

**ADVANCED SOFTWARE SECURITY**

**Individual assignment (50%)**

**CT123-3-3-ASC**

**Intake Code: APU3F2311CS(CYB)**

**Lecturer Name: NOR AZLINA BINTI ABD RAHMAN**

**Hand out date: Week 3**

**Hand in date: Week 14**

**Student name: Abdullah Tareq Salem Banwair  
Student TP: TP056876**

# Abstract

Security is of the highest priority in software development in order to secure sensitive data and maintain the integrity of the system, especially in the current era of digital technology. This research covers the "Enhanced hospital Registrations System" an enhanced version of the original "Vulnerable hospital Registrations System" which was particularly developed to identify and reduce existing security vulnerabilities. The system uses a client-server architecture which includes strict security mechanisms, such as strong user authentication, data encryption, and thorough activity logging. The research identifies major vulnerabilities, including SQL Injection, improper input validation, error handling, and path traversal, and demonstrates how these might be exploited. This study showcases the implementation of secure coding methods to improve the security and dependability of software systems. The main objective is to protect against cyber threats and ensure adherence to industry standards.

Table of Contents

[Abstract 2](#_Toc173284321)

[1. INTRODUCTION 5](#_Toc173284322)

[2. Vulnerability Analysis 6](#_Toc173284323)

[3. Importance of Secure Coding 11](#_Toc173284324)

[4. Secure Coding Implementation 14](#_Toc173284325)

[5. Conclusion 21](#_Toc173284326)

[6. Reference 22](#_Toc173284327)

**LIST OF FIGURES**

[Figure 1: Register User SQL Injection Vulnerability 6](#_Toc173284288)

[Figure 2: Authenticate User Vulnerability 7](#_Toc173284289)

[Figure 3: Register Patient Improper Input Validation vulnerability 8](#_Toc173284290)

[Figure 4: Path Traversal vulnerability 9](#_Toc173284291)

[Figure 5: Libraries 14](#_Toc173284292)

[Figure 6: SQLite3 Library 15](#_Toc173284293)

[Figure 7: getpass Library 15](#_Toc173284294)

[Figure 8: OS Library 15](#_Toc173284295)

[Figure 9: datetime Library 16](#_Toc173284296)

[Figure 10: re Library 16](#_Toc173284297)

[Figure 11: sha256 Library 17](#_Toc173284298)

[Figure 12: Secure User Registration Function 17](#_Toc173284299)

[Figure 13: Secure User Authentication Function 18](#_Toc173284300)

[Figure 14: Input Validation Patterns 21](#_Toc173284301)

# INTRODUCTION

The present digital environment is increasingly filled with security risks that endanger the confidentiality, accuracy, and accessibility of information systems. These risks emphasize the importance for strong security measures to be included into software development from the beginning. By implementing effective safety measures at the very start of the development process, not only are current threats effectively addressed, but also possible future weaknesses are constantly recognized. This strategy ensures that digital platforms maintain their sustainability, dependability, and ability to adapt to constantly evolving cyber threats (Stratoflow, 2023).

For this research, a client-server program called the "Enhanced hospital Registration System" was developed. This system is an improved version of the first "Vulnerable hospital Registration System," particularly restructured to tackle and reduce existing security vulnerabilities. The architecture of the Enhanced hospital Registration System is carefully crafted to comply to strict security protocols while ensuring maximum performance and user convenience. This guarantees that the system can efficiently manage sensitive data, such as personal user information and transactional data, without harming security.

**Architectural and functional requirements:**  
 **Architecture:** The system has a client-server approach in which the server operates as the central hub for processing, data storage, and security management. This architecture was developed specifically to simplify the process of transferring data effectively, while also implementing strict security measures at every point of interaction.  
**Requirements related to the specific functions and capabilities that the system must possess:**

* **User Authentication and Authorization:** Implementing strong methods to ensure that access is only allowed to confirmed users using secure authentication protocols.
* **Data Encryption:** The transmission of data between the client and server is protected via advanced encryption standards, which effectively prevent any unauthorized attempts to intercept the data.
* **Activity Logging:** Completely recording user actions to monitor and examine system usage and identify any unusual behaviors that may suggest security breaches.

# Vulnerability Analysis

The "Vulnerable hospital Registrations System" in this section demonstrates many significant security weaknesses that makes it vulnerable to several attacks. Gaining a comprehensive understanding of these vulnerabilities, their primary processes, and possible situations in which they might be exploited is of critical significance in the development of a secure system. This section explores multiple weaknesses, including SQL Injection, Improper Error Handling, and Path Traversal. Each risk is evaluated to determine its potential threats and its effects of exploitation.

**SQL Injection:**

SQL Injection is a serious vulnerability that occurs when an attacker can modify a SQL query by introducing malicious code into an input field. This vulnerability poses a significant risk since it enables unauthorized access into the database, resulting in potential data breaches and manipulation.

For example, in the register\_user function:

A computer screen shot of a code

Description automatically generated

Figure 1: Register User SQL Injection Vulnerability

If an attacker inputs a string such as admin', 'admin123', 'admin'); DROP TABLE users; -- as the username, the SQL statement is modified, which could result in deleting the users' table. This vulnerability risks the security and privacy of the database, resulting in unauthorized access and modification of confidential data (NIST, 2024).

Similarly, the authenticate\_user function is also vulnerable:

A screen shot of a computer screen

Description automatically generated

Figure 2: Authenticate User Vulnerability

If the inputs are not properly checked, an attacker might use malicious SQL or script code. For example, if a string such as Robert'); DROP TABLE patients; -- is entered in the name column, it might cause the execution of more malicious SQL commands. This has the potential to cause unauthorized modification or removal of data, resulting in data breaches (NIST, 2024). In addition, the injection of script code has the potential to result in Cross-Site Scripting (XSS) attacks, which can enable attackers to take control of user sessions, modify the appearance of websites, or send users to malicious websites (OWASP, 2024).

**Improper Input Validation**

Improper input validation can lead to multiple vulnerabilities, including SQL Injection and XSS. For instance, in the register\_patient function:

A black screen with colorful text

Description automatically generated

Figure 3: Register Patient Improper Input Validation vulnerability

This function provides the user with comprehensive error messages, including stack traces or SQL difficulties, that may expose crucial details about the database structure or server configuration. Attackers may improve their attacks by utilizing this information, specifically by customizing SQL injections or system intrusions (SEI, 2024).

**Path Traversal**

Path traversal vulnerabilities occur when an application ignores to properly evaluate user input, which allows attackers to modify file paths and get access to files located outside the intended directory. This can result in unauthorized access and modification of crucial system files (Snyk, 2024).

A screen shot of a computer program

Description automatically generated

Figure 4: Path Traversal vulnerability

In these functions, user input for filenames is directly used without validation. An attacker could input ../../../etc/passwd to read sensitive system files or manipulate paths to overwrite critical files, causing data corruption or system failures (Secure Coding, 2024).

**Potential Exploitation Scenarios**

The vulnerabilities have a significant impact. SQL Injection can result in unauthorized access and manipulation of data, resulting in both immediate and long-term damage. Insufficient input validation can result in SQL Injection, Cross-Site Scripting (XSS), and other types of attacks, which exploit user confidence and result in data breaches and damage to reputation. Improper error handling enables attackers the data they need to exploit other vulnerabilities. Path traversal attacks can lead to unauthorized access to files and compromise of the system.

**Research and Analysis**

In the absence of secure coding methods, these vulnerabilities can be easily exploited. SQL injection attacks have the potential to cause major data breaches. According to statistics, these attacks represent over 60% of all web application threats reported (NIST, 2024). Improper input validation and error handling could increase these risks by giving attackers additional opportunities and information to enhance their attacks. Path traversal vulnerabilities can have serious consequences as they enable attackers to gain unauthorized access to and manipulate crucial files (Secure Coding, 2024).

# Importance of Secure Coding

Secure coding is an essential part of software development that involves following a set of principles and practices to eliminate multiple vulnerabilities. Developers may significantly enhance the security and integrity of their applications by following secure coding standards, which helps to minimize the chances of cyberattacks. According to (SEI, 2024), secure coding enables programmers to better recognize and resolve common security concerns that are present in software. In addition, (Snyk, 2024) highlights the economic advantages of secure coding, emphasizing that taking proactive measures to optimize for security can help reduce the long-term cost related to data breaches and system exploitations.

**SQL Injection Prevention**

Utilizing secure coding techniques is crucial for reducing the likelihood of risks such as SQL Injection, which involves the injection of malicious SQL expressions into an entry field for action. Developers can enhance the security of the application by utilizing query parameters or prepared statements, which help prevent certain vulnerabilities. The advantages of implementing these strategies include:

* **Prevention of Unauthorized Data Access**: Implementing secure coding practices prevents attackers from manipulating database commands, therefore protecting sensitive data (SEI, 2024).
* **Maintenance of Data Integrity**:It ensures that data cannot be modified using unauthorized SQL commands, thereby maintaining the dependability and integrity of the database (SEI, 2024).
* **Enhanced System Stability**: Implementing measures to prevent SQL injections is crucial for ensuring the stability and performance of database systems (NIST, 2024).

**Cross-Site Scripting (XSS) Mitigation**

Secure coding mitigates vulnerabilities like Cross-Site Scripting (XSS) by filtering user input to prevent the injection of malicious scripts. Implementing thorough input validation is crucial to ensure that the program only handles authorized and secure data, effectively blocking potential attackers from inserting malicious scripts. This feature provides users with protection against session hijacking, fraudulent site redirects, and theft of sensitive data (Snyk, 2024). The benefits include:

* **User Protection:** It serves as a safeguard against attackers attempting to hijack user sessions or redirect users to malicious websites (Snyk, 2024).
* **Enhanced User Trust:** Secure coding ensures the protection of user data and interactions, which is essential for establishing and preserving confidence with the user base (SEI, 2024).
* **Compliance with Security Standards:** Adhering to industry security standards and regulations (SEI, 2024) is facilitated by implementing secure coding methods for XSS mitigation.

**Secure Error Handling**

Implementing effective error-handling techniques in secure coding methods is crucial for mitigating the risk of exposing sensitive system information through error messages. This precautionary measure helps to minimize potential exploitation by attackers seeking to gather knowledge about the system's basic structure. Secure coding improves the security architecture by controlling the amount of information revealed in error messages and logs. Some advantages include:

* **Reduced Information Leakage**: Reducing the level of detail included in error outputs is beneficial for protecting sensitive system information (NIST, 2024).
* **Improved System Resilience**: Implementing strong error-handling mechanisms ensures that systems may effectively manage faults without compromising security (SEI, 2024)..
* **Enhanced Monitoring and Diagnostics**: Implementing structured error handling facilitates improved system health monitoring and expedited issue detection (SEI, 2024).

**Path Traversal Protection**

Path traversal vulnerabilities occur when applications poorly manage user inputs utilized in file paths. Secure coding standards involve meticulous input validation and the utilization of secure APIs that protect file system access, thereby preventing potential attackers from exploiting these weaknesses. Applications can prevent unauthorized file access and modification by verifying file paths and ensuring they are free from unauthorized directory traversal characters (NIST, 2024).

**Comprehensive Security**

Ensuring the implementation of secure coding techniques across the whole software development process is crucial to mitigate vulnerabilities and safeguard the system against potential attacks. Implementing proactive security measures during the development process helps prevent expensive breaches and ensure the integrity and dependability of software systems. Organizations may safeguard their assets, build user confidence, and maintain a competitive advantage in the security-focused digital landscape by using secure coding principles throughout the development lifecycle (Secure Coding, 2024).

# Secure Coding Implementation

The "Enhanced hospital Registrations System" includes many libraries and methodologies to address vulnerabilities and offer strong security through the use of secure coding practices. Every function and its related secure implementation illustrate how to protect against particular vulnerabilities.

**Libraries and Their Functions**

In this section, we start by importing multiple libraries that offer essential functionality to improve the security and usability of the "Enhanced hospital Registrations System" Each library is chosen to perform distinct functions in the secure coding process:

A screen shot of a computer

Description automatically generated

Figure 5: Libraries

The **SQLite3** library enables the interaction with SQLite databases, allowing the secure storage, retrieval, and management of user data. Customized queries protect against SQL injection attacks, which pose significant threats to the security of databases.

A screen shot of a computer

Description automatically generated

Figure 6: SQLite3 Library

The **getpass** library provides a secure method for capturing user input for passwords, ensuring that the input is not displayed on the screen. This approach ensures the confidentiality of passwords during input.



Figure 7: getpass Library

The **os** library serves for communication with the operating system, offering services for creating secure directories and manipulating file paths. Additionally, it assists in configuring environment variables and managing file permissions to improve security.

A black screen with white text

Description automatically generated

Figure 8: OS Library

**datetime** This library is utilized to produce timestamps for the specific purposes of logging and auditing. By recording crucial events with precise time markers, we can maintain a comprehensive record of activities for the purpose of security monitoring and management of incidents.

A black background with white text

Description automatically generated

Figure 9: datetime Library

The **re** library offers functions for performing regular expression matching operations, which are utilized in this context for input validation. To mitigate various injection threats, it is crucial to validate user inputs to ensure they conform to expected patterns.

A screenshot of a computer code

Description automatically generated

Figure 10: re Library

The **sha256** function, which is part of the Hashlib package, is utilized to cryptographically hash passwords before their storage in the database. This method ensures that in the event of a database breach, passwords in their original form are not exposed.

A black background with white text

Description automatically generated

Figure 11: sha256 Library

**User Registration Function**

The user registration function ensures that usernames pass validation and passwords are securely hashed before being saved in the database. This procedure serves to mitigate typical vulnerabilities, such as SQL injection and poor password storage.

A screen shot of a computer program

Description automatically generated

Figure 12: Secure User Registration Function

**User Authentication Function**

The user authentication function validates the user's credentials by comparing the hashed input password with the stored hashed password. This method ensures a secure process of verifying the identity of users and effectively blocks any unauthorized access.

A screen shot of a computer

Description automatically generated

Figure 13: Secure User Authentication Function

**Secure Path Handling Functions**

In order to address path traversal vulnerabilities, the file read and write functions include input validation to verify that filenames do not contain unauthorized characters that could allow directory traversal.

A screen shot of a computer program

Description automatically generated

**Input Validation Patterns**

In order to improve security, the system employs various input validation patterns. These patterns ensure that user inputs conform to expected formats, hence preventing malicious inputs that could result in security vulnerabilities such as SQL injection, cross-site scripting (XSS), and other types of injection attacks.

* **USERNAME\_PATTERN:** Verifies that usernames consist only of alphanumeric letters, underscores, dots, and hyphens. This serves as a preventive measure against the inclusion of special characters that could potentially be exploited in injection attacks.
* **ROLE\_PATTERN:** Limits the role input to either 'admin' or 'user', ensuring that only valid roles are assigned.
* **NAME\_PATTERN:** Verifies that names contain only letters and spaces, hence preventing the insertion of potentially malicious characters.
* **AGE\_PATTERN:** Verifies that the age input is a valid numerical value, hence preventing the inclusion of non-numeric characters that may cause interruptions in database operations.
* **CONTACT\_PATTERN:** validates contact numbers by checking that they consist only of digits, with the possibility of having a plus sign at the beginning, and a length between 10 and 15 characters.
* **FILENAME\_PATTERN:** Verifies that filenames consist only of alphanumeric characters, spaces, and hyphens, and have valid file extensions that include 3 to 4 characters. Limiting input to authorized filenames essentially prevents path traversal attacks.

A computer screen with many colorful text

Description automatically generated with medium confidence

Figure 14: Input Validation Patterns

# Conclusion

In conclusion, the implementation of secure coding techniques is crucial in reducing vulnerabilities and safeguarding software systems against cyber threats. The "Enhanced hospital Registrations System" demonstrates the effectiveness of integrating security measures from the early stages of development to mitigate significant vulnerabilities such as SQL Injection, Input Validation, and path traversal. To provide solid security measures, we implement parameterized queries, validate input properly, handle passwords securely, and maintain good error reporting. These practices not only improve the stability of the system and the integrity of the data but also promote user trust and respect for security requirements. Implementing a proactive strategy in secure coding significantly reduces the likelihood of data breaches and other cyber risks, ensuring the system's long-term viability, reliability, and flexibility in the constantly changing digital environment.

# Reference

Burkett, D. (2024, July 16). *Code Security | Kiuwan*. Kiuwan. https://www.kiuwan.com/

Dixit, D. (2024, April 25). Implementing secure coding practices in TechSEC - Neumetric - 2024. *Neumetric*. https://www.neumetric.com/implementing-secure-coding-practices-in-techsec/

Krysik, A. (2024, March 1). *Maximizing security in software Development: Expert Tips for 2023 - Stratoflow*. Stratoflow. https://stratoflow.com/security-in-software-development/

*OWASP Secure Coding Practices - Quick Reference Guide | Secure Coding Practices | OWASP Foundation*. (n.d.). https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/stable-en/01-introduction/05-introduction

*Secure Development | Software Engineering Institute*. (n.d.). https://www.sei.cmu.edu/our-work/secure-development/

*SEI CERT Coding Standards - CERT Secure Coding - Confluence*. (n.d.). https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards

Snyk. (2020, December 21). *Secure coding practices every developer should know*. https://snyk.io/learn/secure-coding-practices/

Souppaya, M., Scarfone, K., & Dodson, D. (2022). *Secure Software Development Framework (SSDF) version 1.1 :* https://doi.org/10.6028/nist.sp.800-218

*W3Schools.com*. (n.d.-a). https://www.w3schools.com/python/python\_mysql\_getstarted.asp

*W3Schools.com*. (n.d.-b). https://www.w3schools.com/python/python\_ref\_functions.asp

*W3Schools.com*. (n.d.-c). https://www.w3schools.com/python/python\_user\_input.asp

Warden, C. (2024, April 5). *The importance of secure software development*. Lucidum. https://lucidum.io/blog/the-importance-of-secure-software-development/