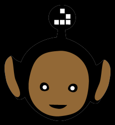
5th laboratory

Rodrigo Castillo

8 de septiembre de 2020



1. Read the introduction to .^A Crash Course In X86 Disassembly.^and identify 3 reasons to do disassembly over a malware:

1. Dissasembly can be used to understand the malware without running it
2. Disassembly is usefull for reconstruct the code of a malware, sometimes there are processes that cannot be seen on process monitor, thats why looking at the assembly code can be usefull
3. it also works for understand how malware works, personally, i use it for hacking on binary exploitation contests

2. Read the section ”Levels of abstraction.^and explain the difference between high-level language, low-level language and machine code.

low level programming ...

- the code is readed by a compiler that transforms code into machine code
- its supposed to be more portable
- the number and abstractions for instructions is lower, you need more instructions to tell the machine to do the same than an high level programming language
- instructions runs faster than in a high level programming language because compiler dont need to assume machine instructions
- machine code can be disassembly

high level programming language ...

- the code is readed by a interoreter line by line
- its supposed to be less portable but i think is more portable nowdays

- coding is faster because interpreter assumes a lot of machine code instructions
- instructions runs slower, but, for example, python interpreter transforms heavy algorithms into low level programming porgrams for make it faster
- its not necesary to disassembly because investigator can read the code of the program, sometimes is encoded but thats another prolem that occurs also with low level programming languages

there are a lot of differences between low level and high level programming languages but listing all of them is impossible in this section

3. Python may be considered a hardware, microcode, machine code, low-level language, high-level language or interpreted language? Why?

Pythons is a high level language

4. What is the difference between an application written in C++ and the equivalent application written in Python?

- application in c++ is faster but the code is also larger
- application in c++ is compiled and in python is interpreted
- application in c++ its supposed to be more portable

5. Why the assembly dialect must not be the same for all the PC architectures? Why is useful to study x86 dialect?

Assembly language is machine code, that means that it can run or not depends on the machine , there are several architectures for computers, x64 , x86 , ARM , 8 bit ... *etc* , that means that instructions for an architecture not necesary will run in other architectures, however, the most used architecture nowdays is $x64$ and codes compiled for $x86$ runs in $x86$ machines and in $x64$ machines, that mean that if an attacker want to make malware, the best option for him is making it for $x86$ because most of the machines in the world will run it .

6. Read the section "The x86 architecture.^{a}nd explain the utility of registers"

registers are the same as variables in the code, they store hex numbers inside them and use it for writing in other spaces or other registers

7. Read the section ”Main Memory.ªnd explain the differences between Stack, Heap, code and Data.

- in *data* its stored all the static variables that are not going to be modified
- in *code* instructions are loaded
- in *heap* all dinamyc variables are loaded, all the variables that are modified constantly.
- in *stack* all local variables , function declarations, memory spaces are loaded

8. Read the section Ïnstructions.ªnd explain what are opcodes (operation codes)

assembly have several different instructions , opcodes are instructions in assembly between registers, such as mv , jmp , jeq , jne , sub , add , push ...
are instructions that lead the machine to perform compiled code instructions , for example, jmp instruction overwrite *eip* to an instruction, or *add* add a hex value over a register *push* loads a hex into a space of memory, all the several instructions are derivates and combinations from those instructions.

9. What is the difference between little-endian and big-endian format?

imagine that we are going to encode the number $0xcafebabe$ ...

- in big endian would be xca xfe xba xbe
- in little endian would be xbe xba xfe xca

10. Put an example of 3 types of instructions used in x86 architectures and explain its purpose (mov, add, push, call, xor, cmp). Full documentation: https://software.intel.co 64-and-ia-32-architectures-sdm-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4.html Pag 116.

i already made it in a previus section

11. Explain the difference between Immediate, Register and Memory Address? Give an example of each one.

inmediate is when an instruction runs without searching for a value in a register , for example push 0xf that runs this instruction into the stack, register and memory would be for example mov eax 0xf

12. How many registers exists in a X86 architecture and what are used for? (General, Segment, Status, Instruction Pointer)

there are 8 registers of 16 bits, 8 of 32 bits , there is the flag register and most important one is *eip* that is instruction pointer that store the next instruction , ones of them are used for storing hex values into memory spaces and others are used for storing hex value into other registers

13. What is the purpose of NOP instruction?

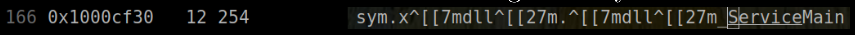
its no operation instruction , it stands for when the programmer want to freeze a register

14. What is the difference between move and lea (load effective address) instructions

as i understand, mov moves a value from a register to another , lea safe the value into the register

15. Find the address and PE Section where the imported function gethost-byname resides

i found service main at 0x1000cf30 and gethost-byname in 0x1000d02e



```
166 0x1000cf30   12 254         sym.x^[[7mdll^[[27m.^[[7mdll^[[27m_ServiceMain
```