4-2 Milestone Three: Enhancement Two: Algorithms and Data Structure

Victoria Franklin

Southern New Hampshire University

CS-499 Computer Science Capstone

Professor Gene Bryant

Sunday, July 28, 2024

CS 499 Milestone Three

The BufferOverflow.cpp artifact from CS 405: Secure Coding is a pivotal piece I've chosen to enhance. It stands as a prime example of a runtime error, specifically a buffer overflow, which occurs when a program writes beyond the end of an array or buffer, leading to the corruption of neighboring memory. This artifact is not just significant but crucial, as it vividly showcases my profound understanding of such vulnerabilities, which are often triggered by unexpected user input.

My ePortfolio must include CS 405: Secure Coding, including the BufferOverflow.cpp artifact, for multiple reasons. Buffer overflows are among the most severe software security issues, and the BufferOverflow.cpp artifact demonstrates my knowledge of them. A key component of secure coding methods is a theoretical concept and a practical tool demonstrating my capacity to detect, evaluate, and address these issues in real-world situations. It reflects my expertise in coding securely and implementing safeguards to avoid typical security holes. A high level of problem-solving knowledge is required to locate and repair a buffer overflow vulnerability. This artifact showcases my systematic approach to analyzing security issues, identifying their causes, and implementing successful solutions. Standardization bodies and recommended practices for safe coding have extensively addressed buffer overflow problems. This artifact shows that I am prepared to work in a professional environment and that my skills are up to par with what is expected in the industry regarding secure software development.

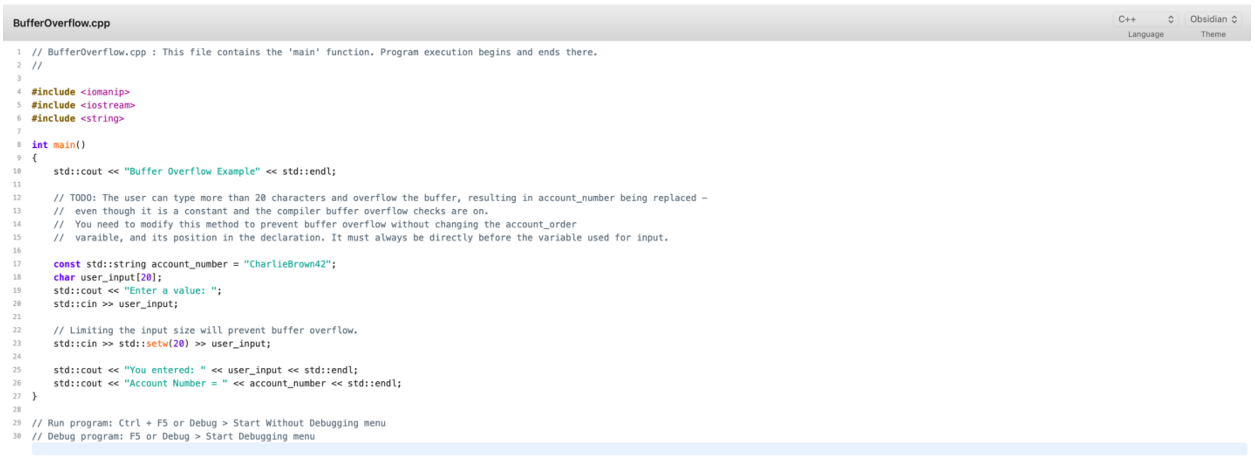
A thorough familiarity with memory management and low-level programming is necessary for dealing with buffer overflow problems. I have demonstrated my technical skills in these areas with my artifact, which is essential for any position that requires safe software development. This item proves I have worked with secure coding in the real world. To strengthen my ePortfolio and establish my credibility, it offers concrete evidence that I have participated in hands-on activities and projects to improve my abilities. I continuously strive to improve and learn, and I can describe the learning process I went through in CS 405 by including the BufferOverflow.cpp artifact. In order to put my skills and experiences into perspective, I can talk about the goals of my learning, the difficulties I encountered, and the information I acquired.

The BufferOverflow.cpp artifact is being enhanced to prevent buffer overflow by handling input safely and providing user feedback when input is erroneous. This involves ensuring that the input size doesn't exceed the buffer size. The user is notified if the input size is larger than the buffer. A for-loop repeatedly asks the user for the correct information, ensuring that any lingering incorrect data in the input stream is precisely handled. The proper way to handle null terminators is to use std::cin.getline as your input reader. The input stream's status is checked to determine if it was interrupted. The user is constantly prompted to provide valid data to avoid a buffer overflow. Std::cin.clear() and std::cin.ignore() and remove erroneous input from the stream. This process prevents buffer overflow and demonstrates my proficiency in secure code, capacity to address security problems, and dedication to industry standards, making the BufferOverflow.cpp artifact is a valuable asset to my ePortfolio.

I did what I set out to do in the first module by improving the BufferOverflow.cpp artifact. This artifact is not just a part of my ePortfolio but a significant piece that helped me achieve the course objectives. My outcome-coverage plans have been updated, and the targets addressed are detailed below. In Module One, I improved the BufferOverflow.cpp artifact to show that I understood security vulnerabilities, how to use secure coding practices, and how to solve problems. This allowed me to achieve the course objectives. My future intentions for result coverage center on practical applications, detailed documentation, incremental upgrades, a more comprehensive range of security ideas, and interactive demos. Thanks to these revisions, my ePortfolio will now more properly display my expertise in secure coding.

I learned a lot from improving and altering the BufferOverflow.cpp artifact. It revealed that secure coding practices, comprehensive testing, and documentation are critical to preventing buffer overflow vulnerabilities. My difficulties, which included finding hidden vulnerabilities, juggling performance and security, and dealing with complicated edge circumstances, aided my development as a software engineer. In sum, I am more equipped to handle security challenges in actual software development thanks to this experience, which has also improved my skills in secure coding.

Original BufferOverflow.cpp

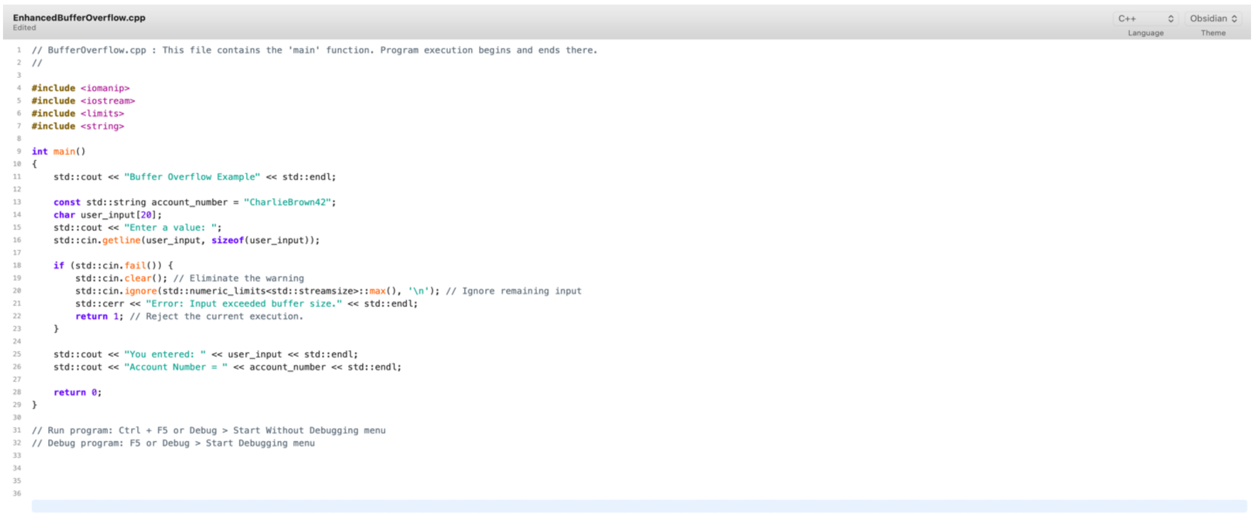


BufferOverflow.cpp

C++ Language | Obsidian Theme

```cpp
// BufferOverflow.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include <iomanip>
#include <iostream>
#include <string>

int main()
{
    std::cout << "Buffer Overflow Example" << std::endl;

    // TODO: The user can type more than 20 characters and overflow the buffer, resulting in account_number being replaced -
    //  even though it is a constant and the compiler buffer overflow checks are on.
    //  You need to modify this method to prevent buffer overflow without changing the account_order
    //  varaible, and its position in the declaration. It must always be directly before the variable used for input.

    const std::string account_number = "CharlieBrown42";
    char user_input[20];
    std::cout << "Enter a value: ";
    std::cin >> user_input;

    // Limiting the input size will prevent buffer overflow.
    std::cin >> std::setw(20) >> user_input;

    std::cout << "You entered: " << user_input << std::endl;
    std::cout << "Account Number = " << account_number << std::endl;
}

// Run program: Ctrl + F5 or Debug > Start Without Debugging menu
// Debug program: F5 or Debug > Start Debugging menu
```

EnhancedBufferOverflow



EnhancedBufferOverflow.cpp
Edited

C++ Language | Obsidian Theme

```cpp
// BufferOverflow.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include <iomanip>
#include <iostream>
#include <limits>
#include <string>

int main()
{
    std::cout << "Buffer Overflow Example" << std::endl;

    const std::string account_number = "CharlieBrown42";
    char user_input[20];
    std::cout << "Enter a value: ";
    std::cin.getline(user_input, sizeof(user_input));

    if (std::cin.fail()) {
        std::cin.clear(); // Eliminate the warning
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); // Ignore remaining input
        std::cerr << "Error: Input exceeded buffer size." << std::endl;
        return 1; // Reject the current execution.
    }

    std::cout << "You entered: " << user_input << std::endl;
    std::cout << "Account Number = " << account_number << std::endl;

    return 0;
}

// Run program: Ctrl + F5 or Debug > Start Without Debugging menu
// Debug program: F5 or Debug > Start Debugging menu
```