GRENOBLE - INP ESISAR

OS 430
Lab - 2

Cyril Bresch

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

1 INTRODUCTION

The last lab focuses on code injection attacks on vulnerable binaries. Two methods have been exposed: the first one was the modification a return address in the stack in order to redirect the execution flow on a malicious function. The second method was the redirection of the binary on a malicious shellcode. We have also seen that it is possible to create a code that automates the exploitation of a vulnerable binary (it is an exploit). However, the previous lab worked because it was possible to execute code directly in the stack. Nowadays operating systems prevent code execution in the stack using the NX bit. Moreover, the address space has become random (ASLR). Consequently, it is difficult to predict the position of a malicious code at execution time. In this lab we will try to bypass the NX protection using "return to libc" attaks. Finally, we will demonstrate that address randomization (ASLR) is not very effective in 32 bits.

1.1 ADVICE

1. Same as Lab 1.
2. Use vi, vim and gdb.

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 : 'RET2FUNC FOR DUMMIES'

1. Remind the basic steps performed on the stack when calling up a 32-bit function. We limit ourselves to the prologue.
2. Pass root and type the following command:

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

   What is PIE?

3. Explain the Return To Libc Method.
4. Find the overflow if it exists and give the **padding**.
5. Use the buffer overflow to make the binary jumping into the function that validates the exercise (Note: the function MUST print its message). Write an exploit script in Python.

3 EXERCISE : 'BASIC RET2LIBC'

1. Is there an overflow ? If yes, Why can't we inject a shellcode into the vulnerable binary ? Prove it with a command.
2. Give the gdb commands that allow you to find the addresses of the LIBC functions you will use to carry out your attack.
3. Explain the effects of the "-fno-stack-protector" and "-z execstack" compiler options.
4. Exploit the flaw to make the binary opening a shell. Write an exploit script in Python.

4 EXERCISE : 'HARDENED BINARY CHEATING THE ASLR'

1. Pass root and type the following command:

```
echo 2 > /proc/sys/kernel/randomize_va_space
```

   Now you are the analyst, find a way to exploit it.