**Branch :- Computer Sci. & Engg. Class :- Final Year**

**Subject :-System and Software Security Lab manual Sem :- VII**

**Teacher Manual**

**PRACTICAL NO 3**

**AIM**:- Analyze the unintentional malicious effect of buffer overflow for string data

**S/W REQUIRED:-** GCC Compiler

**THEORY:-**

Buffer overflow is a software coding error or vulnerability that can be exploited by hackers to gain unauthorized access to corporate systems. It is one of the best-known software security vulnerabilities yet remains fairly common. This is partly because buffer overflows can occur in various ways and the techniques used to prevent them are often error-prone.

The software error focuses on buffers, which are sequential sections of computing memory that hold data temporarily as it is transferred between locations. Also known as a buffer overrun, buffer overflow occurs when the amount of data in the buffer exceeds its storage capacity. That extra data overflows into adjacent memory locations and corrupts or overwrites the data in those locations.

**What Is a Buffer Overflow Attack?**

A buffer overflow attack takes place when an attacker manipulates the coding error to carry out malicious actions and compromise the affected system. The attacker alters the application’s execution path and overwrites elements of its memory, which amends the program’s execution path to damage existing files or expose data.

A buffer overflow attack typically involves violating programming languages and overwriting the bounds of the buffers they exist on. Most buffer overflows are caused by the combination of manipulating memory and mistaken assumptions around the composition or size of data.

A buffer overflow vulnerability will typically occur when code:

Is reliant on external data to control its behavior

Is dependent on data properties that are enforced beyond its immediate scope

Is so complex that programmers are not able to predict its behavior accurately

**Buffer Overflow Exploits**

The buffer overflow exploit techniques a hacker uses depends on the architecture and operating system being used by their target. However, the extra data they issue to a program will likely contain malicious code that enables the attacker to trigger additional actions and send new instructions to the application.

For example, introducing additional code into a program could send it new instructions that give the attacker access to the organization’s IT systems. In the event that an attacker knows a program’s memory layout, they may be able to intentionally input data that cannot be stored by the buffer. This will enable them to overwrite memory locations that store executable code and replace it with malicious code that allows them to take control of the program.

Attackers use a buffer overflow to corrupt a web application’s execution stack, execute arbitrary code, and take over a machine. Flaws in buffer overflows can exist in both application servers and web servers, especially web applications that use libraries like graphics libraries. Buffer overflows can also exist in custom web application codes. This is more likely because they are given less scrutiny by security teams but are less likely to be discovered by hackers and more difficult to exploit.

Buffer Overflow Consequences

Common consequences of a buffer overflow attack include the following:

* System crashes: A buffer overflow attack will typically lead to the system crashing. It may also result in a lack of availability and programs being put into an infinite loop.
* Access control loss: A buffer overflow attack will often involve the use of arbitrary code, which is often outside the scope of programs’ security policies.
* Further security issues: When a buffer overflow attack results in arbitrary code execution, the attacker may use it to exploit other vulnerabilities and subvert other security services.

**Algorithm**

1. Reading from the stack.

2. Writing to the stack.

3. Format String Attack Detection & Defenses

**Program:**

#include <stdio.h>

#include <string.h>

int main(void)

{    char buff[15];

    int pass = 0;

    printf("\n Enter the password : \n");

gets(buff);

if(strcmp(buff, "thegeekstuff"))

    {        printf ("\n Wrong Password \n");

}

else

{    printf ("\n Correct Password \n");

        pass = 1;

    }

    if(pass)

    {       /\* Now Give root or admin rights to user\*/

        printf ("\n Root privileges given to the user \n");

    }

    return 0;

}

**Output**

Enter the password :

thegeekstuff

 Correct Password

 Root privileges given to the user

This works as expected. The passwords match and root privileges are given.

There is a possibility of buffer overflow in this program. The gets() function does not check the array bounds and can even write string of length greater than the size of the buffer to which the string is written.

Here is an example :

$ ./bfrovrflw

 Enter the password :

hhhhhhhhhhhhhhhhhhhh

 Wrong Password

 Root privileges given to the user

In the above example, even after entering a wrong password, the program worked as if you gave the correct password.

There is a logic behind the output above. What attacker did was, he/she supplied an input of length greater than what buffer can hold and at a particular length of input the buffer overflow so took place that it overwrote the memory of integer ‘pass’. So despite of a wrong password, the value of ‘pass’ became non zero and hence root privileges were granted to an attacker.

**CONCLUSION:** Thuswe have analyzed the unintentional malicious effect of buffer overflow for string data.