Experiment No:6

**Aim**: To implement a code to simulate buffer overflow attack

**Theory**:

A buffer is a temporary area for data storage. When more data (than was originally allocated to be stored) gets placed by a program or system process, the extra data overflows. It causes some of that data to leak out into other buffers, which can corrupt or overwrite whatever data they were holding.

The buffer overflow attack was discovered in hacking circles. It uses input to a poorly implemented, but (in intention) completely harmless application, typically with root / administrator privileges. The buffer overflow attack results from input that is longer than the implementor intended. To understand its inner workings, we need to talk a little bit about how computers use memory. In a buffer-overflow attack, the extra data sometimes holds specific instructions for actions intended by a hacker or malicious user; for example, the data could trigger a response that damages files, changes data or unveils private information.

Attacker would use a buffer-overflow exploit to take advantage of a program that is waiting on a user’s input. There are two types of buffer overflows: stack-based and heap-based. Heap-based, which are difficult to execute and the least common of the two, attack an application by flooding the memory space reserved for a program. Stack-based buffer overflows, which are more common among attackers, exploit applications and programs by using what is known as a stack: memory space used to store user input.

When a buffer overflow occurs in a program, it will often crash or become unstable. An attacker attempting to abuse a buffer overflow for a more specific purpose other than crashing the target system, can purposely overwrite important values in the call stack of the target machine such as the instruction pointer (IP) or base pointer (BP) in order to execute his or her potentially malicious unsigned code. Operating system and software vendors often employ countermeasures in their products to prevent Buffer Overflow Attacks; particularly call stack and virtual memory randomization. Given the existence of such protective measures, Buffer Overflow Attacks have been rendered more difficult, although still possible to carry out.

**Program**:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int main(int argc, char \*argv[])

{

char mybuffer[5];

if (argc < 2)

{

printf("strcpy() NOT executed....\n");

printf("Syntax: %s <characters>\n", argv[0]);

exit(0);

}

strcpy(mybuffer, argv[1]);

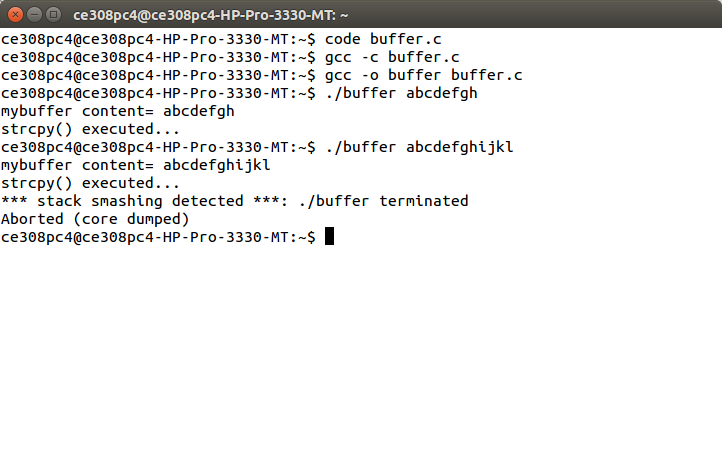
printf("mybuffer content= %s\n", mybuffer);

printf("strcpy() executed...\n");

return 0;

}

**Output**:



**Conclusion**: Hence a program to simulate buffer overflow is studied and executed.