SpendSmart Application Security and SDLC Analysis

Table of contents

Executive Summary

Key Strengths:

- Secure design principles followed for system architecture
- Encryption used for data protection
- Encryption used for data transit

Areas for Improvement:

- Lack of automated security scanning and testing
- Slow/unclear remediation times for identified vulnerabilities
- Minimal use of configuration management for production systems

Critical Findings:

- Unclear patching cadence, should be as specific as possible to assure consistency and compliance
- SQL injection vulnerabilities possible due to lack of input sanitization
- No dependency scanning

Suggested Actions:

- Increase security focused automation in testing and scanning to ensure consistency
- Comprehensive security testing including SAST, DAST, penetration testing
- Improve vulnerability management process to accelerate remediation
- Implement configuration management tools for production systems
- Enforce production deployment checklist to ensure security controls applied
- Conduct security training for developers to build secure coding skills

Methodology

Analysis of the application security and SDLC practices and processes is conducted to identify gaps and areas for improvement. Each phase of the development life cycle is examined with a focus on security requirements and specific implementations. The results are discussed and recommendations are provided.

Results

Planning phase:

- Lack of security team formation.
- Guidelines lack specificity no framework defined.
- The technical implementation details indicate a focus on security requirements like encryption, access controls, logging, and secure storage.
- While a formal risk assessment process is not mentioned, the security measures described aim to mitigate common risks like data breaches.
- Regulatory and compliance needs are not referenced but appear partially addressed through encryption, logging, and access controls that follow security best practices, though formal compliance processes are not described.

Analysis phase:

- Sensitive data is identified and classified per data guidelines, encrypted with AES-256 before storage, and protected in transit (protocal not specified).
- Access is restricted and logged to mitigate data breach, account takeover, and fraud threats.
- New features increase attack surface so role-based access control and principle of least privilege are followed.

Design phase

- Multi-factor authentication, role-based access controls, AES-256 encryption for data at rest and in transit, and secure logging of transactions and activities provide security.
- Input validation and sanitization not mentioned for user-supplied data.
- Proper key management, security audits, and penetration testing are also recommended.

Development phase:

- Code changes undergo peer review before merging.
- Selenium provides browser testing.
- Snyk is not used for dependency and code scanning during development.
- Automation focuses on integration over security.
- Git enables version control and branching for collaboration, with assumed MFA login.
- Jira provides issue tracking.

Testing phase:

- It is unclear if comprehensive security testing like SAST, DAST, and penetration testing is conducted during testing.
- More details on methodology and scope are needed.
- Automation for consistent security testing is not mentioned and would need to be clarified.
- The process for remediation vulnerabilities is undefined and more information is required on prioritization, tracking, and remediation.

Implementation phase:

- It is unclear if deployment scripts and configurations undergo security reviews.

- Configuration management processes for deployed systems are undefined.
- The process for reviewing/approving production changes is not defined.
- An incident response plan for deployment is not mentioned.
- Processes are unspecific for security patch management, continuous monitoring, log reviews, and actions from security events.

Maintenance phase

- While no specific patching process is mentioned, logs are securely stored with access controls and encryption.
- Continuous monitoring set up with tools like Datadog with a focus on performance.
- Snyk used for code inspection but not dependency monitoring.
- Jira used for ticket management and prioritization, but update timeline remains vague.

Recommendations

The security requirements analysis revealed gaps in specifics around input validation, patching process, intrusion detection, network security, compliance, and auditing. Enhancing requirements with additional details in these areas is recommended ensure shared understanding and close gaps of knowledge.

The SDLC security analysis found strengths like code reviews, testing, scanning and logging, but lacks formal processes for security testing methodology, patching, log reviews, and incident response. Implementing automated and manual security processes throughout the development process is advised.

Additionally implementation of SAST, DAST, or IAST is strongly recommended to identify and address vulnerabilities early in the development process where it is much easier to remedy. The combination of automatic and manual testing is recommended to address the possible false positives or negatives. Due to the sensitive nature of the data in use hardening the application is recommended. Snyk should be implemented earlier in the development process to identify and remediate vulnerabilities before they become threats. Furthermore it should be used for dependency management and to help maintain a clear software inventory to mitigate third-party risk.

Specifics for compliance are left out and relies "industry standards" which are undefined. Utilizing a compliance framework to address the gaps in standards is recommended and provides clear accountability. Some frameworks include NIST CSF 2.0, ISO 27001, ISO 27002, SOC2, and GDPR.

Configuration management tools should be implemented to ensure production systems are secure.