Malware Analysis

Chapter 4: A Crash Course in x86 Disassembly

王志
zwang@nankai.edu.cn

updated on Oct. 17th 2021

College of Cyber Science
Nankai University
2021/2022

Agenda

- Levels of Abstraction
- Reverse Engineering
- The x86 Architecture
- Simple Instructions

Basic Techniques

- Basic Static Analysis
  - Looks at malware from the outside
- Basic Dynamic Analysis
  - Only shows you how the malware operates in one case
- Disassembly
  - View code of malware and figure out what it does

Levels of Abstraction

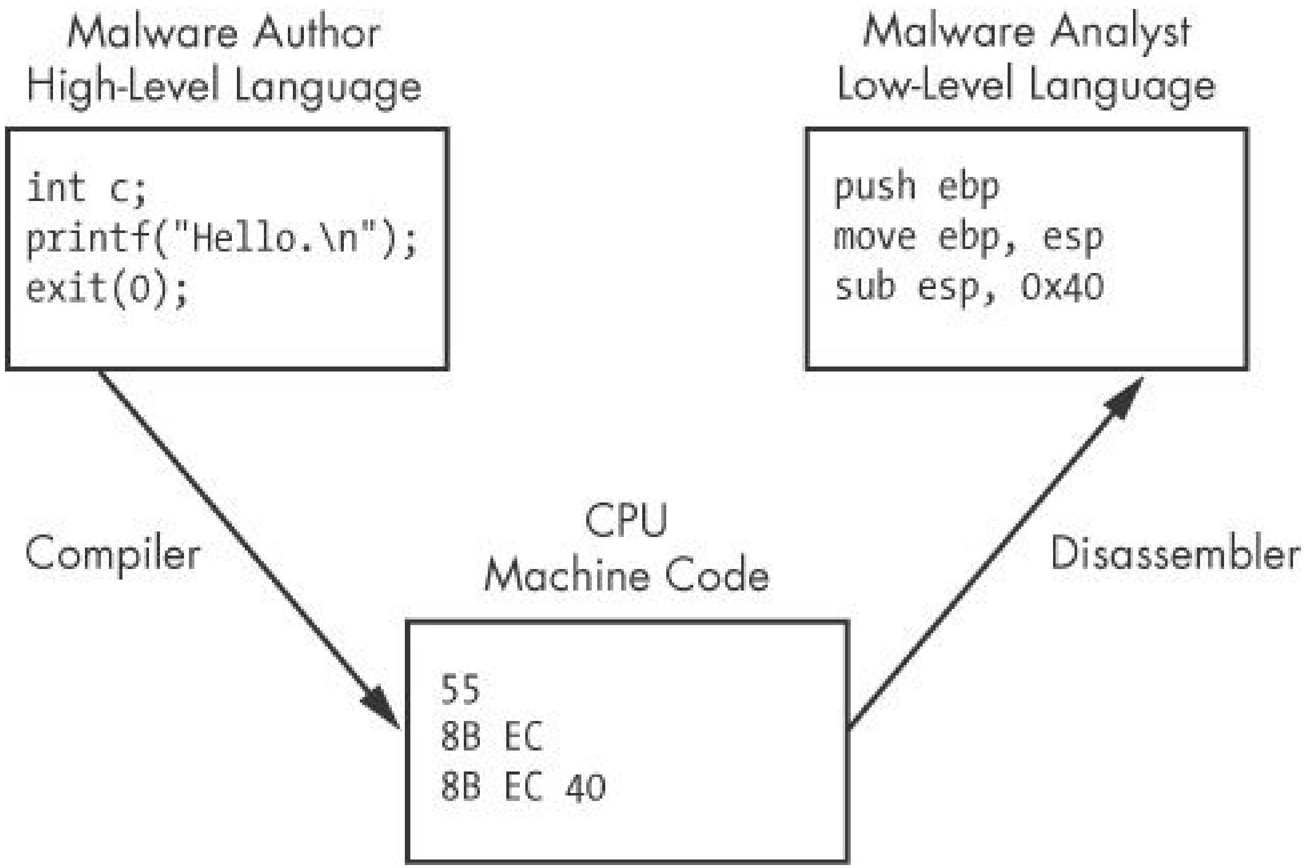

Figure 5-1. Code level examples

Six Levels of Abstraction

- Hardware
- Microcode
- Machine code
- Low-level languages
- High-level languages
- Interpreted languages

Hardware

- Digital circuits
- XOR, AND, OR, NOT gates
- Cannot be easily manipulated by software

Microcode

- Also called firmware
- Only operates on specific hardware it was designed for
- Not usually important for malware analysis

Machine Code

- **Opcodes**
  - Tell the processor to do something
  - Created when a program written in a high-level language is compiled

Low-level Languages

- Human-readable version of processor's instruction set
- Assembly language
  - PUSH, POP, NOP, MOV, JMP ...
- Disassembler generates assembly language
- This is the highest level language that can be reliably recovered from malware when source code is unavailable

High-level Languages

- Most programmers use these
  - C, C++, etc.
  - Converted to machine code by a **compiler**

Interpreted Languages

- Highest level
  - Java, C#, Perl, .NET, Python
- Code is not compiled into machine code
- It is translated into bytecode
  - An intermediate representation
  - Independent of hardware and OS
  - Bytecode executes in an interpreter, which translates bytecode into machine language on the fly at runtime
  - Ex: Java Virtual Machine

Reverse Engineering

Disassembly

- Malware on a disk is in binary form at the machine code level
- Disassembly converts the binary form to assembly language
- **IDA Pro** is the most popular disassembler

Assembly Language

- Different versions for each type of processor
  - x86 – 32-bit Intel (most common)
  - x64 – 64-bit Intel
  - SPARC, PowerPC, MIPS, ARM – others
  - Windows runs on x86 or x64
  - x64 machines can run x86 programs
- Most malware is designed for **x86**

填空题 20分

设置

Disassembly converts the [填空1] to [填空2]

1. binary form
2. assembly language

正常使用填空题需3.0以上版本雨课堂

作答

The x86 Architecture

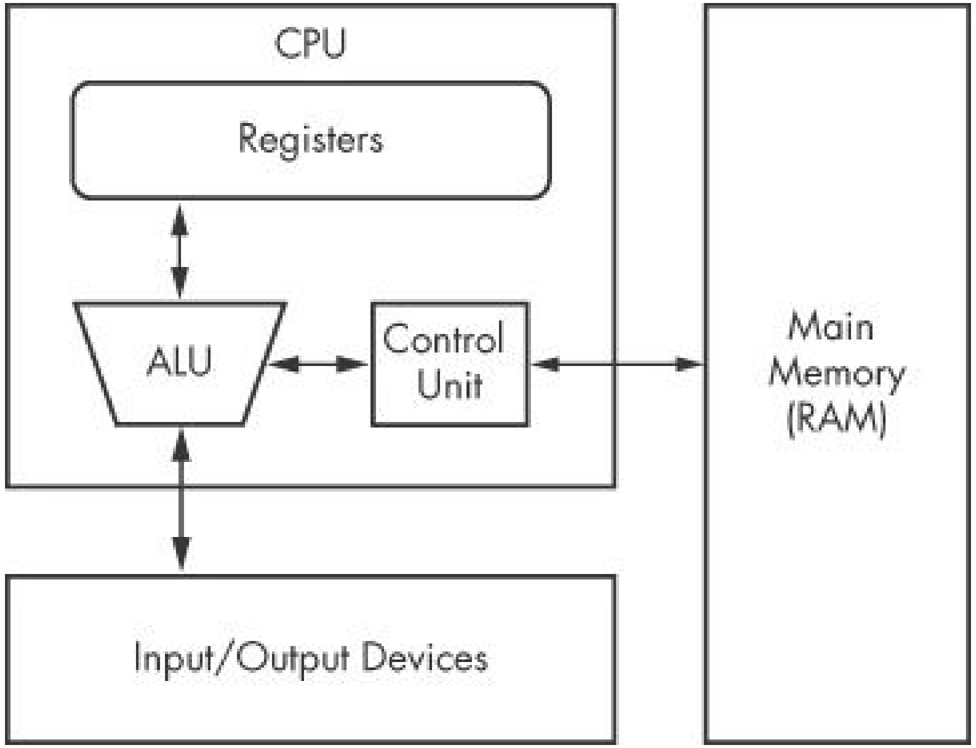

Figure 5-2. Von Neumann architecture

CPU Components

- Control unit
  - Fetches instructions from RAM using a register named the instruction pointer
- Registers
  - Data storage within the CPU
  - Faster than RAM
- ALU (Arithmetic Logic Unit)
  - Executes an instruction and places results in registers or RAM

Main Memory

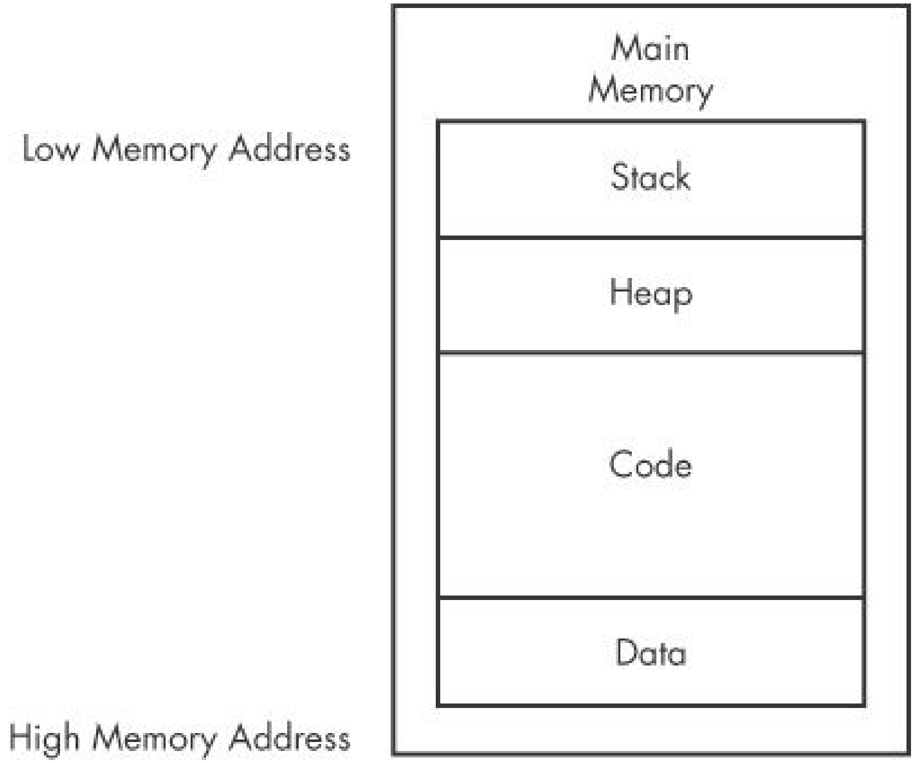

Figure 5-3. Basic memory layout for a program

Data

- Values placed in RAM when a program loads
- These values are static
  - They cannot change while the program is running
- They are also global
  - Available to any part of the program

Code

- Instructions for the CPU
- Controls what the program does

Heap

- Dynamic memory
- Changes frequently during program execution
- Program allocates new values, and frees them when they are no longer needed

Stack

- Local variables and parameters for functions
- Helps programs flow

Instructions

- Mnemonic followed by operands
- mov ecx 0x42
  - Move into Extended C register the value 42(hex)
- In binary this instruction is
  - B9 42 00 00 00

Opcode

- Tell the CPU what operation the programmer wants to perform
- Opcode the machine instruction
  - opcode + oprand
- Disassembler

Endianness

- Big-Endian
  - Most significant byte with lowest memory address
  - 0x12345678 as a 32-bit value would be 0x12345678
- Little-Endian
  - Least significant byte with lowest memory address
  - 0x12345678 as a 32-bit value would be 0x78563412

Endianness

- Network data use big-endian
- x86 programs use little-endian

IP Addresses

- 127.0.0.1, or in hex, 7F 00 00 01
- Sent over the network as 0x7F000001
- Stored in RAM as 0x0100007F
- mov eax, [address of IP in RAM]
  - what's the value in eax?

Operands

- Immediate
  - Fixed values like –x42
- Register
  - eax, ebx, ecx, and so on
- Memory address
  - Denoted with brackets, like [eax]

Registers

Table 5-3. The x86 Registers

General registers	Segment registers	Status register	Instruction pointer
EAX (AX, AH, AL)	CS	EFLAGS	EIP
EBX (BX, BH, BL)	SS		
ECX (CX, CH, CL)	DS		
EDX (DX, DH, DL)	ES		
EBP (BP)	FS		
ESP (SP)	GS		
ESI (SI)			

General Registers

- EAX extended accumulator register
- EBX extended base register
- ECX extended counter register
- EDX extended data register
- EBP extended basic pointer
- ESP extended stack pointer
- ESI extended source index
- EDI extended destination index

Registers

- General registers
  - Used by the CPU during execution
- Segment registers
  - Used to track sections of memory
- Status flags
  - Used to make decisions
- Instruction pointer
  - Address of next instruction to execute

Size of Registers

- General registers are all 32 bits in size
  - Can be referenced as either 32bits (edx) or 16 bits (dx)
- Four registers (eax, ebx, ecx, edx) can also be referenced as 8-bit values
  - AL is lowest 8 bits
  - AH is higher 8 bits

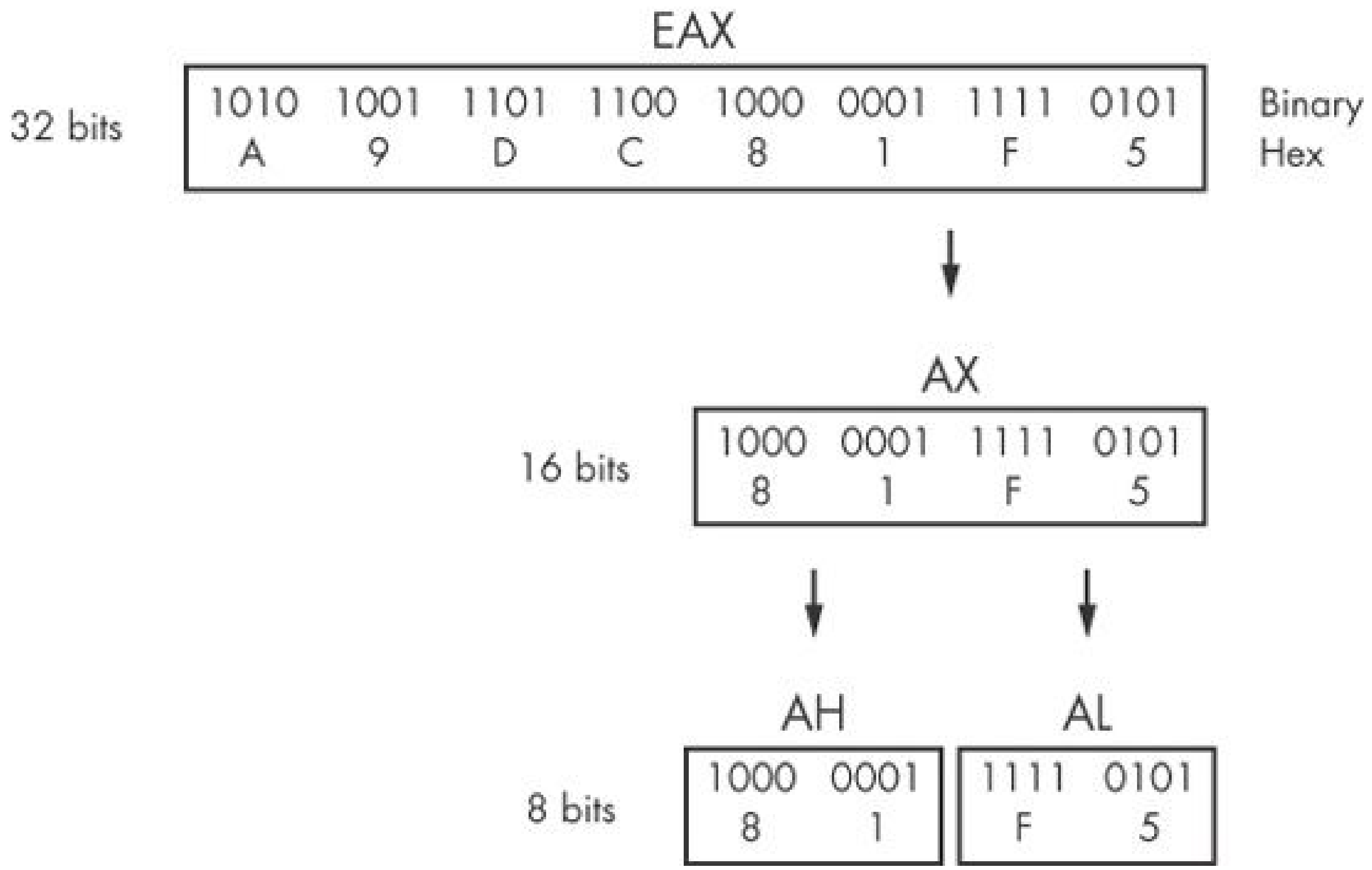

Figure 5-4. x86 EAX register breakdown

General Registers

- Typically store data or memory addresses
- Some instructions reference specific registers
  - Multiplication and division use EAX and EDX
- Conventions
  - Compilers use registers in consistent ways
  - EAX contains the return value for function calls

Flags

- EFLAGS is a status register
- 32 bits in size
- Each bit is a flag
- SET (1) or Cleared (0)

- ZF Zero flag
  - Set when the result of an operation is zero
- CF Carry flag
  - Set when result is too large or small for destination
- SF Sign Flag
  - Set when result is negative, or when most significant bit is set after arithmetic
- TF Trap Flag
  - Used for debugging—if set, processor executes only one instruction at a time

EIP

- Contains the memory address of the next instruction to execute
- If EIP contains wrong data, the CPU will fetch non-legitimate instructions and crash
- Buffer overflows target EIP

Simple Instructions

Simple Instructions

- mov destination, source
    - Moves data from one location to another
- We use Intel format throughout the book, with destination first
- Remember indirect addressing
    - [ebx] means the memory location pointed to by EBX

Table 5-4. `mov` Instruction Examples

Instruction	Description
`mov eax, ebx`	Copies the contents of EBX into the EAX register
`mov eax, 0x42`	Copies the value 0x42 into the EAX register
`mov eax, [0x4037C4]`	Copies the 4 bytes at the memory location 0x4037C4 into the EAX register
`mov eax, [ebx]`	Copies the 4 bytes at the memory location specified by the EBX register into the EAX register
`mov eax, [ebx+esi*4]`	Copies the 4 bytes at the memory location specified by the result of the equation `ebx+esi*4` into the EAX register

lea (Load Effective Address)

- lea destination, source
- lea eax, [ebx+8]
  - Puts ebx + 8 into eax
- Compare
  - mov eax, [ebx+8]
  - Moves the data at location ebx+8 into eax

Figure 5-5 shows values for registers EAX and EBX on the left and the information contained in memory on the right. EBX is set to 0xB30040. At address 0xB30048 is the value `0x20`. The instruction `mov eax, [ebx+8]` places the value `0x20` (obtained from memory) into EAX, and the instruction `lea eax, [ebx+8]` places the value `0xB30048` into EAX.

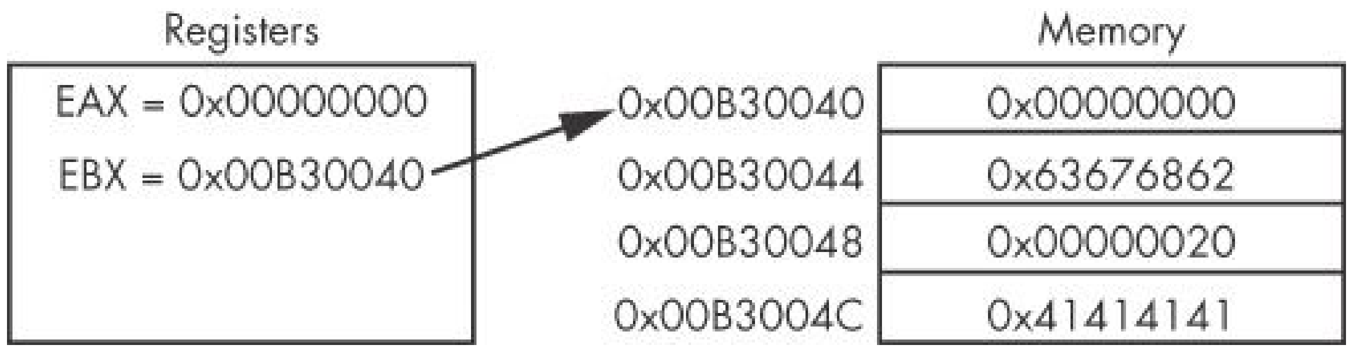

Figure 5-5. EBX register used to access memory

Arithmetic

- sub Subtracts
- add Adds
- inc Increments
- dec Decrements
- mul Multiplies
- div Divides

NOP

- Does nothing (no-operation)
- 0x90
- Commonly used as a NOP Sled
- Allows attackers to run code even if they are imprecise about jumping to it

The Stack

- Memory for functions, local variables, and flow control
- Last in, First out
  - PUSH puts data on the stack
  - POP takes data off the stack

The Stack

- ESP (Extended Stack Pointer) – top of stack
- EBP (Extended Base Pointer) – bottom of stack

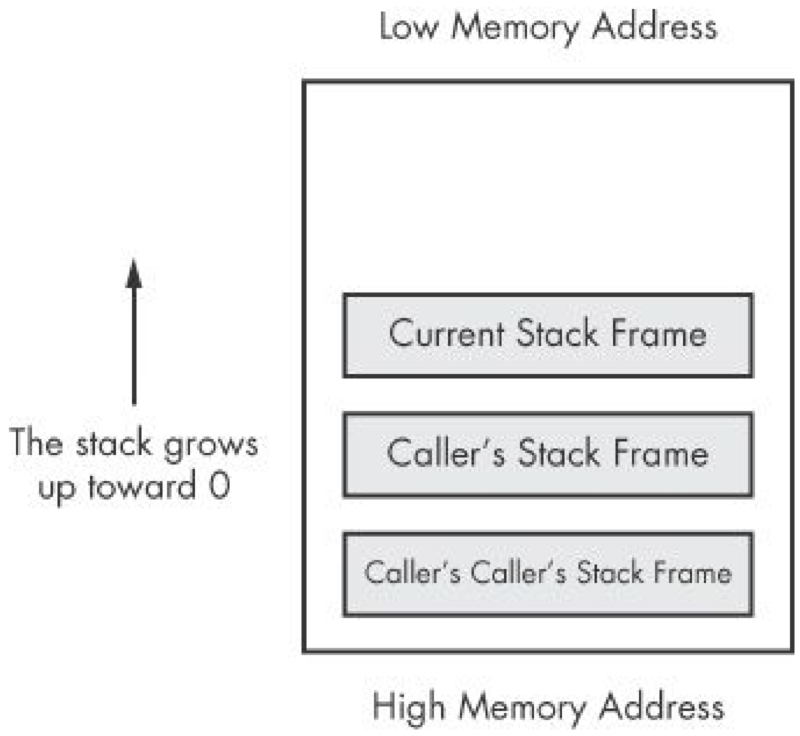

Figure 5-7. x86 stack layout

Other Stack Instructions

- All used with functions
  - Call
  - Leave
  - Enter
  - Ret

Function Calls

- Small programs that do one thing and return, like printf()
- Prologue
  - Instructions at the start of a function that prepare stack and registers for the function to use
- Epilogue
  - Instructions at the end of a function that restore the stack and registers to their original state

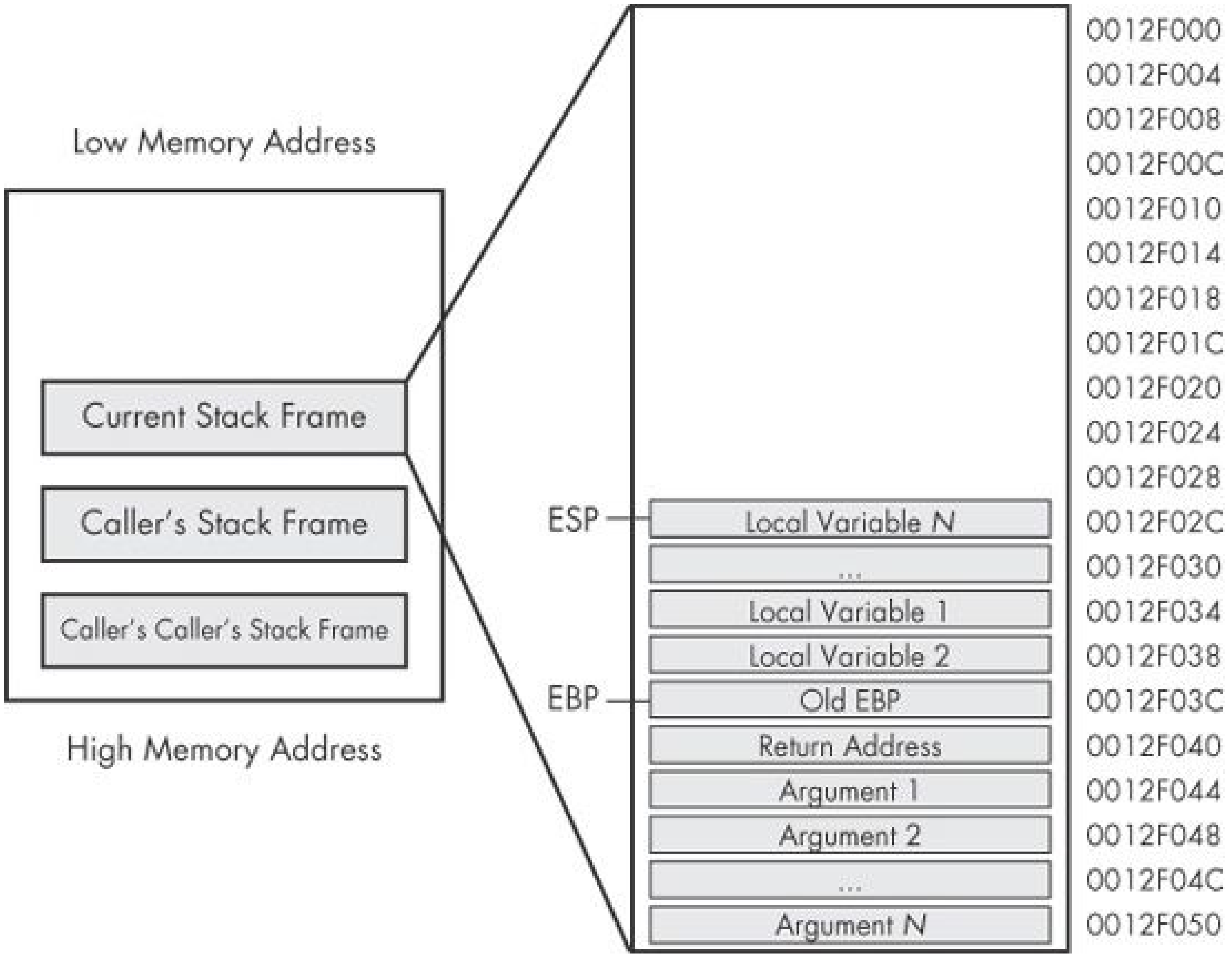

Figure 5-8. Individual stack frame

```
.text:004010FC ; int __stdcall sub_4010FC(LPCSTR lpName)
.text:004010FC sub_4010FC      proc near               ; CODE
.text:004010FC
.text:004010FC TokenHandle     = dword ptr -18h
.text:004010FC var_14          = dword ptr -14h
.text:004010FC NewState        = _TOKEN_PRIVILEGES ptr -10h
.text:004010FC lpName          = dword ptr  8
.text:004010FC
.text:004010FC                 push    ebp
.text:004010FD                 mov     ebp, esp
```

Conditional

- test
  - Compares two values the way AND does, but does not alter them
  - test eax, eax
    - Sets Zero Flag if eax is zero
- cmp eax, ebx
  - Sets Zero Flag if the arguments are equal

Rep Instructions

- Manipulate Data Buffer
  - Intel refers rep instruction as string instruction.
- rep: repeat until ECX=0
- repe, repz : repeat until ECX=0 or ZF=0
- repne, repnz : repeat until ECX=0 or ZF=1

Instruction	Description
`repe cmpsb`	Used to compare two data buffers. EDI and ESI must be set to the two buffer locations, and ECX must be set to the buffer length. The comparison will continue until ECX = 0 or the buffers are not equal.
`rep stosb`	Used to initialize all bytes of a buffer to a certain value. EDI will contain the buffer location, and AL must contain the initialization value. This instruction is often seen used with `xor eax, eax`.
`rep movsb`	Typically used to copy a buffer of bytes. ESI must be set to the source buffer address, EDI must be set to the destination buffer address, and ECX must contain the length to copy. Byte-by-byte copy will continue until ECX = 0.
`repne scasb`	Used for searching a data buffer for a single byte. EDI must contain the address of the buffer, AL must contain the byte you are looking for, and ECX must be set to the buffer length. The comparison will continue until ECX = 0 or until the byte is found.

Branching

- jz loc
  - Jump to loc if the Zero Flag is set
- jnz loc
  - Jump to loc if the Zero Flag is cleared
- Signed comparison jump
  - jg, jge, jl, jle
- Unsigned Comparison jump
  - ja, jae, jb, jbe

C Main Method

- Every C program has a main() function
- int main(int argc, char** argv)
  - **largc** contains the number of arguments on the command line
  - **argv** is a pointer to an array of names containing the arguments

Example

- cp foo bar
  - argc = 3
  - argv[0] = cp
  - argv[1] = foo
  - argv[2] = bar

Conclusion

- Assembly is the key to became a successful malware analyst
- Foundation of x86 concepts
- The only real way to get good at disassembly is to ***practice.***
- Next chapter, we will take a look at IDA Pro

Malware Analysis

Chapter 4: A Crash Course in x86 Disassembly

王志

zwang@nankai.edu.cn

updated on Oct. 17th 2021

College of Cyber Science

Nankai University

2021/2022