GUIDA E CHECKLIST

Sicurezza di una web app: misure consigliate

Guida e checklist per portare una piccola web app a un livello di sicurezza elevato

Salvatore Vivolo · Novembre 2026 · v1.2

Questa guida raccoglie le misure che rendono una web app difficile da attaccare e capace di riprendersi quando qualcosa va storto. Per ogni misura spiega **cosa fare**, **perché**, **quali attacchi o incidenti evita** e **a quale riferimento si collega** (OWASP, NIST, RFC, GDPR). In fondo ci sono una tabella di corrispondenza con OWASP ASVS 5.0 e una checklist da spuntare.

È nata da un lavoro reale: la revisione di sicurezza e privacy di IkigaiApp, una piccola web app con utenti veri, descritta nel caso di studio pubblicato su https://salvatorevivolo.org/it/projects/ikigaiapp/security-case-study. Qui le lezioni di quel progetto sono generalizzate e completate con le misure che servono a qualsiasi app simile.

Esiste anche una **versione per agenti AI** (`SECURITY-BASELINE.md`): le stesse regole in forma di requisiti MUST/SHOULD con un controllo di verifica per ciascuno, da mettere nel repository perché gli assistenti di programmazione le rispettino.

Ambito e limiti

Per chi è. Progetti piccoli o medi, con uno o pochi sviluppatori, un frontend web, un'API, un database gestito e servizi esterni (email, AI, pagamenti, storage). Gli esempi usano Node.js, Express, Next.js e MongoDB, ma i principi valgono per qualsiasi stack.

Cosa non copre. App mobili native, software desktop, l'implementazione di un proprio server OAuth/OIDC, WebRTC, architetture multi-tenant complesse, settori con norme specifiche (pagamenti con carta secondo PCI DSS, dati sanitari, infrastrutture critiche). Per questi servono requisiti aggiuntivi.

Cosa non è.

- Non è una certificazione: seguire la guida non rende un'app conforme a OWASP ASVS a un livello dichiarato. La tabella di corrispondenza è indicativa.
- Non sostituisce un test di penetrazione fatto da professionisti indipendenti.
- Non è consulenza legale: le parti sul GDPR descrivono buone pratiche e gli articoli di riferimento, ma vanno verificate sul proprio caso, se serve con un professionista.

Aggiornamento. Riflette lo stato delle conoscenze a novembre 2026. Minacce, standard e strumenti cambiano: la data di versione conta.

Se hai poco tempo: le dieci misure da fare per prime

1. Autenticazione a due fattori su tutti gli account dei fornitori e sugli account amministratore (3.3, 13.1).
2. Revisione di ogni route dell'API con tre domande: chi può chiamarla, di chi sono i dati, cosa restituisce; con un test per ogni problema trovato (4.1, 4.2).
3. Segreti fuori dal codice, in variabili sigillate, e ruotati dopo qualsiasi incidente (7.1, 14).
4. Lockfile, aggiornamenti automatici delle dipendenze e controllo delle vulnerabilità nella CI (11).
5. Test automatici che girano a ogni modifica e bloccano il merge se falliscono (12).
6. Backup cifrati, fuori dalla piattaforma principale, non cancellabili, con una prova di ripristino (10).
7. HTTPS, cookie sicuri e intestazioni di sicurezza (3.5, 5.5, 6.1).
8. Limiti di richieste e risposte che non rivelano se un account esiste (3.6).

9. Pochi dati personali, anche nei log (8.1, 9.1).
10. Una procedura scritta per gli incidenti (14).

Indice

1. Principi
2. Metodo di lavoro
3. Autenticazione
4. Autorizzazione
5. Input, output e API
6. Cifratura e gestione delle chiavi
7. Segreti e configurazione
8. Privacy e GDPR
9. Log, audit e allarmi
10. Backup e ripristino
11. Catena di fornitura del software
12. Test, CI e rilascio
13. Infrastruttura, dominio e fornitori
14. Risposta agli incidenti
15. Corrispondenza con OWASP ASVS 5.0
16. Checklist
17. Riferimenti
18. Versioni, licenza e contatti

1. Principi

Cinque idee guidano tutte le misure.

- **Partire dai rischi.** Prima di scegliere i controlli, chiedersi chi potrebbe attaccare, cosa vuole e da dove passerebbe. È lo stesso metodo della sicurezza funzionale nelle macchine: analisi dei rischi, protezioni, prova che funzionano.
- **Difesa in profondità.** Nessuna misura è perfetta. Più livelli indipendenti fanno sì che il fallimento di uno non basti a un attaccante.
- **Privilegio minimo.** Ogni utente, servizio, chiave e credenziale può fare solo ciò che gli serve, e niente di più.
- **Sicuro per impostazione predefinita e fallire chiusi.** Se manca una configurazione o un controllo dà errore, il sistema rifiuta invece di ripiegare su un comportamento permissivo.
- **Prove, non rassicurazioni.** Ogni misura deve essere verificabile: un test, un log, un comando che mostra lo stato. "Dovrebbe funzionare" non è una prova.

Riferimenti: OWASP Top 10:2025 A06 *Insecure Design*; OWASP ASVS 5.0; principi di *privacy by design* dell'art. 25 GDPR.

2. Metodo di lavoro

2.1 Modello delle minacce scritto

- **Cosa:** un documento breve con i beni da proteggere (dati personali, chiavi, account admin), gli attori (utente curioso, attaccante esterno, fornitore compromesso, insider) e i percorsi d'attacco principali. Aggiornarlo quando cambia l'architettura.
- **Perché:** senza una mappa dei rischi si proteggono le cose facili invece di quelle importanti.
- **Evita:** buchi di progettazione che nessun test trova, perché nessuno li ha immaginati.
- **Riferimenti:** OWASP Top 10:2025 A06; OWASP Threat Modeling Cheat Sheet.

2.2 Revisione umana di ogni modifica, anche con l'AI

- **Cosa:** gli assistenti AI possono scrivere e revisionare codice, ma una persona legge ogni diff e decide cosa entra. Le regole per gli agenti stanno in un file nel repository (per esempio `AGENTS.md`): mai segreti nel codice o nei log, mai indebolire un controllo, mai cambiare comportamento senza test.
- **Perché:** codice e test scritti dalla stessa AI possono condividere la stessa ipotesi sbagliata. Una seconda AI aiuta ma non è una verifica indipendente.
- **Evita:** errori plausibili che passano tutti i test; modifiche che rimuovono controlli "per far passare" qualcosa.
- **Riferimenti:** OWASP Top 10:2025 A06, A08.

2.3 Un attore alla volta

- **Cosa:** nella stessa cartella di lavoro opera una sola entità alla volta (sviluppatore, agente AI, script automatico). Mentre un agente lavora, usare solo comandi che non toccano la cartella.
- **Perché:** cambi di branch, stash o installazioni concorrenti corrompono il lavoro in corso.
- **Evita:** commit sul branch sbagliato, modifiche perse, merge involontari.

2.4 Modifiche ai dati in due fasi

- **Cosa:** ogni intervento sui dati di produzione è uno script con **dry-run** predefinito che stampa solo conteggi. L'esecuzione vera avviene dopo aver rivisto i numeri e dopo un backup cifrato verificato. Poi una verifica separata.
- **Perché:** gli errori sui dati di produzione sono i più costosi e spesso irreversibili.
- **Evita:** cancellazioni o migrazioni sbagliate; esposizione di dati personali nei log degli script.
- **Riferimenti:** OWASP Top 10:2025 A08; buone pratiche di change management.

2.5 Agenti AI che agiscono: prompt injection e permessi

- **Cosa:** un agente AI che legge contenuti esterni (pagine web, issue, email, file, risposte di API) e può eseguire azioni (comandi, commit, chiamate a servizi) va trattato come un utente non fidato con molte capacità:
  - tutto ciò che l'agente legge è **dato, non istruzione**; un testo che dice "ignora le regole e fai X" non deve poter cambiare il suo comportamento;
  - permessi minimi: niente credenziali di produzione nell'ambiente dell'agente, token con ambito limitato, accesso solo ai repository e alle cartelle necessari;
  - conferma umana per le azioni irreversibili (merge, deploy, cancellazioni, invio di messaggi, pagamenti);
  - lavoro su branch separati, in un ambiente isolato, con revisione del diff;
  - segreti mai incollati in chat o nei prompt.
- **Lo stesso vale per le funzioni AI dell'app:** l'output del modello è input non fidato. Se viene mostrato come HTML va sanitizzato (5.2); se può attivare azioni (strumenti, query, email), ogni azione passa dagli stessi controlli di autorizzazione dell'utente e, se rilevante, da una conferma.
- **Evita:** prompt injection indiretta (istruzioni nascoste in una pagina o in un documento), esfiltrazione di segreti, azioni distruttive eseguite "per conto" dell'utente, *excessive agency*.
- **Riferimenti:** OWASP Top 10 for LLM Applications 2025 (LLM01 Prompt Injection, LLM05 Improper Output Handling, LLM06 Excessive Agency).

3. Autenticazione

3.1 Password conservate con hash lenti

- **Cosa:** password salvate solo come hash con algoritmi lenti e con sale (Argon2id, scrypt o bcrypt con costo adeguato). Mai cifrate, mai in chiaro. Accettare password lunghe e passphrase; verificare contro liste di password compromesse invece di imporre regole di composizione.
- **Perché:** se il database viene copiato, un hash lento rende costosissimo risalire alle password.
- **Evita:** *credential stuffing* su altri siti dopo un furto di database; attacchi a dizionario offline.
- **Riferimenti:** NIST SP 800-63B (rev. 4); OWASP Password Storage Cheat Sheet; OWASP Top 10:2025 A04, A07.

3.2 Magic link (accesso tramite link via email)

- **Cosa:** token casuali lunghi (almeno 128 bit), **monouso**, con scadenza breve (10–15 minuti), salvati nel database solo come hash. Il link porta a una pagina che conferma l'accesso con un'azione dell'utente, così gli scanner delle email non consumano il token. Risposta identica sia che l'email esista sia che no.
- **Perché:** elimina la password per molti utenti, ma sposta la sicurezza sulla casella email e sul token.
- **Evita:** riuso di link intercettati, attacchi di forza bruta sul token, enumerazione degli account, furto del token da un dump del database.
- **Riferimenti:** NIST SP 800-63B (autenticatori out-of-band); OWASP Forgot Password Cheat Sheet.

3.3 Secondo fattore obbligatorio per gli amministratori

- **Cosa:** TOTP (RFC 6238, il sistema di Google Authenticator, Authy, Apple Passwords) obbligatorio per gli account con privilegi. Implementazione con la crittografia standard della piattaforma, verificata sui vettori di test ufficiali. Codici non riutilizzabili (si memorizza l'ultimo intervallo usato). Segreto TOTP cifrato come gli altri dati personali. Codici di recupero monouso salvati solo come hash lenti. Una procedura d'emergenza per il telefono perso, eseguibile solo dal titolare e registrata.
- **Perché:** la password dell'amministratore da sola è un punto unico di rottura.
- **Evita:** presa dell'account admin con password rubata o indovinata; replay di un codice intercettato.
- **Riferimenti:** RFC 6238; NIST SP 800-63B; OWASP Multifactor Authentication Cheat Sheet; OWASP Top 10:2025 A07.

3.4 Passkey

- **Cosa:** WebAuthn/passkey come metodo principale o come secondo fattore resistente al phishing.
- **Perché:** la chiave privata non lascia mai il dispositivo e la firma è legata al dominio.
- **Evita:** phishing in tempo reale, anche contro TOTP.
- **Riferimenti:** NIST SP 800-63B (autenticatori resistenti al phishing); W3C WebAuthn.

3.5 Sessioni e cookie

- **Cosa:** token di sessione in cookie `HttpOnly`, `Secure`, `SameSite`. Token di accesso a vita breve con refresh ruotato a ogni uso; revoca al logout e al cambio password. Scadenza chiara per inattività. Se si usano token firmati (JWT), verificare sempre firma, algoritmo atteso, scadenza e destinatario; mai accettare `alg: none`.
- **Perché:** gli script della pagina non devono poter leggere la sessione; un token rubato deve valere poco e per poco.
- **Evita:** furto di sessione via XSS, CSRF, riuso di token rubati, token falsificati.
- **Riferimenti:** OWASP Session Management Cheat Sheet; OWASP JSON Web Token Cheat Sheet; OWASP Top 10:2025 A07.

3.6 Risposte neutre e limiti di richieste

- **Cosa:** login, registrazione e recupero password rispondono allo stesso modo che l'account esista o no. Limiti di richieste per IP e per account su autenticazione, verifica email, funzioni costose (AI, export).
- **Perché:** l'automazione (oggi anche assistita dall'AI) prova migliaia di combinazioni.
- **Evita:** enumerazione degli utenti, forza bruta, *credential stuffing*, abuso di funzioni a pagamento.
- **Riferimenti:** OWASP Authentication Cheat Sheet; OWASP Top 10:2025 A07; OWASP API Security Top 10 (API4 consumo illimitato di risorse).

4. Autorizzazione

4.1 Revisione route per route con tre domande

- **Cosa:** per ogni route dell'API: **chi può chiamarla? di chi sono i dati che tocca? cosa restituisce?** Tutte le route, non solo quelle sospette. Controllo "nega per impostazione predefinita": una route senza regola esplicita non è accessibile.
- **Perché:** il controllo degli accessi rotto è da anni la prima causa di incidenti nelle web app.
- **Evita:** IDOR (cambiare un ID nella richiesta e vedere i dati altrui), escalation di privilegi, funzioni admin raggiungibili da utenti normali.
- **Riferimenti:** OWASP Top 10:2025 A01; OWASP API Security Top 10 (API1, API5); OWASP Authorization Cheat Sheet.

4.2 Un test che fallisce sul codice vecchio

- **Cosa:** ogni correzione di sicurezza arriva con un test che fallisce prima della correzione e passa dopo.
- **Perché:** dimostra che il problema esisteva, che è chiuso e impedisce che torni.
- **Evita:** regressioni silenziose; controlli che "sembrano" giusti ma non funzionano (per esempio una funzione citata invece che chiamata, che rende un controllo sempre vero).

4.3 Risposte minime (DTO)

- **Cosa:** un'unica funzione costruisce ogni vista di "un altro utente" con i soli campi necessari (per esempio ID e nome visualizzato). Mai restituire l'oggetto del database così com'è.
- **Perché:** i campi in più finiscono nelle risposte quando il modello cresce.
- **Evita:** esposizione di email, ruoli, hash, dati cifrati o campi interni (OWASP API3 *Broken Object Property Level Authorization*).

5. Input, output e API

5.1 Validazione e query sicure

- **Cosa:** validare tipo, formato e lunghezza di ogni input sul server, con uno schema. Query parametrizzate; per i database documentali, rifiutare chiavi che iniziano con `$` o contengono `.` negli input e non passare mai oggetti della richiesta direttamente ai filtri.
- **Evita:** SQL/NoSQL injection, operator injection, prototype pollution, payload enormi.
- **Riferimenti:** OWASP Top 10:2025 A05; OWASP Input Validation Cheat Sheet.

5.2 XSS: output sicuro, anche per Markdown e testo generato dall'AI

- **Cosa:**
  - lasciare che il framework faccia l'escaping (React, Next.js e i template moderni lo fanno per impostazione predefinita);
  - evitare `dangerouslySetInnerHTML`, `innerHTML`, `v-html` e simili; dove sono inevitabili, passare il contenuto da un sanitizzatore con allowlist (per esempio DOMPurify o `rehype-sanitize`);
  - per il Markdown scritto dagli utenti o generato dall'AI: HTML grezzo disattivato, solo elementi e attributi in allowlist, link solo `https:`, `http:` e `mailto:` (mai `javascript:` o `data:`), link esterni con `rel="noopener noreferrer"`;
  - non costruire URL, attributi o codice JavaScript concatenando input;
  - una Content Security Policy come secondo livello (script con nonce o hash, niente `unsafe-inline`), più Trusted Types dove possibile.
- **Perché:** uno script iniettato gira con i permessi dell'utente: legge i dati mostrati, compie azioni a suo nome.
- **Evita:** XSS memorizzato (diario, commenti, profili), XSS riflesso, furto di dati dalla pagina, azioni non volute.
- **Riferimenti:** OWASP Top 10:2025 A05; OWASP Cross Site Scripting Prevention Cheat Sheet; OWASP ASVS 5.0 V1, V3.

5.3 Protezione CSRF e CORS restrittivo

- **Cosa:** per le richieste che modificano dati, verificare `Origin` / `Referer` e usare `SameSite`. CORS limitato alle origini esatte del proprio frontend, mai `*` con le credenziali.
- **Evita:** azioni eseguite a nome dell'utente da siti terzi.
- **Riferimenti:** OWASP CSRF Prevention Cheat Sheet.

5.4 Limiti sul corpo delle richieste e codifiche ammesse

- **Cosa:** dimensione massima del body; accettare solo le codifiche di compressione previste (per esempio `identity`, `gzip`, `deflate`) e rispondere `415` alle altre prima che i parser decomprimano.
- **Perché:** i decompressori sono codice complesso e ogni tanto vulnerabile: nel 2026, per esempio, una vulnerabilità nella decompressione Brotli di Node.js era raggiungibile tramite il body delle richieste.
- **Evita:** *decompression bomb*, sfruttamento dei bug dei decoder, esaurimento della memoria.

5.5 Intestazioni di sicurezza

- **Cosa:** `Strict-Transport-Security`, `Content-Security-Policy` (almeno `frame-ancestors`, `object-src 'none'`, `base-uri 'self'`), `X-Content-Type-Options: nosniff`, `Referrer-Policy`, `Permissions-Policy`. Verificarle con uno scanner gratuito dopo ogni cambio di infrastruttura.
- **Evita:** XSS, clickjacking, downgrade a HTTP, fughe di URL tramite referrer.
- **Riferimenti:** OWASP HTTP Headers Cheat Sheet; OWASP Top 10:2025 A02.

5.6 SSRF: quando il server scarica un URL

- **Cosa:** se l'app scarica qualcosa da un URL fornito dall'utente (anteprime di link, import, immagini remote, webhook in uscita):
  - allowlist di domini o di schemi (`https:` soltanto);
  - risolvere il nome e rifiutare indirizzi privati, di loopback, link-local e di metadati del cloud (per esempio `169.254.169.254`), anche dopo i redirect;
  - timeout brevi, dimensione massima della risposta, niente redirect automatici o redirect ricontrollati;
  - se possibile, fare queste richieste da un servizio separato senza accesso alla rete interna.
- **Evita:** accesso a servizi interni, furto di credenziali dai metadati del cloud, scansione della rete interna.
- **Riferimenti:** OWASP Server Side Request Forgery Prevention Cheat Sheet; OWASP API Security Top 10 (API7); OWASP Top 10:2025 A01.

5.7 Caricamento di file

- **Cosa:**
  - limite di dimensione e di numero;
  - tipo verificato sul contenuto (firma del file), non sull'estensione o sul `Content-Type` dichiarato;
  - nome generato dal server, mai il percorso fornito dall'utente;
  - file salvati fuori dalla cartella pubblica o in un object storage, serviti con `Content-Disposition: attachment` e `nosniff`, meglio da un dominio separato;
  - immagini ricodificate e metadati (EXIF, posizione GPS) rimossi;
  - niente SVG o HTML caricati dagli utenti serviti come pagine;
  - scansione antivirus se i file vengono condivisi con altri utenti.
- **Evita:** esecuzione di codice sul server, XSS tramite file, *path traversal*, esaurimento dello spazio, fughe di posizione tramite le foto.
- **Riferimenti:** OWASP File Upload Cheat Sheet; OWASP ASVS 5.0 V5.

5.8 Webhook in entrata verificati

- **Cosa:** ogni webhook (pagamenti, email, repository) verifica la firma del mittente con HMAC sul **corpo grezzo** della richiesta, con confronto a tempo costante; rifiuta i messaggi con timestamp troppo vecchi; è idempotente (lo stesso evento ricevuto due volte non produce due effetti); risponde in fretta e lavora in coda.
- **Evita:** eventi falsificati (per esempio un "pagamento riuscito" inventato), replay, doppie elaborazioni.
- **Riferimenti:** documentazione del fornitore; OWASP ASVS 5.0 V4.

5.9 Link firmati per le azioni senza login

- **Cosa:** link come "annulla iscrizione" o "scarica export" firmati con HMAC, con scadenza, legati all'utente e all'azione.
- **Evita:** azioni eseguite su altri account modificando un parametro; riuso indefinito dei link.

5.10 Errori senza dettagli

- **Cosa:** all'utente messaggi generici; nei log codici d'errore sicuri, senza messaggi esterni, stack o dati personali. Gli script operativi stampano solo codici da una lista fissa.
- **Evita:** fughe di informazioni su struttura interna, query, segreti.
- **Riferimenti:** OWASP Top 10:2025 A10 *Mishandling of Exceptional Conditions*.

5.11 Nessuna pagina di debug in produzione

- **Cosa:** rimuovere o bloccare pagine di test, diagnostica e endpoint di stato dettagliati; l'health check pubblico restituisce solo "ok/non ok".
- **Evita:** ricognizione da parte di un attaccante, esposizione di versioni e configurazioni.
- **Riferimenti:** OWASP Top 10:2025 A02.

6. Cifratura e gestione delle chiavi

6.1 TLS ovunque

- **Cosa:** HTTPS obbligatorio con HSTS; connessioni cifrate verso database e servizi esterni.
- **Evita:** intercettazione e manomissione del traffico.

6.2 Cifratura applicativa dei campi sensibili

- **Cosa:** l'applicazione cifra i dati personali e i contenuti privati (email, nome, testi scritti dagli utenti, segreti TOTP) **prima** di salvarli, con un algoritmo autenticato (AES-256-GCM), IV casuale per ogni valore. Errore di autenticazione = errore esplicito, mai un valore "vuoto".
- **Perché:** la cifratura "a riposo" del provider protegge i dischi, non da chi accede al database con credenziali valide o da un backup rubato.
- **Evita:** esposizione dei dati in caso di furto del database, dei backup o di accesso del provider.
- **Riferimenti:** OWASP Cryptographic Storage Cheat Sheet; OWASP Top 10:2025 A04; art. 32 GDPR (cifratura come misura tecnica).

6.3 Cifratura a busta (envelope encryption)

- **Cosa:** due livelli di chiavi. Le **DEK** (chiavi dei dati) cifrano i campi; la **KEK** (chiave principale) cifra le DEK. Le DEK cifrate stanno in un *keyring* separato; la KEK sta fuori dal database (variabile sigillata, KMS o HSM). L'app accede al keyring con un utente **di sola lettura**.
- **Perché:** ruotare la KEK è rapido (si ricifrano poche chiavi, non i dati); chi ruba solo il database non ha la KEK.
- **Evita:** decifratura dei dati con il solo database; sostituzione o cancellazione delle chiavi da parte di un'app compromessa.
- **Limite onesto:** se KEK, URI del database e URI del keyring stanno nello stesso posto, chi li ruba tutti può decifrare. Per alzare l'asticella: KEK in un KMS esterno con log e revoca, allowlist di rete sul database.
- **Riferimenti:** NIST SP 800-57 (gestione delle chiavi); OWASP Key Management Cheat Sheet.

6.4 Chiavi con versione e rotazione eseguita davvero

- **Cosa:** ogni dato cifrato porta la versione della chiave (per esempio `k2:...`). La rotazione crea una chiave nuova, la attiva, riavvia le istanze, ricifra i dati in modo riprendibile e verificato (dry-run → create → apply → verify). La chiave vecchia resta archiviata finché esistono backup che la richiedono, poi si ritira. Provare la procedura e poi eseguirla: una rotazione mai fatta non è una misura.
- **Perché:** dopo un sospetto di compromissione, o periodicamente, si deve poter cambiare chiave senza fermare il servizio.
- **Evita:** dipendenza perpetua da una chiave forse esposta; perdita di dati per una rotazione interrotta.
- **Riferimenti:** NIST SP 800-57 (crypto-period e ritiro delle chiavi).

6.5 Indici cechi (blind index) per cercare dati cifrati

- **Cosa:** per trovare un utente dall'email senza decifrare tutto, salvare accanto al campo cifrato un **HMAC con chiave segreta** del valore normalizzato. Mai un hash semplice (SHA-256 senza chiave). Indice univoco sull'HMAC.
- **Perché:** un hash semplice si può ricalcolare da chiunque: con una lista di email trapelate si scopre chi è iscritto.
- **Evita:** re-identificazione degli utenti da un dump del database; attacchi a dizionario sulle email.
- **Limite:** valori uguali producono lo stesso indice; la chiave dell'HMAC va protetta come le chiavi di cifratura. Migrazione in più fasi con verifica in sola lettura prima di togliere il vecchio formato.

6.6 Nessun segreto di riserva nel codice

- **Cosa:** in produzione l'app rifiuta di partire se manca un segreto obbligatorio o se ha un valore di esempio.
- **Evita:** token firmati con valori noti a tutti (`change-me-in-production`), sessioni falsificabili.
- **Riferimenti:** OWASP Top 10:2025 A02.

7. Segreti e configurazione

7.1 Segreti solo nel posto giusto

- **Cosa:** segreti in un password manager e nelle variabili **sigillate** (non leggibili) della piattaforma o in un secrets manager. Mai nel repository, nei log, negli argomenti dei comandi, nella cronologia della shell o in chat. Inserirli nel terminale con prompt nascosti (`read -rs`), svuotare gli appunti dopo l'uso.
- **Evita:** fughe da repository, log, screenshot, cronologie, processi visibili.
- **Riferimenti:** OWASP Secrets Management Cheat Sheet; OWASP Top 10:2025 A02, A04.

7.2 Chiavi API con ambito e limiti

- **Cosa:** una chiave per servizio e per ambiente, nel progetto giusto del fornitore, con permessi ristretti e **limiti di spesa**. Inventario delle chiavi (dove sta, a cosa serve, quando è stata ruotata). Revocare quelle inutilizzate. Dopo ogni rotazione verificare che la chiave nuova sia nel progetto giusto e che la vecchia non funzioni più.
- **Evita:** abuso costoso di chiavi rubate (le chiavi AI sono un bersaglio frequente); chiavi dimenticate ancora attive su un computer.

7.3 Credenziali temporanee per le operazioni rare

- **Cosa:** per operazioni privilegiate (creare una chiave, migrare dati) usare utenti temporanei con scadenza automatica e cancellarli subito dopo.
- **Evita:** credenziali potenti che restano attive per mesi.

7.4 Ambienti separati

- **Cosa:** sviluppo, anteprima e produzione con segreti diversi. I file `.env` locali con permessi `600`, ignorati da git, senza chiavi di produzione.
- **Evita:** che un ambiente di test esponga la produzione; chiavi di produzione sui portatili.

7.5 Scansione dei segreti

- **Cosa:** un controllo automatico che cerca chiavi e password nei commit, prima che arrivino nel repository remoto e poi nella CI. Su GitHub: secret scanning con *push protection* (gratuito sui repository pubblici); in alternativa, o in più, strumenti open source come Gitleaks o TruffleHog in un hook di pre-commit e nella CI. Se un segreto è finito in un commit, va **revocato e ruotato**: cancellarlo dalla cronologia non basta.
- **Evita:** chiavi pubblicate per errore, raccolte in pochi minuti da bot che scandagliano i repository.
- **Riferimenti:** OWASP Secrets Management Cheat Sheet; OWASP Top 10:2025 A02.

8. Privacy e GDPR

8.1 Minimizzazione

- **Cosa:** raccogliere solo ciò che serve. Esempio: statistiche d'uso senza ID utente né IP, con un identificativo casuale di visita di breve durata.
- **Perché:** i dati che non hai non vanno protetti, notificati o giustificati.
- **Riferimenti:** art. 5(1)(c) e art. 25 GDPR.

8.2 Scadenze automatiche

- **Cosa:** ogni tipo di dato ha una conservazione definita e applicata dal database (indici TTL o job). Esempi tipici: statistiche 13 mesi, dati tecnici 90 giorni, bozze incomplete 180 giorni di inattività, log di sicurezza fino a un anno, backup sostituiti entro 6 mesi.
- **Riferimenti:** art. 5(1)(e) GDPR (limitazione della conservazione); per i cookie di statistica, le linee guida del Garante italiano.

8.3 Cancellazione completa e portabilità, con test

- **Cosa:** la cancellazione dell'account elimina i dati in tutte le collezioni; un test riempie ogni collezione e verifica che non resti nulla. Export dei dati in formato leggibile (JSON e CSV) verificato da test, per **ogni** formato offerto.
- **Evita:** dati orfani dimenticati; richieste degli interessati non soddisfatte; un export rotto per mesi senza che nessuno se ne accorga.
- **Riferimenti:** art. 17 e 20 GDPR.

8.4 Informativa scritta partendo dal codice

- **Cosa:** l'informativa descrive ciò che il codice fa davvero (dati, finalità, fornitori, scadenze); si aggiorna quando il codice cambia. Niente interruttori di consenso finti.
- **Riferimenti:** art. 12–14 GDPR.

8.5 Cookie

- **Cosa:** solo cookie tecnici quando possibile. Un cookie di statistica di prima parte, anonimizzato, usato solo in forma aggregata e non incrociato, può essere trattato come tecnico secondo le linee guida del Garante; altrimenti serve il consenso.
- **Riferimenti:** direttiva ePrivacy; linee guida cookie del Garante (2021).

8.6 Registro dei trattamenti e dei fornitori

- **Cosa:** un registro con trattamenti, basi giuridiche, dati, destinatari, fornitori (responsabili) con luogo dei dati, trasferimenti extra UE e stato del DPA. Un nuovo fornitore entra nel registro e nell'informativa **prima** di ricevere dati.
- **Riferimenti:** art. 30 GDPR.

8.7 DPA con ogni fornitore che tratta dati personali

- **Cosa:** un accordo di trattamento (art. 28) con ogni responsabile: hosting, database, email, AI, backup. Spesso è incluso nei termini di servizio: scaricarlo, archiviarlo con la data. Per i fornitori extra UE verificare la base del trasferimento (Data Privacy Framework o clausole contrattuali standard).
- **Riferimenti:** art. 28 e capo V GDPR.

8.8 Registro delle violazioni e notifica

- **Cosa:** ogni incidente, anche senza notifica, va documentato con fatti, valutazione del rischio e misure. Se c'è un rischio per gli interessati: notifica all'autorità entro 72 ore; se il rischio è elevato, comunicazione agli interessati.
- **Riferimenti:** art. 33 (compreso il par. 5) e art. 34 GDPR.

8.9 AI responsabile

- **Cosa:** a ogni funzione AI solo i dati necessari (niente nomi, email o ID quando non servono); contenuti generati etichettati come tali; fornitore che non addestra i modelli sui dati inviati; chiave del fornitore con permessi minimi e limite di spesa; valutazione rispetto all'AI Act.
- **Evita:** fughe di dati verso il fornitore AI; *prompt injection* con effetti sui dati (vedi 2.5); costi fuori controllo.
- **Riferimenti:** Regolamento UE 2024/1689 (AI Act); OWASP Top 10 for LLM Applications.

8.10 Valutazione d'impatto quando serve

- **Cosa:** per trattamenti a rischio elevato (dati sensibili su larga scala, minori, profilazione con effetti significativi) una DPIA prima di iniziare.
- **Riferimenti:** art. 35 GDPR.

9. Log, audit e allarmi

9.1 Log di audit utili ma minimi

- **Cosa:** registrare le azioni rilevanti (login, cambi di ruolo, export, cancellazioni, operazioni sulle chiavi) con ID e codici, non con email o contenuti. Oscurare segreti e indirizzi; dove serve un riferimento usare ID o HMAC, con scadenza.
- **Evita:** log che diventano essi stessi una fuga di dati; incidenti impossibili da ricostruire.
- **Riferimenti:** OWASP Logging Cheat Sheet; OWASP Top 10:2025 A09.

9.2 Allarmi

- **Cosa:** notifiche automatiche per: backup mancato o fallito, accessi anomali al database, picchi di spesa sui servizi a consumo, errori in aumento, login admin. Un messaggio "tutto ok" quotidiano rende visibile anche l'assenza di notizie.
- **Perché:** un attacco scoperto dopo settimane fa molti più danni di uno scoperto in un'ora.
- **Riferimenti:** OWASP Top 10:2025 A09 *Security Logging & Alerting Failures*.

10. Backup e ripristino

10.1 Backup cifrati con chiave pubblica

- **Cosa:** backup automatici giornalieri del database **e** del keyring, cifrati con una chiave pubblica (per esempio `age`) prima di uscire dal server. La chiave privata sta solo offline, nel password manager.
- **Perché:** il servizio di backup può cifrare ma non decifrare: se viene compromesso, i backup restano illeggibili.

10.2 Copie immutabili fuori dalla piattaforma principale

- **Cosa:** destinazione separata dal provider di hosting (regola 3-2-1), con **Object Lock in modalità Compliance** (nessuno può cancellare per N giorni, nemmeno l'admin) e una chiave che può solo aggiungere file.
- **Evita:** ransomware e sabotaggio: chi entra nell'hosting e cancella il database non può cancellare anche i backup.

10.3 Conservazione coerente con l'informativa

- **Cosa:** cancellazione automatica dei backup oltre la scadenza dichiarata (per esempio 6 mesi), coordinata con il ritiro delle chiavi vecchie.
- **Riferimenti:** art. 5(1)(e) GDPR.

10.4 Prova di ripristino periodica

- **Cosa:** almeno una volta al mese, scaricare un backup, verificarne l'hash, decifrarlo e caricarlo in un database **locale** usa e getta, confrontando i conteggi. Un backup mai ripristinato non è un backup.
- **Riferimenti:** CIS Controls v8.1, controllo 11 (Data Recovery).

10.5 Backup prima di ogni cambiamento importante

- **Cosa:** backup cifrato e verificato prima di migrazioni, rotazioni di chiavi o grandi rilasci.

10.6 Primo backup nell'ambiente reale

- **Cosa:** dopo aver attivato i backup, lanciare subito un'esecuzione vera con le credenziali e la rete di produzione, controllare che i file arrivino e che il blocco sia attivo, poi fare la prima prova di ripristino. Non basta che i test siano verdi.
- **Perché:** i test girano di solito con un database locale senza cifratura e con uno storage finto; la produzione usa TLS, DNS e permessi reali. Un caso tipico: un'immagine container *slim* senza i certificati delle autorità, per cui lo strumento di dump non riesce a verificare la connessione cifrata al database.
- **Evita:** credere di avere backup che in realtà falliscono ogni notte.

11. Catena di fornitura del software

11.1 Versioni esatte e lockfile

- **Cosa:** lockfile nel repository; installazioni riproducibili (`npm ci`); versioni esatte delle dipendenze dirette.
- **Evita:** aggiornamenti non voluti con codice malevolo.

11.2 Attesa prima di adottare nuove versioni

- **Cosa:** non installare versioni pubblicate da meno di qualche giorno (per esempio 7), salvo correzioni di sicurezza.
- **Perché:** i pacchetti compromessi vengono di solito scoperti e ritirati in pochi giorni.
- **Evita:** attacchi alla catena di fornitura del tipo "pacchetto popolare compromesso".

11.3 Script di installazione disattivati

- **Cosa:** `--ignore-scripts` dove possibile, con eccezioni esplicite.
- **Evita:** furto di segreti durante `npm install`.

11.4 Aggiornamenti automatici e audit

- **Cosa:** Dependabot (o simile) con aggiornamenti raggruppati e settimanali; correzioni di sicurezza immediate; major a mano. `npm audit` delle dipendenze di produzione nella CI, bloccante su high/critical. Mai `npm audit fix --force` alla cieca: può installare major incompatibili. Il flusso completo è in 12.6.

11.5 Azioni CI, immagini e strumenti fissati

- **Cosa:** GitHub Actions fissate con l'hash del commit; immagini Docker con versione esatta (e digest dove possibile); strumenti scaricati verificati con checksum. Aggiornarli tramite PR automatiche, non con tag mobili. Token della CI con permessi minimi (`permissions:` esplicito, sola lettura per default).
- **Evita:** azioni o immagini sostituite da versioni malevole; build non riproducibili; una CI compromessa che scrive nel repository.
- **Riferimenti:** OWASP Top 10:2025 A03 *Software Supply Chain Failures* e A08; SLSA; OpenSSF Scorecard.

11.6 Runtime aggiornato

- **Cosa:** seguire le security release del runtime (Node.js, Python...), usare una versione LTS fissata in modo identico in sviluppo, CI e produzione, e verificare la versione effettivamente in esecuzione in produzione, non solo quella dichiarata.
- **Evita:** vulnerabilità del runtime raggiungibili da remoto anche con codice applicativo corretto.

11.7 Configurazione morta rimossa

- **Cosa:** eliminare file di configurazione non usati (vecchi builder, pagine di prova, script).
- **Evita:** confusione su cosa gira davvero; configurazioni dimenticate che si riattivano.

12. Test, CI e rilascio

12.1 Perché i test sono una misura di sicurezza

- **Cosa:** considerare i test automatici parte della sicurezza, non solo della qualità.
- **Perché:** quasi ogni misura di questa guida è codice, e il codice si rompe quando cambia. I test servono a tre cose:
  - **dimostrano** che un controllo funziona (chi non è proprietario riceve un 403, la cancellazione non lascia dati, il token scaduto viene rifiutato);
  - **impediscono che si rompa** in silenzio alla modifica successiva;
  - **rendono possibile aggiornare in fretta**: senza test, ogni aggiornamento di sicurezza è una scommessa e tende a essere rimandato, cioè proprio quello che un attaccante spera.
- **Evita:** regressioni di sicurezza, funzioni rotte per mesi senza che nessuno se ne accorga, dipendenze vulnerabili lasciate al loro posto per paura di rompere qualcosa.

12.2 Tre livelli di test

- **Unitari:** singole funzioni, veloci e numerosi. Per la sicurezza: cifratura e decifratura (compreso il rifiuto di un dato manomesso), firma e verifica dei token, validazione degli input, normalizzazione per i blind index, vettori ufficiali per TOTP.
- **Di integrazione e di API:** chiamano le route vere con un database di prova. Per la sicurezza: ogni regola di autorizzazione (proprietario, altro utente, anonimo, admin), risposte senza campi interni, limiti di richieste, cancellazione completa, export in ogni formato, rifiuto di input malformati.
- **Dell'interfaccia (end-to-end):** un browser automatico (per esempio Playwright) percorre i flussi principali: registrazione, login con secondo fattore, pagine chiave, export, cancellazione. Trovano errori che gli altri non vedono: una pagina che va in errore per un import mancante, un pulsante che chiama la route sbagliata.
- **Proporzione:** molti unitari, un buon numero di test di API, pochi end-to-end sui percorsi critici. Gli end-to-end sono lenti e più fragili: vanno tenuti pochi e stabili.
- **Riferimenti:** OWASP Web Security Testing Guide; OWASP ASVS 5.0 (i requisiti si prestano a diventare casi di test).

12.3 Copertura: utile, ma non è l'obiettivo

- **Cosa:** misurare la copertura (righe e rami eseguiti dai test) e guardare soprattutto **dove manca**. Come ordine di grandezza, una copertura ampia del backend è un buon segnale, ma conta di più che i percorsi sensibili (autenticazione, autorizzazione, cifratura, cancellazione, export, pagamenti) siano coperti al completo, compresi i casi di errore. Ogni funzione che gli utenti usano merita almeno un test.
- **Perché:** più codice è coperto, più è probabile che una modifica che rompe qualcosa venga fermata prima del merge. Ma una riga "coperta" da un test senza verifiche significative non protegge nulla, e la percentuale si può gonfiare con test inutili.
- **Evita:** la falsa sicurezza di un numero alto; buchi proprio nei punti che contano.

12.4 Test rosso → verde per ogni correzione

- **Cosa:** ogni bug, e in particolare ogni problema di sicurezza, si corregge così: prima un test che riproduce il problema e **fallisce**, poi la correzione che lo fa **passare** (vedi 4.2). Nel report della modifica si confronta la suite prima e dopo, test per test e non solo nei totali.
- **Evita:** correzioni che non correggono; test che non possono fallire; regressioni.

12.5 CI bloccante e stabile

- **Cosa:** a ogni pull request la CI esegue: tutta la suite di test, la build, il controllo delle vulnerabilità nelle dipendenze di produzione e il lint degli errori certi. Se un controllo fallisce, il merge è bloccato (branch protection con *required status checks*). I test instabili si correggono subito: cause tipiche sono l'esaurimento della memoria con troppi processi paralleli, download concorrenti delle stesse risorse, dipendenze dall'orario o dall'ordine dei test.
- **Perché:** una CI che "a volte è rossa" insegna a ignorare il rosso, e un rosso ignorato vale quanto nessun controllo. La CI gira anche su un sistema diverso da quello dello sviluppatore e trova differenze che in locale non si vedono.
- **Evita:** regressioni in produzione; abitudine a fare merge con i controlli falliti.

12.6 Quando arriva una vulnerabilità: il flusso

Il valore di test e CI si vede soprattutto qui. Un flusso tipico:

1. **Avviso.** Dependabot (o simile) segnala una vulnerabilità in una dipendenza e apre una pull request con la versione corretta.
2. **Valutazione.** Leggere l'avviso: la dipendenza è di produzione o solo di sviluppo? La parte vulnerabile è raggiungibile nell'app? La gravità guida la fretta, non la decisione di aggiornare.
3. **Controlli automatici.** La CI esegue test, build, audit e lint sulla pull request.
4. **Verde → merge.** Se tutti i controlli sono verdi e il diff (lockfile compreso) è ragionevole, si fa il merge. Il comando di merge si dà **dopo** aver visto i controlli, non nello stesso blocco che fa push e apre la PR.
5. **Rilascio e verifica.** Deploy, poi una prova breve dei percorsi principali (login, una funzione chiave, un export) e uno sguardo ai log.
6. **Rollback pronto.** Se qualcosa non va, si torna alla versione precedente in pochi secondi (12.10) e si indaga con calma.
7. **Rosso → indagare.** Se un controllo fallisce, non si forza: si capisce se l'aggiornamento rompe davvero qualcosa o se il test va adeguato. Le versioni major restano decisioni manuali, su un branch dedicato.

Con una buona suite, un aggiornamento di sicurezza richiede minuti. Senza, richiede ore di prove a mano, e per questo viene rimandato.

12.7 Lint e analisi statica (SAST)

- **Cosa:**
  - almeno un controllo statico **bloccante** sugli errori che rompono sicuramente il codice: identificatori non definiti, import che non si risolvono, uso scorretto degli hook; il lint completo diventa bloccante quando gli errori arrivano a zero, con un tetto sugli avvisi;
  - un analizzatore di sicurezza del codice: CodeQL (gratuito sui repository pubblici di GitHub) o Semgrep Community Edition (open source), con le regole per il proprio linguaggio; i risultati si rivedono come i bug, non si silenziano in blocco.
- **Perché:** un errore di questo tipo può rompere una pagina in produzione senza che la build se ne accorga, e i lint solo informativi vengono ignorati anche quando segnalano bug veri.
- **Evita:** pagine in errore in produzione; schemi di codice notoriamente pericolosi (injection, `eval`, deserializzazione insicura, segreti nel codice).
- **Riferimenti:** OWASP Source Code Analysis Tools; OWASP Top 10:2025 A05, A06.

12.8 Scansione dinamica (DAST)

- **Cosa:** una scansione automatica dall'esterno dell'app in esecuzione, per esempio la *baseline scan* di OWASP ZAP, su un ambiente di prova o, con cautela e in modalità passiva, in produzione. Una volta a ogni rilascio importante o mensilmente.
- **Perché:** trova configurazioni sbagliate che il codice non mostra: intestazioni mancanti, cookie senza attributi, pagine dimenticate, versioni esposte.
- **Evita:** errori di configurazione visibili a chiunque; arrivare a un test di penetrazione con problemi banali ancora aperti.
- **Riferimenti:** OWASP ZAP; OWASP Web Security Testing Guide.

12.9 Revisione e merge controllati

- **Cosa:** ogni modifica passa da una PR con controlli verdi; una persona legge il diff (anche quando l'ha scritto un'AI); il merge si fa solo dopo aver visto i controlli, non in automatico nello stesso comando. Il branch principale è protetto: niente push diretti, niente merge con controlli rossi.
- **Evita:** codice non revisionato in produzione; merge accidentali.

12.10 Rollback immediato

- **Cosa:** sapere come tornare alla versione precedente in pochi secondi (per esempio *Instant Rollback* della piattaforma) e averlo provato. Per le modifiche al database, prevedere la compatibilità con la versione precedente del codice o una procedura inversa.
- **Evita:** ore di disservizio mentre si cerca una correzione sotto pressione.

13. Infrastruttura, dominio e fornitori

13.1 Autenticazione a due fattori su tutti gli account dei fornitori

- **Cosa:** hosting, database, DNS e registrar, email, repository, AI, backup: tutti con 2FA, meglio con passkey o chiavi hardware. Alert sulle modifiche importanti.
- **Evita:** presa dell'intera infrastruttura con una sola password.

13.2 Utenti del database con privilegio minimo

- **Cosa:** utente dell'app con permessi solo sul suo database; utente del keyring in sola lettura; utente dei backup in sola lettura; nessun utente amministrativo usato dall'app.

13.3 Accesso di rete al database ristretto

- **Cosa:** allowlist degli IP di uscita dei servizi (serve un IP di uscita fisso), oppure rete privata. Se non è possibile, compensare con credenziali forti, ruotate e monitorate.
- **Evita:** uso di credenziali rubate da qualsiasi punto di internet.

13.4 Dati nell'Unione Europea

- **Cosa:** regioni UE per database, funzioni e backup; registrare dove stanno i dati e quali trasferimenti restano (supporto, edge network).
- **Riferimenti:** capo V GDPR.

13.5 Container non root e immagini minime

- **Cosa:** processi senza privilegi di root; immagini *slim* aggiornate; nessuno strumento superfluo in produzione. Le immagini *slim* spesso non includono i certificati delle autorità (`ca-certificates`): Node.js ha i suoi, ma strumenti scritti in altri linguaggi (per esempio `mongodump`) ne hanno bisogno per verificare le connessioni TLS. Installarli quando servono e controllare in fase di build che il file esista.

13.6 Dominio e DNS

- **Cosa:**
  - registrar con 2FA, *registrar lock* attivo e rinnovo automatico;
  - record CAA che indicano quali autorità possono emettere certificati per il dominio;
  - DNSSEC dove il registrar e il DNS lo supportano;
  - nessun record che punta a servizi dismessi (un sottodominio "orfano" può essere preso da altri);
  - monitoraggio dei certificati emessi per il dominio tramite i log di Certificate Transparency.
- **Evita:** furto del dominio, certificati emessi a terzi, *subdomain takeover*, phishing dal proprio dominio.
- **Riferimenti:** RFC 8659 (CAA); RFC 9364 (DNSSEC); Certificate Transparency (RFC 9162).

13.7 Email del dominio: SPF, DKIM, DMARC

- **Cosa:** SPF che elenca chi può inviare a nome del dominio; DKIM per firmare i messaggi; DMARC, prima in monitoraggio (`p=none` con report) e poi in `quarantine` o `reject`. Per i domini che non inviano email: SPF `v=spf1 -all` e DMARC `p=reject`.
- **Perché:** le email transazionali (magic link, reset password) sono un canale di autenticazione; se il dominio è falsificabile, il phishing diventa credibile. I grandi provider di posta richiedono già queste impostazioni a chi invia in volume.
- **Evita:** email false a nome dell'app; magic link e notifiche che finiscono nello spam.
- **Riferimenti:** RFC 7208 (SPF); RFC 6376 (DKIM); RFC 7489 (DMARC).

13.8 security.txt e segnalazione delle vulnerabilità

- **Cosa:** un file `/.well-known/security.txt` con almeno `Contact` ed `Expires` (obbligatori), più `Preferred-Languages` e un link a una breve politica di segnalazione: come segnalare, cosa aspettarsi, impegno a non agire contro chi segnala in buona fede.
- **Perché:** chi trova un problema deve sapere a chi scriverlo, invece di pubblicarlo o lasciar perdere.
- **Riferimenti:** RFC 9116; ISO/IEC 29147.

14. Risposta agli incidenti

Quando un fornitore o un componente segnala un possibile incidente:

1. **Presumere la compromissione** di tutto ciò che il fornitore poteva vedere (variabili d'ambiente, chiavi, token).
2. **Ruotare tutti i segreti**, partendo da quelli che danno accesso ai dati (database, KEK, firme delle sessioni), poi chiavi di servizi esterni. Verificare dove è stata ruotata ogni chiave e che le vecchie siano revocate.
3. **Controllare i log** di ogni servizio dall'inizio della finestra di esposizione.
4. **Documentare** nel registro delle violazioni: fatti, valutazione del rischio, decisione sulla notifica (art. 33).
5. **Rafforzare** ciò che l'incidente ha mostrato debole (2FA, variabili sigillate, separazione delle chiavi).
6. **Post-mortem** scritto: cosa è successo, cosa ha funzionato, cosa cambia.

Riferimenti: NIST SP 800-61 (gestione degli incidenti); art. 33–34 GDPR.

15. Corrispondenza con OWASP ASVS 5.0

OWASP ASVS è lo standard di riferimento per verificare la sicurezza di un'applicazione web. La tabella indica dove la guida tratta i temi di ciascun capitolo. È una corrispondenza indicativa: per dichiarare un livello ASVS servono la verifica requisito per requisito e, di norma, una valutazione indipendente.

Capitolo ASVS 5.0	Sezioni della guida
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	Fuori ambito (solo uso di provider esterni)
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	Fuori ambito

16. Checklist

Metodo

☐ Modello delle minacce scritto e aggiornato

☐ Regole per gli agenti AI nel repository; revisione umana di ogni diff

☐ Un attore alla volta nella cartella di lavoro

☐ Script di produzione con dry-run, solo conteggi, backup prima

☐ Agenti AI con permessi minimi, senza credenziali di produzione, conferma umana per le azioni irreversibili

Autenticazione

☐ Password con Argon2id/scrypt/bcrypt, controllo contro password compromesse

☐ Magic link monouso, 10–15 minuti, salvati come hash, risposta neutra

☐ TOTP obbligatorio per gli admin, anti-replay, codici di recupero hashati, procedura d'emergenza registrata

☐ Cookie HttpOnly, Secure, SameSite; access token brevi, refresh ruotato; JWT verificati per firma, algoritmo e scadenza

☐ Risposte che non rivelano l'esistenza degli account; limiti di richieste

☐ Passkey (quando possibile)

Autorizzazione

☐ Revisione di tutte le route con le tre domande; nega per impostazione predefinita

☐ Un test rosso→verde per ogni correzione

☐ DTO minimi per i dati di altri utenti

Input, output e API

☐ Validazione server-side con schema, protezione da injection e operator injection

☐ Nessun HTML non sanitizzato; Markdown e output AI con allowlist; link solo http/https/mailto

☐ CSRF (Origin + SameSite), CORS limitato alle proprie origini

☐ Limite al body e solo codifiche ammesse (415 per le altre)

☐ Intestazioni di sicurezza (HSTS, CSP, nosniff…) verificate con uno scanner

☐ URL scaricati dal server: allowlist, niente indirizzi interni o di metadati

☐ Upload: limiti, tipo verificato sul contenuto, nomi generati, metadati rimossi, file serviti come allegati

☐ Webhook: firma sul corpo grezzo, timestamp, idempotenza

☐ Link firmati e con scadenza per le azioni senza login

☐ Errori generici; nessuna pagina di debug in produzione

Cifratura e chiavi

- ☐ TLS ovunque
- ☐ Campi sensibili cifrati nell'app con AES-256-GCM
- ☐ Cifratura a busta: KEK fuori dal database, keyring in sola lettura per l'app
- ☐ Chiavi con versione; rotazione provata ed eseguita; ritiro delle chiavi vecchie pianificato
- ☐ Ricerca su dati cifrati solo con HMAC con chiave
- ☐ Avvio rifiutato se manca un segreto

Segreti

- ☐ Segreti solo in password manager e variabili sigillate; mai in repo, log, argv, chat
- ☐ Chiavi API per progetto/ambiente, con permessi ristretti e limiti di spesa; inventario; rotazioni verificate
- ☐ Utenti temporanei per le operazioni privilegiate
- ☐ Ambienti separati; `.env` locali senza chiavi di produzione
- ☐ Scansione dei segreti prima del push e nella CI; segreto esposto = revoca e rotazione

Privacy e GDPR

- ☐ Minimizzazione (statistiche senza identità)
- ☐ Scadenze automatiche per ogni tipo di dato
- ☐ Cancellazione completa ed export testati, in ogni formato
- ☐ Informativa scritta dal codice; cookie solo tecnici o con consenso
- ☐ Registro dei trattamenti e dei fornitori; nuovo fornitore registrato prima di ricevere dati
- ☐ DPA archiviato per ogni fornitore; trasferimenti extra UE verificati
- ☐ Registro delle violazioni; procedura di notifica entro 72 ore
- ☐ AI: dati minimi, etichette, nessun addestramento, limite di spesa; AI Act valutato
- ☐ DPIA se il trattamento è a rischio elevato

Log e allarmi

- ☐ Audit log minimi, con ID e scadenza; segreti ed email oscurati
- ☐ Allarmi su backup, accessi anomali, spesa, errori

Backup

- ☐ Backup giornalieri cifrati con chiave pubblica, chiave privata offline
- ☐ Copie immutabili (Object Lock Compliance) fuori dall'hosting, chiave solo scrittura
- ☐ Cancellazione oltre la scadenza dichiarata
- ☐ Primo backup verificato nell'ambiente reale, non solo nei test
- ☐ Prova di ripristino mensile su database locale

Catena di fornitura

☐ Lockfile, `npm ci`, versioni esatte

☐ Attesa minima sulle nuove versioni; script di installazione disattivati

☐ Dependabot raggruppato; audit in CI bloccante su high/critical; mai `audit fix --force` alla cieca

☐ Actions per hash, immagini e strumenti fissati con checksum; token CI con permessi minimi

☐ Runtime LTS uguale ovunque e verificato in produzione

☐ Configurazione morta rimossa

Test, CI e rilascio

☐ Test unitari sulle funzioni di sicurezza (cifratura, token, validazione)

☐ Test di API per ogni regola di autorizzazione, cancellazione ed export

☐ Test end-to-end dei percorsi principali

☐ Copertura misurata; percorsi sensibili coperti al completo; ogni funzione usata ha almeno un test

☐ Suite bloccante e stabile; test instabili corretti subito

☐ Flusso per le vulnerabilità: PR automatica → controlli verdi → merge → verifica → rollback pronto

☐ Lint degli errori certi bloccante; SAST attivo

☐ Scansione DAST periodica

☐ Branch principale protetto; merge solo dopo aver visto i controlli

☐ Rollback conosciuto e provato

Infrastruttura, dominio e fornitori

☐ 2FA su tutti i fornitori, registrar compreso

☐ Utenti del database con privilegio minimo (app, keyring, backup)

☐ Accesso di rete al database ristretto (o compensato)

☐ Dati in UE; trasferimenti residui documentati

☐ Container non root, immagini minime; `ca-certificates` presente se strumenti non Node aprono connessioni TLS

☐ Registrar lock, CAA, nessun sottodominio orfano; DNSSEC dove possibile

☐ SPF, DKIM, DMARC (fino a `reject`)

☐ `security.txt` con contatto e scadenza

Incidenti

☐ Procedura scritta: presumere, ruotare, controllare, documentare, rafforzare, post-mortem

17. Riferimenti

- 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/
- Regolamento (UE) 2016/679 (GDPR) — https://eur-lex.europa.eu/eli/reg/2016/679/oj
- Regolamento (UE) 2024/1689 (AI Act) — https://eur-lex.europa.eu/eli/reg/2024/1689/oj
- Garante privacy, Linee guida cookie e altri strumenti di tracciamento (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. Versioni, licenza e contatti

Versione	Data	Modifiche
1.0	Novembre 2026	Prima versione, derivata dalla revisione di sicurezza di IkigaiApp
1.2	17 novembre 2026	Primo backup nell'ambiente reale (10.6); certificati delle autorità nelle immagini *slim* (13.5); voci corrispondenti nella checklist
1.1	11 novembre 2026	Ambito e limiti; le dieci misure prioritarie; XSS e Markdown; SSRF; upload; webhook; scansione dei segreti; agenti AI e prompt injection; sezione su test, CI e gestione delle vulnerabilità ampliata; SAST e DAST; dominio, DNS ed email; security.txt; corrispondenza con ASVS 5.0; versione per agenti

Licenza. Testo distribuito con licenza Creative Commons Attribuzione 4.0 Internazionale (CC BY 4.0): puoi copiarlo, adattarlo e usarlo anche per scopi commerciali, citando l'autore e la fonte. https://creativecommons.org/licenses/by/4.0/deed.it

Fonte e contatti. Versione aggiornata, PDF e versione per agenti: https://salvatorevivolo.org/it/guides/web-app-security. Correzioni e suggerimenti sono benvenuti tramite i contatti indicati sul sito.

Caso di studio. Come queste misure sono state applicate a un'app reale: https://salvatorevivolo.org/it/projects/ikigaiapp/security-case-study