Secure Coding Standard

Version # 1.3

Effective Date: 21/04/2025

Classification: Internal

Secure Coding Standard

Release Notice:

Document Name: Secure Coding Standard

Document Type: Standard

Version: 1.3

Release Date: 21/04/2025

Version Control Table				
Version No.	**Date Revised**	**Change Description**	**Prepared By**	**Reviewed By**
1.0	Dec 2021	Newly created	Governance Officer	GRC Sr. Manager
1.1	21/02/2023	Update and change the theme	Cybersecurity Risk Officer	Cybersecurity Risk and Compliance Manager
1.2	March 2024	Update and Review	Sr. Cybersecurity Compliance Officer	GRC Senior Manager
1.3	Feb 2025	Review and Update	GRC Officer	GRC Director
	Jan 2026	Next Review		

Secure Coding Standard

Document Approval

This document has been duly reviewed and approved by the concerned departments shown below:

Prepared By:			
Department:	GRC		
Job Title:	GRC Officer		
Incumbent:	Khaled M AlSulaiman		
Signature:		Date:	

Reviewed By:							
Department:	**Business Excellence**			**Department:**	GRC		
Job Title:	Quality Manager			Job Title:	ACT GRC Director		
Incumbent:	Rayan T Alahmadi			Incumbent:	Saad A AlQahtani		
Signature:		Date:		Signature:		Date:	

Approved By:			
Department:	Cybersecurity		
Job Title:	VP of Cybersecurity		
Incumbent:	Ayman S AlFadhel		
Signature:		Date:	

Authorized By:			
Department:	**CEO**		
Job Title:	CEO		
Incumbent:	Ahmed Alangari		
Signature:		Date:	

Table of Contents

Secure Coding Standard

1. Objective

The purpose of this standard is to provide cybersecurity requirements based on best practices and standards related to software and application development in Salam and protect them from internal and external threats to reduce cyber risks and protect them from internal and external threats by focusing on the basic objectives of protection: confidentiality, integrity, and availability of information.
This standard aims to comply with the requirements of cybersecurity and the relevant legislative and regulatory requirements and it is a legislative requirement in control no. 1-3-6-1 from the Essential Cybersecurity Controls (ECC-1:2018) that issued by the National Cybersecurity Authority (NCA).

2. Scope

This standard covers all software and application development projects and practices in Salam and it applies to all Salam's employees and contractors.

3. Standard

1	Secure Code Development
Target	Provide cybersecurity requirements to ensure that software, applications, and development process are protected following the cybersecurity controls.
The potential risks	Insecure application development can create vulnerabilities that can be exploited to compromise Salam data's confidentiality, availability, and integrity.
Required procedures	
1-1	Develop a secure software development lifecycle process (SSDLC) and implement it.
1-2	Develop methodology and process "Development, Security and Operations" (DevSecOps) and follow it.
1-3	Ensure that cybersecurity requirements are provided in the initial stages of software development and are integrated into the secure software development lifecycle (SSDLC).
1-4	Ensuring cybersecurity testing in the software development testing phases and integrating it into the secure software development lifecycle (SSDLC).
1-5	Design and prepare a secure environment for development, testing, and quality assurance purposes.
1-6	Apply safe application development guidelines according to the table (a).
1-7	Apply mitigation measures to the top 10 web application security risks according to the Open Web Application Security Project (OWASP) for sensitive systems and applications.
1-8	Implement mechanisms to restrict modification permissions to source code or data for production environments.
1-9	Requiring external suppliers to abide following the cybersecurity policies and standards adopted in Salam.
1-10	Obtain only modern, reliable, and licensed sources of software development tools, libraries, and components
1-11	Ensure the application of web application protection controls following the approved web application protection policy and standard in Salam.

1-12	Use standardized encryption algorithms that are rigorously reviewed by relevant standards and procedures.
1-13	Verify that all versions of software purchased from outside Salam are backed by the developer and appropriately secured based on the developer's security recommendations.
1-14	Training all software development personnel to write the appropriate source code for the programming language and development environment used.
2	**Source Code Repository**
Goal	Provide cybersecurity controls to ensure source code, libraries, and source code repository are protected.
The potential risks	If the source code and libraries are not adequately protected, the source code in Salam can be compromised, tampered with, or unauthorized access to it.
Required procedures	
2-1	Use a secure source code repository with identity verification, issuance, censorship, and login procedures in place.
2-2	Implementation of measures to prevent anyone from accessing the source code and source code repository except for application developers and their responsible parties.
2-3	Use a uniform numbering plan for version controls that shows the date when updated versions of the software were installed.
2-4	Archive older versions of the source code periodically.
2-5	Separate the source code of applications in development from the source code of applications in a production environment.
2-6	Archive the source code of expired applications so that they can be retrieved when needed.
2-7	Obtain a copy of the source code for all applications developed by third parties for Salam and store in the source code repository.
2-8	Developing standards for immunization and security of containers and virtual copies of the system (Docker) and best security practices guidelines and implementation.
2-9	Install secrets management mechanisms to manage secrets, keys, and certificates and prevent secrets from being stored in containers.
2-10	Use copies of containers from certified sources.
2-11	Use a special container registry to ensure that only certified and secure container copies are downloaded to the system so that each copy can be scanned for known and common vulnerabilities.
2-12	Not managing containers with user accounts with high authority and privileges.
3	**Secure Code Review and Testing**
Goal	Provide cybersecurity controls to ensure source code, libraries and source code repository are protected.

The potential risks	If the source code and libraries are not adequately protected, Salam may face significant security breaches, the source code can be compromised, tampered with, or have unauthorized access to.
Required procedures	
3-1	Regularly reviewing the source code of internally developed web applications.
3-2	Apply static and dynamic analysis tools to verify adherence to safe application development practices for internally developed software.
3-3	Regularly review source code security for all applications developed for Salam by outside parties.
3-4	Reviewing and approving the security controls of the internally developed applications before they are installed in the production environment.
3-5	Reassess and re-accredit existing internally developed applications after a major change has been made to them or after a specified period of time.
3-6	Perform a risk assessment of all applications under development or purchased to determine the controls required to reduce application risks to acceptable levels prior to installation in a production environment (refer to the approved risk management policy at Salam).
3-7	Conduct cybersecurity compliance testing for software based on cybersecurity policies approved in Salam before installation in a production environment.
3-8	Use of the Application Protection Verification Standard issued by the Open Web Application Security Project (OWASP) as a guideline for defining security requirements and creating test cases for reviewing sensitive systems and applications.
3-9	Perform a software configuration review including review of settings, immunizations, and update packages prior to installation in a production environment.
3-10	Conduct cybersecurity tests, including vulnerability assessment, penetration testing, and review of secure application development, prior to installation in a production environment.
3-11	Conduct cybersecurity tests, including vulnerability assessment and penetration testing, after installation in a production environment.
3-12	Addresses all security issues in developed applications that are discovered during a secure application development review prior to installation in a production environment.
3-13	Test developed applications to ensure proper separation of duties controls are applied.

3-14	Cancel test accounts that are in a non-production environment before moving applications into a production environment.
3-15	Logically separate the test and development environment from the production environment and other environments using network settings by setting and installing ACLs (ACL) and security policies on firewalls.
3-16	A peer review of the source code was performed by a developer who was not involved in writing any code prior to installation in a production environment in Salam.
3-17	Use of certified and licensed software security assessment tools and source code.
3-18	Conduct security tests for developed applications at all stages of the software development lifecycle testing (SDLC), including non-functional tests, unit testing (UT), systems integration testing (SIT), and user acceptance testing (UAT).
3-19	Establishing a process for managing software defects, vulnerabilities and security problems, and establishing and following up on them.
3-20	Inclusion of tests as part of continuous improvement and continuous development processes (CI/CD).

4. Table A - Safe Application Development Guidelines

1	**Broken Authentication (OWASP: A2:2017)**
1-1	Verify that all pages and sources require identity verification except for those specifically designated to be public (the principle of full and complete verification).
1-2	Verify that password fields do not show users' passwords when they are entered and that the autocomplete feature in password fields is turned off.
1-3	Verify that all identity verification controls fail securely to ensure that attackers cannot log in.
1-4	Verify that credentials and all other identity information the application handles do not pass through unencrypted or unsecured links.
1-5	Verify that the Forgot Password path and other recovery paths do not send current or new passwords unencrypted.
1-6	Verify that username enumeration attacks are executed is not enabled by the "Login", "Reset password" or "Forgot account" functions.
1-7	Verify that there are no default passwords in use for the application framework or any components used by the application (such as "admin/password").
1-8	Check for a Resource Governor to provide protection against Vertical Brute Forcing (an attack that attempts to hack a single account using all possible passwords) and Horizontal Brute Forcing (an attack that attempts to hack all accounts with a single password such as "Password1"). There should be no delay in entering the correct credentials. For example, the IP address of the speculative attack source should be set to close after 60 minutes, and the account to be closed after 15 minutes. Both control mechanisms must be operative simultaneously to protect against diagnostic and distributed attacks.
1-9	Verify that all identity verification controls are effective on the server side.

1-10	Verify that password fields allow the use of passphrases, do not prevent the use of too long or complex passphrases, and provide adequate protection against the use of bicycle passwords.
1-11	Verify that all account management functions, (such as registration, profile update, "forgot username", "forgot password", inactive/lost ID, helpdesk or IVR), which can restore account access, is able to withstand attacks at the same level as the basic identity verification mechanism.
1-12	Verify that users can change their credentials using an attack-resistant mechanism that is as resistant to attacks as basic identity verification. When changing passwords, the current password must be entered before the new password is entered and this is followed by a re-verification of the user.
1-13	Verify that credentials expire after an administratively set period of time. The password expiration period should be short based on the sensitivity of the application, which forces a faster password change.
1-14	Verify that all identity verification decisions are recorded including 'line spacing' and 'temporary locks'.
1-15	Verify that account passwords are hashed using a special random hash method for each account (such as Internet user identity or account creation) and shortened before storage.
1-16	Verify that all identity verification credentials for accessing external services for the application are encrypted and stored in a protected location (not in source code).
1-17	Verify that forgotten password and recovery paths send a time-limited multi-element activation or verification code (eg text messages, tokens, mobile apps, etc.) rather than the password.
1-18	Verify that the "Forgot Password" function does not lock or deactivate the account until the user has successfully changed the password.
1-19	Checking that there are no common cognitive questions and answers (so-called "secret" questions and answers).
1-20	Verify that the system can be set up and set so that it does not allow the use of configurable numbers from previous passwords.
1-21	Verify that all identity checks are implemented centrally (including libraries that call external validation services).
1-22	Verify request re-verification of identity, setup verification, variable identity verification, text message, two-factor application or transaction signature before allowing any sensitive operations on the application according to the risk profile of the application.
1-23	Check for functionality to deactivate or revoke user credentials in the event of a security breach.
1-24	Verify password encryption in accordance with relevant standards and procedures.
1-25	If the application manages a credential store, it must ensure that a one-way shorthand is stored in an encrypted manner with a high degree of complexity for passwords, and that the table and file that stores passwords and keys can only be written by the application. (You should not use an algorithm"MD5" as much as possible).
1-26	Separate the identity verification logic from the source being requested, and use redirection to and from centralized verification control.

1-27	Identity verification failure messages should not indicate the incorrect portion of the verification data. For example, instead of using "Incorrect Username" or "Incorrect Password", "Incorrect Username or Incorrect Password" should be used for both cases. Error messages must be identical in the source code and when viewed.
1-28	The password complexity requirements in the policy or regulation must apply, and the identity verification credentials must be sufficient to counter attacks that are common to threats in the installation environment. It must be verified that the password includes, at a minimum, the following: • at least one capital letter (AZ). • at least one lowercase letter (az). • At least one number (0-9). • At least one special character like:(!” #$%&'()*+,-./:;<=> ?@ [\]^_`{|}~”). It should also be checked that the password does not include at least the following: • More than two consecutive identical numbers or symbols (eg "111" and"aa"). • sequential numbers or symbols (such as "123", "789", or abc"). • The same username. • dictionary words ("password”, or “p@ssw0rd”, or “secret123”).
1-29	Enforce account deactivation after a specified number of incorrect login attempts (for example, five attempts for non-critical applications and three attempts for sensitive applications). The account must be deactivated for a certain period of time sufficient to thwart a credential speculative attack, provided that this period is not long enough to allow a DDoS attack to be carried out (eg deactivation for only 30 minutes).
1-30	The user must be informed of the last use of the account (whether successful or not) upon successful login.
2	Broken Authentication (OWASP: A2:2017)
2-1	Verify that the application is being used to implement automatic session management control of the framework.
2-2	Check that sessions are invalidated when the user is logged out.
2-3	Verify that sessions have expired after a certain period of inactivity.
2-4	Verify that all pages that require identity verification to access include logout links.
2-5	Verify that the identity of the session is never exposed except in cookie addresses (Cookie Headers), specifically in the URL bar)), error messages, or logs. This includes verifying that the application does not support URL rewriting of profile sessions.
2-6	Verify that the session identity has been changed or cleared when logging out.
2-7	Verify that session identifiers authenticated with cookies are protected using the "HttpOnly" (not showing cookies to the user).
2-8	Verify that authenticated session identifiers with cookies are protected by "Secure" and that STP addresses exist (eg: "Strict-Transport-Security: max-age=60000; includeSubDomains").
2-9	Verify that session identity has been changed when logging in to prevent session data theft.

2-10	Verify that the session identity has been changed when the identity is re-verified.
2-11	Verify that the application recognizes outgoing session identities by the application framework itself and considers only those identities to be valid.
2-12	Verify that authenticated session IDs are long and random enough to counter attacks that are common threats in the installation environment.
2-13	Verify that authenticated session IDs that use cookies have a path specified with an exclusive value that is appropriate for that site. Domain cookie property restriction should only be specified if business requires it, such as a single sign-on process.
2-14	Verify that the application does not allow duplicate simultaneous user sessions originating from different devices.
2-15	Verifies that sessions have expired after the maximum administratively set time period has passed regardless of activity.
2-16	Issuance of a new session ID if the connection security changes from HTTP to Hypertext Transfer Protocol Secure (HTTPS), which may occur during the identity verification process. It is recommended that you constantly use HTTPS in the application rather than switching between HTTP and HTTPS.
3	**Broken Access Control (OWASP: A5:2017)**
3-1	Verify that users can only access secure functions or services for which they have special permissions.
3-2	Verify that users can only access secure addresses (Secured URLs) for which they have special permissions.
3-3	Verify that users can only access secure data files for which they have special permissions.
3-4	Verify that direct object references are protected so that only authorized objects can be accessed per user.
3-5	Check that directory browsing is disabled unless required.
3-6	Verify that the user can only access protected information for which he has special permissions (for example, by implementing controls to protect object references from direct manipulation and unauthorized access to data).
3-7	Verify that access controls have failed securely.
3-8	Verify that the same access control rules included in the presentation layer are applied to the server by user role, so that controls and criteria cannot be re-enabled or re-added by users with higher privileges and powers.
3-9	Verify that all user properties, data and policy information used by access controls cannot be manipulated by users unless specifically authorized to do so.
3-10	Verify that all access controls are effective from the server side.
3-11	Verify that access control decisions can be logged and that all unsuccessful decisions are logged.
3-12	Verify that the application or framework issues complex, random Cross-Site Request Forgery ("CSRF") tokens that are specific to the user as part of all high-value transactions or access to

	protected information, and that the application verifies that these identifiers are of the appropriate value for the current user when processing these requests.
3-13	Cumulative access control protection – Verifying that the system can protect against cumulative or continuous access to protected functions, resources, or data, for example by using a resource governor to limit the number of hourly registrations or prevent an individual user from Pull data from the entire database.
3-14	Verify that there is a central mechanism (including libraries that call external permission and permission services) to control access to each type of protected resource.
3-15	Verify that the logic is separated from other application codes.
3-16	Apply appropriate access controls to protected information stored on the server. This information includes stored data, temporary files, and data that can only be accessed by specific system users.
3-17	Verify that service accounts or accounts that support connections to or from external systems have minimal rights and privileges.
3-18	Verify the auditing application and deactivate unused accounts (for example, after more than 30 days have passed since the expiration date of the account password).
3-19	If validated long sessions are allowed, the user's permissions and permissions should be periodically re-verified to ensure that their privileges do not change, and if they do, the user should be logged out and forced to perform the re-verification process.
3-20	Verify that the application supports deactivation of accounts and termination of sessions when permissions and permissions are interrupted (for example, when there is a change in role, employment status, business process, etc.).
4	**Injection & Cross-Site Scripting (OWASP: A7:2017)**
4-1	Verify that the environment is not vulnerable to buffer overflow, and that security controls prevent buffer overflow.
4-2	Verify that the environment is not vulnerable to SQL injection, and that security controls prevent SQL Injection.
4-3	Verify that the environment is not vulnerable to cross-site script injection (XSS), and that security controls prevent script injection across sites (XSS).
4-4	Verify that the environment is not vulnerable to simple directory access protocol injection (LDAP Injection) and that security controls prevent (LDAP Injection).
4-5	Verify that the operating environment is not vulnerable to OS command injection, and that security controls prevent OS command injection.
4-6	Check the data type, range, and length (if applicable).
4-7	When you need to allow potentially dangerous code as input, make sure that additional controls are in place such as coding the input, protecting the task APIs, and knowing who is using that data for the life of the application. Examples of common hazard symbols include: (< > " ' % ( ) & + \ \' \").
4-8	Ensure that all input validations are performed by a central application validation routine.
4-9	Verify that all failed validations result in input being rejected or validated.

4-10	Verify that all verification or application development procedures are implemented and enforced on the server.
4-11	Verify that all untrusted data that is output to HTML (including HTML elements and properties, JavaScript data values, CSS blocks, and URL properties) are discarded. appropriate to the content of the application.
4-12	Verify that symbol sets, such as"UTF-8", specified for all input sources.
4-13	Verify that all data entered is uniform for all data decoding or interpretation software sent to the client before authenticating it.
4-14	If the application framework allows massive automatic parameter assignment (also called automatic variable binding) from an incoming request to a form, it must verify that security-sensitive fields such as "account balance", "role" or "password" are protected from malicious automatic binding.
4-15	Verify that the application is protected from HTTP variables contamination attacks (HTTP), especially if the application framework does not distinguish between the sources of request variables (eg GET request, POST request, cookies, addresses, environment, etc.).
4-16	Verify that the application uses a single input validation control for each type of data that is accepted.
4-17	Verify that all input validation failures are logged.
4-18	Verify that each type of output encoding or disposition performed by the application has a single security control for the intended destination.
5	**Insecure Deserialization (OWASP: A8:2017)**
5-1	Apply information integrity checks, such as digital signatures, to any serialized objects to prevent the creation of hostile objects or data tampering.
5-2	Enforce specific constraints during deserialization before object creation because the code usually expects a selectable set of classes. It is not recommended to rely solely on this method as there are ways to bypass it.
5-3	Isolate the deserialized code and run it in low-featured environments where possible.
5-4	Log deserialization exceptions and failures, such as cases where the received type is not the expected type or where exception deserialization is specified.
5-5	Restrict or monitor incoming and outgoing network interconnection from deserialized containers or servers.
5-6	Monitor deserialization and alert if the user is constantly deserializing.
6	**Protected Data Exposure (OWASP: A3:2017)**
6-1	Verify that all encryption functions used to protect secrets from the application user are applied to the server.
6-2	Verify that all ciphers fail securely.
6-3	Verify that any key secrets are protected from unauthorized access (the master secret is application credentials stored as plaintext on disk which are used to protect access to security settings information).

6-4	Verify that all random numbers, random file names, and uniform identifiers are included (GUIDs), and strings from the cryptographic model random number generator, when the goal of these random values is to make the attacker unable to guess them.
6-5	Verify that the encryption models used in the application have been verified in accordance with the relevant policies and procedures.
6-6	Verify that the encryption models are operating in their approved system in accordance with the relevant policies and procedures.
6-7	Verify that there is an explicit policy on how cryptographic keys are managed (eg how they are issued, distributed, revoked, and expire) and that this policy is properly applied.
6-8	Check for Non-Repudiation through encryption (digital signature) of financial transactions, e-commerce and records.
6-9	Verify that all encryption keys are properly protected. In the event of a security breach, the key cannot be trusted and must be replaced or revoked.
6-10	Verification of encryption of identifiable information (PII), protected information and data stored when not in use.
7	**Insufficient Logging & Monitoring (OWASP: A10:2017)**
7-1	Ensure that explicit error checking is performed for internally developed software, and is documented for all inputs, including size, data type and allowed ranges or formats.
7-2	Verify that the application does not display error messages or accumulate traces of protected information, including session identity and personal information, that can help an attacker carry out their activities.
7-3	Verify that all error handling operations are performed on trusted devices.
7-4	Verify that all registry controls are applied to the server.
7-5	Verify that the error handling logic in the security controls automatically blocks access.
7-6	Verify that security logging controls allow for logging of success and failure events that have been identified as a security task.
7-7	Verify that each event in the log includes a timestamp from a trusted source, event severity, an indication that the event is security significant (if mixed with other logs), the identity of the user who caused the event (if a user is associated with the event), and a source protocol address Internet for the request accompanying the event, whether the event was successful or unsuccessful, and a description of the event.
7-8	Verify that all records are protected from unauthorized access and modification.
7-9	It shall be verified that the application does not log application-specific protected data that could assist an attacker, including user's session IDs and personal or protected information.
7-10	Checks availability of the Log Analyzer allowing the analyzer to search for log events based on a combination of search criteria on all fields in the system supported log syntax.
7-11	Verify that not all events with untrusted data are executed as code in the respective log review software.
7-12	Verify that there is a standardized registration implementation used in the application.

7-13	Verify that records have a regular, standard procedure for backups or archiving.
7-14	Implement "exception handling in cipher" where possible.
7-15	Verify that the logs below are enabled: • A record of all failures to verify entries. • A record of all identity verification attempts, especially failures. • A record of all access control failures. • A record of all apparent manipulation events, including unexpected changes to the state of the data. • A history of all attempts to connect to expired or invalid session IDs. • Record includes all system exceptions. • A record of all administrative functions, including changes to security settings and configurations. • A record of all TLS connection failures to endpoint devices. • A record of all coding failures.
8	**Protected Data Exposure (OWASP: A3:2017)**
8-1	Verify that the storage of forms with client-protected information, including autocomplete properties, is disabled.
8-2	Verify that all protected information is sent to the server in the body of the HTTP message (HTTP), (i.e. prevent URL standards from being used to send protected data).
8-3	Verify that all stored or temporary copies of protected information stored on the server are protected from unauthorized access, and that temporary working files are deleted as soon as they are no longer needed.
8-4	Deactivating storage or saving temporary copies of pages containing protected information with the customer, verifying that these copies are protected from unauthorized access, or deleting or revoking their validity after the authorized user accesses them. (can use "Cache-Control: no-store with the HTTP address controller "Pragma: no-cache", which is less powerful, but compatible with older versions of HTTP 1.0).
8-5	Verify that a list of protected information handled by the application is defined, that there is an explicit policy on how to control access to that information, when it should be encrypted (while not in use, during transit and in use), and that that policy is properly applied.
8-6	Verify that there is a way to delete all types of protected information in the application at the end of the required retention period.
8-7	Verify that the application reduces the number of criteria sent to untrusted systems such as hidden fields and variables Ajax"" files, cookies, and address values.
8-8	Verify the application's ability to detect and alert about abnormal numbers of information requests, or to process high-value user role transactions such as screen swiping, automatic web service abstraction, or data loss prevention. For example, the average user should not be able to access more than 5 records per hour or more than 30 records per day.
8-9	Verify that credentials used by the application on the server, such as database connection, password, and encryption secret keys, are not embedded in the code. Any credentials must be stored in a separate settings file on a trusted system and encrypted.
8-10	Verify that the autocomplete properties are not enabled on forms except for forms that contain protected information, including identity verification.

9	Security Misconfiguration (OWASP: A6:2017)
9-1	Verifies that a path can be built from a trusted certificate authority for each Transport Layer Security Server Cryptographic Certificate, and that each server's certificate has been validated.
9-2	Verify that the latest version of Transport Layer Security is being used in all communications (including external connections and endpoint device connections) that are authenticated or have protected information or functionality.
9-3	Verify that TLS communication failures are logged to end point devices.
9-4	Verify the authentication of all communications with external systems that include protected information or functionality.
9-5	Verify that all communications with external systems that include protected information or functionality use an account that has been set up and given the minimum number of features and permissions necessary for the application to function properly.
9-6	Verify that transport layer security connections a failed connection does not result in an insecure (not encrypted) connection.
9-7	Verify that cryptographic certificate paths are built and validated for all client cipher certificates using trusted authorities and revocation information.
9-8	Check for a Transport Layer Security implementation is standardized to be used in the application and prepared to operate in an approved business system.
9-9	Verify that the specified symbol encoding is defined for all connections (such as "UTF-8").
10	**Security Misconfiguration & XML External Entities (OWASP: A6:2017 & OWASP:A4:2017)**
10-1	Verify that the application accepts only a specified set of HTTP request methods (HTTP) as a GET request and a POST request and that unused methods are prohibited.
10-2	Verify that each HTTP response (HTTP) includes a content type header that defines a secure character set (eg "UTF-8").
10-3	Verify that hypertext transfer protocol addresses (HTTP) and/or other mechanisms of older browsers are included in order to protect against click jacking attacks.
10-4	Verify that hypertext transfer protocol addresses (HTTP) in requests and responses include only ASCII codes.
10-5	Verify that less complex data formats such as JavaScript are used (JSON), and avoid making protected information serialized.
10-6	Update, repair or upgrade XML processors (XML) and libraries in use in the underlying application or operating system, use dependency checks, and update Simple Object Access Protocol (SOAP) to version 1.2 or later.
10-7	Deactivation of third-party XML and processing “DTD" in all XML parsers in the application as directed by the Open Project on Web Application Security "XXE Prevention".
10-8	Apply positive input validation on the server (allowing a specific list), filtering, or checking to prevent hostile data within XML documents, addresses, or nodes (XML).

10-9	Check that the file upload function is in the XML file format (XML) or Extensible Method Language (XSL validation using XML validation using XSD Companion Language validation) or a similar validation method.
10-10	Use of static application security testing tools (SAST) and Dynamic Application Security Testing (DAST) to help expose XXE in the source code, given that manually reviewing the code is the preferred method for large and complex applications with many overlaps.
10-11	If these controls cannot be applied, the use of default update packages, security gateways for APIs, or a web application firewall should be considered to detect third-party XML attacks (XXE), monitor and block.
11	**Using Components with Known Vulnerabilities (OWASP: A9:2017)**
11-1	Verify that there is no malicious code in any code developed or modified for the purpose of creating the application.
11-2	Ensures that the integrity of interpreted code, libraries, executables, and configuration files is verified using checksums or distinct text summary calculations.
11-3	Verify that all code that applies or uses identity checks have not been affected by any malicious code.
11-4	Verify that all code that applies or uses session management are not affected by any malicious code.
11-5	Verify that all code that applies or uses access controls are not affected by any malicious code.
11-6	Verify that all input validation controls are not affected by any malicious code.
11-7	Verify that all code that implements or uses output validation controls has not been affected by any malicious code.
11-8	Verify that all code that implements or uses the encryption model has not been affected by any malicious code.
11-9	Verify that all code that applies or uses error-handling controls are not affected by any malicious code.
11-10	Verify that all malicious activity has been subjected to quarantine protection technology (Sandboxing).
11-11	Verify that protected information stored in memory is quickly disposed of when it is no longer needed.
11-12	Update components with the latest updates and fixes when the user becomes aware of published vulnerabilities.
11-13	Eliminate unused dependencies, unnecessary properties, components, files, and documents.
11-14	Make an on-going inventory of client and server-side component versions (such as frameworks and libraries) and their dependencies using tools such as versions, and Dependency Check, retire.js, etc., continuous monitoring of resources such as the Common Vulnerability Enumeration (CVE) and the National Vulnerability Database (NVD) for component vulnerabilities, the use of software configuration analysis tools to automate the process, and subscription to alerts email for vulnerabilities related to components in use.

11-15	Obtain components only from official sources and via protected links, and prefer signed packages to reduce the chances of a malicious modified component.
11-16	Monitor libraries and components that are out of date or out of date with security updates and fixes. If the installation of update packages is not possible, the installation of default updates and fixes should be studied to monitor, detect, or protect against detected problems.
12	**Business Logic**
12-1	Verify application processes and all high-value business rule flows in a trusted environment such as a protected and monitored server.
12-2	Verify that the application does not allow spoofed high-value transactions, such as allowing an attacking user (A) to process a transaction as the victim user (B) by manipulating, re-setting the session, transaction state, user identity, or transaction.
12-3	Verify that the application does not allow the manipulation of high-value business rules standards which include, but are not limited to, price, interest, discounts, and identifiable information (PII), stocks, stock identities, and others.
12-4	Checking for defensive measures in the application to protect against denial attacks. These include protected and verifiable transaction logs, audit logs or system logs, and in higher value systems, direct monitoring of user and transaction activities for any abnormal activities.
12-5	Verify that the application provides protection against disclosure attacks such as direct object referrals, tampering, use of speculative attacks to hack session, and other types of attacks.
12-6	Verify that there are adequate detection and tuning controls in place in the application to protect against speculative attacks (such as continuous use of a function) or denial of service attacks.
12-7	Verify that there are adequate access controls in the application to prevent feature and privilege upgrade attacks. These controls include preventing anonymous users from accessing protected data or protected functions, or preventing users from accessing other users' information, or using functions with important and sensitive features and powers.
12-8	Verify that the application processes business rules batches in successive steps only, so that all steps are processed directly, avoiding erratic processing or skipping, or processing other user steps or submitted transactions quickly.
12-9	Verify that the application includes additional permissions (such as setup verification or variable identity verification) for low value systems and/or separate tasks for high value applications to enforce anti-fraud controls according to the application's risks and past fraud.
12-10	Verify that the application has operating limits that it applies in a trusted location (such as on a protected server) to each user or on a daily basis, which include configurable alerts and automatic responses to automated or asymmetric attacks.
13	**Using Components with Known Vulnerabilities (OWASP: A9:2017)**
13-1	Verify that forwarding and sending in the address bar (URL) does not include unauthorized data.
13-2	Verify the uniformity of filenames and path data obtained from untrusted sources to negate path-bypass attacks.
13-3	Verify that files obtained from untrusted sources are scanned by antivirus programs to prevent the download of known malicious software.

13-4	Verify that standards obtained from untrusted sources are not used to manipulate file names, path names, or system files and objects without first standardizing them and verify their input to prevent local file insertion attacks.
13-5	Standardization checks obtained from untrusted sources, checks their inputs, and encodes their outputs to prevent remote file insertion attacks, especially when it is possible to implement inputs such as addresses, sources, or insert templates.
13-6	Check that remote spam is not allowed when sharing resources” IFRAMEs" and "HTML 5" across domains.
13-7	Check that files obtained from untrusted sources are stored outside "Webroot".
13-8	Verify that the web server or application is set up automatically to block access to remote sources or systems outside the web server or application.
13-9	Verify that application code does not implement uploaded data obtained from untrusted sources.
13-10	Check settings for app resource sharingFlash, Silverlight, or other cross-domain Rich Internet Application (RIA) that prevents unauthorized access or unauthorized remote access.
13-11	Verify that all types of files allowed to be uploaded are limited to work purposes and as needed (such as filesPDF and Office software documents.
13-12	Ensure that file type verification is done by checking file titles and not just the file extension name.
13-13	Verify that execution privileges are not enabled in file upload directories.
13-14	Verify that the application's file and resource settings are automatically set to read-only mode.
13-15	Check to cancel all types of posts and unnecessary administrative posts, and restrict access to posts or make it require identity verification.
13-16	Request identity verification before uploading files.
13-17	Limit the size of files that can be uploaded to the size required for business purposes (eg 1 MB maximum), and add a note on the web page regarding acceptable file sizes.
14	**Mobile Verification**
14-1	Ensure that the customer verifies the SSL encryption certificates (SSL).
14-2	Verify that unique device identification number values are not being used (UDID) as security controls.
14-3	Verify that the mobile application does not store protected information on shared resources on the device (such as a card "SD" or shared folders).
14-4	Verify that protected information is not stored in a database” SQLite” on the device.
14-5	Verify that secret keys and passwords are not embedded in the code in executable programs.
14-6	Verify that the mobile application prevents the leakage of protected information by the automatic imaging feature of the operating system “iOS”.
14-7	Verify that the app cannot run on a device that has its restrictions removed (Jailbroken) or a device with important and sensitive (Rooted) powers and features.

14-8	Verify that the session expiry time has a Boolean value.
14-9	Verify which permits are being requested and which sources are being granted access (AndroidManifest.xml, and iOS Entitlements).
14-10	Verify that system crash logs do not include protected information.
14-11	Verify that the binary system is not visible in the application.
14-12	Verify that all test data has been removed from the application container (.apk. bar .ipa).
14-13	Verify that the application does not log protected information to the system registry or system files.
14-14	Verify that the app does not enable autocomplete for sensitive texts in input fields such as passwords, personal information or credit card fields.
14-15	Verify that the mobile application applies the certificate installation process (Certificate Pinning) to prevent the proxy application's traffic management.
14-16	Checking that there are incorrect settings in the configuration files (set corrective tags, global readable and writable permissions).
14-17	Verify that third-party libraries in use are up to date and have no known vulnerabilities.
14-18	Verify that web data such as HTTPS traffic is not stored (HTTPS).
14-19	Verify that the character string is not used for the query (Query String) with protected information. Instead, a "POST" request should be used over SSL with a cross-site request forgery (CSRF) identification token.
14-20	Verify, if possible, that personal account numbers are broken before storing them on the device.
14-21	Verify that the application is taking advantage of address space map randomization (ASLR).
14-22	Verify that the recorded data is via the keyboard (iOS) does not include credentials, financial information, or other protected information.
14-23	In Android applications, check that the application does not create files with permissions MODE_WORLD_READABLE or MODE_WORLD_WRITABLE.
14-24	Verify that protected information is stored encrypted and securely (even when stored in a keychain" iOS").
14-25	Verify the application of anti-debugging and reverse engineering mechanisms in the application.
14-26	Verify that the app does not import sensitive activities, content providers, or others on Android.
14-27	Verify that variable structs are used for random strings Sensitive Strings such as account numbers, overwritten when not in use (to reduce damage from memory parsing attacks).
14-28	Ensure that full data validation is implemented on the inputs for any Vulnerable Activity Messages, Content Providers and Broadcast Receivers (Android).
15	**Security Misconfiguration (OWASP: A6:2017)**
15-1	Verify the use of justified queries to prevent SQL injection (SQL Injection).
15-2	Verify that complex and secure credentials are used to access databases.

15-3	Verify that the application accessing the databases has the lowest possible level of required privileges and permissions.
15-4	Verify that strings of random characters (Strings) for the connection are not embedded in the code within the application, especially the validation credentials from the database.
15-5	Verify that the connection to the database is closed as soon as possible.
15-6	Verify the deletion or deactivation of all unnecessary or unused database functions, including automatic resource content, and the installation of minimum features and options necessary for the application to function. For example, deactivating stored procedures or services and bundles of unnecessary useful properties.
15-7	Verify the deactivation of any automatic or unnecessary accounts through which databases not needed to support business requirements can be accessed.
15-8	Verify that the application uses different credentials for each feature and authority (eg user, read-only user, guest, and admins) when it connects to the database.
15-9	Verify that remote login and anonymous sessions are disabled if they are not needed.
15-10	For database-based applications, standardized setup and fortification templates must be used, and all systems tested as part of critical business processes must be used.