Somaiya Vidyavihar University
K. J. Somaiya College of Engineering
Department of Computer Engineering

Batch: B1 Roll No.: 16010121045

Experiment No. 8

Title: Buffer Overflow Vulnerability

Objective:
Buffer Overflow Vulnerability

CO	Outcome
CO3	Comprehend post exploitation phase of penetration testing.

Books/ Journals/ Websites referred:

https://www.imperva.com/learn/application-security/buffer-overflow/

Abstract:-

A buffer overflow attack is a type of security vulnerability that occurs when an attacker inputs more data into a buffer or memory area than it can handle, overwriting adjacent memory locations. This can cause unexpected behavior or crashes and can potentially allow the attacker to execute malicious code. Buffer overflow attacks can occur in software written in any programming language and are typically caused by errors in code such as using insufficient bounds checking or failing to properly validate input data. To prevent buffer overflow attacks, developers should follow best practices such as using secure coding techniques, performing input validation, and using safer programming languages and libraries.

Related Theory: -

Buffer overflow errors are characterized by the overwriting of memory fragments of the process, which should have never been modified intentionally or unintentionally. Overwriting values of the IP (Instruction Pointer), BP (Base Pointer) and other registers causes exceptions, segmentation faults, and other errors to occur. Usually, these errors end execution of the application in an unexpected way. Buffer overflow errors occur when we operate on buffers of char type.

Buffer overflows can consist of overflowing the stack [Stack overflow] or overflowing the heap [Heap overflow]. For example, a buffer for log-in credentials may be designed to expect username and password inputs of 8 bytes, so if a transaction involves an input of 10 bytes (that is, 2 bytes more than expected), the program may write the excess data past the buffer boundary. Buffer overflows can affect all types of software. They typically result from malformed inputs or failure to allocate enough space for the buffer. If the transaction overwrites executable code, it can cause the program to behave unpredictably and generate incorrect results, memory access errors, or crashes.

Let's understand with the help of an example:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void main()
{
    char *ptr;
    char *dptr;
    ptr = (char *)malloc(10 * sizeof(char));
    dptr = (char *)malloc(10 * sizeof(char));
    printf("Address of ptr: %d\n", ptr);
    printf("Address of dptr: %d\n", dptr);
    printf("Enter The String:\n");
    gets(ptr);
    system(dptr);
}
```

The code you provided has a potential buffer overflow vulnerability due to the use of the **gets** function. The **gets** function is inherently unsafe because it does not perform any bounds checking on the input, and it can easily lead to buffer overflows.

Hence an attacker can execute commands directly by storing the commands in the dptr as it gets executed as a system command. In the below example I have executed the **ls -a** command which displays all the stored files in the directory.

```
❯ cd "/Users/pargatsinghdhanjal/Documents/College/Sem6/IS/Programs/" && gcc exp2.c -o exp2 && "/Users/pargat
singhdhanjal/Documents/College/Sem6/IS/Programs/"exp2
Address of ptr: 904937936
Address of dptr: 904937952
Enter The String:
warning: this program uses gets(), which is unsafe.
pargatsinghdhanjls -a
.       ..      exp2    exp2.c
```

Conclusion:- Learnt to use and implement Buffer overflow concept.