Security Assessment Report for a SnakeGame Project

Version N.0

January 1, 1900

Table of Contents

1. Summary 3
    1. Assessment Scope 3
    2. Summary of Findings 3
    3. Summary of Recommendations 5
2. Goals, Findings, and Recommendations 5
    1. Assessment Goals 5
    2. Detailed Findings 6
    3. Recommendations 6
3. Methodology for the Security Control Assessment 6
4. Figures and Code 8
        4.1.1 Process flow of System (this one just describes the process for requesting) 8
        4.1.2 Other figure of code 12
5. Works Cited 13

1. Summary

The purpose of this security assessment is to evaluate the security of a Snake game developed using C++ programming language and the JetBrains CLion IDE. The assessment is used to discover any potential vulnerabilities in the game's code and provide recommendations to mitigate security risks or issues that are found.

1. Assessment Scope

This security assessment used different types of technologies and tools to test the code. Some of the programming languages and technologies that were used are C++ as the primary language, the JetBarins CLion IDE, GitHub VCS, and MacOS. The assessment was limited to a single OS due to not having additional machines or resources to test the code.

2. Summary of Findings

Of the findings discovered during our assessment, 0 were considered High risks, 2 Moderate risks, 0 Low, and 0 Informational risks. The SWOT used for planning the assessment are broken down as shown in Figure 1.

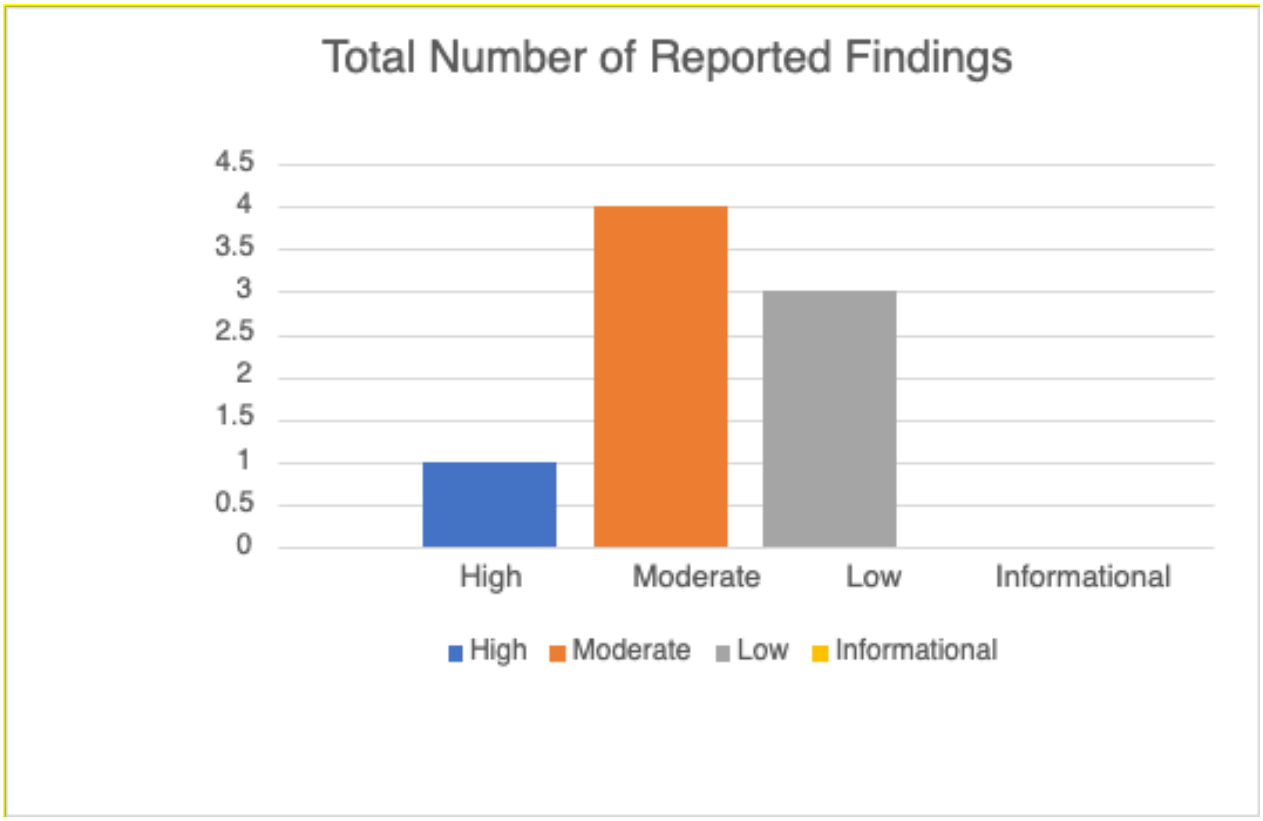


Figure 1. Findings by Risk Level

Explain above and link to full table of explanation of top risks like Figure 2.

Figure 2. SWOT

Explain which issues were used from above SWOT (which are addressed in this assessment).

The security assessment has found several issues in the Snake game's codebase, due to not having enough input validation, not enough error handling, little to none exception handling, lack of any type of logging, use of public or vulnerable libraries (SMFL), lack of version of control, no use of classes, doesn't contain any code modularization (Encapsulation). These vulnerabilities have potential risk which can be exploited by attackers to execute random harmful code, steal sensitive information of your system, or cause a denial of service. To address the identified vulnerabilities in the Snake game, it is recommended that the game adds some security measures, such as input validation to prevent buffer overflow and injection attacks, error handling to prevent functionality that isn't expected from the user, library updates to fix known security issues or bugs, access control to prevent unauthorized access, exception handling to prevent crashes, and logging and monitoring to detect and prevent any harmful code that may have been injected. Implementing these measures, the game will have better security overall and reduce the risk of potential security issues in the future.

3. Summary of Recommendationss

Based on the assessment's findings, I was able to add input validation and encryption to mitigate the identified moderate security risks. Another recommendation would be to, ensure that all public libraries for the code are up-to-date with the latest security patches, as well as performing frequent security tests to identify any potentially new security risks. These measures will help to strengthen the game's security and reduce the likelihood of successful attacks by malicious actors. Testing on multiple operating systems would need to be done in the future to ensure that the game is secure and performs across all platforms or operating systems.

2. Goals, Findings, and Recommendations

1. Assessment Goals

The purpose of this security assessment was to identify any potential vulnerabilities in the Snake game's code and provide if any recommendations for mitigating security risks. The assessment is made to evaluate the security of the game's base, systems, and data. It is used to identify any potential risks and threats to these parts of the project. The goal is to ensure that the Snake game is and stays secured and in compliance with industry standards. It is also used to implement any best practices for secure code and software development.

2. Detailed Findings

All security vulenrabilities that were found during the security assessment of my SnakeGame project have been listed in **Table 1: Summary of Vulnerabilities** under **Process or Data flow of System**.

3. Recommendations

All security vulenrabilities recommendations that have been formulated during the security assessment of my SnakeGame project have been listed in **Table 2: Recommendations** under **Process or Data flow of System.**

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. The definitions in **Error! Reference source not found.** apply to risk level assessment values (based o n 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 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

Rating	Definition of Risk Rating
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

The security assessment of the Snake game was conducted using a combination of black-box and white-box testing. When using black-box testing, many different inputs were tested for improper input validation, including large valued inputs or inputs that weren't alphabetical characters. White-box testing required me to continue reviewing the code for common coding vulnerabilities or problemss such as buffer overflows and injection attacks.

3.1.3 Tools

This was completed using GitHub as our tool and some of the commands used to work with the repository include:

- Git clone
- Git add
- Git commit
- Git push
- Git pull
- Git branch
- Git merge
- Git status
- Git checkout
- Git fetch

4. Figures and Code

Insert any pictures here (including of major code issues or code that was used as a tool – can just screenshot and add link to github). This section must include at least 4 figures or code portions:

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

Table 1: Summary of Vulnerabilities

Vulnerability	Risk Value	Threat/Weakness/ Vulnerability	Explanation
Not Enough Input Validation	Moderate	Vulnerability	Not having input validation can lead to buffer overflow vulnerabilities and other injection attacks. A buffer overflow occurs when the input provided is larger than the space in memory allocated, allowing an attacker to overwrite memory locations with malicious code. This can result in arbitrary code execution or the theft of sensitive information. Proper input validation prevents these types of attacks and protects the security of the game and its users to ensure it is working properly.
Not Enough Error Handling	Moderate	Weakness	Insufficient error handling refers not having enough procedures for detecting and handling errors that occur during the game's runtime. Insufficient error handling can lead to unexpected behavior and potential vulnerabilities. Without having proper error handling, if malicious code has been inputted into the game, there aren't any measures to verify the features are working correctly giving the attackers direct gateway to messing with the code.
Little to None Exception Handling	Moderate	Weakness	Improper exception handling is able to allow crashes to the code when compiled/ran. This can result in denial of service attacks or arbitrary code, making it a easier for attackers to attack the basis of the code and change the way the game works. It is important to add exception handling to mitigate the risks associated with these vulnerabilities.

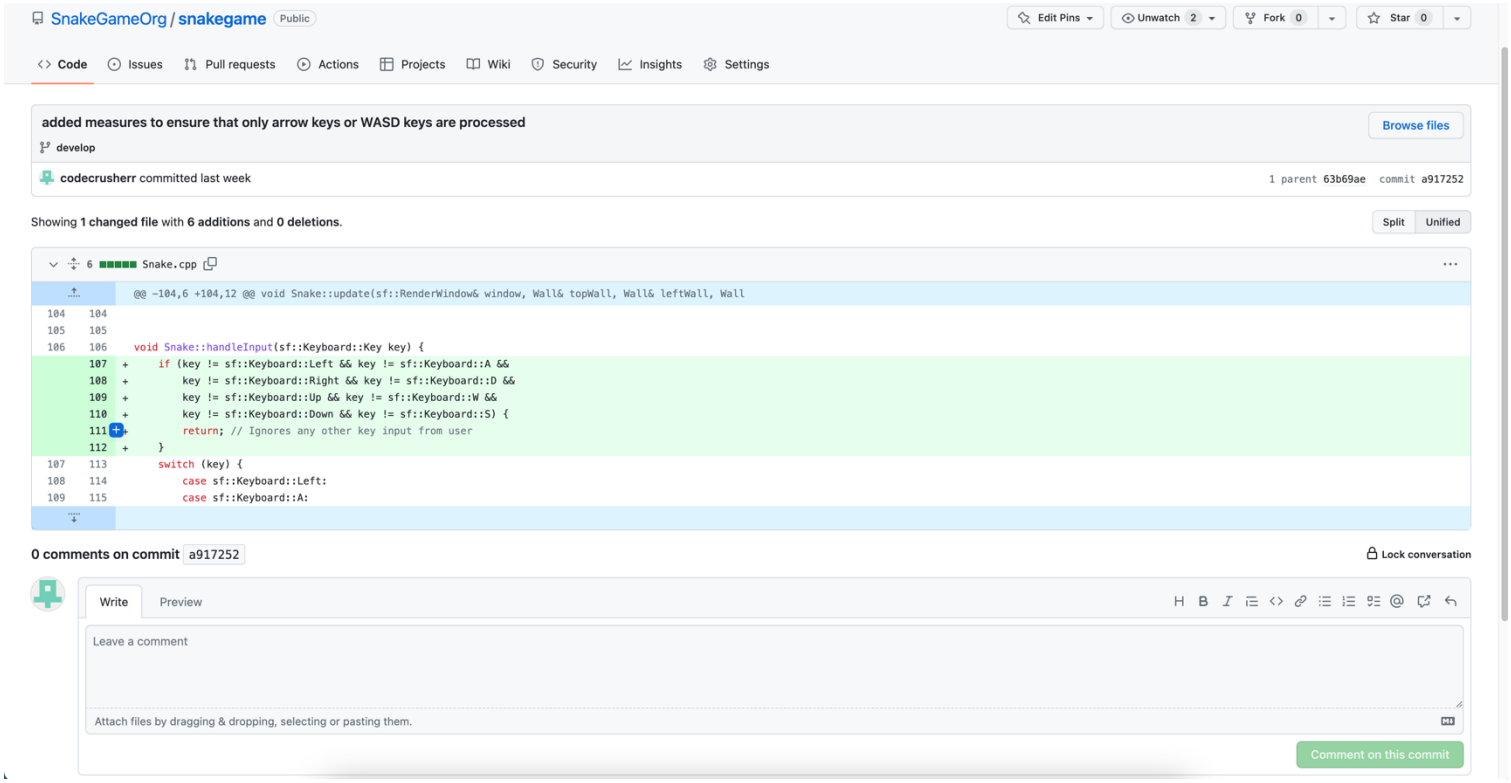
Lack of Any Type of Logging	Low	Weakness	Not having proper logging can prevent the system from recording or saving errors that may occur during the execution of the game. For example, if the game crashes due to an error, without logging, the reason for the crash is not recorded, making it hard to figure out the root of the problem. Additionally, if an attacker attempts any malicious activity, such as injecting code into the game, the lack of proper logging makes it hard to detect the it and take actions to prevent harm from occuring. Not having proper logging in the game can prevent the detection of security breaches and other potential threats making it easier for the attacker to carry out their plans.
Using Public or Vulnerable Libraries (SMFL)	High	Vulnerability	The use of public or vulnerable libraries, in this case is the SMFL Graphics library, can contain known security issues or vulnerabilities that can be exploited by hackers. This can result in random harmful code, making it a significant vulnerability that is available for an attacker to target it directly.
Lack of Version Control	Low	Weakness	Not using version control can cause code to be lost, and version differentiations if changes aren't saved or backed up. This can lead to potential code loss or unauthorized access to any important information. This allows for an attacker to gain access that wasn't authorized or cause harm to the code.
No Use of Classes	Low	Vulnerability	Not using classes when the code was first produced can make it easier for attackers to understand the game's internal system and identify potential vulnerabilities. This can allow for unauthorized access or malicious activity without being detected, making it a significant vulnerability that can be used to attackers advantange.
Doesn't Contain Any Code Modularization (Encapsulation)	Moderate	Weakness	Not using code modularization, like encapsulation, can make it easier for attackers to change or access important functions in the game. This can lead to unauthorized access or harmful actions without being detected, making it a huge weakness that attackers are able to take advantage of.

Table 2: Recommendations

Recommendation	Description	Risk Value
Not Enough Input Validation	The game should implement proper input validation to prevent buffer overflow and other injection attacks. For example, the game should check the length and format of user input. OWASP provides guidelines for input validation.	Moderate
Not Enough Error Handling	The game should implement error handling to prevent unexpected behavior and potential vulnerabilities. For example, the game should handle unexpected user input, such as null or empty values, and provide informative error messages to the user. OWASP provides guidelines for error handling.	Moderate
Little to None Exception Handling	The game should use exception handling to catch and handle runtime errors. This can be achieved by adding print statements such as using **cout** statements to output error messages during runtime. By using these statements, the code can better handle unexpected crashes or errors and reduce the risk of potential security breaches.	Moderate
Lack of Any Type of Logging	The game should use proper logging and monitoring to detect and respond to any security breaches and other potential threats to the code. For example, the game should log user actions to monitor the actions happening within the game. OWASP provides	Low

	guidelines for logging and monitoring.	
Use up-to-date and secure libraries if they are public	Update to the latest version of the library and ensure that all deprecated or vulnerable libraries are removed or updated.	High
Lack of Version Control	Implement proper version control processes for managing changes to the code. This can be done by using version control systems such as Git or SVN.	Low
No Use of Classes	Implement proper class structures to hide sensitive data and protect critical game features from unauthorized access. Using classes, the game's internal varibles or code can be better suited, making it more difficult for attackers to understand and find vulnerabilities.	Low
Doesn't Contain Any Code Modularization (Encapsulation)	Implement proper code modularization techniques such as encapsulation and abstraction. This can be done by using classes and object-oriented programming techniques.	Moderate

4.1.2 Other figure of code



https://github.com/SnakeGameOrg/snakegame/commit/a917252abd6b17d142d97fa628e709a7d160a60a

This is a video represerntation of the game functioning and some security features that were implemented.

5. Works Cited

OWASP (n.d.). Secure coding practices. Retrieved from https://owasp.org/www-project-secure-coding-practices/

OWASP (n.d.). Access control cheat sheet. Retrieved from https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html

OWASP (n.d.). Exception handling cheat sheet. Retrieved from https://cheatsheetseries.owasp.org/cheatsheets/Exception_Handling_Cheat_Sheet.html

Microsoft. (2021). Classes (C++). Microsoft Learn. https://learn.microsoft.com/en-us/cpp/cpp/class-cpp?view=msvc-170.

GitHub, Inc. (2021a). Create a repo - GitHub Docs. https://docs.github.com/en/get-started/quickstart/create-a-repo.

GitHub, Inc. (2021b). Managing teams and people with access to your repository - GitHub Docs. https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/managing-repository-settings/managing-teams-and-people-with-access-to-your-repository.