# OT Lab 1 - understanding assembly

## 1. Preparation:

a. You can use any debugger/disassembler as long as it supports the given task architecture (32bit or 64bit)
b. Try to stay away from decompilers, as this will help you to better understand the nature of your task based on assembly only, remember in real-world tasks you will have to deal with much larger binaries.
c. IMPORTANT: Please check what each binary does before running it (don't trust us 😈)
d. Check some writeups about some CTF to see what you should/shouldn't include in your report
e. Try to do the lab in a Linux VM, as you might need to disable ASLR

## 2. Theory

a. What is ASLR, and why do we need it?
b. What kind of file did you receive (which arch? 32bit or 64bit?)?
c. What do stripped binaries mean?
d. What are GOT and PLT?
e. How can the debugger insert a breakpoint in the debugged binary/application?

## 3. Disassembly

a. Disable ASLR using the following command "sudo sysctl -w kernel.randomize_va_space=0"
b. Load the binaries **from the Task 3 folder** into a disassembler/debugger
c. Do the function prologue and epilogue differ in 32bit and 64bit?
d. Do function calls differ in 32bit and 64bit? What about argument passing?
e. What does the command **ldd** do? "ldd BINARY-NAME"
f. Why in the "sample64-2" binary, the value of **i** didn't change even if our input was very long?

## 4. Reversing

a. You will have multiple binaries **in the Task 4 folder**, Try to reverse them by recreating them using any programming language of your choice (C is preferred)

Notes:

- Make the report as technical as possible (no installation guide please).
- ALWAYS include a PoC code with your report (if required)
- Try to have assembly code in a formatted block (check markdown)
- If you paste some data (command output), make sure it is readable and the format did not change
- If you want to include a command in the report, please highlight it (bold, italic, different layout, ...)
- The compile command for each file is the following:
    - sample64 = gcc -fno-stack-protector -g -o sample64 sample.c
    - sample64-2 = gcc -o sample64-2 sample.c
    - sample32 = gcc -m32 -fno-stack-protector -g -o sample32 sample.c