AIA Group

Application Security Standard (ITSR.040)

Incorporating

All legal entities

Document Details

Document Name	Application Security Standard
Document Version	V3.4
Originating Business Function	Group Information Security
Policy Owner	Fields, Christopher – Associate Director, Digital Security
Primary Policy Contact Person	Zhang, Duker – Digital Security, Information Security
Secondary Policy Contact Person	NA
Date of First Issuance	11-Aug-2014
Date of Last Approval	25-Feb-2021
Version Effective Date	25-Feb-2021
Notified to	N/A
Approved by	Group CTO
Review Frequency	At least once every 2 years or more frequently if needed
Next Review Date	25-Feb-2023
Document Type *Per Standard for Corporate Policy Governance*	Functional-specific Standard
Information Classification *Per Group Data Protection Standard*	Restricted
Related Policies and Standards	ITPR.001 Information Security Policy ITPR.004 IT Asset Management Policy ITSR.005 Technology Acceptable Use ITSR.004 Cryptography Standard ITSR.034 Security Logging and Monitoring Standard ITSR.037 User Identity and Access Management Standard ITSR.003 Information Security Incident Management Standard ITSR.036 Data Protection Standard ITSR.020 Third Party Security Standard ITSR.047 Mobile Application Security Standard ITPO.001 IT Change Management Policy

VERSION CONTROL

Version	Amendments	By	Date
1.0	First Release.	Kenneth Fung	11th Aug 2014
2.0	Second Release 1. Introduction of Security tollgate. 2. Introduction of Security Requirements in the SDLC process. 3. Simplification of the baseline security controls required for all applications. 4. Introduced detailed security requirements for web applications based on OWASP guidelines.	Tanvir Ahmed	2nd Dec 2016
3.0	Third Release 1. Update to section 3.2 for new standard deviation reference 2. Minor updates to section 4 roles and responsibility 3. Section 5.2.2 removed as it is covered in IDAM standard 4. Section 5.3 Data handling modified to Data protection 5. Refined section 5.5.3, 6 6. Modified section 7 –updated existing content with new requirements, Removed Mobile security 7. Modified section 7 –added Input validation, Output encoding, System Configuration, File Management, Memory management, Database security	Alekh Gadekar	21 Dec 2017
3.1	1. Added section 7.15 on application programming interface (API) security requirement. 2. Rename “Shall” to “Must” 3. Updated Appendix A – Glossary and Reference document.	Anand, Sandesh	13 Nov 2018
3.2	4. Update requirements in section 6.3 testing phase for security and penetration testing.	Jimmy Wong	28 Jan 2019
Revamp as per Corporate Policy Standard			
Version	**Amendments**	**Approval Date**	**Approved by**
3.3	Review, revamp standard and content to align with CPG requirement	03-Apr-2020	RCEs, Group CRO, Group General Counsel
3.4	Renamed “Local/Global Technology Risk” to “Local/Global Information Security”	25-Feb-2021	Group CTO

	Updated “should” to “must” Added Section 2.3.3 on grace period for compliance Refined the document flow by introducing Section 4 Incorporating Security Requirements into Software Development Life Cycle prior to Section 5 General Security Control Requirements for all Applications Modified Section 4 - added planning phase, threat modeling in requirement and design phase, integrated development environment analyses and source code assessment in construction phase Updated the requirement on Multi-factor Authentication in Section 5.1 Added Section 5.10 on Business Logic and Section 5.17 on application programming interface security Added Appendix A on API security best practices and Appendix B on DevSecOps pipeline		

DISTRIBUTION LIST

TITLES
Group and BU IT

Contents

1. Objectives

This standard outlines the minimum-security requirements on application development to ensure AIA information / data are protected and common application security controls are embedded in the software development lifecycle.

2. Scope and Definitions

2.1. Scope

All applications developed by/for AIA or purchased from third party for AIA Business (excluding office automation applications) must adhere to this standard.

Control requirements defined in this Standard support and facilitate compliance with relevant laws and regulations of the countries in which AIA conducts business. Where local laws and regulations require controls that are more restrictive than those identified in this Standard, those more restrictive control requirements must be fully complied with.

2.2. Definitions

Please refer to Appendix C of this Standard.

2.3. Assumption & limitations

2.3.1. This Standard enables compliance with related legal and regulatory requirements of those countries that AIA have local presence. In the event of contradiction or conflict between the two, the more restrictive one must be fully complied.

2.3.2. Upon approval of this Standard (approval date), a grace period of 6 month for compliance will be applied to the newly added/amended control requirements to be decided by Group Senior Management.

2.3.3. Local Information Security or IT Security is responsible for monitoring compliance of all mandatory security requirements and assisting in implementing appropriate measures.

2.3.4. This Standard must be read in conjunction with other relevant policies and security standards such as, but not limited, Cryptography Standard, Data Protection Standard, Cloud Security Standard and User Access and Identity Management Standard.

2.3.5. This Standard must be reviewed at least once every 2 years or more frequent if needed.

3. Roles and Responsibilities

Functional Roles	Responsibilities
Developer	• Align with industry best security practices to develop an application (e.g. OWASP, CERT) • Perform automated vulnerability scanning and static code analysis to identify vulnerabilities during application development. • Take AIA IT security policies and standards requirements into consideration during design phase and implement secure coding practices during application development lifecycle.
Application development team lead	• Review the automatic vulnerability scan results and ensure all vulnerabilities are remediated. • Review the penetration test and static code analysis results and ensure all vulnerabilities are remediated.
Application Department Head	• Ensure team members conform to this standard when developing applications. • Ensure secure code training is completed by team all relevant members for understanding of secure coding practices.
Local Information Security Team	• Perform application security review on the new projects or projects with major changes.
Group Information Security Team	• Oversight and governance of the Application Security process and review local application Information Security assessment.
Project Manager	• Ensure all the security requirements are considered from Requirement Phase and throughout the application development lifecycle.

4. Incorporating Security Requirements into Software Development Life Cycle ("SDLC")

Incorporating information security during the project development life cycle phases allows the security requirements to be matured and integrated in a cost-effective manner. Security must be included during the requirements generation phase of the project. Designing a solution with security consideration could substantially reduce the need for additional compensating security controls.

.

4.1. Planning Phase

4.1.1. An application risk classification assessment must be conducted to assign risk classification/sensitivity ratings to the business functions within the application. The identified risk classification must be rated, and security levels must be assigned based on the ratings obtained from the application characteristics.

4.2. Requirement and Design Phase

4.2.1. Threat modelling must be performed to proactively identify and enumerate threats to an application. Confirmed or potential vulnerabilities, related threats, and their counter measures pertaining to a software system should be documented.

4.2.2. All new projects which involve new application or major changes to an existing application must go through Security Assessment Service (SAS) Process to assess at least the minimum application security controls provided below where applicable for the project and provide appropriate recommendation

4.3. Construction Phase

4.3.1. Integrated Development Environment ("IDE") analyses must be conducted to analyse components such as software development platforms technologies, potentially critical functions, and third-party libraries etc. Code assessment tools must be integrated with IDE.

4.3.2. Source code assessment must be performed to examine security critical modules to identify common security anti-patterns. Identifying code security flaws can be achieved with automated and manual processes.

4.3.3. Vulnerability scanning and secure code review must be performed. For more details, please refer to Vulnerability Management Standard.

4.4. Testing Phase

4.4.1. All security testing must be coordinated via Group Office services and must be done at appropriate hours in accordance with Vulnerability Management Standard.

4.4.2. Penetration test and secure code review must be performed. For more details, please refer to Vulnerability Management Standard.

4.5. Deployment Phase

4.5.1. Project manager must review the operational readiness. All the changes performed during application transition to production environment must follow change management process.

4.5.2. For significant changes, testing of security controls must be re-conducted to ensure integrity of the security control mechanism remains the same.

4.5.3. The discovered vulnerabilities must be remediated and/or mitigated following the requirements in Vulnerability Management Standard.

5. General Security Control Requirements for all Applications

The security controls to be adopted for all applications are determined through the application risk classification process during the Planning phase of the SDLC process (Section 4.1.), and

also dependent on the nature and characteristics of the existing or to-be-developed system. The following minimum security requirements must be in place wherever applicable. For more details, please refer to OWASP Application Security Verification Standard.

5.1. Authentication

- User authentication mechanism (i.e. User account and password) must be in place to verify the identity of user.
- Require authentication for all pages and resources, except those specifically intended to be public.
- All authentication controls must be enforced on a trusted system (e.g. Server).
- Establish and utilize standard, tested, authentication services whenever possible
- Use a centralized implementation for all authentication controls, including libraries that call external authentication services
- Segregate authentication logic from the resource being requested and use redirection to and from the centralized authentication control.
- All authentication controls must fail securely.
- Authentication failure must result in generic error message.
- Authentication credentials for accessing services external to the application must be encrypted and stored in a protected location on a trusted system (e.g. server). The source code is NOT a secure location.
- Ensure that only cryptographically strong one-way salted hashes of passwords are stored and that the table/file that stores the passwords and keys is write-able only by the application.
- Password hashing must be implemented on a trusted system (e.g. Server).
- Password entry fields allow, or encourage, the use of passphrases, and do not prevent long passphrases/highly complex passwords being entered.
- Ensure all account identity authentication functions (such as update profile, forgot password, disabled / lost token, help desk or IVR) that might regain access to the account are at least as resistant to attack as the primary authentication mechanism.
- Re-authenticate users prior to performing critical operations.
- Use Multi-Factor Authentication for highly sensitive or high-risk transactional operations involving payments, and changing/updating personal information, etc.
- Ensure that the password changing functionality includes the old password, the new password, and a password confirmation.
- Credentials must be transported over a secure TLS channel. Any attempt to access the URL over non-secure channel must be prevented.

- Use only HTTP POST requests to transmit authentication credentials.
- The Password Reset function and other recovery paths must not reveal the current password. The new password must not be sent in clear text to the user.
- Temporary passwords and links must have a short expiration time.
- Enforce the changing of temporary passwords on next use.
- Notify users when a password reset occurs.
- Enforce password length and complexity requirements as defined in the AIA Identity and Access Management Standard.
- Password entry must be obscured on the user's screen. (e.g., on web forms use the input type "password")
- Ensure user enumeration is not possible via login, password reset, or forgot password functionality.
- Ensure that forgot password and other recovery paths use a soft token, mobile push, or an offline recovery mechanism.
- Enforce account disabling after an established number of invalid login attempts. The account must be disabled for a period sufficient to discourage brute force guessing of credentials, but not so long as to allow for a denial-of-service attack to be performed
- If knowledge-based questions (also known as "secret questions") are required, the questions must be strong enough to support sufficiently random answers.
- Strong measures must be in place to prevent the use of commonly chosen passwords and weak passphrases.
- Ensure that administrative interfaces are not accessible by untrusted parties.
- If using third party code for authentication, inspect the code carefully to ensure it is not affected by any malicious code.
- For more details on authentication requirements, please refer to Identity and Access Management Standard.
- Local regulatory requirement on authentication must be complied with if it is applicable.

5.2. Session Management

- Use of server or framework's session management controls. The application must only recognize these session identifiers as valid.
- Session identifier creation must always be done on a trusted system (server).
- Session management controls must use well-vetted algorithms that ensure sufficiently random session identifiers.

- Set the domain and path for cookies containing authenticated session identifiers to an appropriately restricted value for the site.
- Logout functionality must fully terminate the associated session or connection.
- Logout functionality must be available from all pages protected by authorization.
- Establish a session inactivity timeout that is as short as possible, based on balancing risk and business functional requirements. Ensure that session timeout is in line with User Identity and Access Management Standard.
- Disallow persistent logins and enforce periodic session terminations, even when the session is active. Especially for applications supporting rich network connections or connecting to critical systems. Termination times must support business requirements and the user must receive sufficient notification to mitigate negative impacts.
- If a session was established before login, close that session and establish a new session after a successful login.
- Generate a new session identifier on any re-authentication.
- Do not allow concurrent logins with the same user ID.
- Do not expose session identifiers in URLs, error messages or logs. Session identifiers must only be in the HTTP cookie header. Do not pass session identifiers as GET parameters.
- Protect server-side session data from unauthorized access, by other users of the server, by implementing appropriate access controls on the server.
- Generate a new session identifier if the connection security changes from HTTP to HTTPS, as can occur during authentication. Within an application, it is recommended to consistently utilize HTTPS rather than switching between HTTP and HTTPS.
- Ensure that all pages that require authentication have easy and visible access to logout functionality.
- Supplement standard session management for sensitive server-side operations, like account management, by utilizing per-session strong random tokens or parameters.
- Supplement standard session management for highly sensitive or critical operations by utilizing per-request, as opposed to per-session, strong random tokens, or parameters
- Set the "secure" attribute for cookies transmitted over an TLS connection.
- Set cookies with the HTTP Only attribute unless you specifically require client-side scripts within your application to read or set a cookie's value.

5.3. Access Control

- The principle of least privilege must be implemented. Users must only be able to access functions, data files, URLs, controllers, services, and other resources, for which they

possess specific authorization. This ensures protection against spoofing and elevation of privilege.

- Use only trusted system objects, e.g. server-side session objects, for making access authorization decisions.
- Use a single site-wide component to check access authorization. This includes libraries that call external authorization services.
- Access controls must fail securely.
- Deny all access if the application cannot access its security configuration information.
- Enforce authorization controls on every request, including those made by server-side scripts, "includes" and requests from rich client-side technologies like AJAX and Flash.
- Segregate privileged logic from other application code.
- Restrict access to files or other resources, including those outside the application's direct control, to authorized users only.
- Restrict access to protected URLs to authorized users only.
- Restrict access to protected functions to authorized users only.
- Restrict direct object references to authorized users only.
- Restrict access to services to authorized users only.
- Restrict access to application data to authorized users only.
- Restrict access to user and data attributes and policy information used by access controls.
- Restrict access to security-relevant configuration information to authorized users only.
- Server-side implementation and presentation layer representations of access control rules must match.
- If state data must be stored on the client, use encryption and integrity checking on the server side to catch state tampering.
- Enforce application logic flows to comply with business rules.
- Limit the number of transactions a single user or device can perform in a given period of time. The transactions/time must be above the actual business requirement, but low enough to deter automated attacks.
- Use the "referrer" header as a supplemental check only, it must never be the sole authorization check, as it is can be spoofed.

- If long authenticated sessions are allowed, periodically re-validate a user's authorization to ensure that their privileges have not changed and if they have, log the user out and force them to re-authenticate.
- Implement account auditing and enforce the disabling of unused accounts (e.g., after no more than 30 days from the expiration of an account's password.)
- The application must support disabling of accounts and terminating sessions when authorization ceases (e.g., Changes to role, employment status, business process, etc.)
- Service accounts or accounts supporting connections to or from external systems must have the least privilege possible.
- Create an Access Control Policy to document an application's business rules, data types, access authorization criteria and/or processes so that access can be properly provisioned and controlled. This includes identifying access requirements for both the data and system resources.
- Directory browsing must be disabled unless deliberately desired.
- The same access control rules implied by the presentation layer must be enforced on the server side.
- User, data attributes and policy information used by access controls must not be manipulated by end users unless specifically authorized.
- All access control decisions must be logged.
- The application or framework must use strong anti-CSRF tokens or another transaction protection mechanism.
- Access to program source code must be restricted and only shared on a need-to-know basis.
- Program source code must be stored in a secured source code repository and must be protected against unauthorized access, modification, and/or deletion.
- An audit log must be maintained for accountability for any read/write operation on the source code repository. Delete operation on source code must be restricted and must only be allowed for privileged accounts after approval.

5.4. Input Validation

- Conduct all data validation on a trusted system (Server).
- Identify all data sources and classify them as 'trusted' or 'untrusted'. Validate all data from untrusted sources (e.g., Databases, file streams, etc.).
- There must be a centralized input validation routine for the application.

- Specify proper character sets, such as UTF-8, for all sources of input.
- Encode data to a common character set before validation (Canonicalize).
- All validation failures must result in input rejection .
- Determine if the system supports UTF-8 extended character sets and if so, validate after UTF-8 decoding is completed.
- Validate all client provided data before processing, including all parameters, URLs and HTTP header content (e.g. Cookie names and values). Be sure to include automated post backs from JavaScript, Flash or other embedded code
- Verify that header values in both requests and responses contain only ASCII characters.
- Validate data from redirects. An attacker may submit malicious content directly to the target of the redirect, thus circumventing application logic and any validation performed before the redirect.
- Validate for expected data types.
- Validate data range.
- Validate data length.
- Validate all input against a whitelist of allowed characters, whenever possible
- If any potentially hazardous characters must be allowed as input, be sure that you implement additional controls like output encoding, secure task specific APIs and accounting for the utilization of that data throughout the application. Examples of common hazardous characters include: < > " ' % ( ) & + \ \' \"
- If your standard validation routine cannot address the following inputs, then they must be checked discretely. Check for null bytes (%00). Check for new line characters (%0d, %0a, \r, \n) .Check for “dot-dot-slash" (../ or ..\) path alterations characters. In cases where UTF-8 extended character set encoding is supported, address alternate representation like: %c0%ae%c0%ae/ (Utilize canonicalization to address double encoding or other forms of obfuscation attacks).

5.5. Output Encoding

- Conduct all encoding on a trusted system (Server).
- Utilize a standard, tested routine for each type of outbound encoding.

- Contextually output encode all data returned to the client that originated outside the application's trust boundary. HTML entity encoding is one example, but does not work for all cases.
- Encode all characters unless they are known to be safe for the intended interpreter.
- Contextually sanitize all output of un-trusted data for SQL, XML, and LDAP queries.
- Sanitize all output of un-trusted data for operating system commands.

5.6. Malicious input handling

- Ensure that the runtime environment is not susceptible to buffer overflows, or that security controls prevent buffer overflows.
- Ensure that server-side input validation failures result in request rejection and are logged.
- Ensure that input validation routines are enforced on the server side.
- Ensure that a single input validation control is used by the application for each type of data that is accepted.
- Ensure that all SQL queries, HQL, OSQL, NOSQL and stored procedures or calling of stored procedures are protected using prepared statements or query parameterization, and thus not susceptible to SQL injection.
- Ensure that the application is not susceptible to LDAP Injection, or that security controls prevent LDAP Injection.
- Ensure that the application is not susceptible to OS Command Injection, or that security controls prevent OS Command Injection.
- Ensure that the application is not susceptible to Remote File Inclusion (RFI) or Local File Inclusion (LFI) when content is used that is a path to a file.
- Ensure that the application is not susceptible to common XML attacks, such as XPath query tampering, XML External Entity attacks, and XML injection attacks.
- Ensure that all string variables placed into HTML or other web client code is either properly contextually encoded manually, or utilize templates that automatically encode contextually to ensure the application is not susceptible to reflected, stored and DOM Cross-Site Scripting (XSS) attacks.

5.7. Cryptographic Practices

- All cryptographic functions used to protect secrets from the application user must be implemented on a trusted system (Server).
- Protect master secrets from unauthorized access.
- Cryptographic modules must fail securely
- All random numbers, random file names, random GUIDs, and random strings must be generated using the cryptographic module's approved random number generator when these random values are intended to be un-guessable.
- Cryptographic modules used by the application must be compliant to AIA ITSR.004 Cryptography Standard.
- Establish and utilize a policy and process for how cryptographic keys will be managed.

5.8. Error Handling and Logging

- Application event logs recording user activities, exceptions, faults and information security events must be produced, kept and regularly reviewed using an automated monitoring system. The logs must not store any highly confidential data (PCI, PII). Please refer to the Security Logging and Monitoring Standard for more details.
- Do not disclose sensitive information in error responses, including system details, session identifiers or account information.
- Use error handlers that do not display debugging or stack trace information.
- Implement generic error messages and use custom error pages.
- The application must handle application errors and not rely on the server configuration.
- Properly free allocated memory when error conditions occur.
- Error handling logic associated with security controls must deny access by default.
- All logging controls must be implemented on a trusted system (Server).
- Logging controls must support both success and failure of specified security events.
- Ensure logs contain important log event data.
- Ensure log entries that include un-trusted data will not execute as code in the intended log viewing interface or software.
- Restrict access to logs to authorized individuals only.
- Utilize a master routine for all logging operations.
- Do not store sensitive information in logs, including unnecessary system details, session identifiers or passwords.
- Ensure that a mechanism exists to conduct log analysis.
- Log all input validation failures.
- Log all authentication attempts, especially failures.
- Log all access control failures.

- Log all apparent tampering events, including unexpected changes to state data.
- Log attempts to connect with invalid or expired session tokens.
- Log all system exceptions.
- Log all administrative functions, including changes to the security configuration settings.
- Log all backend TLS connection failures.
- Log cryptographic module failures.
- Use a cryptographic hash function to validate log entry integrity.

5.9. System Configuration

- Ensure servers, frameworks and system components are running the latest approved version.
- Ensure servers, frameworks and system components have all patches issued for the version in use.
- Turn off directory listings.
- Restrict the web server, process and service accounts to the least privileges possible.
- When exceptions occur, fail securely.
- Remove all unnecessary functionality and files.
- Remove test code or any functionality not intended for production, prior to deployment.
- Prevent disclosure of your directory structure in the robots.txt file by placing directories not intended for public indexing into an isolated parent directory. Then "Disallow" that entire parent directory in the robots.txt file rather than disallowing each individual directory.
- Define which HTTP methods, GET or POST, the application will support and whether it will be handled differently in different pages in the application.
- Disable unnecessary HTTP methods, such as WebDAV extensions. If an extended HTTP method that supports file handling is required, utilize a well-vetted authentication mechanism.
- If the web server handles both HTTP 1.0 and 1.1, ensure that both are configured in a similar manner or ensure that you understand any difference that may exist (e.g. handling of extended HTTP methods).
- Remove unnecessary information from HTTP response headers related to the OS, web-server version and application frameworks.
- The security configuration store for the application must be able to be output in human readable form to support auditing.
- Implement an asset management system and register system components and software in it.

- Isolate development environments from the production network and provide access only to authorized development and test groups. Development environments are often configured less securely than production environments and attackers may use this difference to discover shared weaknesses or as an avenue for exploitation.
- Implement a software change control system to manage and record changes to the code both in development and production.

5.10. Business Logic

- The business logic flow must be sequential, processed in order, and cannot be bypassed.
- Business logic must include limits to detect and prevent automated attacks, such as continuous small funds transfers, or adding a million friends one at a time, and so on.
- High value business logic flows must consider abuse cases and malicious actors, and have protections against spoofing, tampering, repudiation, information disclosure, and elevation of privileged attacks.

5.11. File Management

- Do not pass user supplied data directly to any dynamic ‘include’ function.
- Require authentication before allowing a file to be uploaded.
- Limit the type of files that can be uploaded to only those types that are needed for business purposes.
- Validate uploaded files are the expected type by checking file headers. Checking for file type by extension alone is not sufficient.
- Do not save files in the same web context as the application. Files must either go to the content server or in the database.
- Prevent or restrict the uploading of any file that may be interpreted by the web server.
- Turn off execution privileges on file upload directories.
- Implement safe uploading in UNIX by mounting the targeted file directory as a logical drive using the associated path or the chrooted environment.
- When referencing existing files, use a whitelist of allowed file names and types. Validate the value of the parameter being passed and if it does not match one of the expected values, either reject it or use a hard-coded default file value for the content instead.

- Do not pass user supplied data into a dynamic redirect. If this must be allowed, then the redirect must accept only validated, relative path URLs.
- Do not pass directory or file paths, use index values mapped to pre-defined list of paths. Never send the absolute file path to the client.
- Ensure application files and resources are read-only.
- Scan user uploaded files for viruses and malware.

5.12. Memory Management

- Utilize input and output control for un-trusted data.
- Double check that the buffer is as large as specified.
- When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string.
- Check buffer boundaries if calling the function in a loop and make sure there is no danger of writing past the allocated space.
- Truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.
- Specifically, for close resources, do not rely on garbage collection. (e.g., connection objects, file handles, etc.).
- Use non-executable stacks when available.
- Avoid the use of known vulnerable functions (e.g., printf, strcat, strcpy etc.).
- Properly free allocated memory upon the completion of functions and at all exit points.

5.13. Data Protection

- All sensitive and confidential data must be encrypted as stated in Data Protection Standard and Cryptography Standard.

5.14. Database Security

- Use strongly typed parameterized queries.
- Utilize input validation and output encoding and be sure to address meta-characters. If these fails, do not run the database command.
- Ensure that variables are strongly typed.
- The application must use the lowest possible level of privilege when accessing the database.
- Use secure credentials for database access.
- Connection strings must not be hard coded within the application. Connection strings must be stored in a separate configuration file on a trusted system and they must be encrypted.
- Use stored procedures to abstract data access and allow for the removal of permissions to the base tables in the database.
- Close the connection as soon as possible.
- Remove or change all default database administrative passwords. Utilize strong passwords/phrases or implement multi-factor authentication.
- Turn off all unnecessary database functionality (e.g., unnecessary stored procedures or services, utility packages, install only the minimum set of features and options required (surface area reduction))

5.15. Mobile Security

Refer to the Mobile Application Security Standard for details.

5.16. Web Services

- Ensure that the same encoding style is used between the client and the server.
- Ensure that access to administration and management functions within the Web Service Application is limited to web service administrators.
- Ensure that XML or JSON schema is in place and verified before accepting input.
- Ensure that all input is limited to an appropriate size limit.
- Ensure that SOAP based web services are compliant with Web Services-Interoperability (WS-I) Basic Profile at minimum.

- Ensure the use of session-based authentication and authorization.
- Ensure that the REST service is protected from Cross-Site Request Forgery.
- Ensure the REST service explicitly checks the incoming Content-Type to be the expected one, such as application/xml or application/json.
- Ensure that the message payload is signed to ensure reliable transport between client and service.
- Ensure that alternative and less secure access paths do not exist.

5.17. Application Programming Interface (API) Security

- API criticality of the application must be assessed as part of the threat modelling process (Please refer to Section 4.2.1). An application shall be rated as API-critical when it exposes one or more characteristics of the following via API:
    - Privileged function
    - Sensitive data
    - Any data insertion / modification / deletion
    - Authentication / registration / authorization
    - File upload
    - Integration with cloud / external partners
    - Publicly accessible
    - Containing / processing any security or log related data
- API-critical application must document the schema for the API design and definition.
- For external-facing API-critical application, an API gateway must be used, and the schema must be validated by the API gateway.
- Other security controls shall be adopted wherever applicable, please refer to Appendix A for more details.

5.18. Application Production Changes Approval

- A list of application change approvers, including Technical Services Team and Business System Owner, for each IT system or business application must be maintained by local BU IT and approved by local CTO or CTO's delegate.

- Any changes to production systems must follow the Group IT Change Management Policy and be approved by the corresponding application and if necessary, infrastructure change approvers prior to the change implementation.

5.19. Application Hosted in Cloud

- For applications hosted in Cloud as “platform as a service” or “software as a service” please refer to the Cloud Security Standard and Third-Party Security Assessment (TPSA) for more information.

5.20. Capacity Management

- For all applications, capacity management requirements must be defined and reviewed on an agreed basis with the application or system owner.

5.21. Business Continuity Planning and Disaster Recovery

- Business Continuity and Disaster Recovery plan must be documented and approved for all applications.

6. Adoptions of DevSecOps

6.1. Applications adopting the use of DevSecOps to build, test, and release more frequently and reliably must consider the following governance measures:

6.1.1. Roles and responsibilities must be well defined in in cross functional DevSecOps teams to reduce delays and disruption from handoffs and control gates.

6.1.2. DevSecOps specific policies and procedures must be established to keep up with the pace of application development in a DevOps environment.

6.1.3. Automated security tools in the DevSecOps pipeline must be enabled to improve overall security by reducing vulnerabilities and security flaws due to human error.

6.1.4. Security monitoring and notification systems in DevSecOps must be set up for creating an automated audit trail throughout the software development lifecycle, which facilitates compliance reporting.

6.1.5. Continuously monitoring security metrics must be introduced for DevOps teams to constantly improve their security decisions and stay on top of the game.

6.2. Please refer to Appendix B for more details on key activities involved in DevSecOps pipeline.

7. Security Requirements for Software Procurement

Before procuring an application, buyer must evaluate the security requirements of the potential application to see whether it can conform to AIA IT security standards, including this Standard (as far as applicable to the application to be procured), and TPSA exercise must be undertaken (Please also refer to Third Party Security Standard for more details). When evaluating the application, the security requirements specified in Section 5 of this Standard must be analyzed and applicable requirements must be complied by the procured application. For non-applicable security requirements, please go through the TIM Process.

8. Exemptions

All requirements stipulated in this standard are mandatory. Any deviations from this Standard must be approved by Information Security through the Technology Issue Management Process.

8.1. Breach Management and Escalation

Any breaches of any requirements in this document prior to authorisation must be escalated to Local BU Information Security (for Local BU) or Group Information Security (for Group functions) and document owner.

Group and BU are required to go through the Technology Issue Management process with Information Security.

8.2. Monitoring, Review and Amendments

Local Information Security and Controls function are responsible for monitoring compliance of all mandatory security requirements and assisting to implement appropriate measures. Instances of non-compliance are required to be escalated and approved by Information Security through TIM process. This document will be reviewed on at least once every 2 years or as frequently as needed. Minor amendments or cosmetic changes such as typo, text or table alignment and formatting that do not have significant impact to the requirements discussed in this document; do not require re-approvals from appointed parties mentioned in Document Details.

8.3. Delegation of Authority

Controls functions are responsible to ensure the execution of this document requirement. No delegation of authority to any parties. Latest effective version of this document must

reside on Corporate Policy Portal (“CPP”) as per CPG standard. Any inquiries of this document must be made to the document owner or primary contact as listed.

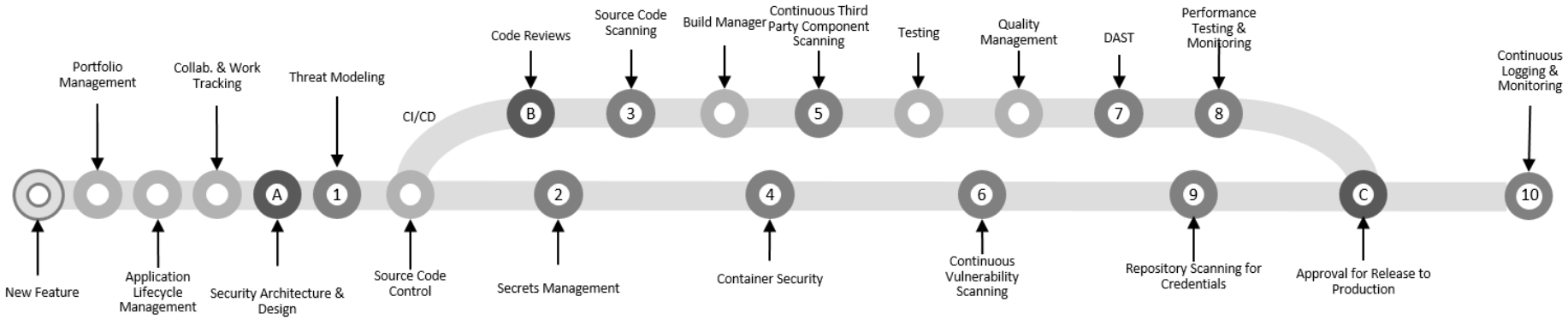
9. Approvals

This document is approved by the Group CTO.

10. Appendix A – API Security Best Practices

- All API traffic must be encrypted using based upon ITSR.004 Cryptography Standard.
- The application must define a reasonable rate limit for each user depending on business requirement and drop any request which exceeds the limit.
- In addition to standards described in Section 5.4, Section 5.5 & Section 5.6, the application must validate that each request conforms to a range of applicable inputs, based on business needs (input validation) and drops any request which does not conform to the list of applicable inputs.
- The application must treat the Access token as a session identifier. All standards relevant to Session IDs are applicable to access tokens (Refer to ITSR.037 User Identity and Access Management Standard for more details).
- For User to machine communication, if there is a business need for an active session for more than 15 minutes, the application must generate an Access token and a Refresh token on authentication.
- Refresh token must expire after no longer than 6 hours of inactivity.
- Refresh token must be securely stored on client devices (e.g. iOS keychain) in a manner than other malicious apps cannot access them.
- The application must use Proof Key for Code Exchange (PKCE) to securely transmit Refresh tokens to the server.
- API logging must follow the requirements as stipulated in ITSR.034 Security Logging and Monitoring Standard.
- API Request should validate the Authentication and Authorization at both front and back end prior serving the request at backend.
- All Public, Private and Partner APIs must go through following Tests:
    - Fuzzing Test,
    - Input validation Injection Attacks,
    - Parameter Tampering,
    - HTTP Method handling.
    - Error and Output Handling
    - Input and output domain validation

11. Appendix B – DevSecOps Pipeline



	Activity No.	DevSecOps Activity
Phase 1	A	Security Architecture and Design
	B	Code Reviews
	C	Approval for Release to Production
	3	Source Code Scanning
	4	Container Security
	5	Continuous Third-Party Component Scanning
	6	Continuous Vulnerability Scanning
	7	Dynamic Application Security Testing
Phase 2	1	Threat Modeling
	2	Secrets Management
	8	Performance Testing & Performance Monitoring
	9	Repository Scanning for Credentials
	10	Continuous Monitoring & Logging

12. Appendix C – Glossary

"must"	The use of the word "must" indicates a mandatory requirement.
"should"	The use of the word "should" indicates a requirement for good practice, which is important to be implemented whenever possible.
"may"	The use of the word "may" indicates a desirable requirement.
AIA	AIA Group Limited and each of its subsidiaries and branches
Business Unit	Business entity that is wholly-owned by AIA Group Limited or a subsidiary of AIA. Examples are: AIA Australia, AIA Singapore/Brunei, AIA China, AIA Hong Kong/Macau, AIA Financial, AIA Korea, AIA Insurance Lanka PLC, AIA Malaysia, AIA New Zealand, AIA PT, AIA Thailand, AIA Taiwan, AIA Vietnam, AIA Shared Services Hong Kong, TSS (BJ), TSS(GZ), OSS(ML), Philam Life
Accountability	Simply, who is responsible? Accountability is the condition of determining the ownership of the data. Ultimately, the owners of the data are responsible for ensuring the integrity, confidentiality, and availability of the data. The IT support functions support the owners as needed.
Availability	The condition whereby electronically stored information is where it needs to be, when it needs to be there and in the necessary form. It is closely related to the availability of the information processing technology. Whether because the process is unavailable (Denial of Service), or the information itself is somehow unavailable, makes no difference to the organization that depends on the information to conduct its business or mission. The value of the information's availability is reflected in the costs incurred, over time, by the organization, because the information was not available, regardless of the cause.
Business Data	Any data utilized or maintained by the business.
Clear	To use software or hardware products to overwrite all addressable storage space on the media with non-sensitive data.
Confidentiality	Ensuring that information is accessible only to those who are authorised to view or use the information. It is a cornerstone to information security.
Data Masking	A data protection technique that replaces existing characters with designated characters, symbols, or numbers.
Data Owner	The person or persons responsible for a set of data. This person has the ability to create, edit, modify, share and restrict access to the data.
Data Scrambling	A data protection technique that uses an algorithm to scramble data into an indecipherable form.
Electronic / Physical Media Disposal	The functionality of providing a process to ensure that the data is properly disposed of or eradicated to ensure that sensitive information is not retrievable via electronic or physical means. Shredding of documents and sanitizing electronic media are good examples to prevent dumpster diving and data retrieval from reused electronic media or storage devices.

Encryption (Storage)	The utilization of encryption to protect the data at rest is required for sensitive data. There are several methods which are applicable depending on circumstances. This includes: • Database encryption; • File encryption; • Disk encryption; • Data Element encryption within a database; and • Encryption for removable media. Further details are available in the Cryptography Standard.
Encryption (Transmission)	The utilization of network-based encryption from point-to-point to ensure that the security of the data remains intact during transmission. Several technical solutions apply to this issue including the use of HTTPS, SSL, SSH, SFTP, as well as other protocols using approved encryption algorithms. Further details are available in the Cryptography Standard.
Integrity	Is the data correct and unaltered? Integrity is the condition that information in or produced by the IT environment accurately reflects the source or process it represents. Integrity may be compromised in many ways, from data entry errors to the software errors to intentional modification. Integrity may be thoroughly compromised, for example, by simply contaminating the account numbers of policies. Since the account numbers are a primary reference for all associated data, the information is effectively no longer available. Technically, if a single character is wrong in a file with millions of records, the file's integrity has been compromised.
Proprietary Information	AIA Proprietary information is any information, tangible or intangible, that is created, acquired, or controlled by AIA and is not published or released to others without restriction, and/or AIA wishes to maintain in confidence. AIA also regards the information of others, which it is obligated to maintain in confidence, as proprietary information.
Sanitisation	The process to remove information from media such that data recovery is not possible. It includes removing all classified labels, markings, and activity logs.
User	For the purposes of this document, Users refer to all employees, temporary employees, contractors, consultants, etc. with access to AIA and its member companies' technology resources.

AIA Group

Application Security Standard

Version 3.4