GRENOBLE - INP ESISAR

OS 430
Lab - 1.3

Cyril Bresch

Supervisor : Arthur Desuert (*arthur.desuert@grenoble-inp.org*)

1 INTRODUCTION

This lab aims at introducing students to the exploitation of simple buffer overflows. The vulnerability is largely used by attackers to modify the behaviour of a software, a server or an operating system. Nowadays, buffer overflows are still exploited in many applications which use unsafe low level languages such as C/C++. Although the exploitation of memory bugs are becoming increasingly difficult due to protections such as ASLR, canary, and NX, experienced attackers can still exploit them. You understood it, in this lab we're gonna pwn binaries! But! In a first time we will free ourselves from the protections provided by the linux kernel and the compiler.

1.1 ADVICE

1. Exercises are in order of increasing difficulty.
2. Quality takes precedence over quantity.
3. The most important points of the Lab are: understanding simple attacks, the ability to propose ideas and the ability to search information on the web.
4. Don't trust Internet, each unjustified code copy will result in a score of zer0.

1.2 Definition

In the following exercises, the **padding** is defined as the number of bytes needed to reach the return address in the stack (without overwriting it).

2 Exercise : 'Dummy Exploitation'

1. What is the return address of the vulnerable function and how do you find it?
2. Pass root and type the following command:

   ```
   echo 0 > /proc/sys/kernel/randomize_va_space
   ```

   Explain the effect of the command on the system.
3. Give a command to display the char 'A' two hundred and forty-three times. (You can use Python, Ruby, Perl, Bash).
4. Find the overflow if there is one and give the **padding**.
5. Use the vulnerability to redirect the execution flow into the function that is never called.

3 Exercise : 'Dummy ShellCoding'

1. Write a simple C program which calls a shell using an exec() function. Compile the program with the "-g" and "-ggdb" options.
2. Give the assembly instruction that performs a 64-bit syscall, then, give its 32-bit equivalent.
3. Using gdb, give the status of the registers just before the syscall pops a shell. Explain the values contained in these registers.
4. Write an assembly code that allows you to launch a shell. This shellcode will be in 64 bits.

4 Exercise : 'Basic Exploitation'

1. Find the overflow in the binary and give the **padding**.
2. Explain what is a shellcode and why certain shellcode needs to be null byte free, give an example.
3. Explain the effects of the "-fno-stack-protector" and "-z execstack" compiler options.
4. Exploit the flaw to make the binary open a shell. You can take a shellcode from the "exploit database" or "shellstorm".

5 Exercise : 'Hardened Binary the beginning'

1. You are the analyst, find a way to exploit it.