# Security Assessment Report for SpendSmart Digital Budgeting App

Prepared by Oyindamola Olayiwola

16th July, 2024

## Executive Summary

This document identifies critical issues across the Software Development Life Cycle (SDLC) phasessuch as incomplete requirements, poor communication, inadequate security, inconsistent coding standards, insufficient testing, manual deployment errors, and reactive maintenance—and proposes strategies to address them. These strategies include adopting Agile methodologies, enhancing security practices, improving testing procedures, strengthening documentation, and automating deployment processes. By implementing these recommendations, organizations can significantly enhance the efficiency, security, and quality of their software development, ensuring that their products meet user expectations, are robust against security threats, and maintain a competitive edge in the market.

- **Scope of Analysis**: Evaluation of security requirements and analysis of the SDLC process, providing recommendations for enhancement.
- **Methodology**: Review of existing security requirements and SDLC practices, identifying gaps and potential improvements through a structured assessment.

## Security Requirements for SpendSmart Digital Budgeting App

1. Token-Based Authentication: Use OAuth 2.0 for secure authentication with external data sources. Tokens should be securely stored and refreshed periodically.
2. Encryption Protocols for Data Transmission: Use TLS to encrypt data during transmission, adhering to the latest standards and cipher suites.
Evaluation:
3. Data Parsing and Normalization: Develop algorithms to parse and normalize data from diverse sources, with error handling for discrepancies.
4.Multi-Factor Authentication (MFA): Use SMS, email, or app-based authenticators for additional security.
5. Role-Based Access Control (RBAC): Define roles with specific permissions, regularly reviewed and updated.
6. AES-256 Encryption for Data Storage: Encrypt sensitive data using AES-256 and implement robust key management.
7. Key Management: Use a secure Key Management System (KMS) for key lifecycle management.
8. Secure Logging Mechanism: Implement a secure logging framework with encryption and restricted access.
9. User-Friendly Interfaces: Design intuitive interfaces with instructions and tooltips for secure practices.
Evaluation:
10. Regular User Communication: Regularly communicate security features, updates, and best practices to users.

## Evaluation of Security Requirement for the SpendSmart App

| Security Requirement | Strength | Weakness |
|---|---|---|
| Token-Based Authentication | -The use of OAuth 2.0 is explicitly defined, ensuring secure authentication.<br><br>-There is an Emphasis on secure storage and periodic refresh of tokens | Lack of detailed procedures for token revocation and handling compromised tokens. |
| Encryption Protocols for Data Transmission | - The use of TLS for encrypting data during transmission is Specified<br>- There is an Emphasis on Using the latest standard ad Cipher Suites | There is no specific schedule for reviewing and updating TLS configuration |
| Data Parsing and Normalization | - Details the need for algorithms to handle diverse data formats and error handling.<br><br>- Includes the need for managing discrepancies and anomalies. | - There is a Lack of specific error-handling procedures and logging mechanisms. |
| Multi-Factor Authentication | - The New requirement specifies using MFA methods like SMS, email, or app-based authenticators to provide an additional layer of security beyond passwords. | - The new security requirement does not consider adaptive authentication based on risk assessments**.** |
| Role-Based Access Control (RBAC) | - It defines roles with specific permissions and emphasizes regular review and updates. This ensures access to features and functionalities based on user role. | There is a lack of procedures for dynamic adjustments based on user activity and behavior. |

| | | |
|---|---|---|
| AES-256 Encryption for Data Storage | It Specifies the use of AES-256 for encrypting sensitive data at rest. | It Does not provide specific details on key rotation frequency and secure storage solutions. |
| Key Management | - Describes the use of a secure Key Management System (KMS) for managing encryption keys.<br><br>- Covers key generation, storage, rotation, and access policies. | - Lacks detailed procedures for key destruction. |
| Secure Logging Mechanism | Defines the need for a secure logging framework with encryption and restricted access. It also ensures logs are protected against unauthorized access and tampering | - Does not define specific logging retention policies and review procedures. |
| User-Friendly Interfaces | Emphasizes the need for intuitive interfaces with instructions and tooltips. | Lacks mechanisms for gathering user feedback on the usability of security features. |
| Regular User Communication | Specifies regular communication of security features and updates to users. This Promotes user trust through transparent communication. | Does not specify the methods and frequency of communication. |

## Recommendation for improvement

1. Token-Based Authentication
   - Invalidate compromised tokens immediately.
   - Detect and address unusual token usage patterns.
   - Use hardware security modules (HSM) for token protection.
2. Encryption Protocols for Data Transmission
   - Schedule TLS configuration reviews and updates.
   - Regularly scan for encryption protocol weaknesses.
3. Data Parsing and Normalization
   - Develop detailed error handling and logging procedures.
   - Ensure consistent data formats across sources.
4. Multi-Factor Authentication (MFA)
   - Adjust MFA based on user behavior and risk.
   - Provide alternative MFA options like backup codes.
5. Role-Based Access Control (RBAC)
   - Update roles based on real-time activity.
   - Frequently audit role assignments and permissions.
6. AES-256 Encryption for Data Storage
   - Regularly change encryption keys.
   - Use HSMs for key protection.
   - Limit access to encryption keys.
7. Key Management
   - Develop procedures for secure key disposal.
   - Audit key management practices.
8. Secure Logging Mechanism
   - Define and enforce log retention and archiving.
   - Regularly review logs for suspicious activities.
   - Encrypt logs to prevent unauthorized access.
9. User-Friendly Interfaces
   -Gather and act on user feedback for security features.
   - Provide ongoing security education within the app.
10. Regular User Communication
    - Use email, in-app notifications, and SMS.
    - Regularly inform users about security updates and threats.

# SDLC Process Analysis

**Planning Phase**

Finding 1: Incomplete Requirements Specification

Requirements are not fully captured or understood, leading to potential scope creep and project delays.

- Impact: This can result in the development of features that do not meet user needs or missing critical functionalities.

Finding 2: Poor Stakeholder Communication

Ineffective communication with stakeholders leads to misunderstandings and misaligned expectations.

- Impact: Stakeholders might not be satisfied with the final product, and important requirements might be overlooked.

**Design Phase**

Finding 1: Insufficient Security Considerations

Security aspects are not adequately addressed during the design phase, leading to vulnerabilities in the system.

- Impact: Potential for security breaches and data loss, compromising the integrity and confidentiality of the system.

Finding 2: Lack of Scalability Planning

The system design does not account for future growth, leading to performance issues as the user base expands.

- Impact: The system may become slow or unresponsive under increased load, requiring significant rework to address scalability.

**Implementation Phase**

Finding 1: Inconsistent Coding Standards

Developers do not adhere to a unified coding standard, resulting in code that is difficult to read and maintain.

- Impact: Increased likelihood of bugs, higher maintenance costs, and challenges in onboarding new developers.

Finding 2: Insufficient Testing During Development

Lack of rigorous testing during the coding phase leads to undetected bugs and functional issues.

- Impact: Bugs discovered late in the development cycle can be costly and time-consuming to fix.

**Testing Phase**

Finding 1: Inadequate Test Coverage

Test cases do not cover all possible scenarios, leaving parts of the application untested.

- Impact: Undetected bugs can lead to system failures in production, impacting user satisfaction and trust.

Finding 2: Poor Integration Testing

Integration tests are not performed thoroughly, leading to issues when different modules are combined.

- Impact: Interoperability issues between modules can cause system malfunctions.

**Deployment Phase**

Finding 1: Insufficient Deployment Automation

Manual deployment processes are error-prone and time-consuming.

- Impact: Increased risk of deployment errors, longer deployment times, and difficulty in rolling back failed deployments.

Finding 2: Lack of Proper Rollback Mechanisms

No clear strategy for rolling back deployments in case of failures.

- Impact: Difficulty in recovering from deployment issues, leading to prolonged downtime.

**Maintenance Phase**

Finding 1: Reactive Maintenance Approach

Maintenance activities are primarily reactive rather than proactive, addressing issues only after they arise.

- Impact: Increased downtime and system instability, as issues are not prevented or detected early.

Finding 2: Inadequate Documentation

Poor documentation of the system makes it challenging to understand and maintain.

- Impact: Higher effort is required for troubleshooting and onboarding new team members, leading to increased costs.

## Strategies for Improving the SDLC

1**. Implement Agile Methodologies**

Adopting Agile methodologies can improve communication, flexibility, and iterative development.

Implementation Steps:

1. Train the team on Agile principles and practices.
2. Establish regular sprints and sprint reviews.
3. Conduct daily stand-up meetings to ensure continuous communication.
4. Utilize tools like Jira or Trello to track progress and manage tasks.

**2. Enhance Security Practices**

Incorporate security considerations throughout the SDLC to mitigate vulnerabilities.

- Implementation Steps:

1. Conduct security training for all team members.
2. Integrate security reviews into each phase of the SDLC.
3. Perform regular security testing, including vulnerability assessments and penetration testing.
4. Implement secure coding standards and practices.

**3. Improve Testing Practices**

Enhance testing procedures to ensure thorough validation of the system.

-Implementation Steps:

1. Develop comprehensive test plans covering all scenarios.
2. Automate testing where possible to increase coverage and efficiency.
3. Conduct thorough integration testing to ensure module compatibility.
4. Perform regular regression testing to detect issues introduced by new changes.

**4. Strengthen Documentation**

Improve documentation practices to facilitate easier maintenance and knowledge transfer.

- Implementation Steps:

1. Create detailed documentation for all phases of the SDLC.
2. Use standardized templates and tools for documentation.

3. Ensure documentation is updated regularly and maintained in a central repository.
4. Conduct regular reviews and audits of documentation to ensure accuracy and completeness.

**5. Automate Deployment Processes**

Implement automation in deployment to reduce errors and increase efficiency.

- Implementation Steps:

1. Use continuous integration/continuous deployment (CI/CD) tools like Jenkins or GitLab CI.
2. Develop scripts to automate the deployment process.
3. Establish automated rollback mechanisms to recover from failed deployments.
4. Conduct regular training for the team on using deployment automation tools.

## Conclusion

This security assessment identified vulnerabilities in the SpendSmart budgeting app and weaknesses throughout the development process. Implementing Agile methodologies, enhanced security practices, improved testing procedures, stronger documentation, and deployment automation will significantly improve the app's security and overall quality, fostering user trust and ensuring a competitive edge in the market.