ASSIGNMENT – 1
Special Topic :- Secure Programming with C

SRN: PES1UG19CS542
Name: Trisha Jain
Section: I2
Mail Id: jtrisha0403@gmail.com

Question 1: a)#include<stdio.h>
```
#include <string.h>
int main(){
        char *p=NULL;
        p=(char*)malloc(12);
        strcpy(p,"Hello World!");
        return 0;
}
```
i) Does the above code has any defects which leads to vulnerability, if so indicate?

Answer 1. a)
LINE 5 (p = (char*)malloc(12);) :-
The **allocation of memory by malloc is not checked**. So, if malloc fails to allocate memory for the pointer p and returns NULL, the string copy function will fail. Hence it is necessary to check if the malloc returned NULL pointer which would indicate that memory was not allocated successfully.
While providing the size for malloc, we should use sizeof(char) instead of assuming char occupies one byte.

LINE 6 (strcpy(p,"Hello World!");) :-
Since strcpy is used the **length of the string being copied is not checked** which may lead to vulnerability if the string length exceeds the allocated memory for the pointer p. Instead of strcpy we can use strncpy to make sure the length is checked.
In this example if the memory allocation is successful then the allocated bytes will be 12 and the string that is being copied is also of twelve bytes. But we need an extra byte for accommodating null character at the end of the string. So the **string is not null – terminated**. In this case the null character is not added to string copied in p. This also may lead to vulnerability.

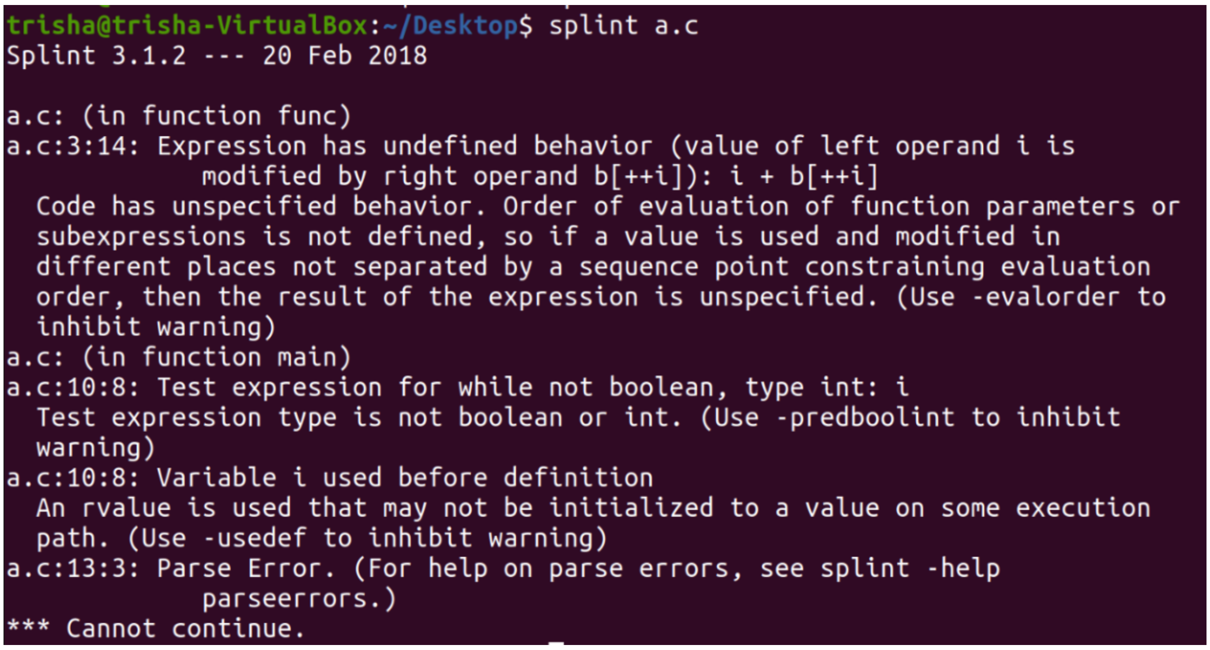
The **memory allocated to p is not released** before leaving the function main.

b) #include <stdio.h>
```
void func(int i, int *b) {
        int a = i + b[++i];
        printf("%d, %d", a, i);
}
int main(){
        int l=0; int a[]={11,22,33,44,55}; int i;
        func(l,a);
        while(i){
                printf("%d\n",i);
                ++i;
        }
        return 0;
}
```

i) List the defects identified by splint if any, in the above code?

Answer 1. b)

Screenshot of Splint :-



```
trisha@trisha-VirtualBox:~/Desktop$ splint a.c
Splint 3.1.2 --- 20 Feb 2018

a.c: (in function func)
a.c:3:14: Expression has undefined behavior (value of left operand i is
             modified by right operand b[++i]): i + b[++i]
  Code has unspecified behavior. Order of evaluation of function parameters or
  subexpressions is not defined, so if a value is used and modified in
  different places not separated by a sequence point constraining evaluation
  order, then the result of the expression is unspecified. (Use -evalorder to
  inhibit warning)
a.c: (in function main)
a.c:10:8: Test expression for while not boolean, type int: i
  Test expression type is not boolean or int. (Use -predboolint to inhibit
  warning)
a.c:10:8: Variable i used before definition
  An rvalue is used that may not be initialized to a value on some execution
  path. (Use -usedef to inhibit warning)
a.c:13:3: Parse Error. (For help on parse errors, see splint -help
             parseerrors.)
*** Cannot continue.
```

<u>As seen in the above screenshot, the following defects are identified by splint :-</u>

1) In the **line 3** of the program, the expression has unspecified behaviour because we can't determine the order in which the operands will be evaluated.
2) In the **line 10** of the program, the error is that the variable "i" is not declared and the test expression for the while loop is not Boolean as in it never resolves to false which leads to infinite loop.

<u>Question2:</u>

<u>a)</u> #include <stdio.h>

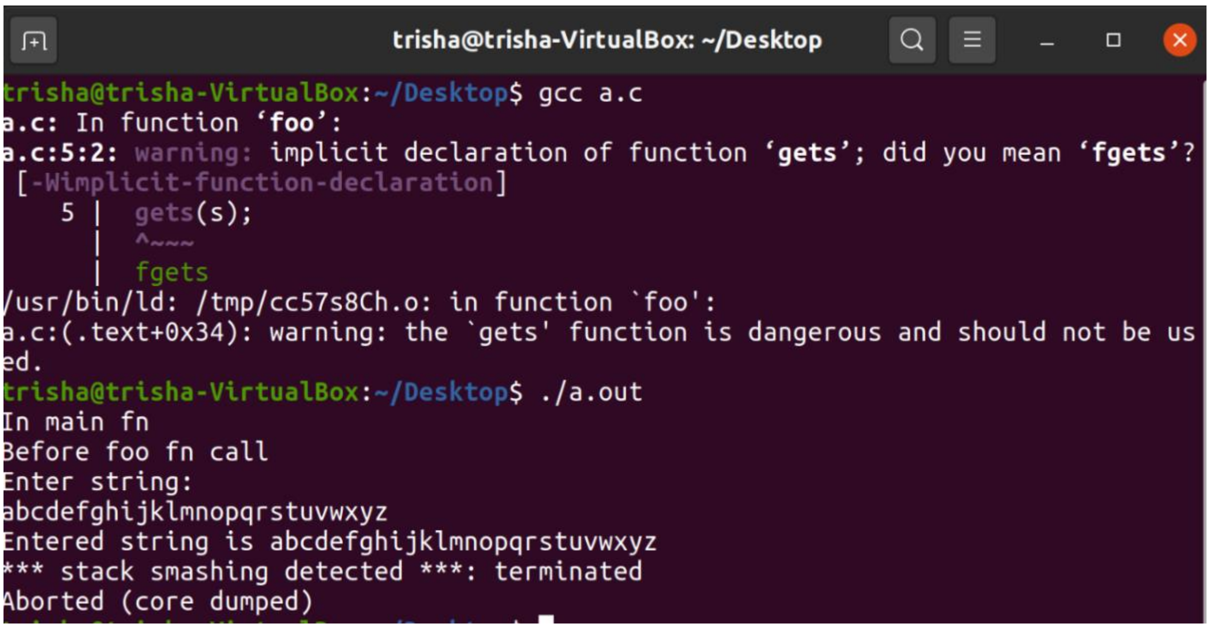
```
void foo(){
        char s[20];
        printf("Enter string:\n");
        gets(s);
        printf("Entered string is %s\n",s);
}
int main(){
        printf("In main fn\n");
        printf("Before foo fn call\n");
        foo();
        printf("After foo fn call\n");
        return 0;
}
```

(i) Compile & run the above code. And note down the output, if user inputs string as "abcdefghijklmnopqrstuvwxyz" and justify the output.

(ii) Why is the function gets() unsafe? //1m

Answer 2. (i)

Screenshot of the output :-



```
trisha@trisha-VirtualBox:~/Desktop$ gcc a.c
a.c: In function ‘foo’:
a.c:5:2: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’?
 [-Wimplicit-function-declaration]
    5 |  gets(s);
      |  ^~~~
      |  fgets
/usr/bin/ld: /tmp/cc57s8Ch.o: in function `foo':
a.c:(.text+0x34): warning: the `gets' function is dangerous and should not be us
ed.
trisha@trisha-VirtualBox:~/Desktop$ ./a.out
In main fn
Before foo fn call
Enter string:
abcdefghijklmnopqrstuvwxyz
Entered string is abcdefghijklmnopqrstuvwxyz
*** stack smashing detected ***: terminated
Aborted (core dumped)
```

Output :

```
In main fn
Before foo fn call
Enter string:
abcdefghijklmnopqrstuvwxyz
Entered string is abcdefghijklmnopqrstuvwxyz
***stack smashing detected ***: terminated
Aborted (core dumped)
```

The output displayed is “stack smashing detected”. This is because “s” can only store up to 20 characters (as specified in the foo function -> char s[20]), but we have entered 26 characters (when prompted by gets(s) in the foo function). This leads to buffer overflow which results in the above shown error. (stack smashing error)

Answer 2. (ii) The gets() function is unsafe because gets accepts input until it encounters a new line character or EOF. So, we need to explicitly check the length of the entered string otherwise it may lead to buffer overflow as it did in the above example. Thus, using gets() should be avoided as it may lead to buffer overflow.

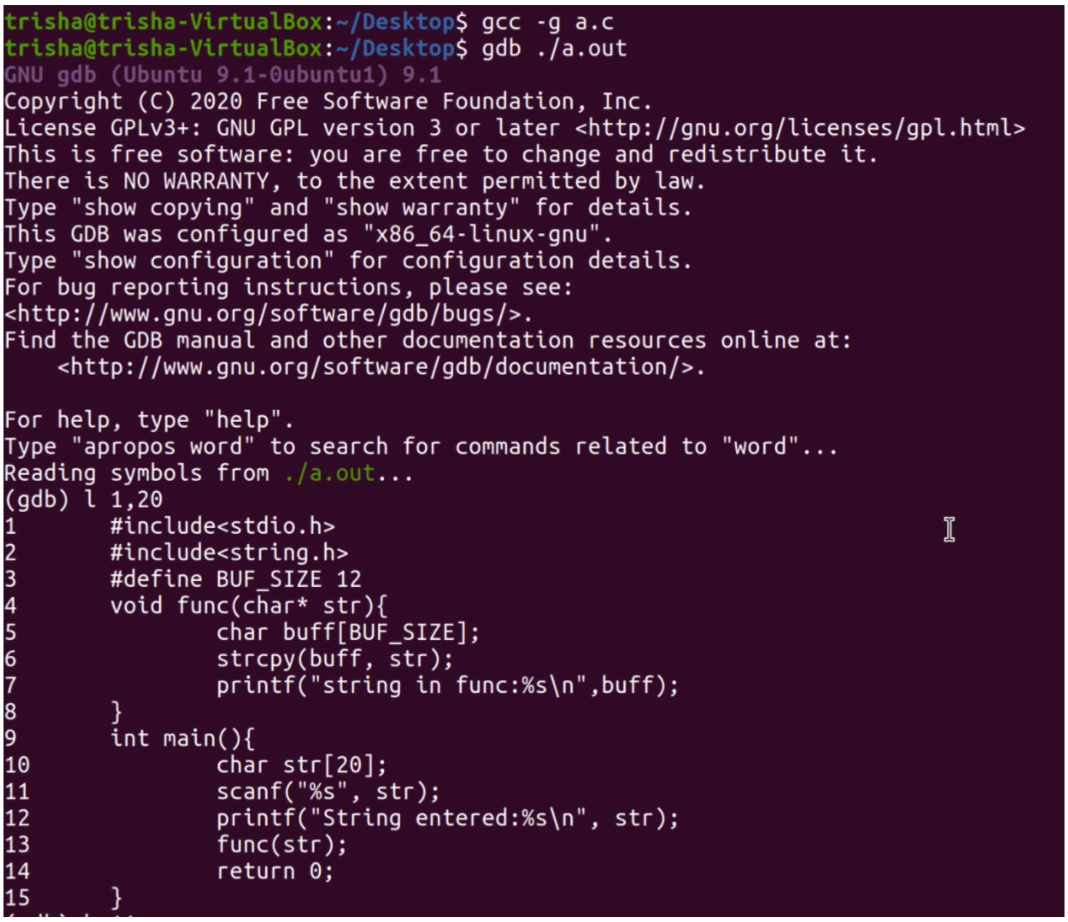
Question 3:

a)
```
#include <stdio.h>
#include <string.h>
#define BUF_SIZE 12
void func(char *str){
        char buff[BUF_SIZE];
        strcpy(buff,str);
        printf("string in func:%s\n",buff);
}
int main(){
        char str[20];
        scanf("%s",str);
        printf("String entered:%s\n",str);
        func(str);
        return 0;
}
```

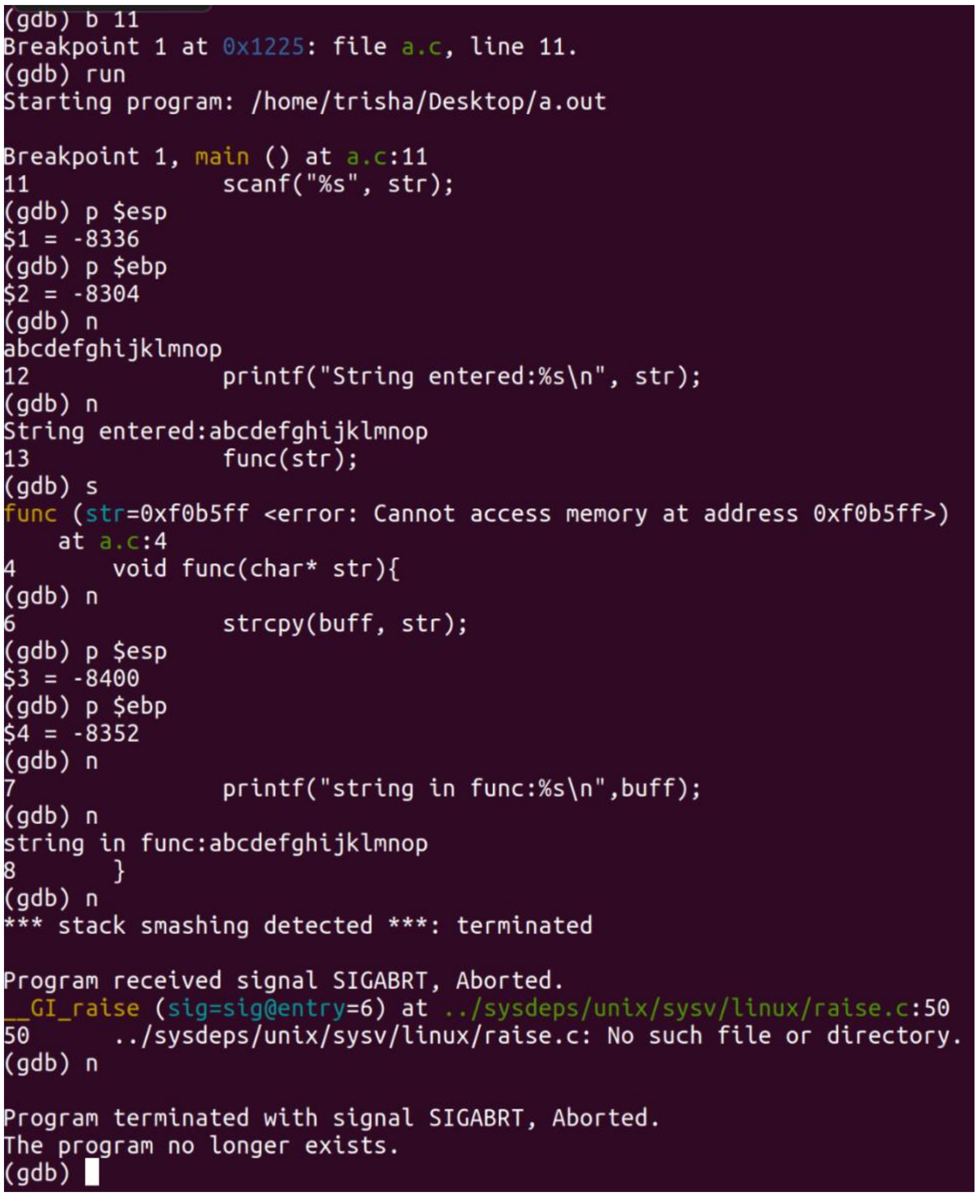
i) Identify the defect which leads to vulnerability, if any. And debug it using gdb debugger

In the given program the defect which leads to vulnerability is present in the func function (**line 5 and line 6**) because the buff string should have been dynamically allocated to accommodate all the characters of str and instead of the strcpy function strncpy should have been used to avoid buffer overflow error . These defects result in vulnerable code.

Code Debugging using GDB debugger :-



```
trisha@trisha-VirtualBox:~/Desktop$ gcc -g a.c
trisha@trisha-VirtualBox:~/Desktop$ gdb ./a.out
GNU gdb (Ubuntu 9.1-0ubuntu1) 9.1
Copyright (C) 2020 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 "x86_64-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 ./a.out...
(gdb) l 1,20
1       #include<stdio.h>
2       #include<string.h>
3       #define BUF_SIZE 12
4       void func(char* str){
5               char buff[BUF_SIZE];
6               strcpy(buff, str);
7               printf("string in func:%s\n",buff);
8       }
9       int main(){
10              char str[20];
11              scanf("%s", str);
12              printf("String entered:%s\n", str);
13              func(str);
14              return 0;
15      }
```



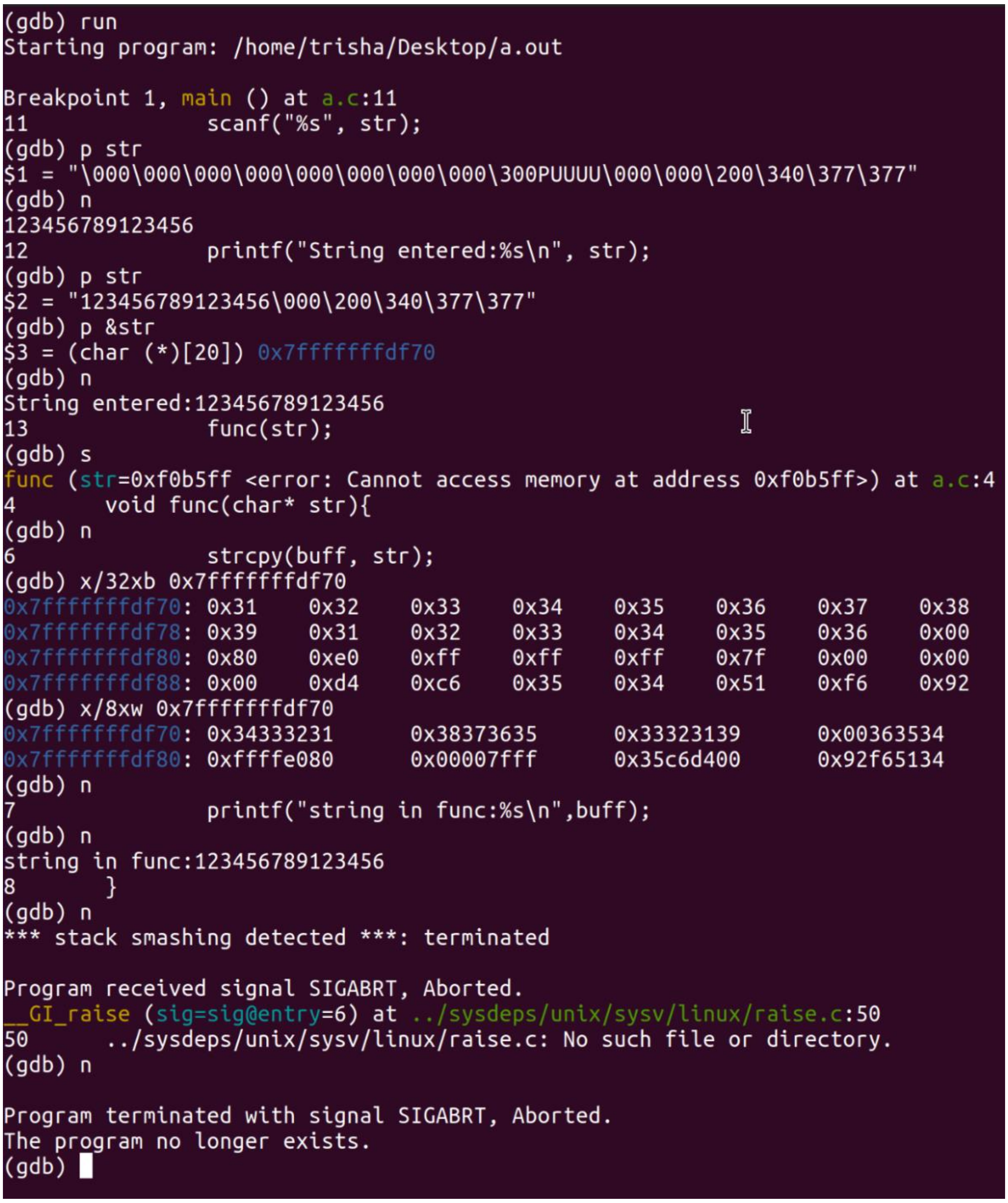
```
(gdb) b 11
Breakpoint 1 at 0x1225: file a.c, line 11.
(gdb) run
Starting program: /home/trisha/Desktop/a.out

Breakpoint 1, main () at a.c:11
11              scanf("%s", str);
(gdb) p $esp
$1 = -8336
(gdb) p $ebp
$2 = -8304
(gdb) n
abcdefghijklmnop
12              printf("String entered:%s\n", str);
(gdb) n
String entered:abcdefghijklmnop
13              func(str);
(gdb) s
func (str=0xf0b5ff <error: Cannot access memory at address 0xf0b5ff>)
    at a.c:4
4       void func(char* str){
(gdb) n
6               strcpy(buff, str);
(gdb) p $esp
$3 = -8400
(gdb) p $ebp
$4 = -8352
(gdb) n
7               printf("string in func:%s\n",buff);
(gdb) n
string in func:abcdefghijklmnop
8       }
(gdb) n
*** stack smashing detected ***: terminated

Program received signal SIGABRT, Aborted.
__GI_raise (sig=sig@entry=6) at ../sysdeps/unix/sysv/linux/raise.c:50
50      ../sysdeps/unix/sysv/linux/raise.c: No such file or directory.
(gdb) n

Program terminated with signal SIGABRT, Aborted.
The program no longer exists.
(gdb)
```

<u>Steps used for debugging</u> : First the first 20 lines of the program are displayed to help us fix the break points. Then the breakpoint is fixed to 11 because that's where the input is taken. After this we run the program and keep track of the esp and ebp values as we step into the main and the func functions. Finally the program ends with the error being displayed.

As shown in the above screenshot, we see that when we step into the "func" function the esp and ebp values change. And when we print the string in the "func" function its correctly printed but when we leave the function "stack smashing detected" gets displayed.

This is because, in the above program, the string that is entered can be of 20 characters. But in the func function, the max length of the string (buffer) where str will be copied is 12. So when we use strlen function, the length of the string that will be copied is not checked and as a result buffer overflow takes place due to which stack smashing takes place as can be seen in the above screenshot.

The screenshot below shows the memory layout of the string :-



```
(gdb) run
Starting program: /home/trisha/Desktop/a.out

Breakpoint 1, main () at a.c:11
11              scanf("%s", str);
(gdb) p str
$1 = "\000\000\000\000\000\000\000\000\300PUUUU\000\000\200\340\377\377"
(gdb) n
123456789123456
12              printf("String entered:%s\n", str);
(gdb) p str
$2 = "123456789123456\000\200\340\377\377"
(gdb) p &str
$3 = (char (*)[20]) 0x7fffffffdf70
(gdb) n
String entered:123456789123456
13              func(str);
(gdb) s
func (str=0xf0b5ff <error: Cannot access memory at address 0xf0b5ff>) at a.c:4
4       void func(char* str){
(gdb) n
6               strcpy(buff, str);
(gdb) x/32xb 0x7fffffffdf70
0x7fffffffdf70: 0x31    0x32    0x33    0x34    0x35    0x36    0x37    0x38
0x7fffffffdf78: 0x39    0x31    0x32    0x33    0x34    0x35    0x36    0x00
0x7fffffffdf80: 0x80    0xe0    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffdf88: 0x00    0xd4    0xc6    0x35    0x34    0x51    0xf6    0x92
(gdb) x/8xw 0x7fffffffdf70
0x7fffffffdf70: 0x34333231      0x38373635      0x33323139      0x00363534
0x7fffffffdf80: 0xffffe080      0x00007fff      0x35c6d400      0x92f65134
(gdb) n
7               printf("string in func:%s\n",buff);
(gdb) n
string in func:123456789123456
8       }
(gdb) n
*** stack smashing detected ***: terminated

Program received signal SIGABRT, Aborted.
__GI_raise (sig=sig@entry=6) at ../sysdeps/unix/sysv/linux/raise.c:50
50      ../sysdeps/unix/sysv/linux/raise.c: No such file or directory.
(gdb) n

Program terminated with signal SIGABRT, Aborted.
The program no longer exists.
(gdb)
```