

Lab 5

Sheikh Muhammad Farjad
CS-16059
Section A
Batch 2016-17

Course Instructor: Sir Umar Iftikhar
Computer Systems Security

Lab Environment

For lab 5, I preferred to use the Kali Linux machine that is my base system. The seed ubuntu VM had some configuration issues and this is why I simulated the environment in my base system and performed the tasks of Lab 5.

Task 1

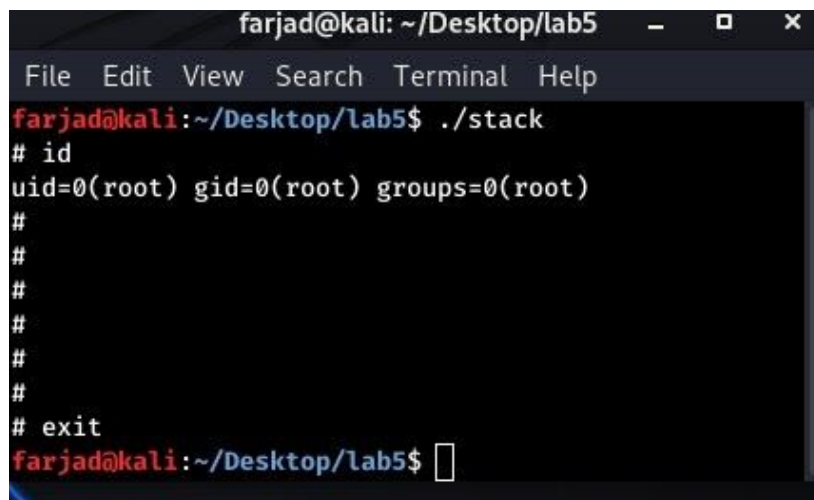
In this task, the shellcode was executed via buffer overflow vulnerability. The shellcode provided the root access to the executing program or shell. The shellcode was stored in the exploit.c program. The commands were executed on Kali's terminal in the following sequence:

```
$ gcc -o exploit exploit.c
$ ./exploit
$ ./stack
```

It is also to be noted that the feature of address space randomization was disabled for conveniently guessing the starting address of the code of interest. Following command was executed in the terminal for this purpose:

```
# sysctl -w kernel.randomize_va_space=0
```

The access to the root shell can be observed in the screenshot attached below.

A screenshot of a terminal window titled 'farjad@kali: ~/Desktop/lab5'. The terminal shows the execution of the './stack' command, followed by the 'id' command which outputs 'uid=0(root) gid=0(root) groups=0(root)'. The user then enters several '#' characters and finally 'exit', returning to the prompt 'farjad@kali:~/Desktop/lab5\$'.

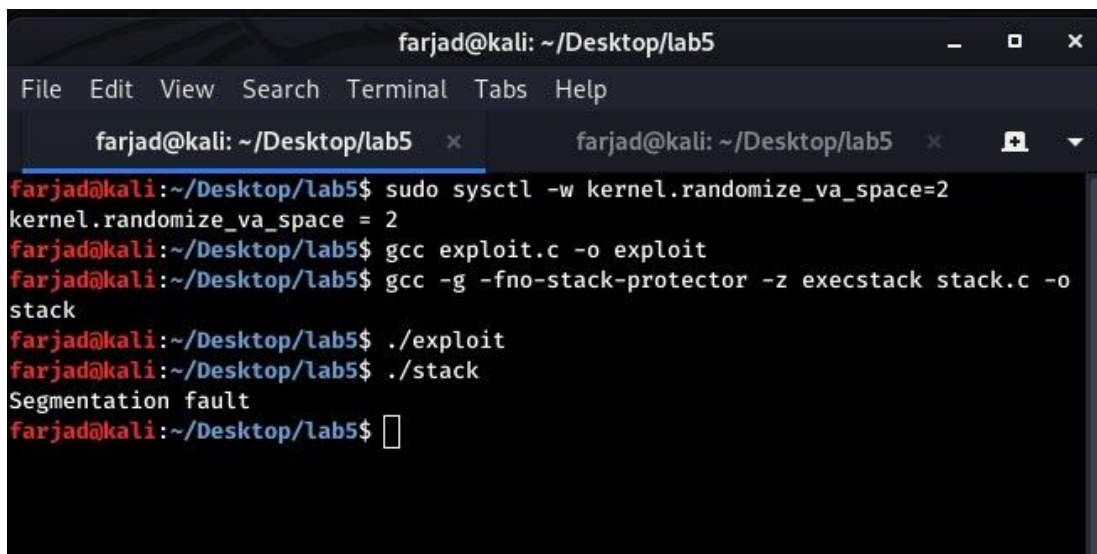
```
farjad@kali: ~/Desktop/lab5
File Edit View Search Terminal Help
farjad@kali:~/Desktop/lab5$ ./stack
# id
uid=0(root) gid=0(root) groups=0(root)
#
#
#
#
#
#
# exit
farjad@kali:~/Desktop/lab5$
```

Task 2

In this task, the address space randomization feature was enabled for executing the attack of Task 1. The address space randomization feature breaks the sequential address space of the program. The LOC (Lines of Code) not arranged in a particular sequence make it difficult for the program to guess the exact starting address of the malicious code stored by the hacker.

When the address space randomization is enabled, the program gives **Segmentation Fault**. Following command was run in the terminal for enabling the address space randomization feature:

```
# sysctl -w kernel.randomize_va_space=2
```



```
farjad@kali: ~/Desktop/lab5
File Edit View Search Terminal Tabs Help
farjad@kali: ~/Desktop/lab5 x farjad@kali: ~/Desktop/lab5 x +
farjad@kali:~/Desktop/lab5$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
farjad@kali:~/Desktop/lab5$ gcc exploit.c -o exploit
farjad@kali:~/Desktop/lab5$ gcc -g -fno-stack-protector -z execstack stack.c -o
stack
farjad@kali:~/Desktop/lab5$ ./exploit
farjad@kali:~/Desktop/lab5$ ./stack
Segmentation fault
farjad@kali:~/Desktop/lab5$
```

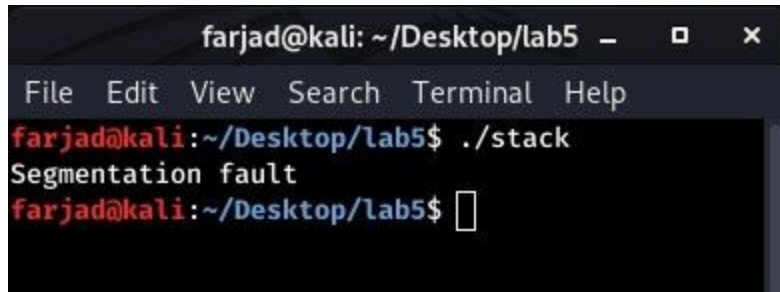
In order to make the attack successful, the code can be executed recursively and this practice increases the chance of success. In my machine, it took approximately 2 hours to correctly execute the attack. Following command was used for executing the **stack** program repeatedly.

```
$ sh -c "while [ 1 ]; do ./stack; done;"
```

After the time-span of approximately 2 hours, the access to the root shell was provided.

Task 4

When I used the *noexecstack* option during compilation of the program, the program execution did not result in access to the root shell. The main reason for this behaviour is the transformation of the executable stack to the non-executable. In non-executable stack, no program or shellcode can be executed within the stack region and this is why the attack was failed.

A terminal window titled 'farjad@kali: ~/Desktop/lab5' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'farjad@kali:~/Desktop/lab5\$'. The user enters './stack', and the terminal outputs 'Segmentation fault' followed by a new prompt 'farjad@kali:~/Desktop/lab5\$' with a cursor.

```
farjad@kali: ~/Desktop/lab5
File Edit View Search Terminal Help
farjad@kali:~/Desktop/lab5$ ./stack
Segmentation fault
farjad@kali:~/Desktop/lab5$
```