Secure Coding Review Report – Task 3

Language Used: C++

Project Name: Event Management System

Lines of Code Reviewed: ~400

Review Type: Manual inspection

# *Identified Vulnerabilities and Weaknesses:*

|  |  |  |  |
| --- | --- | --- | --- |
| # | Vulnerability | Why it’s a Problem | Suggested Fix |
| 1 | No input validation | cin >> variable accepts any input, including invalid data | Validate inputs using conditions (e.g., check for positive costs, non-empty strings) |
| 2 | No bounds checking | User can enter an invalid event number (e.g., negative or > 10) | Check if input is within 0 <= eventNumber < eventCount |
| 3 | No file/database storage | All event data is lost when program closes | Use file I/O (e.g., ofstream/ifstream) to save/load events |
| 4 | System commands (system("cls"), system("color")) | Can be exploited or break in different OS environments | Use OS-specific safe alternatives or avoid system calls entirely |
| 5 | No error handling for cin | If user enters a character instead of an integer, program may crash | Use cin.fail() check and cin.clear() to handle bad input |

# *Best Practice Recommendations:*

- Always validate and sanitize user input

- Use functions to validate numeric range, non-empty text, and proper format

- Add file handling (text or binary) to persist events between sessions

- Avoid system() commands for portability and safety

- Use try-catch where applicable (e.g., in future implementations using file handling or advanced modules)

# *Sample Fix (Input Validation Example):*

Before:

cin >> evt[eventCount].foodCost;

After:

double food;  
cout << "Enter food cost: ";  
cin >> food;  
while (cin.fail() || food < 0) {  
 cin.clear();  
 cin.ignore(1000, '\n');  
 cout << "Invalid input. Please enter a valid positive number: ";  
 cin >> food;  
}  
evt[eventCount].foodCost = food;

# *Tools Used*:

- Manual inspection  
- C++ basic I/O validation methods

# *Conclusion:*

Your C++ Event Management System works well for a local console application. However, for real-world secure software, basic input validation, data persistence, and avoidance of system-level commands are essential. Adding these practices will make your system more robust and professional.