Libc-Attack Lab

Introduction:

This lab is an interesting variant of buffer-overflow attack; this attack can bypass an existing protection scheme currently implemented in major Linux operating systems. A common way to exploit a buffer-overflow vulnerability is to overflow the buffer with a malicious shellcode, and then cause the vulnerable program to jump to the shellcode stored in the stack. To prevent these types of attacks, some operating systems allow programs to make their stacks non-executable; therefore, jumping to the shellcode causes the program to fail. Unfortunately, the above protection scheme is not fool-proof. There exists a variant of buffer-overflow attacks called Return-to-libc, which does not need an executable stack; it does not even use shellcode. Instead, it causes the vulnerable program to jump to some existing code, such as the system() function in the libc library, which is already loaded into a process's memory space.

Instructions:

In this lab, you are given a program (retlib.c) with a buffer-overflow vulnerability; your task is to develop a Return-to-libc attack to exploit the vulnerability and finally to gain root privilege.

ASLR: Ubuntu and several other Linux-based systems use address space randomization to randomize the starting address of heap and stack, making guessing the exact addresses difficult, and therefore it will be disabled in this lab:
$ sudo sysctl -w kernel.randomize_va_space=0

The StackGuard Protection Scheme: The gcc compiler implements a security mechanism called StackGuard to prevent buffer overflows. In the presence of this protection, buffer overflow attacks do not work. We will disable this protection during the compilation using the -fno-stack-protector option.

Non-Executable Stack: Ubuntu used to allow executable stacks, but this has now changed. The binary images of programs (and shared libraries) must declare whether they require executable stacks or not, i.e., they need to mark a field in the program header. Because the objective of this lab is to show that the non-executable stack protection does not work, you should always compile your program using the "*-z noexecstack*" option in this lab.

You must complete the lab by following the book/slides, and completing the missing components of file "exploit.py"

If successful you will get a shell open, then try to get a root shell like the previous lab.