Experiment 10

Date of Performance: 7 May 2023 **Date of Submission:** 8 May 2023

SAP Id: 60004200132 **Name :** Ayush Jain

Div: B **Batch :** B3

Aim of Experiment

To study and implement Buffer Overflow attack.

Theory / Algorithm / Conceptual Description:

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. 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: stackbased 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.

2 Types of Buffer Overflow:

A. Heap Overflow:

Heap is a region of process's memory which is used to store dynamic variables. These variables are allocated using malloc() and calloc() functions and resize using realloc() function, which are inbuilt functions of C. These variables can be accessed globally and once we allocate memory on heap it is our responsibility to free that memory space after use.

B. Stack Overflow:

Stack is a special region of our process's memory which is used to store local variables used inside the function, parameters passed through a function and their return addresses. Whenever a new local variable is declared it is pushed onto the stack. All the variables associated with a function are deleted and memory they use is freed up, after the function finishes running. The user does not have any need to free up stack space manually. Stack is Last-In-First-Out data structure. In our computer's memory, stack size is limited. If a program uses more memory space than the stack size then stack overflow will occur and can result in a program crash.

Program

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

{
    // should allocate 8 bytes = 2 double words,      // To

overflow, need more than 8 bytes...

    char buffer[5];       if

(argc < 2)

    {
        printf("strcpy() NOT executed....\n");          printf("Syntax: %s

<characters>\n", argv[0]);           exit(0);

    }
    strcpy(buffer, argv[1]);      printf("buffer content=

%s\n", buffer);      printf("strcpy() executed...\n");



    return 0;
}
```

Output

Input: 12345678 (8 bytes), the program run smoothly.

Input: 123456789 (9 bytes)
"Segmentation fault" message will be displayed and the program terminates.

Conclusion: Thus, we successfully studied and implemented Buffer Overflow Attack.