Lab 1: Constructing Buffer Overflow Exploits

Preparation

Recommended Operating System: Ubuntu 32 bits or 64 bits, >=14.04

A vulnerable program can be downloaded at https://www.cs.ucr.edu/~heng/teaching/cs250-sp25/example01.c. The "Password" defined in the function "IsPasswordOkay" is vulnerable to buffer overflow attacks. You need to construct two exploits to take advantage of this vulnerability.

To compile the vulnerable program, use the following command line:

gcc –g –fno-stack-protector -m32 –z execstack –o example01 example01.c

We need to disable the address layout randomization. You have two options:

Option 1: disable ASLR globally. You need root privileges.

```
sudo su
echo 0 > /proc/sys/kernel/randomize_va_space
exit
```

Option 2: disable ASLR locally and temporarily. You don't need root privileges.

setarch `uname -m` -R /bin/bash

This will open a new Bash shell for you with ASLR disabled, including all child processes created from this shell. Just exist the shell once you are done.

Task1: Code Injection

Objective: Construct an exploit input that can inject a shellcode to be executed in the vulnerable program. A template exploit is provided at https://www.cs.ucr.edu/~heng/teaching/cs250-sp25/codeinjection.bin.

Hint: The shell code is appended in the end of this template. This shell code will run "ps" program to print the running processes. To complete this exploit, we need to figure out what value should be used to overflow the return address. In this case, the value should be the start address of the shell code. Since the shell code will be injected on the stack, figuring out the address of "Password" on the stack is the key to solve this problem. We can attach a debugger gdb to the running program and locate the address of "Password".

Task2: Return to libc Attack

Objective: Construct an exploit input that takes advantage of the existing "system" library call in libc to achieve the same goal as in the first task. That is, we want to run "ps" command, without injecting any code into the stack of the vulnerable program.

Hint: Instead of jumping to the shell code from the exploit input, we want to redirect the program to jump into the “system” function call. We need to locate the entrypoint of the “system” function call. gdb can help us with this. We also need to prepare the stack properly to supply the parameter “ps” to the system function call. A template exploit is provided at https://www.cs.ucr.edu/~heng/teaching/cs250-sp25/returntolibc.bin.

Requirement for submission

Students need to submit your lab report through eLearn. In the report, a screenshot with clear explanation is needed for each key step.