Douglas Rowland

CS-405

Buffer Overflow

A buffer overflow happens when a program tries to store more data in a memory buffer than it was designed to hold. This can overwrite adjacent data, leading to unintended behavior or security vulnerabilities. I modified the program to ensure users can safely input up to 20 characters without causing an overflow. If more than 20 characters are entered, the program automatically trucates the excess and warns the user, preventing any potential issues.

Technical Approach:

1. Input Handling with std::cin.get:  
   The std::cin.get function was used to safely read up to 20 characters into the buffer, leaving room for a null terminator to properly terminate the string.
2. Overflow Detection and Prevention:  
   After reading the input, the program checks if any additional characters remain in the input stream using std::cin.peek(). If so, it clears the error state with std::cin.clear() and discards the excess input using std::cin.ignore().
3. User Notification:  
   If the input exceeds 20 characters, the program displays a warning message to inform the user that the extra characters were discarded.
4. Preservation of account\_number:  
   By preventing input from exceeding the buffer size, the account\_number variable, which resides in memory adjacent to the buffer, remains unchanged and secure.

This ensures that the input is safely contained within the bounds of the buffer, eliminating the risk of buffer overflow. It preserves the integrity of other variables and provides clear feedback to the user if their input is too long.

Issues Encountered and Resolutions:

* Issue: Ensuring the program could handle excessive input without truncating valid input or affecting adjacent variables.  
  Resolution: Used std::cin.get to limit input size and implemented error-handling logic to discard extra characters.
* Issue: Avoiding changes to the account\_number or buffer sizes, as required by the assignment.  
  Resolution: The buffer was correctly sized to 21 bytes (20 for input and 1 for the null terminator) without altering variable placement.

The main challenge was making the program safely handle excessive input without affecting adjacent variables or altering the account\_number or buffer sizes. I resolved this by using std::cin.get to limit input size and discarding extra characters while preserving variable integrity.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated