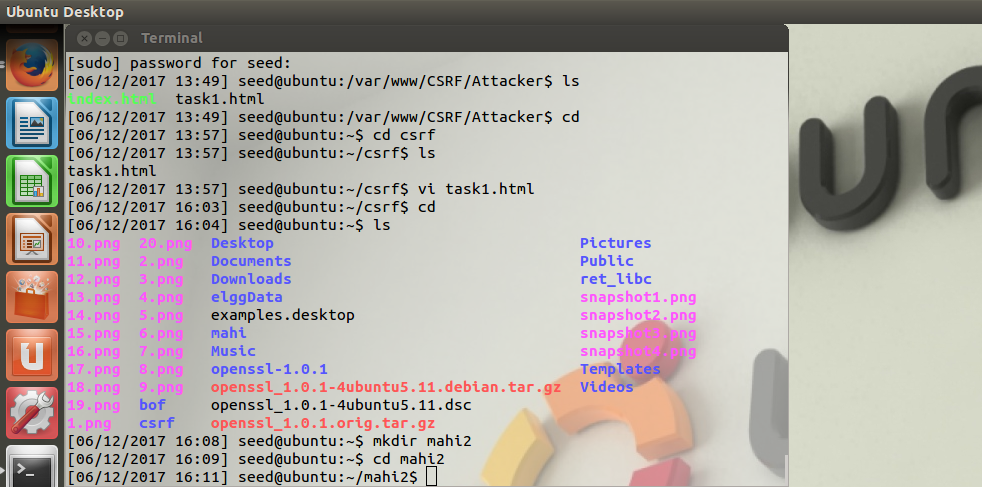
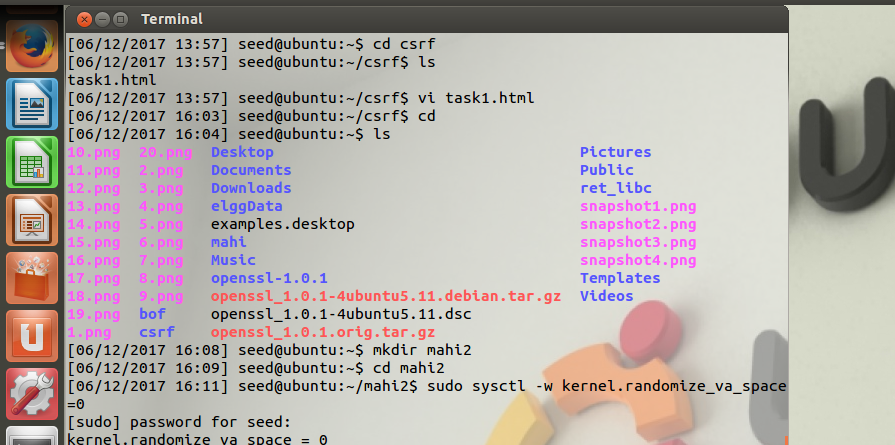
Return-to-libc attack is nothing but the variant of the buffer overflow attack where there is no need of executable stack and shell code. This return to libc attack comes in handy where some operating systems only allow non executable stacks to perform operations. It does not let the vulnerable program to jump into the bad file. Instead of it, it makes it jump into the existing code. No executable files cannot be bypassed by buffer overflow attack and that is why we implement the return to libc attack which is the extension of buffer overflow attack.

Basically, every Linux or Unix based system has address randomization turned on and as done prior to the buffer overflow attack i.e., turning off randomization, we must also turn off address randomization prior to perform return to libc attack too. The main advantage of this attack is we can perform actions through non executable stack.

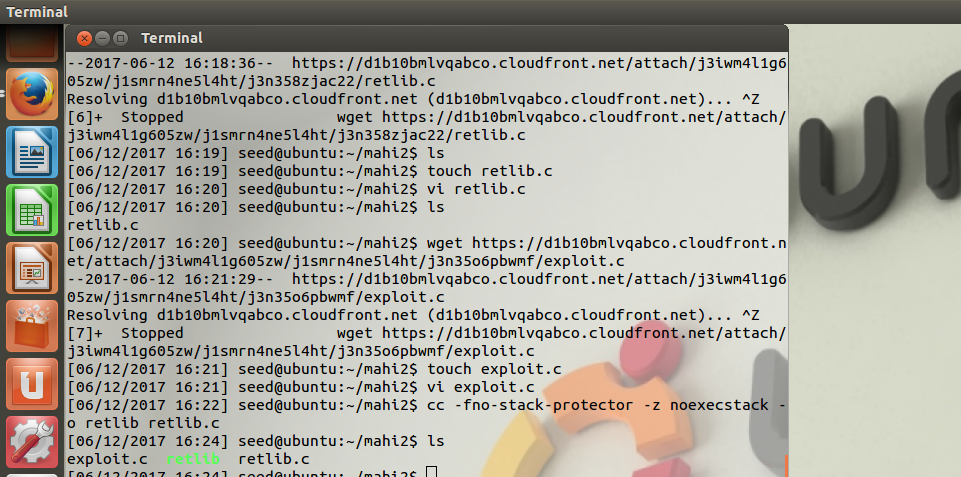
1. First of all we must create a directory for performing return to libc attack.



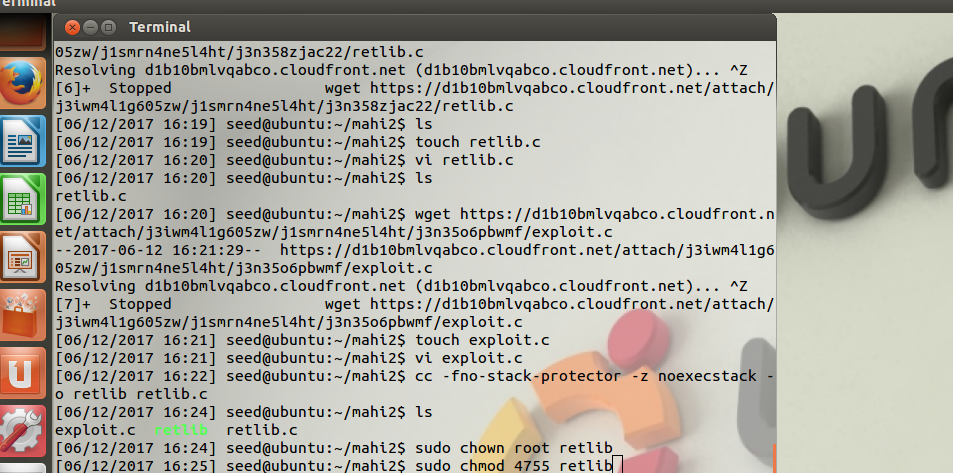
1. Turing off the address randomization by making the randomize attribute equals to zero.



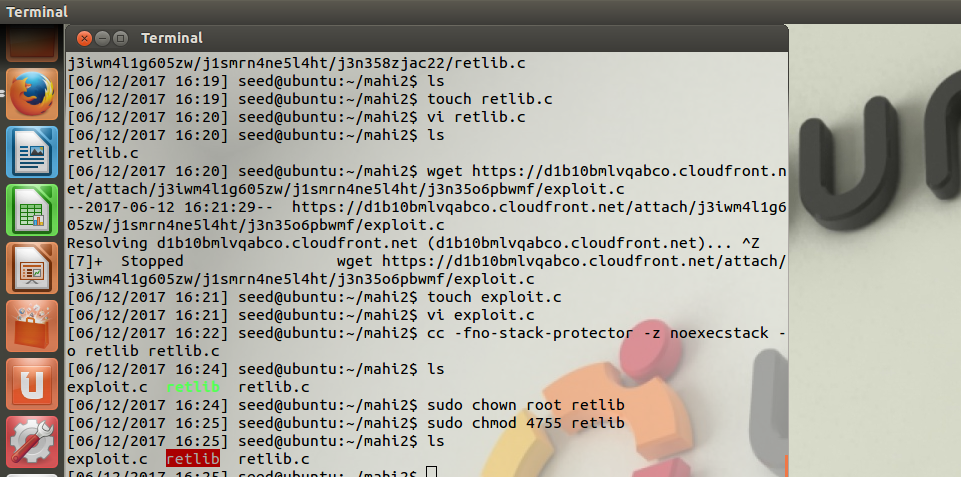
1. Downloading the retlib.c and exploit.c files into the prescribed directory. And now we should compile the retlib.c by turning of the stack protector and using non executable stack. We turn off the stack protector because it doesn’t let the attack to happen. And after compiling you can notice that executable retlib file is turned into green color.



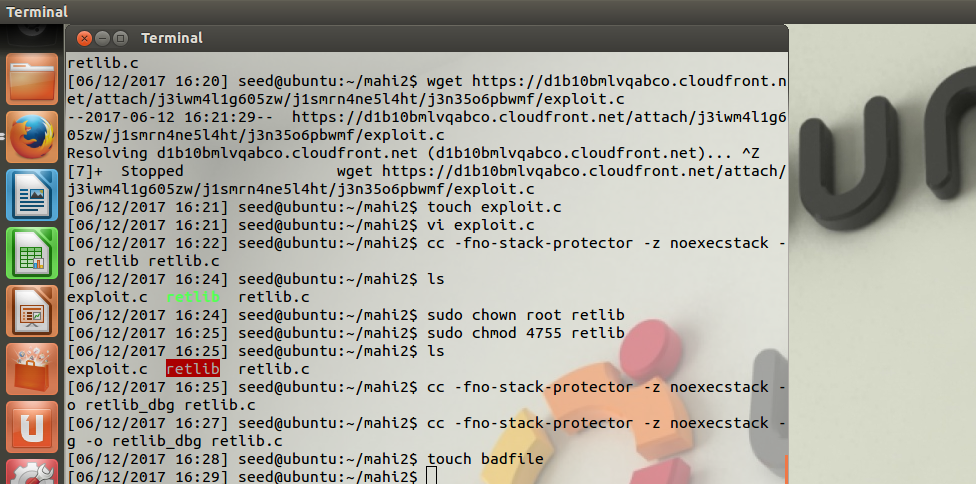
1. And now perform the remaining steps of the set uid root version i/e/, changing the permissions.



1. After performing the set uid root compilation the executable retlib file is turned in to vulnerable program.



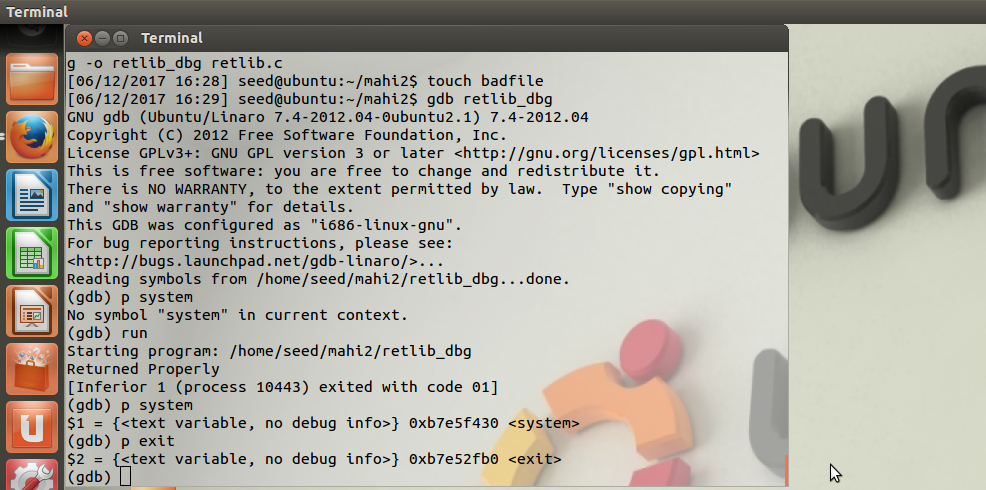
1. Now just compiling the debug version of the retlib file for performing the GDB operations.



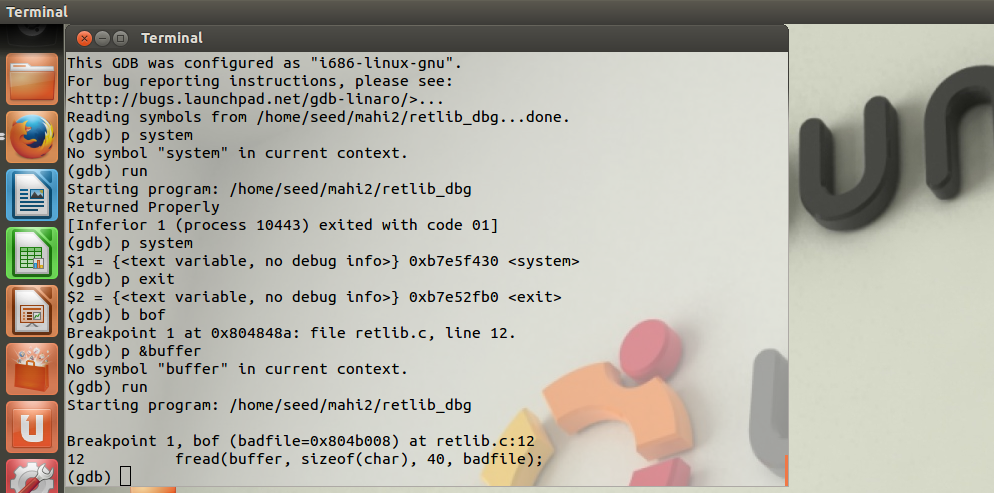
1. Now after entering into the DBG system by clicking “gdb retlib\_dbg”, just press run for creating the breakpoint for the program.



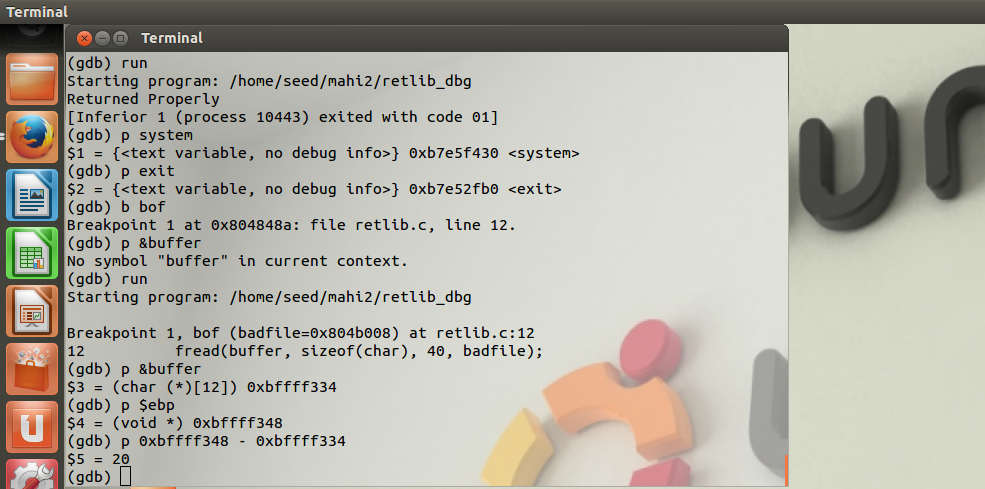
1. If we get “returned properly” after running the GDB then we can proceed for finding the distance. So to find the distance we find the address of the buffer array and the EBP register. Basically exit is used so that we can come back as soon as we complete the execution.



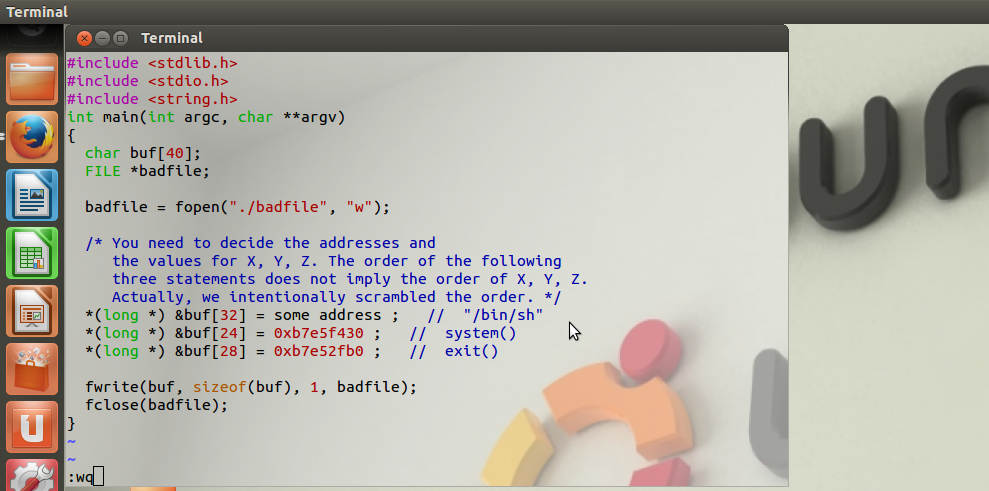
1. We should note down the system and the exit addreses which is used for changing the address in the exploit.c file. And now creating the breakpoint for the exploit.c file. After pressing the run command breakpoint is created wherever buffer overflow statement occurs in the program.



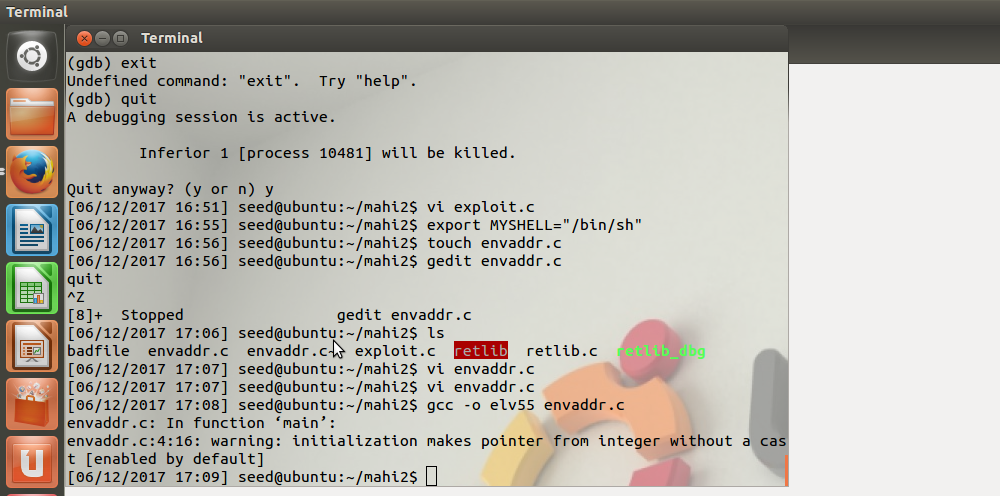
1. After starting the GDB just run the command “p &buffer”. This gives the address of the buffer. In the same way we can get the address of EBP register using the command ‘p $ebp’. So that we can note down both the addresses. So from these two registers we can calculate the distance.



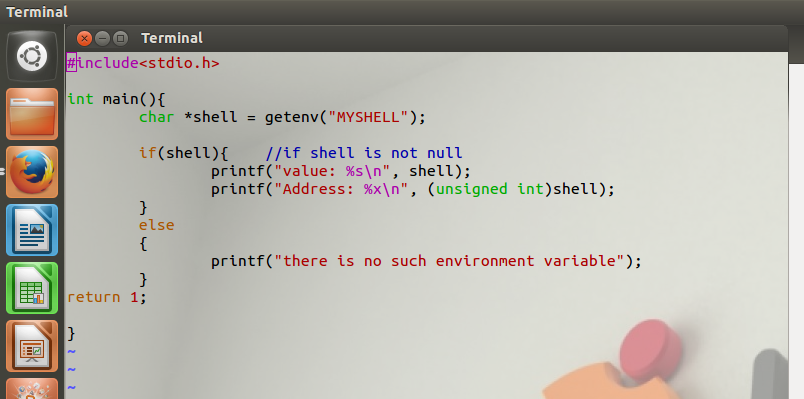
* 1. So now in the exploit.c file we must update the value of the system and the exit addresses with the exact array indices.



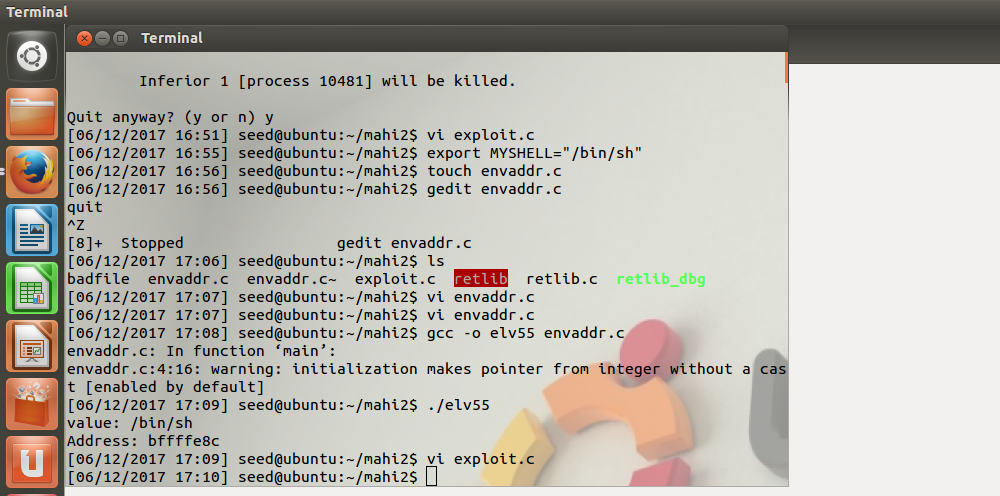
1. And now we must exit from the GDB system. What we have found out is the system address form the utilization of GDB system and now we have to find the address of the string “/bin/sh”. So, to find out this string address we place the string in the buffer and get the address using environment variable. For this process we write a small c program which is used to print the address of this variable.



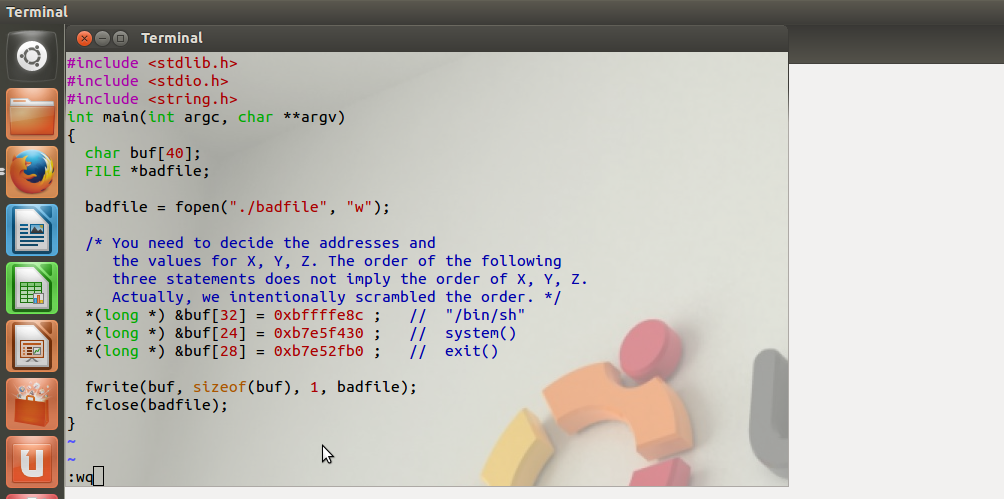
11.1 The envaddr.c will be as follows



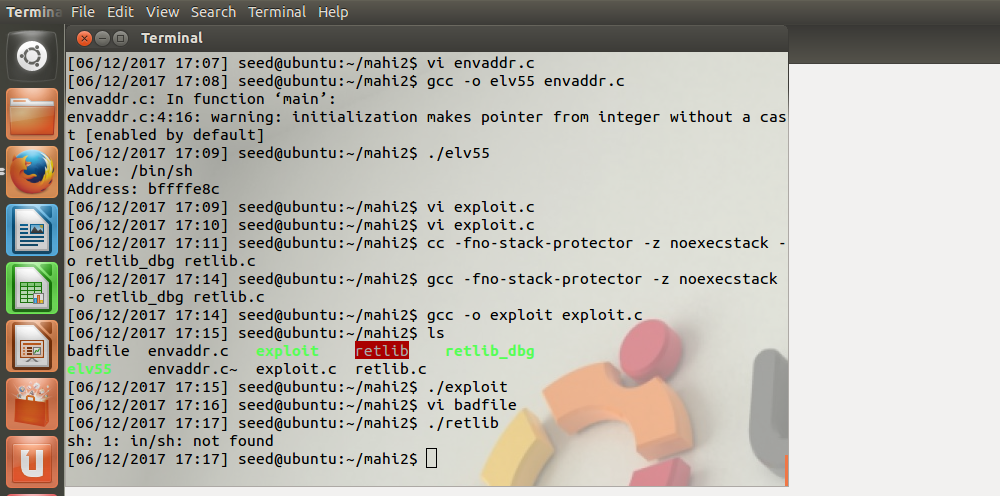
1. After compiling and executing this file we get the address for the string. We must paste this address in the string address variable in the exploit.c file.



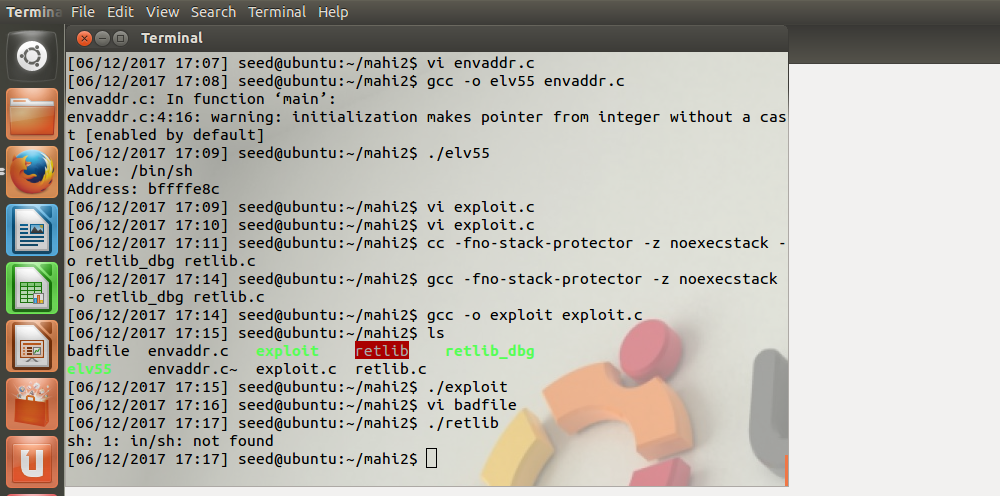
1. So, now in the exploit.c file we replace the address of the string obtained from the previous string.



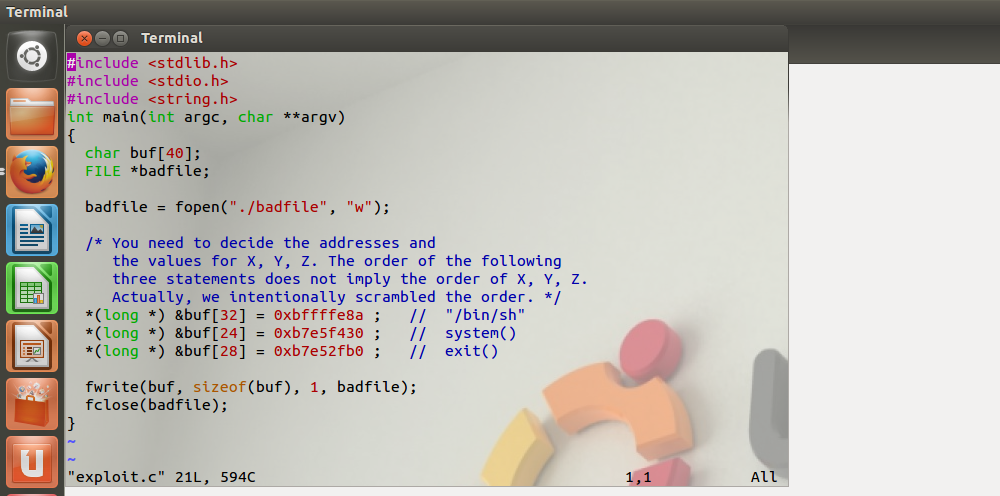
1. Both the retlib\_dbg and the exploit executable files must be recompiled again and if we list out all the files in the directory prior to this step we can find retlib as the vulnerable file.And after that exploit file must be executed.



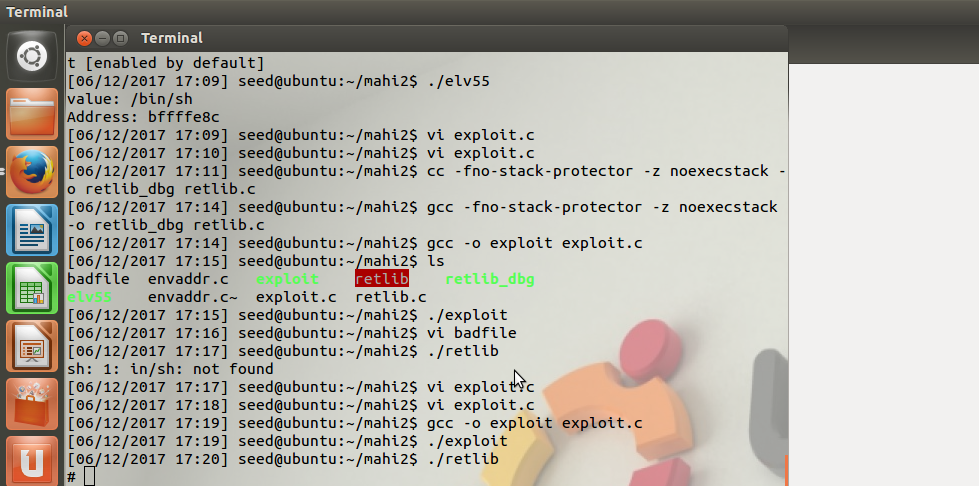
1. And when we execute the retlib file the following occurs on the screen.



1. To overcome this we have to change the address of the argument from “e8c” to “e8a”.



1. So again recompiling the exploit.c files and the retlib file we finally get the root which means that the hack attack is happened. But when we try removing the exit address in exploit.c file and if we recompile again we will get the root but while exiting we will get a segmentation fault which is different from normal.



1. Changing the name of the executable file cannot lead to root because length of the file name matters.

