GUIDE AND CHECKLIST

Web application security: recommended measures

A guide and checklist for bringing a small web app to a high level of security

Salvatore Vivolo · November 2026 · v1.2

This guide collects the measures that make a web app hard to attack and able to recover when something goes wrong. For each measure it explains **what to do**, **why**, **which attacks or incidents it prevents** and **which reference it maps to** (OWASP, NIST, RFCs, GDPR). At the end there is a mapping to OWASP ASVS 5.0 and a checklist to tick off.

It grew out of real work: the security and privacy review of IkigaiApp, a small web app with real users, described in the case study published at https://salvatorevivolo.org/en/projects/ikigaiapp/security-case-study. Here the lessons of that project are generalised and completed with the measures any similar app needs.

There is also an **AI-agent version** (`SECURITY-BASELINE.md`): the same rules written as MUST/SHOULD requirements, each with a verification check, to keep in the repository so that coding assistants follow them.

Scope and limits

Who it is for. Small and medium projects with one or a few developers, a web frontend, an API, a managed database and external services (email, AI, payments, storage). Examples use Node.js, Express, Next.js and MongoDB, but the principles apply to any stack.

What it does not cover. Native mobile apps, desktop software, implementing your own OAuth/OIDC server, WebRTC, complex multi-tenant architectures, sectors with specific regulation (card payments under PCI DSS, health data, critical infrastructure). These need additional requirements.

What it is not.

- It is not a certification: following the guide does not make an app compliant with a stated OWASP ASVS level. The mapping table is indicative.
- It does not replace a penetration test by independent professionals.
- It is not legal advice: the GDPR parts describe good practice and the relevant articles, but must be checked against your own case, with a professional if needed.

Updates. It reflects the state of knowledge in November 2026. Threats, standards and tools change: the version date matters.

Short on time? The ten measures to do first

1. Two-factor authentication on every provider account and on administrator accounts (3.3, 13.1).
2. Review every API route with three questions: who may call it, whose data does it touch, what does it return; with a test for every problem found (4.1, 4.2).
3. Secrets out of the code, in sealed variables, and rotated after any incident (7.1, 14).
4. Lockfiles, automatic dependency updates and a vulnerability check in CI (11).
5. Automated tests that run on every change and block the merge when they fail (12).
6. Encrypted backups, off the main platform, undeletable, with a restore test (10).
7. HTTPS, secure cookies and security headers (3.5, 5.5, 6.1).
8. Rate limits and responses that do not reveal whether an account exists (3.6).
9. Little personal data, including in logs (8.1, 9.1).

10. A written incident procedure (14).

Contents

1. Principles
2. Way of working
3. Authentication
4. Authorisation
5. Input, output and APIs
6. Encryption and key management
7. Secrets and configuration
8. Privacy and GDPR
9. Logging, audit and alerts
10. Backup and recovery
11. Software supply chain
12. Testing, CI and release
13. Infrastructure, domain and providers
14. Incident response
15. Mapping to OWASP ASVS 5.0
16. Checklist
17. References
18. Versions, licence and contact

1. Principles

Five ideas drive every measure.

- **Start from the risks.** Before choosing controls, ask who might attack, what they want and how they would get in. It is the same method used in functional safety for machinery: risk assessment, safeguards, proof that they work.
- **Defence in depth.** No measure is perfect. Several independent layers mean that the failure of one is not enough for an attacker.
- **Least privilege.** Every user, service, key and credential can do only what it needs, and nothing more.
- **Secure by default, fail closed.** If a setting is missing or a check errors, the system refuses instead of falling back to permissive behaviour.
- **Evidence, not reassurance.** Every measure must be verifiable: a test, a log, a command that shows the state. "It should work" is not evidence.

References: OWASP Top 10:2025 A06 *Insecure Design*; OWASP ASVS 5.0; *privacy by design* principles of Art. 25 GDPR.

2. Way of working

2.1 A written threat model

- **What:** a short document listing the assets to protect (personal data, keys, admin accounts), the actors (curious user, external attacker, compromised provider, insider) and the main attack paths. Update it when the architecture changes.
- **Why:** without a map of the risks you protect the easy things instead of the important ones.
- **Prevents:** design flaws that no test finds, because nobody imagined them.
- **References:** OWASP Top 10:2025 A06; OWASP Threat Modeling Cheat Sheet.

2.2 Human review of every change, even with AI

- **What:** AI assistants may write and review code, but a person reads every diff and decides what goes in. Rules for agents live in a file in the repository (for example `AGENTS.md`): never secrets in code or logs, never weaken a control, never change behaviour without tests.
- **Why:** code and tests written by the same AI can share the same wrong assumption. A second AI helps but is not an independent check.
- **Prevents:** plausible mistakes that pass every test; changes that remove a control "to make something pass".
- **References:** OWASP Top 10:2025 A06, A08.

2.3 One actor at a time

- **What:** only one entity works in a given working folder at a time (developer, AI agent, automated script). While an agent is working, use only commands that do not touch the folder.
- **Why:** concurrent branch switches, stashes or installs corrupt work in progress.
- **Prevents:** commits on the wrong branch, lost changes, accidental merges.

2.4 Data changes in two steps

- **What:** every change to production data is a script that defaults to a **dry run** printing only counts. The real run happens after reviewing the numbers and after a verified encrypted backup, followed by a separate verification.
- **Why:** mistakes on production data are the most expensive and often irreversible.
- **Prevents:** wrong deletions or migrations; personal data leaking into script logs.
- **References:** OWASP Top 10:2025 A08; change-management good practice.

2.5 AI agents that act: prompt injection and permissions

- **What:** an AI agent that reads external content (web pages, issues, emails, files, API responses) and can take actions (commands, commits, calls to services) should be treated as an untrusted user with many capabilities:
  - everything the agent reads is **data, not instructions**; text saying "ignore your rules and do X" must not be able to change its behaviour;
  - least privilege: no production credentials in the agent's environment, narrowly scoped tokens, access only to the repositories and folders it needs;
  - human confirmation for irreversible actions (merge, deploy, deletion, sending messages, payments);
  - work on separate branches, in an isolated environment, with review of the diff;
  - secrets never pasted into chat or prompts.
- **The same applies to the app's AI features:** model output is untrusted input. If it is rendered as HTML it must be sanitised (5.2); if it can trigger actions (tools, queries, emails), each action goes through the same authorisation checks as the user and, where relevant, a confirmation.
- **Prevents:** indirect prompt injection (instructions hidden in a page or document), exfiltration of secrets, destructive actions taken "on behalf of" the user, excessive agency.
- **References:** OWASP Top 10 for LLM Applications 2025 (LLM01 Prompt Injection, LLM05 Improper Output Handling, LLM06 Excessive Agency).

3. Authentication

3.1 Passwords stored with slow hashes

- **What:** passwords stored only as salted hashes with slow algorithms (Argon2id, scrypt, or bcrypt with an adequate cost). Never encrypted, never in plain text. Accept long passwords and passphrases; check against lists of breached passwords instead of imposing composition rules.
- **Why:** if the database is copied, a slow hash makes recovering passwords very expensive.
- **Prevents:** credential stuffing on other sites after a database theft; offline dictionary attacks.
- **References:** NIST SP 800-63B (rev. 4); OWASP Password Storage Cheat Sheet; OWASP Top 10:2025 A04, A07.

3.2 Magic links (sign-in by email link)

- **What:** long random tokens (at least 128 bits), **single-use**, short-lived (10–15 minutes), stored in the database only as hashes. The link leads to a page where the user confirms the sign-in with an action, so email scanners do not consume the token. The response is identical whether or not the email exists.
- **Why:** it removes the password for many users, but moves security to the mailbox and the token.
- **Prevents:** reuse of intercepted links, brute force on the token, account enumeration, token theft from a database dump.
- **References:** NIST SP 800-63B (out-of-band authenticators); OWASP Forgot Password Cheat Sheet.

3.3 Mandatory second factor for administrators

- **What:** TOTP (RFC 6238, the scheme used by Google Authenticator, Authy, Apple Passwords) mandatory for privileged accounts. Implemented with the platform's standard cryptography and verified against the official test vectors. Codes cannot be reused (the last used time step is stored). The TOTP secret is encrypted like other personal data. Single-use recovery codes stored only as slow hashes. An emergency procedure for a lost phone, which only the owner can run and which is logged.
- **Why:** the admin password alone is a single point of failure.
- **Prevents:** takeover of the admin account with a stolen or guessed password; replay of an intercepted code.
- **References:** RFC 6238; NIST SP 800-63B; OWASP Multifactor Authentication Cheat Sheet; OWASP Top 10:2025 A07.

3.4 Passkeys

- **What:** WebAuthn/passkeys as the main method or as a phishing-resistant second factor.
- **Why:** the private key never leaves the device and the signature is bound to the domain.
- **Prevents:** real-time phishing, including against TOTP.
- **References:** NIST SP 800-63B (phishing-resistant authenticators); W3C WebAuthn.

3.5 Sessions and cookies

- **What:** session tokens in `HttpOnly`, `Secure`, `SameSite` cookies. Short-lived access tokens with a refresh token rotated on every use; revocation on logout and password change. A clear idle timeout. If signed tokens (JWT) are used, always verify signature, expected algorithm, expiry and audience; never accept `alg: none`.
- **Why:** page scripts must not be able to read the session; a stolen token should be worth little, for a short time.
- **Prevents:** session theft via XSS, CSRF, reuse of stolen tokens, forged tokens.
- **References:** OWASP Session Management Cheat Sheet; OWASP JSON Web Token Cheat Sheet; OWASP Top 10:2025 A07.

3.6 Neutral responses and rate limits

- **What:** sign-in, registration and password recovery respond in the same way whether or not the account exists. Rate limits per IP and per account on authentication, email verification and expensive features (AI, export).
- **Why:** automation (now often AI-assisted) tries thousands of combinations.
- **Prevents:** user enumeration, brute force, credential stuffing, abuse of paid features.
- **References:** OWASP Authentication Cheat Sheet; OWASP Top 10:2025 A07; OWASP API Security Top 10 (API4 Unrestricted Resource Consumption).

4. Authorisation

4.1 Route-by-route review with three questions

- **What:** for every API route: **who may call it? whose data does it touch? what does it return?** All routes, not just the suspicious ones. Deny by default: a route without an explicit rule is not reachable.
- **Why:** broken access control has been the leading cause of web app incidents for years.
- **Prevents:** IDOR (changing an ID in the request and seeing someone else's data), privilege escalation, admin functions reachable by normal users.
- **References:** OWASP Top 10:2025 A01; OWASP API Security Top 10 (API1, API5); OWASP Authorization Cheat Sheet.

4.2 A test that fails on the old code

- **What:** every security fix comes with a test that fails before the fix and passes after it.
- **Why:** it proves the problem existed, that it is closed, and stops it from coming back.
- **Prevents:** silent regressions; checks that "look" right but do not work (for example a function referenced instead of called, which makes a check always true).

4.3 Minimal responses (DTOs)

- **What:** a single function builds every view of "another user" with only the fields needed (for example ID and display name). Never return the database object as is.
- **Why:** extra fields end up in responses as the model grows.
- **Prevents:** exposure of email addresses, roles, hashes, ciphertext or internal fields (OWASP API3 *Broken Object Property Level Authorization*).

5. Input, output and APIs

5.1 Validation and safe queries

- **What:** validate type, format and length of every input on the server, with a schema. Parameterised queries; for document databases, reject keys starting with `$` or containing `.` in inputs, and never pass request objects straight into filters.
- **Prevents:** SQL/NoSQL injection, operator injection, prototype pollution, huge payloads.
- **References:** OWASP Top 10:2025 A05; OWASP Input Validation Cheat Sheet.

5.2 XSS: safe output, including Markdown and AI-generated text

- **What:**
  - let the framework escape output (React, Next.js and modern templates do so by default);
  - avoid `dangerouslySetInnerHTML`, `innerHTML`, `v-html` and the like; where unavoidable, pass the content through an allowlist sanitiser (for example DOMPurify or `rehype-sanitize`);
  - for Markdown written by users or generated by AI: raw HTML disabled, only allowlisted elements and attributes, links only `https:`, `http:` and `mailto:` (never `javascript:` or `data:`), external links with `rel="noopener noreferrer"`;
  - never build URLs, attributes or JavaScript code by concatenating input;
  - a Content Security Policy as a second layer (scripts with nonces or hashes, no `unsafe-inline`), plus Trusted Types where possible.
- **Why:** an injected script runs with the user's permissions: it reads the data on the page and acts in their name.
- **Prevents:** stored XSS (journal, comments, profiles), reflected XSS, data theft from the page, unwanted actions.
- **References:** OWASP Top 10:2025 A05; OWASP Cross Site Scripting Prevention Cheat Sheet; OWASP ASVS 5.0 V1, V3.

5.3 CSRF protection and strict CORS

- **What:** for state-changing requests, check `Origin` / `Referer` and use `SameSite`. CORS limited to the exact origins of your frontend, never `*` with credentials.
- **Prevents:** actions performed in the user's name by third-party sites.
- **References:** OWASP CSRF Prevention Cheat Sheet.

5.4 Request body limits and allowed encodings

- **What:** a maximum body size; accept only the expected compression encodings (for example `identity`, `gzip`, `deflate`) and answer `415` to the others before parsers decompress them.
- **Why:** decompressors are complex code and occasionally vulnerable: in 2026, for example, a vulnerability in Node.js Brotli decompression was reachable through request bodies.
- **Prevents:** decompression bombs, exploitation of decoder bugs, memory exhaustion.

5.5 Security headers

- **What:** `Strict-Transport-Security`, `Content-Security-Policy` (at least `frame-ancestors`, `object-src 'none'`, `base-uri 'self'`), `X-Content-Type-Options: nosniff`, `Referrer-Policy`, `Permissions-Policy`. Check them with a free scanner after every infrastructure change.
- **Prevents:** XSS, clickjacking, downgrade to HTTP, URL leaks via the referrer.
- **References:** OWASP HTTP Headers Cheat Sheet; OWASP Top 10:2025 A02.

5.6 SSRF: when the server fetches a URL

- **What:** if the app fetches something from a user-supplied URL (link previews, imports, remote images, outgoing webhooks):
  - allowlist domains or schemes (`https:` only);
  - resolve the name and reject private, loopback, link-local and cloud-metadata addresses (for example `169.254.169.254`), also after redirects;
  - short timeouts, a maximum response size, no automatic redirects or re-checked redirects;
  - if possible, make these requests from a separate service with no access to the internal network.
- **Prevents:** access to internal services, credential theft from cloud metadata, internal network scanning.
- **References:** OWASP Server Side Request Forgery Prevention Cheat Sheet; OWASP API Security Top 10 (API7); OWASP Top 10:2025 A01.

5.7 File uploads

- **What:**
  - size and count limits;
  - type checked on the content (file signature), not on the extension or the declared `Content-Type`;
  - file name generated by the server, never a user-supplied path;
  - files stored outside the public folder or in object storage, served with `Content-Disposition: attachment` and `nosniff`, ideally from a separate domain;
  - images re-encoded and metadata (EXIF, GPS location) stripped;
  - no user-uploaded SVG or HTML served as pages;
  - antivirus scanning if files are shared with other users.
- **Prevents:** code execution on the server, XSS through files, path traversal, disk exhaustion, location leaks through photos.
- **References:** OWASP File Upload Cheat Sheet; OWASP ASVS 5.0 V5.

5.8 Verified incoming webhooks

- **What:** every webhook (payments, email, repository) verifies the sender's signature with an HMAC over the **raw** request body, using a constant-time comparison; rejects messages with timestamps that are too old; is idempotent (the same event received twice does not have two effects); responds quickly and does the work in a queue.
- **Prevents:** forged events (for example an invented "payment succeeded"), replay, double processing.
- **References:** provider documentation; OWASP ASVS 5.0 V4.

5.9 Signed links for actions without sign-in

- **What:** links such as "unsubscribe" or "download export" signed with HMAC, with an expiry, bound to the user and the action.
- **Prevents:** actions on other accounts by changing a parameter; indefinite reuse of links.

5.10 Errors without details

- **What:** generic messages for the user; in logs, safe error codes, without external messages, stack traces or personal data. Operational scripts print only codes from a fixed list.
- **Prevents:** leaks about internal structure, queries, secrets.
- **References:** OWASP Top 10:2025 A10 *Mishandling of Exceptional Conditions*.

5.11 No debug pages in production

- **What:** remove or block test pages, diagnostics and detailed status endpoints; the public health check returns only "ok/not ok".
- **Prevents:** reconnaissance by an attacker, exposure of versions and configuration.
- **References:** OWASP Top 10:2025 A02.

6. Encryption and key management

6.1 TLS everywhere

- **What:** mandatory HTTPS with HSTS; encrypted connections to the database and external services.
- **Prevents:** interception and tampering of traffic.

6.2 Application-level encryption of sensitive fields

- **What:** the application encrypts personal data and private content (email, name, user-written text, TOTP secrets) **before** storing it, with an authenticated algorithm (AES-256-GCM) and a random IV per value. An authentication failure is an explicit error, never an "empty" value.
- **Why:** the provider's "at rest" encryption protects the disks, not against someone accessing the database with valid credentials or a stolen backup.
- **Prevents:** exposure of data if the database or backups are stolen, or through provider access.
- **References:** OWASP Cryptographic Storage Cheat Sheet; OWASP Top 10:2025 A04; Art. 32 GDPR (encryption as a technical measure).

6.3 Envelope encryption

- **What:** two layers of keys. **DEKs** (data encryption keys) encrypt the fields; the **KEK** (key encryption key) encrypts the DEKs. Encrypted DEKs live in a separate *keyring*; the KEK lives outside the database (sealed variable, KMS or HSM). The app reads the keyring with a **read-only** user.
- **Why:** rotating the KEK is quick (a few keys are re-encrypted, not the data); someone who steals only the database does not have the KEK.
- **Prevents:** decryption of data with the database alone; replacement or deletion of keys by a compromised app.
- **Honest limit:** if the KEK, the database URI and the keyring URI live in the same place, whoever steals them all can decrypt. To raise the bar: KEK in an external KMS with logging and revocation, network allowlist on the database.
- **References:** NIST SP 800-57 (key management); OWASP Key Management Cheat Sheet.

6.4 Versioned keys, and rotation actually performed

- **What:** every encrypted value carries its key version (for example `k2:...`). Rotation creates a new key, activates it, restarts the instances, re-encrypts data in a resumable and verified way (dry-run → create → apply → verify). The old key is archived while backups that need it still exist, then retired. Rehearse the procedure, then run it: a rotation never performed is not a control.
- **Why:** after a suspected compromise, or periodically, you must be able to change keys without stopping the service.
- **Prevents:** permanent dependence on a possibly exposed key; data loss from an interrupted rotation.
- **References:** NIST SP 800-57 (crypto-periods and key retirement).

6.5 Blind indexes to search encrypted data

- **What:** to find a user by email without decrypting everything, store next to the encrypted field a **keyed HMAC** of the normalised value. Never a plain hash (unkeyed SHA-256). A unique index on the HMAC.
- **Why:** a plain hash can be recomputed by anyone: with a leaked list of emails you can find out who is registered.
- **Prevents:** re-identification of users from a database dump; dictionary attacks on email addresses.
- **Limit:** equal values give the same index; the HMAC key must be protected like the encryption keys. Migrate in several steps, with a read-only verification before removing the old format.

6.6 No fallback secrets in code

- **What:** in production the app refuses to start if a required secret is missing or has a placeholder value.
- **Prevents:** tokens signed with publicly known values (`change-me-in-production`), forgeable sessions.
- **References:** OWASP Top 10:2025 A02.

7. Secrets and configuration

7.1 Secrets only in the right place

- **What:** secrets in a password manager and in the platform's **sealed** (unreadable) variables or a secrets manager. Never in the repository, logs, command arguments, shell history or chat. Enter them in the terminal with hidden prompts (`read -rs`) and clear the clipboard afterwards.
- **Prevents:** leaks from repositories, logs, screenshots, histories, visible processes.
- **References:** OWASP Secrets Management Cheat Sheet; OWASP Top 10:2025 A02, A04.

7.2 Scoped, limited API keys

- **What:** one key per service and per environment, in the right project at the provider, with narrow permissions and **spending limits**. An inventory of keys (where it lives, what it is for, when it was rotated). Revoke unused keys. After every rotation, check that the new key is in the right project and that the old one no longer works.
- **Prevents:** expensive abuse of stolen keys (AI keys are a frequent target); forgotten keys still active on a laptop.

7.3 Temporary credentials for rare operations

- **What:** for privileged operations (creating a key, migrating data) use temporary users with automatic expiry and delete them right afterwards.
- **Prevents:** powerful credentials left active for months.

7.4 Separate environments

- **What:** development, preview and production with different secrets. Local `.env` files with `600` permissions, ignored by git, without production keys.
- **Prevents:** a test environment exposing production; production keys on laptops.

7.5 Secret scanning

- **What:** an automatic check that looks for keys and passwords in commits, before they reach the remote repository and again in CI. On GitHub: secret scanning with *push protection* (free on public repositories); instead or as well, open-source tools such as Gitleaks or TruffleHog in a pre-commit hook and in CI. If a secret has landed in a commit it must be **revoked and rotated**: deleting it from history is not enough.
- **Prevents:** keys published by mistake and harvested within minutes by bots that scan repositories.
- **References:** OWASP Secrets Management Cheat Sheet; OWASP Top 10:2025 A02.

8. Privacy and GDPR

8.1 Data minimisation

- **What:** collect only what you need. Example: usage statistics without user ID or IP, with a short-lived random visit identifier.
- **Why:** data you do not have does not need protecting, notifying or justifying.
- **References:** Art. 5(1)(c) and Art. 25 GDPR.

8.2 Automatic expiry

- **What:** every type of data has a defined retention period enforced by the database (TTL indexes or jobs). Typical examples: statistics 13 months, technical data 90 days, incomplete drafts 180 days of inactivity, security logs up to a year, backups replaced within 6 months.
- **References:** Art. 5(1)(e) GDPR (storage limitation); for analytics cookies, the Italian data protection authority's guidelines.

8.3 Complete deletion and portability, with tests

- **What:** account deletion removes data in every collection; a test fills every collection and checks that nothing remains. Data export in a readable format (JSON and CSV) verified by tests, for **every** format offered.
- **Prevents:** forgotten orphan data; unmet data-subject requests; an export broken for months without anyone noticing.
- **References:** Art. 17 and 20 GDPR.

8.4 A privacy policy written from the code

- **What:** the privacy policy describes what the code actually does (data, purposes, providers, retention); it is updated when the code changes. No fake consent switches.
- **References:** Art. 12–14 GDPR.

8.5 Cookies

- **What:** only technical cookies where possible. A first-party, anonymised analytics cookie used only in aggregate and not combined with other data can be treated as technical under the Italian authority's guidelines; otherwise consent is required.
- **References:** ePrivacy Directive; Garante cookie guidelines (2021).

8.6 Record of processing and processors

- **What:** a record listing processing activities, legal bases, data, recipients, processors with data location, transfers outside the EU and DPA status. A new provider goes into the record and the privacy policy **before** receiving any data.
- **References:** Art. 30 GDPR.

8.7 A DPA with every provider that processes personal data

- **What:** a data processing agreement (Art. 28) with every processor: hosting, database, email, AI, backups. It is often part of the terms of service: download it and archive it with the date. For providers outside the EU, check the transfer basis (Data Privacy Framework or standard contractual clauses).
- **References:** Art. 28 and Chapter V GDPR.

8.8 Breach register and notification

- **What:** every incident, even one that is not notified, is documented with facts, risk assessment and measures. If there is a risk to data subjects: notify the authority within 72 hours; if the risk is high, inform the data subjects.
- **References:** Art. 33 (including paragraph 5) and Art. 34 GDPR.

8.9 Responsible AI

- **What:** each AI feature receives only the data it needs (no names, emails or IDs when not required); generated content is labelled as such; the provider does not train on the data sent; the provider key has minimal permissions and a spending limit; an assessment against the AI Act.
- **Prevents:** data leaks to the AI provider; prompt injection with effects on data (see 2.5); runaway costs.
- **References:** Regulation (EU) 2024/1689 (AI Act); OWASP Top 10 for LLM Applications.

8.10 Impact assessment when needed

- **What:** for high-risk processing (sensitive data at scale, minors, profiling with significant effects) a DPIA before starting.
- **References:** Art. 35 GDPR.

9. Logging, audit and alerts

9.1 Useful but minimal audit logs

- **What:** record relevant actions (sign-ins, role changes, exports, deletions, key operations) with IDs and codes, not with email addresses or content. Redact secrets and addresses; where a reference is needed, use IDs or HMACs, with an expiry.
- **Prevents:** logs that become a data leak themselves; incidents that cannot be reconstructed.
- **References:** OWASP Logging Cheat Sheet; OWASP Top 10:2025 A09.

9.2 Alerts

- **What:** automatic notifications for: missing or failed backups, unusual database access, spending spikes on pay-per-use services, rising errors, admin sign-ins. A daily "all good" message makes the absence of news visible too.
- **Why:** an attack discovered after weeks does far more damage than one discovered within an hour.
- **References:** OWASP Top 10:2025 A09 *Security Logging & Alerting Failures*.

10. Backup and recovery

10.1 Backups encrypted with a public key

- **What:** daily automatic backups of the database **and** the keyring, encrypted with a public key (for example `age`) before leaving the server. The private key lives only offline, in the password manager.
- **Why:** the backup service can encrypt but not decrypt: if it is compromised, the backups stay unreadable.

10.2 Immutable copies off the main platform

- **What:** a destination separate from the hosting provider (3-2-1 rule), with **Object Lock in Compliance mode** (nobody can delete for N days, not even the admin) and a key that can only add files.
- **Prevents:** ransomware and sabotage: someone who gets into the hosting and deletes the database cannot delete the backups too.

10.3 Retention consistent with the privacy policy

- **What:** automatic deletion of backups past the stated retention (for example 6 months), coordinated with the retirement of old keys.
- **References:** Art. 5(1)(e) GDPR.

10.4 Periodic restore test

- **What:** at least once a month, download a backup, verify its hash, decrypt it and load it into a throwaway **local** database, comparing counts. A backup that has never been restored is not a backup.
- **References:** CIS Controls v8.1, Control 11 (Data Recovery).

10.5 A backup before every major change

- **What:** an encrypted, verified backup before migrations, key rotations or large releases.

10.6 First backup in the real environment

- **What:** after enabling backups, run one real execution straight away with production credentials and network, check that the files arrive and the lock is active, then do the first restore test. Green tests are not enough.
- **Why:** tests usually run against a local database without encryption and a fake storage; production uses real TLS, DNS and permissions. A typical case: a *slim* container image without certificate-authority bundles, so the dump tool cannot verify the encrypted connection to the database.
- **Prevents:** believing you have backups that in fact fail every night.

11. Software supply chain

11.1 Exact versions and lockfiles

- **What:** lockfiles committed to the repository; reproducible installs (`npm ci`); exact versions for direct dependencies.
- **Prevents:** unwanted updates carrying malicious code.

11.2 A waiting period before adopting new releases

- **What:** do not install versions published less than a few days ago (for example 7), except for security fixes.
- **Why:** compromised packages are usually detected and pulled within a few days.
- **Prevents:** "popular package compromised" supply-chain attacks.

11.3 Install scripts disabled

- **What:** `--ignore-scripts` where possible, with explicit exceptions.
- **Prevents:** secret theft during `npm install`.

11.4 Automatic updates and audit

- **What:** Dependabot (or similar) with grouped weekly updates; security fixes immediately; major versions by hand. `npm audit` of production dependencies in CI, blocking on high/critical. Never run `npm audit fix --force` blindly: it can install incompatible major versions. The full flow is in 12.6.

11.5 Pinned CI actions, images and tools

- **What:** GitHub Actions pinned by commit hash; Docker images with an exact version (and digest where possible); downloaded tools verified with checksums. Update them through automatic PRs, not floating tags. CI tokens with minimal permissions (explicit `permissions:`, read-only by default).
- **Prevents:** actions or images replaced with malicious versions; non-reproducible builds; a compromised CI writing to the repository.
- **References:** OWASP Top 10:2025 A03 *Software Supply Chain Failures* and A08; SLSA; OpenSSF Scorecard.

11.6 An up-to-date runtime

- **What:** follow the runtime's security releases (Node.js, Python...), use an LTS version pinned identically in development, CI and production, and check the version actually running in production, not just the declared one.
- **Prevents:** remotely reachable runtime vulnerabilities, even with correct application code.

11.7 Dead configuration removed

- **What:** delete unused configuration files (old builders, test pages, scripts).
- **Prevents:** confusion about what really runs; forgotten configuration that comes back to life.

12. Testing, CI and release

12.1 Why tests are a security control

- **What:** treat automated tests as part of security, not just quality.
- **Why:** almost every measure in this guide is code, and code breaks when it changes. Tests do three things:
  - they **prove** that a control works (a non-owner gets a 403, deletion leaves no data behind, an expired token is rejected);
  - they **stop it from breaking** silently with the next change;
  - they **make it possible to update quickly**: without tests every security update is a gamble and tends to be postponed, which is exactly what an attacker hopes for.
- **Prevents:** security regressions, features broken for months without anyone noticing, vulnerable dependencies left in place for fear of breaking something.

12.2 Three levels of tests

- **Unit tests:** single functions, fast and numerous. For security: encryption and decryption (including rejection of tampered data), token signing and verification, input validation, normalisation for blind indexes, official TOTP test vectors.
- **Integration and API tests:** call the real routes against a test database. For security: every authorisation rule (owner, other user, anonymous, admin), responses without internal fields, rate limits, complete deletion, export in every format, rejection of malformed input.
- **User-interface (end-to-end) tests:** an automated browser (for example Playwright) walks through the main flows: registration, sign-in with a second factor, key pages, export, deletion. They find errors the others miss: a page that crashes because of a missing import, a button that calls the wrong route.
- **Proportions:** many unit tests, a good number of API tests, a few end-to-end tests on critical paths. End-to-end tests are slow and more fragile: keep them few and stable.
- **References:** OWASP Web Security Testing Guide; OWASP ASVS 5.0 (its requirements translate naturally into test cases).

12.3 Coverage: useful, but not the goal

- **What:** measure coverage (lines and branches executed by tests) and look above all at **where it is missing**. As a rough guide, broad backend coverage is a good sign, but it matters more that sensitive paths (authentication, authorisation, encryption, deletion, export, payments) are fully covered, including error cases. Every feature users rely on deserves at least one test.
- **Why:** the more code is covered, the more likely it is that a breaking change is stopped before the merge. But a line "covered" by a test with no meaningful assertions protects nothing, and the percentage can be inflated with useless tests.
- **Prevents:** the false comfort of a high number; gaps exactly where it matters.

12.4 Red → green tests for every fix

- **What:** every bug, and especially every security problem, is fixed like this: first a test that reproduces the problem and **fails**, then the fix that makes it **pass** (see 4.2). The change report compares the suite before and after, test by test and not just in totals.
- **Prevents:** fixes that do not fix; tests that cannot fail; regressions.

12.5 Blocking, stable CI

- **What:** on every pull request CI runs: the whole test suite, the build, the vulnerability check on production dependencies and the lint for certain errors. If a check fails, the merge is blocked (branch protection with *required status checks*). Flaky tests are fixed straight away: typical causes are memory exhaustion with too many parallel processes, concurrent downloads of the same resources, dependence on the clock or on test order.
- **Why:** a CI that is "sometimes red" teaches people to ignore red, and an ignored red is worth as much as no check at all. CI also runs on a different system from the developer's and finds differences that do not show up locally.
- **Prevents:** regressions in production; the habit of merging with failing checks.

12.6 When a vulnerability arrives: the flow

This is where tests and CI prove their worth. A typical flow:

1. **Alert.** Dependabot (or similar) reports a vulnerability in a dependency and opens a pull request with the fixed version.
2. **Assessment.** Read the advisory: is it a production or a development-only dependency? Is the vulnerable part reachable in the app? Severity drives urgency, not the decision to update.
3. **Automatic checks.** CI runs tests, build, audit and lint on the pull request.
4. **Green → merge.** If every check is green and the diff (lockfile included) looks reasonable, merge. The merge command comes **after** looking at the checks, not in the same block that pushes and opens the PR.
5. **Release and verification.** Deploy, then a short check of the main paths (sign-in, a key feature, an export) and a look at the logs.
6. **Rollback ready.** If something is wrong, go back to the previous version in seconds (12.10) and investigate calmly.
7. **Red → investigate.** If a check fails, do not force it: find out whether the update really breaks something or the test needs adjusting. Major versions remain manual decisions, on a dedicated branch.

With a good suite, a security update takes minutes. Without one it takes hours of manual testing, and that is why it gets postponed.

12.7 Lint and static analysis (SAST)

- **What:**
  - at least one **blocking** static check for errors that certainly break the code: undefined identifiers, imports that do not resolve, incorrect use of hooks; the full lint becomes blocking once errors reach zero, with a cap on warnings;
  - a code security analyser: CodeQL (free on public GitHub repositories) or Semgrep Community Edition (open source), with the rules for your language; findings are triaged like bugs, not silenced wholesale.
- **Why:** an error of this kind can break a page in production without the build noticing, and informational-only lint gets ignored even when it reports real bugs.
- **Prevents:** pages crashing in production; known dangerous code patterns (injection, `eval`, unsafe deserialisation, secrets in code).
- **References:** OWASP Source Code Analysis Tools; OWASP Top 10:2025 A05, A06.

12.8 Dynamic scanning (DAST)

- **What:** an automated scan of the running app from the outside, for example the OWASP ZAP *baseline scan*, against a test environment or, carefully and in passive mode, against production. On every major release or monthly.
- **Why:** it finds misconfigurations the code does not show: missing headers, cookies without attributes, forgotten pages, exposed versions.
- **Prevents:** configuration mistakes visible to anyone; arriving at a penetration test with trivial problems still open.
- **References:** OWASP ZAP; OWASP Web Security Testing Guide.

12.9 Controlled review and merge

- **What:** every change goes through a PR with green checks; a person reads the diff (even when an AI wrote it); the merge happens only after looking at the checks, not automatically in the same command. The main branch is protected: no direct pushes, no merges with red checks.
- **Prevents:** unreviewed code in production; accidental merges.

12.10 Immediate rollback

- **What:** know how to go back to the previous version in seconds (for example the platform's *Instant Rollback*) and have tried it. For database changes, keep compatibility with the previous version of the code or have a reverse procedure.
- **Prevents:** hours of downtime while looking for a fix under pressure.

13. Infrastructure, domain and providers

13.1 Two-factor authentication on every provider account

- **What:** hosting, database, DNS and registrar, email, repository, AI, backups: all with 2FA, preferably passkeys or hardware keys. Alerts on important changes.
- **Prevents:** takeover of the whole infrastructure with a single password.

13.2 Least-privilege database users

- **What:** an app user with permissions only on its own database; a read-only keyring user; a read-only backup user; no administrative user used by the app.

13.3 Restricted network access to the database

- **What:** allowlist the services' egress IPs (requires a static egress IP), or use a private network. If that is not possible, compensate with strong, rotated and monitored credentials.
- **Prevents:** use of stolen credentials from anywhere on the internet.

13.4 Data in the European Union

- **What:** EU regions for database, functions and backups; record where data lives and which transfers remain (support, edge network).
- **References:** Chapter V GDPR.

13.5 Non-root containers and minimal images

- **What:** processes without root privileges; up-to-date *slim* images; no unnecessary tools in production. *Slim* images often lack certificate-authority bundles (`ca-certificates`): Node.js ships its own, but tools written in other languages (for example `mongodump`) need them to verify TLS connections. Install them when needed and check at build time that the file exists.

13.6 Domain and DNS

- **What:**
  - a registrar with 2FA, *registrar lock* enabled and auto-renewal;
  - CAA records stating which authorities may issue certificates for the domain;
  - DNSSEC where the registrar and DNS provider support it;
  - no records pointing to decommissioned services (an "orphan" subdomain can be taken over by someone else);
  - monitoring of certificates issued for the domain through Certificate Transparency logs.
- **Prevents:** domain theft, certificates issued to third parties, subdomain takeover, phishing from your own domain.
- **References:** RFC 8659 (CAA); RFC 9364 (DNSSEC); Certificate Transparency (RFC 9162).

13.7 Domain email: SPF, DKIM, DMARC

- **What:** SPF listing who may send on behalf of the domain; DKIM to sign messages; DMARC, first in monitoring mode (`p=none` with reports) and then `quarantine` or `reject`. For domains that send no email: SPF `v=spf1 -all` and DMARC `p=reject`.
- **Why:** transactional emails (magic links, password resets) are an authentication channel; if the domain can be spoofed, phishing becomes credible. Large mailbox providers already require these settings from bulk senders.
- **Prevents:** fake emails in the app's name; magic links and notifications ending up in spam.
- **References:** RFC 7208 (SPF); RFC 6376 (DKIM); RFC 7489 (DMARC).

13.8 security.txt and vulnerability reporting

- **What:** a `/.well-known/security.txt` file with at least `Contact` and `Expires` (both required), plus `Preferred-Languages` and a link to a short disclosure policy: how to report, what to expect, a commitment not to act against good-faith reporters.
- **Why:** whoever finds a problem should know whom to tell, instead of publishing it or giving up.
- **References:** RFC 9116; ISO/IEC 29147.

14. Incident response

When a provider or a component reports a possible incident:

1. **Assume compromise** of everything the provider could see (environment variables, keys, tokens).
2. **Rotate every secret**, starting with those that give access to data (database, KEK, session signing), then external service keys. Check where each key was rotated and that the old ones are revoked.
3. **Review the logs** of every service from the start of the exposure window.
4. **Document** in the breach register: facts, risk assessment, decision on notification (Art. 33).
5. **Strengthen** what the incident showed to be weak (2FA, sealed variables, key separation).
6. **Write a post-mortem**: what happened, what worked, what changes.

References: NIST SP 800-61 (incident handling); Art. 33–34 GDPR.

15. Mapping to OWASP ASVS 5.0

OWASP ASVS is the reference standard for verifying the security of a web application. The table shows where this guide covers the topics of each chapter. The mapping is indicative: claiming an ASVS level requires requirement-by-requirement verification and, normally, an independent assessment.

ASVS 5.0 chapter	Guide sections
V1 Encoding and Sanitization	5.1, 5.2
V2 Validation and Business Logic	5.1, 3.6
V3 Web Frontend Security	3.5, 5.2, 5.3, 5.5
V4 API and Web Service	5.3, 5.4, 5.8
V5 File Handling	5.7
V6 Authentication	3.1–3.4, 3.6
V7 Session Management	3.5
V8 Authorization	4
V9 Self-contained Tokens	3.5, 5.9
V10 OAuth and OIDC	Out of scope (use of external providers only)
V11 Cryptography	6
V12 Secure Communication	6.1, 13.6, 13.7
V13 Configuration	5.11, 6.6, 7, 13
V14 Data Protection	6.2, 8
V15 Secure Coding and Architecture	2, 11, 12
V16 Security Logging and Error Handling	5.10, 9
V17 WebRTC	Out of scope

16. Checklist

Method

☐ Written, up-to-date threat model

☐ Rules for AI agents in the repository; human review of every diff

☐ One actor at a time in the working folder

☐ Production scripts with dry run, counts only, backup first

☐ AI agents with least privilege, no production credentials, human confirmation for irreversible actions

Authentication

☐ Passwords with Argon2id/scrypt/bcrypt, checked against breached passwords

☐ Single-use magic links, 10–15 minutes, stored as hashes, neutral response

☐ Mandatory TOTP for admins, anti-replay, hashed recovery codes, logged emergency procedure

☐ HttpOnly, Secure, SameSite cookies; short access tokens, rotated refresh; JWTs checked for signature, algorithm and expiry

☐ Responses that do not reveal whether accounts exist; rate limits

☐ Passkeys (when possible)

Authorisation

☐ Every route reviewed with the three questions; deny by default

☐ A red→green test for every fix

☐ Minimal DTOs for other users' data

Input, output and APIs

☐ Server-side schema validation, protection against injection and operator injection

☐ No unsanitised HTML; Markdown and AI output through an allowlist; links only http/https/mailto

☐ CSRF (Origin + SameSite), CORS limited to your own origins

☐ Body limit and allowed encodings only (415 for the others)

☐ Security headers (HSTS, CSP, nosniff…) checked with a scanner

☐ Server-side URL fetching: allowlist, no internal or metadata addresses

☐ Uploads: limits, type checked on content, generated names, metadata stripped, files served as attachments

☐ Webhooks: signature over the raw body, timestamp, idempotency

☐ Signed, expiring links for actions without sign-in

☐ Generic errors; no debug pages in production

Encryption and keys

- ☐ TLS everywhere
- ☐ Sensitive fields encrypted in the app with AES-256-GCM
- ☐ Envelope encryption: KEK outside the database, read-only keyring for the app
- ☐ Versioned keys; rotation rehearsed and performed; retirement of old keys planned
- ☐ Search on encrypted data only through keyed HMACs
- ☐ Startup refused if a secret is missing

Secrets

- ☐ Secrets only in a password manager and sealed variables; never in repo, logs, argv, chat
- ☐ API keys per project/environment, narrow permissions and spending limits; inventory; rotations verified
- ☐ Temporary users for privileged operations
- ☐ Separate environments; local `.env` files without production keys
- ☐ Secret scanning before push and in CI; exposed secret = revoke and rotate

Privacy and GDPR

- ☐ Minimisation (analytics without identity)
- ☐ Automatic expiry for every type of data
- ☐ Complete deletion and export tested, in every format
- ☐ Privacy policy written from the code; technical cookies only, or consent
- ☐ Record of processing and processors; new provider recorded before receiving data
- ☐ DPA archived for every provider; non-EU transfers checked
- ☐ Breach register; 72-hour notification procedure
- ☐ AI: minimal data, labels, no training, spending limit; AI Act assessed
- ☐ DPIA if processing is high-risk

Logging and alerts

- ☐ Minimal audit logs, with IDs and expiry; secrets and emails redacted
- ☐ Alerts on backups, unusual access, spending, errors

Backups

- ☐ Daily backups encrypted with a public key, private key offline
- ☐ Immutable copies (Object Lock Compliance) off the hosting, write-only key
- ☐ Deletion past the stated retention
- ☐ First backup verified in the real environment, not only in tests
- ☐ Monthly restore test on a local database

Supply chain

- ☐ Lockfiles, `npm ci`, exact versions
- ☐ Minimum age for new releases; install scripts disabled
- ☐ Grouped Dependabot; CI audit blocking on high/critical; never `audit fix --force` blindly
- ☐ Actions pinned by hash, images and tools pinned with checksums; least-privilege CI tokens
- ☐ Same LTS runtime everywhere, verified in production
- ☐ Dead configuration removed

Testing, CI and release

- ☐ Unit tests on security functions (encryption, tokens, validation)
- ☐ API tests for every authorisation rule, deletion and export
- ☐ End-to-end tests of the main paths
- ☐ Coverage measured; sensitive paths fully covered; every feature in use has at least one test
- ☐ Blocking, stable suite; flaky tests fixed straight away
- ☐ Vulnerability flow: automatic PR → green checks → merge → verification → rollback ready
- ☐ Lint for certain errors blocking; SAST enabled
- ☐ Periodic DAST scan
- ☐ Main branch protected; merge only after looking at the checks
- ☐ Rollback known and rehearsed

Infrastructure, domain and providers

- ☐ 2FA on every provider, registrar included
- ☐ Least-privilege database users (app, keyring, backup)
- ☐ Restricted (or compensated) network access to the database
- ☐ Data in the EU; remaining transfers documented
- ☐ Non-root containers, minimal images; `ca-certificates` present if non-Node tools open TLS connections
- ☐ Registrar lock, CAA, no orphan subdomains; DNSSEC where possible
- ☐ SPF, DKIM, DMARC (up to `reject`)
- ☐ `security.txt` with contact and expiry

Incidents

- ☐ Written procedure: assume, rotate, review, document, strengthen, post-mortem

17. References

- OWASP Top 10:2025 — https://owasp.org/Top10/2025/
- OWASP Application Security Verification Standard (ASVS) 5.0 — https://owasp.org/www-project-application-security-verification-standard/
- OWASP API Security Top 10 — https://owasp.org/API-Security/
- OWASP Cheat Sheet Series (Authentication, Session Management, Password Storage, Cryptographic Storage, Key Management, Secrets Management, Logging, CSRF, XSS Prevention, SSRF Prevention, File Upload, HTTP Headers, JSON Web Token) — https://cheatsheetseries.owasp.org/
- OWASP Web Security Testing Guide — https://owasp.org/www-project-web-security-testing-guide/
- OWASP ZAP — https://www.zaproxy.org/
- OWASP Top 10 for LLM Applications 2025 — https://genai.owasp.org/
- NIST SP 800-63B (rev. 4), Digital Identity Guidelines: Authentication — https://pages.nist.gov/800-63-4/
- NIST SP 800-57, Recommendation for Key Management — https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final
- NIST SP 800-61, Incident Response — https://csrc.nist.gov/pubs/sp/800/61/r3/final
- RFC 6238, TOTP — https://www.rfc-editor.org/rfc/rfc6238
- RFC 9116, security.txt — https://www.rfc-editor.org/rfc/rfc9116
- RFC 7208 (SPF), RFC 6376 (DKIM), RFC 7489 (DMARC) — https://www.rfc-editor.org/
- RFC 8659, DNS CAA — https://www.rfc-editor.org/rfc/rfc8659
- W3C Web Authentication (WebAuthn) — https://www.w3.org/TR/webauthn/
- Regulation (EU) 2016/679 (GDPR) — https://eur-lex.europa.eu/eli/reg/2016/679/oj
- Regulation (EU) 2024/1689 (AI Act) — https://eur-lex.europa.eu/eli/reg/2024/1689/oj
- Garante privacy, Guidelines on cookies and other tracking tools (2021) — https://www.garanteprivacy.it/
- CIS Critical Security Controls v8.1 — https://www.cisecurity.org/controls
- SLSA (Supply-chain Levels for Software Artifacts) — https://slsa.dev/
- OpenSSF Scorecard — https://scorecard.dev/

18. Versions, licence and contact

Version	Date	Changes
1.0	November 2026	First version, derived from the IkigaiApp security review
1.2	17 November 2026	First backup in the real environment (10.6); certificate-authority bundles in *slim* images (13.5); matching checklist items
1.1	11 November 2026	Scope and limits; the ten priority measures; XSS and Markdown; SSRF; uploads; webhooks; secret scanning; AI agents and prompt injection; expanded section on testing, CI and vulnerability handling; SAST and DAST; domain, DNS and email; security.txt; mapping to ASVS 5.0; agent version

Licence. This text is licensed under Creative Commons Attribution 4.0 International (CC BY 4.0): you may copy, adapt and use it, including commercially, as long as you credit the author and the source. https://creativecommons.org/licenses/by/4.0/

Source and contact. Latest version, PDF and agent version: https://salvatorevivolo.org/en/guides/web-app-security. Corrections and suggestions are welcome through the contacts on the site.

Case study. How these measures were applied to a real app: https://salvatorevivolo.org/en/projects/ikigaiapp/security-case-study