**CS6008 Cryptography and Network Security.**

**Assignment No.2**

**Name:** G Vignesh **Reg No:**2018103078

**Date:** 9.4.2022

**Module-2:** Memory Corruption

**External Learning-**Implementing simple buffer overflows.

**Aim:**

To implements a c program that has buffer overflow vulnerability and use that vulnerability to change the value of a function pointer to point to different address thereby executing some other method instead of normal execution.

**Tools-used:**

1.Linux Terminal

2.GCC compiler.

3.GNU Debugger.

**Description:**

Buffer overflow occurs when the input is not validated and we use gets() function to read input from the user and store it in a buffer , during the gets function takes whatever the user types into the buffer, if the buffer fills it starts overflowing causing buffer overflow which overwrites the values in the stack, leading to crashing of the program or if used in proper way can exploit to make the program do the specified task.

**Input:** Input to the buffer along with the address of the malicious function in the code in hexadecimal format.

**Output:** The execution of the malicious method which will not execute in normal case.

**C Program:**

**Buff.c**

#include<stdio.h>

#include<string.h>

void bad(){

    printf("\nmalicious code executed\n");

}

void good(){

    printf("\nNormal execution\n");

}

void main(){

    void (\*fp)(void)=good;

    char buffer[10];

    gets(buffer);

    printf("fp is pointing to %p\n",fp);

    fp();

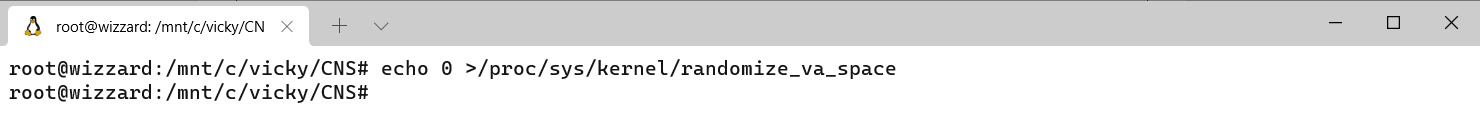
    return;

}

**Execution:**

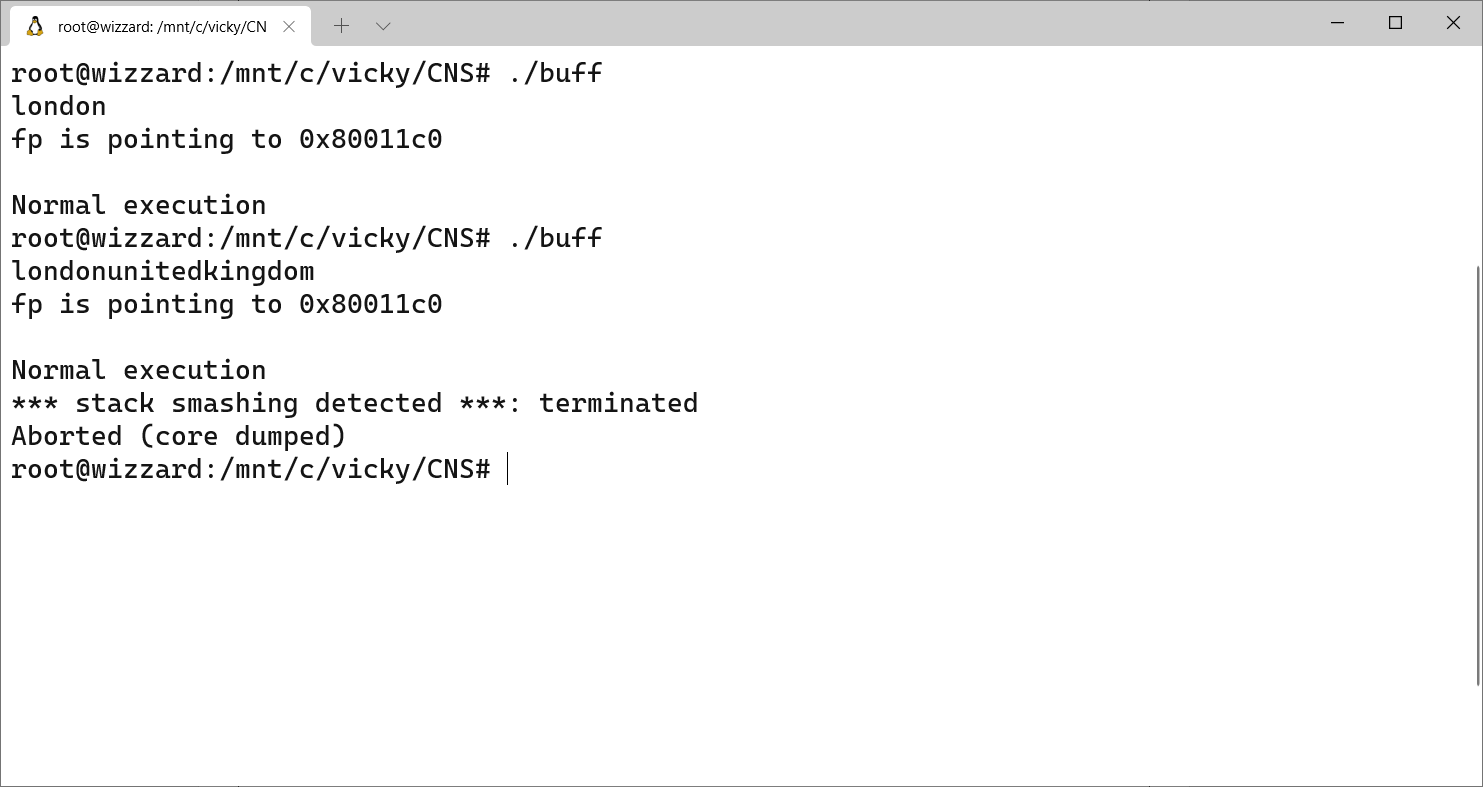
**Step:1**

Address space layout randomization may make it difficult to make our program execute what we are aiming so we must turn it off.

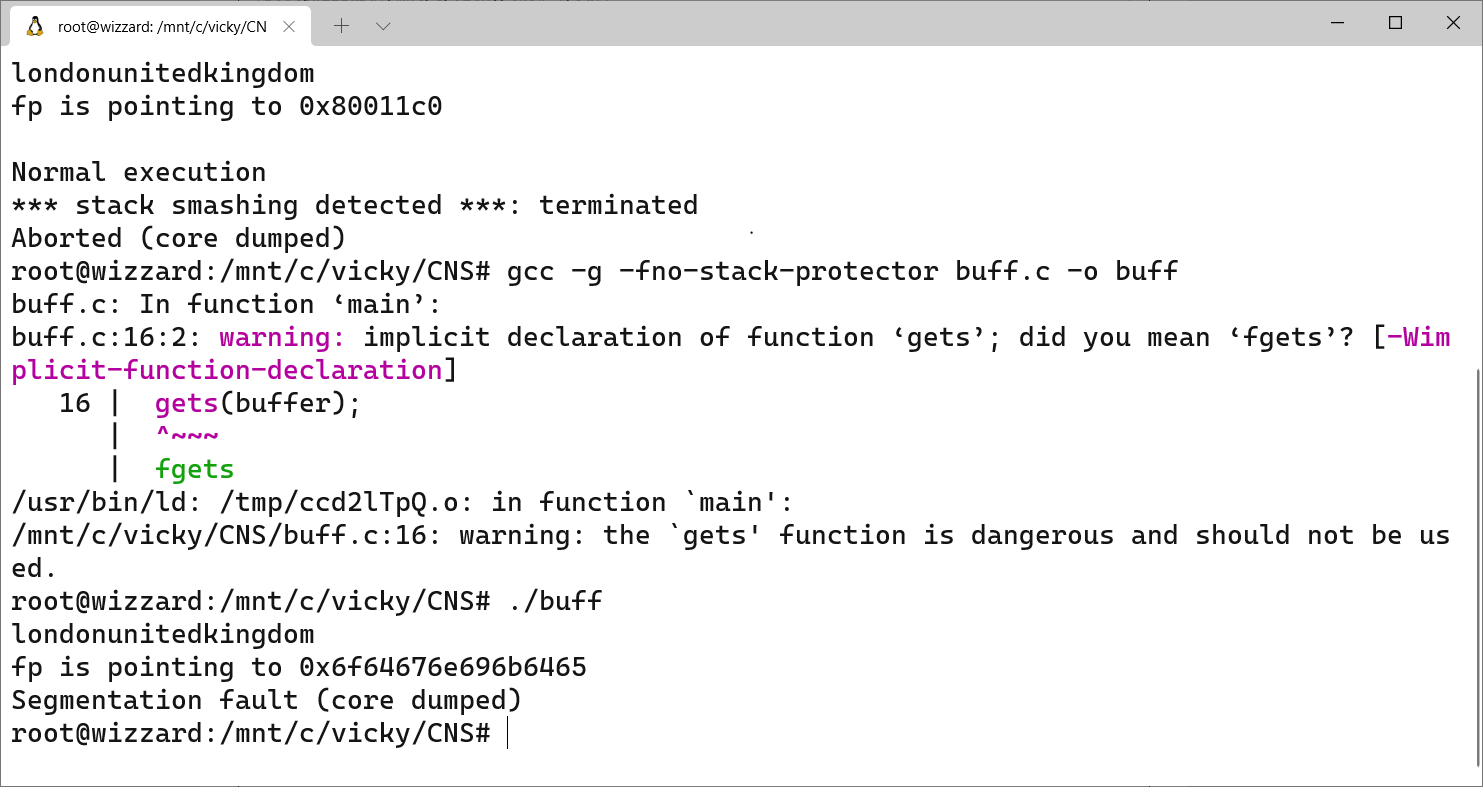


In our program we have function pointer fp pointing to function good, and buffer of size 10, gets read input from user and store it in buffer, when we give input within the size no problem occurs if we given more than the size buffer overflows and rewritesin stack that’s why stack smashing occurs.

We are printing where fp points to in our program.



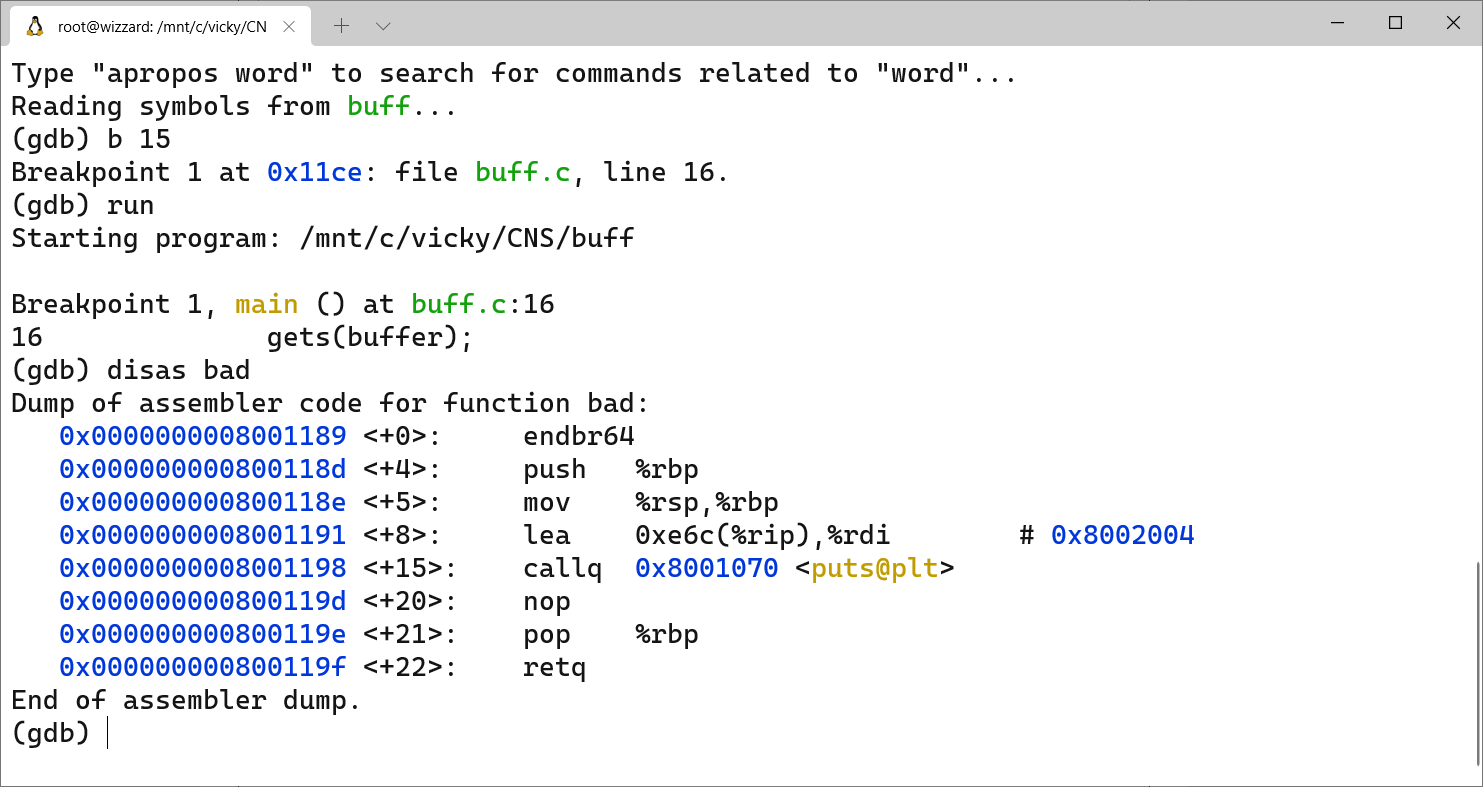
**Step:2**

In order to prevent stack smashing and overwrite stack values, we must stop stack protector by using **-fno-stack-protector** during compilation.

We can see no stack smashing happened and the notice that the value of stack pointer is being overwritten by the values what we give since the local variables (say fp) is also placed in stack, so when stack is being overwritten by buffer overflow values it gets changes.

**Step-3:**

When we go into the gdb mode of the executable buff and keep break point anywhere and try to run the program and disassemble the bad function, in our code we get the starting address of the bad function.



The starting address of bad function is 0x0000000008001189 , we have disabled Address space layout randomization(ASLR) this starting address wont change, so using buffer overflow and overwriting the function pointer with this address we can execute the bad function.

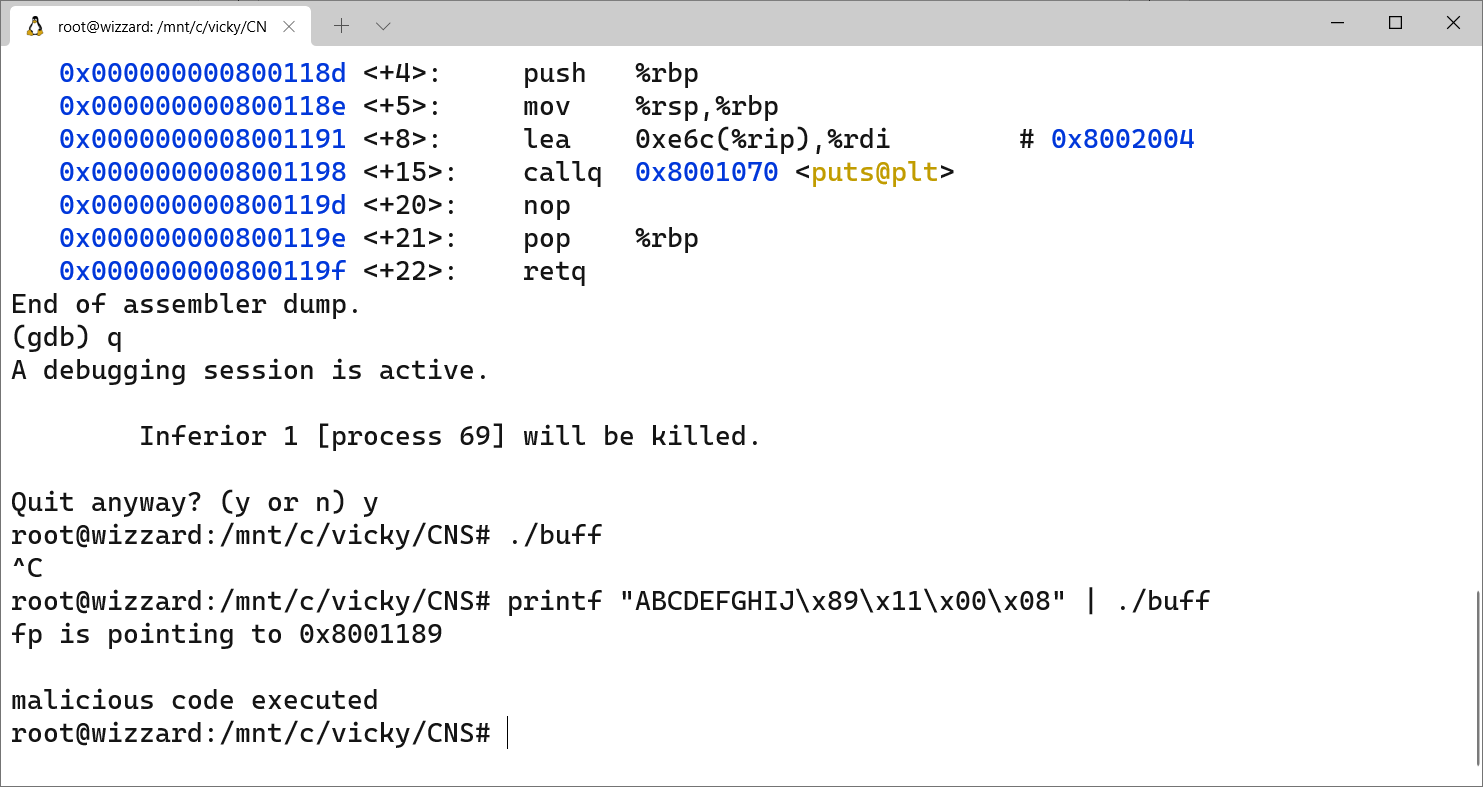
**Printf function man page shows:**

\xHH byte with hexadecimal value HH (1 to 2 digits)

**Step-4:**

In order to give this hexadecimal value as input we use printf function which used with **\x** will specify the value as hexadecimal , we need to correctly overwrite the value fp , which should be given in reverse order say,

is 0x0000000008001189 should be written as \x89\x11\x00\x08



The value ABCDEFGHIJ is written to pad up until the overflow of buffer reaches the exact location where the fp is stored in stack.

We can see that the fp points to 0x8001189 exactly the same location we see as the address of bad function in GDB mode. We have successfully overwritten fp to point to bad function instead of good function (initially what it had).So the printf statement in bad function is displayed in terminal.

**Conclusion:**

Buffer overflow can be used to exploit our program, in order to prevent this and make our code secured we must follow several steps such as

Validate the input, address space layout randomization should be followed which can the address of the stack every time so making it difficult find any address and importantly use **fgets()** instead of gets(),**strncmp()** instead **strcmp().**