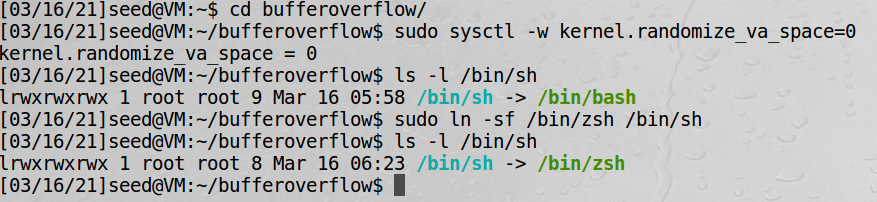
# CS763 LAB2

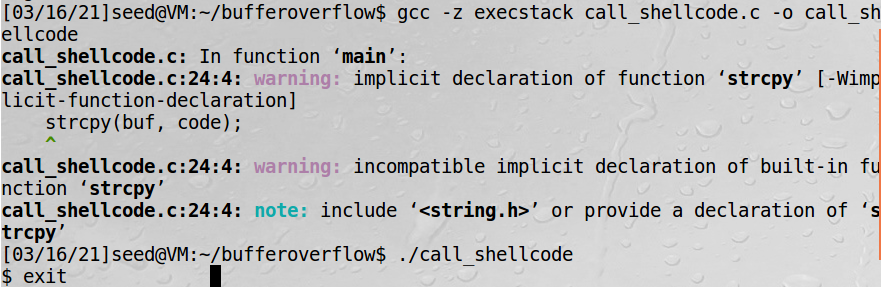
DONGHANG HE

Before staring we need to disable the address space randomization. So that we could guess the exact address. Also, we need to link the zsh to /bin/sh.



## Task 1: Running Shellcode

Running the basic program to test the execstack option, which allows code to be executed from the stack. After compile the code using the command show below, run the program we found that the program jump into the shell. But after the next task we could know that’s not the root shell.



## Task 2: Exploiting the Vulnerability

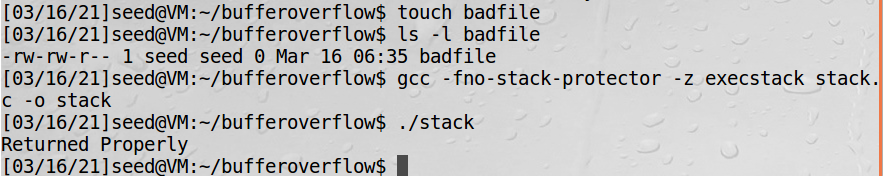
In this task, first we need to finish the program. Using gdb to set a breakpoint at bof method. Then run the program in gdb and check the address of buffer and ebp. So I use 0x24 for offset and 0xbfffeaf8 for ret.

文本

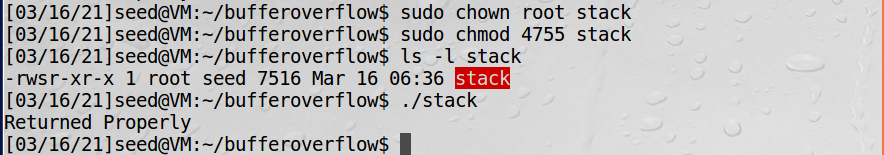
描述已自动生成

Using the exploit.c file we could create a badfile which causing the buffer overflow. But first we could create an empty file to test the stack.c program.

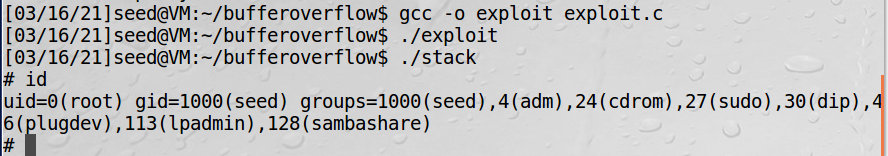
By compile and running the stack.c file we could found, If the file is small then the program will execute normally and output "returned properly".



To ensure that you can enter the root shell after using the new badfile. We need to change the ownership of the program to root and change the permission to 4755 to enable the Set-UID bit.

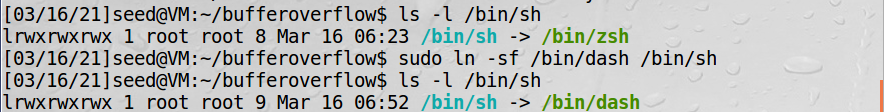


Because we are not going to overflow the buffer in the exploit program. So we could compile this program using the default StackGuard protection. After compiling it we run the exploit and stack. We found that we enter the root shell and enter the “id” then saw the uid is root. That means we made the buffer overflow and enter the root shell.



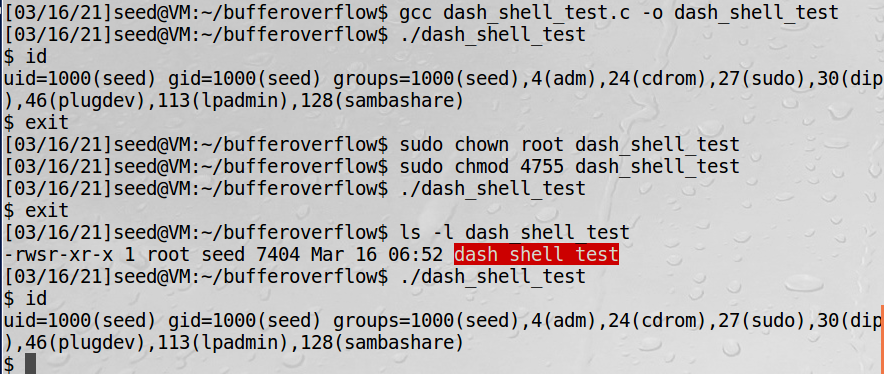
## Task 3: Defeating dash’s Countermeasure

To finish this task first we need to change the link with /bin/sh from zsh to dash.

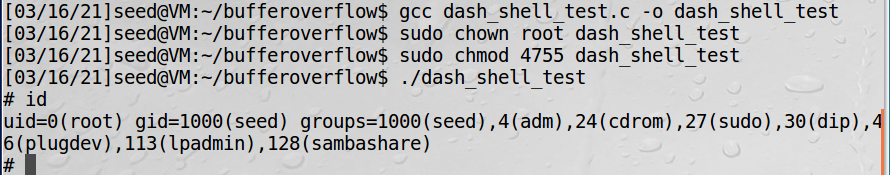


In the dash\_shell\_test file we need to comment out the ‘setuid’ and run it first.

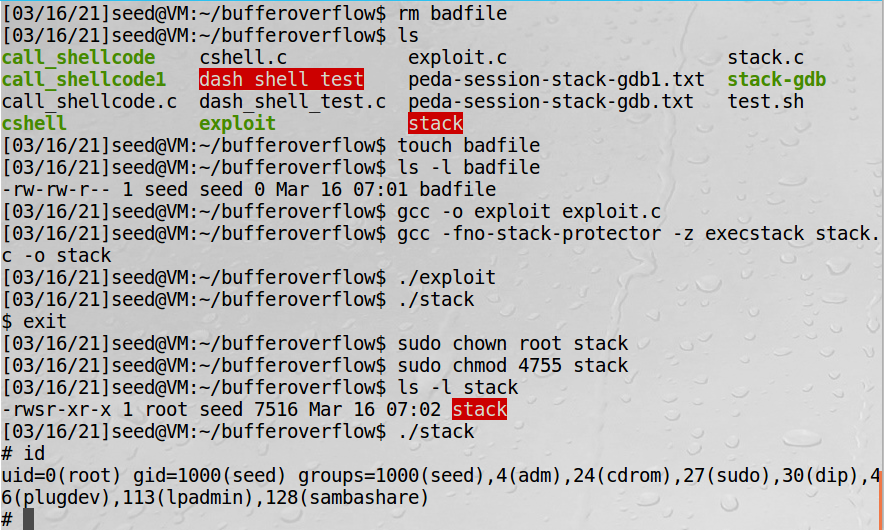
After compile and run the code we found that we didn’t enter the root shell. Then we change the owner and permission which we did in task 2 to test. But also, not enter the root shell. So, we know the old measure could not work.



Then we uncomment the ‘setuid’ and run it again. Also, we need to change the owner and the permission. We found that program enter the root shell. So, using the dash we need to write the setuid code in the program to let us enter the root shell.



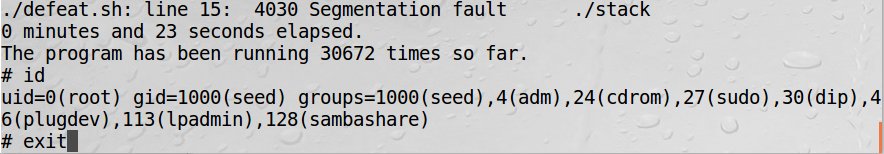
Also, we could add the setuid in the badfile to check whether we could enter the root shell. Remove the old one and using the new program to create a new badfile. Then follow the steps in task 2 and we found the program could enter the root shell. Once again verified our previous conclusions.



## Task 4: Defeating Address Randomization

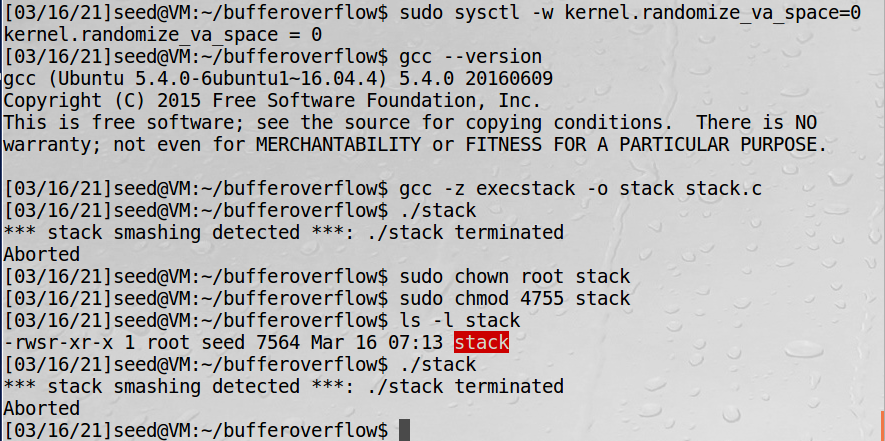
In this task we need to enable the address randomization so that we could find some measure to defeat it. And I forgot to take a screenshot of the command because all the screenshots were taken during I write the document. I did it for the second time to wirte the document.

First, we need to create a shell script file to run the vulnerable program in an infinite loop. After running the script, we found the program enter the root shell in 23 seconds. The script stops and enter the root shell mean we succeed invoke the stack program. The program has been running 30k times. And if I enter exit and let it continue running, it will enter the root shell in about 3 minutes after running 300k times. In the document it said the address can have 520k possibilities. So it doesn’t take a long time to crack it with brute force.



## Task 5: Turn on the StackGuard Protection

First we need to turn off the address randomization which we turn on before. And in the document it said that the StackGuard only work in GCC version 4.3.3 and above. So, we need to check the it first. Then compile and run the stack program. We could see that in the terminal it said the stack smashing. Also, we try to change the owner and the permission. Again, the error found. That’s mean we need the -fno-stack-protector command to finish the task.



## Task 6: Turn on the Non-executable Stack Protection

In the last task we need to turn on the non-executable stack protection. We use the non-executable option to compile the stack program. And we found it said segmentation fault. That’s mean we are failed to enter the root shell.

