Shapes Calculator

Security Assessment

Version 1.0

Release Date: May 1st, 2023

Table of Contents

Summary

The overall goal of this assessment was to improve the security of the Shapes Calculator. The major findings of this assessment showed that many security vulnerabilities existed within the project, but all action items created as part of this assessment to improve the security were able to be resolved successfully.

Assessment Scope

The major tools utilized for this assessment include CLion and GitHub on the Windows 11 operating system. Major limitations included time (the assessment period was only a few weeks) and resources as the development team had other obligations throughout the duration of the project.

Summary of Findings

A total of 14 major issues were evaluated from a standard security checklist for this assessment. Six action items were created from the issues. These action items involved issues with documentation of changes (accounting), implementation of security policy, and code vulnerabilities related to memory leak and buffer overflow (GeeksforGeeks). The calculator was created in the C++ programming language utilizing default pointers. This left the code vulnerable to buffer overflow and memory leaks due to the way C++ allocates memory with default pointers. Otherwise, the code files existed on a OneDrive account with no proper accounting or logging of changes.

Chart 1 – Count of Ease of Fix for Security Checklist

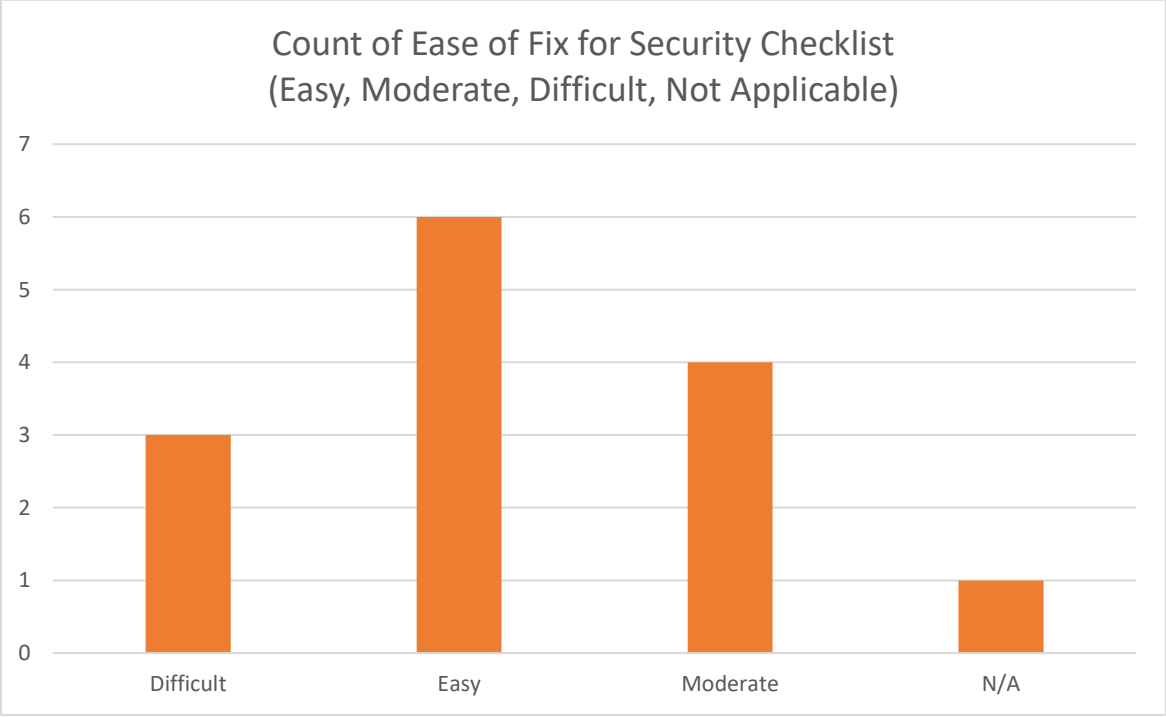
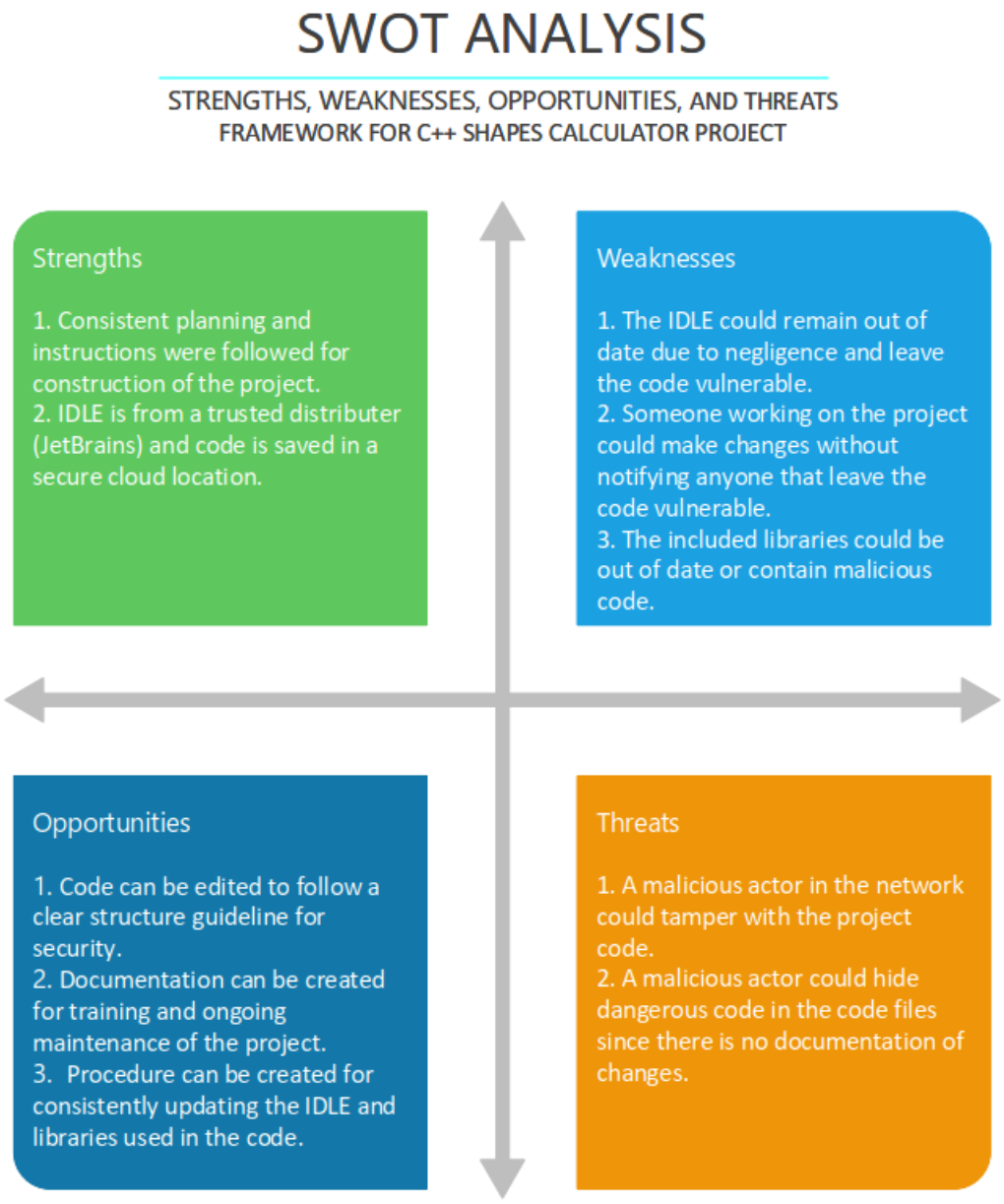


Figure 1 – SWOT Analysis



Summary of Recommendations

Out of the six action items identified from the 14 major issues evaluated, all six items were implemented successfully into the project. Future improvements to the security of the project should include implementing a more robust logging process and adding more methods of catching errors created by faulty user inputs.

Goals, Findings, and Recommendations

Assessment Goals

The goal of the assessment was to improve the security of the Shapes Calculator. The Shapes Calculator was originally created to learn about implementation of classes in the C++ programming language. However, vulnerabilities present in the construction of the project created the necessity for a security assessment.

Detailed Findings

Several vulnerabilities existed in the project prior to the security assessment. Each vulnerability was assessed individually and labeled as a threat, weakness, or vulnerability. Threats occur from outside the codebase and are out of the control of the development team. Weaknesses exist within the policy of the development team and leave the code vulnerable. Vulnerabilities are weaknesses within the code itself that can give malicious actors the opportunity to exploit the code.

Table 1 - Vulnerabilities

ID	Vulnerability	Category	Description
1	Libraries	Vulnerability	The libraires included in the project were not checked for vulnerabilities or other issues that could leave the project vulnerable. This leaves the project open to malicious actors who may exploit these libraries within the program. Libraries utilized include iostream, vector, and cmath (cplusplus.com).
2	Security Policy	Weakness	No security policy was established for the project prior to the assessment. This leaves no action plan in the scenario that the project is compromised, or a malicious actor modifies the code.
3	Accounting	Weakness	No accounting of changes to the codebase were logged prior to the assessment. This means anyone with access to the code could modify it and no record would exist of the changes or author.
4	User Input	Threat	No sanitization of user input or error handling with user input took place prior to the assessment. Any input was permitted and could crash the program. Any sort of exploit could have been input by a malicious actor while running the program.
5	Default Pointers	Vulnerability	Default C++ pointers were utilized in the program prior to the assessment. This means that memory allocation could easily be exploited and proper use of the “new” and “delete” key words for proper memory allocation was not accounted for.
6	Logging	Weakness	No logging of errors in the program were recorded prior to the assessment. This means that if any fatal or concerning errors took place there was no record to see when and where in the code execution the error took place.

7	Internal Actors	Threat	No policy to handle malicious internal actors existed prior to the assessment. Any malicious actor with access to the account with the code could have added exploitative scripts that would have been executed on the device of a user running the program.
8	Access Control	Threat	No policy or safeguards regarding access to the codebase existed prior to the assessment. Any individual with access to the OneDrive account or device logged into the OneDrive could have accessed and modified the code to any extent.

Recommendations

The recommendations made following the security assessment were all successfully implemented. The largest update to the project was moving the codebase to GitHub. Creating an online repository added a new layer of security and infrastructure that benefitted the overall security of the Shapes Calculator. Most recommendations fell into the category of "Policy" meaning that they had to do with documentation concerning the codebase. Two recommendations fell into the category of "Buffer Overflow/Memory Leak" meaning that they concerned issues with memory management or input handling in the C++ code.

Table 2 - Recommendations

ID	Recommendation	Category (Chart 2)	Vulnerability Handled (Table 1)	Description
1	Ensure Libraries are Up to Date	Policy	1	The libraries were confirmed to all be part of the standard C++ language, so they were updated when the programming language was updated. An additional concern was the version of C++ being utilized (C++ 17). However, this version is very secure compared to older versions of C++ that did not check for certain security concerns related to pointers and memory management (TylerMSFT).
2	Create Security Policy	Policy	2, 7	A security policy was created within GitHub to add a descriptive policy for how to keep the codebase secure, report newfound vulnerabilities, and handle internal actors who may compromise the online repository.
3	Track Changes and Access with GitHub	Policy	3, 8, 7	Migrating the codebase to GitHub implemented an automatic tracking system that records all changes made to the master branch of code as well as the author of the changes. This is helpful for identifying malicious internal actors who may try to compromise the code.
4	Sanitize Inputs	Buffer Overflow/ Memory Leak	4	A function implemented in C++ was added to sanitize input and ensure that no input that is not the appropriate data type (integer or floating-point) is accepted by the program. This will ensure that unwanted user input or accidental mistaken input does not crash the program.

5	Add Smart Pointers	Buffer Overflow/ Memory Leak	5	Smart pointers (specifically unique pointers) were implemented in the program for dynamic memory allocation and management. This addition ensures that the program is not at risk for buffer overflow or memory leaks caused by mismanaged pointers (GeeksforGeeks).
6	Add Error Logging Class	Policy	6, 7	A class implemented in C++ was added to create a log text file that records errors created by user input. Future work on the project will expand the class implementation to account for other types of errors that may cause the program to crash or indicate malicious behavior (Rollbar). A timestamp functionality was also added (Techie Delight).

Methodology for the Security Control Assessment

Risk Assessment

The risk assessment can be located in the GitHub online repository here. Overall, the risk assessment was generally accurate as the assessment was completed by the developer of the original program. The risks identified are as follows:

1. Input data will be put at risk due to a lack of encryption or protection. (Moderate Risk)
2. Malicious users will take advantage of a lack of user input restriction. (Moderate Risk)
3. The IDLE used to run the code will leave the code vulnerable when it is not updated. (Moderate Risk)
4. Outdated libraries used in the code will leave the code vulnerable to infiltration. (Moderate Risk)
5. Lack of careful construction of class properties will leave the code vulnerable. (Low Risk)
6. Leaving code open to editing and not in an executable file will make the code vulnerable. (High Risk)
7. No logging process of updates or changes to the files will leave attacks undetected. (High Risk)
8. Not following official code structure guidelines will leave hidden vulnerabilities in the code. (Moderate Risk)
9. User data will be vulnerable when the data is not stored in a secure location after the program closes. (Low Risk)
10. No training or ongoing maintenance to the code will leave the project vulnerable. (Moderate Risk)

The Shapes Calculator was originally created as an academic exercise, so risk assessment items generally reflect the fact that the project was not originally intended for deployment in an environment where a large amount of security precautions would be necessary. However, this does not mean that the codebase was not at risk of exploitation and therefore needed to undergo a security assessment.

Most of the findings of the risk assessment were accurate excluding items 1 and 9. Both of these issues are in regard to user data which is not saved after the program closes. Therefore, these risks did not end up applying to the implementation of the findings of the security assessments. In the future, if a calculator history functionality is implemented to save user data after the program closes, it will be beneficial to revisit these risks as they will be relevant to security evaluations of the project at that point.

Penetration Testing Process

Penetration testing was utilized to identify risks within the project and formulate security action items. CLion was used to execute the program and evaluate the security. The intial assessment revealed many security flaws as any input from the user was permitted by the program. If an integer or floating-point data type was not entered, the program would crash immediately.

White Box and Black Box Testing

Both white box and black box testing were conducted within this assessment. White box testing was used to evaluate the actual executable program, while black box testing was used to evaluate the errors in the code when an exception occurred. White box testing revealed flaws in the utilization of default pointers in the program which led to mismanagement of memory. Black box testing revealed further issues with user input throughout the program. For example, the menu accepted any type of user input and would simply crash the program immediately after submission rather than prompting for a second response.

Tools Used

CLion - a JetBrains integrated development environment (IDE) that complies and executes C++ code. This product was utilized to assess, develop, and test the Shapes Calculator.

GitHub - an online repository management platform that utilizes Git to allow developers to easily collaborate and work on software projects. This platform was used to host the online repository that the codebase was moved to. All documentation for the security assessment is located within the online repository.

Analysis of Test Results

After thorough analysis of the results of the different testing methods implemented, it was determined that vulnerabilities in the code pertained to buffer overflow, memory leaks, and a lack of input sanitization (Synk). Buffer overflow and memory leaks were primarily caused by the lack of utilization of smart pointers (GeeksforGeeks). The lack of input sanitization was due to the nature of this project as an academic exercise that did not seriously consider security.

Additionally, the lack of proper documentation of any formal security policy, accounting, or logging practices left the codebase vulnerable to a plethora of different types of malicious attacks. Outside of technical security flaws in the code, security policy and procedure documentation for the project needed to be created to protect the code from malicious actors looking to exploit the codebase.

Figures and Code

Process Flow of System

Figure 2 – Data Flow Diagram for Shapes Calculator

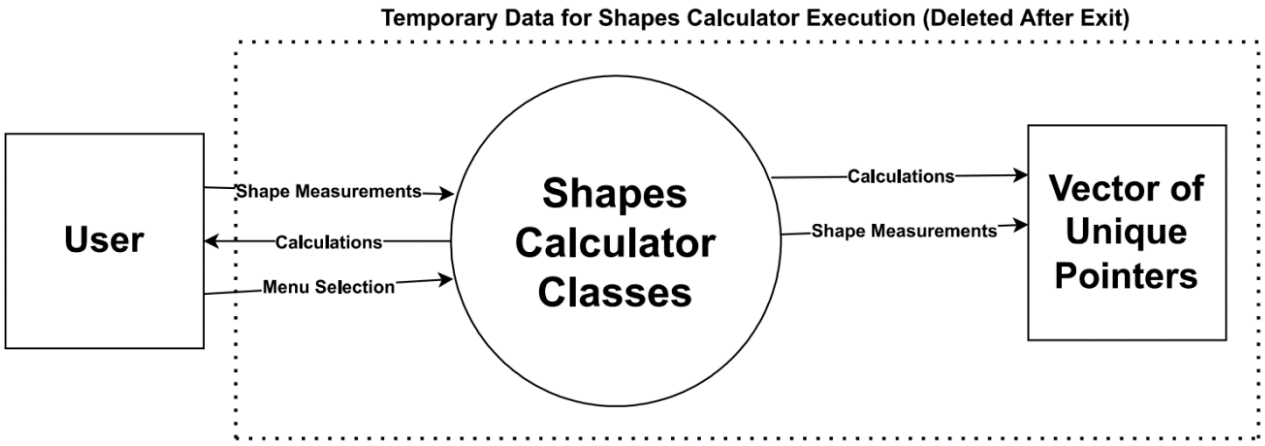
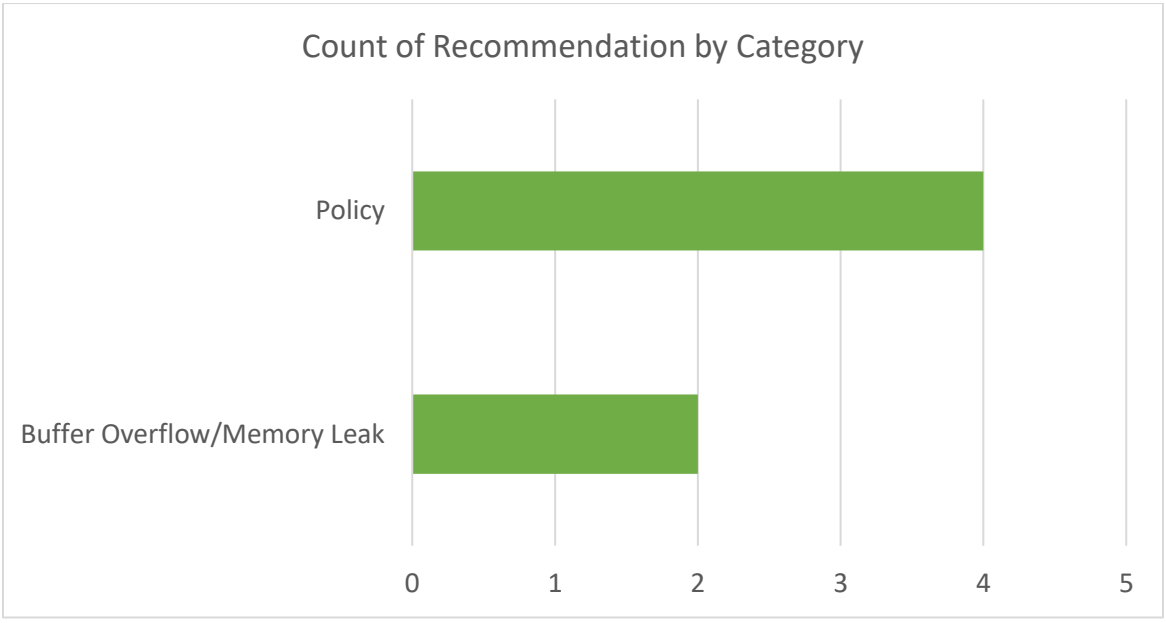


Chart 2 – Recommendations by Category



Code Figures

Figure 3 – Implementation of Security Policy

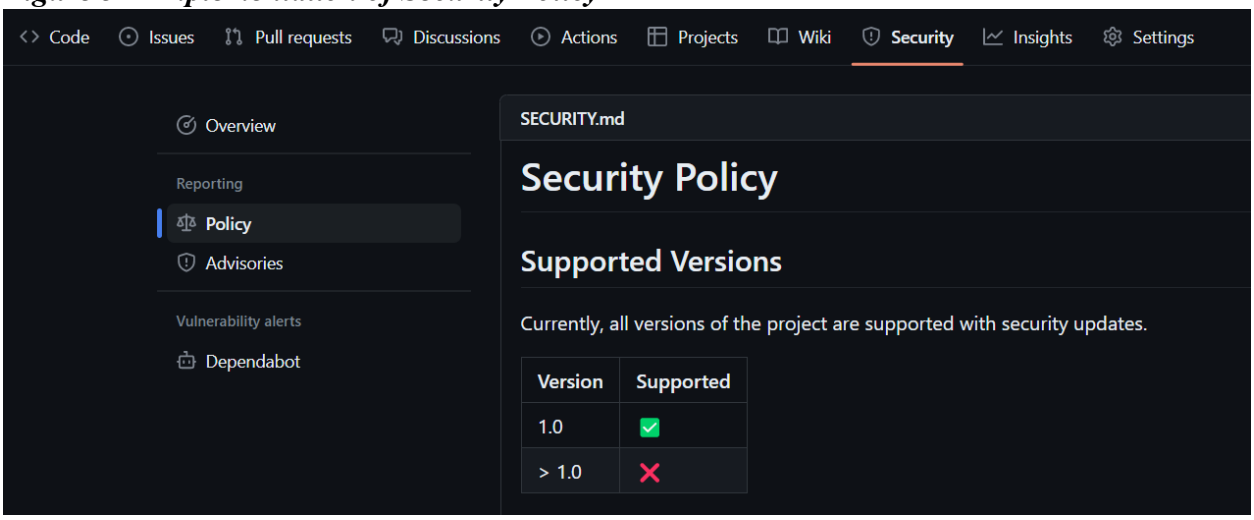


Figure 4 – Error Log File Example

```
1  Wed Apr 19 00:47:48 2023
2   ERROR: shapeMode input was invalid
3  Wed Apr 19 00:47:51 2023
4   ERROR: shapeMode input was invalid
5  Wed Apr 19 00:47:54 2023
6   ERROR: shapeMode input was invalid
7  Wed Apr 19 00:48:00 2023
8   ERROR: validateInput() input was invalid
```

Figure 5 – Input Sanitization Function

```
/**
 * validate float input
 * @param input - from the user
 * @return valid float
 */
float validateInput(float input) {
    float userInput = input;
    bool validInput = false;

    // loop until valid input is provided
    while (!validInput) {
        if (std::cin.fail()) { // if cin fails
            Logger::log(ERROR, "validateInput() input was invalid");
            std::cin.clear(); // clear the error state of the buffer
            // ignore the rest of the line after the first instance of the error
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cout << "You have entered an invalid input. Please retry: ";
            std::cin >> userInput; // prompt for new input
        } else { // cin was successful
            validInput = true;
        }
    }

    return userInput;
}
```

Works Cited

"<cmath> (Math.h)." *Cplusplus.Com*, https://cplusplus.com/reference/cmath/. Accessed 30 Apr. 2023.

"<iostream>." *Cplusplus.Com* , https://cplusplus.com/reference/iostream/. Accessed 30 Apr. 2023.

"Get Current Time and Date in C++." *Techie Delight*, 8 Feb. 2018, https://www.techiedelight.com/get-current-time-and-date-in-cpp/.

"Smart Pointers in C++." *GeeksforGeeks*, 8 June 2014, https://www.geeksforgeeks.org/smart-pointers-cpp/.

"Std::Vector." *Cplusplus.Com*, https://cplusplus.com/reference/vector/vector/. Accessed 30 Apr. 2023.

"Top 5 C++ Security Risks." *Snyk*, 16 Aug. 2022, https://snyk.io/blog/top-5-c-security-risks/.

TylerMSFT. *Welcome Back to C++ - Modern C++*. 7 Nov. 2022, https://learn.microsoft.com/en-us/cpp/cpp/welcome-back-to-cpp-modern-cpp.

"What Is Error Logging in C++." *Rollbar*, https://rollbar.com/guides/cpp/cpp-error-logging/. Accessed 30 Apr. 2023.