

REVERSE ENGINEERING – CLASS 0x03

THE STRUCTURE OF PE FILES

Cristian Rusu

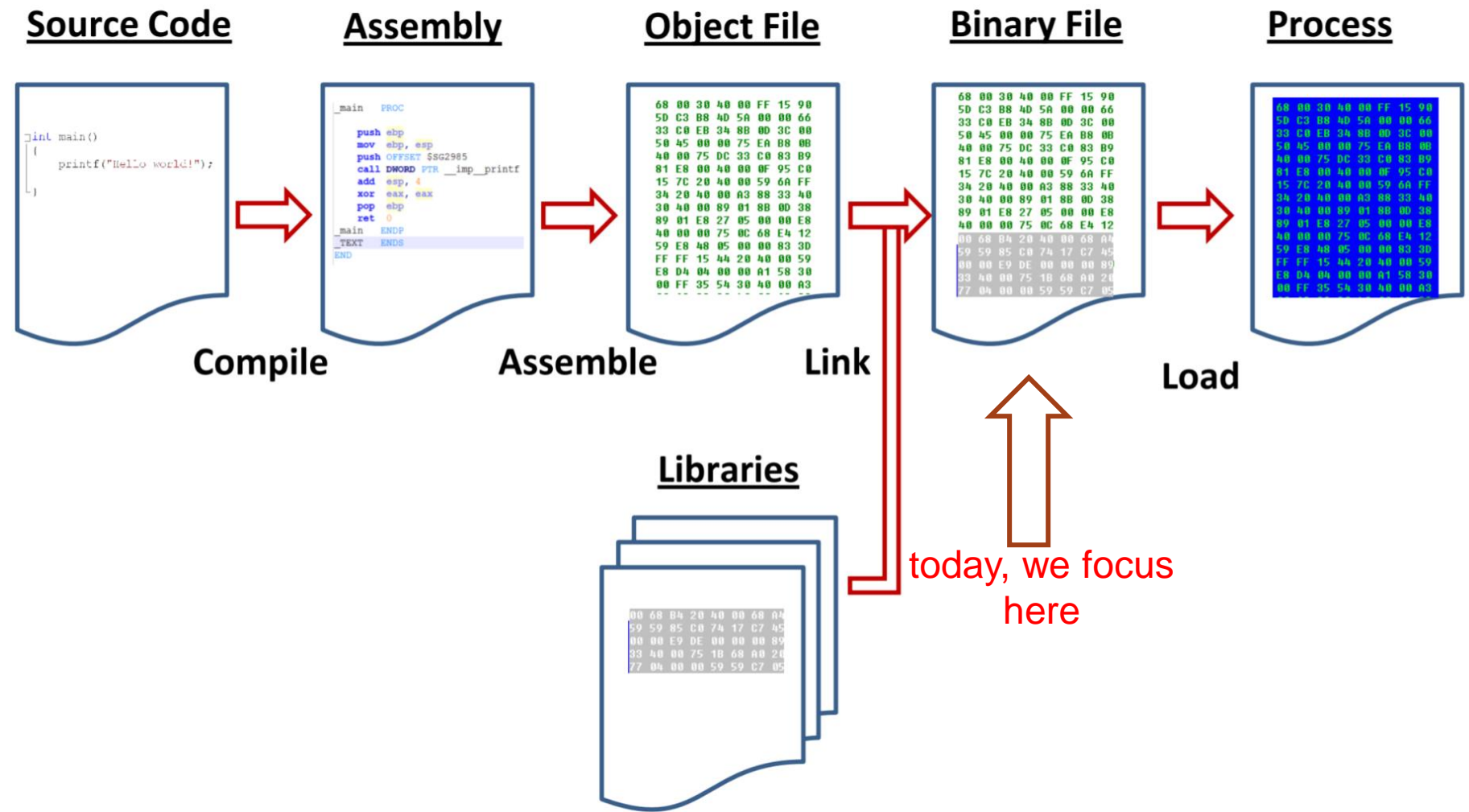
LAST TIME

- **assembly in context**
- **the structure of binary files**
- **study of the ELF binaries**
- **PE for next week**

TODAY

- the structure of binary files
- study of the PE binaries

FROM SOURCE CODE TO EXECUTION



BINARY FILES

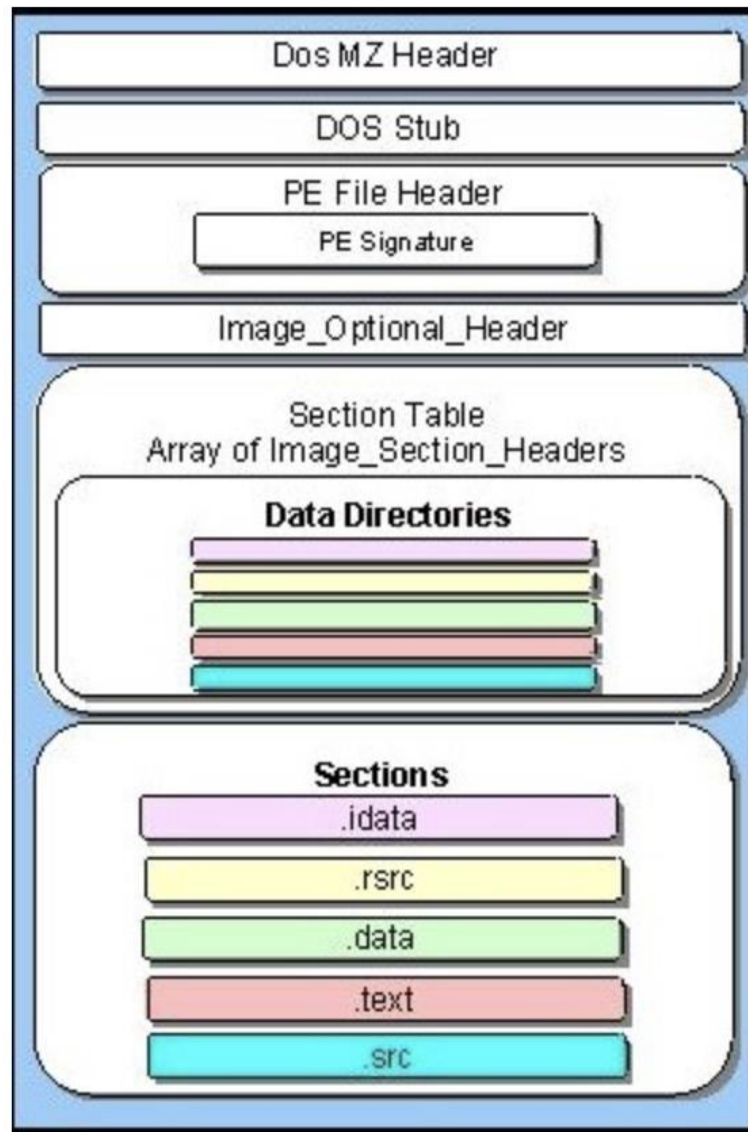
- **ELF/SO**
- **PE/DLL**
- **WASM**
- **machine code (assembly translated to CPU readable instructions) is only part of the executable**
- **all of them have some particular structure we need to understand to in order to execute the binary (ABI)**

PE BINARY

- **Portable Executable**
- **for both Windows x86 and x64**

PE BINARY

- headers & sections



PE BINARY

- **DOS header**
 - first 64 bytes of binary
 - MZ (magic number, just as .ELF)
 - offset to the start of the PE

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


PE BINARY

- **DOS header**
- **DOS stub**
 - „This program cannot be run in DOS mode”
- **PE file header**
 - Signature
 - PE followed by two zeros
 - Machines
 - Target system: intel, AMD, etc.
 - Number of sections
 - Size of the section table
 - Size of optional header, contains information about: binary, such as initial stack size, program entry point location, preferred base address, operating system version, section alignment information

PE BINARY

- dissassembly

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f		
00000000h:	4D	5A	50	00	02	00	00	00	04	00	0F	00	FF	FF	00	00	: MZP.....99..	
00000010h:	88	00	00	00	00	00	00	00	40	00	1A	00	00	00	00	00	:0.....	DOS
00000020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	:	HEADER
00000030h:	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00	00	:	
00000040h:	BA	10	00	0E	1F	B4	09	CD	21	B8	01	4C	CD	21	90	90	: *.i!..Li!00	
00000050h:	54	68	69	73	20	70	72	6F	67	72	61	6D	20	6D	75	73	: This program mus	DOS
00000060h:	74	20	62	65	20	72	75	6E	20	75	6E	64	65	72	20	57	: t be run under W	STUB
00000070h:	69	6E	33	32	0D	0A	24	37	00	00	00	00	00	00	00	00	: in32...47.....	
00000080h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	:	
00000090h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	:	
000000a0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	:	
000000b0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	:	
000000c0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	:	
000000d0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	:	
000000e0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	:	
000000f0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	:	
00000100h:	50	45	00	00	4C	01	08	00	19	5E	42	2A	00	00	00	00	: PE..L.....^B*....	PE
00000110h:	00	00	00	00	E0	00	8E	81	0B	01	02	19	00	A0	02	00	:a..20.....	HEADER
00000120h:	00	D2	00	00	00	00	00	00	B4	AD	02	00	00	10	00	00	: .p.....'-.....	
00000130h:	00	B0	02	00	00	00	40	00	00	10	00	00	00	02	00	00	: .*....0.....	Signature
00000140h:	01	00	00	00	00	00	00	00	04	00	00	00	00	00	00	00	:	
00000150h:	00	D0	03	00	00	04	00	00	00	00	00	00	02	00	00	00	: .D.....	FileHeader
00000160h:	00	00	10	00	00	40	00	00	00	10	00	00	10	00	00	00	:0.....	
00000170h:	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	00	:	OptionalHeader
00000180h:	00	D0	02	00	1E	18	00	00	00	40	03	00	00	8E	00	00	: .D.....0...2..	
00000190h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	:	
000001a0h:	00	10	03	00	04	2B	00	00	00	00	00	00	00	00	00	00	:+.....	DATA
000001b0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	:	DIRECTORY
000001c0h:	00	00	03	00	18	00	00	00	00	00	00	00	00	00	00	00	:	
000001d0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	:	
000001e0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	:	
000001f0h:	00	00	00	00	00	00	00	00	43	4F	44	45	00	00	00	00	:CODE....	
00000200h:	88	9E	02	00	00	10	00	00	00	A0	02	00	00	04	00	00	: "2.....	
00000210h:	00	00	00	00	00	00	00	00	00	00	00	00	20	00	00	60	:	SECTION
00000220h:	44	41	54	41	00	00	00	00	D4	06	00	00	00	B0	02	00	: DATA....ô....*..	TABLE

PE BINARY

- **tools for PE analysis**
- **PE Studio**
 - a utility for inspecting PE formatted binaries such as windows EXEs and DLLs
- **CFF Explorer**
 - a freeware suite of tools including a PE editor and a process viewer
- **PE bear**
 - 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

BINARY ANALYSIS

- **general tools**
- **Ghidra**
 - Open-sourced NSA tool
 - Pro: free and hackable
 - Pro: decompiles anything it can disassemble
 - Con: looks horrible (UI/UX skills zero)
 - Con: sometimes the decompilation is hard/impossible to follow
 - Prefers gotos (no for loop support)
- **IDA**
 - Swiss army knife of Reverse Engineering
 - Pro: Tried and tested
 - Pro: Analyze most executable file formats
 - Pro: Disassemble most architectures (x86, arm, mips, z80, etc)
 - Pro: Decompile some architectures (x86/amd64, arm/arm64, ppc/ppc64, mips32)
 - Con: Too expensive
 - Con: Piracy is rampant

IDA SHOWCASE

- go from machine code back to source code (ideally)

The screenshot displays the IDA Pro interface with the following components:

- BINARY:** A hex dump of the binary data at the top of the window.
- IDA:** The disassembler view on the left, showing assembly instructions. Key instructions include:
 - `loc_804BCC7: ; CODE XREF: sub_804BB10+A42j`
 - `mov [esp+28h+var_24], offset aUnzip ;`
 - `xor eax, eax`
 - `test esi, esi`
 - `setnz al`
 - `mov edx, 1`
 - `mov ds:dword_804FBAC, edx`
 - `lea eax, [eax+eax+1]`
 - `mov ds:dword_804F780, eax`
 - `mov eax, ds:dword_804FFD4`
 - `mov [esp+28h+var_28], eax`
 - `call _strstr`
 - `test eax, eax`
 - `jz loc_804C4F1`
- DECOMPILER:** The decompiler view on the right, showing the corresponding C-like code:
 - `DWORD_804F780 = 2 * (v9 != 0) + 1;`
 - `if (strstr(DWORD_804FFD4, "unzip") || strstr(DWORD_804FFD4, "UNZIP"))`
 - `DWORD_804FBAC = 2;`
 - `if (strstr(DWORD_804FFD4, "z2cat") || strstr(DWORD_804FFD4, "ZCAT") || strstr(DWORD_804FFD4, "zcat") || strstr(DWORD_804FFD4, "ZCAT"))`
 - `{`
 - `DWORD_804FBAC = 2;`
 - `DWORD_804F780 = (v9 != 0) + 1;`
 - `}`

WHAT WE DID TODAY

- **PE binaries**
 - DOS/PE structure
- **general static binary analysis tools**
 - Gidra
 - IDA (las session today)

NEXT TIME ...

- **Dynamic analysis & reverse engineering**

REFERENCES

- **Decompiler explorer**
 - <https://dogbolt.org/>
- **PE insights**
 - https://en.wikibooks.org/wiki/X86_Disassembly/Windows_Executable_Files
 - <https://resources.infosecinstitute.com/topic/2-malware-researchers-handbook-demystifying-pe-file/>
- **Introduction to IDA pro**
 - <https://www.youtube.com/watch?v=qCQRKLaz2nQ>
- **Intro to RE with IDA on Pes**
 - <https://www.youtube.com/watch?v=1MotMBPX7tY>

