# **FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING**

**Department of Computer Engineering**

# **Course, Subject & Experiment Details**

|  |  |
| --- | --- |
| **Practical No:** |  |
| **Title:** | **Buffer Overflow** |
| **Name of the Student:** | **Warren Fernandes** |
| **Roll No:** | **8940** |
| **Date of Performance:** | **21-03-2022** |
| **Date of Submission:** | **09-04-2022** |

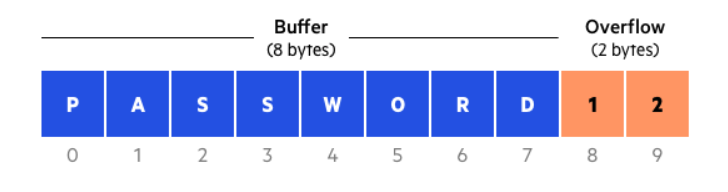
Evaluation:

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Rubric** | **Grade** |
| **1** | **On time submission/completion (2)** |  |
| **2** | **Preparedness (2)** |  |
| **3** | **Skill (4)** |  |
| **4** | **Output (2)** |  |

# **Signature of the Teacher**

**Buffer Overflow:**

Buffers are memory storage regions that temporarily hold data while it is being transferred from one location to another. A buffer overflow (or buffer overrun) occurs when the volume of data exceeds the storage capacity of the memory buffer. As a result, the program attempting to write the data to the buffer overwrites adjacent memory locations. For example, a buffer for log-in credentials may be designed to expect username and password inputs of 7 bytes, so if a transaction involves an input of 9 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.

****

#### Buffer Overflow Attacks:

Attackers exploit buffer overflow issues by overwriting the memory of an application. This changes the execution path of the program, triggering a response that damages files or exposes private information. For example, an attacker may introduce extra code, sending new instructions to the application to gain access to IT systems.

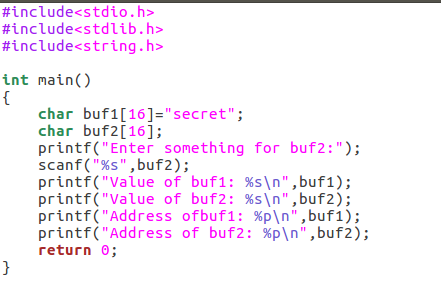
If attackers know the memory layout of a program, they can intentionally feed input that the buffer cannot store, and overwrite areas that hold executable code, replacing it with their own code. For example, an attacker can overwrite a pointer (an object that points to another area in memory) and point it to an exploit payload, to gain control over the program.

#### Types of Buffer Overflow Attacks:

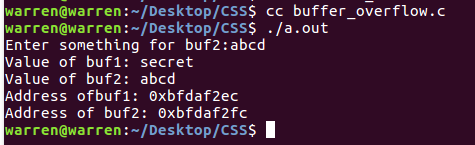
* Stack-based buffer overflows are more common, and leverage stack memory that only exists during the execution time of a function.
* Heap-based attacks are harder to carry out and involve flooding the memory space allocated for a program beyond memory used for current runtime operations.

#### Program Implementation:

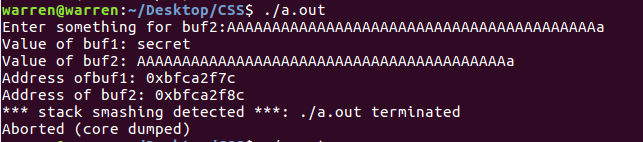
##### Program 1:



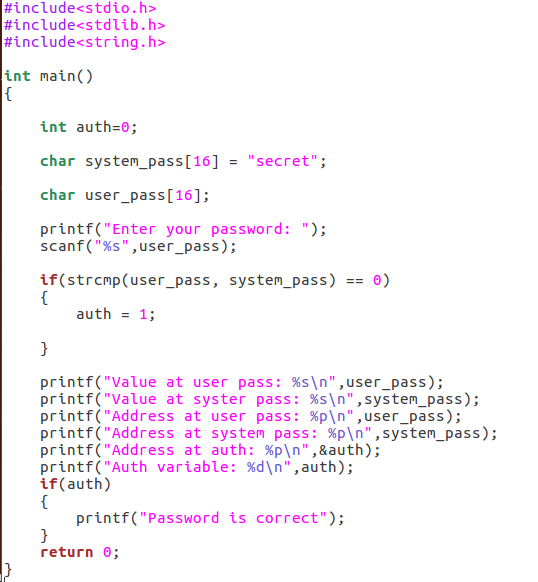
Here we have created two strings of size 16 bytes each and printed the value of strings and the addresses of strings.



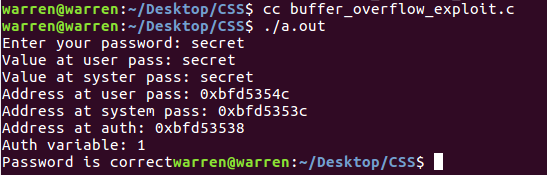
Here as the size of input is less than 16 bytes the program there is no problem of buffer overflow and when the size of input is more than 16 bytes the remaining string is saved in adjacent address. In this case the address of “buf1” and buf2” are adjacent so the input string will overwrite the value of buf1.



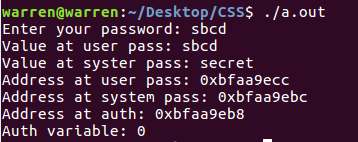
##### Program 2:



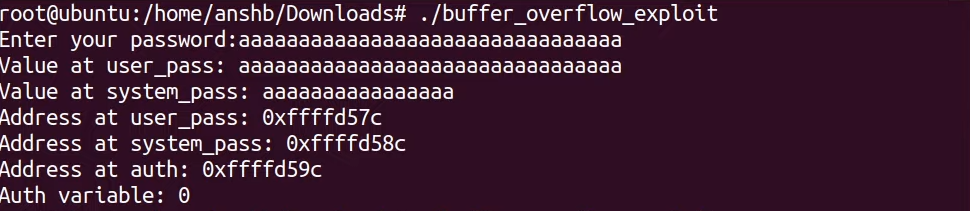
In addition to previous program we have created auth variable and we are comparing both the strings if both the strings are same then auth variable is set to 1 and if auth is not 0 it will print password is correct.



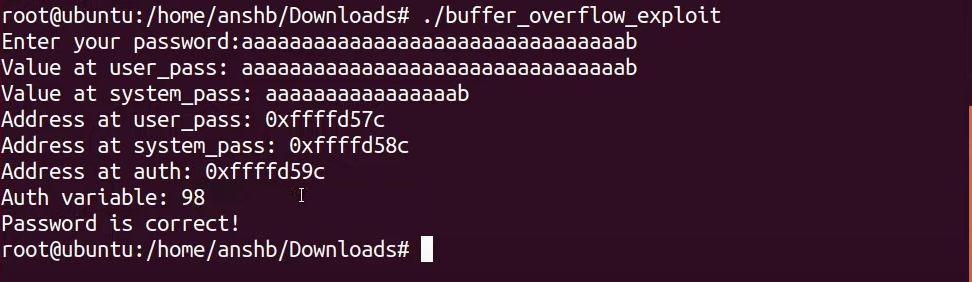
Here we have entered correct password and in output it shows password is correct but in addition we can also see that the address of all the three variables are adjacent so it will be easy to implement buffer overflow on this program.



Here we have entered incorrect password and the value of auth variable is 0 so it shows that password doesn’t matches

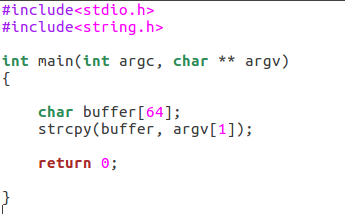


Here we have executed buffer overflow but we have not reached the address of auth variable so we it doesn’t prints “Password is correct”.



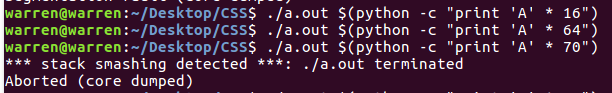
Here we have overwritten the value of auth variable i.e. ASCII value of b and it shows password is correct.

##### Program 3:



Here we have created string of 64 bytes and we will give the input in command line and then copy the input to thee buffer string.

So



Here when we have given input using python command line and in 1st three cases code has been executed successfully and in 4th case we got segmentation fault that shows from the user input at ehat offset the return address is located and then hacker can get access to shell and use the machine.

#### Postlab:

1. How to Prevent Buffer Overflows?

Developers can protect against buffer overflow [vulnerabilities](https://www.imperva.com/learn/application-security/vulnerability-management/) via security measures in their code, or by using languages that offer built-in protection.

In addition, modern operating systems have runtime protection. Three common protections are:

**Address space randomization (ASLR)**—randomly moves around the address space locations of data regions. Typically, buffer overflow attacks need to know the locality of executable code, and randomizing address spaces makes this virtually impossible.

**Data execution prevention**—flags certain areas of memory as non-executable or executable, which stops an attack from running code in a non-executable region.

**Structured exception handler overwrite protection (SEHOP)**—helps stop malicious code from attacking Structured Exception Handling (SEH), a built-in system for managing hardware and software exceptions. It thus prevents an attacker from being able to make use of the SEH overwrite exploitation technique. At a functional level, an SEH overwrite is achieved using a stack-based buffer overflow to overwrite an exception registration record, stored on a thread’s stack.

1. What Programming Languages are More Vulnerable?

C and C++ are two languages that are highly susceptible to buffer overflow attacks, as they don’t have built-in safeguards against overwriting or accessing data in their memory. Mac OSX, Windows, and Linux all use code written in C and C++.

Languages such as PERL, Java, JavaScript, and C# use built-in safety mechanisms that minimize the likelihood of buffer overflow.