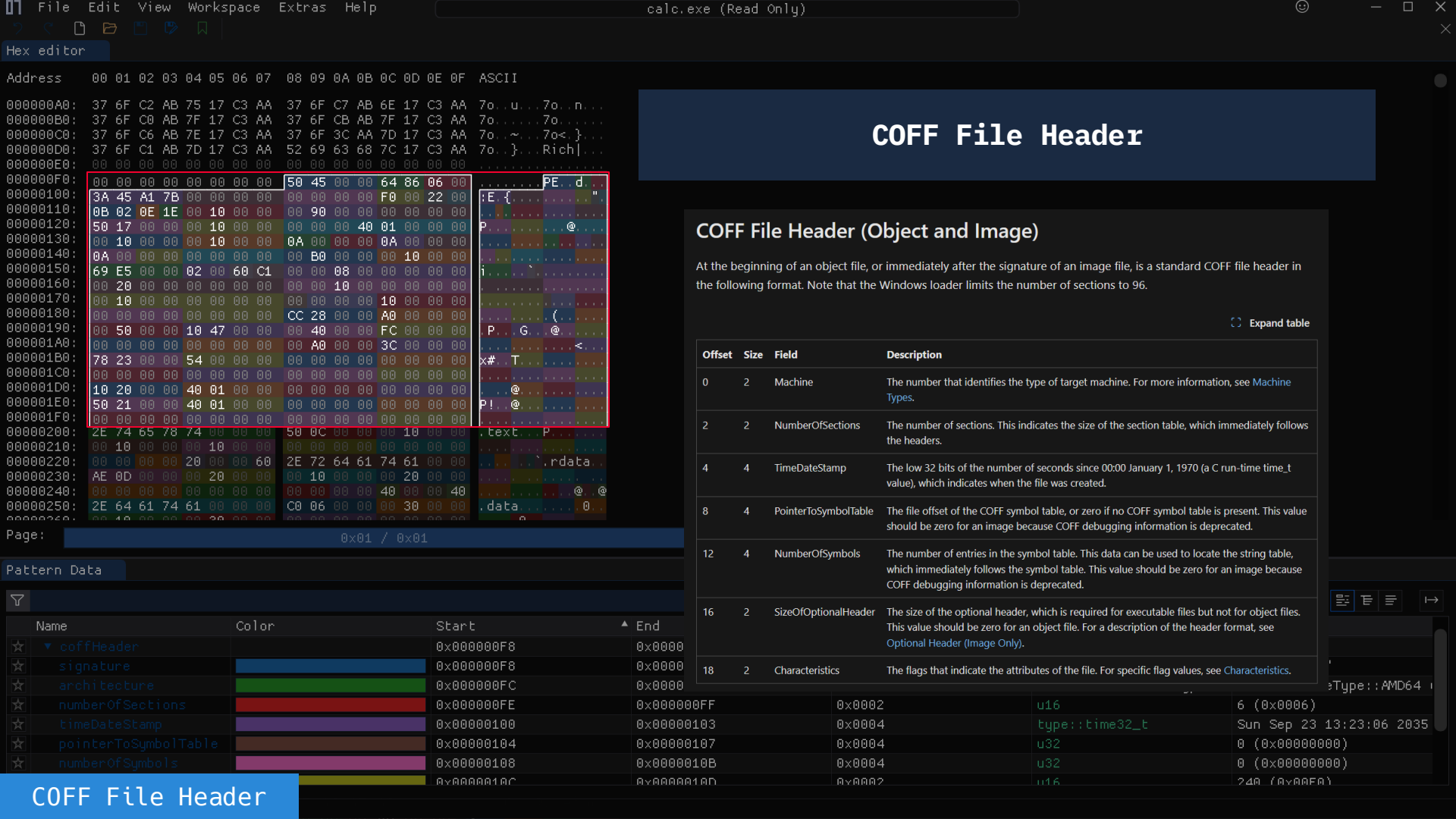


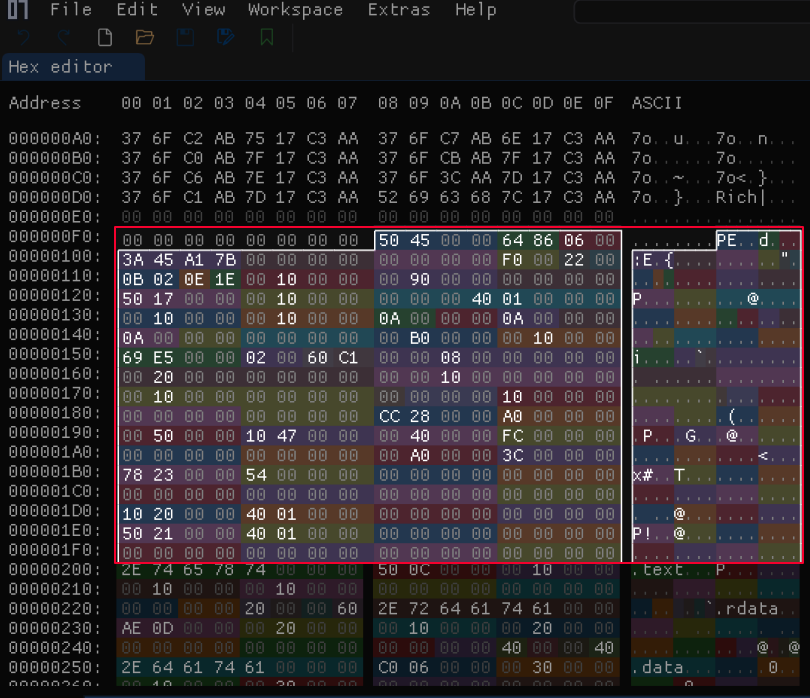
0x3c contains offset to PE signature.

At location 0x3c, the stub has the file offset to the PE signature. This information enables Windows to properly execute the image file, even though it has an MS-DOS stub. This file offset is placed at location 0x3c during linking.

MS-DOS Stub


File Edit View Workspace Extras Help
calc.exe (Read Only)
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
00000000: 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ
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 F8 00 00 00
00000040: 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ! 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: 38 76 AD F9 7C 17 C3 AA 7C 17 C3 AA 7C 17 C3 AA 8v |...|...|...
00000090: 75 6F 50 AA 7A 17 C3 AA 7C 17 C2 AA 54 17 C3 AA uoP.z...|...T...
000000A0: 37 6F C2 AB 75 17 C3 AA 37 6F C7 AB 6E 17 C3 AA 7o..u...7o..n...
000000B0: 37 6F C0 AB 7F 17 C3 AA 37 6F CB AB 7F 17 C3 AA 7o.....7o.....
000000C0: 37 6F C6 AB 7E 17 C3 AA 37 6F 3C AA 7D 17 C3 AA 7o..~...7o<.}...
000000D0: 37 6F C1 AB 7D 17 C3 AA 52 69 63 68 7C 17 C3 AA 7o..}...Rich|...
000000E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000F0: 00 00 00 00 00 00 00 00 50 45 00 00 64 86 06 00 PE..d...
00000100: 3A 45 A1 7B 00 00 00 00 F0 00 22 00 :E {.....".
00000110: 0B 02 0E 1E 00 10 00 00 00 90 00 00 00 00 00 00
00000120: 50 17 00 00 00 10 00 00 00 00 00 40 01 00 00 00 P.....@...
00000130: 00 10 00 00 00 10 00 00 0A 00 00 00 0A 00 00 00
00000140: 0A 00 00 00 00 00 00 00 00 B0 00 00 00 10 00 00
00000150: 69 E5 00 00 02 00 60 C1 00 00 08 00 00 00 00 00 i.....`.....
00000160: 00 20 00 00 00 00 00 00 00 00 10 00 00 00 00 00
00000170: 00 10 00 00 00 00 00 00 00 00 00 00 10 00 00 00
00000180: 00 00 00 00 00 00 00 00 CC 28 00 00 A0 00 00 00 (
00000190: 00 50 00 00 10 47 00 00 00 40 00 00 FC 00 00 00 .P...G...@....
000001A0: 00 00 00 00 00 00 00 00 00 A0 00 00 3C 00 00 00 <...
000001B0: 78 23 00 00 54 00 00 00 00 00 00 00 00 00 00 00 x#..T.....
000001C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001D0: 10 20 00 00 40 01 00 00 00 00 00 00 00 00 00 00 ...@.....
000001E0: 50 21 00 00 40 01 00 00 00 00 00 00 00 00 00 00 P!..@.....
000001F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000200: 2E 74 65 78 74 00 00 00 50 0C 00 00 00 10 00 00 .text...P.....
00000210: 00 10 00 00 00 10 00 00 00 00 00 00 00 00 00 00
00000220: 00 00 00 00 20 00 00 60 2E 72 64 61 74 61 00 00 ....`.rdata..
00000230: AE 0D 00 00 00 20 00 00 00 10 00 00 00 20 00 00
00000240: 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 40 @..@
00000250: 2E 64 61 74 61 00 00 00 C0 06 00 00 00 30 00 00 .data.....0..
00000260: 00 10 00 00 00 30 00 00 00 00 00 00 00 00 00 00 .....0.....
00000270: 00 00 00 00 40 00 00 C0 2E 70 64 61 74 61 00 00 ....@....pdata..
00000280: FC 00 00 00 00 40 00 00 00 10 00 00 00 40 00 00 .....@.....@..
00000290: 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 40 @..@
000002A0: 2E 72 73 72 63 00 00 00 10 47 00 00 00 50 00 00 .rsrc....G...P..
000002B0: 00 50 00 00 00 50 00 00 00 00 00 00 00 00 00 00 .P...P.....
000002C0: 00 00 00 00 40 00 00 40 2E 72 65 6C 6F 63 00 00 ....@..@.reloc..
000002D0: 3C 00 00 00 00 A0 00 00 00 10 00 00 00 A0 00 00 <.....
Page: 0x01 / 0x01
Region: 0x00000000 - 0x0000AFFF (0 - 45055)
…in this case, it points to 0xF8.
Signature (Image Only)
After the MS-DOS stub, at the file offset specified at offset 0x3c, is a 4-byte signature that identifies the file as a PE format image file. This signature is "PE\0\0" (the letters "P" and "E" followed by two null bytes).
PE Signature


File Edit View Workspace Extras Help
calc.exe (Read Only)
Hex editor
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
Page: 0x01 / 0x01
Pattern Data
Name Color Start End
coffHeader 0x000000F8 0x0000
signature 0x000000F8 0x0000
architecture 0x000000FC 0x0000
numberOfSections 0x000000FE 0x000000FF 0x0002 u16 6 (0x0006)
timeDateStamp 0x00000100 0x00000103 0x0004 type::time32_t Sun Sep 23 13:23:06 2035
pointerToSymbolTable 0x00000104 0x00000107 0x0004 u32 0 (0x00000000)
numberOfSymbols 0x00000108 0x0000010B 0x0004 u32 0 (0x00000000)
COFF File Header
COFF File Header
COFF File Header (Object and Image)
At the beginning of an object file, or immediately after the signature of an image file, is a standard COFF file header in the following format. Note that the Windows loader limits the number of sections to 96.
Expand table
Offset Size Field Description
0 2 Machine The number that identifies the type of target machine. For more information, see Machine Types.
2 2 NumberOfSections The number of sections. This indicates the size of the section table, which immediately follows the headers.
4 4 TimeDateStamp The low 32 bits of the number of seconds since 00:00 January 1, 1970 (a C run-time time_t value), which indicates when the file was created.
8 4 PointerToSymbolTable The file offset of the COFF symbol table, or zero if no COFF symbol table is present. This value should be zero for an image because COFF debugging information is deprecated.
12 4 NumberOfSymbols The number of entries in the symbol table. This data can be used to locate the string table, which immediately follows the symbol table. This value should be zero for an image because COFF debugging information is deprecated.
16 2 SizeOfOptionalHeader The size of the optional header, which is required for executable files but not for object files. This value should be zero for an object file. For a description of the header format, see Optional Header (Image Only).
18 2 Characteristics The flags that indicate the attributes of the file. For specific flag values, see Characteristics.



We'll pick a few important ones to talk about…

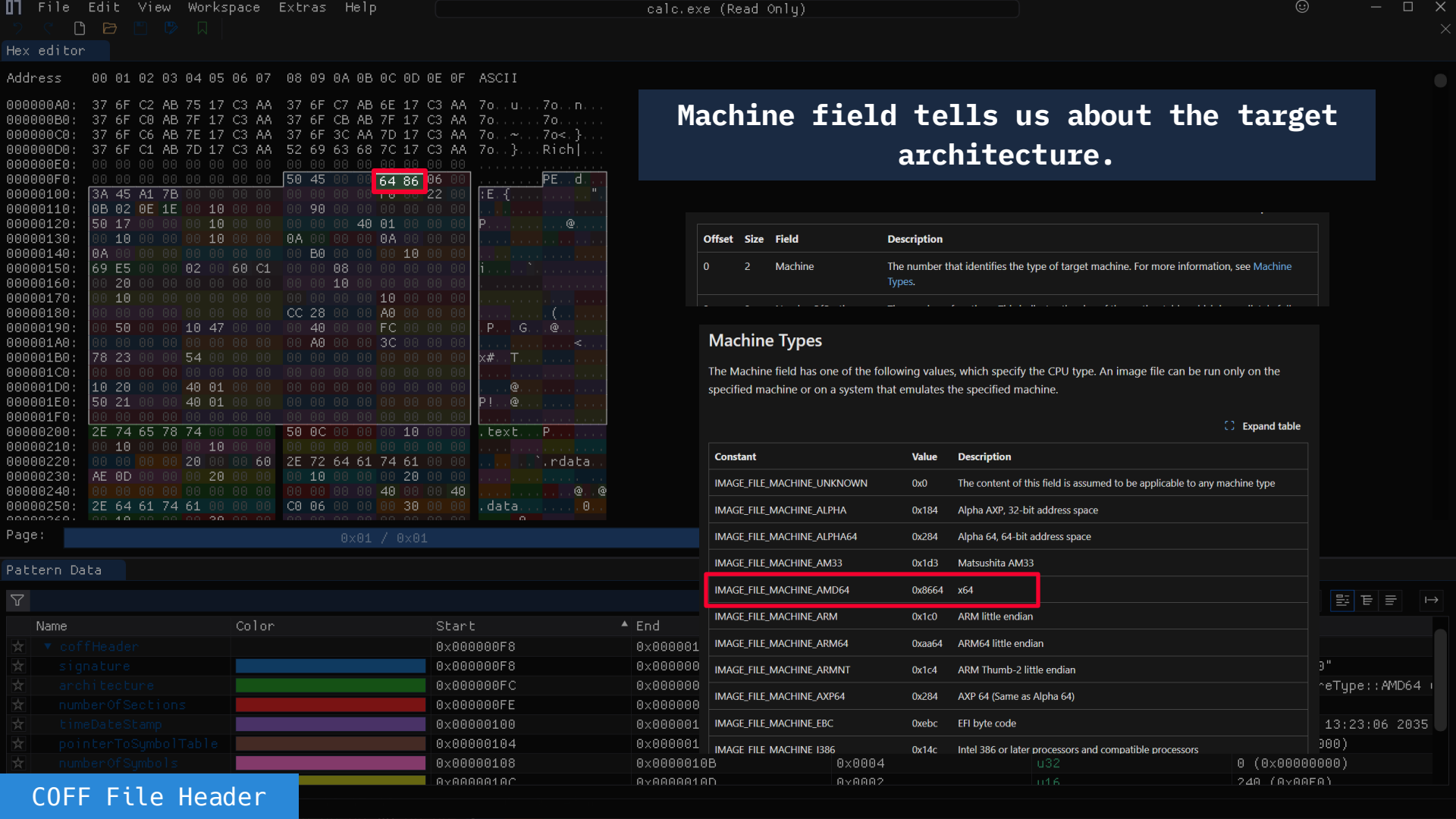
COFF File Header (Object and Image)

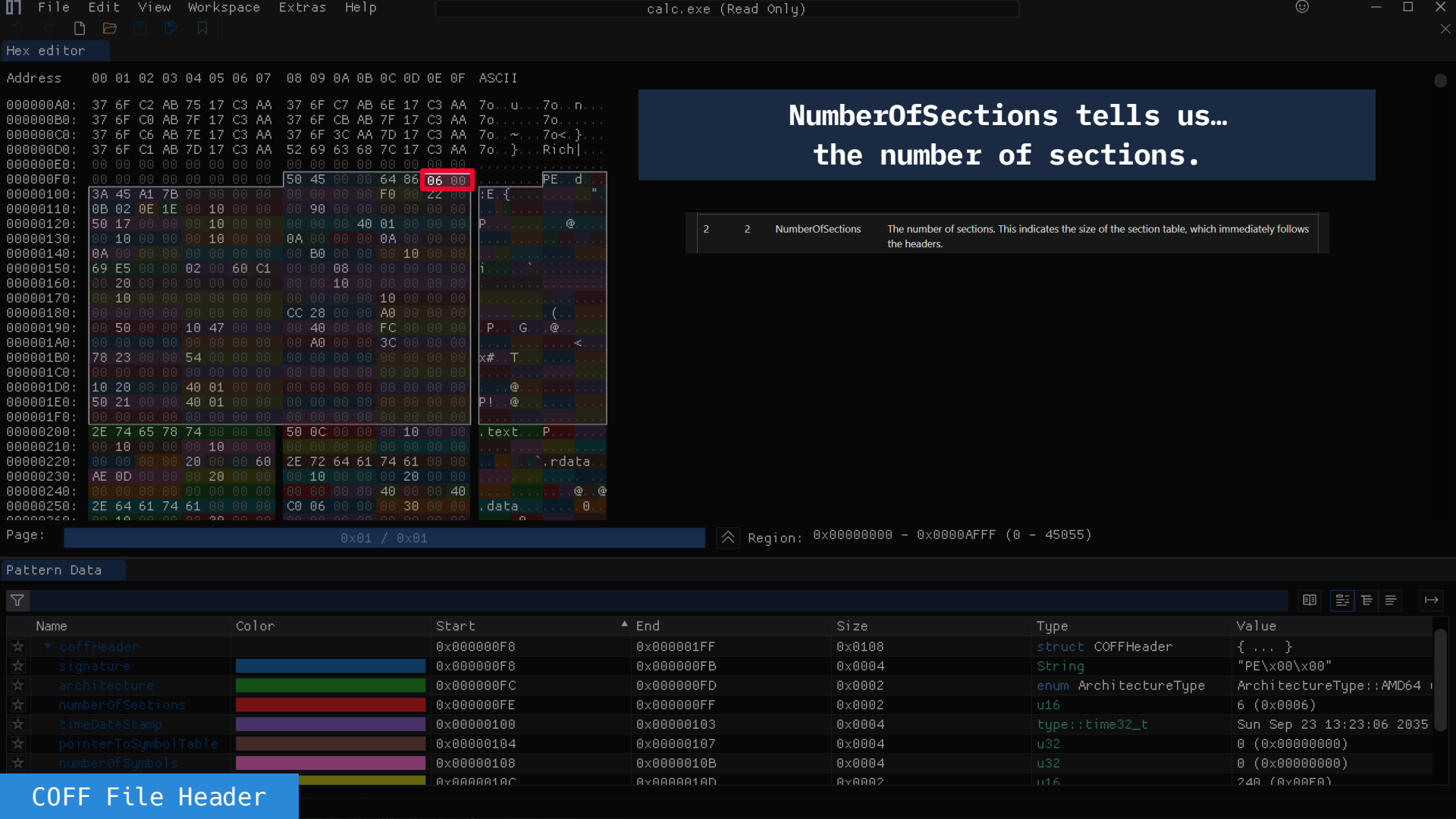
At the beginning of an object file, or immediately after the signature of an image file, is a standard COFF file header in the following format. Note that the Windows loader limits the number of sections to 96.

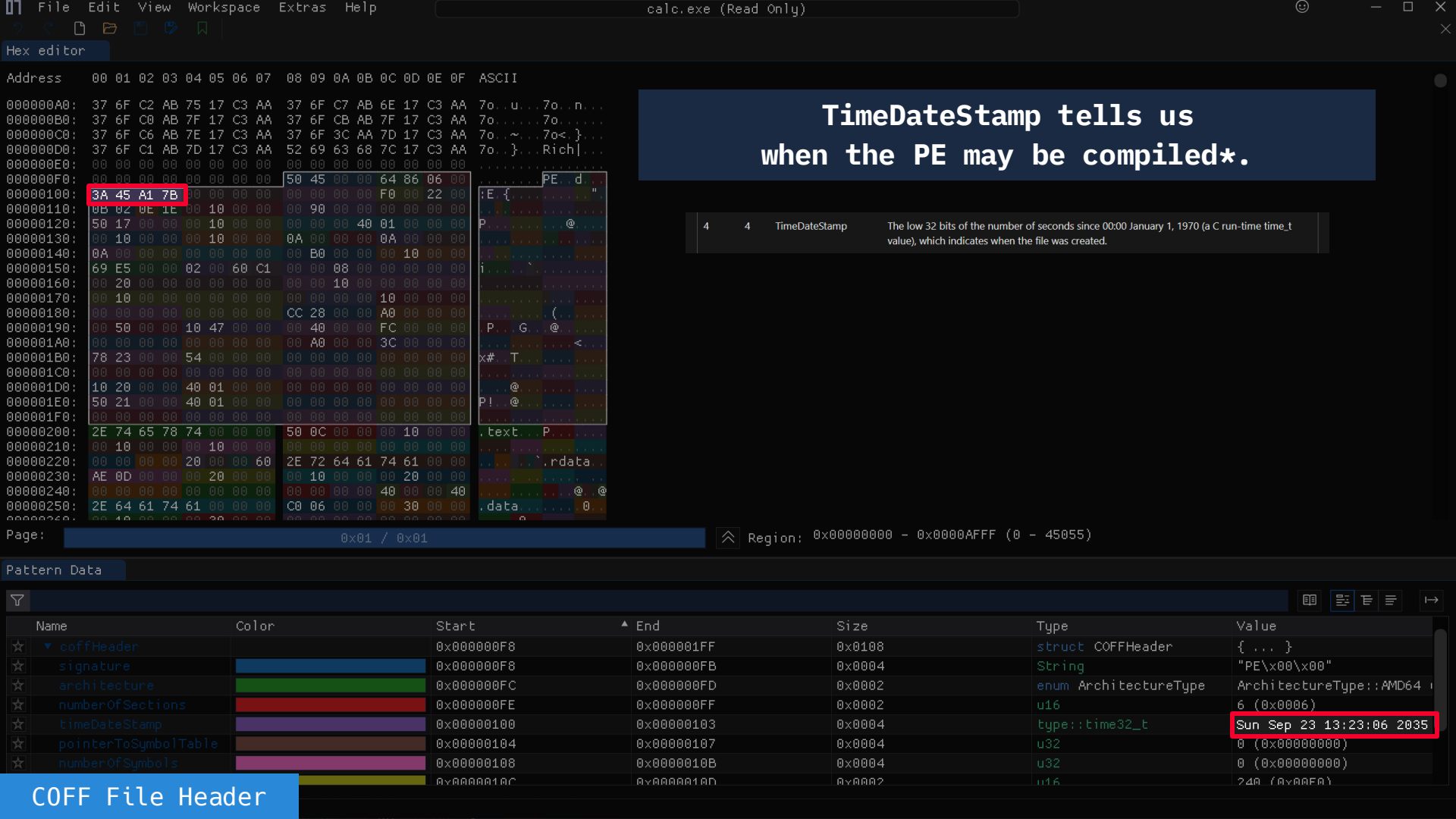
Expand table

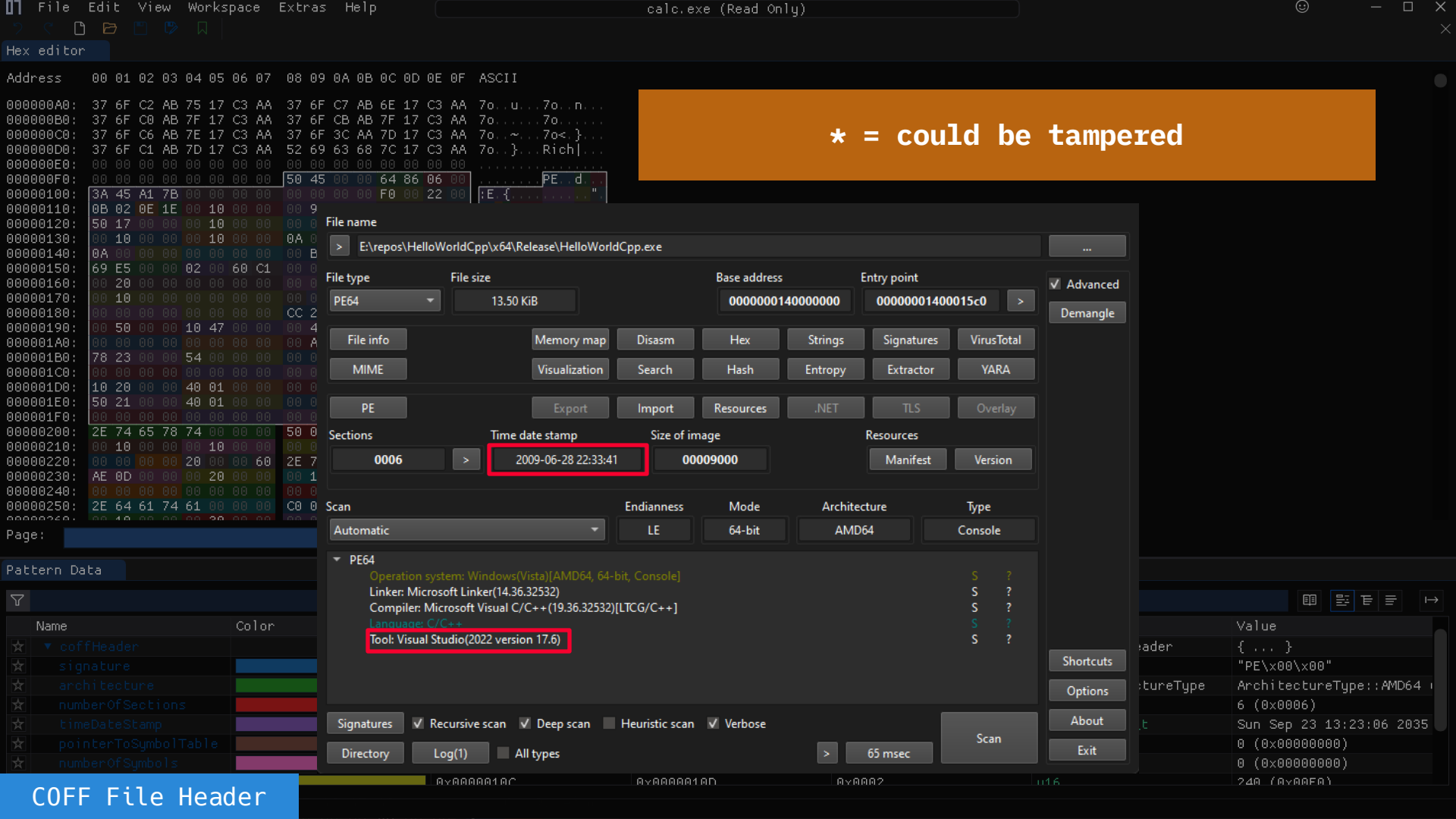
Offset	Size	Field	Description
0	2	Machine	The number that identifies the type of target machine. For more information, see Machine Types.
2	2	NumberOfSections	The number of sections. This indicates the size of the section table, which immediately follows the headers.
4	4	TimeDateStamp	The low 32 bits of the number of seconds since 00:00 January 1, 1970 (a C run-time time_t value), which indicates when the file was created.
8	4	PointerToSymbolTable	The file offset of the COFF symbol table, or zero if no COFF symbol table is present. This value should be zero for an image because COFF debugging information is deprecated.
12	4	NumberOfSymbols	The number of entries in the symbol table. This data can be used to locate the string table, which immediately follows the symbol table. This value should be zero for an image because COFF debugging information is deprecated.
16	2	SizeOfOptionalHeader	The size of the optional header, which is required for executable files but not for object files. This value should be zero for an object file. For a description of the header format, see Optional Header (Image Only).
18	2	Characteristics	The flags that indicate the attributes of the file. For specific flag values, see Characteristics.

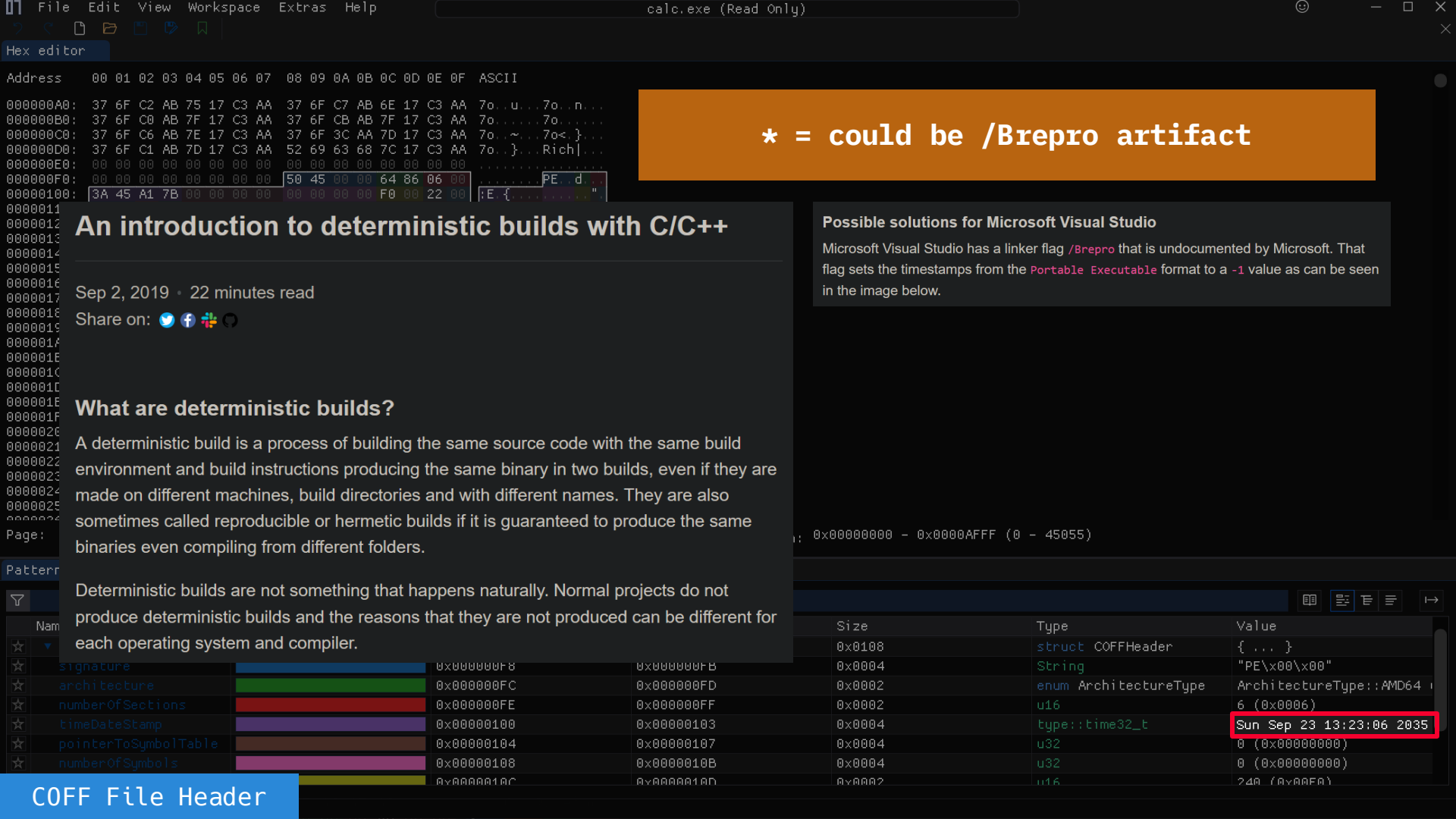
COFF File Header


File Edit View Workspace Extras Help
calc.exe (Read Only)
Hex editor
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
Machine field tells us about the target architecture.
Offset Size Field Description
0 2 Machine The number that identifies the type of target machine. For more information, see Machine Types.
Machine Types
The Machine field has one of the following values, which specify the CPU type. An image file can be run only on the specified machine or on a system that emulates the specified machine.
Expand table
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 0x1c4 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
Page: 0x01 / 0x01
Pattern Data
Name Color Start End
coffHeader 0x000000F8 0x000001
signature 0x000000F8 0x000000
architecture 0x000000FC 0x000000
numberOfSections 0x000000FE 0x000000
timeDateStamp 0x00000100 0x000001
pointerToSymbolTable 0x00000104 0x000001
numberOfSymbols 0x00000108 0x0000010B 0x0004 u32 0 (0x00000000)
COFF File Header

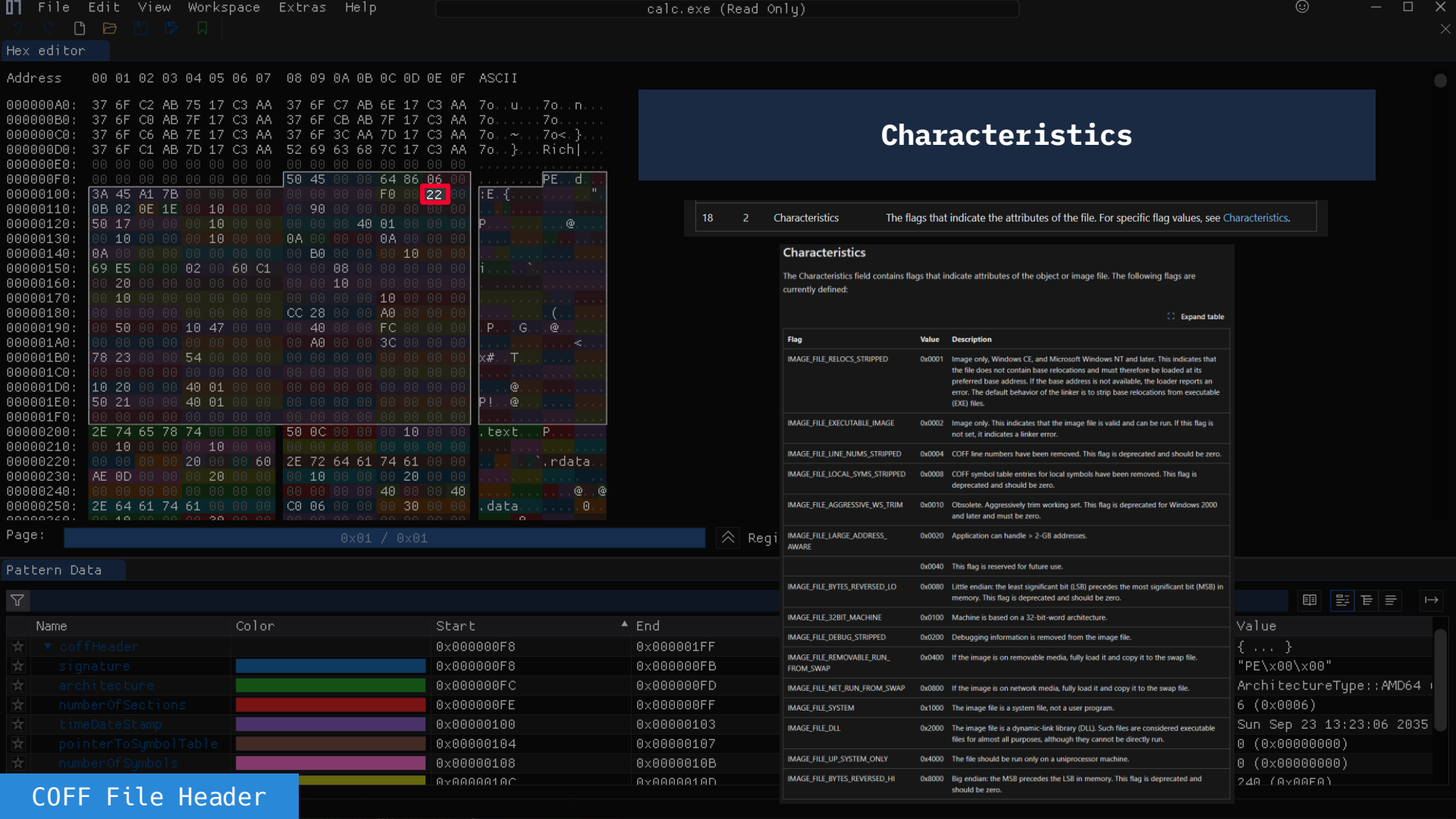

File Edit View Workspace Extras Help
calc.exe (Read Only)
Hex editor
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
NumberOfSections tells us…
the number of sections.
2 2 NumberOfSections The number of sections. This indicates the size of the section table, which immediately follows the headers.
Page: 0x01 / 0x01
Region: 0x00000000 - 0x0000AFFF (0 - 45055)
Pattern Data
Name Color Start End Size Type Value
coffHeader 0x000000F8 0x000001FF 0x0108 struct COFFHeader { ... }
signature 0x000000F8 0x000000FB 0x0004 String "PE\x00\x00"
architecture 0x000000FC 0x000000FD 0x0002 enum ArchitectureType ArchitectureType::AMD64
numberOfSections 0x000000FE 0x000000FF 0x0002 u16 6 (0x0006)
timeDateStamp 0x00000100 0x00000103 0x0004 type::time32_t Sun Sep 23 13:23:06 2035
pointerToSymbolTable 0x00000104 0x00000107 0x0004 u32 0 (0x00000000)
numberOfSymbols 0x00000108 0x0000010B 0x0004 u32 0 (0x00000000)
COFF File Header

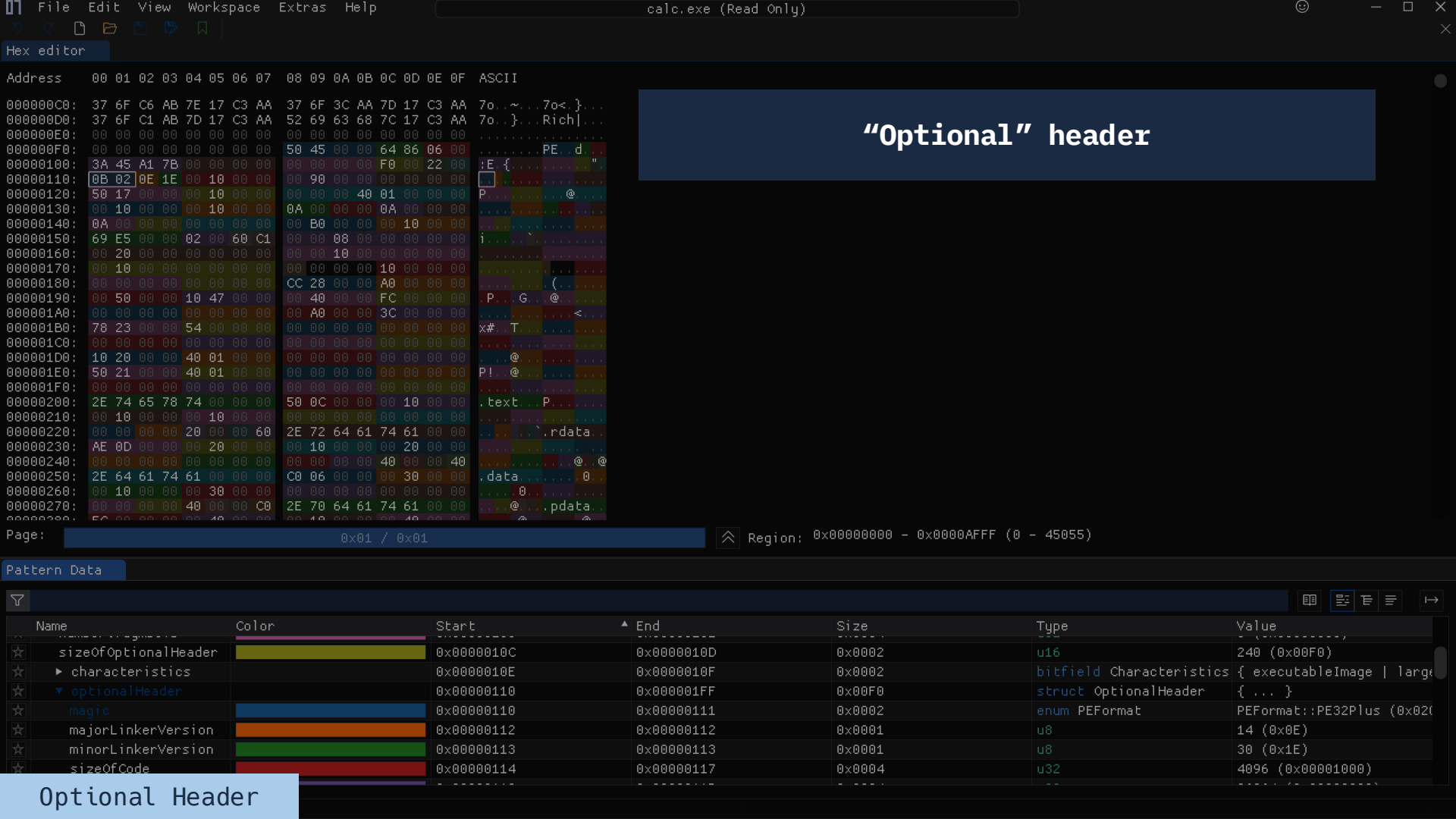

File Edit View Workspace Extras Help
calc.exe (Read Only)
Hex editor
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
3A 45 A1 7B
TimeDateStamp tells us
when the PE may be compiled*.
4 4 TimeDateStamp The low 32 bits of the number of seconds since 00:00 January 1, 1970 (a C run-time time_t value), which indicates when the file was created.
Page: 0x01 / 0x01
Region: 0x00000000 - 0x0000AFFF (0 - 45055)
Pattern Data
Name Color Start End Size Type Value
coffHeader 0x000000F8 0x000001FF 0x0108 struct COFFHeader { ... }
signature 0x000000F8 0x000000FB 0x0004 String "PE\x00\x00"
architecture 0x000000FC 0x000000FD 0x0002 enum ArchitectureType ArchitectureType::AMD64
numberOfSections 0x000000FE 0x000000FF 0x0002 u16 6 (0x0006)
timeDateStamp 0x00000100 0x00000103 0x0004 type::time32_t Sun Sep 23 13:23:06 2035
pointerToSymbolTable 0x00000104 0x00000107 0x0004 u32 0 (0x00000000)
numberOfSymbols 0x00000108 0x0000010B 0x0004 u32 0 (0x00000000)
COFF File Header

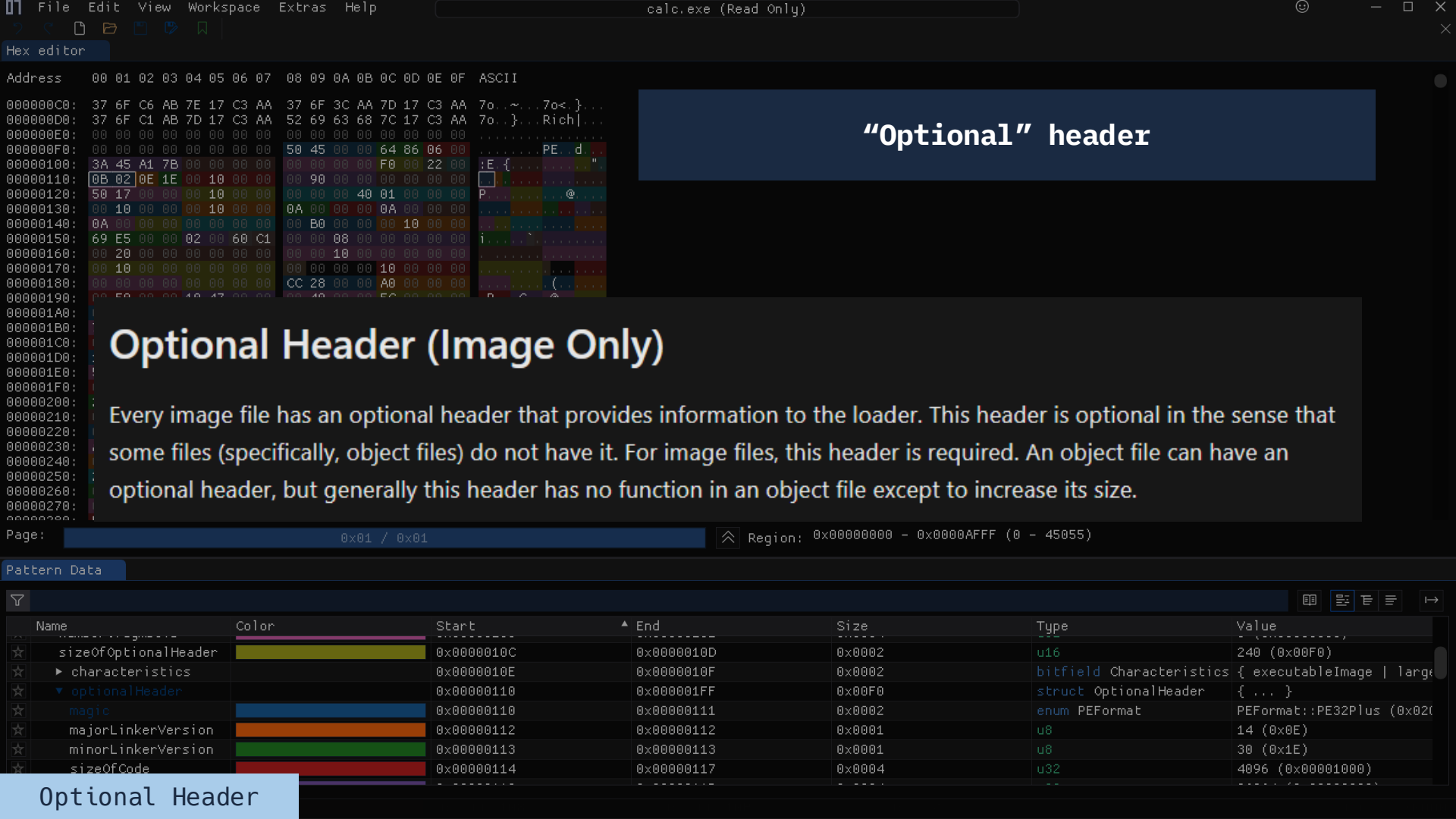

File Edit View Workspace Extras Help
calc.exe (Read Only)
Hex editor
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
* = could be tampered
File name
E:\repos\HelloWorldCpp\x64\Release\HelloWorldCpp.exe
File type
PE64
File size
13.50 KiB
Base address
0000000140000000
Entry point
00000001400015c0
Advanced
Demangle
File info
Memory map
Disasm
Hex
Strings
Signatures
VirusTotal
MIME
Visualization
Search
Hash
Entropy
Extractor
YARA
PE
Export
Import
Resources
.NET
TLS
Overlay
Sections
0006
Time date stamp
2009-06-28 22:33:41
Size of image
00009000
Resources
Manifest
Version
Scan
Automatic
Endianness
LE
Mode
64-bit
Architecture
AMD64
Type
Console
PE64
Operation system: Windows(Vista)[AMD64, 64-bit, Console]
Linker: Microsoft Linker(14.36.32532)
Compiler: Microsoft Visual C/C++(19.36.32532)[LTCG/C++]
Language: C/C++
Tool: Visual Studio(2022 version 17.6)
Shortcuts
Options
About
Exit
Signatures
Recursive scan
Deep scan
Heuristic scan
Verbose
Directory
Log(1)
All types
65 msec
Scan
Pattern Data
Name
Color
Value
coffHeader
signature
architecture
numberOfSections
timeDateStamp
pointerToSymbolTable
numberOfSymbols
"PE\x00\x00"
ArchitectureType::AMD64
6 (0x0006)
Sun Sep 23 13:23:06 2035
0 (0x00000000)
0 (0x00000000)
COFF File Header

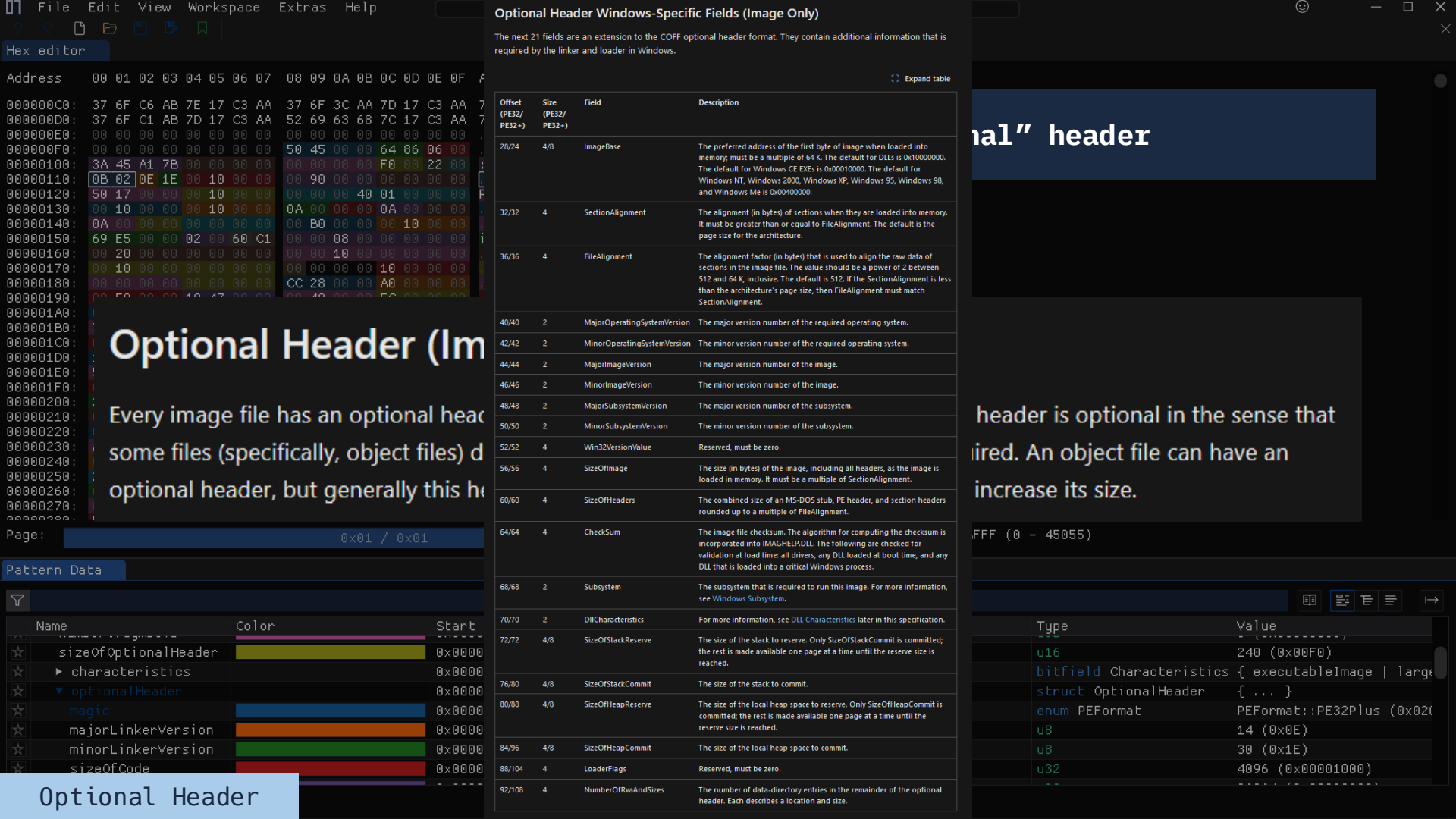

File Edit View Workspace Extras Help
calc.exe (Read Only)
Hex editor
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
000000A0: 37 6F C2 AB 75 17 C3 AA 37 6F C7 AB 6E 17 C3 AA 7o..u...7o..n...
000000B0: 37 6F C0 AB 7F 17 C3 AA 37 6F CB AB 7F 17 C3 AA 7o......7o......
000000C0: 37 6F C6 AB 7E 17 C3 AA 37 6F 3C AA 7D 17 C3 AA 7o..~...7o<.}...
000000D0: 37 6F C1 AB 7D 17 C3 AA 52 69 63 68 7C 17 C3 AA 7o..}...Rich|...
000000E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000F0: 00 00 00 00 00 00 00 00 50 45 00 00 64 86 06 00 ....PE..d...
00000100: 3A 45 A1 7B 00 00 00 00 00 00 00 00 F0 00 22 00 :E.{..........".
* = could be /Brepro artifact
An introduction to deterministic builds with C/C++
Sep 2, 2019 • 22 minutes read
Share on:
What are deterministic builds?
A deterministic build is a process of building the same source code with the same build environment and build instructions producing the same binary in two builds, even if they are made on different machines, build directories and with different names. They are also sometimes called reproducible or hermetic builds if it is guaranteed to produce the same binaries even compiling from different folders.
Deterministic builds are not something that happens naturally. Normal projects do not produce deterministic builds and the reasons that they are not produced can be different for each operating system and compiler.
Possible solutions for Microsoft Visual Studio
Microsoft Visual Studio has a linker flag /Brepro that is undocumented by Microsoft. That flag sets the timestamps from the Portable Executable format to a -1 value as can be seen in the image below.
Page:
0x00000000 - 0x0000AFFF (0 - 45055)
Name Size Type Value
signature 0x00000F8 0x00000FB 0x0004 String "PE\x00\x00"
architecture 0x000000FC 0x000000FD 0x0002 enum ArchitectureType ArchitectureType::AMD64
numberOfSections 0x000000FE 0x000000FF 0x0002 u16 6 (0x0006)
timeDateStamp 0x00000100 0x00000103 0x0004 type::time32_t Sun Sep 23 13:23:06 2035
pointerToSymbolTable 0x00000104 0x00000107 0x0004 u32 0 (0x00000000)
numberOfSymbols 0x00000108 0x0000010B 0x0004 u32 0 (0x00000000)
COFF File Header


File Edit View Workspace Extras Help
calc.exe (Read Only)
Hex editor
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
* = could be /Brepro artifact
An introdu
Sep 2, 2019 • 22
Share on:
What are dete
A deterministic bui
environment and b
made on different
sometimes called
binaries even com
Deterministic build
produce deterministic builds and the reasons that they are not produced can be different for each operating system and compiler.
ed by Microsoft. That
1 value as can be seen
Use the value of the field as a reference, not a fact!
Size
Type
Value
0x0108
struct COFFHeader
{ ... }
0x0004
String
"PE\x00\x00"
0x000000FC
0x000000FD
0x0002
enum ArchitectureType
ArchitectureType::AMD64
0x000000FE
0x000000FF
0x0002
u16
6 (0x0006)
0x00000100
0x00000103
0x0004
type::time32_t
Sun Sep 23 13:23:06 2035
0x00000104
0x00000107
0x0004
u32
0 (0x00000000)
0x00000108
0x0000010B
0x0004
u32
0 (0x00000000)
COFF File Header


File Edit View Workspace Extras Help
calc.exe (Read Only)
Hex editor
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
Characteristics
18 2 Characteristics The flags that indicate the attributes of the file. For specific flag values, see Characteristics.
Characteristics
The Characteristics field contains flags that indicate attributes of the object or image file. The following flags are currently defined:
Expand table
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.
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.
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.
Page: 0x01 / 0x01
Pattern Data
Name Color Start End Value
coffHeader 0x000000F8 0x000001FF { ... }
signature 0x000000F8 0x000000FB "PE\x00\x00"
architecture 0x000000FC 0x000000FD ArchitectureType::AMD64
numberOfSections 0x000000FE 0x000000FF 6 (0x0006)
timeDateStamp 0x00000100 0x00000103 Sun Sep 23 13:23:06 2035
pointerToSymbolTable 0x00000104 0x00000107 0 (0x00000000)
numberOfSymbols 0x00000108 0x0000010B 0 (0x00000000)
COFF File Header


File Edit View Workspace Extras Help
calc.exe (Read Only)
Hex editor
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
000000C0: 37 6F C6 AB 7E 17 C3 AA 37 6F 3C AA 7D 17 C3 AA 7o..~...7o<.}...
000000D0: 37 6F C1 AB 7D 17 C3 AA 52 69 63 68 7C 17 C3 AA 7o..}...Rich|...
000000E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000F0: 00 00 00 00 00 00 00 00 50 45 00 00 64 86 06 00 ....PE..d...
00000100: 3A 45 A1 7B 00 00 00 00 00 00 00 00 F0 00 22 00 :E.{..........".
00000110: 0B 02 0E 1E 00 10 00 00 00 90 00 00 00 00 00 00 ....
00000120: 50 17 00 00 00 10 00 00 00 00 00 00 40 01 00 00 P...........@...
00000130: 00 10 00 00 00 10 00 00 0A 00 00 00 0A 00 00 00 ....
00000140: 0A 00 00 00 00 00 00 00 00 B0 00 00 00 10 00 00 ....
00000150: 69 E5 00 00 02 00 60 C1 00 00 08 00 00 00 00 00 i.....`.........
00000160: 00 20 00 00 00 00 00 00 00 00 10 00 00 00 00 00 . ...
00000170: 00 10 00 00 00 00 00 00 00 00 00 00 10 00 00 00 ....
00000180: 00 00 00 00 00 00 00 00 CC 28 00 00 A0 00 00 00 ....(......
00000190: 00 50 00 00 10 47 00 00 00 40 00 00 FC 00 00 00 .P...G...@......
000001A0: 00 00 00 00 00 00 00 00 00 A0 00 00 3C 00 00 00 ....<...
000001B0: 78 23 00 00 54 00 00 00 00 00 00 00 00 00 00 00 x#..T...........
000001C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000001D0: 10 20 00 00 40 01 00 00 00 00 00 00 00 00 00 00 . ..@...........
000001E0: 50 21 00 00 40 01 00 00 00 00 00 00 00 00 00 00 P!..@...........
000001F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000200: 2E 74 65 78 74 00 00 00 50 0C 00 00 00 10 00 00 .text...P.......
00000210: 00 10 00 00 00 10 00 00 00 00 00 00 00 00 00 00 ....
00000220: 00 00 00 00 20 00 00 60 2E 72 64 61 74 61 00 00 .... ..`.rdata..
00000230: AE 0D 00 00 00 20 00 00 00 10 00 00 00 20 00 00 ....
00000240: 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 40 ....@..@
00000250: 2E 64 61 74 61 00 00 00 C0 06 00 00 00 30 00 00 .data........0..
00000260: 00 10 00 00 00 30 00 00 00 00 00 00 00 00 00 00 .....0..........
00000270: 00 00 00 00 40 00 00 C0 2E 70 64 61 74 61 00 00 ....@....pdata..
Page: 0x01 / 0x01
Region: 0x00000000 - 0x0000AFFF (0 - 45055)
Pattern Data
Name Color Start End Size Type Value
sizeOfOptionalHeader 0x0000010C 0x0000010D 0x0002 u16 240 (0x00F0)
characteristics 0x0000010E 0x0000010F 0x0002 bitfield Characteristics { executableImage | large
optionalHeader 0x00000110 0x000001FF 0x00F0 struct OptionalHeader { ... }
magic 0x00000110 0x00000111 0x0002 enum PEFormat PEFormat::PE32Plus (0x020
majorLinkerVersion 0x00000112 0x00000112 0x0001 u8 14 (0x0E)
minorLinkerVersion 0x00000113 0x00000113 0x0001 u8 30 (0x1E)
sizeOfCode 0x00000114 0x00000117 0x0004 u32 4096 (0x00001000)
“Optional” header
Optional Header


File Edit View Workspace Extras Help
calc.exe (Read Only)
Hex editor
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
"Optional" header
Optional Header (Image Only)
Every image file has an optional header that provides information to the loader. This header is optional in the sense that some files (specifically, object files) do not have it. For image files, this header is required. An object file can have an optional header, but generally this header has no function in an object file except to increase its size.
Page: 0x01 / 0x01
Region: 0x00000000 - 0x0000AFFF (0 - 45055)
Pattern Data
Name Color Start End Size Type Value
sizeOfOptionalHeader 0x0000010C 0x0000010D 0x0002 u16 240 (0x00F0)
characteristics 0x0000010E 0x0000010F 0x0002 bitfield Characteristics { executableImage | large
optionalHeader 0x00000110 0x000001FF 0x00F0 struct OptionalHeader { ... }
magic 0x00000110 0x00000111 0x0002 enum PEFormat PEFormat::PE32Plus (0x020
majorLinkerVersion 0x00000112 0x00000112 0x0001 u8 14 (0x0E)
minorLinkerVersion 0x00000113 0x00000113 0x0001 u8 30 (0x1E)
sizeOfCode 0x00000114 0x00000117 0x0004 u32 4096 (0x00001000)
Optional Header

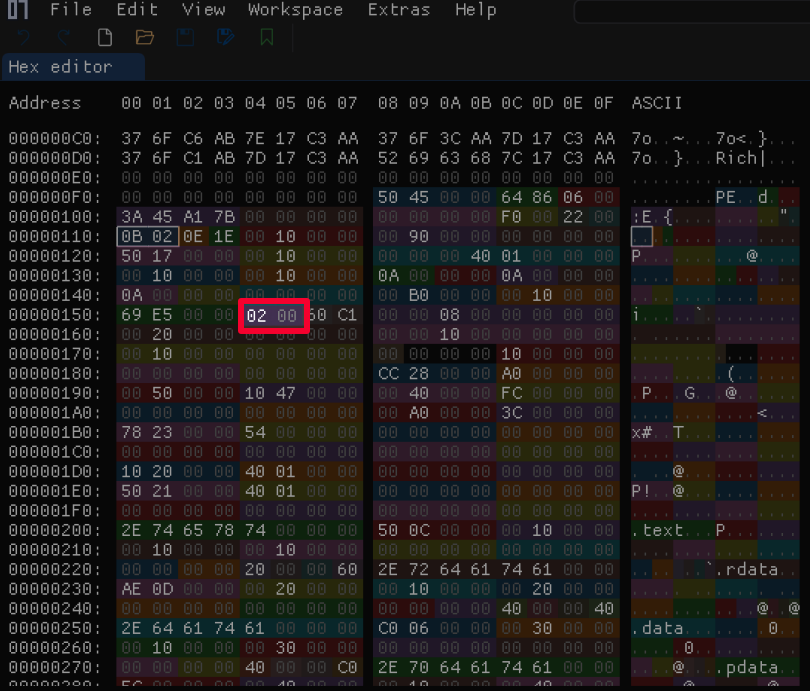


Optional Header Windows-Specific Fields (Image Only)

The next 21 fields are an extension to the COFF optional header format. They contain additional information that is required by the linker and loader in Windows.

Expand table

Offset (PE32/ PE32+)	Size (PE32/ PE32+)	Field	Description
28/24	4/8	ImageBase	The preferred address of the first byte of image when loaded into memory; must be a multiple of 64 K. The default for DLLs is 0x10000000. The default for Windows CE EXEs is 0x00010000. The default for Windows NT, Windows 2000, Windows XP, Windows 95, Windows 98, and Windows Me is 0x00400000.
32/32	4	SectionAlignment	The alignment (in bytes) of sections when they are loaded into memory. It must be greater than or equal to FileAlignment. The default is the page size for the architecture.
36/36	4	FileAlignment	The alignment factor (in bytes) that is used to align the raw data of sections in the image file. The value should be a power of 2 between 512 and 64 K, inclusive. The default is 512. If the SectionAlignment is less than the architecture's page size, then FileAlignment must match SectionAlignment.
40/40	2	MajorOperatingSystemVersion	The major version number of the required operating system.
42/42	2	MinorOperatingSystemVersion	The minor version number of the required operating system.
44/44	2	MajorImageVersion	The major version number of the image.
46/46	2	MinorImageVersion	The minor version number of the image.
48/48	2	MajorSubsystemVersion	The major version number of the subsystem.
50/50	2	MinorSubsystemVersion	The minor version number of the subsystem.
52/52	4	Win32VersionValue	Reserved, must be zero.
56/56	4	SizeOfImage	The size (in bytes) of the image, including all headers, as the image is loaded in memory. It must be a multiple of SectionAlignment.
60/60	4	SizeOfHeaders	The combined size of an MS-DOS stub, PE header, and section headers rounded up to a multiple of FileAlignment.
64/64	4	CheckSum	The image file checksum. The algorithm for computing the checksum is incorporated into IMAGHELP.DLL. The following are checked for validation at load time: all drivers, any DLL loaded at boot time, and any DLL that is loaded into a critical Windows process.
68/68	2	Subsystem	The subsystem that is required to run this image. For more information, see Windows Subsystem.
70/70	2	DllCharacteristics	For more information, see DLL Characteristics later in this specification.
72/72	4/8	SizeOfStackReserve	The size of the stack to reserve. Only SizeOfStackCommit is committed; the rest is made available one page at a time until the reserve size is reached.
76/80	4/8	SizeOfStackCommit	The size of the stack to commit.
80/88	4/8	SizeOfHeapReserve	The size of the local heap space to reserve. Only SizeOfHeapCommit is committed; the rest is made available one page at a time until the reserve size is reached.
84/96	4/8	SizeOfHeapCommit	The size of the local heap space to commit.
88/104	4	LoaderFlags	Reserved, must be zero.
92/108	4	NumberOfRvaAndSizes	The number of data-directory entries in the remainder of the optional header. Each describes a location and size.



Target subsystem

68/68	2	Subsystem	The subsystem that is required to run this image. For more information, see Windows Subsystem.

Windows Subsystem

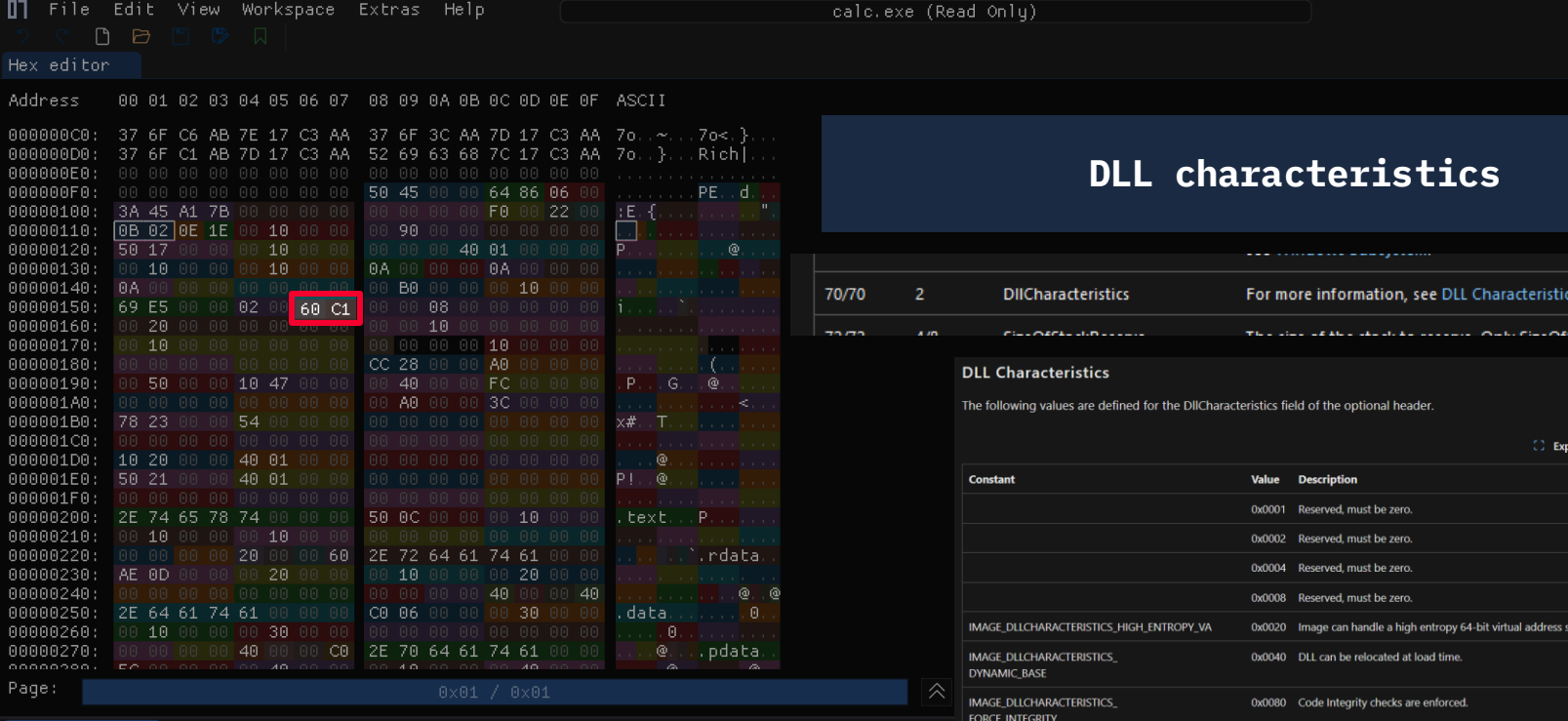
The following values defined for the Subsystem field of the optional header determine which Windows subsystem (if any) is required to run the image.

Expand table

Constant	Value	Description
IMAGE_SUBSYSTEM_UNKNOWN	0	An unknown subsystem
IMAGE_SUBSYSTEM_NATIVE	1	Device drivers and native Windows processes
IMAGE_SUBSYSTEM_WINDOWS_GUI	2	The Windows graphical user interface (GUI) subsystem
IMAGE_SUBSYSTEM_WINDOWS_CUI	3	The Windows character subsystem
IMAGE_SUBSYSTEM_OS2_CUI	5	The OS/2 character subsystem
IMAGE_SUBSYSTEM_POSIX_CUI	7	The Posix character subsystem
IMAGE_SUBSYSTEM_NATIVE_WINDOWS	8	Native Win9x driver
IMAGE_SUBSYSTEM_WINDOWS_CE_GUI	9	Windows CE
IMAGE_SUBSYSTEM_EFI_APPLICATION	10	An Extensible Firmware Interface (EFI) application
IMAGE_SUBSYSTEM_EFI_BOOT_ SERVICE_DRIVER	11	An EFI driver with boot services
IMAGE_SUBSYSTEM_EFI_RUNTIME_ DRIVER	12	An EFI driver with run-time services
IMAGE_SUBSYSTEM_EFI_ROM	13	An EFI ROM image
IMAGE_SUBSYSTEM_XBOX	14	XBOX
IMAGE_SUBSYSTEM_WINDOWS_BOOT_APPLICATION	16	Windows boot application.

Optional Header

DLL characteristics



70/70	2	DllCharacteristics	For more information, see DLL Characteristics later in this specification.

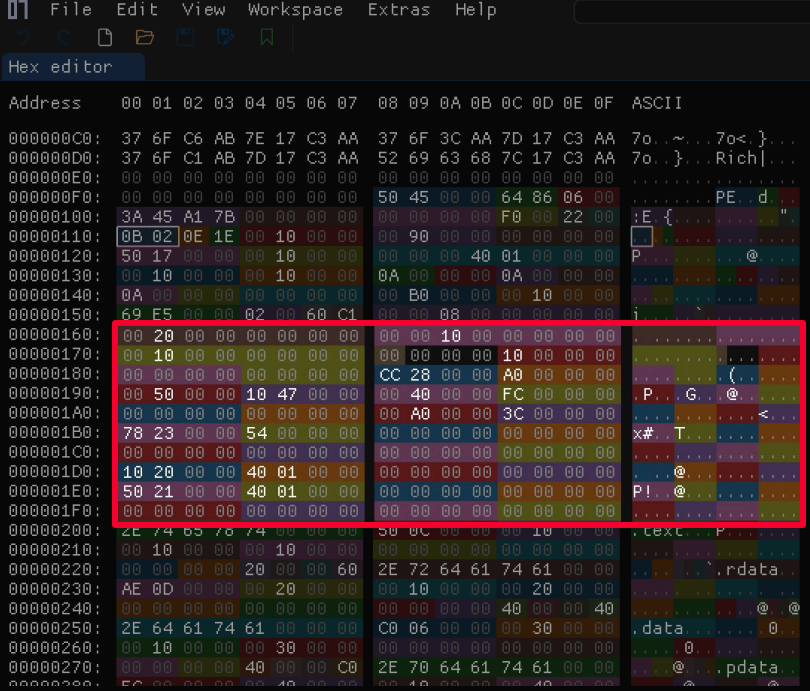
DLL Characteristics

The following values are defined for the DllCharacteristics field of the optional header.

Expand table

Constant	Value	Description
	0x0001	Reserved, must be zero.
	0x0002	Reserved, must be zero.
	0x0004	Reserved, must be zero.
	0x0008	Reserved, must be zero.
IMAGE_DLLCHARACTERISTICS_HIGH_ENTROPY_VA	0x0020	Image can handle a high entropy 64-bit virtual address space.
IMAGE_DLLCHARACTERISTICS_ DYNAMIC_BASE	0x0040	DLL can be relocated at load time.
IMAGE_DLLCHARACTERISTICS_ FORCE_INTEGRITY	0x0080	Code Integrity checks are enforced.
IMAGE_DLLCHARACTERISTICS_ NX_COMPAT	0x0100	Image is NX compatible.
IMAGE_DLLCHARACTERISTICS_ NO_ISOLATION	0x0200	Isolation aware, but do not isolate the image.
IMAGE_DLLCHARACTERISTICS_ NO_SEH	0x0400	Does not use structured exception (SE) handling. No SE handler may be called in this image.
IMAGE_DLLCHARACTERISTICS_ NO_BIND	0x0800	Do not bind the image.
IMAGE_DLLCHARACTERISTICS_APPCONTAINER	0x1000	Image must execute in an AppContainer.
IMAGE_DLLCHARACTERISTICS_ WDM_DRIVER	0x2000	A WDM driver.
IMAGE_DLLCHARACTERISTICS_GUARD_CF	0x4000	Image supports Control Flow Guard.
IMAGE_DLLCHARACTERISTICS_ TERMINAL_SERVER_AWARE	0x8000	Terminal Server aware.

Optional Header



Optional Header Data Directories

Each data directory gives the address and size of a table or string that Windows uses. These data directory entries are all loaded into memory so that the system can use them at run time. A data directory is an 8-byte field that has the following declaration:

```
C++

typedef struct _IMAGE_DATA_DIRECTORY {
    DWORD   VirtualAddress;
    DWORD   Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

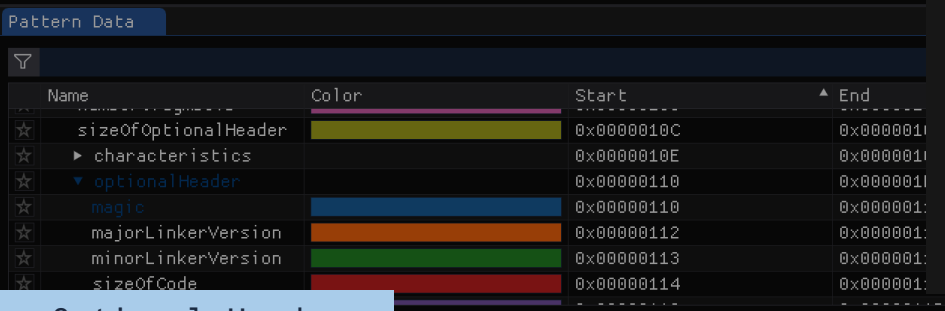
The first field, VirtualAddress, is actually the RVA of the table. The RVA is the address of the table relative to the base address of the image when the table is loaded. The second field gives the size in bytes. The data directories, which form the last part of the optional header, are listed in the following table.

Note that the number of directories is not fixed. Before looking for a specific directory, check the NumberOfRvaAndSizes field in the optional header.

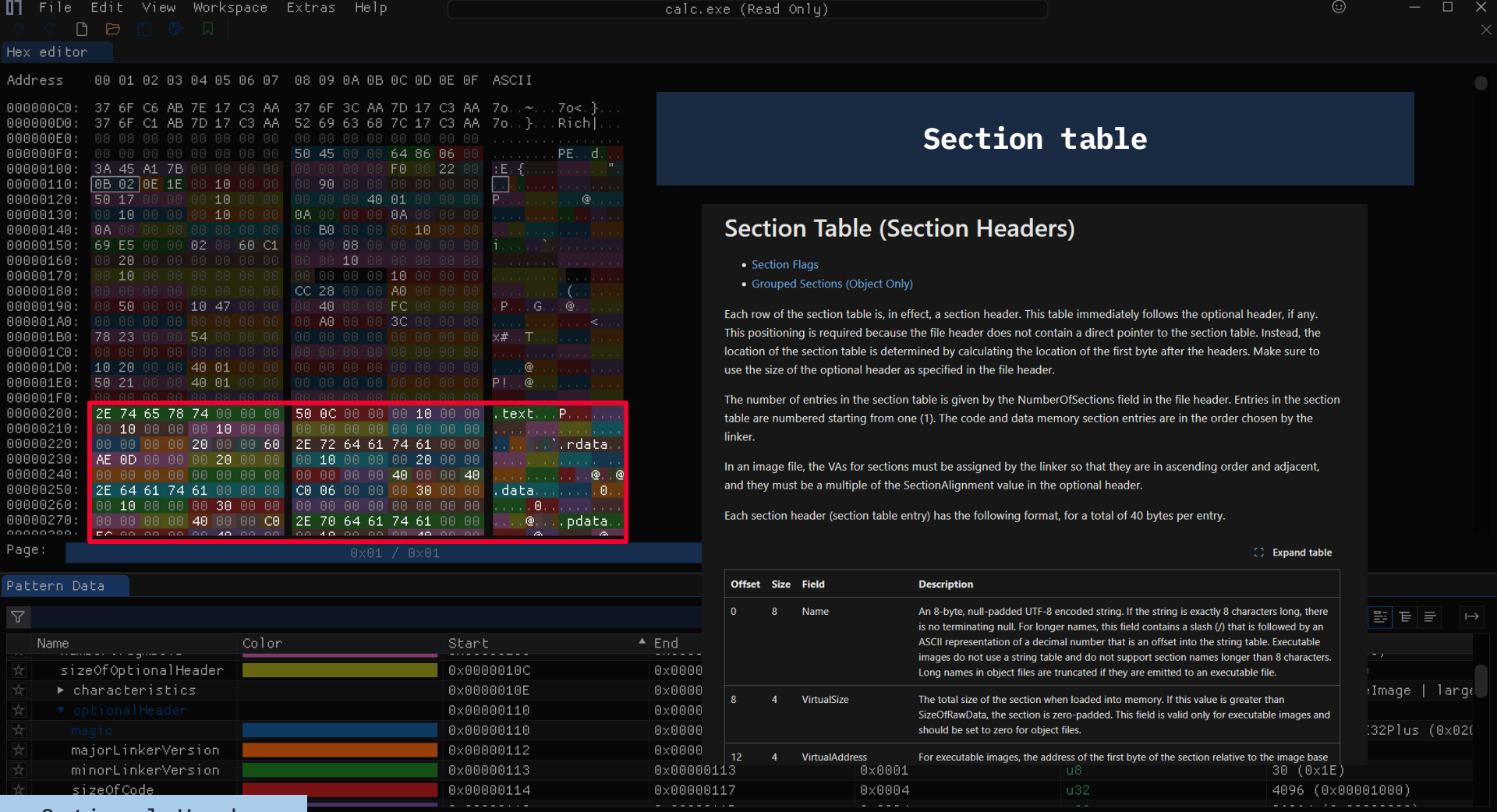
Also, do not assume that the RVAs in this table point to the beginning of a section or that the sections that contain specific tables have specific names.

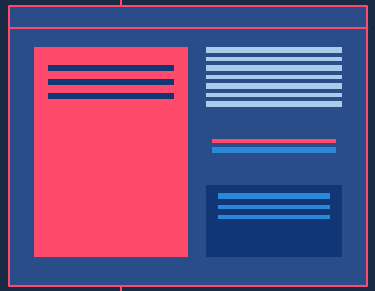
Expand table

Offset (PE/ PE32+)	Size	Field	Description
96/112	8	Export Table	The export table address and size. For more information see .edata Section (Image Only).
104/120	8	Import Table	The import table address and size. For more information, see The .idata Section.
112/128	8	Resource Table	The resource table address and size. For more information, see The .rsrc Section.
120/136	8	Exception Table	The exception table address and size. For more information, see The .pdata Section.
128/144	8	Certificate Table	The attribute certificate table address and size. For more information, see The Attribute Certificate Table (Image Only).

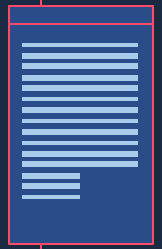


Optional Header


File Edit View Workspace Extras Help
calc.exe (Read Only)
Hex editor
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
Page: 0x01 / 0x01
Pattern Data
Name Color Start End
sizeOfOptionalHeader 0x0000010C 0x0000
characteristics 0x0000010E 0x0000
optionalHeader 0x00000110 0x0000
magic 0x00000110 0x0000
majorLinkerVersion 0x00000112 0x0000
minorLinkerVersion 0x00000113 0x00000113 0x0001 u8 30 (0x1E)
sizeOfCode 0x00000114 0x00000117 0x0004 u32 4096 (0x00001000)
Section table
Section Table (Section Headers)
Section Flags
Grouped Sections (Object Only)
Each row of the section table is, in effect, a section header. This table immediately follows the optional header, if any. This positioning is required because the file header does not contain a direct pointer to the section table. Instead, the location of the section table is determined by calculating the location of the first byte after the headers. Make sure to use the size of the optional header as specified in the file header.
The number of entries in the section table is given by the NumberOfSections field in the file header. Entries in the section table are numbered starting from one (1). The code and data memory section entries are in the order chosen by the linker.
In an image file, the VAs for sections must be assigned by the linker so that they are in ascending order and adjacent, and they must be a multiple of the SectionAlignment value in the optional header.
Each section header (section table entry) has the following format, for a total of 40 bytes per entry.
Expand table
Offset Size Field Description
0 8 Name An 8-byte, null-padded UTF-8 encoded string. If the string is exactly 8 characters long, there is no terminating null. For longer names, this field contains a slash (/) that is followed by an ASCII representation of a decimal number that is an offset into the string table. Executable images do not use a string table and do not support section names longer than 8 characters. Long names in object files are truncated if they are emitted to an executable file.
8 4 VirtualSize The total size of the section when loaded into memory. If this value is greater than SizeOfRawData, the section is zero-padded. This field is valid only for executable images and should be set to zero for object files.
12 4 VirtualAddress For executable images, the address of the first byte of the section relative to the image base
Optional Header

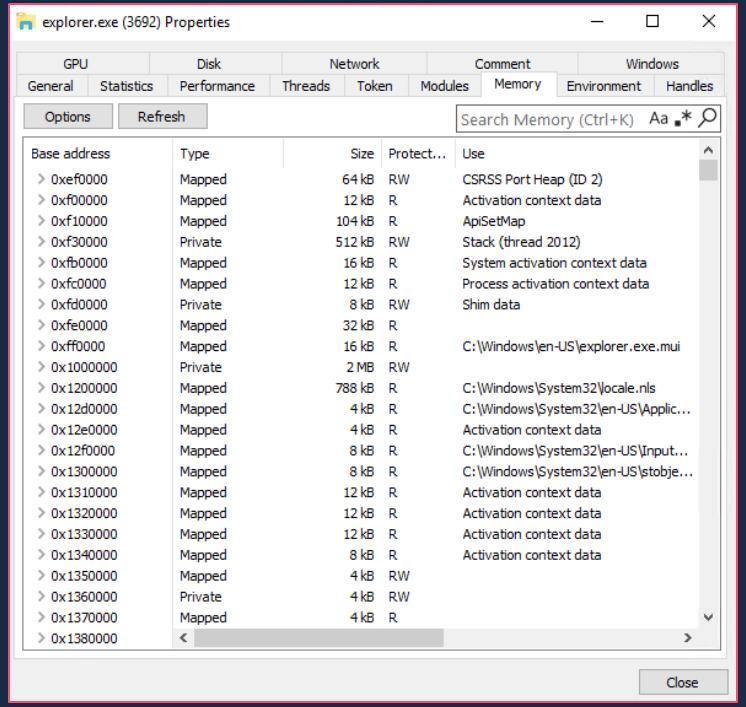


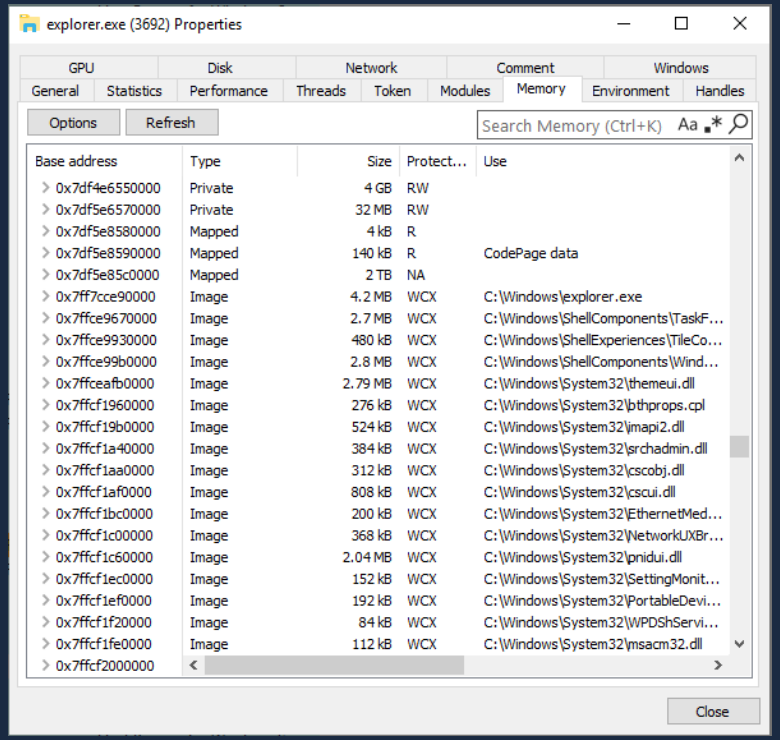
What does it look like in practice?



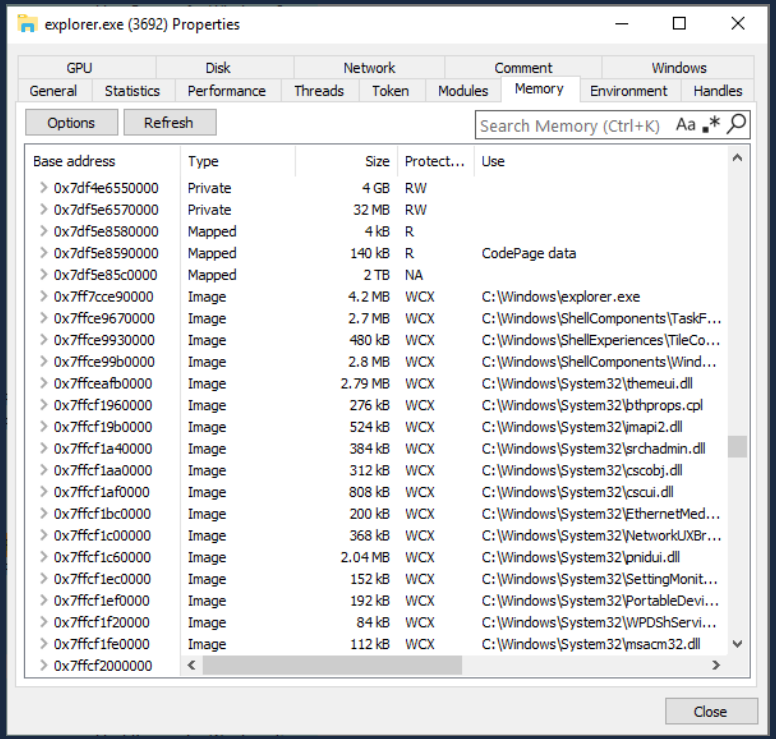
Workshop #2

Examine a process' loaded modules via System Informer

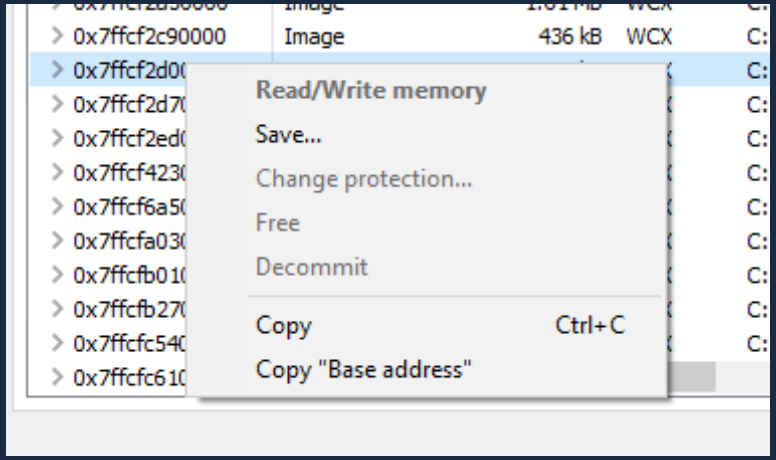




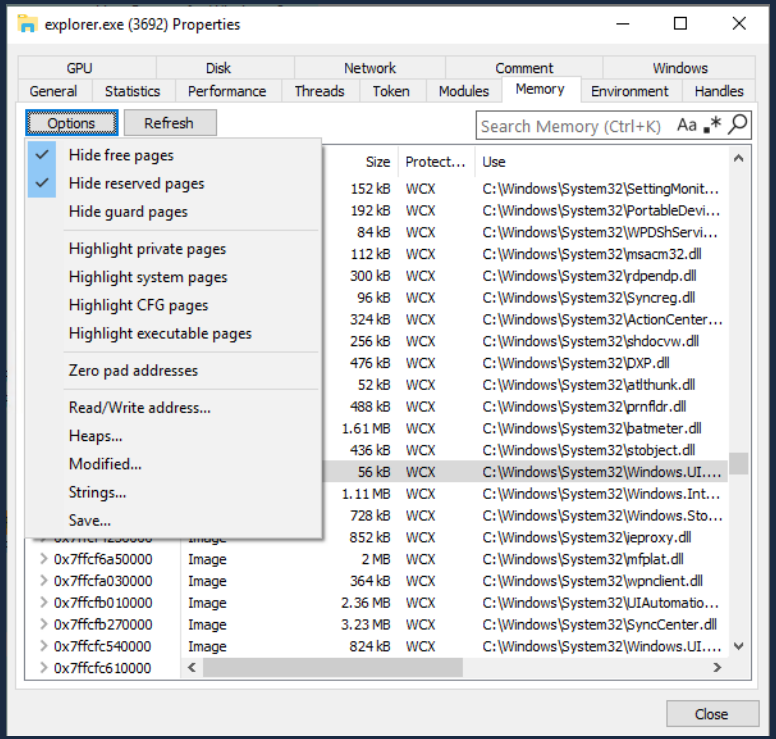
Select the “Memory” tab



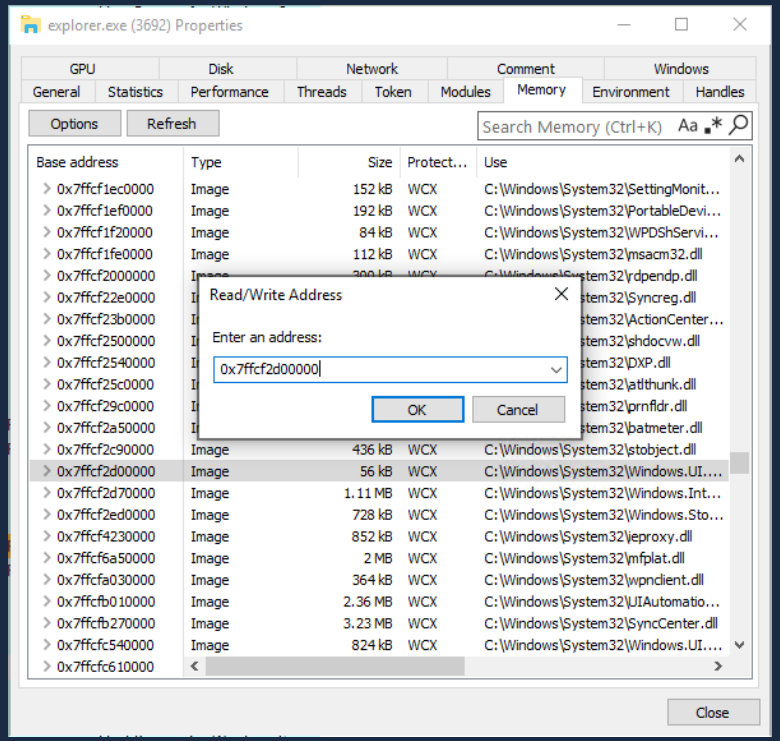
Select the “Memory” tab, scroll down to rows labeled “Image”



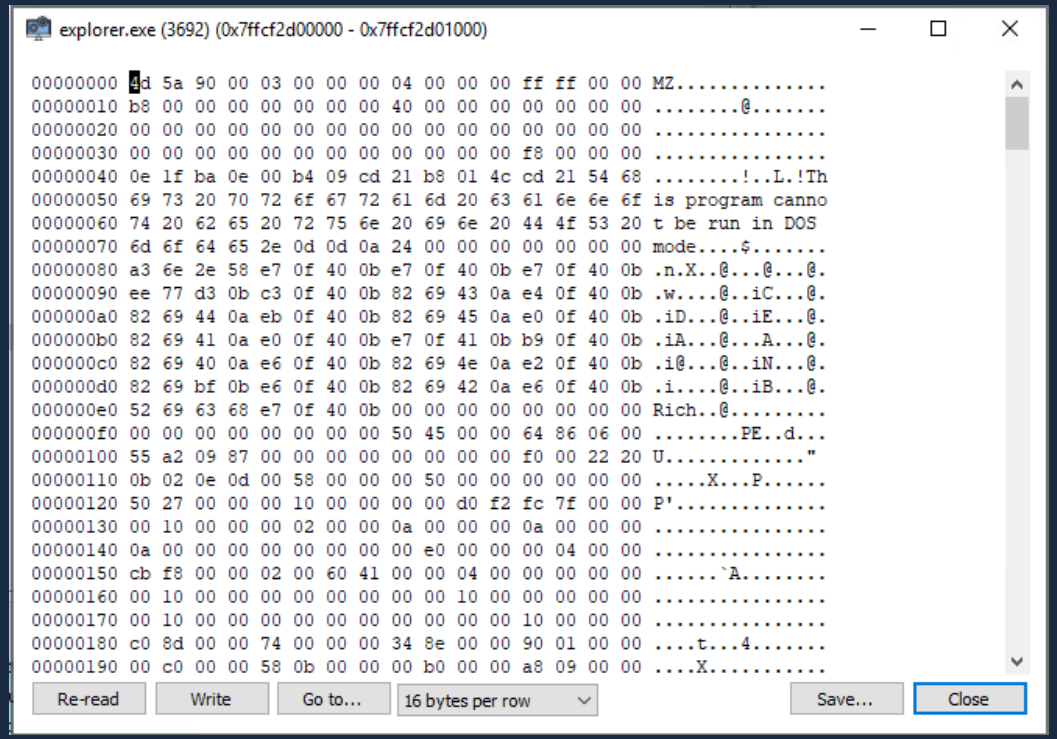
Select any of them, right click, Copy “Base address”



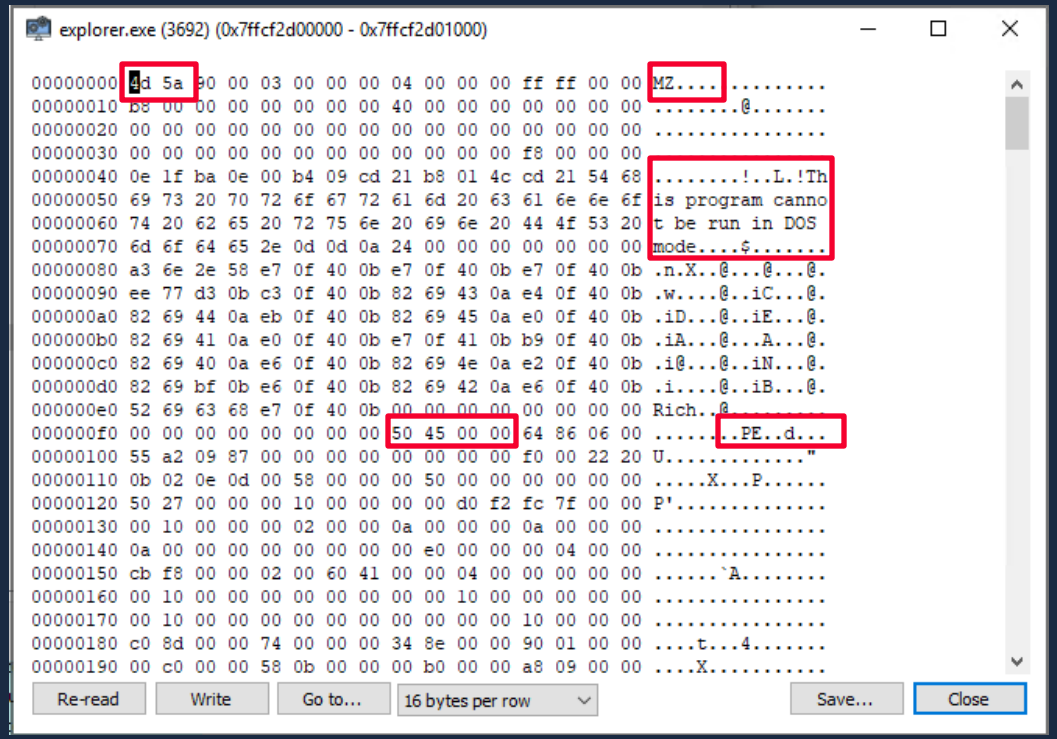
Click “Options” > “Read/Write address…”



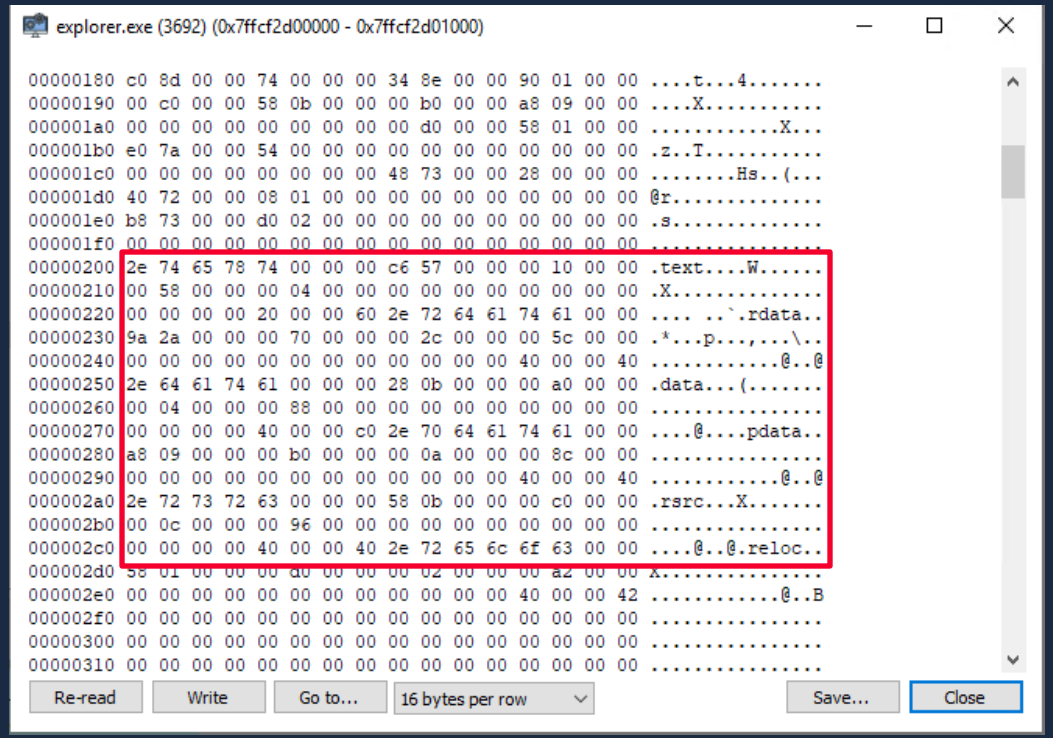
Paste the address you copied



This is what the loaded PE module in memory looks like!



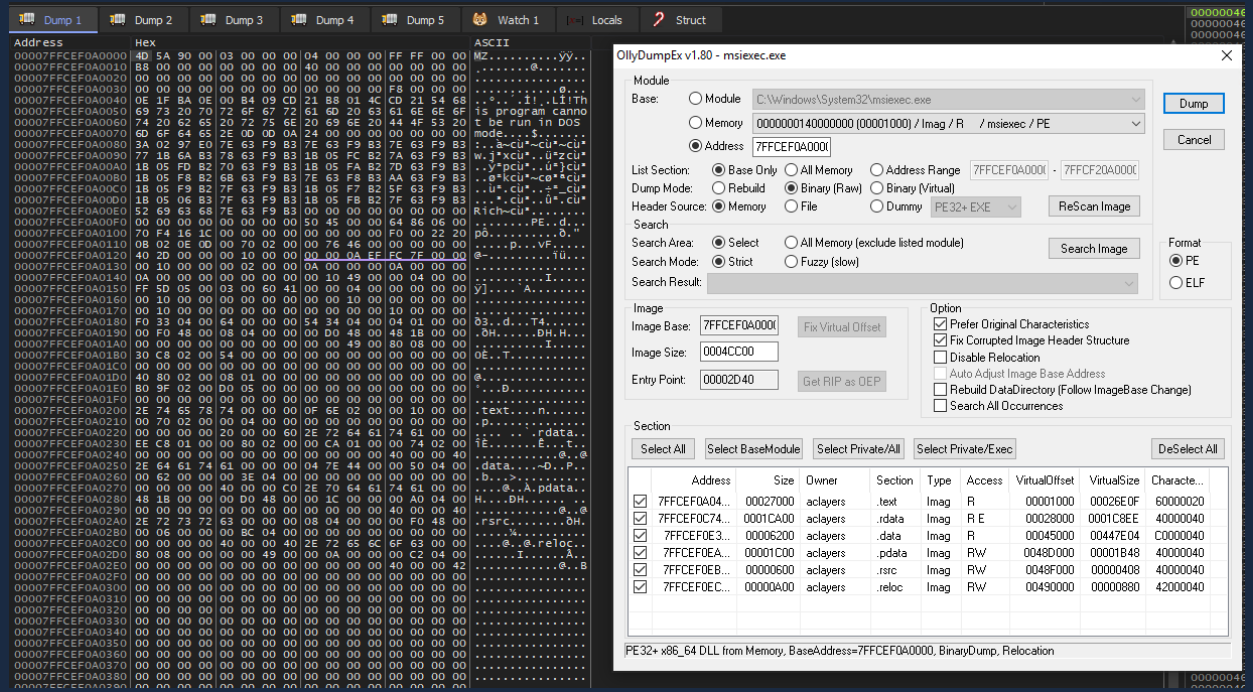
MZ + PE headers



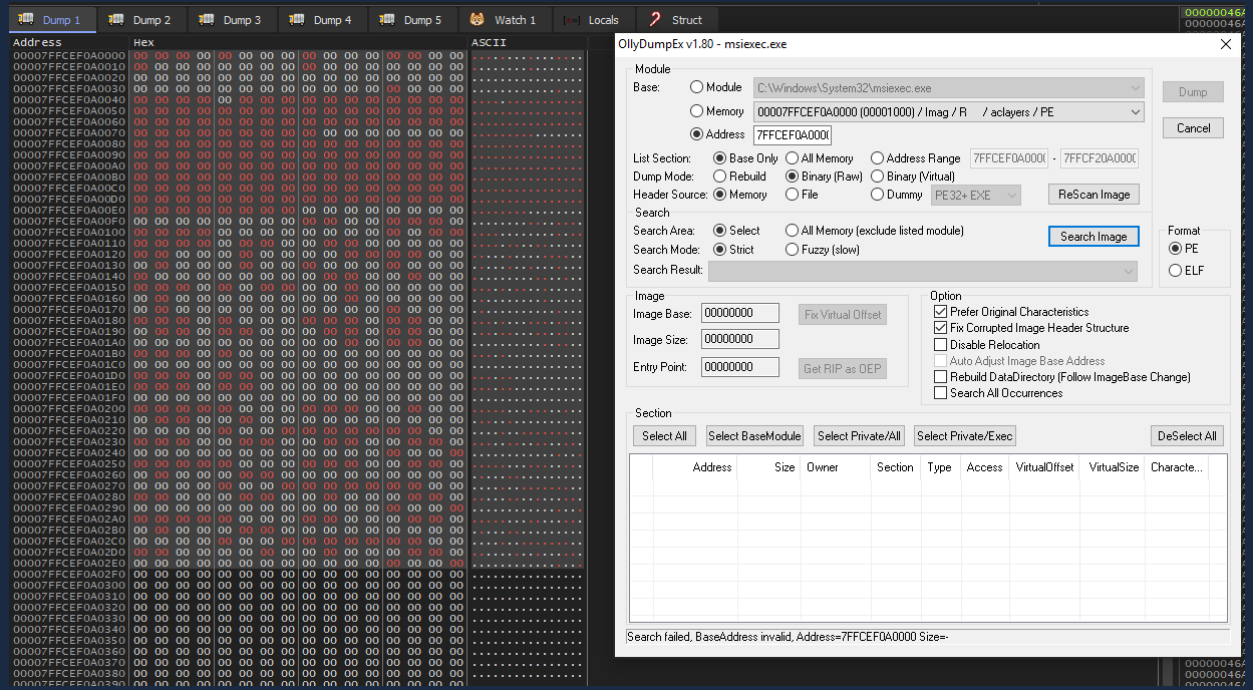
Common section names

What if I take those headers away?

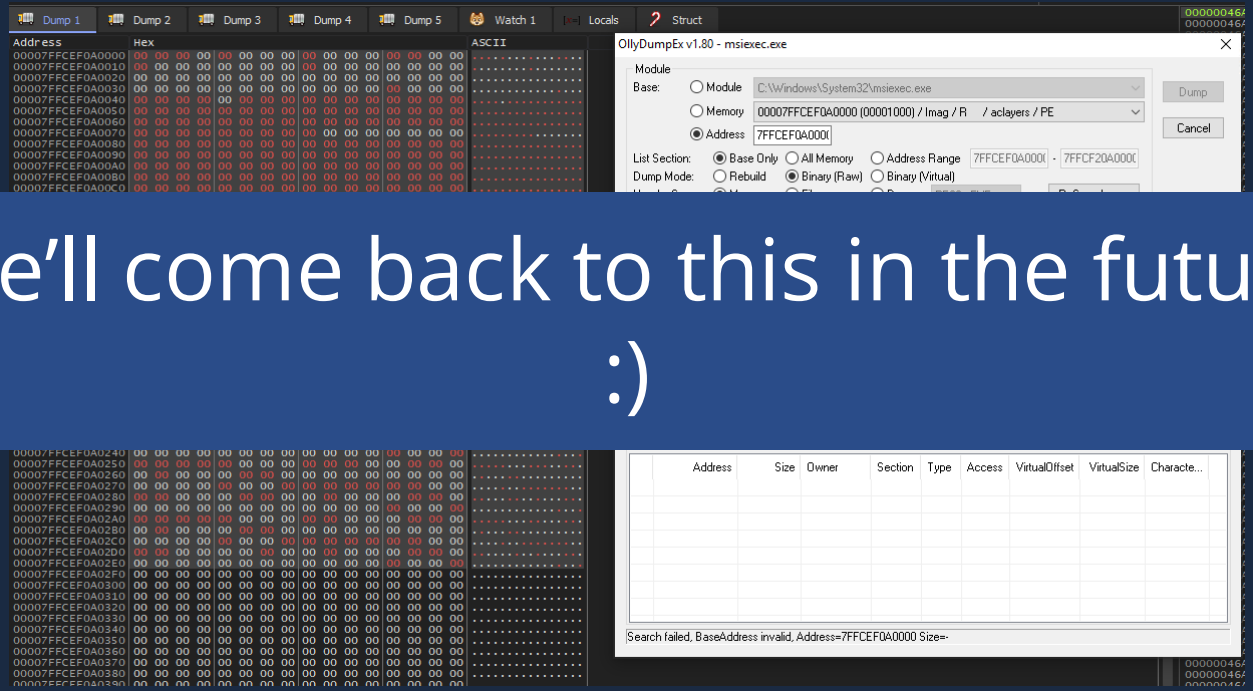
Dump w/ PE header in tact = OK



Dump w/o PE header = ???



Dump w/o PE header = ???

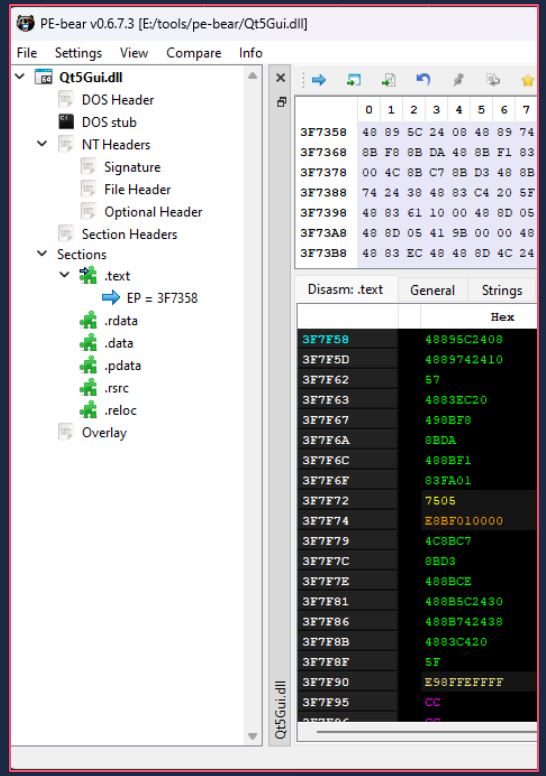


We'll come back to this in the future :)

“ok but I ain’t bout to bust out a hex editor every time”

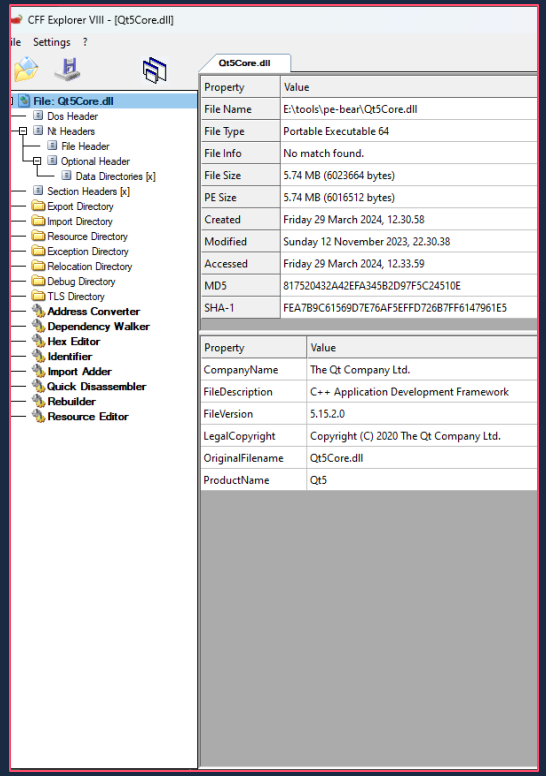
PE Examining Tools

- PE-Bear
  - Developed by hasherezade
  - “PE-bear is a multiplatform reversing tool for PE files. Its objective is to deliver fast and flexible “first view” for malware analysts, stable and capable to handle malformed PE files.”



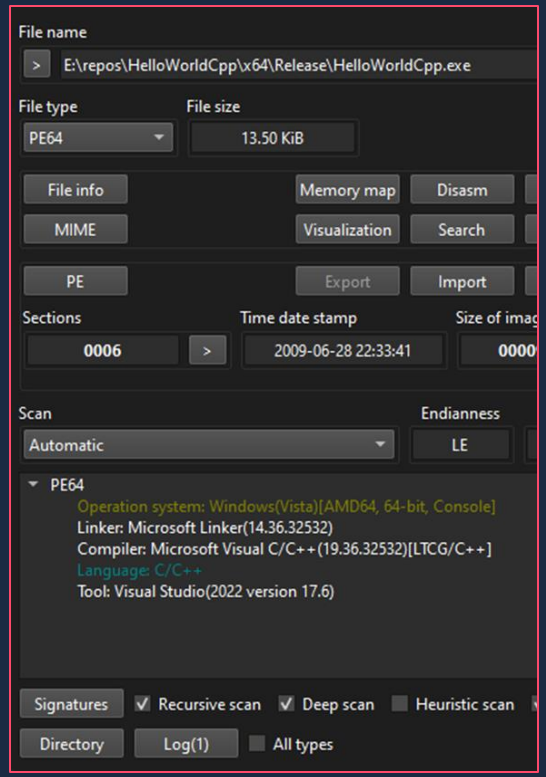
PE Examining Tools

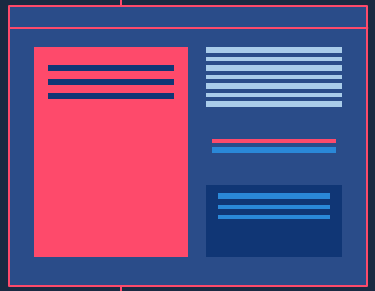
- CFF Explorer
  - Developed by NTCore / Erik Pistelli (@erikpistelli)
  - “CFF Explorer was designed to make PE editing as easy as possible, but without losing sight on the portable executable’s internal structure.”



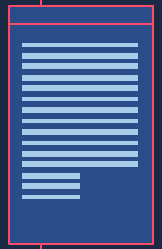
PE Examining Tools

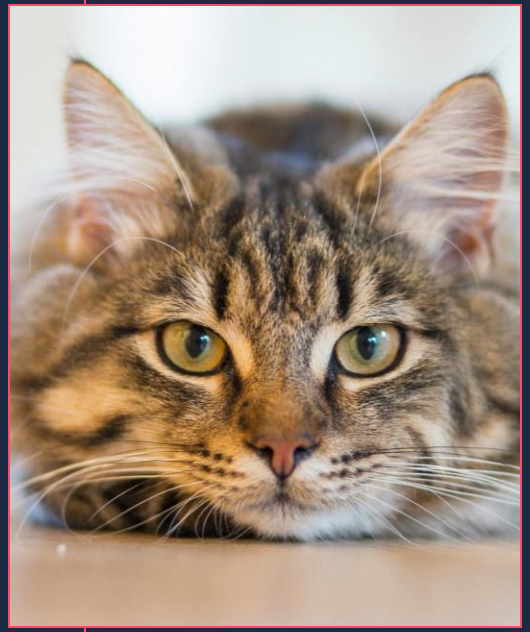
- Detect it Easy
  - Developed by Hors (@horsicq)
  - "DIE" is a cross-platform application, apart from Windows version there are also available versions for Linux and Mac OS.





Break time Q&A





04

Core Windows Security Concepts

Memory Permissions

- Windows & most modern OS have the concept of permissions when it comes with memory, most commonly:
  - Read (R)
  - Write (W)
  - Execute (X)

Memory Permissions

- Here's a region of memory that is set with Read (R) + Execute (X)…

```
mov qword ptr [rsp + 8], rbx
mov qword ptr [rsp + 0x10], rsi
push rdi
sub rsp, 0x20
…
```

Memory Permissions

- Here's a region of memory that is set with Read (R) + Execute (X)…

```
mov qword ptr [rsp + 8], rbx
mov qword ptr [rsp + 0x10], rsi
push rdi
sub rsp, 0x20
…
```

Memory Permissions

- Here's a region of memory that is set with Read (R) + Execute (X)…

```
mov qword ptr [rsp + 8], rbx
mov qword ptr [rsp + 0x10], rsi
push rdi
sub rsp, 0x20
…
```

Memory Permissions

- Here's a region of memory that is set with Read (R) + Execute (X)…

```
mov qword ptr [rsp + 8], rbx
mov qword ptr [rsp + 0x10], rsi
push rdi
sub rsp, 0x20
…
```

Memory Permissions

- Here's the same page but only Read/Write (R/W)…

```
mov qword ptr [rsp + 8], rbx
mov qword ptr [rsp + 0x10], rsi
push rdi
sub rsp, 0x20
…
```

Memory Permissions

- Here's the same page but only Read/Write (R/W)…

```
mov qword ptr [rsp + 8], rbx
mov qword ptr [rsp + 0x10], rsi
push rdi
sub rsp, 0x20
…
```

STATUS_ACCESS_VIOLATION

Data Execution Prevention (DEP)

Memory Permissions

- How allocating a memory works in Windows
  - Developer calls `VirtualAlloc`
  - `VirtualAlloc` -> `NtAllocateVirtualMemory`

Memory Permissions

- How allocating a memory works in Windows
  - Developer calls `VirtualAlloc`
  - `VirtualAlloc` -> `NtAllocateVirtualMemory`

```
LPVOID VirtualAlloc(
  [in, optional] LPVOID lpAddress,
  [in]           SIZE_T dwSize,
  [in]           DWORD  flAllocationType,
  [in]           DWORD  flProtect
);
```

Memory Permissions

```
LPVOID VirtualAlloc(
  [in, optional] LPVOID lpAddress,
  [in]           SIZE_T dwSize,
  [in]           DWORD  flAllocationType,
  [in]           DWORD  flProtect
);
```

Constants

Expand table

Constant/value	Description
PAGE_EXECUTE 0x10	Enables execute access to the committed region of pages. An attempt to write to the committed region results in an access violation. This flag is not supported by the CreateFileMapping function.
PAGE_EXECUTE_READ 0x20	Enables execute or read-only access to the committed region of pages. An attempt to write to the committed region results in an access violation. **Windows Server 2003 and Windows XP:** This attribute is not supported by the CreateFileMapping function until Windows XP with SP2 and Windows Server 2003 with SP1.
PAGE_EXECUTE_READWRITE 0x40	Enables execute, read-only, or read/write access to the committed region of pages. **Windows Server 2003 and Windows XP:** This attribute is not supported by the CreateFileMapping function until Windows XP with SP2 and Windows Server 2003 with SP1.
PAGE_EXECUTE_WRITECOPY 0x80	Enables execute, read-only, or copy-on-write access to a mapped view of a file mapping object. An attempt to write to a committed copy-on-write page results in a private copy of the page being made for the process. The private page is marked as **PAGE_EXECUTE_READWRITE**, and the change is written to the new page. This flag is not supported by the VirtualAlloc or VirtualAllocEx functions. **Windows Vista, Windows Server 2003 and Windows XP:** This attribute is not supported by the CreateFileMapping function until Windows Vista with SP1 and Windows Server 2008.
PAGE_NOACCESS 0x01	Disables all access to the committed region of pages. An attempt to read from, write to, or execute the committed region results in an access violation. This flag is not supported by the CreateFileMapping function.
PAGE_READONLY 0x02	Enables read-only access to the committed region of pages. An attempt to write to the committed region results in an access violation. If Data Execution Prevention is enabled, an attempt to execute code in the committed region results in an access violation.
PAGE_READWRITE 0x04	Enables read-only or read/write access to the committed region of pages. If Data Execution Prevention is enabled, attempting to execute code in the committed region results in an access violation.
PAGE_WRITECOPY 0x08	Enables read-only or copy-on-write access to a mapped view of a file mapping object. An attempt to write to a committed copy-on-write page results in a private copy of the page being made for the process. The private page is marked as **PAGE_READWRITE**, and the change is written to the new page. If Data Execution Prevention is enabled, attempting to execute code in the committed region results in an access violation. This flag is not supported by the VirtualAlloc or VirtualAllocEx functions.

Memory Permissions

- You can also change the memory permission at runtime with `VirtualProtect`

```
BOOL VirtualProtect(
  [in]  LPVOID lpAddress,
  [in]  SIZE_T dwSize,
  [in]  DWORD  flNewProtect,
  [out] PDWORD lpflOldProtect
);
```

Workshop #3

Memory permissions workshop

- `VirtualAlloc` / `VirtualProtect` usage
- Typical shellcode scenario

```c
int main(void) {
    // Based on https://github.com/vyrus001/shellGo/blob/master/main.go
    // Spawns calc via WinExec
    unsigned char shellcode[] = {
        0x50, 0x51, 0x52, 0x53, 0x56, 0x57, 0x55, 0x6A, 0x60, 0x5A, 0x68, 0x
        0x59, 0x48, 0x83, 0xEC, 0x28, 0x65, 0x48, 0x8B, 0x32, 0x48, 0x8B, 0x
        0x10, 0x48, 0xAD, 0x48, 0x8B, 0x30, 0x48, 0x8B, 0x7E, 0x30, 0x03, 0x
        0x28, 0x8B, 0x74, 0x1F, 0x20, 0x48, 0x01, 0xFE, 0x8B, 0x54, 0x1F, 0x
        0x8D, 0x52, 0x02, 0xAD, 0x81, 0x3C, 0x07, 0x57, 0x69, 0x6E, 0x45, 0x
        0x1C, 0x48, 0x01, 0xFE, 0x8B, 0x34, 0xAE, 0x48, 0x01, 0xF7, 0x99, 0x
        0x30, 0x5D, 0x5F, 0x5E, 0x5B, 0x5A, 0x59, 0x58, 0xC3,
    };

    // Allocate a memory region with RWX permissions
    void* exec_mem = VirtualAlloc(0, sizeof(shellcode), MEM_COMMIT | MEM_RES
    if (!exec_mem) {
        printf("Memory allocation failed\n");
        return 1;
    }

    // Copy the shellcode to the allocated memory
    RtlMoveMemory(exec_mem, shellcode, sizeof(shellcode));

    // Cast the allocated memory as a function and execute it
    exec_shellcode(exec_mem);
```

```
int main(void) {
    // Based on https://github.com/vyrus001/shellGo/blob/master/main.go
    // Spawns calc via WinExec
    unsigned char shellcode[] = {
        0x50, 0x51, 0x52, 0x53, 0x56, 0x57, 0x55, 0x6A, 0x60, 0x5A, 0x68, 0x63, 0x61, 0x6C,
0x63, 0x54,
        0x59, 0x48, 0x83, 0xEC, 0x28, 0x65, 0x48, 0x8B, 0x32, 0x48, 0x8B, 0x76, 0x18, 0x48,
0x8B, 0x76,
        0x10, 0x48, 0xAD, 0x48, 0x8B, 0x30, 0x48, 0x8B, 0x7E, 0x30, 0x03, 0x57, 0x3C, 0x8B,
0x5C, 0x17,
        0x28, 0x8B, 0x74, 0x1F, 0x20, 0x48, 0x01, 0xFE, 0x8B, 0x54, 0x1F, 0x24, 0x0F, 0xB7,
0x2C, 0x17,
        0x8D, 0x52, 0x02, 0xAD, 0x81, 0x3C, 0x07, 0x57, 0x69, 0x6E, 0x45, 0x75, 0xEF, 0x8B,
0x74, 0x1F,
        0x1C, 0x48, 0x01, 0xFE, 0x8B, 0x34, 0xAE, 0x48, 0x01, 0xF7, 0x99, 0xFF, 0xD7, 0x48,
0x83, 0xC4,
        0x30, 0x5D, 0x5F, 0x5E, 0x5B, 0x5A, 0x59, 0x58, 0xC3,
    };

    // Allocate a memory region with RWX permissions
    void* exec_mem = VirtualAlloc(0, sizeof(shellcode), MEM_COMMIT | MEM_RESERVE,
PAGE_EXECUTE_READWRITE);
```

Shellcode for calling WinExec(“calc”)

```
    };

    // Allocate a memory region with RWX permissions
    void* exec_mem = VirtualAlloc(0, sizeof(shellcode), MEM_COMMIT | MEM_RESERVE,
PAGE_EXECUTE_READWRITE);
    if (!exec_mem) {
        printf("Memory allocation failed\n");
        return 1;
    }

    // Copy the shellcode to the allocated memory
    RtlMoveMemory(exec_mem, shellcode, sizeof(shellcode));

    // Cast the allocated memory as a function and execute it
    exec_shellcode(exec_mem);

    // Change memory to RW and try to execute again
    DWORD old{};
    VirtualProtect(exec_mem, sizeof(shellcode), PAGE_READWRITE, &old);
    exec_shellcode(exec_mem);

    return 0;
}
```

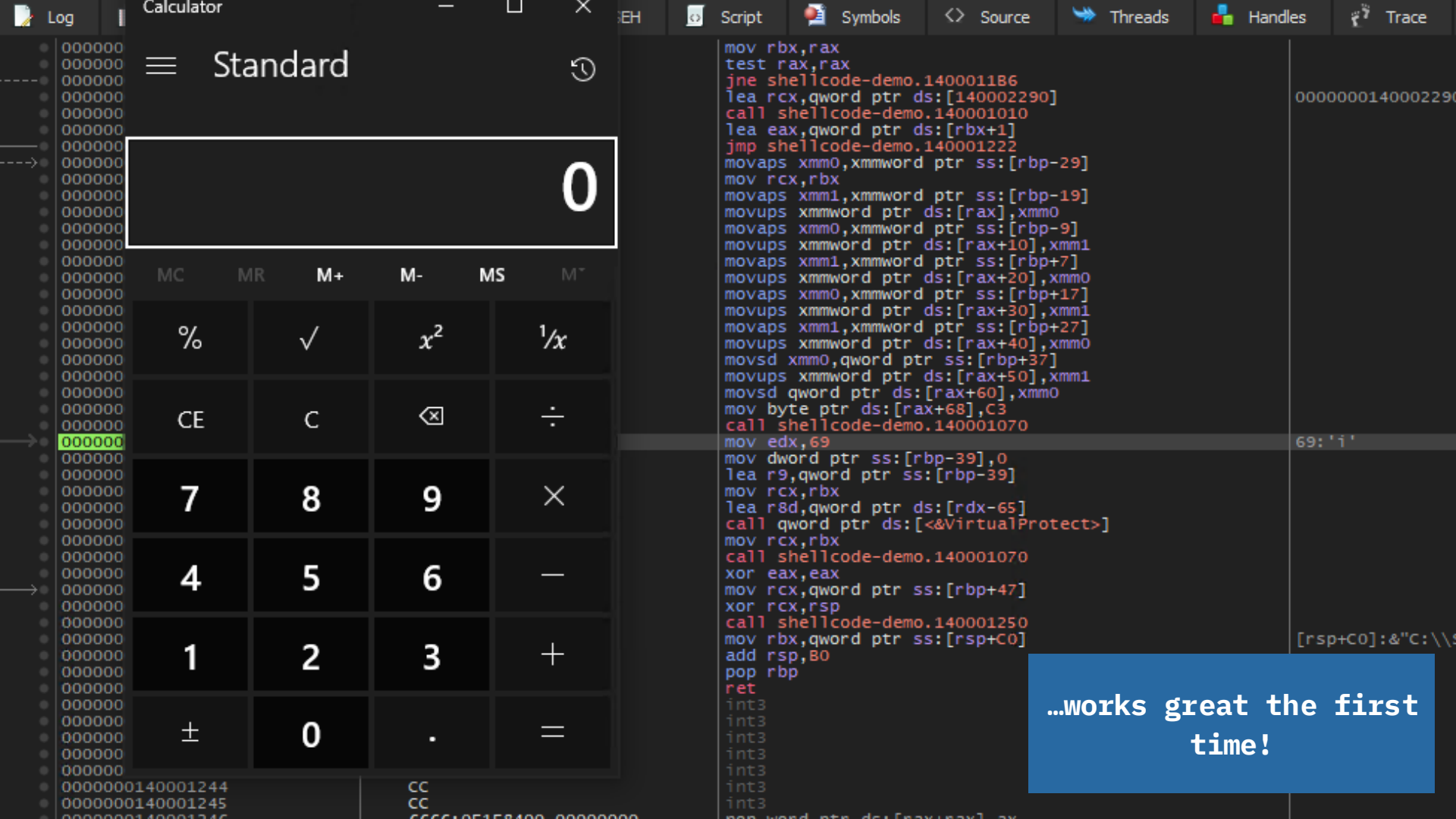
Use VirtualAlloc to allocate a RWX memory

```
void exec_shellcode(void* addr) {
    __try {
        ((void(*)())addr)();
        printf("[+] Executed %p\n", addr);
    }
    __except (1) {
        printf("[-] Exception %08lx\n", GetExceptionCode());
    }
    return;
}

int main(void) {
    // Based on https://github.com/vyrus001/shellGo/blob/master/main.go
    // Spawns calc via WinExec
    unsigned char shellcode[] = {
        0x50, 0x51, 0x52, 0x53, 0x56, 0x57, 0x55, 0x6A, 0x60, 0x5A, 0x68, 0x63, 0x61, 0x6C,
0x63, 0x54,
        0x59, 0x48, 0x83, 0xEC, 0x28, 0x65, 0x48, 0x8B, 0x32, 0x48, 0x8B, 0x76, 0x18, 0x48,
0x8B, 0x76,
        0x10, 0x48, 0xAD, 0x48, 0x8B, 0x30, 0x48, 0x8B, 0x7E, 0x30, 0x03, 0x57, 0x3C, 0x8B,
0x5C, 0x17,
        0x28, 0x8B, 0x74, 0x1F, 0x20, 0x48, 0x01, 0xFE, 0x8B, 0x54, 0x1F, 0x24, 0x0F, 0xB7,
```

Try to execute the shellcode, and if it throws, catch the exception

```
    };

    // Allocate a memory region with RWX permissions
    void* exec_mem = VirtualAlloc(0, sizeof(shellcode), MEM_COMMIT | MEM_RESERVE,
PAGE_EXECUTE_READWRITE);
    if (!exec_mem) {
        printf("Memory allocation failed\n");
        return 1;
    }

    // Copy the shellcode to the allocated memory
    RtlMoveMemory(exec_mem, shellcode, sizeof(shellcode));

    // Cast the allocated memory as a function and execute it
    exec_shellcode(exec_mem);

    // Change memory to RW and try to execute again
    DWORD old{};
    VirtualProtect(exec_mem, sizeof(shellcode), PAGE_READWRITE, &old);
    exec_shellcode(exec_mem);

    return 0;
}
```

Try again by changing the memory permission to RW without the X


Log
Script
Symbols
Source
Threads
Handles
Trace
Calculator
Standard
0
MC MR M+ M- MS M˅
% √ x² 1/x
CE C ⌫ ÷
7 8 9 ×
4 5 6 −
1 2 3 +
± 0 . =
mov rbx,rax
test rax,rax
jne shellcode-demo.140001186
lea rcx,qword ptr ds:[140002290]
call shellcode-demo.140001010
lea eax,qword ptr ds:[rbx+1]
jmp shellcode-demo.140001222
movaps xmm0,xmmword ptr ss:[rbp-29]
mov rcx,rbx
movaps xmm1,xmmword ptr ss:[rbp-19]
movups xmmword ptr ds:[rax],xmm0
movaps xmm0,xmmword ptr ss:[rbp-9]
movups xmmword ptr ds:[rax+10],xmm1
movaps xmm1,xmmword ptr ss:[rbp+7]
movups xmmword ptr ds:[rax+20],xmm0
movaps xmm0,xmmword ptr ss:[rbp+17]
movups xmmword ptr ds:[rax+30],xmm1
movaps xmm1,xmmword ptr ss:[rbp+27]
movups xmmword ptr ds:[rax+40],xmm0
movsd xmm0,qword ptr ss:[rbp+37]
movups xmmword ptr ds:[rax+50],xmm1
movsd qword ptr ds:[rax+60],xmm0
mov byte ptr ds:[rax+68],C3
call shellcode-demo.140001070
mov edx,69
mov dword ptr ss:[rbp-39],0
lea r9,qword ptr ss:[rbp-39]
mov rcx,rbx
lea r8d,qword ptr ds:[rdx-65]
call qword ptr ds:[<&VirtualProtect>]
mov rcx,rbx
call shellcode-demo.140001070
xor eax,eax
mov rcx,qword ptr ss:[rbp+47]
xor rcx,rsp
call shellcode-demo.140001250
mov rbx,qword ptr ss:[rsp+C0]
add rsp,B0
pop rbp
ret
int3
int3
int3
int3
int3
int3
int3
0000000140002290
69:'i'
[rsp+C0]:&"C:\\
0000000140001244 CC
0000000140001245 CC
…works great the first time!

```
7
0
5 37
0
0 60
C3
FFF
000
00000000
C7

9B
D0000

FFF

47

000
4 C0000000
B0000000

F8400 00000000
A92D0000

10
FFFF
```

```
movaps xmm1,xmmword ptr ss:[rbp+27]
movups xmmword ptr ds:[rax+40],xmm0
movsd xmm0,qword ptr ss:[rbp+37]
movups xmmword ptr ds:[rax+50],xmm1
movsd qword ptr ds:[rax+60],xmm0
mov byte ptr ds:[rax+68],C3
call shellcode-demo.140001070
mov edx,69
mov dword ptr ss:[rbp-39],0
lea r9,qword ptr ss:[rbp-39]
mov rcx,rbx
lea r8d,qword ptr ds:[rdx-65]
call qword ptr ds:[<&VirtualProtect>]
mov rcx,rbx
call shellcode-demo.140001070
xor eax,eax
mov rcx,qword ptr ss:[rbp+47]
xor rcx,rsp
call shellcode-demo.140001250
mov rbx,qword ptr ss:[rsp+C0]
add rsp,B0
pop rbp
ret
int3
int3
int3
int3
int3
int3
int3
nop word ptr ds:[rax+rax],ax
cmp rcx,qword ptr ds:[140004000]
jne shellcode-demo.140001269
rol rcx,10
test cx,FFFF
jne shellcode-demo.140001265
```

```
69:'i'
```

```
R12
R13
R14
R15

RIP

RFLA
ZF
OF
CF

Last
Last
```

C:\Space\shellcode-demo.exe

```
[+] Executed 000001604E990000
[-] Exception c0000005
```

…not so great once whe changed the perms to RW!

Takeaway

- `VirtualAlloc` / `VirtualProtect` with RWX are almost always a red flag when dealing with *known* malware; great for placing breakpoint.

```
int main(void) {
    // Based on https://github.com/vyrus001/shellGo/blob/master/main.go
    // Spawns calc via WinExec
    unsigned char shellcode[] = {
        0x50, 0x51, 0x52, 0x53, 0x56, 0x57, 0x55, 0x6A, 0x60, 0x5A, 0x68, 0x
        0x59, 0x48, 0x83, 0xEC, 0x28, 0x65, 0x48, 0x8B, 0x32, 0x48, 0x8B, 0x
        0x10, 0x48, 0xAD, 0x48, 0x8B, 0x30, 0x48, 0x8B, 0x7E, 0x30, 0x03, 0x
        0x28, 0x8B, 0x74, 0x1F, 0x20, 0x48, 0x01, 0xFE, 0x8B, 0x54, 0x1F, 0x
        0x8D, 0x52, 0x02, 0xAD, 0x81, 0x3C, 0x07, 0x57, 0x69, 0x6E, 0x45, 0x
        0x1C, 0x48, 0x01, 0xFE, 0x8B, 0x34, 0xAE, 0x48, 0x01, 0xF7, 0x99, 0x
        0x30, 0x5D, 0x5F, 0x5E, 0x5B, 0x5A, 0x59, 0x58, 0xC3,
    };

    // Allocate a memory region with RWX permissions
    void* exec_mem = VirtualAlloc(0, sizeof(shellcode), MEM_COMMIT | MEM_RES
    if (!exec_mem) {
        printf("Memory allocation failed\n");
        return 1;
    }

    // Copy the shellcode to the allocated memory
    RtlMoveMemory(exec_mem, shellcode, sizeof(shellcode));

    // Cast the allocated memory as a function and execute it
    exec_shellcode(exec_mem);
```

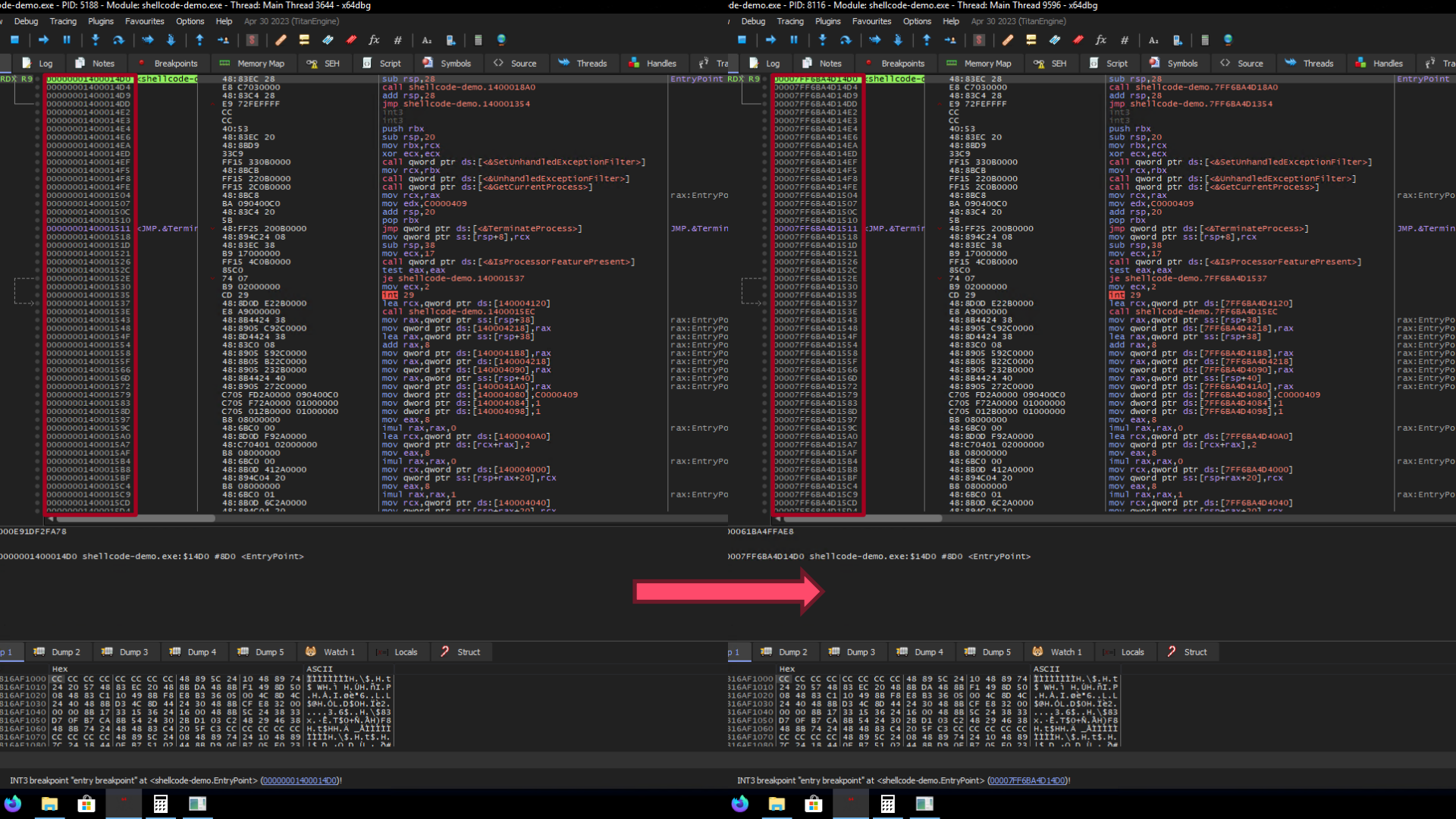
Takeaway

- `VirtualAlloc` / `VirtualProtect` with RWX are almost always a red flag when dealing with *known* malware; great for placing breakpoint.
- But not always - there are legitimate uses that involve RWX memory!
  - Dynamic code generation/JIT
  - Custom memory allocators

```
int main(void) {
    // Based on https://github.com/vyrus001/shellGo/blob/master/main.go
    // Spawns calc via WinExec
    unsigned char shellcode[] = {
        0x50, 0x51, 0x52, 0x53, 0x56, 0x57, 0x55, 0x6A, 0x60, 0x5A, 0x68, 0x
        0x59, 0x48, 0x83, 0xEC, 0x28, 0x65, 0x48, 0x8B, 0x32, 0x48, 0x8B, 0x
        0x10, 0x48, 0xAD, 0x48, 0x8B, 0x30, 0x48, 0x8B, 0x7E, 0x30, 0x03, 0x
        0x28, 0x8B, 0x74, 0x1F, 0x20, 0x48, 0x01, 0xFE, 0x8B, 0x54, 0x1F, 0x
        0x8D, 0x52, 0x02, 0xAD, 0x81, 0x3C, 0x07, 0x57, 0x69, 0x6E, 0x45, 0x
        0x1C, 0x48, 0x01, 0xFE, 0x8B, 0x34, 0xAE, 0x48, 0x01, 0xF7, 0x99, 0x
        0x30, 0x5D, 0x5F, 0x5E, 0x5B, 0x5A, 0x59, 0x58, 0xC3,
    };

    // Allocate a memory region with RWX permissions
    void* exec_mem = VirtualAlloc(0, sizeof(shellcode), MEM_COMMIT | MEM_RES
    if (!exec_mem) {
        printf("Memory allocation failed\n");
        return 1;
    }

    // Copy the shellcode to the allocated memory
    RtlMoveMemory(exec_mem, shellcode, sizeof(shellcode));

    // Cast the allocated memory as a function and execute it
    exec_shellcode(exec_mem);
```

ASLR

- ASLR = Address Space Layout Randomization
- ASLR randomizes memory location for programs.
- Protects against exploits by making memory addresses unpredictable.


de-demo.exe - PID: 5188 - Module: shellcode-demo.exe - Thread: Main Thread 3644 - x64dbg
Debug Tracing Plugins Favourites Options Help Apr 30 2023 (TitanEngine)
Log Notes Breakpoints Memory Map SEH Script Symbols Source Threads Handles
shellcode-demo.exe:$14D0 #8D0 <EntryPoint>
INT3 breakpoint "entry breakpoint" at <shellcode-demo.EntryPoint> (0000000140001 4D0)!
de-demo.exe - PID: 8116 - Module: shellcode-demo.exe - Thread: Main Thread 9596 - x64dbg
Debug Tracing Plugins Favourites Options Help Apr 30 2023 (TitanEngine)
Log Notes Breakpoints Memory Map SEH Script Symbols Source Threads Handles
shellcode-demo.exe:$14D0 #8D0 <EntryPoint>
INT3 breakpoint "entry breakpoint" at <shellcode-demo.EntryPoint> (00007FF6BA4D14D0)!

ASLR on Windows

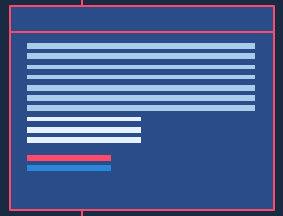
- (Usually) Controlled on a per-application basis
  - Based on the `DYNAMICBASE` flag in the PE header

ASLR on Windows

- However, this is not always true:
  - ASLR can be forcibly applied starting from Windows 8 via Mandatory ASLR in Windows Security options (may cause issues for applications that aren't designed to be ASLR aware).
  - ASLR is mandatory for Windows on ARM as the concept of ASLR is always applied on this architecture (allegedly).

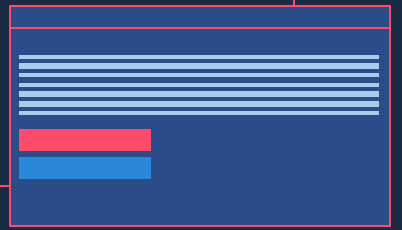
Why mention ASLR?

- It's a "good-to-know" knowledge.
- Less confusing when you see address jumping everywhere whenever you debug something after restarting the app/computer.
- You can turn off ASLR in most modern Windows debuggers these days (x64dbg/IDA).



…and here’s where the abrupt end takes place!

I’ve run out of time preparing the materials.



What now?

Further Reading

Windows Internals 7th. Ed. / ired.team / vxunderground papers are all great starting points.

Feedback

It’s my first time doing this type of stream! Tell me what you like and don’t like or mistakes I’ve made!

Thanks !

azaka.fun

@AzakaSekai_