openIMIS – Security Recommendations

8/15/2022

SecurityONE

Document details:

Title	Current Security Recommendations Review
Project Resources	**Alexandru MOTOC**
Project Timeline	**08/04/2022 – 08/15/2022**

Document History

Version	Date	Author	Comments
1.0	08/12/2022	Alexandru Motoc	Creation
1.1	08/15/2022	Alexandru Motoc	Review

Contents

Purpose and scope

The purpose of this document is to secure the openIMIS web application.

The objectives of this document are to ensure that:

- the implementation or modification of web applications does not lead to the introduction of insecure code which may compromise the confidentiality or integrity of information assets
- baseline web application security controls are implemented to safeguard against unauthorised modification of web content
- software development processes incorporate adequate security so as to prevent adverse impact to company information technology infrastructure
- a whole approach for developing secure web applications is established.

The scope of this review is to make a list of security requirements for openIMIS web application.

This list has been done having best practices in mind, as defined by Open Web Application Security Project (OWASP).

The security requirements recommendations are based on OWASP Application Security Verification Standard 4.0.3. OWASP ASVS 4.0.3 is released under the Creative Commons Attribution ShareAlike 3.0 license.

OWASP is an open community dedicated to enabling organizations to develop, purchase, and maintain applications that can be trusted. All of the OWASP tools, documents, forums, and chapters are free and open to anyone interested in improving application security.

This document contains the most important security requirements for the openIMIS web application, and we recommend that all requirements classified as Required to be implemented as much as possible by the openIMIS's development team.

Conventions

Term	Description
Required	Means that the definition is an absolute requirement.
Recommended	Means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

Security Requirements

1 Architecture, Design and Threat Modelling

Architecture is not an implementation, but a way of thinking about a problem that has potentially many different answers, and no one single "correct" answer.
A specific implementation of a web application is likely to be revised continuously throughout its lifetime, but the overall architecture will likely rarely change but evolve slowly.
In this chapter, the security requirements cover off the primary aspects of any sound security architecture: availability, confidentiality, processing integrity, non-repudiation, and privacy. Each of these security principles must be built in and be innate to all applications. It is critical to "shift left", starting with developer enablement with secure coding checklists, mentoring and training, coding and testing, building, deployment, configuration, and operations, and finishing with follow up independent testing to assure that all of the security controls are present and functional.

Subcategory	#	Requirement	Assessment
1.1 Secure Software Development Lifecycle	**1.1.1**	Use a secure software development lifecycle that addresses security in all stages of development.	Required
	1.1.2	Use a threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.	Required
	1.1.3	All user stories and features contain functional security constraints, such as "As a user, I should be able to view and edit my profile. I should not be able to view or edit anyone else's profile"	Required
	1.1.4	Documentation and justification of all the application's trust boundaries, components, and significant data flows.	Required
	1.1.5	Definition and security analysis of the application's high-level architecture and all connected remote services.	Required
	1.1.6	Implementation of centralized, simple (economy of design), vetted, secure, and reusable security controls to avoid duplicate, missing, ineffective, or insecure controls.	Required
	1.1.7	The availability of a secure coding checklist, security requirements, guideline, or policy to all developers and testers.	Required

Subcategory	#	Requirement	Assessment
1.2 Authentication Architecture	**1.2.1**	Use of unique or special low-privilege operating system accounts for all application components, services, and servers.	Required
	1.2.2	The communications between application components, including APIs, middleware and data layers, are authenticated. Components should have the least necessary privileges needed.	Required
	1.2.3	The application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication, and has sufficient logging and monitoring to detect account abuse or breaches.	Required
	1.2.4	All authentication pathways and identity management APIs implement consistent authentication security control strength, such that there are no weaker alternatives per the risk of the application.	Required
1.3 Access Control Architecture	**1.3.1**	Trusted enforcement points, such as access control gateways, servers, and serverless functions, enforce access controls. Never enforce access controls on the client.	Required
	1.3.2	The application uses a single and well-vetted access control mechanism for accessing protected data and resources. All requests must pass through this single mechanism to avoid copy and paste or insecure alternative paths.	Required
	1.3.3	The attribute or feature-based access control is used whereby the code checks the user's authorization for a feature/data item rather than just their role. Permissions should still be allocated using roles.	Required
1.4 Input and Output Architecture	**1.4.1**	The input and output requirements clearly define how to handle and process data based on type, content, and applicable laws, regulations, and other policy compliance.	Required
	1.4.2	The serialization is not used when communicating with untrusted clients. If this is not possible, ensure that adequate integrity controls (and possibly encryption if sensitive data is sent) are enforced to prevent deserialization attacks including object injection.	Required
	1.4.3	The input validation is enforced on a trusted service layer.	Required
	1.4.4	The output encoding occurs close to or by the interpreter for which it is intended.	Required
1.5 Cryptographic Architecture	**1.5.1**	There is an explicit policy for management of cryptographic keys and that a cryptographic key lifecycle follows a key management standard.	Required
	1.5.2	The consumers of cryptographic services protect key material and other secrets by using key vaults or API based alternatives.	Required

Subcategory	#	Requirement	Assessment
	1.5.3	All keys and passwords are replaceable and are part of a well-defined process to re-encrypt sensitive data.	Required
	1.5.4	The architecture treats client-side secrets--such as symmetric keys, passwords, or API tokens--as insecure and never uses them to protect or access sensitive data.	Required
1.6 Errors, Logging and Auditing Architecture	**1.6.1**	A common logging format and approach is used across the system.	Required
	1.6.2	The logs are securely transmitted to a preferably remote system for analysis, detection, alerting, and escalation.	Required
1.7 Data Protection and Privacy Architecture	**1.7.1**	All sensitive data is identified and classified into protection levels.	Required
	1.7.2	All protection levels have an associated set of protection requirements, such as encryption requirements, integrity requirements, retention, privacy and other confidentiality requirements, and that these are applied in the architecture.	Required
1.8 Communications Architecture	**1.8.1**	The application encrypts communications between components, particularly when these components are in different containers, systems, sites, or cloud providers.	Required
	1.8.2	The application components verify the authenticity of each side in a communication link to prevent person-in-the-middle attacks. For example, application components should validate TLS certificates and chains.	Required
1.9 Malicious Software Architecture	**1.9.1**	A source code control system is in use, with procedures to ensure that check-ins are accompanied by issues or change tickets. The source code control system should have access control and identifiable users to allow traceability of any changes.	Required
1.10 Business Logic Architecture	**1.10.1**	The definition and documentation of all application components in terms of the business or security functions they provide.	Required
	1.10.2	All high-value business logic flows, including authentication, session management and access control, do not share unsynchronized state.	Required
	1.10.3	All high-value business logic flows, including authentication, session management and access control are thread safe and resistant to time-of-check and time-of-use race conditions.	Recommended

Subcategory	#	Requirement	Assessment
1.11 Secure File Upload Architecture	**1.11.1**	The user-uploaded files - if required to be displayed or downloaded from the application - are served by either octet stream downloads, or from an unrelated domain, such as a cloud file storage bucket. Implement a suitable Content Security Policy (CSP) to reduce the risk from XSS vectors or other attacks from the uploaded file.	Required
1.12 Configuration Architecture	**1.12.1**	The segregation of components of differing trust levels through well-defined security controls, firewall rules, API gateways, reverse proxies, cloud-based security groups, or similar mechanisms.	Required
	1.12.2	The binary signatures, trusted connections, and verified endpoints are used to deploy binaries to remote devices.	Required
	1.12.3	The build pipeline warns of out-of-date or insecure components and takes appropriate actions.	Required
	1.12.4	The build pipeline contains a build step to automatically build and verify the secure deployment of the application, particularly if the application infrastructure is software defined, such as cloud environment build scripts.	Required
	1.12.5	The application deployments adequately sandbox, containerize and/or isolate at the network level to delay and deter attackers from attacking other applications, especially when they are performing sensitive or dangerous actions such as deserialization.	Required
	1.12.6	The application does not use unsupported, insecure, or deprecated client-side technologies such as NSAPI plugins, Flash, Shockwave, ActiveX, Silverlight, NACL, or client-side Java applets.	Required

2 Authentication

Authentication is the act of establishing, or confirming, someone (or something) as authentic and that claims made by a person or about a device are correct, resistant to impersonation, and prevent recovery or interception of passwords.

Subcategory	#	Requirement	Assessment
2.1 Password Security	**2.1.1**	The user set passwords are at least 12 characters in length (after multiple spaces are combined).	Required
	2.1.10	There are no periodic credential rotation or password history requirements.	Required
	2.1.11	The "paste" functionality, browser password helpers, and external password managers are permitted.	Required
	2.1.12	The user can choose to either temporarily view the entire masked password, or temporarily view the last typed character of the password on platforms that do not have this as built-in functionality.	Required
	2.1.2	The passwords of at least 64 characters are permitted, and that passwords of more than 128 characters are denied.	Required
	2.1.3	The password truncation is not performed. However, consecutive multiple spaces may be replaced by a single space.	Required
	2.1.4	Any printable Unicode character, including language neutral characters such as spaces and Emojis are permitted in passwords.	Required
	2.1.5	The users can change their password.	Required
	2.1.6	The password change functionality requires the user's current and new password.	Required
	2.1.7	The passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally (such as the top 1,000 or 10,000 most common passwords which match the system's password policy) or using an external API. If using an API a zero knowledge proof or other mechanism should be used to ensure that the plain text password is not sent or used in verifying the breach status of the password. If the password is breached, the application must require the user to set a new non-breached password.	Required
	2.1.8	A password strength meter is provided to help users set a stronger password.	Required
	2.1.9	There are no password composition rules limiting the type of characters permitted. There should be no requirement for upper or lower case or numbers or special characters.	Required

Subcategory	#	Requirement	Assessment
2.2 General Authenticator Security	**2.2.1**	The anti-automation controls are effective at mitigating breached credential testing, brute force, and account lockout attacks. Such controls include blocking the most common breached passwords, soft lockouts, rate limiting, CAPTCHA, ever increasing delays between attempts, IP address restrictions, or risk-based restrictions such as location, first login on a device, recent attempts to unlock the account, or similar. Verify that no more than 10 failed attempts per hour is possible on a single account.	Required
	2.2.2	The use of weak authenticators (such as SMS and email) is limited to secondary verification and transaction approval and not as a replacement for more secure authentication methods. Verify that stronger methods are offered before weak methods, users are aware of the risks, or that proper measures are in place to limit the risks of account compromise.	Required
	2.2.3	Secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, logging in from unknown or risky locations. The use of push notifications - rather than SMS or email - is preferred, but in the absence of push notifications, SMS or email is acceptable as long as no sensitive information is disclosed in the notification.	Required
	2.2.4	The impersonation resistance against phishing, such as the use of multi-factor authentication, cryptographic devices with intent (such as connected keys with a push to authenticate), or at higher AAL levels, client-side certificates.	Recommended
	2.2.5	Where a Credential Service Provider (CSP) and the application verifying authentication are separated, mutually authenticated TLS is in place between the two endpoints.	Recommended
	2.2.6	The replay resistance through the mandated use of One-time Passwords (OTP) devices, cryptographic authenticators, or lookup codes.	Recommended
	2.2.7	The intent to authenticate by requiring the entry of an OTP token or user-initiated action such as a button press on a FIDO hardware key.	Recommended
2.3 Authenticator Lifecycle	**2.3.1**	The system generated initial passwords or activation codes SHOULD be securely randomly generated, SHOULD be at least 6 characters long, and MAY contain letters and numbers, and expire after a short period of time. These initial secrets must not be permitted to become the long term password.	Required
	2.3.2	The enrollment and use of user-provided authentication devices are supported, such as a U2F or FIDO tokens.	Required
	2.3.3	The renewal instructions are sent with sufficient time to renew time bound authenticators.	Required

Subcategory	#	Requirement	Assessment
2.4 Credential Storage	**2.4.1**	The passwords are stored in a form that is resistant to offline attacks. Passwords SHALL be salted and hashed using an approved one-way key derivation or password hashing function. Key derivation and password hashing functions take a password, a salt, and a cost factor as inputs when generating a password hash.	Required
	2.4.2	Verify that the salt is at least 32 bits in length and be chosen arbitrarily to minimize salt value collisions among stored hashes. For each credential, a unique salt value and the resulting hash SHALL be stored. (C6)	Required
	2.4.3	If PBKDF2 is used, the iteration count SHOULD be as large as verification server performance will allow, typically at least 100,000 iterations.	Required
	2.4.4	If bcrypt is used, the work factor SHOULD be as large as verification server performance will allow, with a minimum of 10.	Required
	2.4.5	An additional iteration of a key derivation function is performed, using a salt value that is secret and known only to the verifier. Generate the salt value using an approved random bit generator and provide at least the minimum security strength. The secret salt value SHALL be stored separately from the hashed passwords (e.g., in a specialized device like a hardware security module).	Required
2.5 Credential Recovery	**2.5.1**	A system generated initial activation or recovery secret is not sent in clear text to the user.	Required
	2.5.2	The password hints or knowledge-based authentication (so-called "secret questions") are not present.	Required
	2.5.3	The password credential recovery does not reveal the current password in any way.	Required
	2.5.4	Shared or default accounts are not present (e.g. "root", "admin", or "sa").	Required
	2.5.5	If an authentication factor is changed or replaced, that the user is notified of this event.	Required
	2.5.6	The forgotten password, and other recovery paths use a secure recovery mechanism, such as time-based OTP (TOTP) or other soft token, mobile push, or another offline recovery mechanism.	Required
	2.5.7	If OTP or multi-factor authentication factors are lost, that evidence of identity proofing is performed at the same level as during enrollment.	Required
2.6 Look-up Secret Verifier	**2.6.1**	The lookup secrets can be used only once.	Required
	2.6.2	The lookup secrets have sufficient randomness (112 bits of entropy), or if less than 112 bits of entropy, salted with a unique and random 32-bit salt and hashed with an approved one-way hash.	Required
	2.6.3	The lookup secrets are resistant to offline attacks, such as predictable values.	Required

Subcategory	#	Requirement	Assessment
2.7 Out of Band Verifier	**2.7.1**	The clear text out of band authenticators, such as SMS or PSTN, are not offered by default, and stronger alternatives such as push notifications are offered first.	Required
	2.7.2	The out of band verifier expires out of band authentication requests, codes, or tokens after 10 minutes.	Required
	2.7.3	The out of band verifier authentication requests, codes, or tokens are only usable once, and only for the original authentication request.	Required
	2.7.4	The out of band authenticator and verifier communicates over a secure independent channel.	Required
	2.7.5	The out of band verifier retains only a hashed version of the authentication code.	Required
	2.7.6	The initial authentication code is generated by a secure random number generator, containing at least 20 bits of entropy (typically a six digital random number is sufficient).	Required
2.8 One Time Verifier	**2.8.1**	The time-based OTPs have a defined lifetime before expiring.	Required
	2.8.2	The symmetric keys used to verify submitted OTPs are highly protected, such as by using a hardware security module or secure operating system based key storage.	Required
	2.8.3	The approved cryptographic algorithms are used in the generation, seeding, and verification of OTPs.	Required
	2.8.4	The time-based OTP can be used only once within the validity period.	Required
	2.8.5	If a time-based multi-factor OTP token is re-used during the validity period, it is logged and rejected with secure notifications being sent to the holder of the device.	Required
	2.8.6	The physical single-factor OTP generator can be revoked in case of theft or other loss. Ensure that revocation is immediately effective across logged in sessions, regardless of location.	Required
	2.8.7	The biometric authenticators are limited to use only as secondary factors in conjunction with either something you have and something you know.	Recommended
2.9 Cryptographic Verifier	**2.9.1**	The cryptographic keys used in verification are stored securely and protected against disclosure, such as using a Trusted Platform Module (TPM) or Hardware Security Module (HSM), or an OS service that can use this secure storage.	Required
	2.9.2	The challenge nonce is at least 64 bits in length, and statistically unique or unique over the lifetime of the cryptographic device.	Required
	2.9.3	The approved cryptographic algorithms are used in the generation, seeding, and verification.	Required

Subcategory	#	Requirement	Assessment
2.10 Service Authentication	**2.10.1**	The intra-service secrets do not rely on unchanging credentials such as passwords, API keys or shared accounts with privileged access. [OS assisted]	Required
	2.10.2	If passwords are required for service authentication, the service account used is not a default credential. (e.g. root/root or admin/admin are default in some services during installation). [OS assisted]	Required
	2.10.3	The passwords are stored with sufficient protection to prevent offline recovery attacks, including local system access. [OS assisted]	Required
	2.10.4	The passwords, integrations with databases and third-party systems, seeds and internal secrets, and API keys are managed securely and not included in the source code or stored within source code repositories. Such storage SHOULD resist offline attacks. The use of a secure software key store, hardware TPM, or an HSM is recommended for password storage. [OS assisted]	Required

3 Session Management

One of the core components of any web-based application or stateful API is the mechanism by which it controls and maintains the state for a user or device interacting with it. Session management changes a stateless protocol to stateful, which is critical for differentiating different users or devices. Ensure that a verified application satisfies the following high-level session management requirements:

- Sessions are unique to each individual and cannot be guessed or shared.
- Sessions are invalidated when no longer required and timed out during periods of inactivity.

Subcategory	#	Requirement	Assessment
3.1 Fundamental Session Management Security	**3.1.1**	The application never reveals session tokens in URL parameters.	Required
3.2 Session Binding	**3.2.1**	The application generates a new session token on user authentication.	Required
	3.2.2	The session tokens possess at least 64 bits of entropy.	Required
	3.2.3	The application only stores session tokens in the browser using secure methods such as appropriately secured cookies or HTML 5 session storage.	Required
	3.2.4	The session tokens are generated using approved cryptographic algorithms.	Required
3.3 Session Termination	**3.3.1**	The logout and expiration invalidate the session token, such that the back button or a downstream relying party does not resume an authenticated session, including across relying parties.	Required
	3.3.2	If authenticators permit users to remain logged in, re-authentication occurs periodically both when actively used or after an idle period. [12 hours or 15 minutes of inactivity, with 2FA optional]	Required
	3.3.3	The application gives the option to terminate all other active sessions after a successful password change (including change via password reset/recovery), and that this is effective across the application, federated login (if present), and any relying parties.	Required
	3.3.4	The users are able to view and (having re-entered login credentials) log out of any or all currently active sessions and devices.	Required
3.4 Cookie-based Session Management	**3.4.1**	The cookie-based session tokens have the 'Secure' attribute set.	Required
	3.4.2	The cookie-based session tokens have the 'HttpOnly' attribute set.	Required

Subcategory	#	Requirement	Assessment
	3.4.3	The cookie-based session tokens utilize the 'SameSite' attribute to limit exposure to cross-site request forgery attacks.	Required
	3.4.4	The cookie-based session tokens use the "__Host-" prefix so cookies are only sent to the host that initially set the cookie.	Required
	3.4.5	If the application is published under a domain name with other applications that set or use session cookies that might disclose the session cookies, set the path attribute in cookie-based session tokens using the most precise path possible.	Required
3.5 Token-based Session Management	**3.5.1**	The application allows users to revoke OAuth tokens that form trust relationships with linked applications.	Required
	3.5.2	The application uses session tokens rather than static API secrets and keys, except with legacy implementations.	Required
	3.5.3	The stateless session tokens use digital signatures, encryption, and other countermeasures to protect against tampering, enveloping, replay, null cipher, and key substitution attacks.	Required
3.6 Federated Re-authentication	**3.6.1**	The Relying Parties (RPs) specify the maximum authentication time to Credential Service Providers (CSPs) and that CSPs re-authenticate the user if they haven't used a session within that period.	Recommended
	3.6.2	The Credential Service Providers (CSPs) inform Relying Parties (RPs) of the last authentication event, to allow RPs to determine if they need to re-authenticate the user.	Recommended
3.7 Defenses Against Session Management Exploits	**3.7.1**	The application ensures a full, valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.	Required

4 Access Control

Authorization is the concept of allowing access to resources only to those permitted to use them. Ensure that a verified application satisfies the following high level requirements:

- Persons accessing resources hold valid credentials to do so.
- Users are associated with a well-defined set of roles and privileges.
- Role and permission metadata is protected from replay or tampering.

Subcategory	#	Requirement	Assessment
4.1 General Access Control Design	**4.1.1**	The application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.	Required
	4.1.2	All user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.	Required
	4.1.3	The principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege.	Required
	4.1.4	The access controls fail securely including when an exception occurs.	Required
4.2 Operation Level Access Control	**4.2.1**	The sensitive data and APIs are protected against Insecure Direct Object Reference (IDOR) attacks targeting creation, reading, updating and deletion of records, such as creating or updating someone else's record, viewing everyone's records, or deleting all records.	Required
	4.2.2	The application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.	Required
4.3 Other Access Control Considerations	**4.3.1**	The administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.	Required
	4.3.2	The directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.	Required

Subcategory	#	Requirement	Assessment
	4.3.3	The application has additional authorization (such as step up or adaptive authentication) for lower value systems, and / or segregation of duties for high value applications to enforce anti-fraud controls as per the risk of application and past fraud.	Required

5 Validation, Sanitization and Encoding

The most common web application security weakness is the failure to properly validate input coming from the client or the environment before directly using it without any output encoding. This weakness leads to almost all of the significant vulnerabilities in web applications, such as Cross-Site Scripting (XSS), SQL injection, interpreter injection, locale/Unicode attacks, file system attacks, and buffer overflows.
Ensure that a verified application satisfies the following high-level requirements:

- Input validation and output encoding architecture have an agreed pipeline to prevent injection attacks.
- Input data is strongly typed, validated, range or length checked, or at worst, sanitized or filtered.
- Output data is encoded or escaped as per the context of the data as close to the interpreter as possible.

Subcategory	#	Requirement	Assessment
5.1 Input Validation	**5.1.1**	The application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, or environment variables).	Required
	5.1.2	The frameworks protect against mass parameter assignment attacks, or that the application has countermeasures to protect against unsafe parameter assignment, such as marking fields private or similar.	Required
	5.1.3	All input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (allow lists).	Required
	5.1.4	The structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers, e-mail addresses, telephone numbers, or validating that two related fields are reasonable, such as checking that suburb and zip/postcode match).	Required

Subcategory	#	Requirement	Assessment
	5.1.5	The URL redirects and forwards only allow destinations which appear on an allow list, or show a warning when redirecting to potentially untrusted content.	Required
5.2 Sanitization and Sandboxing	**5.2.1**	All untrusted HTML input from WYSIWYG editors or similar is properly sanitized with an HTML sanitizer library or framework feature.	Required
	5.2.2	The unstructured data is sanitized to enforce safety measures such as allowed characters and length.	Required
	5.2.3	The application sanitizes user input before passing to mail systems to protect against SMTP or IMAP injection.	Required
	5.2.4	The application avoids the use of eval() or other dynamic code execution features. Where there is no alternative, any user input being included must be sanitized or sandboxed before being executed.	Required
	5.2.5	The application protects against template injection attacks by ensuring that any user input being included is sanitized or sandboxed.	Required
	5.2.6	The application protects against SSRF attacks, by validating or sanitizing untrusted data or HTTP file metadata, such as filenames and URL input fields, and uses allow lists of protocols, domains, paths and ports.	Required
	5.2.7	The application sanitizes, disables, or sandboxes user-supplied Scalable Vector Graphics (SVG) scriptable content, especially as they relate to XSS resulting from inline scripts, and foreignObject.	Required
	5.2.8	The application sanitizes, disables, or sandboxes user-supplied scriptable or expression template language content, such as Markdown, CSS or XSL stylesheets, BBCode, or similar.	Required
5.3 Output Encoding and Injection Prevention	**5.3.1**	The output encoding is relevant for the interpreter and context required. For example, use encoders specifically for HTML values, HTML attributes, JavaScript, URL parameters, HTTP headers, SMTP, and others as the context requires, especially from untrusted inputs (e.g. names with Unicode or apostrophes, such as ねこ or O'Hara).	Required
	5.3.10	The application protects against XPath injection or XML injection attacks.	Required
	5.3.2	The output encoding preserves the user's chosen character set and locale, such that any Unicode character point is valid and safely handled.	Required
	5.3.3	The context-aware, preferably automated - or at worst, manual - output escaping protects against reflected, stored, and DOM based XSS.	Required
	5.3.4	The data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from database injection attacks.	Required

Subcategory	#	Requirement	Assessment
	5.3.5	Where parameterized or safer mechanisms are not present, context-specific output encoding is used to protect against injection attacks, such as the use of SQL escaping to protect against SQL injection.	Required
	5.3.6	The application protects against JSON injection attacks, JSON eval attacks, and JavaScript expression evaluation.	Required
	5.3.7	The application protects against LDAP injection vulnerabilities, or that specific security controls to prevent LDAP injection have been implemented.	Required
	5.3.8	The application protects against OS command injection and that operating system calls use parameterized OS queries or use contextual command line output encoding.	Required
	5.3.9	The application protects against Local File Inclusion (LFI) or Remote File Inclusion (RFI) attacks.	Required
5.4 Memory, String, and Unmanaged Code	**5.4.1**	The application uses memory-safe string, safer memory copy and pointer arithmetic to detect or prevent stack, buffer, or heap overflows.	Required
	5.4.2	The format strings do not take potentially hostile input, and are constant.	Required
	5.4.3	The sign, range, and input validation techniques are used to prevent integer overflows.	Required
5.5 Deserialization Prevention	**5.5.1**	The serialized objects use integrity checks or are encrypted to prevent hostile object creation or data tampering.	Required
	5.5.2	The application correctly restricts XML parsers to only use the most restrictive configuration possible and to ensure that unsafe features such as resolving external entities are disabled to prevent XML eXternal Entity (XXE) attacks.	Required
	5.5.3	The deserialization of untrusted data is avoided or is protected in both custom code and third-party libraries (such as JSON, XML and YAML parsers).	Required
	5.5.4	When parsing JSON in browsers or JavaScript-based backends, JSON.parse is used to parse the JSON document. Do not use eval() to parse JSON.	Required

6 Stored Cryptography

Ensure that a verified application satisfies the following high level requirements:

- All cryptographic modules fail in a secure manner and that errors are handled correctly.
- A suitable random number generator is used.
- Access to keys is securely managed.

Subcategory	#	Requirement	Assessment
6.1 Data Classification	**6.1.1**	The regulated private data is stored encrypted while at rest, such as Personally Identifiable Information (PII), sensitive personal information, or data assessed likely to be subject to EU's GDPR.	Required
	6.1.2	The regulated health data is stored encrypted while at rest, such as medical records, medical device details, or de-anonymized research records.	Required
	6.1.3	The regulated financial data is stored encrypted while at rest, such as financial accounts, defaults or credit history, tax records, pay history, beneficiaries, or de-anonymized market or research records.	Required
6.2 Algorithms	**6.2.1**	All cryptographic modules fail securely, and errors are handled in a way that does not enable Padding Oracle attacks.	Required
	6.2.2	The industry proven or government approved cryptographic algorithms, modes, and libraries are used, instead of custom coded cryptography.	Required
	6.2.3	The encryption initialization vector, cipher configuration, and block modes are configured securely using the latest advice.	Required
	6.2.4	The random number, encryption or hashing algorithms, key lengths, rounds, ciphers or modes, can be reconfigured, upgraded, or swapped at any time, to protect against cryptographic breaks.	Required
	6.2.5	The known insecure block modes (i.e. ECB, etc.), padding modes (i.e. PKCS#1 v1.5, etc.), ciphers with small block sizes (i.e. Triple-DES, Blowfish, etc.), and weak hashing algorithms (i.e. MD5, SHA1, etc.) are not used unless required for backwards compatibility.	Required
	6.2.6	The nonces, initialization vectors, and other single use numbers must not be used more than once with a given encryption key. The method of generation must be appropriate for the algorithm being used.	Required

Subcategory	#	Requirement	Assessment
	6.2.7	The encrypted data is authenticated via signatures, authenticated cipher modes, or HMAC to ensure that ciphertext is not altered by an unauthorized party.	Recommended
	6.2.8	All cryptographic operations are constant-time, with no 'short-circuit' operations in comparisons, calculations, or returns, to avoid leaking information.	Recommended
6.3 Random Values	**6.3.1**	All random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module's approved cryptographically secure random number generator when these random values are intended to be not guessable by an attacker.	Required
	6.3.2	The random GUIDs are created using the GUID v4 algorithm, and a Cryptographically-secure Pseudo-random Number Generator (CSPRNG). GUIDs created using other pseudo-random number generators may be predictable.	Required
	6.3.3	The random numbers are created with proper entropy even when the application is under heavy load, or that the application degrades gracefully in such circumstances.	Recommended
6.4 Secret Management	**6.4.1**	A secrets management solution such as a key vault is used to securely create, store, control access to and destroy secrets.	Required
	6.4.2	The key material is not exposed to the application but instead uses an isolated security module like a vault for cryptographic operations.	Required

7 Error Handling and Logging

The primary objective of error handling and logging is to provide useful information for the user, administrators, and incident response teams. The objective is not to create massive amounts of logs, but high quality logs, with more signal than discarded noise.
High quality logs will often contain sensitive data and must be protected as per local data privacy laws or directives. This should include:

- Not collecting or logging sensitive information unless specifically required.
- Ensuring all logged information is handled securely and protected as per its data classification.
- Ensuring that logs are not stored forever, but have an absolute lifetime that is as short as possible.

Subcategory	#	Requirement	Assessment
7.1 Log Content	**7.1.1**	The application does not log credentials or payment details. Session tokens should only be stored in logs in an irreversible, hashed form.	Required
	7.1.2	The application does not log other sensitive data as defined under local privacy laws or relevant security policy.	Required
	7.1.3	The application logs security relevant events including successful and failed authentication events, access control failures, deserialization failures and input validation failures.	Required
	7.1.4	Each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens.	Required
7.2 Log Processing	**7.2.1**	All authentication decisions are logged, without storing sensitive session tokens or passwords. This should include requests with relevant metadata needed for security investigations.	Required
	7.2.2	All access control decisions can be logged and all failed decisions are logged. This should include requests with relevant metadata needed for security investigations.	Required
7.3 Log Protection	**7.3.1**	All logging components appropriately encode data to prevent log injection.	Required
	7.3.2	The security logs are protected from unauthorized access and modification.	Required
	7.3.3	The time sources are synchronized to the correct time and time zone. Strongly consider logging only in UTC if systems are global to assist with post-incident forensic analysis.	Required
7.4 Error Handling	**7.4.1**	A generic message is shown when an unexpected or security sensitive error occurs, potentially with a unique ID which support personnel can use to investigate.	Required
	7.4.2	Exception handling (or a functional equivalent) is used across the codebase to account for expected and unexpected error conditions.	Required
	7.4.3	A "last resort" error handler is defined which will catch all unhandled exceptions.	Required

8 Data Protection

There are three key elements to sound data protection: Confidentiality, Integrity and Availability (CIA). This standard assumes that data protection is enforced on a trusted system, such as a server, which has been hardened and has sufficient protections.
Ensure that a verified application satisfies the following high level data protection requirements:

- Confidentiality: Data should be protected from unauthorized observation or disclosure both in transit and when stored.
- Integrity: Data should be protected from being maliciously created, altered or deleted by unauthorized attackers.
- Availability: Data should be available to authorized users as required.

Subcategory	#	Requirement	Assessment
8.1 General Data Protection	**8.1.1**	The application protects sensitive data from being cached in server components such as load balancers and application caches.	Required
	8.1.2	All cached or temporary copies of sensitive data stored on the server are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data.	Required
	8.1.3	The application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values.	Required
	8.1.4	The application can detect and alert on abnormal numbers of requests, such as by IP, user, total per hour or day, or whatever makes sense for the application.	Required
	8.1.5	Regular backups of important data are performed and that test restoration of data is performed.	Recommended
	8.1.6	The backups are stored securely to prevent data from being stolen or corrupted.	Recommended
8.2 Client-side Data Protection	**8.2.1**	The application sets sufficient anti-caching headers so that sensitive data is not cached in modern browsers.	Required
	8.2.2	The data stored in browser storage (such as localStorage, sessionStorage, IndexedDB, or cookies) does not contain sensitive data.	Required
	8.2.3	The authenticated data is cleared from client storage, such as the browser DOM, after the client or session is terminated.	Required
8.3 Sensitive Private Data	**8.3.1**	The sensitive data is sent to the server in the HTTP message body or headers, and that query string parameters from any HTTP verb do not contain sensitive data.	Required
	8.3.2	The users have a method to remove or export their data on demand.	Required

Subcategory	#	Requirement	Assessment
	8.3.3	The users are provided clear language regarding collection and use of supplied personal information and that users have provided opt-in consent for the use of that data before it is used in any way.	Required
	8.3.4	All sensitive data created and processed by the application has been identified, and ensure that a policy is in place on how to deal with sensitive data.	Required
	8.3.5	Accessing sensitive data is audited (without logging the sensitive data itself), if the data is collected under relevant data protection directives or where logging of access is required.	Required
	8.3.6	The sensitive information contained in memory is overwritten as soon as it is no longer required to mitigate memory dumping attacks, using zeroes or random data.	Required
	8.3.7	The sensitive or private information that is required to be encrypted, is encrypted using approved algorithms that provide both confidentiality and integrity.	Required
	8.3.8	The sensitive personal information is subject to data retention classification, such that old or out of date data is deleted automatically, on a schedule, or as the situation requires.	Required

9 Communication

Ensure that a verified application meets the following high level requirements:

- Require TLS or strong encryption, independent of sensitivity of the content.
- Follow the latest guidance, including:
    - Configuration advice
    - Preferred algorithms and ciphers
- Avoid weak or soon to be deprecated algorithms and ciphers, except as a last resort
- Disable deprecated or known insecure algorithms and ciphers.

Subcategory	#	Requirement	Assessment
9.1 Client Communication Security	**9.1.1**	The TLS is used for all client connectivity, and does not fall back to insecure or unencrypted communications.	Required
	9.1.2	Use up to date TLS testing tools that only strong cipher suites are enabled, with the strongest cipher suites set as preferred.	Required
	9.1.3	Only the latest recommended versions of the TLS protocol are enabled, such as TLS 1.2 and TLS 1.3. The latest version of the TLS protocol should be the preferred option.	Required
9.2 Server Communication Security	**9.2.1**	The connections to and from the server use trusted TLS certificates. Where internally generated or self-signed certificates are used, the server must be configured to only trust specific internal CAs and specific self-signed certificates. All others should be rejected.	Required
	9.2.2	Encrypted communications such as TLS is used for all inbound and outbound connections, including for management ports, monitoring, authentication, API, or web service calls, database, cloud, serverless, mainframe, external, and partner connections. The server must not fall back to insecure or unencrypted protocols.	Required
	9.2.3	All encrypted connections to external systems that involve sensitive information or functions are authenticated.	Required
	9.2.4	Proper certification revocation, such as Online Certificate Status Protocol (OCSP) Stapling, is enabled and configured.	Required
	9.2.5	The backend TLS connection failures are logged.	Recommended

10 Malicious Code

Ensure that code satisfies the following high level requirements:

- Malicious activity is handled securely and properly to not affect the rest of the application.
- Does not have time bombs or other time-based attacks.
- Does not "phone home" to malicious or unauthorized destinations.
- Does not have back doors, Easter eggs, salami attacks, rootkits, or unauthorized code that can be controlled by an attacker.

Subcategory	#	Requirement	Assessment
10.1 Code Integrity	**10.1.1**	A code analysis tool is in use that can detect potentially malicious code, such as time functions, unsafe file operations and network connections.	Recommended
10.2 Malicious Code Search	**10.2.1**	The application source code and third party libraries do not contain unauthorized phone home or data collection capabilities. Where such functionality exists, obtain the user's permission for it to operate before collecting any data.	Required
	10.2.2	The application does not ask for unnecessary or excessive permissions to privacy related features or sensors, such as contacts, cameras, microphones, or location.	Required
	10.2.3	The application source code and third party libraries do not contain back doors, such as hard-coded or additional undocumented accounts or keys, code obfuscation, undocumented binary blobs, rootkits, or anti-debugging, insecure debugging features, or otherwise out of date, insecure, or hidden functionality that could be used maliciously if discovered.	Recommended
	10.2.4	The application source code and third party libraries do not contain time bombs by searching for date and time related functions.	Recommended
	10.2.5	The application source code and third party libraries do not contain malicious code, such as salami attacks, logic bypasses, or logic bombs.	Recommended
	10.2.6	The application source code and third party libraries do not contain Easter eggs or any other potentially unwanted functionality.	Recommended
10.3 Application Integrity	**10.3.1**	If the application has a client or server auto-update feature, updates should be obtained over secure channels and digitally signed. The update code must validate the digital signature of the update before installing or executing the update.	Required
	10.3.2	The application employs integrity protections, such as code signing or subresource integrity. The application must not load or execute code from untrusted sources, such as loading includes, modules, plugins, code, or libraries from untrusted sources or the Internet.	Required

Subcategory	#	Requirement	Assessment
	10.3.3	The application has protection from subdomain takeovers if the application relies upon DNS entries or DNS subdomains, such as expired domain names, out of date DNS pointers or CNAMEs, expired projects at public source code repos, or transient cloud APIs, serverless functions, or storage buckets (autogen-bucket-id.cloud.example.com) or similar. Protections can include ensuring that DNS names used by applications are regularly checked for expiry or change.	Required

11 Business Logic

Ensure that a verified application satisfies the following high level requirements:

- The business logic flow is sequential, processed in order, and cannot be bypassed.
- Business logic includes 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 have considered abuse cases and malicious actors, and have protections against spoofing, tampering, information disclosure, and elevation of privilege attacks.

Subcategory	#	Requirement	Assessment
11.1 Business Logic Security	**11.1.1**	The application will only process business logic flows for the same user in sequential step order and without skipping steps.	Required
	11.1.2	The application will only process business logic flows with all steps being processed in realistic human time, i.e. transactions are not submitted too quickly.	Required
	11.1.3	The application has appropriate limits for specific business actions or transactions which are correctly enforced on a per user basis.	Required
	11.1.4	The application has anti-automation controls to protect against excessive calls such as mass data exfiltration, business logic requests, file uploads or denial of service attacks.	Required
	11.1.5	The application has business logic limits or validation to protect against likely business risks or threats, identified using threat modeling or similar methodologies.	Required
	11.1.6	The application does not suffer from "Time Of Check to Time Of Use" (TOCTOU) issues or other race conditions for sensitive operations.	Required

Subcategory	#	Requirement	Assessment
	11.1.7	The application monitors for unusual events or activity from a business logic perspective. For example, attempts to perform actions out of order or actions which a normal user would never attempt.	Required
	11.1.8	The application has configurable alerting when automated attacks or unusual activity is detected.	Required

12 Files and Resources

Ensure that a verified application satisfies the following high level requirements:

- Untrusted file data should be handled accordingly and in a secure manner.
- Untrusted file data obtained from untrusted sources are stored outside the web root and with limited permissions.

Subcategory	#	Requirement	Assessment
12.1 File Upload	**12.1.1**	The application will not accept large files that could fill up storage or cause a denial of service.	Required
	12.1.2	The application checks compressed files (e.g. zip, gz, docx, odt) against maximum allowed uncompressed size and against maximum number of files before uncompressing the file.	Required
	12.1.3	A file size quota and maximum number of files per user is enforced to ensure that a single user cannot fill up the storage with too many files, or excessively large files.	Required
12.2 File Integrity	**12.2.1**	Files obtained from untrusted sources are validated to be of expected type based on the file's content.	Required
12.3 File Execution	**12.3.1**	The user-submitted filename metadata is not used directly by system or framework filesystems and that a URL API is used to protect against path traversal.	Required
	12.3.2	The user-submitted filename metadata is validated or ignored to prevent the disclosure, creation, updating or removal of local files (LFI).	Required
	12.3.3	The user-submitted filename metadata is validated or ignored to prevent the disclosure or execution of remote files via Remote File Inclusion (RFI) or Server-side Request Forgery (SSRF) attacks.	Required
	12.3.4	The application protects against Reflective File Download (RFD) by validating or ignoring user-submitted filenames in a JSON, JSONP, or URL parameter, the response Content-Type header should be set to text/plain, and the Content-Disposition header should have a fixed filename.	Required

Subcategory	#	Requirement	Assessment
	12.3.5	The untrusted file metadata is not used directly with system API or libraries, to protect against OS command injection.	Required
	12.3.6	The application does not include and execute functionality from untrusted sources, such as unverified content distribution networks, JavaScript libraries, node npm libraries, or server-side DLLs.	Required
12.4 File Storage	**12.4.1**	The files obtained from untrusted sources are stored outside the web root, with limited permissions.	Required
	12.4.2	The files obtained from untrusted sources are scanned by antivirus scanners to prevent upload and serving of known malicious content.	Required
12.5 File Download	**12.5.1**	The web tier is configured to serve only files with specific file extensions to prevent unintentional information and source code leakage. For example, backup files (e.g. .bak), temporary working files (e.g. .swp), compressed files (.zip, .tar.gz, etc) and other extensions commonly used by editors should be blocked unless required.	Required
	12.5.2	The direct requests to uploaded files will never be executed as HTML/JavaScript content.	Required
12.6 SSRF Protection	**12.6.1**	The web or application server is configured with an allow list of resources or systems to which the server can send requests or load data/files from.	Required

13 API and Web Service

Ensure that a verified application that uses trusted service layer APIs (commonly using JSON or XML or GraphQL) has:

- Adequate authentication, session management and authorization of all web services.
- Input validation of all parameters that transit from a lower to higher trust level.
- Effective security controls for all API types, including cloud and Serverless API

Subcategory	#	Requirement	Assessment
13.1 Generic Web Service Security	**13.1.1**	All application components use the same encodings and parsers to avoid parsing attacks that exploit different URI or file parsing behavior that could be used in SSRF and RFI attacks.	Required
	13.1.2	The API URLs do not expose sensitive information, such as the API key, session tokens etc.	Required

Subcategory	#	Requirement	Assessment
	13.1.3	The authorization decisions are made at both the URI, enforced by programmatic or declarative security at the controller or router, and at the resource level, enforced by model-based permissions.	Required
	13.1.4	The requests containing unexpected or missing content types are rejected with appropriate headers (HTTP response status 406 Unacceptable or 415 Unsupported Media Type).	Required
13.2 RESTful Web Service	**13.2.1**	The enabled RESTful HTTP methods are a valid choice for the user or action, such as preventing normal users using DELETE or PUT on protected API or resources.	Required
	13.2.2	The JSON schema validation is in place and verified before accepting input.	Required
	13.2.3	The RESTful web services that utilize cookies are protected from Cross-Site Request Forgery via the use of at least one or more of the following: double submit cookie pattern, CSRF nonces, or Origin request header checks.	Required
	13.2.4	The REST services explicitly check the incoming Content-Type to be the expected one, such as application/xml or application/json.	Required
	13.2.5	The message headers and payload are trustworthy and not modified in transit. Requiring strong encryption for transport (TLS only) may be sufficient in many cases as it provides both confidentiality and integrity protection. Per-message digital signatures can provide additional assurance on top of the transport protections for high-security applications but bring with them additional complexity and risks to weigh against the benefits.	Required
13.3 SOAP Web Service	**13.3.1**	The XSD schema validation takes place to ensure a properly formed XML document, followed by validation of each input field before any processing of that data takes place.	Required
	13.3.2	The message payload is signed using WS-Security to ensure reliable transport between client and service.	Required
13.4 GraphQL	**13.4.1**	A query allow list or a combination of depth limiting and amount limiting is used to prevent GraphQL or data layer expression Denial of Service (DoS) as a result of expensive, nested queries. For more advanced scenarios, query cost analysis should be used.	Required
	13.4.2	The GraphQL or other data layer authorization logic should be implemented at the business logic layer instead of the GraphQL layer.	Required

14 Configuration

Ensure that a verified application has:

- A secure, repeatable, automatable build environment.
- Hardened third party library, dependency and configuration management such that out of date or insecure components are not included by the application.

Subcategory	#	Requirement	Assessment
14.1 Build and Deploy	**14.1.1**	The application build and deployment processes are performed in a secure and repeatable way, such as CI / CD automation, automated configuration management, and automated deployment scripts.	Required
	14.1.2	The compiler flags are configured to enable all available buffer overflow protections and warnings, including stack randomization, data execution prevention, and to break the build if an unsafe pointer, memory, format string, integer, or string operations are found.	Required
	14.1.3	The server configuration is hardened as per the recommendations of the application server and frameworks in use.	Required
	14.1.4	The application, configuration, and all dependencies can be re-deployed using automated deployment scripts, built from a documented and tested runbook in a reasonable time, or restored from backups in a timely fashion.	Required
	14.1.5	The authorized administrators can verify the integrity of all security-relevant configurations to detect tampering.	Required
14.2 Dependency	**14.2.1**	All components are up to date, preferably using a dependency checker during build or compile time.	Required
	14.2.2	All unneeded features, documentation, sample applications and configurations are removed.	Required
	14.2.3	If application assets, such as JavaScript libraries, CSS or web fonts, are hosted externally on a Content Delivery Network (CDN) or external provider, Subresource Integrity (SRI) is used to validate the integrity of the asset.	Required
	14.2.4	The third party components come from pre-defined, trusted and continually maintained repositories.	Required
	14.2.5	A Software Bill of Materials (SBOM) is maintained of all third party libraries in use.	Required
	14.2.6	The attack surface is reduced by sandboxing or encapsulating third party libraries to expose only the required behaviour into the application.	Required
14.3 Unintended Security Disclosure	**14.3.1**	The web or application server and application framework debug modes are disabled in production to eliminate debug features, developer consoles, and unintended security disclosures.	Required

Subcategory	#	Requirement	Assessment
	14.3.2	The HTTP headers or any part of the HTTP response do not expose detailed version information of system components.	Required
14.4 HTTP Security Headers	**14.4.1**	Every HTTP response contains a Content-Type header. Also specify a safe character set (e.g., UTF-8, ISO-8859-1) if the content types are text/*, /+xml and application/xml. Content must match with the provided Content-Type header.	Required
	14.4.2	All API responses contain a Content-Disposition: attachment; filename="api.json" header (or other appropriate filename for the content type).	Required
	14.4.3	A Content Security Policy (CSP) response header is in place that helps mitigate impact for XSS attacks like HTML, DOM, JSON, and JavaScript injection vulnerabilities.	Required
	14.4.4	All responses contain a X-Content-Type-Options: nosniff header.	Required
	14.4.5	A Strict-Transport-Security header is included on all responses and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains.	Required
	14.4.6	A suitable Referrer-Policy header is included to avoid exposing sensitive information in the URL through the Referer header to untrusted parties.	Required
	14.4.7	The content of a web application cannot be embedded in a third-party site by default and that embedding of the exact resources is only allowed where necessary by using suitable Content-Security-Policy: frame-ancestors and X-Frame-Options response headers.	Required
14.5 HTTP Request Header Validation	**14.5.1**	The application server only accepts the HTTP methods in use by the application/API, including pre-flight OPTIONS, and logs/alerts on any requests that are not valid for the application context.	Required
	14.5.2	The supplied Origin header is not used for authentication or access control decisions, as the Origin header can easily be changed by an attacker.	Required
	14.5.3	The Cross-Origin Resource Sharing (CORS) Access-Control-Allow-Origin header uses a strict allow list of trusted domains and subdomains to match against and does not support the "null" origin.	Required
	14.5.4	The HTTP headers added by a trusted proxy or SSO devices, such as a bearer token, are authenticated by the application.	Required

15 Other requirements

Subcategory	#	Requirement	Assessment
15.1 Testing	**15.1.1**	Test plans should be developed and documented. Test cases should consider attack and abuse use cases, with a specific focus on misuse of inputs and outputs to compromise the security of the application. Testing of complex applications with numerous inputs may be conducted on sample basis.	Required
	15.1.2	Security testing (e.g. code reviews and penetration testing) should be performed regularly. Testing should be performed at critical milestones to validate that controls operate as designed.	Required
	15.1.3	Security testing must be performed by individuals other than the originating code author. Testing must be performed by individuals with qualifications that are deemed appropriate by the Business Owner	Required
	15.1.4	Security vulnerabilities identified during testing should be addressed prior to production implementation. Any untreated security vulnerabilities must be documented, and the documentation reviewed and approved by the Business Owner.	Required
	15.1.5	Development and test environments must be kept separate from production environments.	Required
	15.1.6	Personnel assigned to the development or test environments must not have access to the production environment or data unless authorised by the Business Owner.	Required
	15.1.7	Production data should not be used for testing or development unless authorised by the Business Owner.	Required
	15.1.8	Data supplied for development must not reveal or allow the recreation of sensitive information including personal information. If production data is to be used for testing, security controls must be implemented to adequately safeguard agency data.	Required
15.2 Implementation	**15.2.1**	All documentation must be adequately protected from unauthorised access.	Required
	15.2.2	Web application components and supporting services with known or published high risk or critical vulnerabilities must not be used, or must be patched within an acceptable timeframe of the vulnerability becoming known.	Required
	15.2.3	All unnecessary application content should be removed prior to application acceptance into production. This includes removing all test and default files, test user accounts and other unnecessary content.	Required
	15.2.4	Application administration access interfaces (e.g. admin login pages) should be disabled or be restricted.	Required

Subcategory	#	Requirement	Assessment
	15.2.5	Web applications must be configured to use a service account assigned the least privileges necessary to run the applications	Required
15.3 Operations and Maintenance	**15.3.1**	Version control must be maintained for all application updates and changes.	Required
	15.3.2	All changes to applications, including updates and patches, must be reviewed and tested to ensure that there is no adverse impact on operation or security. This includes: 1 formal change control procedures must be established and documented, and evidence retained that the procedure is implemented and complied with 2 changes must be approved by the Business Owner 3 systems must only be deployed on production and public facing networks after assessment and final approval by authorised parties 4 adequate testing must take place prior to changes being applied to production systems.	Required
	15.3.3	Business continuity and recovery plans should be updated to reflect changes to production systems.	Required
	15.3.4	When significant changes or enhancements are made, a risk assessment must be performed to consider the security implications of such changes. Additional security testing should be undertaken as deemed necessary by risk assessment.	Required
	15.3.5	Vulnerability identification and patch management procedures, roles and responsibilities must be defined and followed to ensure security vulnerabilities in web applications are identified and patched.	Required
	15.3.6	Web application monitoring tools should be implemented to detect breaches or misuse of web applications.	Required
15.4 Protection of source code	**15.4.1**	The reference copy of source code must be stored in a source code library approved by the Business Owner.	Required
	15.4.2	Source code libraries must be adequately secured to protect against unauthorised or inappropriate access or changes.	Required
	15.4.3	An audit log must be maintained of all access to program source libraries.	Required
	15.4.4	The reference copy of source code must not exist on production web servers.	Required
	15.4.5	Old versions of source programs should be archived, with a clear indication of the precise dates and times when they were operational.	Required