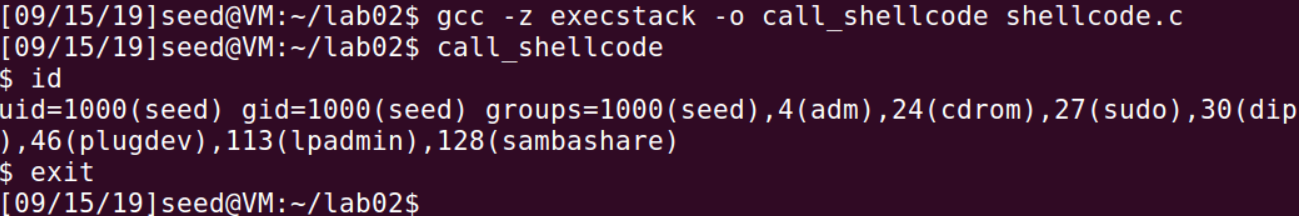
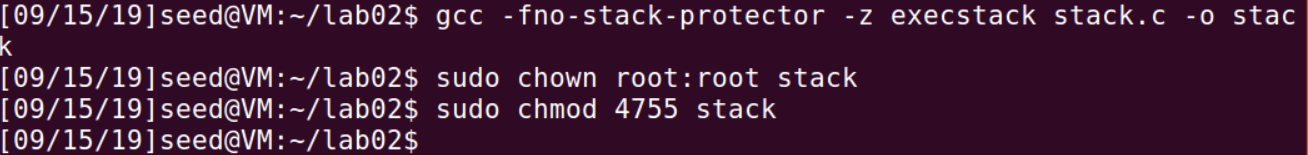
Lab02

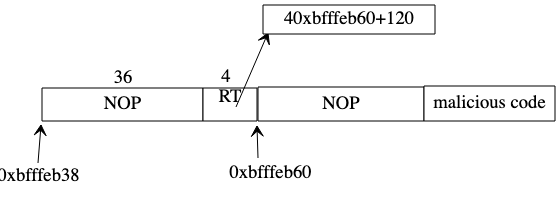
# Task01

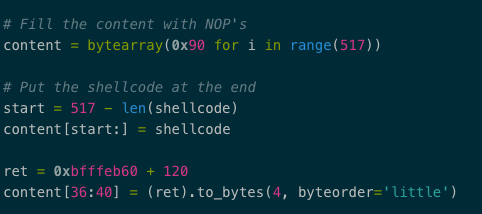
## Steps

1. Try the program provided by the guidance, which intended to write some shellcode in the buffer and invoke it.
2. Create and compile the vulnerable program(stack.c) and made it a set-UID program for root. 

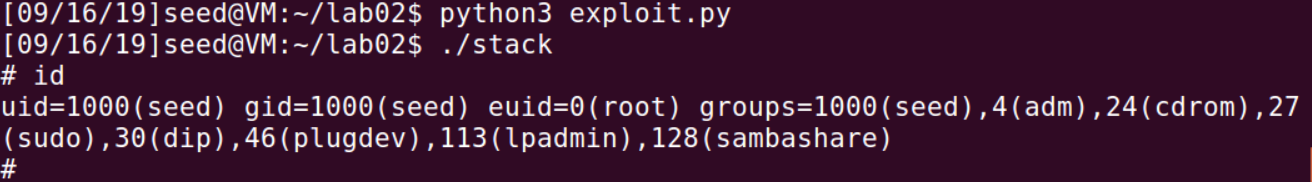
# Task02

## Steps

1. Use gdb mode to run the ‘Stack’ program, find the address of the ‘ebp’and the start address of the ‘buffer’.
2. Through calculation, we can determine that the structure of bad file is like below:
3. Create a Python file(exploit.py) to generate the bad file. Part of the code:

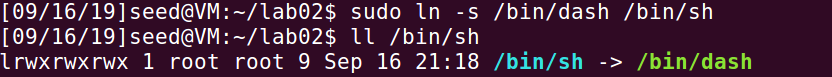
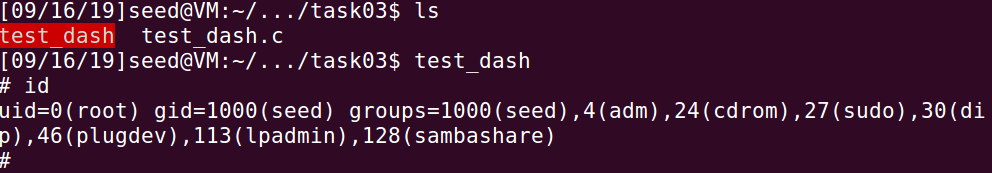
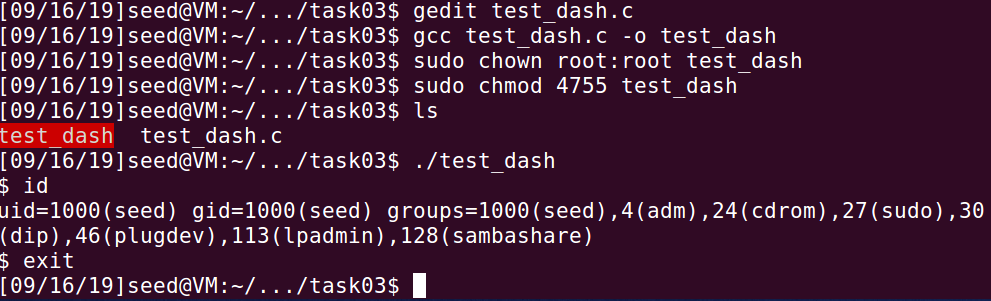
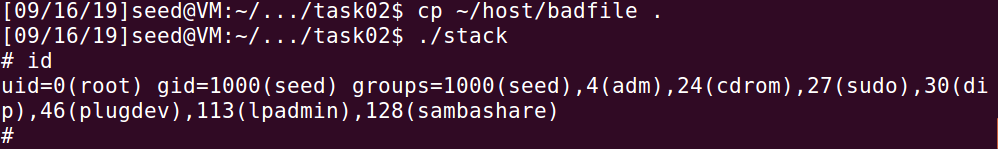


The ‘ret’ is the address that needed be put in the return field.

1. Generating the bad file and then run the Set-UID program ‘stack’.

# Task03

## Steps

1. Change the ‘/bin/sh’ from ‘zsh’ to ‘dash’.
2. Create and compile the program and made it a Set-UID program for root. This program just simply invokes a ‘shell’ without using ‘setuid(0)’ before.
3. If we add ‘setuid(0)’ in the program, the shell will be invoked in root privilege.
4. So, we add some shellcode before the shellcode we used at task 2. And execute the ‘stack’ program again.

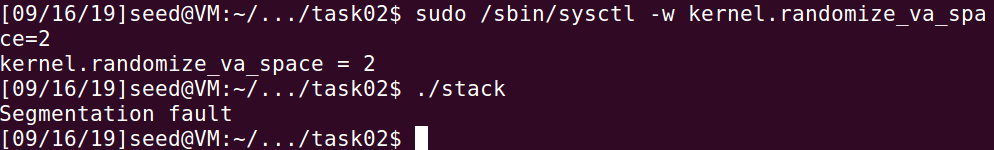
## Explanation

The ‘dash’ has the mechanism that when we invoked a shell from a set\_UID program, it will compare the real id with effective id. If they are different from each other, the ‘dash’ will made the effective user-id equals the real user-id.

But if we can let the real user-id equals the 0, which represents the root, we can make this mechanism useless. By adding the shellcode which will set the ‘uid’ 0 before the shell invoked, we can complete the attack in the dash environment.

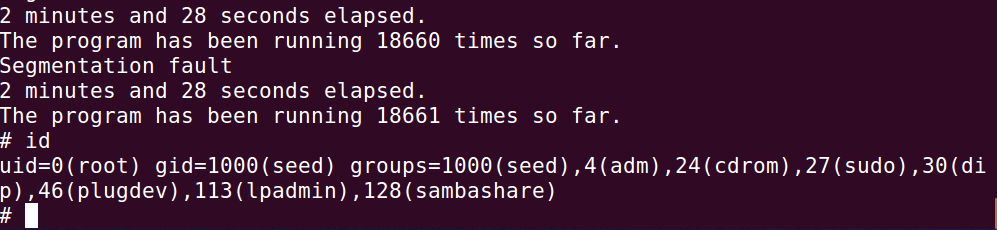
# Task04

## Steps

1. We turn on Ubuntu’s address randomization and rerun the ‘stack’ program in task 02.

The program fails to obtain the root privilege but throws an exception.

1. Use the brute-force approach to attack the vulnerable program.

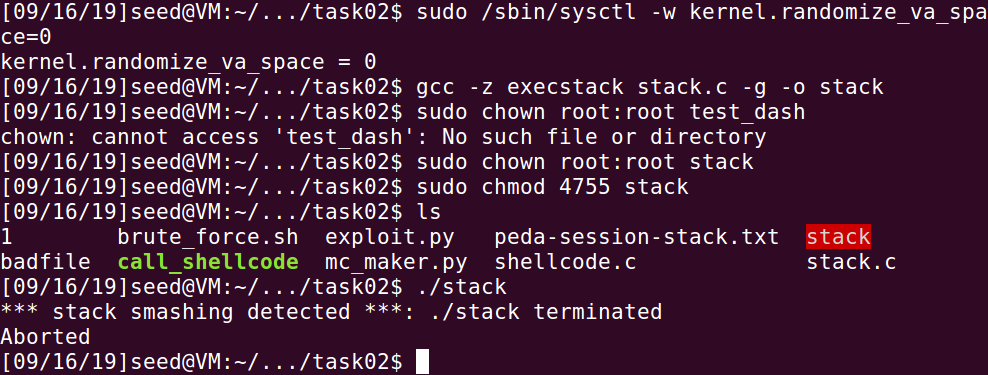
The result:

## Observation

By using this brute-force approach, we can successfully achieve the goal with the same bad file. I spent about 2 minutes, tried nearly 20000 times. Finally, the address that I put in the return area points to the right place.

# Task05

## Steps

1. Turn off the address randomization and compile the program ‘stack.c’ without the ‘-fno-stack-protector’ option. Then run it.

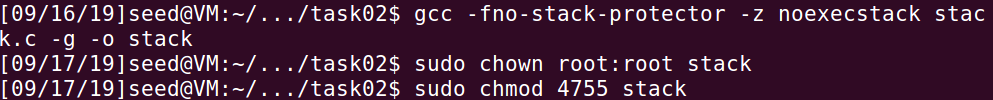
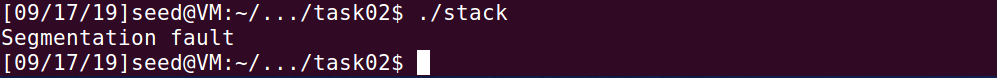
The program was terminated by the StackGuard.

## Explanation

The StackGuard inserts a small random value between the buffers and the function return address. And store the random value in a place isolated from the stack to prevented be overwritten. Before the function return, the tiny value will be checked and if the value has changed, the program will be terminated.

# Task06

## Steps

1. Turn off the address randomization. Recompile the program ‘stack’ with ‘noexecstack’ option and made it Set-UID program.
2. Do the steps in task 02.

## Explanation

The ‘noexecstack’ will make it impossible to run shellcode in the stack. So, if we inject our code in the stack with this option open, the program will just throws a segmentation fault exception.