Section		
V1: Architecture, Design and Threat Modeling Requirements		
V1.1 Secure Software Development Lifecycle Requirements		
V1	Architecture	1.1.1
V1	Architecture	1.1.2
V1	Architecture	1.1.3
V1	Architecture	1.1.4
V1	Architecture	1.1.5
V1	Architecture	1.1.6
V1	Architecture	1.1.7
V1.2 Authentication Architectural Requirements		
V1	Architecture	1.2.1
V1	Architecture	1.2.2
V1	Architecture	1.2.3
V1	Architecture	1.2.4
V1.4 Access Control Architectural Requirements		
V1	Architecture	1.4.1
V1	Architecture	1.4.2

V1	Architecture	1.4.3
V1	Architecture	1.4.4
V1	Architecture	1.4.5
V1.5 Input and Output Architectural Requirements		
V1	Architecture	1.5.1
V1	Architecture	1.5.2
V1	Architecture	1.5.3
V1	Architecture	1.5.4
V1.6 Cryptographic Architectural Requirements		
V1	Architecture	1.6.1
V1	Architecture	1.6.2
V1	Architecture	1.6.3
V1	Architecture	1.6.4
V1.7 Errors, Logging and Auditing Architectural Requirements		
V1	Architecture	1.7.1
V1	Architecture	1.7.2
V1.8 Data Protection and Privacy Architectural Requirements		

V1	Architecture	1.8.1
V1	Architecture	1.8.2
V1.9 Communications Architectural Requirements		
V1	Architecture	1.9.1
V1	Architecture	1.9.2
V1.10 Malicious Software Architectural Requirements		
V1	Architecture	1.10.1
V1.11 Business Logic Architectural Requirements		
V1	Architecture	1.11.1
V1	Architecture	1.11.2
V1	Architecture	1.11.3
V1.12 Secure File Upload Architectural Requirements		
V1	Architecture	1.12.1
V1	Architecture	1.12.2
V1.14 Configuration Architectural Requirements		
V1	Architecture	1.14.1
V1	Architecture	1.14.2

V1	Architecture	1.14.3
V1	Architecture	1.14.4
V1	Architecture	1.14.5
V1	Architecture	1.14.6
V2: Authentication Verification Requirements		
V2.1 Password Security Requirements		
V2	Authentication	2.1.1
V2	Authentication	2.1.2
V2	Authentication	2.1.3
V2	Authentication	2.1.4
V2	Authentication	2.1.5
V2	Authentication	2.1.6
V2	Authentication	2.1.7
V2	Authentication	2.1.8
V2	Authentication	2.1.9
V2	Authentication	2.1.10
V2	Authentication	2.1.11

Control
Verify the use of a secure software development lifecycle that addresses security in all stages of development.
Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.
Verify that 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"
Verify documentation and justification of all the application's trust boundaries, components, and significant data flows.
Verify definition and security analysis of the application's high-level architecture and all connected remote services.
Verify implementation of centralized, simple (economy of design), vetted, secure, and reusable security controls to avoid duplicate, missing, ineffective, or insecure controls.
Verify availability of a secure coding checklist, security requirements, guideline, or policy to all developers and testers.
Verify the use of unique or special low-privilege operating system accounts for all application components, services, and servers.
Verify that communications between application components, including APIs, middleware and data layers, are authenticated. Components should have the least necessary privileges needed.
Verify that 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.
Verify that 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.
Verify that trusted enforcement points such as at access control gateways, servers, and serverless functions enforce access controls. Never enforce access controls on the client.
Verify that the chosen access control solution is flexible enough to meet the application's needs.

Verify enforcement of the principle of least privilege in functions, data files, URLs, controllers, services, and other resources. This implies protection against spoofing and elevation of privilege.
Verify 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.
Verify that 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.
Verify that input and output requirements clearly define how to handle and process data based on type, content, and applicable laws, regulations, and other policy compliance.
Verify that 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.
Verify that input validation is enforced on a trusted service layer.
Verify that output encoding occurs close to or by the interpreter for which it is intended. ([C4](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Nu mbering))
Verify that there is an explicit policy for management of cryptographic keys and that a cryptographic key lifecycle follows a key management standard such as NIST SP 800- 57.
Verify that consumers of cryptographic services protect key material and other secrets by using key vaults or API based alternatives.
Verify that all keys and passwords are replaceable and are part of a well-defined process to re-encrypt sensitive data.
Verify that symmetric keys, passwords, or API secrets generated by or shared with clients are used only in protecting low risk secrets, such as encrypting local storage, or temporary ephemeral uses such as parameter obfuscation. Sharing secrets with clients is clear-text equivalent and architecturally should be treated as such.
Verify that a common logging format and approach is used across the system.
Verify that logs are securely transmitted to a preferably remote system for analysis, detection, alerting, and escalation.

Verify that all sensitive data is identified and classified into protection levels.
Verify that 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.
Verify the application encrypts communications between components, particularly when these components are in different containers, systems, sites, or cloud providers.
Verify that 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.
Verify that 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.
Verify the definition and documentation of all application components in terms of the business or security functions they provide.
Verify that all high-value business logic flows, including authentication, session management and access control, do not share unsynchronized state.
Verify that 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.
Verify that user-uploaded files are stored outside of the web root.
Verify that 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 to reduce the risk from XSS vectors or other attacks from the uploaded file.
Verify 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.
Verify that if deploying binaries to untrusted devices makes use of binary signatures, trusted connections, and verified endpoints.

Verify that the build pipeline warns of out-of-date or insecure components and takes appropriate actions.
Verify that 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.
Verify that 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.
Verify 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.
Verify that user set passwords are at least 12 characters in length.
Verify that passwords 64 characters or longer are permitted.
Verify that passwords can contain spaces and truncation is not performed. Consecutive multiple spaces MAY optionally be coalesced.
Verify that Unicode characters are permitted in passwords. A single Unicode code point is considered a character, so 12 emoji or 64 kanji characters should be valid and permitted.
Verify users can change their password.
Verify that password change functionality requires the user's current and new password.
Verify that 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.
Verify that a password strength meter is provided to help users set a stronger password.
Verify that 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.
Verify that there are no periodic credential rotation or password history requirements.
Verify that "paste" functionality, browser password helpers, and external password managers are permitted.

V2	Authentication	2.1.12
V2.2 General Authenticator Requirements		
V2	Authentication	2.2.1
V2	Authentication	2.2.2
V2	Authentication	2.2.3
V2	Authentication	2.2.4
V2	Authentication	2.2.5
V2	Authentication	2.2.6
V2	Authentication	2.2.7
V2.3 Authenticator Lifecycle Requirements		
V2	Authentication	2.3.1
V2	Authentication	2.3.2
V2	Authentication	2.3.3
V2.4 Credential Storage Requirements		

V2	Authentication	2.4.1
V2	Authentication	2.4.2
V2	Authentication	2.4.3
V2	Authentication	2.4.4
V2	Authentication	2.4.5
V2.5 Credential Recovery Requirements		
V2	Authentication	2.5.1
V2	Authentication	2.5.2
V2	Authentication	2.5.3
V2	Authentication	2.5.4
V2	Authentication	2.5.5
V2	Authentication	2.5.6
V2	Authentication	2.5.7
V2.6 Look-up Secret Verifier Requirements		
V2	Authentication	2.6.1
V2	Authentication	2.6.2
V2	Authentication	2.6.3
V2.7 Out of Band Verifier Requirements		
V2	Authentication	2.7.1
V2	Authentication	2.7.2

V2	Authentication	2.7.3
V2	Authentication	2.7.4
V2	Authentication	2.7.5
V2	Authentication	2.7.6
V2.8 Single or Multi Factor One Time Verifier Requirements		
V2	Authentication	2.8.1
V2	Authentication	2.8.2
V2	Authentication	2.8.3
V2	Authentication	2.8.4
V2	Authentication	2.8.5
V2	Authentication	2.8.6
V2	Authentication	2.8.7
V2.9 Cryptographic Software and Devices Verifier Requirements		
V2	Authentication	2.9.1
V2	Authentication	2.9.2
V2	Authentication	2.9.3
V2.10 Service Authentication Requirements		
V2	Authentication	2.10.1
V2	Authentication	2.10.2
V2	Authentication	2.10.3
V2	Authentication	2.10.4

V3: Session Management Verification Requirements		
V3.1 Fundamental Session Management Requirements		
V3	Session	3.1.1
V3.2 Session Binding Requirements		
V3	Session	3.2.1
V3	Session	3.2.2
V3	Session	3.2.3
V3	Session	3.2.4
V3.3 Session Logout and Timeout Requirements		
V3	Session	3.3.1
V3	Session	3.3.2
V3	Session	3.3.3
V3	Session	3.3.4
V3.4 Cookie-based Session Management		
V3	Session	3.4.1
V3	Session	3.4.2
V3	Session	3.4.3
V3	Session	3.4.4
V3	Session	3.4.5
V3.5 Token-based Session Management		
V3	Session	3.5.1
V3	Session	3.5.2
V3	Session	3.5.3
V3.6 Re-authentication from a Federation or Assertion		

V3	Session	3.6.1
V3	Session	3.6.2
V3.7 Defenses Against Session Management Exploits		
V3	Session	3.7.1
V4: Access Control Verification Requirements		
V4.1 General Access Control Design		
V4	Access	4.1.1
V4	Access	4.1.2
V4	Access	4.1.3
V4	Access	4.1.4
V4	Access	4.1.5
V4.2 Operation Level Access Control		
V4	Access	4.2.1
V4	Access	4.2.2
V4.3 Other Access Control Considerations		
V4	Access	4.3.1
V4	Access	4.3.2
V4	Access	4.3.3
V5: Validation, Sanitization and Encoding Verification Requirements		
V5.1 Input Validation Requirements		
V5	Validation	5.1.1

V5	Validation	5.1.2
V5	Validation	5.1.3
V5	Validation	5.1.4
V5	Validation	5.1.5
V5.2 Sanitization and Sandboxing Requirements		
V5	Validation	5.2.1
V5	Validation	5.2.2
V5	Validation	5.2.3
V5	Validation	5.2.4
V5	Validation	5.2.5
V5	Validation	5.2.6
V5	Validation	5.2.7
V5	Validation	5.2.8
V5.3 Output encoding and Injection Prevention Requirements		
V5	Validation	5.3.1
V5	Validation	5.3.2
V5	Validation	5.3.3

V5	Validation	5.3.4
V5	Validation	5.3.5
V5	Validation	5.3.6
V5	Validation	5.3.7
V5	Validation	5.3.8
V5	Validation	5.3.9
V5	Validation	5.3.10
V5.4 Memory, String, and Unmanaged Code Requirements		
V5	Validation	5.4.1
V5	Validation	5.4.2
V5	Validation	5.4.3
V5.5 Deserialization Prevention Requirements		
V5	Validation	5.5.1
V5	Validation	5.5.2
V5	Validation	5.5.3
V5	Validation	5.5.4
V6: Stored Cryptography Verification Requirements		
V6.1 Data Classification		
V6	Cryptography	6.1.1
V6	Cryptography	6.1.2

V6	Cryptography	6.1.3
V6.2 Algorithms		
V6	Cryptography	6.2.1
V6	Cryptography	6.2.2
V6	Cryptography	6.2.3
V6	Cryptography	6.2.4
V6	Cryptography	6.2.5
V6	Cryptography	6.2.6
V6	Cryptography	6.2.7
V6	Cryptography	6.2.8
V6.3 Random Values		
V6	Cryptography	6.3.1
V6	Cryptography	6.3.2
V6	Cryptography	6.3.3
V6.4 Secret Management		
V6	Cryptography	6.4.1
V6	Cryptography	6.4.2
V7: Error Handling and Logging Verification Requirements		

V7.1 Log Content Requirements		
V7	Error	7.1.1
V7	Error	7.1.2
V7	Error	7.1.3
V7	Error	7.1.4
V7.2 Log Processing Requirements		
V7	Error	7.2.1
V7	Error	7.2.2
V7.3 Log Protection Requirements		
V7	Error	7.3.1
V7	Error	7.3.2
V7	Error	7.3.3
V7	Error	7.3.4
V7.4 Error Handling		
V7	Error	7.4.1
V7	Error	7.4.2
V7	Error	7.4.3
V8: Data Protection Verification Requirements		
V8.1 General Data Protection		
V8	Data	8.1.1
V8	Data	8.1.2

V8	Data	8.1.3
V8	Data	8.1.4
V8	Data	8.1.5
V8	Data	8.1.6
V8.2 Client-side Data Protection		
V8	Data	8.2.1
V8	Data	8.2.2
V8	Data	8.2.3
V8.3 Sensitive Private Data		
V8	Data	8.3.1
V8	Data	8.3.2
V8	Data	8.3.3
V8	Data	8.3.4
V8	Data	8.3.5
V8	Data	8.3.6
V8	Data	8.3.7
V8	Data	8.3.8
V9: Communications Verification Requirements		
V9.1 Communications Security Requirements		

V9	Communication s	9.1.1
V9	Communication s	9.1.2
V9	Communication s	9.1.3
V9.2 Server Communications Security Requirements		
V9	Communication s	9.2.1
V9	Communication s	9.2.2
V9	Communication s	9.2.3
V9	Communication s	9.2.4
V9	Communication s	9.2.5
V10: Malicious Code Verification Requirements		
V10.1 Code Integrity Controls		

Verify that 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 native functionality.
Verify that 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 100 failed attempts per hour is possible on a single account.
Verify that 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.
Verify that 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.
Verify 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.
Verify that where a credential service provider (CSP) and the application verifying authentication are separated, mutually authenticated TLS is in place between the two endpoints.
Verify replay resistance through the mandated use of OTP devices, cryptographic authenticators, or lookup codes.
Verify 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.
Verify 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.
Verify that enrollment and use of subscriber-provided authentication devices are supported, such as a U2F or FIDO tokens.
Verify that renewal instructions are sent with sufficient time to renew time bound authenticators.

Verify that 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.
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.
Verify that if PBKDF2 is used, the iteration count SHOULD be as large as verification server performance will allow, typically at least 100,000 iterations.
Verify that if bcrypt is used, the work factor SHOULD be as large as verification server performance will allow, typically at least 13.
Verify that 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 [SP 800-90Ar1] and provide at least the minimum security strength specified in the latest revision of SP 800-131A. The secret salt value SHALL be stored separately from the hashed passwords (e.g., in a specialized device like a hardware security module).
Verify that a system generated initial activation or recovery secret is not sent in clear text to the user.
Verify password hints or knowledge-based authentication (so-called "secret questions") are not present.
Verify password credential recovery does not reveal the current password in any way.
Verify shared or default accounts are not present (e.g. "root", "admin", or "sa").
Verify that if an authentication factor is changed or replaced, that the user is notified of this event.
Verify forgotten password, and other recovery paths use a secure recovery mechanism, such as TOTP or other soft token, mobile push, or another offline recovery mechanism.
Verify that if OTP or multi-factor authentication factors are lost, that evidence of identity proofing is performed at the same level as during enrollment.
Verify that lookup secrets can be used only once.
Verify that 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.
Verify that lookup secrets are resistant to offline attacks, such as predictable values.
Verify that clear text out of band (NIST "restricted") authenticators, such as SMS or PSTN, are not offered by default, and stronger alternatives such as push notifications are offered first.
Verify that the out of band verifier expires out of band authentication requests, codes, or tokens after 10 minutes.

Verify that the out of band verifier authentication requests, codes, or tokens are only usable once, and only for the original authentication request.
Verify that the out of band authenticator and verifier communicates over a secure independent channel.
Verify that the out of band verifier retains only a hashed version of the authentication code.
Verify that 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).
Verify that time-based OTPs have a defined lifetime before expiring.
Verify that 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.
Verify that approved cryptographic algorithms are used in the generation, seeding, and verification.
Verify that time-based OTP can be used only once within the validity period.
Verify that 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.
Verify 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.
Verify that biometric authenticators are limited to use only as secondary factors in conjunction with either something you have and something you know.
Verify that cryptographic keys used in verification are stored securely and protected against disclosure, such as using a TPM or HSM, or an OS service that can use this secure storage.
Verify that the challenge nonce is at least 64 bits in length, and statistically unique or unique over the lifetime of the cryptographic device.
Verify that approved cryptographic algorithms are used in the generation, seeding, and verification.
Verify that integration secrets do not rely on unchanging passwords, such as API keys or shared privileged accounts.
Verify that if passwords are required, the credentials are not a default account.
Verify that passwords are stored with sufficient protection to prevent offline recovery attacks, including local system access.
Verify 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 (L1), hardware trusted platform module (TPM), or a hardware security module (L3) is recommended for password storage.

Verify the application never reveals session tokens in URL parameters or error messages.
Verify the application generates a new session token on user authentication.
Verify that session tokens possess at least 64 bits of entropy.
Verify the application only stores session tokens in the browser using secure methods such as appropriately secured cookies (see section 3.4) or HTML 5 session storage.
Verify that session token are generated using approved cryptographic algorithms.
Verify that 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.
If authenticators permit users to remain logged in, verify that re-authentication occurs periodically both when actively used or after an idle period.
Verify that the application terminates all other active sessions after a successful password change, and that this is effective across the application, federated login (if present), and any relying parties.
Verify that users are able to view and log out of any or all currently active sessions and devices.
Verify that cookie-based session tokens have the 'Secure' attribute set.
Verify that cookie-based session tokens have the 'HttpOnly' attribute set.
Verify that cookie-based session tokens utilize the 'SameSite' attribute to limit exposure to cross-site request forgery attacks.
Verify that cookie-based session tokens use "__Host-" prefix (see references) to provide session cookie confidentiality.
Verify that if the application is published under a domain name with other applications that set or use session cookies that might override or disclose the session cookies, set the path attribute in cookie-based session tokens using the most precise path possible.
Verify the application does not treat OAuth and refresh tokens - on their own - as the presence of the subscriber and allows users to terminate trust relationships with linked applications.
Verify the application uses session tokens rather than static API secrets and keys, except with legacy implementations.
Verify that stateless session tokens use digital signatures, encryption, and other countermeasures to protect against tampering, enveloping, replay, null cipher, and key substitution attacks.

Verify that relying parties specify the maximum authentication time to CSPs and that CSPs re-authenticate the subscriber if they haven't used a session within that period.
Verify that CSPs inform relying parties of the last authentication event, to allow RPs to determine if they need to re-authenticate the user.
Verify the application ensures a valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.
Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.
Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.
Verify that 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.
Verify that the principle of deny by default exists whereby new users/roles start with minimal or no permissions and users/roles do not receive access to new features until access is explicitly assigned.
Verify that access controls fail securely including when an exception occurs.
Verify that sensitive data and APIs are protected against direct object 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.
Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.
Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.
Verify that 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.
Verify 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.
Verify that 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).

Verify that 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.
Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (whitelisting).
Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers or telephone, or validating that two related fields are reasonable, such as checking that suburb and zip/postcode match).
Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content.
Verify that all untrusted HTML input from WYSIWYG editors or similar is properly sanitized with an HTML sanitizer library or framework feature.
Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length.
Verify that the application sanitizes user input before passing to mail systems to protect against SMTP or IMAP injection.
Verify that 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.
Verify that the application protects against template injection attacks by ensuring that any user input being included is sanitized or sandboxed.
Verify that the application protects against SSRF attacks, by validating or sanitizing untrusted data or HTTP file metadata, such as filenames and URL input fields, use whitelisting of protocols, domains, paths and ports.
Verify that the application sanitizes, disables, or sandboxes user-supplied SVG scriptable content, especially as they relate to XSS resulting from inline scripts, and foreignObject.
Verify that the application sanitizes, disables, or sandboxes user-supplied scriptable or expression template language content, such as Markdown, CSS or XSL stylesheets, BBCode, or similar.
Verify that 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 ã?ã?“ or O'Hara).
Verify that output encoding preserves the user's chosen character set and locale, such that any Unicode character point is valid and safely handled.
Verify that context-aware, preferably automated - or at worst, manual - output escaping protects against reflected, stored, and DOM based XSS.

Verify that 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.
Verify that 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.
Verify that the application projects against JavaScript or JSON injection attacks, including for eval attacks, remote JavaScript includes, CSP bypasses, DOM XSS, and JavaScript expression evaluation.
Verify that the application protects against LDAP Injection vulnerabilities, or that specific security controls to prevent LDAP Injection have been implemented.
Verify that the application protects against OS command injection and that operating system calls use parameterized OS queries or use contextual command line output encoding.
Verify that the application protects against Local File Inclusion (LFI) or Remote File Inclusion (RFI) attacks.
Verify that the application protects against XPath injection or XML injection attacks.
Verify that the application uses memory-safe string, safer memory copy and pointer arithmetic to detect or prevent stack, buffer, or heap overflows.
Verify that format strings do not take potentially hostile input, and are constant.
Verify that sign, range, and input validation techniques are used to prevent integer overflows.
Verify that serialized objects use integrity checks or are encrypted to prevent hostile object creation or data tampering.
Verify that 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 XXE.
Verify that deserialization of untrusted data is avoided or is protected in both custom code and third-party libraries (such as JSON, XML and YAML parsers).
Verify that 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.
Verify that 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.
Verify that regulated health data is stored encrypted while at rest, such as medical records, medical device details, or de-anonymized research records.

Verify that 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.
Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable Padding Oracle attacks.
Verify that industry proven or government approved cryptographic algorithms, modes, and libraries are used, instead of custom coded cryptography.
Verify that encryption initialization vector, cipher configuration, and block modes are configured securely using the latest advice.
Verify that 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.
Verify that 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.
Verify that 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.
Verify that encrypted data is authenticated via signatures, authenticated cipher modes, or HMAC to ensure that ciphertext is not altered by an unauthorized party.
Verify that all cryptographic operations are constant-time, with no 'short-circuit' operations in comparisons, calculations, or returns, to avoid leaking information.
Verify that 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.
Verify that 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.
Verify that random numbers are created with proper entropy even when the application is under heavy load, or that the application degrades gracefully in such circumstances.
Verify that a secrets management solution such as a key vault is used to securely create, store, control access to and destroy secrets.
Verify that key material is not exposed to the application but instead uses an isolated security module like a vault for cryptographic operations.

Verify that the application does not log credentials or payment details. Session tokens should only be stored in logs in an irreversible, hashed form.
Verify that the application does not log other sensitive data as defined under local privacy laws or relevant security policy.
Verify that the application logs security relevant events including successful and failed authentication events, access control failures, deserialization failures and input validation failures.
Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens.
Verify that all authentication decisions are logged, without storing sensitive session identifiers or passwords. This should include requests with relevant metadata needed for security investigations.
Verify that all access control decisions can be logged and all failed decisions are logged. This should include requests with relevant metadata needed for security investigations.
Verify that the application appropriately encodes user-supplied data to prevent log injection.
Verify that all events are protected from injection when viewed in log viewing software.
Verify that security logs are protected from unauthorized access and modification.
Verify that 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.
Verify that 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.
Verify that exception handling (or a functional equivalent) is used across the codebase to account for expected and unexpected error conditions.
Verify that a "last resort" error handler is defined which will catch all unhandled exceptions.
Verify the application protects sensitive data from being cached in server components such as load balancers and application caches.
Verify that 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.

Verify the application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values.
Verify 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.
Verify that regular backups of important data are performed and that test restoration of data is performed.
Verify that backups are stored securely to prevent data from being stolen or corrupted.
Verify the application sets sufficient anti-caching headers so that sensitive data is not cached in modern browsers.
Verify that data stored in client side storage (such as HTML5 local storage, session storage, IndexedDB, regular cookies or Flash cookies) does not contain sensitive data or PII.
Verify that authenticated data is cleared from client storage, such as the browser DOM, after the client or session is terminated.
Verify that 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.
Verify that users have a method to remove or export their data on demand.
Verify that 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.
Verify that 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.
Verify 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.
Verify that 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.
Verify that sensitive or private information that is required to be encrypted, is encrypted using approved algorithms that provide both confidentiality and integrity.
Verify that 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.

Verify that secured TLS is used for all client connectivity, and does not fall back to insecure or unencrypted protocols.
Verify using online or up to date TLS testing tools that only strong algorithms, ciphers, and protocols are enabled, with the strongest algorithms and ciphers set as preferred.
Verify that old versions of SSL and TLS protocols, algorithms, ciphers, and configuration are disabled, such as SSLv2, SSLv3, or TLS 1.0 and TLS 1.1. The latest version of TLS should be the preferred cipher suite.
Verify that 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.
Verify that 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.
Verify that all encrypted connections to external systems that involve sensitive information or functions are authenticated.
Verify that proper certification revocation, such as Online Certificate Status Protocol (OCSP) Stapling, is enabled and configured.
Verify that backend TLS connection failures are logged.

V10	Malicious	10.1.1	Verify that a code a functions, unsafe fi
V10.2 Malicious Code Search			
V10	Malicious	10.2.1	Verify that the app home or data colle to operate before
V10	Malicious	10.2.2	Verify that the app features or sensors
V10	Malicious	10.2.3	Verify that the app coded or additiona rootkits, or anti-de functionality that c
V10	Malicious	10.2.4	Verify that the app searching for date
V10	Malicious	10.2.5	Verify that the app salami attacks, logi
V10	Malicious	10.2.6	Verify that the app potentially unwant
V10.3 Deployed Application Integrity Controls			
V10	Malicious	10.3.1	Verify that if the ap over secure channe update before inst
V10	Malicious	10.3.2	Verify that the app The application mu modules, plugins, c
V10	Malicious	10.3.3	Verify that the app entries or DNS sub- expired projects at buckets (autogen-b names used by app
V11: Business Logic Verification Requirements			
V11.1 Business Logic Security Requirements			
V11	BusLogic	11.1.1	Verify the applicati and without skippi
V11	BusLogic	11.1.2	Verify the applicati human time, i.e. tr

V11	BusLogic	11.1.3	Verify the applicati correctly enforced
V11	BusLogic	11.1.4	Verify the applicati exfiltration, excess
V11	BusLogic	11.1.5	Verify the applicati threats, identified
V11	BusLogic	11.1.6	Verify the applicati conditions for sens
V11	BusLogic	11.1.7	Verify the applicati example, attempts
V11	BusLogic	11.1.8	Verify the applicati
V12: File and Resources Verification Requirements			
V12.1 File Upload Requirements			
V12	Files	12.1.1	Verify that the app attack.
V12	Files	12.1.2	Verify that compre huge files thus exh
V12	Files	12.1.3	Verify that a file siz user cannot fill up
V12.2 File Integrity Requirements			
V12	Files	12.2.1	Verify that files obt content.
V12.3 File execution Requirements			
V12	Files	12.3.1	Verify that user-sul URL API to protect
V12	Files	12.3.2	Verify that user-sul creation, updating
V12	Files	12.3.3	Verify that user-sul execution of remot
V12	Files	12.3.4	Verify that the app submitted filenam set to text/plain, a

V12	Files	12.3.5	Verify that untrust OS command injec
V12	Files	12.3.6	Verify that the app unverified content
V12.4 File Storage Requirements			
V12	Files	12.4.1	Verify that files obt permissions, prefe
V12	Files	12.4.2	Verify that files obt known malicious c
V12.5 File Download Requirements			
V12	Files	12.5.1	Verify that the web unintentional infor working files (e.g. . editors should be b
V12	Files	12.5.2	Verify that direct r
V12.6 SSRF Protection Requirements			
V12	Files	12.6.1	Verify that the web the server can send
V13: API and Web Service Verification Requirements			
V13.1 Generic Web Service Security Verification Requirements			
V13	API	13.1.1	Verify that all appli exploit different UI
V13	API	13.1.2	Verify that access t
V13	API	13.1.3	Verify API URLs do
V13	API	13.1.4	Verify that authori security at the con
V13	API	13.1.5	Verify that request headers (HTTP resp
V13.2 RESTful Web Service Verification Requirements			
V13	API	13.2.1	Verify that enablec normal users using
V13	API	13.2.2	Verify that JSON sc

V13	API	13.2.3	Verify that RESTful the use of at least CSRF nonces, or OF
V13	API	13.2.4	Verify that REST se the API is unauther
V13	API	13.2.5	Verify that REST se application/xml or
V13	API	13.2.6	Verify that the mes strong encryption f confidentiality and on top of the trans complexity and risk
V13.3 SOAP Web Service Verification Requirements			
V13	API	13.3.1	Verify that XSD sch validation of each i
V13	API	13.3.2	Verify that the mes and service.
V13.4 GraphQL and other Web Service Data Layer Security Requirements			
V13	API	13.4.1	Verify that query w prevent GraphQL c queries. For more a
V13	API	13.4.2	Verify that GraphQ layer instead of the
V14: Configuration Verification Requirements			
V14.1 Build			
V14	Config	14.1.1	Verify that the app way, such as CI / CI scripts.
V14	Config	14.1.2	Verify that compile warnings, including pointer, memory, f
V14	Config	14.1.3	Verify that server c frameworks in use.
V14	Config	14.1.4	Verify that the app deployment scripts from backups in a t

V14	Config	14.1.5	Verify that authori detect tampering.
V14.2 Dependency			
V14	Config	14.2.1	Verify that all com compile time.
V14	Config	14.2.2	Verify that all unne applications, platfc
V14	Config	14.2.3	Verify that if applic externally on a con to validate the inte
V14	Config	14.2.4	Verify that third pa repositories.
V14	Config	14.2.5	Verify that an inve
V14	Config	14.2.6	Verify that the atta only the required k
V14.3 Unintended Security Disclosure Requirements			
V14	Config	14.3.1	Verify that web or actionable, custom
V14	Config	14.3.2	Verify that web or production to elim
V14	Config	14.3.3	Verify that the HTT information of syst
V14.4 HTTP Security Headers Requirements			
V14	Config	14.4.1	Verify that every H 8, ISO 8859-1).
V14	Config	14.4.2	Verify that all API r appropriate filenar
V14	Config	14.4.3	Verify that a conte HTML, DOM, JSON,
V14	Config	14.4.4	Verify that all resp
V14	Config	14.4.5	Verify that HTTP St such as Strict-Trans
V14	Config	14.4.6	Verify that a suitab
V14	Config	14.4.7	Verify that a suitab sites where conten

V14.5 Validate HTTP Request Header Requirements			
V14	Config	14.5.1	Verify that the app including pre-flight
V14	Config	14.5.2	Verify that the sup Origin header can
V14	Config	14.5.3	Verify that the cros white-list of truste
V14	Config	14.5.4	Verify that HTTP he authenticated by t

Source: OWASP Application Security Verification Standard v4.0

MADE EASY

analysis tool is in use that can detect potentially malicious code, such as time ile operations and network connections.
lication source code and third party libraries do not contain unauthorized phone ction capabilities. Where such functionality exists, obtain the user's permission for it collecting any data.
lication does not ask for unnecessary or excessive permissions to privacy related , such as contacts, cameras, microphones, or location.
lication source code and third party libraries do not contain back doors, such as hard- l undocumented accounts or keys, code obfuscation, undocumented binary blobs, bugging, insecure debugging features, or otherwise out of date, insecure, or hidden could be used maliciously if discovered.
lication source code and third party libraries does not contain time bombs by and time related functions.
lication source code and third party libraries does not contain malicious code, such as c bypasses, or logic bombs.
lication source code and third party libraries do not contain Easter eggs or any other ed functionality.
plication has a client or server auto-update feature, updates should be obtained els and digitally signed. The update code must validate the digital signature of the alling or executing the update.
lication employs integrity protections, such as code signing or sub- resource integrity. ust not load or execute code from untrusted sources, such as loading includes, code, or libraries from untrusted sources or the Internet.
lication has protection from sub-domain takeovers if the application relies upon DNS -domains, such as expired domain names, out of date DNS pointers or CNAMEs, public source code repos, or transient cloud APIs, serverless functions, or storage ucket- id.cloud.example.com) or similar. Protections can include ensuring that DNS plications are regularly checked for expiry or change.
on will only process business logic flows for the same user in sequential step order ng steps.
on will only process business logic flows with all steps being processed in realistic ansactions are not submitted too quickly.

on has appropriate limits for specific business actions or transactions which are on a per user basis.
on has sufficient anti-automation controls to detect and protect against data ive business logic requests, excessive file uploads or denial of service attacks.
on has business logic limits or validation to protect against likely business risks or using threat modelling or similar methodologies.
on does not suffer from "time of check to time of use" (TOCTOU) issues or other race itive operations.
on monitors for unusual events or activity from a business logic perspective. For to perform actions out of order or actions which a normal user would never attempt.
on has configurable alerting when automated attacks or unusual activity is detected.
lication will not accept large files that could fill up storage or cause a denial of service
ssed files are checked for "zip bombs" - small input files that will decompress into austing file storage limits.
e quota and maximum number of files per user is enforced to ensure that a single the storage with too many files, or excessively large files.
tained from untrusted sources are validated to be of expected type based on the file's
bmitted filename metadata is not used directly with system or framework file and against path traversal.
bmitted filename metadata is validated or ignored to prevent the disclosure, or removal of local files (LFI).
bmitted filename metadata is validated or ignored to prevent the disclosure or e files (RFI), which may also lead to SSRF.
lication protects against reflective file download (RFD) by validating or ignoring user- es in a JSON, JSONP, or URL parameter, the response Content-Type header should be nd the Content- Disposition header should have a fixed filename.

ed file metadata is not used directly with system API or libraries, to protect against tion.
lication does not include and execute functionality from untrusted sources, such as distribution networks, JavaScript libraries, node npm libraries, or server-side DLLs.
tained from untrusted sources are stored outside the web root, with limited rably with strong validation.
tained from untrusted sources are scanned by antivirus scanners to prevent upload of ontent.
tier is configured to serve only files with specific file extensions to prevent mation and source code leakage. For example, backup files (e.g. .bak), temporary swp), compressed files (.zip, .tar.gz, etc) and other extensions commonly used by locked unless required.
equests to uploaded files will never be executed as HTML/JavaScript content.
or application server is configured with a whitelist of resources or systems to which requests or load data/files from.
ication components use the same encodings and parsers to avoid parsing attacks that RI or file parsing behavior that could be used in SSRF and RFI attacks.
o administration and management functions is limited to authorized administrators.
not expose sensitive information, such as the API key, session tokens etc.
zation decisions are made at both the URI, enforced by programmatic or declarative troller or router, and at the resource level, enforced by model-based permissions.
s containing unexpected or missing content types are rejected with appropriate onse status 406 Unacceptable or 415 Unsupported Media Type).
RESTful HTTP methods are a valid choice for the user or action, such as preventing DELETE or PUT on protected API or resources.
hema validation is in place and verified before accepting input.

web services that utilize cookies are protected from Cross-Site Request Forgery via one or more of the following: triple or double submit cookie pattern (see references), RIGIN request header checks.
rvices have anti-automation controls to protect against excessive calls, especially if nticated.
rvices explicitly check the incoming Content-Type to be the expected one, such as application/JSON.
ssage headers and payload are trustworthy and not modified in transit. Requiring for transport (TLS only) may be sufficient in many cases as it provides both integrity protection. Per-message digital signatures can provide additional assurance port protections for high-security applications but bring with them additional ks to weigh against the benefits.
ema validation takes place to ensure a properly formed XML document, followed by nput field before any processing of that data takes place.
ssage payload is signed using WS-Security to ensure reliable transport between client
whitelisting or a combination of depth limiting and amount limiting should be used to or data layer expression denial of service (DoS) as a result of expensive, nested advanced scenarios, query cost analysis should be used.
QL or other data layer authorization logic should be implemented at the business logic e GraphQL layer.
lication build and deployment processes are performed in a secure and repeatable D automation, automated configuration management, and automated deployment
er flags are configured to enable all available buffer overflow protections and g stack randomization, data execution prevention, and to break the build if an unsafe format string, integer, or string operations are found.
configuration is hardened as per the recommendations of the application server and
lication, configuration, and all dependencies can be re-deployed using automated s, built from a documented and tested runbook in a reasonable time, or restored timely fashion.

zed administrators can verify the integrity of all security-relevant configurations to
ponents are up to date, preferably using a dependency checker during build or
eeded features, documentation, samples, configurations are removed, such as sample rm documentation, and default or example users.
ation assets, such as JavaScript libraries, CSS stylesheets or web fonts, are hosted tent delivery network (CDN) or external provider, Subresource Integrity (SRI) is used grity of the asset.
rty components come from pre-defined, trusted and continually maintained
ntory catalog is maintained of all third party libraries in use.
ck surface is reduced by sandboxing or encapsulating third party libraries to expose behaviour into the application.
application server and framework error messages are configured to deliver user ized responses to eliminate any unintended security disclosures.
application server and application framework debug modes are disabled in inate debug features, developer consoles, and unintended security disclosures.
P headers or any part of the HTTP response do not expose detailed version em components.
TTP response contains a content type header specifying a safe character set (e.g., UTF-
esponses contain Content-Disposition: attachment; filename="api.json" (or other ne for the content type).
nt security policy (CSPv2) is in place that helps mitigate impact for XSS attacks like , and JavaScript injection vulnerabilities.
onses contain X-Content-Type-Options: nosniff.
rict Transport Security headers are included on all responses and for all subdomains, sport-Security: max-age=15724800; includeSubdomains.
le "Referrer-Policy" header is included, such as "no-referrer" or "same-origin".
le X-Frame-Options or Content-Security-Policy: frame-ancestors header is in use for t should not be embedded in a third-party site.

lication server only accepts the HTTP methods in use by the application or API, OPTIONS.
plied Origin header is not used for authentication or access control decisions, as the easily be changed by an attacker.
ss-domain resource sharing (CORS) Access-Control-Allow-Origin header uses a strict d domains to match against and does not support the "null" origin.
eaders added by a trusted proxy or SSO devices, such as a bearer token, are he application.