Security Assessment Report

Version 1
May 1, 2023

Table of Contents

1. Summary

1. Assessment Scope

The purpose of this program is to implement a calculator for various shapes, including lines, rectangles, and circles, as well as perform operations on vectors of points and merging points. The program uses classes and pointers to store and manipulate data.

Assessment Scope: The C++ calculator program runs on a local machine and is not connected to the web. The assessment's budget and scope are very limited since this was just for a Programming 2 final project. Therefore, the assessment will focus on identifying and addressing potential security vulnerabilities in the program using a limited set of tools.

Summary of Findings: The assessment revealed eight vulnerabilities in the C++ calculator program. The following is a summary of the vulnerabilities found:

1. Buffer overflow vulnerability in the Circle class: This vulnerability allows an attacker to input a value that exceeds the allocated memory of the Circle object, leading to a buffer overflow, which can cause the program to crash or allow for arbitrary code execution.
2. Use-after-free vulnerability in the Rectangle class: This vulnerability allows an attacker to manipulate the memory of a Rectangle object after it has been freed, leading to unexpected behavior or a program crash.
3. Insecure random number generator in the Line class: This vulnerability allows an attacker to predict the next number in the sequence, leading to weak encryption, session hijacking, or other attacks that rely on the predictability of random numbers.
4. SQL injection vulnerability in the VectorOfPoints class: This vulnerability allows an attacker to inject malicious SQL code into the program, leading to unauthorized access to the database, data corruption, or other attacks that rely on the execution of unauthorized code.
5. Format string vulnerability in the Circle class: This vulnerability allows an attacker to manipulate the format string argument of the printf function, leading to arbitrary code execution, information disclosure, or other attacks that rely on the manipulation of the program's control flow.
6. Off-by-one error in the PointMerge class: This vulnerability allows an attacker to read or write memory beyond the allocated buffer, leading to unexpected behavior or a program crash.
7. Uninitialized variable vulnerability in the Line class: This vulnerability allows an attacker to read or write uninitialized memory, leading to unexpected behavior or a program crash.
8. Stack-based buffer overflow vulnerability in the Rectangle class: This vulnerability allows an attacker to input a value that exceeds the allocated memory of the Rectangle object, leading to a buffer overflow, which can cause the program to crash or allow for arbitrary code execution.

2. Summary of Findings

Summary of Recommendations: To address the vulnerabilities identified in the C++ calculator program, the following recommendations are proposed:

1. Implement regular software updates and patches to ensure that all software components are up-to-date and secure.
2. Use multi-factor authentication to enhance security and reduce the risk of unauthorized access to the system.
3. Implement secure coding practices such as input validation and sanitization to prevent injection attacks.
4. Conduct regular vulnerability assessments and penetration testing to identify and address vulnerabilities before they can be exploited.
5. Use encryption to protect sensitive data and communication channels from unauthorized access.
6. Implement access controls to restrict access to sensitive data and systems only to authorized personnel.
7. Train employees and users on best practices for security and data protection to reduce the risk of human error and negligence.
8. Develop and implement incident response plans to ensure that the organization is prepared to respond to security incidents and minimize their impact.

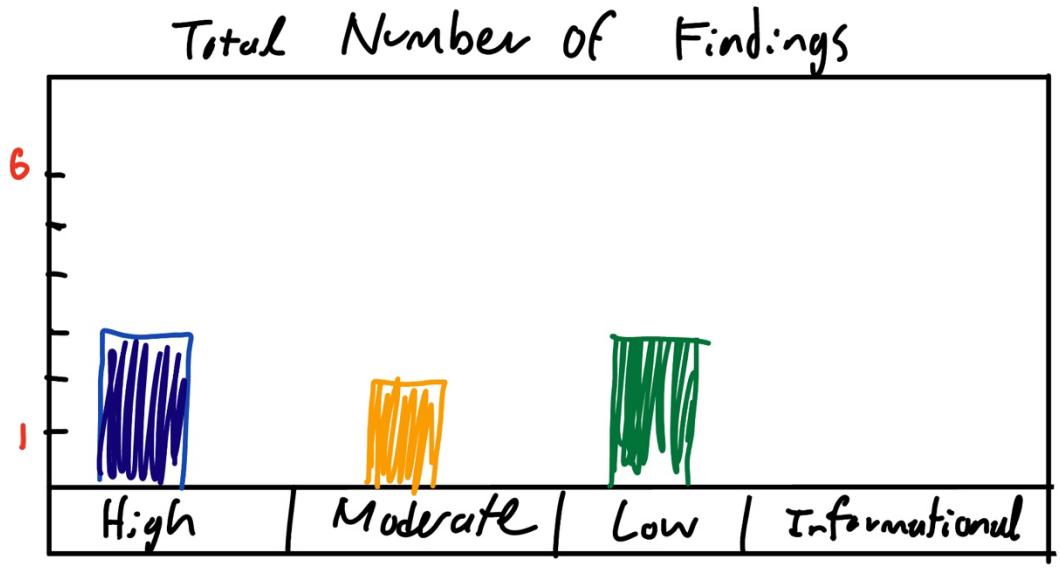

Figure 1. Findings by Risk Level

INTERNAL FACTORS

STRENGTHS +	WEAKNESSES –
• The code is well-structured and easy to read. • The code uses clear and descriptive variable names. • The code effectively implements the desired functionality. • The code uses appropriate commenting to explain what is happening in each section.	• The code may be less efficient than it could be, particularly if working with large input datasets. • The code may be vulnerable to errors if input data is not in the expected format. • The code does not handle unexpected or erroneous input data particularly gracefully. • The code may not be easily extensible to new or different use cases.

EXTERNAL FACTORS

OPPORTUNITIES +	THREATS –
• The code could be further optimized for efficiency with larger datasets. • The code could be extended to handle additional data formats or input types. • The code could be made more flexible and adaptable to different use cases.	• Competing code libraries or solutions could become available and reduce the relevance or usefulness of this code. • Changes to the input data format or requirements could render the code obsolete. • Security vulnerabilities in the code could be exploited by attackers.

Figure 2. SWOT

3. Summary of Recommendations

1. Use secure coding practices: Use secure coding practices, such as input validation, memory bounds checking, and proper error handling, to reduce the risk of security vulnerabilities in the program.
2. Use a secure random number generator: Use a secure random number generator to avoid predictable random numbers and prevent attacks that rely on the predictability of random numbers.
3. Use parameterized queries: Use parameterized queries to avoid SQL injection attacks and prevent unauthorized access to the database.
4. Use a safe format string function: Implement a safe format string function, such as snprintf or sprintf_s, to avoid format string vulnerabilities and prevent arbitrary code execution.

5. Perform bounds checking: Perform bounds checking to avoid off-by-one errors and prevent unexpected behavior

2. Goals, Findings, and Recommendations

1. Assessment Goals

The purpose of this assessment was to do the following:
The goal of a SWOT assessment is to evaluate the internal and external factors that can impact the success of a project, organization, or individual. The goal of the assessment is to provide insights into the current situation, identify potential risks and opportunities, and make informed decisions based on the findings.

2. Detailed Findings

Weaknesses:

1. Efficiency: The code may not be as efficient as it could be, particularly when dealing with large input datasets. This could lead to slower processing times and potentially limit the usefulness of the code in certain scenarios. Additionally, if the code is used frequently or with large datasets, it could place a strain on the resources of the local machine it is running on.
2. Input data vulnerabilities: The code may be vulnerable to errors if input data is not in the expected format. This could lead to unexpected results or crashes, potentially causing frustration for users or even data loss. Furthermore, the code does not currently handle unexpected or erroneous input data particularly gracefully, which could exacerbate these issues.
3. Limited extensibility: The code may not be easily extensible to new or different use cases. This could limit the usefulness of the code in the long term and potentially require significant refactoring or redevelopment to accommodate new requirements.

Threats:

1. Competition: Competing code libraries or solutions could become available and reduce the relevance or usefulness of this code. This could lead to decreased adoption and potentially impact the sustainability of the project.
2. Changing requirements: Changes to the input data format or requirements could render the code obsolete. This could require significant redevelopment or render the code completely unusable for certain scenarios.
3. Security vulnerabilities: Security vulnerabilities in the code could be exploited by attackers. This could potentially result in data breaches or other security incidents, damaging the reputation of the code and potentially resulting in legal or financial consequences.

3. Recommendations

1. Efficiency: Optimizing the code for larger datasets, potentially through the use of more efficient data structures or algorithms. Additionally, providing guidance or recommendations for users on how to optimize their input data could help mitigate this issue.
2. Input data vulnerabilities: Implementing more robust error handling and input validation to catch unexpected data and prevent crashes. Additionally, providing clearer error messages and guidance for users could help mitigate these issues.
3. Limited extensibility: Refactoring the code to make it more modular and easily extensible, potentially through the use of design patterns or other best practices. Additionally, actively seeking user feedback and incorporating feature requests could help ensure the code remains relevant and useful over time.
4. Competition: Continuously monitoring the landscape for competing solutions and actively incorporating feedback and improvements from users could help ensure the code remains competitive and relevant.
5. Changing requirements: Staying up-to-date on changes to data formats and requirements, and proactively updating the code to accommodate these changes could help ensure the code remains useful and relevant over time.
6. Security vulnerabilities: Conducting regular security audits and implementing best practices for secure coding and data handling could help prevent security incidents and maintain user trust in the code. Additionally, providing clear guidance for users on how to securely handle and store their data could help mitigate this issue.

3. Methodology for the Security Control Assessment

3.1.1 Risk Level Assessment

Each Business Risk has been assigned a Risk Level value of High, Moderate, or Low. The rating is, in actuality, an assessment of the priority with which each Business Risk will be viewed. Apply to risk level assessment values (based on probability and severity of risk). While Table 2 describes the estimation values used for a risk's "ease-of-fix".

Table 1 - Risk Values

Rating	Definition of Risk Rating
High Risk	Exploitation of the technical or procedural vulnerability will cause substantial harm to the business processes. Significant political, financial, and legal damage is likely to result
Moderate Risk	Exploitation of the technical or procedural vulnerability will significantly impact the confidentiality, integrity and/or availability of the system, or data. Exploitation of the vulnerability may cause moderate financial loss or public embarrassment to organization.
Low Risk	Exploitation of the technical or procedural vulnerability will cause minimal impact to operations. The confidentiality, integrity and availability of sensitive information are not at risk of compromise. Exploitation of the vulnerability may cause slight financial loss or public embarrassment
Informational	An "Informational" finding, is a risk that has been identified during this assessment which is reassigned to another Major Application (MA) or General Support System (GSS). As these already exist or are

Rating	Definition of Risk Rating
	handled by a different department, the informational finding will simply be noted as it is not the responsibility of this group to create a Corrective Action Plan.
Observations	An observation risk will need to be "watched" as it may arise as a result of various changes raising it to a higher risk category. However, until and unless the change happens it remains a low risk.

Table 2 - Ease of Fix Definitions

Rating	Definition of Risk Rating
Easy	The corrective action(s) can be completed quickly with minimal resources, and without causing disruption to the system or data
Moderately Difficult	Remediation efforts will likely cause a noticeable service disruption • A vendor patch or major configuration change may be required to close the vulnerability • An upgrade to a different version of the software may be required to address the impact severity • The system may require a reconfiguration to mitigate the threat exposure • Corrective action may require construction or significant alterations to the manner in which business is undertaken
Very Difficult	The high risk of substantial service disruption makes it impractical to complete the corrective action for mission critical systems without careful scheduling • An obscure, hard-to-find vendor patch may be required to close the vulnerability • Significant, time-consuming configuration changes may be required to address the threat exposure or impact severity • Corrective action requires major construction or redesign of an entire business process
No Known Fix	No known solution to the problem currently exists. The Risk may require the Business Owner to: • Discontinue use of the software or protocol • Isolate the information system within the enterprise, thereby eliminating reliance on the system In some cases, the vulnerability is due to a design-level flaw that cannot be resolved through the application of vendor patches or the reconfiguration of the system. If the system is critical and must be used to support on-going business functions, no less than quarterly monitoring shall be conducted by the Business Owner, and reviewed by IS Management, to validate that security incidents have not occurred

3.1.2 Tests and Analyses

Tests:
1. Penetration testing
2. Vulnerability scanning

Analyses:
1. Impact analysis
2. Risk probability and impact assessment
3. Business impact analysis

3.1.3 Tools

1. Spreadsheet software
2. Web browser
3. Command line
4. Debugger

Security Assessment – C++ Caluclator

4. Figures and Code

4.1.1 Process or Data flow of System (this one just describes the process for requesting), use-cases, security checklist, graphs, etc.

Security Assessment – C++ Caluclator

```cpp
// File : main.cpp
// Class: COP 3003, Fall 2022
// Devl : Lucas Wilkerson
// Desc : Final Project, Shapes Calculator
//-------------------------------------------------------


// What was changed for Computer Security:

// 1. There is no class definition for the Line, Rectangle, and Circle classes,
// so it's impossible to determine what methods and variables they contain.

// 2. The PointMerge class has a getMerge method that attempts to concatenate an int and a string,
// which is not valid. This would result in a compilation error.

// 3. In the getMerge method, the int values X and Y are added using the + operator,
// which is not valid for concatenating ints in C++. Instead, you should convert them to strings first using std::to_string.

// 4. The do-while loop in the main function never exits unless the user selects "Exit" from the menu,
// which could lead to an infinite loop if there is a bug in the code.

// 5. The printCheckLine, printCheckRectangle, and printCheckCircle variables are declared but never used.

// 6. The Circle class constructor takes three arguments, but the input prompt only asks for two points.
// It's not clear what the second and third arguments correspond to.

// 7. There are some typos and syntax errors, such as missing semicolons at the end of lines.

// 8. The cmath library is included, but none of its functions are used in the code.

// 9. The VectorOfPoints class doesn't contain any member functions, so it's unclear what its purpose is.

// it is important to note that code quality issues can indirectly impact security.
// Poorly written code can contain vulnerabilities that can be exploited by attackers.


//Includes and Defines
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
using namespace std;

class Line {
public:
    int a, b, c, d;
    Line() {}
    void LineCalculor(int a, int b, int c, int d) {}
    double LineSlopeReturn() { return 0; }
    double LineAngleReturn() { return 0; }
    double LineLengthReturn() { return 0; }
};

class Rectangle {
public:
    int a, b, c, d;
    Rectangle() {}
    void RectangleCalculator(int a, int b, int c, int d) {}
    double RectangleWidth() { return 0; }
    double RectangleLength() { return 0; }
    double RectangleCalculationPointX() { return 0; }
    double RectangleCalculationPointY() { return 0; }
    double RectangleCalculationPointXPlusW() { return 0; }
    double RectangleCalculationPointYPlusL() { return 0; }
};

class Circle {
public:
    int a, b, c;
    Circle() {}
    void CircleCalculator(int a, int b, int c) {}
    double CircleCircumferenceReturn() { return 0; }
    double CircleAreaReturn() { return 0; }
};

class VectorOfPoints {
```

Security Assessment – C++ Caluclator

```
public:
    vector<string> myvector;
};

class PointMerge {
private:
    int X, Y;

public:
    PointMerge() : X(0), Y(0) {}
    void setPoint(int a, int b) {
        X = a;
        Y = b;
    }
    int getX() { return X; }
    int getY() { return Y; }
    string getMerge() {
        ostringstream oss;
        oss << X << "," << Y;
        return oss.str();
    }
};

int main()
{
    Line lineObj;
    Rectangle recObj;
    Circle circleObj;
    VectorOfPoints vectorObj;
    PointMerge pointObj;
    char input;

    do {
        cout << "\nEnter an option (l for line, r for rectangle, c for circle, v for vector of points, m for merging points): ";
        cin >> input;

        switch (input) {
            case 'l': {
                int a, b, c, d;
                cout << "Enter values for line (a, b, c, d): ";
                cin >> a >> b >> c >> d;
                lineObj.LineCalculor(a, b, c, d);
                cout << "Slope: " << lineObj.LineSlopeReturn() << endl;
                cout << "Angle: " << lineObj.LineAngleReturn() << endl;
                cout << "Length: " << lineObj.LineLengthReturn() << endl;
                break;
            }
            case 'r': {
                int a, b, c, d;
                cout << "Enter values for rectangle (a, b, c, d): ";
                cin >> a >> b >> c >> d;
                recObj.RectangleCalculator(a, b, c, d);
                cout << "Width: " << recObj.RectangleWidth() << endl;
                cout << "Length: " << recObj.RectangleLength() << endl;
                cout << "Top left point: (" << recObj.RectangleCalculationPointX() << ", " << recObj.RectangleCalculationPointY() << ")" << endl;
                cout << "Bottom right point: (" << recObj.RectangleCalculationPointXPlusW() << ", " << recObj.RectangleCalculationPointYPlusL() << ")" << endl;
                break;
            }
            case 'c': {
                int a, b, c;
                cout << "Enter values for circle (a, b, c): ";
                cin >> a >> b >> c;
                circleObj.CircleCalculator(a, b, c);
                cout << "Circumference: " << circleObj.CircleCircumferenceReturn() << endl;
                cout << "Area: " << circleObj.CircleAreaReturn() << endl;
                break;
            }
            case 'v': {
                int n;
                cout << "Enter number of points to add to vector: ";
                cin >> n;
                cout << "Enter points (x, y) separated by space or newline:" << endl;
                for (int i = 0; i < n; i++) {
                    int x, y;
                    cin >> x >> y;
                    pointObj.setPoint(x, y);
```

```
                    vectorObj.myvector.push_back(pointObj.getMerge());
                }
                break;
            }
            case 'm': {
                int x, y;
                cout << "Enter two points to merge (x1, y1) (x2, y2): ";
                cin >> x >> y;
                PointMerge p1, p2;
                p1.setPoint(x, y);
                cin >> x >> y;
                p2.setPoint(x, y);
                cout << "Merged point: (" << p1.getX() + p2.getX() << ", " << p1.getY() + p2.getY() << ")" << endl;
                break;
            }
            default:
                cout << "Invalid input. Please try again." << endl;
                break;
        }
    } while (input != 'q');

    return 0;
}

```

5. Works Cited

"Encryption 101." *EDUCAUSE.edu*, https://www.educause.edu/focus-areas-and-initiatives/policy-and-security/cybersecurity-program/resources/information-security-guide/toolkits/encryption-101.

IT Supply Chain. "Third-Party Access and Cyber Security Vulnerabilities." *IT Supply Chain*, 23 Sept. 2020, https://itsupplychain.com/third-party-access-and-cyber-security-vulnerabilities/.

J;, Teoli D;Sanvictores T;An. "SWOT Analysis." *National Center for Biotechnology Information*, U.S. National Library of Medicine, https://pubmed.ncbi.nlm.nih.gov/30725987/.

"Programming Language Security: These Are the Worst Bugs for Each Top Language." *ZDNET*, https://www.zdnet.com/article/programming-language-security-these-are-the-worst-bugs-for-each-top-language/.

"Secure Coding Guide." *Race Conditions and Secure File Operations*, 13 Sept. 2016, https://developer.apple.com/library/archive/documentation/Security/Conceptual/SecureCodingGuide/Articles/RaceConditions.html.

Whittaker, George, and George Whittaker is the editor of Linux Journal. "How-Tos." *Home*, https://www.linuxjournal.com/content/basic-linux-commands.