NetGain systems	No of Pages	**1** of **7**
	Document Classification:	Internal
	Effective Date	1 Aug 2025
SECURE DEVELOPMENT LIFECYCLE AND APPLICATION SECURITY	Doc No	ISMS-TECH-10
	Revision	1.0

AMENDMENTS LOG

Revision History

Version	Revision Author	Reviewer / Approver	Date	Summary of Changes
1.0	Nor Asfiah Binte Jamalludin (ISMS MR)	James Chia (CEO)	1 Aug 2025	Initial Release

Contents

PURPOSE

This procedure defines the secure software development lifecycle practices adopted by the organization to ensure that applications and services are securely designed, developed, tested, deployed, and maintained.

SCOPE

This applies to all internally developed systems, services, APIs, and infrastructure-as-code used to support the delivery of the organization’s IT observability and security management solutions.

REFERENCE

ISO/IEC 27001 standard	Annex A 8.25 Secure development lifecycle
	Annex A 8.26 Application security requirements
	Annex A 8.27 Secure system architecture and engineering principles
	Annex A 8.28 Secure coding
	Annex A 8.29 Security testing in development and acceptance
	Annex A 8.31 Separation of development, test and production environments
	Annex A 8.33 Test information
	Annex A 8.34 Protection of information systems during audit testing
	Annex A 8.4 Access to source code
	Annex A 8.10 Information deletion
	Annex A 8.11 Data masking
	Annex A 8.12 Data leakage prevention

RESPONSIBILITIES & AUTHORITIES

Top Management has the prime responsibility and approval authority for this control.

The development team shall ensure that the code produce is as secure as possible, and that development, testing and operational environments are separated, and roles and responsibilities are segregated.

1 Software Development Lifecycle

The organization adopts a secure SDLC that includes the following phases and embedded security activities:

- Requirements
  - Functional and **security requirements** are gathered at the start of the project
  - Requirements include:
    - Authentication and access control
    - Encryption needs
    - Data retention, masking, and privacy requirements (if applicable)
    - Logging and audit trails
    - Integration with security monitoring

- Design
  - Architecture and data flow diagrams are prepared and reviewed.
  - Design includes input validation, output encoding, and session management practices.

- Development
  - Developers use secure coding practices based on OWASP, SEI CERT, etc.
  - Only approved libraries and frameworks are used.
  - Code is version-controlled using tools with access restrictions
  - Use of data masking is enforced for sensitive values (e.g., personally identifiable information) in logs and UIs.

- Testing
  - All new features and fixes undergo functional, regression, and security testing:
    - Static Application Security Testing (SAST)
    - Dynamic Application Security Testing (DAST)
    - Manual code reviews for critical modules
  - Test environments must not use production data unless anonymized or masked
  - Developers and testers must not share credentials or data access.
  - Test data is securely deleted after test cycles.

- Deployment
  - Deployment to production is handled via automated CI/CD pipelines.
  - Approval gates and manual review stages are included for major releases.
  - Infrastructure-as-Code is used for consistency and traceability.

- Maintenance and Monitoring
  - Post-deployment, applications are continuously monitored for performance and anomalies.
  - Logs are reviewed for signs of data leakage or unauthorized access.
  - Security updates and patches are applied regularly.

2 Environment Segregation

The following environment must be logically separated:

Environment	**Purpose**	**Access Control**
Development	Active coding and testing by developers	Developers only
Test	QA testing of code and features	QA/Test team and authorized DevOps
Production	Live environment used by customers	Read-only for developers; limited write access for DevOps and Ops teams

All environments use separate credentials, databases, and network configurations. No direct access from development or test environments to production systems is allowed.

3 Access to Source Code

Source code repositories are hosted on cloud-based version control systems. Access is granted based on least privilege and job role. MFA and logging are enforced, and branch protection rules are enabled (e.g., code review before merge).

4 Data Leakage Prevention (DLP) and Secure Information Deletion

Although the organization does not currently operate a dedicated DLP tool, compensating controls are implemented to prevent unauthorized disclosure or transfer of sensitive data across systems, users, and services.

- Employees must use only approved, cloud-based collaboration and storage platforms with built-in access control and encryption.
- Access to confidential data including personal data is restricted to authorized personnel based on job roles and the principle of least privilege.
- All company-managed devices must have: full disk encryption, updated antivirus/anti-malware, restriction of removable media (e.g., USBs)
- Outbound email systems are configured to block or warn users before sending emails with large attachments or unapproved file types.
- Administrative and audit logs are enabled on cloud services to detect unusual file sharing, downloads, or access patterns.

Developers are responsible for ensuring their environments do not retain unnecessary data. Deletion practices must follow the following principles:

- Automatic clean-up mechanisms are used in development and cloud environments (e.g., CI/CD cache expiration, artifact TTL).
- For cloud storage or persistent volumes, deletion follows cloud provider best practices, including deletion of backup snapshots or versions.
- Any personal data in test environments is masked before use and securely deleted afterward.
- Secure deletion (e.g., logical wipe or cryptographic erasure) is required for storage where deletion does not immediately remove data (e.g., object storage).

5 Protection during Audit and Testing

All testing activities including audits, vulnerability assessments and security tests must be pre-approved by the relevant System Owner. Testing must be scheduled outside of business-critical hours or during designated maintenance windows, to minimize disruption to services or clients. Affected stakeholders, including DevOps, engineering, and client support teams, must be informed in advance of test schedules.

Only masked, anonymized, or synthetic data is to be used for testing purposes, in accordance with data protection requirements. Any logs, screenshots, or artifacts collected during testing must be classified and securely stored with limited access.

6 Principles for Secure System Engineering

To ensure systems are resilient against threats and vulnerabilities throughout their lifecycle, the organization adopts security-by-design and security-by-default principles during the engineering of applications and systems.

The following secure engineering principles must be applied throughout all software development and system integration activities:

- Principle of Least Privilege: Give users and systems only the access they need to do their job – nothing more.
- Defense in Depth: Use multiple layers of security so if one layer fails, others still protect the system.
- Secure Defaults: Systems and apps should start out in the most secure configuration by default.
- Fail Secure: When something goes wrong, systems should block access or shut down safely – not leave data exposed.
- Minimize Attack Surface: Only enable the features and services you actually need to reduce ways an attacker could get in.
- Secure Input Handling: Always check and clean user input to prevent harmful data from breaking or hijacking your system.
- Secure Output Handling: Filter and encode what your system sends out, especially to users, to prevent exposing sensitive data or code.
- Secure Cryptographic Design: Use strong, modern encryption standards to protect data at rest and in transit—never invent your own.
- Secure Session Management: Manage user sessions carefully to prevent hijacking—like auto-logout, strong tokens, and secure cookies.
- Secure Logging and Monitoring: Track important system activity without exposing sensitive data and monitor for signs of attacks.
- Separation of Duties: Split responsibilities so no one person has full control over critical actions like code changes or deployments.
- Security Testing Integration: Build security checks into every stage of development, not just at the end.
- Data Protection and Privacy by Design: Design systems to protect personal and sensitive data from the start, not as an afterthought.

These engineering principles are not standalone. They must be embedded into each SDLC phase (requirements, design, development, testing, deployment) and reflected in coding standards, test plans and deployment configurations.