

---

# SECURITY ASSESSMENT AND ENHANCEMENT OF A LIBRARY MANAGEMENT SYSTEM

---

A Comprehensive Analysis and Implementation of Security Mechanisms



UNIVERSITY OF  
MAURITIUS

SEPTEMBER 3, 2024

ABDURRAHMAN NOOR-UL-HAQQ GURIB | ID 2413319

University Of Mauritius

Faculty of Information, Communication and Digital Technologies [FolCDT]

Mr. Gavin Sathan, Lecturer Data Security, BSc Top-Up Cyber Security

# **Table of Contents**

## 1. Summary

## 2. Introduction

## 3. Threat Assessment

- 3.1 Input Validation and Injection Attacks
- 3.2 Data Integrity
- 3.3 Data Confidentiality
- 3.4 Data Persistence
- 3.5 Logging and Monitoring

## 4. Security Mechanisms Implementation

- 4.1 Input Validation and Sanitization
- 4.2 Data Encryption
- 4.3 File Permissions
- 4.4 Data Integrity Checks
- 4.5 Error Handling and Logging
- 4.6 Data Backup

## 5. Conclusion

# **1. Summary**

The Library Management System is a Java-based application that handles student and book data stored in CSV files (Database).

While the system is functional, it is vulnerable to various security threats such as input injection, data tampering, and unauthorized access.

This report outlines a comprehensive assessment of the application's vulnerabilities and details the security mechanisms implemented to mitigate these risks.

The enhancements include input validation, data encryption, file permission settings, integrity checks, error handling, and regular data backups.

These measures significantly improve the security and resilience of the system, ensuring data confidentiality, integrity, and availability.

# **2. Introduction**

The security of software systems is paramount in today's digital landscape, where data breaches and cyber-attacks are increasingly common.

This report presents a security assessment and enhancement of a Java-based Library Management System, which manages student and book loan records through CSV files.

The assessment identifies potential threats to the system, while the enhancement focuses on implementing robust security mechanisms to protect against these threats.

The goal is to ensure the application maintains high data integrity, confidentiality, and availability.

## **3. Threat Assessment**

### **3.1 Input Validation and Injection Attacks**

The system accepts user input for student names and book selections, making it vulnerable to injection attacks. Malicious inputs could compromise data integrity or lead to unauthorized actions within the application.

### **3.2 Data Integrity**

The use of CSV files for data storage introduces risks related to data tampering. Without checks, data could be altered, leading to inaccurate loan records and mismanagement of the library's inventory.

### **3.3 Data Confidentiality**

The CSV files contain sensitive information such as student names and loaned books, which could be exposed to unauthorized users if proper access controls are not in place.

### **3.4 Data Persistence**

Improper file handling or system failures could result in data loss, disrupting the library's operations and leading to a loss of critical information.

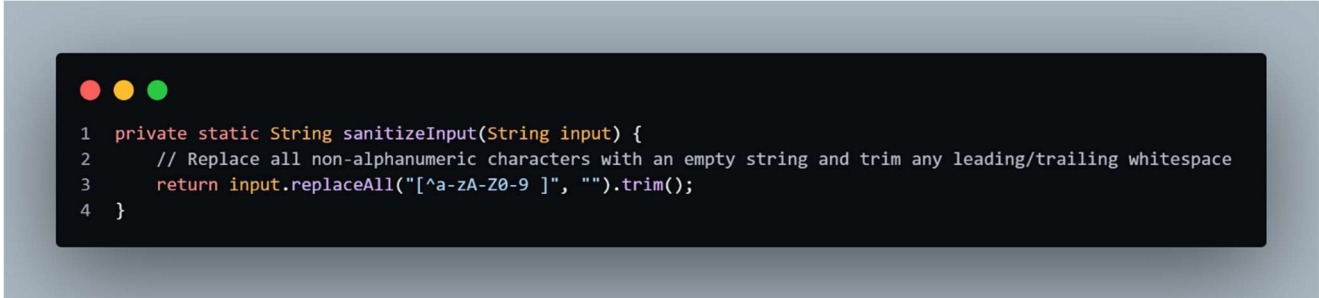
### **3.5 Logging and Monitoring**

The absence of logging mechanisms makes it difficult to track system activity, diagnose issues, or respond to security incidents promptly.

## 4. Security Mechanisms Implementation (with Code Snippets)

### 4.1 Input Validation and Sanitization

Code Snippet:



```
1 private static String sanitizeInput(String input) {  
2     // Replace all non-alphanumeric characters with an empty string and trim any leading/trailing whitespace  
3     return input.replaceAll("[^a-zA-Z0-9 ]", "").trim();  
4 }
```

#### Explanation:

The `sanitizeInput` method ensures that user inputs contain only alphanumeric characters (letters and numbers) and spaces. This prevents users from entering malicious input, such as SQL injection attempts or commands that could compromise the system. The `replaceAll` method removes any special characters that could be used to manipulate the application. This is especially important for applications that store user data, like our Library Management System, to prevent CSV injection.

#### Protection Against SQL Injection:

- SQL injection is a common attack where an attacker tries to execute malicious SQL queries through user input. By sanitizing the input to include only safe characters, the risk of executing unintended SQL commands is minimized.

#### Prevention of Command Injection:

- Command injection occurs when an attacker tries to execute arbitrary commands on the host system via vulnerable input fields. The `sanitizeInput` method helps block these attempts by removing any special characters that could be used to manipulate the command execution process.

#### Defense Against CSV Injection:

- CSV injection, also known as Formula Injection, happens when malicious data is entered into a CSV file that could later be executed by spreadsheet software (like Excel). By restricting the input to safe characters, this method ensures that no harmful formulas or commands are embedded in the CSV files generated by the application.

## 4.2 Data Encryption

Code Snippet (AES Encryption):

```
1  import javax.crypto.Cipher;
2  import javax.crypto.KeyGenerator;
3  import javax.crypto.SecretKey;
4  import javax.crypto.spec.SecretKeySpec;
5  import java.util.Base64;
6
7  public class EncryptionUtil {
8
9      // Generate AES key
10     public static SecretKey generateKey() throws Exception {
11         KeyGenerator keyGen = KeyGenerator.getInstance("AES");
12         keyGen.init(128);
13         return keyGen.generateKey();
14     }
15
16     // Encrypt the data
17     public static String encrypt(String data, SecretKey key) throws Exception {
18         Cipher cipher = Cipher.getInstance("AES");
19         cipher.init(Cipher.ENCRYPT_MODE, key);
20         byte[] encryptedBytes = cipher.doFinal(data.getBytes());
21         return Base64.getEncoder().encodeToString(encryptedBytes);
22     }
23
24     // Decrypt the data
25     public static String decrypt(String encryptedData, SecretKey key) throws Exception {
26         Cipher cipher = Cipher.getInstance("AES");
27         cipher.init(Cipher.DECRYPT_MODE, key);
28         byte[] decodedBytes = Base64.getDecoder().decode(encryptedData);
29         byte[] decryptedBytes = cipher.doFinal(decodedBytes);
30         return new String(decryptedBytes);
31     }
32 }
```

### Explanation:


This AES (Advanced Encryption Standard) encryption utility provides methods for encrypting and decrypting sensitive data such as student names and loan records before storing them in CSV files. The encrypt method takes in plain text data, encrypts it using a secret key, and outputs a Base64-encoded string. The decrypt method reverses this process, ensuring that only authorized systems can read the data. This adds a layer of security, ensuring that even if the CSV file is accessed, the contents remain unreadable without the encryption key.

### Application to Library Management System:

- Encrypting data like student names and loan records before storing them in CSV files helps protect against unauthorized access and data theft.
- This encryption utility ensures that even if the CSV files are compromised, the data remains secure and unreadable without the correct decryption key, safeguarding the privacy and integrity of user information.

## 4.3 File Permissions

Code Snippet:



```
1  import java.io.File;
2
3  public class FilePermissions {
4      public static void setPermissions(String filePath) {
5          File file = new File(filePath);
6
7          // Set read and write permissions only for the owner of the file
8          file.setReadable(true, true);
9          file.setWritable(true, true);
10         file.setExecutable(false, false); // No execution permissions
11     }
12 }
```

### Explanation:

This method configures strict file permissions to prevent unauthorized access to the CSV files. The `setReadable` and `setWritable` methods ensure that only the owner of the file (in this case, the application itself) has read and write access. The `setExecutable(false, false)` line ensures that the file cannot be executed, which helps to avoid accidental execution of files that are not intended to be executed as programs.

### Relevance to Library Management System:

- Ensuring strict file permissions for CSV files is crucial for protecting the data stored in these files. This approach safeguards against unauthorized modifications or access, helping to maintain the integrity and security of the library's records.
- Limiting access to only the application itself helps prevent external entities from tampering with the files. This reduces the chances of data corruption, unauthorized data exposure, or malicious attacks like privilege escalation.
- Configuring file permissions is a critical security measure that helps control who can read, write, or execute a file. This method ensures that the CSV files used in the application are protected from unauthorized access or manipulation.

## 4.4 Data Integrity Checks

Code Snippet (Checksum Generation and Validation):

```
1  import java.nio.file.Files;
2  import java.nio.file.Paths;
3  import java.security.MessageDigest;
4
5  public class IntegrityCheck {
6
7      // Generate checksum using SHA-256
8      public static String generateChecksum(String filePath) throws Exception {
9          byte[] fileBytes = Files.readAllBytes(Paths.get(filePath));
10         MessageDigest md = MessageDigest.getInstance("SHA-256");
11         byte[] checksumBytes = md.digest(fileBytes);
12         StringBuilder sb = new StringBuilder();
13         for (byte b : checksumBytes) {
14             sb.append(String.format("%02x", b));
15         }
16         return sb.toString();
17     }
18
19     // Validate checksum to ensure data integrity
20     public static boolean validateChecksum(String filePath, String expectedChecksum) throws Exception {
21         String actualChecksum = generateChecksum(filePath);
22         return actualChecksum.equals(expectedChecksum);
23     }
24 }
25
```

### Explanation:

The generateChecksum method reads the content of the CSV file and creates a unique hash (checksum) using the SHA-256 algorithm. This checksum can be used to verify the integrity of the file. The validateChecksum method compares the expected checksum with the actual checksum of the file to detect any unauthorized changes. If the checksums match, the file is considered intact; otherwise, the system will flag a potential tampering incident.

### Application to Library Management System:

- Using checksum generation and validation methods adds a layer of security by ensuring that the CSV files containing sensitive data (e.g., student names and loan records) have not been altered.
- This helps maintain the integrity of the data stored in the system, providing assurance that the information is accurate and has not been compromised.

### Detection of Unauthorized Changes:

- Any discrepancy between the expected and actual checksums triggers an alert, allowing the system administrators to take immediate action to investigate the potential security breach or data corruption.

### Example of Usage:

- Before saving or retrieving data from the CSV file, the application can generate a checksum and compare it with a previously stored checksum to verify that the file's integrity has been maintained.
- This process ensures that the data being used by the application is reliable and has not been subject to unauthorized modifications.



## 4.5 Error Handling and Logging

### Code Snippet (Logging Using Java's Logger):

```
1 import java.util.logging.Logger;
2 import java.util.logging.Level;
3
4 public class LibraryManagementLogger {
5
6     private static final Logger logger = Logger.getLogger(LibraryManagementLogger.class.getName());
7
8     public static void logInfo(String message) {
9         logger.log(Level.INFO, message);
10    }
11
12    public static void logError(String message, Exception e) {
13        logger.log(Level.SEVERE, message, e);
14    }
15 }
16
```

### Explanation:

This logging utility allows the system to log informational messages (logInfo) and errors (logError) that occur during the application's runtime. By logging events like file read/write operations, encryption/decryption steps, and user actions, the application can track its activity and respond to issues more effectively. In the event of an exception, the error is logged with the message and stack trace, which is crucial for diagnosing and fixing problems.

### Application to Library Management System:

- Logging is used to monitor the system's operations and track user interactions with the application.
- Informational logs help verify that operations such as reading and writing data to CSV files, encrypting and decrypting records, and user actions are performed correctly.
- Error logs provide valuable information for diagnosing issues related to file handling, data processing, or encryption/decryption, ensuring that any problems can be addressed promptly.

### Benefits of Effective Logging:

- Effective logging helps maintain the overall health of the application by providing insights into its performance and behavior.
- It aids in proactive maintenance, ensuring that potential issues are identified and resolved before they impact users or system functionality.

## 4.6 Data Backup

### Code Snippet (File Backup):

```
1 public class BackupManager {
2
3     public static void backupFile(String sourceFilePath, String backupFilePath) throws IOException {
4         File sourceFile = new File(sourceFilePath);
5         File backupFile = new File(backupFilePath);
6
7         // Copy file from source to backup location
8         Files.copy(sourceFile.toPath(), backupFile.toPath(), StandardCopyOption.REPLACE_EXISTING);
9         System.out.println("Backup completed successfully.");
10    }
11
12    public static void main(String[] args) {
13        try {
14            // Example file paths for demonstration
15            String sourceFilePath = "C:\\Users\\SD233484\\OneDrive - SD Worx\\Desktop\\Assignment_Data_Security\\LibraryManagementSystem\\data\\LibraryData.csv";
16            String backupFilePath = "C:\\Users\\SD233484\\OneDrive - SD Worx\\Desktop\\Assignment_Data_Security\\LibraryManagementSystem\\data\\Backups\\LibraryData_backup.csv";
17
18            C:\\Users\\SD233484\\OneDrive - SD Worx\\Desktop\\Assignment_Data_Security\\LibraryManagementSystem\\data\\students.csv
19
20            // Perform the backup
21            backupFile(sourceFilePath, backupFilePath);
22        } catch (IOException e) {
23            System.err.println("Backup failed: " + e.getMessage());
24        }
25    }
26 }
27
28
```

### Explanation:

This method ensures the CSV file is backed up by copying it to a secure location. The `Files.copy` method copies the file from the source path to the backup path. By regularly scheduling this backup operation, the system ensures that data can be restored in case of file corruption, deletion, or any system failures, thus preventing data loss.

### Preventing Data Loss:

- Regular backups are a proactive measure to prevent data loss, ensuring that important information, such as student records and loan details, is preserved even if unforeseen issues occur.
- This method provides a safety net, allowing the system to recover from data-related incidents and maintain data integrity.

### Application to Library Management System:

- Backing up CSV files that contain sensitive data (e.g., student names and loan records) is crucial for data protection.
- By implementing regular backups, the system ensures that valuable data is not lost and can be restored if necessary, thus maintaining the continuity and reliability of the application.

### Benefits of Effective Backup:

- Effective backup practices help safeguard against data loss and system failures, providing a reliable way to recover data.
- Regular backups contribute to the overall robustness of the system, ensuring that it can handle disruptions and continue to operate smoothly.

## 6. Conclusion

The security enhancements implemented in the Library Management System address the key vulnerabilities identified during the threat assessment. By incorporating input validation, encryption, access controls, integrity checks, and logging mechanisms, the system is now better equipped to handle potential security threats. These measures ensure that the data within the system remains secure, reliable, and available, thereby supporting the ongoing operations of the library. Continued monitoring and updates will be necessary to maintain the security of the application in the face of evolving threats.

The integration of the discussed security mechanisms into the Library Management System profoundly enhances its overall security and reliability. Input validation is a critical first line of defense, ensuring that only clean and valid data is processed, thereby mitigating the risk of attacks such as SQL injection or code manipulation. Encryption adds a robust layer of protection for sensitive information by converting it into an unreadable format, ensuring that unauthorized access does not compromise the confidentiality of the data.

File permissions play a crucial role in restricting access to the CSV files, preventing unauthorized users or processes from reading or modifying critical data. This not only secures the data from accidental or malicious tampering but also upholds the integrity of the information. Data integrity checks, through the use of checksums, provide a mechanism to verify that the files have not been altered or corrupted, thus ensuring that the data remains accurate and trustworthy over time.

Furthermore, effective error handling and logging offer valuable insights into the system's operation, recording both informational events and errors. This capability facilitates prompt diagnosis and resolution of issues, supports auditing efforts, and contributes to the overall stability and reliability of the application. By incorporating these security measures, the Library Management System is better protected against a range of security threats and is positioned to maintain high standards of data integrity and operational reliability. Together, these enhancements provide a comprehensive approach to safeguarding the system, ensuring that it operates securely and effectively in a dynamic and potentially hazardous environment.