Secure Development Policy

DOCUMENT CLASSIFICATION	Internal
VERISON	1.0
DATE	
DOCUMENT AUTHOR	**Ayaz Sabir**
DOCUMENT OWNER	

REVISION HISTORY

VERSION	DATE	REVISION AUTHOR	SUMMARY OF CHANGES

DISTRIBUTION LIST

NAME	SUMMARY OF CHANGE

APPROVAL

NAME	POSITION	SIGN

Contents

1. Introduction

Software applications and systems form the foundation of organizational operations, customer services, and business processes. The increasing complexity of software development, combined with sophisticated cyber threats and stringent regulatory requirements, necessitates comprehensive security integration throughout the software development lifecycle (SDLC). Vulnerabilities introduced during development can lead to data breaches, system compromises, service disruptions, and significant financial and reputational damage.

Modern software development faces numerous security challenges including insecure coding practices, inadequate security testing, insufficient access controls, third-party component vulnerabilities, and deployment configuration errors. The rapid adoption of agile development methodologies, DevOps practices, cloud platforms, and open-source components further amplifies the need for systematic security integration throughout the development process.

This Secure Development Policy establishes the framework for integrating information security controls and practices throughout all phases of software development in accordance with ISO/IEC 27001:2022.

2. Purpose

The primary purpose of this Secure Development Policy is to establish comprehensive security controls and practices for all software development activities. This policy aims to:

- **Integrate Security by Design**: Embed security considerations into all phases of the software development lifecycle from initial requirements gathering through deployment and maintenance.
- **Prevent Security Vulnerabilities**: Implement development practices and controls that prevent the introduction of security vulnerabilities and weaknesses into software applications and systems.

- **Ensure Secure Coding**: Establish secure coding standards and practices that protect against common vulnerabilities and attack vectors.
- **Control Development Environment Security**: Implement appropriate security controls for development, testing, and staging environments to protect source code, development tools, and sensitive data.
- **Manage Third-Party Components**: Establish processes for securely managing and integrating third-party libraries, frameworks, and components into developed software.
- **Enable Security Testing**: Implement comprehensive security testing practices including static analysis, dynamic testing, and penetration testing throughout the development process.
- **Support Regulatory Compliance**: Ensure that developed software meets applicable legal, regulatory, and contractual security requirements.
- **Facilitate Secure Deployment**: Establish secure deployment practices and configuration management that maintain security throughout the software lifecycle.

3. Scope

This Secure Development Policy applies to all software development activities conducted by or on behalf of the organization, including all development teams, environments, tools, and processes involved in creating, modifying, or maintaining software applications and systems. The policy encompasses:

- **All Development Activities**: Requirements analysis, design, coding, testing, integration, deployment, and maintenance of software applications, systems, and components.
- **All Development Methodologies**: Waterfall, agile, DevOps, continuous integration/continuous deployment (CI/CD), and any other development

approaches used by the organization.

- **All Software Types**: Web applications, mobile applications, desktop software, embedded systems, APIs, microservices, and any other software developed or customized by the organization.
- **All Development Environments**: Development, testing, staging, and production environments used for software development and deployment activities.
- **All Development Personnel**: Internal developers, external contractors, consultants, and third-party development teams involved in software development activities.
- **All Development Tools**: Integrated development environments (IDEs), source code management systems, build tools, testing frameworks, and deployment platforms.
- **Entire Development Lifecycle**: From initial project conception and requirements gathering through ongoing maintenance, updates, and eventual retirement of software systems.

This policy establishes minimum security requirements for software development activities. Specific detailed procedures and technical standards will be documented separately and referenced herein.

4. Policy Statements

This section outlines the mandatory principles and practices for secure software development, aligning with ISO/IEC 27001:2022 requirements. These statements provide clear management direction and support for all development security activities.

4.1 General Principles for Secure Development

All software development activities must adhere to the following general principles to ensure comprehensive security integration:

Security by Design: Security must be considered and integrated from the earliest phases of development rather than being added as an afterthought or bolt-on solution.

Defense in Depth: Software security must implement multiple layers of protection including input validation, access controls, encryption, and monitoring capabilities.

Least Privilege: Software applications must implement and enforce the principle of least privilege for user access, system permissions, and component interactions.

Fail Securely: Software must be designed to fail in a secure manner that does not expose sensitive information or create security vulnerabilities.

Risk-Based Approach: Security controls and practices must be proportionate to the risks associated with the software's function, data sensitivity, and operating environment.

Continuous Security: Security must be maintained throughout the entire software lifecycle through ongoing monitoring, testing, and updates.

4.2 Secure Development Lifecycle

The organization shall implement a secure development cycle that integrates security into all development phases:

Requirements Phase: Security requirements shall be identified, documented, and integrated into functional requirements based on risk assessments and regulatory obligations.

Design Phase: Security architecture and design shall be developed with appropriate threat modeling, security controls specification, and design reviews.

Implementation Phase: Secure coding practices shall be followed with appropriate code reviews, static analysis, and security testing integration.

Testing Phase: Comprehensive security testing shall be conducted including vulnerability assessments, penetration testing, and security functionality verification.

Deployment Phase: Secure deployment practices shall be implemented with

appropriate configuration management, access controls, and security verification.

Maintenance Phase: Ongoing security maintenance shall include vulnerability management, security updates, and continuous monitoring.

4.3 Development Environment Security

The organization shall implement comprehensive security controls for development environments:

Access Controls: Development environments shall implement appropriate access controls, authentication mechanisms, and authorization processes for developers and systems.

Environment Isolation: Development, testing, and production environments shall be appropriately isolated with controlled data flows and access restrictions.

Data Protection: Sensitive data used in development and testing shall be protected through appropriate controls including data masking, encryption, and access restrictions.

Configuration Management: Development environment configurations shall be managed securely with appropriate version control, change management, and security baselines.

Monitoring and Logging: Development activities shall be monitored and logged appropriately to support security oversight and incident investigation.

4.4 Code Security and Quality

The organization shall establish comprehensive code security and quality requirements:

Secure Coding Standards: Development shall follow established secure coding standards and guidelines appropriate to the programming languages and platforms used.

Code Review: All code shall be subject to appropriate review processes including peer reviews, security reviews, and automated analysis.

Vulnerability Prevention: Development practices shall specifically address prevention

of common vulnerabilities including injection attacks, authentication bypasses, and data exposure.

Third-Party Components: Third-party libraries, frameworks, and components shall be evaluated, approved, and managed for security throughout their lifecycle.

Documentation: Code shall be appropriately documented with security considerations, dependencies, and configuration requirements clearly specified.

5. Roles and Responsibilities

5.1 Senior Management

Senior management is responsible for:

- Providing leadership and commitment to secure development practices
- Allocating adequate resources for secure development activities
- Approving secure development policies and major architectural decisions
- Reviewing secure development performance and security metrics
- Ensuring integration of development security with business planning

5.2 Chief Information Security Officer (CISO)

The CISO is responsible for:

- Developing and maintaining secure development policies and procedures
- Overseeing development of security architecture and standards
- Coordinating secure development activities across the organization
- Monitoring development security compliance and performance
- Reporting development security status to senior management

5.3 Development Security Team

The development security team is responsible for:

- Implementing and maintaining development security controls
- Conducting security assessments of development processes and outputs
- Providing security guidance and support to development teams
- Performing security testing and vulnerability assessments
- Responding to development-related security incidents

5.4 Development Team Leads

Development team leads are responsible for:

- Ensuring team compliance with secure development policies and procedures
- Integrating security requirements into development planning and execution
- Coordinating security activities within development projects
- Monitoring team adherence to secure coding practices
- Reporting development security issues and concerns

5.5 Software Developers

Software developers are responsible for:

- Following secure coding standards and practices
- Participating in security training and awareness programs
- Conducting peer reviews and security assessments
- Implementing security controls and features as specified
- Reporting security vulnerabilities and concerns

5.6 Quality Assurance Team

The quality assurance team is responsible for:

- Integrating security testing into quality assurance processes
- Conducting security-focused testing and validation
- Verifying implementation of security requirements
- Documenting and tracking security-related defects
- Supporting security incident investigation and response

5.7 DevOps and Infrastructure Team

The DevOps and infrastructure team is responsible for:

- Implementing secure deployment and configuration management
- Maintaining security of development and deployment infrastructure
- Integrating security tools into CI/CD pipelines
- Monitoring development environment security
- Supporting secure development tool implementation and maintenance

6. Secure Development Lifecycle Implementation

6.1 Requirements and Planning Phase

Security shall be integrated into requirements gathering and project planning:

Security Requirements Identification: Security requirements shall be identified based on risk assessments, regulatory obligations, and business needs, and integrated into functional requirements documentation.

Threat Modeling: Threat modeling shall be conducted to identify potential security threats, attack vectors, and required security controls for the planned software.

Risk Assessment: Security risks associated with the planned software shall be assessed and appropriate risk treatment measures identified and planned.

Compliance Analysis: Applicable regulatory and contractual security requirements shall be identified and incorporated into development planning.

Resource Planning: Adequate resources for security activities shall be identified and allocated throughout the development project lifecycle.

6.2 Design and Architecture Phase

Secure design principles shall be applied during software architecture and design:

Security Architecture: Security architecture shall be developed that specifies security controls, data flows, trust boundaries, and security mechanisms.

Design Reviews: Security design reviews shall be conducted to evaluate the adequacy and effectiveness of proposed security controls and architecture.

Interface Security: Security requirements for system interfaces, APIs, and data exchanges shall be specified and documented.

Data Protection Design: Data protection mechanisms including encryption, access controls, and data handling procedures shall be designed and specified.

Authentication and Authorization: Authentication and authorization mechanisms shall be designed to support appropriate access controls and user management.

6.3 Implementation and Coding Phase

Secure coding practices shall be implemented throughout software development:

Coding Standards: Secure coding standards appropriate to the programming languages and platforms shall be followed consistently across all development activities.

Input Validation: Comprehensive input validation shall be implemented to prevent injection attacks and data corruption.

Output Encoding: Appropriate output encoding shall be implemented to prevent cross-site scripting and other output-based attacks.

Error Handling: Secure error handling shall be implemented that prevents information disclosure while providing appropriate user feedback.

Cryptographic Implementation: Cryptographic functions shall be implemented using approved algorithms, libraries, and key management practices.

6.4 Testing and Validation Phase

Comprehensive security testing shall be conducted throughout development:

Static Code Analysis: Automated static code analysis shall be performed to identify potential security vulnerabilities and coding standard violations.

Dynamic Security Testing: Dynamic security testing shall be conducted to identify runtime vulnerabilities and security control effectiveness.

Penetration Testing: Penetration testing shall be performed by qualified personnel to assess the overall security posture of developed software.

Security Functionality Testing: Security features and controls shall be tested to verify correct implementation and effectiveness.

Integration Testing: Security aspects of system integration shall be tested to ensure secure interactions between components and systems.

7. Development Environment Security

7.1 Access Control and Authentication

Development of environments shall implement comprehensive access controls:

Developer Authentication: All access to development environments shall require strong authentication using approved authentication mechanisms.

Role-Based Access: Access to development resources shall be granted based on job roles and responsibilities using the principle of least privilege.

Privileged Access Management: Administrative and privileged access to development

systems shall be controlled, monitored, and regularly reviewed.

Session Management: Development environment sessions shall be managed with appropriate timeouts, monitoring, and security controls.

Access Reviews: Development environment access shall be reviewed regularly to ensure continued appropriateness and business need.

7.2 Environment Isolation and Segmentation

Development environments shall be appropriately isolated and segmented:

Network Segmentation: Development networks shall be segmented from production networks with appropriate security controls and monitoring.

Environment Separation: Development, testing, staging, and production environments shall be logically and physically separated with controlled data flows.

Data Isolation: Sensitive production data shall not be used in development environments without appropriate protection and approval.

Resource Isolation: Development resources shall be isolated to prevent unauthorized access to production systems and data.

Change Control: Changes between environments shall be controlled through appropriate change management processes.

7.3 Source Code Management

Source code shall be protected through comprehensive management controls:

Version Control: All source code shall be managed through approved version control systems with appropriate access controls and audit trails.

Code Repository Security: Source code repositories shall be secured with appropriate authentication, authorization, and encryption controls.

Branching Strategy: Secure branching strategies shall be implemented to control code changes and protect stable code branches.

Code Backup: Source code shall be backed up regularly with appropriate protection and recovery procedures.

Intellectual Property Protection: Source code shall be protected as intellectual property with appropriate confidentiality and access controls.

7.4 Development Tool Security

Development tools and infrastructure shall be secured appropriately:

Tool Approval: Development tools shall be approved and assessed for security before deployment and use.

Tool Configuration: Development tools shall be configured securely with appropriate security settings and access controls.

Tool Updates: Development tools shall be maintained with current security updates and patches.

Tool Monitoring: Development tool usage shall be monitored for security events and policy violations.

License Management: Development tool licenses shall be managed appropriately to ensure compliance and security.

8. Secure Coding Practices

8.1 Input Validation and Sanitization

Comprehensive input validation shall be implemented in all software:

Validation Requirements: All input data shall be validated for type, length, format, and range according to defined specifications.

Sanitization Procedures: Input data shall be sanitized to remove or neutralize potentially malicious content before processing.

Injection Prevention: Input validation shall specifically prevent SQL injection, command

injection, and other injection-based attacks.

File Upload Security: File upload functionality shall implement appropriate validation, scanning, and storage controls.

API Input Validation: API endpoints shall implement comprehensive input validation and error handling.

8.2 Authentication and Session Management

Secure authentication and session management shall be implemented:

Authentication Mechanisms: Strong authentication mechanisms shall be implemented including multi-factor authentication where appropriate.

Password Security: Password handling shall follow secure practices including hashing, salting, and complexity requirements.

Session Security: Session management shall implement secure session creation, maintenance, and termination practices.

Token Management: Authentication tokens shall be generated, transmitted, and stored securely with appropriate expiration and validation.

Account Management: User account management shall implement appropriate controls for creation, modification, and deactivation.

8.3 Data Protection and Encryption

Data protection shall be implemented throughout software applications:

Data Classification: Data shall be classified and handled according to its sensitivity and protection requirements.

Encryption Implementation: Sensitive data shall be encrypted both at rest and in transit using approved encryption algorithms and key management.

Key Management: Cryptographic keys shall be managed securely throughout their lifecycle including generation, distribution, storage, and destruction.

Data Masking: Sensitive data shall be masked or tokenized in non-production environments and when displayed to unauthorized users.

Data Retention: Data retention and disposal shall be implemented according to established policies and regulatory requirements.

8.4 Error Handling and Logging

Secure error handling and logging shall be implemented:

Error Messages: Error messages shall provide appropriate user feedback without disclosing sensitive system information.

Exception Handling: Exception handling shall be implemented to prevent application crashes and information disclosure.

Security Logging: Security-relevant events shall be logged with appropriate detail for monitoring and incident investigation.

Log Protection: Log files shall be protected against unauthorized access, modification, and deletion.

Log Monitoring: Logs shall be monitored for security events and anomalies with appropriate alerting and response procedures.

9. Third-Party Component Management

9.1 Component Assessment and Approval

Third-party components shall be assessed and approved before use:

Security Assessment: Third-party components shall be assessed for known vulnerabilities, security features, and overall security posture.

License Review: Component licenses shall be reviewed for compatibility with organizational policies and legal requirements.

Vendor Evaluation: Component vendors shall be evaluated for security practices, support capabilities, and long-term viability.

Approval Process: Third-party components shall be approved through established processes before integration into development projects.

Documentation: Approved components shall be documented with security considerations, dependencies, and usage guidelines.

9.2 Vulnerability Management

Third-party component vulnerabilities shall be managed systematically:

Vulnerability Monitoring: Third-party components shall be monitored for newly discovered vulnerabilities and security advisories.

Risk Assessment: Component vulnerabilities shall be assessed for risk and impact on organizational systems and data.

Update Management: Component updates and patches shall be evaluated, tested, and applied according to established procedures.

Alternative Assessment: Alternative components shall be evaluated when security issues cannot be adequately addressed through updates.

Incident Response: Security incidents involving third-party components shall be responded to according to established incident response procedures.

9.3 Component Inventory and Tracking

Third-party components shall be inventoried and tracked throughout their lifecycle:

Component Registry: A comprehensive registry of approved third-party components shall be maintained with version and security information.

Usage Tracking: Component usage across development projects shall be tracked to support vulnerability management and licensing compliance.

Dependency Mapping: Component dependencies shall be mapped and documented to understand security implications and update requirements.

Lifecycle Management: Component lifecycle shall be managed including evaluation, approval, deployment, maintenance, and retirement.

Compliance Monitoring: Component usage shall be monitored for compliance with approval policies and license requirements.

10. Security Testing and Validation

10.1 Static Application Security Testing

Static analysis shall be integrated into development processes:

Automated Scanning: Automated static analysis tools shall be used to scan source code for security vulnerabilities and coding standard violations.

Rule Configuration: Static analysis tools shall be configured with appropriate security rules and standards for the programming languages and frameworks used.

Integration: Static analysis shall be integrated into development workflows and CI/CD pipelines for continuous security assessment.

Results Management: Static analysis results shall be reviewed, prioritized, and remediate according to established procedures.

Tool Maintenance: Static analysis tools shall be maintained with current rule sets and vulnerability signatures.

10.2 Dynamic Application Security Testing

Dynamic testing shall be conducted to identify runtime vulnerabilities:

Automated Testing: Automated dynamic testing tools shall be used to test running applications for security vulnerabilities.

Manual Testing: Manual security testing shall be conducted to identify vulnerabilities that may not be detected by automated tools.

Environment Testing: Security testing shall be conducted in environments that closely replicate production conditions.

Test Coverage: Security testing shall provide comprehensive coverage of application functionality and attack vectors.

Results Validation: Dynamic testing results shall be validated and verified to confirm the existence and exploitability of identified vulnerabilities.

10.3 Penetration Testing

Penetration testing shall be conducted by qualified personnel:

Testing Scope: Penetration testing scope shall be defined to cover critical application functionality and potential attack vectors.

Qualified Testers: Penetration testing shall be conducted by qualified internal or external security professionals.

Testing Methodology: Penetration testing shall follow established methodologies and industry best practices.

Results Documentation: Penetration testing results shall be documented with detailed findings, risk assessments, and remediation recommendations.

Remediation Verification: Penetration testing shall include verification of remediation efforts and re-testing of identified vulnerabilities.

11. Deployment and Configuration Management

11.1 Secure Deployment Practices

Software deployment shall follow secure practices and procedures:

Deployment Planning: Deployment activities shall be planned with appropriate security considerations and risk assessments.

Configuration Management: Deployment configurations shall be managed securely with appropriate version control and change management.

Environment Preparation: Target environments shall be prepared and secured before software deployment.

Deployment Verification: Deployed software shall be verified for correct installation, configuration, and security control implementation.

Rollback Procedures: Secure rollback procedures shall be established and tested to enable rapid recovery from deployment issues.

11.2 Configuration Security

Software configurations shall be secured appropriately:

Security Baselines: Security configuration baselines shall be established and maintained for deployed software.
Hardening Procedures: Software shall be hardened according to established security procedures and industry's best practices.

Default Settings: Default configurations shall be reviewed and modified to remove unnecessary features and improve security.

Configuration Monitoring: Software configurations shall be monitored for unauthorized changes and compliance with security baselines.

Change Control: Configuration changes shall be controlled through appropriate change management processes.

11.3 Production Environment Security

Production environments shall implement comprehensive security controls:

Access Controls: Production environment access shall be strictly controlled and limited to authorized personnel.

Monitoring: Production environments shall be monitored continuously for security events and performance issues.

Incident Response: Production security incidents shall be responded to according to established incident response procedures.

Backup and Recovery: Production systems shall be backed up regularly with tested recovery procedures.

Maintenance: Production systems shall be maintained with appropriate security updates and patches.

12. Training and Competency

12.1 Developer Security Training

Comprehensive security training shall be provided for development personnel:

Secure Coding Training: Developers shall receive training in secure coding practices appropriate to their programming languages and platforms.

Threat Awareness: Developers shall be trained in current security threats and attack techniques relevant to their development activities.

Tool Training: Developers shall receive training on security tools and technologies used in the development process.

Compliance Training: Developers shall be trained in applicable regulatory and contractual security requirements.

Ongoing Education: Continuing education shall be provided to maintain current knowledge of security threats and best practices.

12.2 Competency Management

Development security competencies shall be managed systematically:

Competency Requirements: Security competency requirements shall be established for different development roles and responsibilities.

Skills Assessment: Developer security skills shall be assessed regularly to identify training needs and competency gaps.

Professional Development: Opportunities for professional development shall be provided to enhance security capabilities.

Certification Support: Relevant security certifications shall be encouraged and supported for development personnel.

Knowledge Sharing: Knowledge sharing mechanisms shall be established to promote

security awareness and best practices.

13. Definitions

Application Programming Interface (API): A set of protocols and tools for building software applications that specifies how software components should interact.

Continuous Integration/Continuous Deployment (CI/CD): A development practice that involves automatically building, testing, and deploying code changes.

Cross-Site Scripting (XSS): A type of security vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users.

DevOps: A set of practices that combine software development and IT operations to shorten development of lifecycles and provide continuous delivery.

Dynamic Application Security Testing (DAST): A security testing methodology that tests running applications for vulnerabilities.

Injection Attack: A type of attack that involves inserting malicious code or commands into an application through input fields.

Integrated Development Environment (IDE): A software application that provides comprehensive facilities for software development.

Penetration Testing: A simulated cyberattack against a system to check for exploitable vulnerabilities.

Secure Development Lifecycle (SDLC): A framework that integrates security activities into every phase of software development.

Software Composition Analysis (SCA): The process of identifying and managing open source and third-party components in applications.

SQL Injection: A type of injection attack that involves inserting malicious SQL code into application queries.

Static Application Security Testing (SAST): A security testing methodology that analyzes

source code for vulnerabilities without executing the program.

Threat Modeling: A process for identifying and evaluating potential security threats to a system or application.

Version Control System: A system that tracks changes to files and coordinates work on those files among multiple people.

Vulnerability Assessment: The process of identifying, quantifying, and prioritizing security vulnerabilities in a system.

Web Application Firewall (WAF): A security solution that filters, monitors, and blocks HTTP traffic to and from web applications.

Zero-Day Vulnerability: A security vulnerability that is unknown to security vendors and has no available patch or fix.

14. References

- Information Security Policy
- Access Control Policy
- Change Management Policy
- Incident Response Policy
- Third-Party Risk Management Policy
- Data Classification Policy