



Before You Ship: Failure Modes and Pre-Launch Checks for AI-Built Applications

Stargazing Creativities Lab

Signal. Functional success does not establish that an AI-built application is secure or ready to launch.

Prompt-led development can produce a convincing demonstration before the builder has examined permissions, failure states, recovery or operating cost. This survey reviews 68 academic, standards, advisory and documentation sources available on 22 October 2026. Studies support a recurring gap between functional and security outcomes, while estimates of deployed-app risk remain sensitive to sampling and tool versions. We distinguish application defects, inherited platform failures and development-agent risks. The taxonomy covers security, correctness, payments, privacy, operations, accessibility and performance, with mappings to the web OWASP Top 10:2025 and LLM Top 10:2026. A consequence-based framework produces 36 pre-launch checks with free or manual verification, contextual severity and explicit pass conditions. Prompts can state these obligations, but their implementation requires independent evidence. The framework is a practical synthesis, not a validated estimate of defect prevalence or a guarantee of safe operation.

1 Introduction

A demonstration usually exercises a cooperative user, a clean database and available services. A launch exposes the same application to different identities, repeated requests, interrupted payments, malformed inputs and failed dependencies. AI-generated backends can pass functional tests while failing security tests, so visible success is an incomplete release criterion (Vero et al., 2025; Zhao et al., 2026). [STRONG]

This survey asks which failures builders should expect to look for, what evidence supports those concerns, and which affordable checks expose them. It covers applications built substantially through prompts, including conventional applications that contain no model at runtime. It also covers the extra trust boundaries introduced when an application or its development agent invokes an LLM.

The selected evidence does not support a single ecosystem-wide percentage of unsafe AI-built applications. Studies examine different units, from isolated suggestions to repository changes and deployed services. A recent preprint supplies direct deployed-app evidence, but its public-repository sample and two tool fingerprints do not represent every builder or deployment (Deng et al., 2026). [LIMITED] The useful question is therefore narrower. Which observable failures recur in the studied settings, and which checks can establish a specific property of the application being launched?

We prioritize loss of access control, secrets, money and recoverable data before cosmetic polish. This ordering is the lab's consequence-based judgment, not an empirical ranking of worldwide losses. The proposed checks are a synthesis of observed defects, documented system behavior and established verification guidance. OWASP's Application Security Verification Standard (ASVS) and Web Security Testing Guide (WSTG) supply a broader basis for assurance than treating a Top 10 list as a complete test plan (OWASP Foundation, 2025a, 2020). [MODERATE]

Signal

1. A failure taxonomy and empirical synthesis that distinguish generated code, inherited platform defects and development-agent risks.
2. Current OWASP mappings, bounded incident evidence and explicit limits on prevalence and causal claims.
3. A ranked 36-item launch checklist with free or manual verification, a release procedure, prevention prompts and the complete search log.

2 Method of the Survey

Scope and search. This is a structured narrative survey, not a systematic review or statistical meta-analysis. The final search and source checks were performed on 22 October 2026. We ran 47 exact queries in 33 search batches using two web search services, then followed relevant links to original papers, official documentation and disclosures. Searches covered arXiv, USENIX proceedings, PMLR, NeurIPS proceedings, publisher records and official project sites. Query families combined *vibe coding*, *AI-generated code*, *security*, *correctness*, *package hallucination*, *webhooks*, *backups*, *accessibility* and the named platforms. Appendix B records the exact queries, dates, engines, selections and direct-navigation sources.

Inclusion and extraction. We retained 68 sources that supplied an identifiable study, directly documented failure, technical requirement or concrete verification method. Preference went to peer-reviewed work, inspectable preprints, standards, maintainer advisories and official documentation. We recorded the title, author or responsible organization, year, venue or URL, access date, supported claim and limitation. Living documents without a verified publication date are cited as n.d. Vendor announcements establish dates and capabilities, not product quality. Practitioner sources are labeled where used. Social posts and videos were not used as independent evidence of incidents or prevalence.

What the units mean. A vulnerable suggestion, a scanner warning, a failed benchmark task, an affected platform and a compromised user are different outcomes. We do not pool their percentages. We distinguish the existence of a failure mechanism, its observed frequency in a bounded sample, and evidence that a check can detect it. None of the selected studies establishes the sensitivity of the complete checklist in Appendix A.

Evidence tags. [STRONG] marks a directly documented technical property or a finding supported across independent sources with compatible scope. [MODERATE] marks a bounded empirical result or an engineering recommendation supported by credible but context-dependent evidence. [LIMITED] marks a recent unreviewed result, a narrow observation or an extrapolation that remains uncertain. For a standard or API contract, *strong* concerns what the requirement or system does. It does not establish how often builders violate it. The tags are editorial assessments, not numerical confidence scores.

Versions and verification. Mappings use the web OWASP Top 10:2025 and the LLM Top 10:2026. The latter was verified in the official downloadable report because an older web archive still presents the 2025 list ([OWASP Foundation, 2025b](#); [OWASP GenAI Security Project, 2026](#)). Preprint versions are pinned where findings changed. For example, SUSVIBES version 4 has 186 tasks; earlier versions have different counts ([Zhao et al., 2026](#)). Replication work also shows why a result belongs to a particular model, task and scanner configuration ([Majdinasab et al., 2023](#)). [MODERATE] Bibliographic metadata was checked against opened source records; inaccessible pages supplied no substantive claims.

3 What Vibe Coding Is

We use *vibe coding* to mean prompt-led application construction in which a person delegates substantial implementation to an AI system and iterates on its output. Usage ranges from closely reviewed assistance to accepting code without understanding it. These workflows should not be treated as equivalent. [Sarkar and Drosos \(2025\)](#) document the term’s introduction by Andrej Karpathy on 2 February 2025 and study conversational development practices. Their qualitative sample explains a workflow, not its population prevalence. [MODERATE]

Table 1 is a functional classification. An app builder can contain an agent, and an assistant can acquire agentic capabilities. We infer the relevant boundary from what the tool can read, change and deploy, rather than from its marketing category.

3.1 What the empirical studies establish

The evidence is strongest for a mismatch between functional success and security assurance. It is weaker for universal claims about whether AI assistance increases risk relative to human development. Table 2 keeps each study’s observation next to its unit and limitation. [MODERATE]

Table 1: Overlapping tool types and verified historical anchors. Dates refer to the opened announcement, not to an inferred first deployment.

Tool type	Typical delegated work	Verified anchor
Coding assistant	Suggestions or edits inside a developer’s coding workflow; the human may still control integration and execution.	GitHub Copilot technical preview, 29 June 2021 (Friedman, 2021).
AI app builder	Natural-language specification connected to application assembly, environment setup and deployment.	Replit Agent announcement, 16 September 2024 (The Replit Team, 2024).
Agentic coding tool	Multi-step repository inspection, edits, commands and tests, with consequential tool permissions.	Claude Code limited research preview, 24 February 2025 (Anthropic, 2025).

Table 2: Selected empirical evidence. Percentages describe the named sample, not all AI-built applications.

Study	Unit and design	Supported observation	Interpretation limit
Pearce et al. (2022)	1,689 generated programs from 89 security-relevant scenarios.	Approximately 40% were judged vulnerable. [MODERATE]	Early Copilot and deliberately targeted weaknesses.
Perry et al. (2023)	Controlled study with 47 participants and five security-related tasks.	Assistant users produced less secure code overall and expressed greater confidence in its security. [MODERATE]	A particular assistant, participant sample and task set.
Sandoval et al. (2023)	58 students completing a C programming task.	The observed security effect was small in this setting. [MODERATE]	Contrasts with a blanket claim that assistance always worsens security.
Fu et al. (2025)	733 identified generated snippets in public GitHub projects.	Security weaknesses appeared in 29.5% of Python and 24.2% of JavaScript snippets. [MODERATE]	Selected snippets and scanner coverage; not a deployed-app compromise rate.
Vero et al. (2025)	BaxBench, 392 backend-generation tasks with separate functional and security evaluation.	Roughly half of functionally correct outputs were vulnerable on average across tested models. [MODERATE]	Security-focused modules and historical model settings.
Zhao et al. (2026)	SUSVIBES v4, 186 repository tasks drawn from historically vulnerable implementations.	SWE-Agent with Claude 4 Sonnet passed functional tests on 57% of all tasks and both functional and security tests on 11.8%. [MODERATE]	Vulnerability-enriched benchmark; the two percentages are not field rates.
Deng et al. (2026)	Preprint audit of 200 randomly selected deployments from 984 deployed repositories in a fingerprint-selected corpus.	182 audited apps had at least one validated vulnerability. Access control, injection and authentication led the categories. [LIMITED]	Open-source apps associated with two tools; no matched conventional-app control.
Wu et al. (2026)	October 2026 preprint, 450 tasks derived from 150 known vulnerabilities.	Functional and security outcomes diverged; security reminders did not remove all failures. [LIMITED]	Very recent, vulnerability-enriched evaluation.

Two distinctions matter. First, controlled user studies disagree in magnitude and setting; that is a reason to preserve their conditions, not choose the more dramatic result. Second, the deployed-app audit begins with 9,041 fingerprint-identified GitHub applications and narrows to publicly reachable deployments. Its 182/200 result is important direct

evidence for that sample, but cannot supply a universal prevalence or causal AI-versus-human comparison (Deng et al., 2026). [LIMITED]

Correctness has a similar measurement problem. EvalPlus expanded HumanEval’s test cases and revealed incorrect outputs that the original tests missed (Liu et al., 2023). [MODERATE] A test suite demonstrates behavior on its exercised cases. Increasing coverage of meaningful boundary conditions can change a conclusion that looked settled after a happy-path demonstration.

4 A Taxonomy of Failure Modes

Figure 1 groups failures by the property users lose. The categories overlap. An unchecked upload can expose private files, consume storage and block a workflow. An unverified payment event can corrupt both money and application state. Classification helps assign checks; it does not imply independent causes.

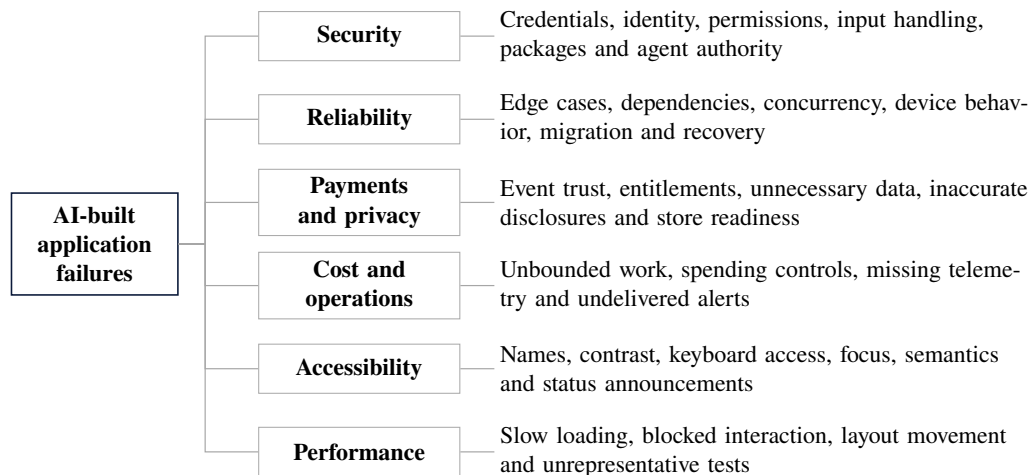


Figure 1: Failure taxonomy used in this survey. The branches organize checks by user-visible consequence. Each branch can contain code, configuration, dependency or platform failures.

Severity. *Critical* denotes a credible path to cross-user disclosure, privileged control, financial-state compromise or irreversible material data loss. *High* denotes failure of an essential workflow, substantial availability or cost exposure, or exclusion from a core task. *Medium* denotes a bounded degradation whose consequences and recovery are understood. These are contextual release priorities, not CVSS scores. Exposure, data sensitivity and deployment scope can change an item’s level.

We also distinguish three origins. *Application failures* arise in the delivered code or configuration. *Inherited failures* arise in a platform, library or deployment service. *Development-agent failures* arise while an agent reads untrusted material or acts with tool permissions. A platform incident is evidence that a shared control failed, not proof that AI generated the faulty code.

5 Security

5.1 Credentials, identity and access

Secrets must be classified before scanning. A privileged API key in a browser bundle is available to the browser’s user. Repository history and generated assets can retain credentials after source files change. Secret management therefore includes inventory, restricted delivery and revocation after exposure (OWASP Foundation, n.d.f). [STRONG] A scan should cover both the repository and what is actually shipped. Gitleaks supports history and directory scanning, but pattern matching cannot prove that every credential was found (Gitleaks contributors, n.d.). [MODERATE]

Not every visible key is secret. Supabase explicitly distinguishes publishable client keys from server-side secret keys. The former identify the application and depend on database controls; the latter can bypass row-level security (RLS)

Table 3: Failure overview. Evidence strength concerns the stated mechanism or observation; severity is the lab's impact assessment.

Category	Failure	How it shows up	Severity	Evidence	Sources
Security	Secret exposure	Privileged credential in shipped files or repository history.	Critical	[STRONG]	(OWASP Foundation, n.d.f; Supabase, n.d.a)
Security	Broken access	A user reaches another user's objects or privileged actions.	Critical	[MODERATE]	(Deng et al., 2026; OWASP Foundation, n.d.b)
Security	Injection or unsafe files	Untrusted content reaches an interpreter, rendering sink or unsafe storage path.	High / critical	[STRONG]	(OWASP Foundation, n.d.e,n,n)
Supply chain	Untrusted packages	Dependency is nonexistent, unaudited or affected by a known advisory.	High	[MODERATE]	(Spracklen et al., 2025; TanStack, 2026)
Reliability	Bad state transitions	Lost updates, duplicate effects, failed edge cases or unreadable migrated data.	High / critical	[MODERATE]	(Liu et al., 2023; Prisma, n.d.; Amazon Web Services, n.d.a)
Payments	Event or entitlement errors	Untrusted or repeated notifications change paid access incorrectly.	Critical	[STRONG]	(Stripe, n.d.b,n)
Privacy / store	Undisclosed collection	Runtime behavior conflicts with notices or store declarations.	High	[STRONG]	(Apple, n.d.a; Google, n.d.a)
Operations	Unbounded or unseen work	Usage costs grow; failures have no accountable alert recipient.	High	[STRONG]	(OWASP Foundation, 2023, n.d.c)
Access / speed	Inaccessible or slow flows	Users cannot operate controls, or representative loads exceed performance targets.	High / medium	[STRONG]	(World Wide Web Consortium, 2024; Walton, 2024)

(Supabase, n.d.a). [STRONG] Calling every public key a breach misdiagnoses the problem. The check is whether the exposed credential grants authority that the client should not hold.

Login is only the identity boundary. Authentication establishes who a user is. Authorization decides which action that user may perform on which object. A hidden button or client-side route guard does not establish server-side permission. OWASP recommends denying unspecified access and evaluating authorization on every request (OWASP Foundation, n.d.b). [STRONG] Verification needs an explicit matrix of identities, resources and actions, including an anonymous session and two separate ordinary users. Each forbidden action must fail without disclosure or state change.

Recovery and session behavior also belong in the launch pass. Expired sessions, password recovery and changes to privileged account state must preserve the intended identity boundary (OWASP Foundation, n.d.a). [STRONG] Test these paths with dedicated fixtures; a successful initial login says little about them.

Database permissions are deployed behavior. For Supabase, inspect schema exposure, grants and RLS policies together. Disabling RLS does not alone prove public access, since grants and exposed schemas also matter. Conversely, enabling RLS without policies that enforce the intended owner or tenant rule does not establish isolation. Service-role credentials can bypass policies (Supabase, n.d.b). [STRONG] Inspect the deployed configuration and perform fixture-based checks using the same client role as the application, including denied access. Apply the same ownership reasoning to file storage.

5.2 Input, rendering, uploads and browser boundaries

Parameterized database operations keep values separate from executable query structure. Raw string construction remains hazardous even when most of the application uses an ORM. Where identifiers cannot be parameterized, their permitted set must be constrained (OWASP Foundation, n.d.e). [STRONG] Code review should follow each externally supplied value to its final query or interpreter boundary. Tests should assert safe handling and unchanged unrelated records without using attack payloads.

For cross-site scripting, the relevant distinction is between text and executable markup. Framework escaping can be bypassed by raw HTML APIs. Use context-appropriate encoding and maintained sanitization where rich content

is required; a content security policy is supplementary protection ([OWASP Foundation, n.d.i](#)). ^[STRONG] Treat model-generated text as untrusted if it is inserted into a page or another interpreter. Test rich-text, preview and error states, not only ordinary form submission.

Uploads require authorization, allowed formats, independent type validation, size limits and safe naming and serving. A filename extension or client-provided content type is insufficient ([OWASP Foundation, n.d.g](#)). ^[STRONG] Benign fixtures can establish that oversized files and disallowed formats are rejected and that private uploads cannot be read across fixture accounts. A successful upload does not demonstrate safe storage.

CORS controls browser access to cross-origin responses. It is not API authentication. A wildcard can be appropriate for intentionally public, noncredentialed content; credentialed responses need an allowed explicit origin under browser rules ([MDN contributors, n.d.](#)). ^[STRONG] Check the actual origin policy and retain endpoint authorization independently ([OWASP Foundation, n.d.d](#)). Do not treat permissive CORS as a universal diagnosis or its removal as a universal fix.

5.3 Dependencies and coding-agent permissions

Package hallucination creates a distinct review obligation. [Spracklen et al. \(2025\)](#) tested 16 models across 576,000 code samples and observed persistent recommendations of nonexistent packages. This establishes a potential dependency-confusion pathway, not that every suggested name was maliciously registered or installed. ^[MODERATE] Verify a new package against its official project, registry ownership and documentation before accepting it. Pin resolved versions and retain the lockfile.

OSV-Scanner and npm audit compare dependencies with known vulnerability information. That is useful evidence about reported flaws, but not proof that a new package is legitimate or benign ([OSV-Scanner contributors, n.d.](#); [npm contributors, n.d.](#)). ^[STRONG] A dependency update must also reach the deployed artifact. The TanStack Start advisory in September 2026 is an example of a framework fix that required affected deployments to be rebuilt and updated ([Linsley, 2026](#); [TanStack, 2026](#)). ^[STRONG]

An agent that edits files, runs commands and accesses credentials introduces a development-time boundary. Cursor's advisory for CVE-2025-54135 describes a prompt-injection chain that crossed an approval boundary through sensitive configuration changes; version 1.3.9 added approval requirements for those writes ([Cursor, 2025](#)). ^[STRONG] This is evidence about the tool, not a vulnerability rate for code it generated. Limit the agent's credentials and deployment authority, keep production data outside ordinary development tasks, and review consequential actions.

5.4 OWASP mappings and their scope

Table 4 maps mechanisms to the 2025 *web* and 2026 *LLM* lists ([OWASP Foundation, 2025b](#); [OWASP GenAI Security Project, 2026](#)). These are the lab's interpretive crosswalks, not official one-to-one assignments. Using AI to write an ordinary app does not make every defect an LLM vulnerability. An LLM code applies only when a model or agent is part of the relevant input, output, dependency, authority or resource path.

Table 4: Conditional OWASP crosswalk. Web codes are from 2025; LLM codes are from 2026.

Mechanism	Web Top 10	LLM Top 10, only where applicable
Privileged secret exposure	A02 Misconfiguration; A04 Cryptographic Failures when cryptographic key material is unprotected	LLM02 Sensitive Information Disclosure if model or agent context/output leaks the information.
Authentication, object authorization or open data rules	A07 Authentication Failures; A01 Broken Access Control; A02 Misconfiguration	LLM03 Excessive Agency for overly privileged model actions; LLM02 for model-mediated disclosure; LLM09 only for vector/embedding-specific weaknesses.
SQL injection, XSS or unsafe interpretation	A05 Injection	LLM10 Improper Output Handling when model output reaches an unsafe sink. LLM01 Prompt Injection applies to instruction manipulation, not ordinary SQL injection.

Table 4: Conditional OWASP crosswalk, continued.

Mechanism	Web Top 10	LLM Top 10, only where applicable
Unsafe file upload	A02 Misconfiguration; A05 Injection if content is interpreted	LLM01 only if uploaded content redirects a model; LLM10 if resulting model output is handled unsafely. Ordinary upload flaws have no automatic LLM mapping.
Unverified webhook	A08 Software or Data Integrity Failures; A07 if sender identity is not established	No direct mapping without a model-specific trust boundary.
Permissive cross-origin policy	A01 Broken Access Control; A02 Misconfiguration	No direct mapping for ordinary browser configuration.
Missing limits or runaway work	A06 Insecure Design; A10 Mishandling of Exceptional Conditions for cascading failure	LLM06 Unbounded Consumption for model or agent resource exhaustion. API4:2023 adds a more specific API classification.
Hallucinated or malicious dependency	A03 Software Supply Chain Failures	LLM04 Supply Chain if it enters the model/tool supply chain. A model's inaccurate package suggestion can involve LLM07 Misinformation; actual installation is a separate trust decision.
Unsafe agent instruction or authority	A06 Insecure Design when relevant to application permissions	LLM01 Prompt Injection and LLM03 Excessive Agency. LLM08 Hidden Context Exposure concerns hidden instructions/context, not every leaked application secret.
Missing detection and alerting	A09 Security Logging and Alerting Failures	Cross-cutting control gap; no standalone LLM category is forced onto it.

5.5 What public disclosures add

Disclosures establish concrete mechanisms and fixes, but usually lack a denominator for prevalence. Table 5 separates the affected surface from the inference the evidence permits.

Table 5: Documented cases. Reported exposure or a verified vulnerability does not by itself establish exploitation.

Case and date	Surface	Supported outcome and limitation
Base44, July 2025	Shared builder platform	Wiz verified an authentication flaw affecting private applications and its fix. Reported 9 July, fixed within 24 hours, disclosed 29 July. Wix reported no evidence of earlier exploitation (Nagli, 2025).
Lovable, April 2026	Builder/editor permissions	Vendor reports potential access to public-project chat history and source code from 3 February to 20 April. Private projects and Lovable Cloud were unaffected. Editor visibility and published-site access are distinct (Lovable, 2026).
TanStack Start, September 2026	Framework dependency	Maintainer advisory confirms reflected XSS, CVE-2026-102989. Lovable describes hosted mitigation and a patch requirement for external deployments. Its log review found no evidence of exploitation (TanStack, 2026 ; Moyal, 2026).
Cursor, August 2025	Development agent	CVE-2025-54135 documents an approval-boundary failure; fixed in 1.3.9. It does not establish a runtime vulnerability in generated apps (Cursor, 2025).

These accounts strongly support the existence of the named defects and the stated vendor responses. ^[STRONG] They provide limited evidence about occurrence outside their affected systems. None establishes that AI generation caused the disclosed flaw.

+ + +

6 Reliability and Correctness

Exercise state changes, not just screens. A test should specify the resulting record, permission, balance or notification, including what must remain unchanged. EvalPlus demonstrates that stronger test inputs uncover outputs that initially appeared correct (Liu et al., 2023). [MODERATE] For an application, the analogous launch pass includes empty data, boundary values, invalid input, repeated submission, concurrent edits and stale sessions. This application-level matrix is our synthesis, not a measured defect ranking.

Dependency failures should produce a defined state. Test denied permissions, unavailable services, malformed responses and interrupted requests in the application's own test environment. Playwright can intercept responses to make such cases repeatable (Microsoft, n.d.d). [STRONG] The pass condition is more than an error message. Partial work must not be represented as complete, a retry must not repeat an irreversible side effect, and the user must have an appropriate recovery path.

Retries can amplify overload, especially when several layers retry the same work. AWS's practitioner guidance recommends bounded attempts and appropriate backoff, and warns about repeated non-idempotent operations (Amazon Web Services, n.d.b). [MODERATE] Decide which operations are safe to repeat, preserve a durable operation identity where necessary, and inspect final state after a controlled duplicate or interruption.

Browser coverage is explicit. Automated Chromium, Firefox and WebKit checks provide useful diversity, while emulation can vary viewport, touch, locale and timezone (Microsoft, n.d.b,n). [STRONG] Emulation is not identical to an actual device. Exercise critical flows on supported physical devices, especially those involving the virtual keyboard, browser storage, downloads or native integrations. Record the tested matrix instead of claiming support for every device.

Schema changes need a recovery story. The expand-and-contract pattern separates adding a new representation, moving data and retiring the old representation. Prisma's official engineering guide illustrates this approach (Prisma, n.d.). [MODERATE] Review migration statements before execution, rehearse them on representative nonproduction data, compare invariants before and after, and check compatibility while old and new application versions coexist. An application rollback may fail if a migration already removed data the old version needs.

A backup is useful only if the relevant service can be reconstructed from it. AWS's practitioner guidance calls for restoration and integrity checks against stated recovery objectives (Amazon Web Services, n.d.a). [MODERATE] PostgreSQL tools can support a local rehearsal, but pg_dump exports one database and is not a universal production-backup strategy; cluster-wide objects and other storage require separate coverage (PostgreSQL Global Development Group, n.d.). [STRONG] Measure recovery time and acceptable data loss, and include uploads, credentials and configuration in the recovery inventory.

Operational reports show why these controls matter without establishing AI-specific prevalence. Lovable's January 2025 postmortem describes approximately 19 hours of builder disruption, inconsistent states during an emergency migration and paging that had not been enabled. It reports no lost projects, although some edits were not applied (Eriksson, 2025). [STRONG] This is a builder outage, not evidence that every hosted application went offline.

7 Payments, Privacy and Compliance

7.1 Payment events and durable entitlements

Payment correctness is a server-state problem. Verify the provider's signature over the prescribed payload with its maintained library and the appropriate endpoint secret. Stripe's documentation requires the raw request body and warns that notifications may be repeated or arrive out of order (Stripe, n.d.b). [STRONG] Verification, freshness checks and duplicate handling are different controls; other providers may authenticate different fields (OWASP Foundation, n.d.h).

A checkout success page is not proof that fulfillment occurred. Users can pay and never reach the return page. Stripe therefore documents server-side fulfillment and a handler that tolerates repeated or concurrent invocation (Stripe,

[n.d.a](#)). [STRONG] In sandbox testing, compare the provider’s payment state with the app’s persisted entitlement after interrupted redirects, repeated events and a later refund or cancellation. Prices, account ownership and eligibility must be decided from trusted server state. The exact transitions depend on the product’s business rules.

7.2 Data collection and accurate disclosures

The launch review should identify every collected field, its purpose, storage destination, recipients and deletion behavior. Apple’s guidelines include data minimization and privacy-policy requirements; Google Play’s User Data policy addresses expected purposes, transparency and handling of personal data ([Apple, 2026](#); [Google, n.d.c](#)). [STRONG] These are platform requirements, not evidence that a generic generated privacy policy accurately describes an app.

Third-party SDK behavior belongs in the same inventory. Apple’s privacy details and Google Play’s Data safety declarations include relevant third-party collection and sharing ([Apple, n.d.a](#); [Google, n.d.a](#)). [STRONG] Inspect the shipped dependency set and observed network destinations, then reconcile them with the notice and store forms. A claim that the app collects nothing cannot be accepted merely because the builder did not intentionally design analytics.

Where account creation triggers a platform deletion requirement, implement and test the applicable route. Apple’s and Google’s rules differ in detail, so use the current requirement for the intended distribution channel ([Apple, 2026](#); [Google, n.d.c](#)). [STRONG] Deletion and retention behavior should be explicit, including what remains temporarily in backups or is retained for a stated purpose. This report describes technical and platform requirements; it does not determine jurisdiction-specific legal duties or certify compliance.

7.3 Store readiness

Apple identifies crashes, broken links, placeholders, incomplete review information and unclear data-access requests among common review issues ([Apple, n.d.b](#)). Google Play disallows broken and insufficient functionality ([Google, n.d.b](#)). [STRONG] These sources establish review categories, not an AI-app rejection rate.

Use the exact release build for a final sign-in, purchase and recovery pass. Confirm that support and privacy links work, permission explanations match use, store screenshots describe real features, and reviewers can reach account-protected functionality with the supplied review information. Treat review acceptance and security verification as separate outcomes.

8

Cost and Operations

Bound work at the point of admission. Unrestricted APIs can exhaust resources or create substantial usage-based charges. OWASP’s API guidance identifies limits on payload size, execution time, batch size and paid downstream work ([OWASP Foundation, 2023](#)). [STRONG] For an LLM feature, bound input and output size, concurrent calls, tool invocations and retries. The 2026 LLM Top 10 identifies unbounded consumption as LLM06 ([OWASP GenAI Security Project, 2026](#)). A user-facing cooldown without a corresponding server limit is not sufficient enforcement.

Rate, concurrency and spending limits solve different problems. A slow permitted request can still be expensive, and a small per-user allowance can multiply across many users. Set limits per operation, account and service where appropriate, and verify the refusal path using low quotas in an isolated fixture environment. Keep rejection graceful and observable.

Read the provider’s actual billing control. An alerts-only Google Cloud budget sends notifications without automatically stopping usage ([Google Cloud, 2026a](#)). Its newer spend-cap budgets can pause eligible service usage within a project, but enforcement is not instantaneous, in-flight work may accrue costs, and persistent resources may keep charging ([Google Cloud, 2026b](#)). [STRONG] Neither the word budget nor a warning email establishes an absolute global ceiling. Record scope, enforcement delay, reset behavior and an application-level stop mechanism.

Detection requires a delivered signal. Google’s SRE practitioner guidance organizes monitoring around latency, traffic, errors and saturation, with alerts that support action ([Ewaschuk, 2016](#)). [MODERATE] A small application also needs visibility into rejected logins, denied access, failing payment events and unusually expensive requests. Avoid recording secrets, access tokens or unnecessary personal data in diagnostic output ([OWASP Foundation, n.d.c](#)). [STRONG]

Before launch, deliberately trigger a safe test failure and confirm that the intended person receives a useful alert. Record the release version, affected operation and a short response procedure. A dashboard that nobody watches or an alert destination that never received a test is incomplete operational evidence.

9 Accessibility and Performance

9.1 Common barriers and complementary checks

The WebAIM Million is a practitioner measurement report about the general web, not AI-built applications. Its February 2026 popularity-derived sample found automatically detectable WCAG failures on 95.9% of the sampled homepages. Low contrast, missing image alternatives, missing form labels, empty links, empty buttons and missing document language accounted for 96% of detected errors ([WebAIM, 2026](#)). [MODERATE] These are useful inspection priorities, but the figures cannot establish an AI-specific accessibility rate or whole-application conformance.

WCAG 2.2 provides testable criteria for keyboard operation, focus, labels, contrast, text alternatives, error identification and other behaviors ([World Wide Web Consortium, 2024](#)). [STRONG] Automated tools can find some structural defects. Playwright's accessibility guide demonstrates axe integration and recommends combining it with manual assessment ([Microsoft, n.d.a](#)). W3C explicitly states that automated tools alone cannot determine conformance ([World Wide Web Consortium, 2026](#)). [STRONG]

The manual pass should complete the main journey without a mouse, follow focus through dialogs, enlarge content, and use a screen reader to inspect names and status changes. Include validation errors, loading states and success messages. A labeled empty form is not evidence that an error can be located or that completion is announced.

9.2 Performance is measured in a representative state

Core Web Vitals define good thresholds of LCP at most 2.5 seconds, INP at most 200 milliseconds and CLS at most 0.1, assessed at the 75th percentile separately for mobile and desktop ([Walton, 2024](#)). [STRONG] LCP concerns major content appearing, INP interaction response, and CLS unexpected layout movement. These are field targets. A pre-launch lab run cannot establish the future field distribution.

Lighthouse is a free automated audit tool that can inspect authenticated pages and support regression checking ([Google, 2025](#)). [STRONG] Use representative content, a constrained device and network profile, and the interactions users actually need. Inspect excessive assets, long tasks and layout movement that the run exposes. Declare a performance budget and prepare field measurement for after launch. A high score on an empty landing page does not establish acceptable performance of a populated, interactive product.

10 A Prevention Framework

The framework treats a release as a set of claims that require evidence. Its priorities come from plausible impact and observability, not from an unavailable global ranking of AI-app losses. Appendix A contains the complete 36-item pass, including what to check, how to check it and the expected result.

F1. *Functional success and security success require separate evidence. A working screen cannot establish ownership checks, safe interpretation or denial of forbidden actions.*

Independent functional and security evaluations support this distinction ([Vero et al., 2025](#); [Zhao et al., 2026](#)). [STRONG] Write acceptance criteria for permitted and forbidden state changes before accepting implementation.

F2. *The highest-priority checks follow authority and durable state. Inspect credentials, identity, object permissions, payment transitions and recoverability before releasing consequential features.*

This ordering combines access-control guidance, provider contracts and recovery practice ([OWASP Foundation, n.d.b](#); [Stripe, n.d.a](#); [Amazon Web Services, n.d.a](#)). [MODERATE] Severity should rise when a failure crosses users, spends money or removes the ability to recover.

F3. *Different checks observe different failure classes. Combine source and dependency review, configuration inspection, behavioral tests and manual interaction.*

Passive ZAP baseline scans observe accessible responses; dependency scanners match known advisories; accessibility automation detects only some barriers ([ZAP Development Team, n.d.](#); [OSV-Scanner contributors, n.d.](#); [World Wide Web Consortium, 2026](#)). [STRONG] Passing one does not substitute for the others.

F4. *The release evidence must include limits, restoration and a delivered alert. Prevention is incomplete when the operator cannot contain or detect a failure.*

Provider billing limits have scope and delay, while recovery requires an actual restoration check ([Google Cloud, 2026b](#); [Amazon Web Services, n.d.a](#)). [MODERATE] A test of these controls is more informative than the mere presence of their configuration.

F5. *Prompts can specify obligations, but the implementation must enforce them and independent checks must confirm them.*

Repair studies report useful but incomplete correction after feedback ([Fu et al., 2025](#); [Belozarov et al., 2026](#)); recent benchmark reminders also leave unresolved failures ([Wu et al., 2026](#)). [MODERATE] A model’s statement that it reviewed its work is not a substitute for observed results.

10.1 What the checks can and cannot establish

Table 6: A compact verification portfolio. What a check establishes is restricted to the inspected version, configuration and exercised cases.

Method	Can establish	Does not establish
Repository and bundle secret scan	Known credential patterns appear in the scanned files or history.	Absence of every secret or revocation of an exposed key (Gitleaks contributors, n.d. ; OWASP Foundation, n.d.f).
Dependency audit	Resolved versions match known vulnerability records.	Package legitimacy or absence of unknown malicious behavior (OSV-Scanner contributors, n.d. ; npm contributors, n.d.).
Access matrix and fixture tests	Declared identities receive expected decisions for tested actions.	Exhaustiveness across every endpoint or future policy change (OWASP Foundation, n.d.b).
State-transition tests	Tested failure, retry and payment sequences end in the expected state.	Correctness under all schedules or untested provider behavior (Microsoft, n.d.d ; Stripe, n.d.b).
Passive response scan	Some response and configuration issues are observable.	Complete business-logic or authorization assessment (ZAP Development Team, n.d.).
Automated and manual UI review	Observed barriers and regressions in tested states.	Universal usability or complete conformance from automation alone (World Wide Web Consortium, 2026 ; Google, 2025).

10.2 A release decision that can be reviewed

Keep a compact evidence record for the candidate build. It should name the code revision, dependency lockfile, deployment configuration, tester, date, affected checklist IDs and result. A passing result includes the observed state or a redacted report. A failed result identifies impact and the next action. A not-applicable result explains why the feature or boundary is absent. Do not silently interpret a skipped test as a pass.

Algorithm 1 Pre-launch verification pass

1. Freeze the release candidate and list users, data, paid actions and external services.
2. Translate those boundaries into allowed and forbidden state transitions.
3. Assign applicable checklist items, severity, an owner and a pass condition.
4. Inspect shipped assets, dependency versions and deployed permissions.
5. Run fixture-based functional, authorization, failure and payment checks.
6. Perform browser/device, manual accessibility and representative performance checks.
7. Rehearse migration, restoration, limits and one delivered operational alert.
8. Review evidence independently of the builder's completion claim.
9. **if** an applicable Critical or High item lacks passing evidence **then**
10. Repair and retest, or remove the affected capability from the release.
11. **else**
12. Record any bounded residual issues with an owner and correction date.
13. Deploy the reviewed candidate and perform nondestructive smoke checks.
14. **end if**
15. Retain evidence and repeat affected checks after consequential changes.

The severity policy is deliberately asymmetric. Unverified privileged access, financial-state changes or recovery should block the relevant release. A bounded cosmetic defect can be recorded for follow-up. A critical accessibility barrier that prevents the core task belongs with high-impact functional failures, regardless of how polished the page appears.

For a small team, independent review means checking against a specification and observed output that did not come solely from the same generation step. Another person can supply it, but so can a carefully designed fixture with an external oracle, such as the expected owner, provider payment state or restored record count. This recommendation is a verification design principle; its overall effectiveness for vibe-coded apps remains unquantified. [LIMITED]

11 Prompting for Prevention

Give the builder concrete constraints before it selects an architecture. Request the affected files, configuration and test evidence, and require uncertainty to be reported. These example lines are a specification aid synthesized from the controls above. They are not validated prompt templates.

Credentials and permissions.

Keep privileged credentials out of client assets and repository history. Define an anonymous/user/admin and owner/-tenant access matrix, enforce it server-side and provide fixture tests for every denied action.

Input, uploads and dependencies.

Use parameterized queries, safe text rendering and explicit upload type, size and access rules. Verify each new dependency against its official project, retain a lockfile and report unresolved advisories.

Agents and runtime models.

Treat repository content, retrieved text, uploads and model output as untrusted data. Restrict tools and credentials to the required scope, and require review before privileged configuration or production changes.

Correctness and recovery.

Specify final state for empty, invalid, repeated, concurrent and failed requests, then test those outcomes. Provide a reviewed migration, compatibility plan and a restoration rehearsal using nonproduction data.

Payments.

Verify provider events with the official signature routine and correct payload bytes. Make fulfillment durable and safe under duplicate or out-of-order events, and verify entitlement state independently of the checkout redirect.

Privacy and store readiness.

Collect only fields with a stated product purpose, and inventory third-party SDK data flows. Make disclosures, deletion behavior, permissions, store metadata and review access match the release build.

Cost and operations.

Enforce server-side limits on requests, payloads, model work, retries and concurrency. Separate budget alerts from enforced caps, redact sensitive logs and demonstrate a received failure alert and a usable stop mechanism.

Accessibility and performance.

Target WCAG 2.2 AA in all core states and preserve keyboard, focus and screen-reader behavior. Run automated and manual checks plus performance tests with representative data on a constrained device profile.

Use these lines to request inspectable work. [Belozarov et al. \(2026\)](#) found incomplete repair even with modern models; their dataset also contained user-supplied vulnerable code, so it cannot be described as entirely AI-introduced. [MODERATE] Security reminders and self-review may improve some results, but they cannot grant permissions safely, enforce a quota or prove a backup usable by themselves.

12 Limitations

The source set is broad but purposive. Search rankings, English-language coverage and publicly accessible material shape it. We did not inspect private builder telemetry, reproduce the cited experiments, audit the disclosed systems or measure this checklist's detection rate. The search log supports retracing selections; it does not make the corpus exhaustive.

Much empirical work targets security-relevant tasks or identifiable public code. Tool fingerprints can miss real AI use and include projects whose human review was substantial. Model versions, agent permissions, frameworks and scanners change. Direct comparisons across studies can therefore be misleading, and vendor incident accounts may not expose every root cause or affected user.

The evidence for global frequency is particularly weak outside security. Official payment contracts, accessibility standards and store policies support concrete checks, but do not measure how commonly AI builders violate them. WebAIM's general-web results are not a substitute for an AI-specific sample. We found no basis for ranking worldwide financial losses, claiming that all vibe-coded apps share the same failures, or declaring this checklist optimal.

The checklist targets small web and mobile applications with typical hosted services. Safety-critical, regulated, highly sensitive or unusually distributed systems need additional domain-specific assurance. A launch pass also ages. Changes to dependencies, permissions, schemas, models or business rules require renewed checks. These limits constrain the claim of this report to a practical, evidence-linked starting point.

13 Conclusion

The defensible lesson is to demand evidence at the boundaries a demonstration does not exercise. Separate functionality from security, verify who can act on which data, reconcile payment and application state, bound costly work, and rehearse recovery and alert delivery. Combine automated inspection with deliberate behavioral and manual checks. The 36-item appendix turns those obligations into a reviewable launch pass while preserving what the available research cannot yet establish.

14 AI Disclosure

These research papers were ideated by the researchers at Stargazing Creativities Lab, generated by artificial intelligence, and human revised.

References

- Amazon Web Services. Rel09-bp04 perform periodic recovery of the data to verify backup integrity and processes. AWS Well-Architected Framework, n.d.a. URL https://docs.aws.amazon.com/wellarchitected/latest/framework/rel_backing_up_data_periodic_recovery_testing_data.html. Accessed 22 October 2026.
- Amazon Web Services. Rel05-bp03 control and limit retry calls. AWS Well-Architected Reliability Pillar, n.d.b. URL https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/rel_mitigate_interaction_failure_limit_retries.html. Accessed 22 October 2026.
- Anthropic. Claude 3.7 sonnet and claude code. Anthropic, 2025. URL <https://www.anthropic.com/news/claude-3-7-sonnet>. Official vendor announcement, 24 February 2025. Accessed 22 October 2026.
- Apple. App review guidelines. Apple Developer, 2026. URL <https://developer.apple.com/app-store/review/guidelines/>. Accessed 22 October 2026; revision announced 16 October 2026.
- Apple. App privacy details. Apple Developer, n.d.a. URL <https://developer.apple.com/app-store/app-privacy-details/>. Accessed 22 October 2026.
- Apple. App review. Apple Developer, n.d.b. URL <https://developer.apple.com/distribute/app-review/>. Accessed 22 October 2026.
- Vladislav Belozarov, Peter J Barclay, and Ashkan Sami. Secure coding with AI – from detection to repair. *Empirical Software Engineering*, 31:93, 2026. doi: 10.1007/s10664-026-10812-8. URL <https://link.springer.com/article/10.1007/s10664-026-10812-8>. Article 93. Published 13 March 2026. Accessed 22 October 2026.
- Cursor. Arbitrary code execution from Cursor Agent through a prompt injection via MCP Special Files. GitHub Security Advisories, 2025. URL <https://github.com/cursor/cursor/security/advisories/GHSA-4cxx-hrm3-49rm>. Published 2025-08-02. Accessed 22 October 2026.
- Junquan Deng, Zhiyu Fan, and Ruijie Meng. Understanding the (in)security of vibe-coded applications. arXiv:2606.23130v4, 2026. URL <https://arxiv.org/abs/2606.23130v4>. Preprint, revised 14 September 2026. Accessed 22 October 2026.
- Viktor Eriksson. Incident Jan 2: Github outage. Lovable blog, 2025. URL <https://lovable.dev/blog/incident-github-outage>. Published 2025-01-06. Accessed 22 October 2026.
- Rob Ewaschuk. Monitoring distributed systems. In Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy, editors, *Site Reliability Engineering: How Google Runs Production Systems*, chapter 6. O'Reilly, 2016. URL <https://sre.google/sre-book/monitoring-distributed-systems/>. Practitioner source. Accessed 22 October 2026.
- Nat Friedman. Introducing GitHub Copilot: Your AI pair programmer. The GitHub Blog, 2021. URL <https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/>. Official vendor announcement, 29 June 2021. Accessed 22 October 2026.
- Yujia Fu, Peng Liang, Amjed Tahir, Zengyang Li, Mojtaba Shahin, Jiaxin Yu, and Jinfu Chen. Security weaknesses of Copilot-generated code in GitHub projects: An empirical study. *ACM Transactions on Software Engineering and Methodology*, 2025. URL <https://arxiv.org/abs/2310.02059>. Accepted manuscript, arXiv:2310.02059v4. Accessed 22 October 2026.
- Gitleaks contributors. Gitleaks. GitHub repository and README, n.d. URL <https://github.com/gitleaks/gitleaks>. Accessed 22 October 2026.
- Google. Introduction to Lighthouse. Chrome for Developers, 2025. URL <https://developer.chrome.com/docs/lighthouse/overview>. Accessed 22 October 2026.
- Google. Provide information for Google Play's data safety section. Google Play Console Help, n.d.a. URL <https://support.google.com/googleplay/android-developer/answer/10787469?hl=en>. Accessed 22 October 2026.
- Google. Functionality, content, and user experience. Google Play Console Help, n.d.b. URL <https://support.google.com/googleplay/android-developer/answer/9898783?hl=en>. Accessed 22 October 2026.
- Google. User data. Google Play Console Help, n.d.c. URL <https://support.google.com/googleplay/android-developer/answer/10144311?hl=en>. Accessed 22 October 2026.
- Google Cloud. Create, edit, or delete budgets and budget alerts. Cloud Billing Documentation, 2026a. URL <https://docs.cloud.google.com/billing/docs/how-to/budgets>. Accessed 22 October 2026.
- Google Cloud. Manage spend cap budgets. Cloud Billing Documentation, 2026b. URL <https://docs.cloud.google.com/billing/docs/how-to/budgets-spend-caps>. Accessed 22 October 2026.
- Tanner Linsley. TanStack Start security update: CVE-2026-102989. TanStack Blog, 2026. URL <https://tanstack.com/blog/tanstack-start-security-update-cve-2026-102989>. Published 2026-09-30. Accessed 22 October 2026.
- Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html. Accessed 22 October 2026.

- Lovable. Our response to the April 2026 incident. Lovable blog, 2026. URL <https://lovable.dev/blog/our-response-to-the-april-2026-incident>. Published 2026-04-22. Accessed 22 October 2026.
- Vahid Majdinasab, Michael Joshua Bishop, Shawn Rasheed, Arghavan Moradidakhel, Amjed Tahir, and Foutse Khomh. Assessing the security of GitHub Copilot generated code – a targeted replication study. arXiv:2311.11177, 2023. URL <https://arxiv.org/abs/2311.11177>. Preprint. Accessed 22 October 2026.
- MDN contributors. Cross-Origin Resource Sharing (CORS). MDN Web Docs, n.d. URL <https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS>. Accessed 22 October 2026.
- Microsoft. Accessibility testing. Playwright Documentation, n.d.a. URL <https://playwright.dev/docs/accessibility-testing>. Accessed 22 October 2026.
- Microsoft. Browsers. Playwright Documentation, n.d.b. URL <https://playwright.dev/docs/browsers>. Accessed 22 October 2026.
- Microsoft. Emulation. Playwright Documentation, n.d.c. URL <https://playwright.dev/docs/emulation>. Accessed 22 October 2026.
- Microsoft. Mock apis. Playwright Documentation, n.d.d. URL <https://playwright.dev/docs/mock>. Accessed 22 October 2026.
- Talia Moyal. A vulnerability in TanStack Start: what we found and what we did to protect your apps. Lovable blog, 2026. URL <https://lovable.dev/blog/how-lovable-protects-your-app-from-a-tanstack-start-vulnerability>. Published 2026-09-30. Accessed 22 October 2026.
- Gal Nagli. Wiz Research Uncovers Critical Vulnerability in AI Vibe Coding platform Base44 Allowing Unauthorized Access to Private Applications. Wiz Research, 2025. URL <https://www.wiz.io/blog/critical-vulnerability-base44>. Published 2025-07-29. Accessed 22 October 2026.
- npm contributors. npm-audit. npm CLI documentation, n.d. URL <https://docs.npmjs.com/cli/v11/commands/npm-audit/>. Accessed 22 October 2026.
- OSV-Scanner contributors. Project Source Scanning. OSV-Scanner documentation, n.d. URL <https://google.github.io/osv-scanner/usage/scan-source>. Accessed 22 October 2026.
- OWASP Foundation. OWASP Web Security Testing Guide, Version 4.2. OWASP WSTG, 2020. URL <https://wstg.owasp.org/v4.2/>. Published 2020-12-03. Accessed 22 October 2026.
- OWASP Foundation. API4:2023 Unrestricted Resource Consumption. OWASP API Security Top 10, 2023. URL <https://api-security.owasp.org/editions/2023/en/0xa4-unrestricted-resource-consumption/>. Accessed 22 October 2026.
- OWASP Foundation. OWASP Application Security Verification Standard 5.0.0. OWASP ASVS, 2025a. URL <https://asvs.owasp.org/>. Published 2025-05-30. Accessed 22 October 2026.
- OWASP Foundation. OWASP Top 10:2025. OWASP Top 10, 2025b. URL <https://top10.owasp.org/2025/>. Accessed 22 October 2026.
- OWASP Foundation. Authentication Cheat Sheet. OWASP Cheat Sheet Series, n.d.a. URL https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html. Accessed 22 October 2026.
- OWASP Foundation. Authorization Cheat Sheet. OWASP Cheat Sheet Series, n.d.b. URL https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html. Accessed 22 October 2026.
- OWASP Foundation. Logging Cheat Sheet. OWASP Cheat Sheet Series, n.d.c. URL https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html. Accessed 22 October 2026.
- OWASP Foundation. REST Security Cheat Sheet. OWASP Cheat Sheet Series, n.d.d. URL https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html. Accessed 22 October 2026.
- OWASP Foundation. SQL Injection Prevention Cheat Sheet. OWASP Cheat Sheet Series, n.d.e. URL https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html. Accessed 22 October 2026.
- OWASP Foundation. Secrets Management Cheat Sheet. OWASP Cheat Sheet Series, n.d.f. URL https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html. Accessed 22 October 2026.
- OWASP Foundation. File Upload Cheat Sheet. OWASP Cheat Sheet Series, n.d.g. URL https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html. Accessed 22 October 2026.
- OWASP Foundation. Webhook Security Cheat Sheet. OWASP Cheat Sheet Series, n.d.h. URL https://cheatsheetseries.owasp.org/cheatsheets/Webhook_Security_Cheat_Sheet.html. Accessed 22 October 2026.
- OWASP Foundation. Cross Site Scripting Prevention Cheat Sheet. OWASP Cheat Sheet Series, n.d.i. URL https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html. Accessed 22 October 2026.
- OWASP GenAI Security Project. OWASP Top 10 for LLM Applications 2026. OWASP GenAI Security Project, 2026. URL <https://genai.owasp.org/resource/owasp-genai-llm-top-10-2026/>. Published 2026-08-03. Accessed 22 October 2026.
- Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. Asleep at the keyboard? assessing the security of GitHub Copilot’s code contributions. In *IEEE Symposium on Security and Privacy*, 2022. URL <https://arxiv.org/abs/2108.09293>. Accepted venue verified on the author’s arXiv record. Accessed 22 October 2026.

- Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. Do users write more insecure code with AI assistants? In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2785–2799, 2023. doi: 10.1145/3576915.3623157. URL <https://arxiv.org/abs/2211.03622>. Full version v3. Accessed 22 October 2026.
- PostgreSQL Global Development Group. `pg_dump`. PostgreSQL 18 Documentation, n.d. URL <https://www.postgresql.org/docs/current/app-pgdump.html>. Accessed 22 October 2026.
- Prisma. Expand-and-contract migrations. Prisma Documentation, n.d. URL <https://www.prisma.io/docs/guides/database/data-migration>. Accessed 22 October 2026.
- Gustavo Sandoval, Hammond Pearce, Teo Nys, Ramesh Karri, Siddharth Garg, and Brendan Dolan-Gavitt. Lost at C: A user study on the security implications of large language model code assistants. In *32nd USENIX Security Symposium*, pages 2205–2222. USENIX Association, 2023. URL <https://www.usenix.org/conference/usenixsecurity23/presentation/sandoval>. Accessed 22 October 2026.
- Advait Sarkar and Ian Drosos. Vibe coding: Programming through conversation with artificial intelligence. In *Proceedings of the 36th Annual Conference of the Psychology of Programming Interest Group*, 2025. URL <https://arxiv.org/abs/2506.23253>. Version 2, 3 October 2025. Accessed 22 October 2026.
- Joseph Spracklen, Raveen Wijewickrama, A H M Nazmus Sakib, Anindya Maiti, Bimal Viswanath, and Murtuza Jadliwala. We have a package for you! a comprehensive analysis of package hallucinations by code generating LLMs. In *34th USENIX Security Symposium*, pages 3687–3706. USENIX Association, 2025. URL <https://www.usenix.org/conference/usenixsecurity25/presentation/spracklen>. Accessed 22 October 2026.
- Stripe. Fulfill orders. Stripe Documentation, n.d.a. URL <https://docs.stripe.com/checkout/fulfillment?payment-ui=checkout-form>. Accessed 22 October 2026.
- Stripe. Receive Stripe events in your webhook endpoint. Stripe Documentation, n.d.b. URL <https://docs.stripe.com/webhooks>. Accessed 22 October 2026.
- Supabase. API keys. Supabase Docs, n.d.a. URL <https://supabase.com/docs/guides/getting-started/api-keys>. Accessed 22 October 2026.
- Supabase. Row Level Security. Supabase Docs, n.d.b. URL <https://supabase.com/docs/guides/database/postgres/row-level-security>. Accessed 22 October 2026.
- TanStack. Unauthenticated reflected XSS in TanStack Start server-function responses. GitHub Security Advisories, 2026. URL <https://github.com/TanStack/router/security/advisories/GHSA-qx66-fv34-fjm8>. Published 2026-09-30. Accessed 22 October 2026.
- The Replit Team. Introducing Replit Agent. Replit, 2024. URL <https://replit.com/blog/introducing-replit-agent>. Official vendor announcement, 16 September 2024; updated 25 April 2025. Accessed 22 October 2026.
- Mark Vero, Niels Mündler, Victor Chibotaru, Veselin Raychev, Maximilian Baader, Nikola Jovanović, Jingxuan He, and Martin Vechev. BaxBench: Can LLMs generate correct and secure backends? In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 61344–61390. PMLR, 2025. URL <https://proceedings.mlr.press/v267/vero25a.html>. Accessed 22 October 2026.
- Philip Walton. Web vitals. web.dev, 2024. URL <https://web.dev/articles/vitals>. Accessed 22 October 2026; published 4 May 2020, updated 31 October 2024.
- WebAIM. The webaim million: The 2026 report on the accessibility of the top 1,000,000 home pages. WebAIM research report, 2026. URL <https://webaim.org/projects/million/>. Accessed 22 October 2026; practitioner measurement report.
- World Wide Web Consortium. Web content accessibility guidelines (WCAG) 2.2. W3C Recommendation, 12 December 2024, 2024. URL <https://www.w3.org/TR/WCAG22/>. Accessed 22 October 2026.
- World Wide Web Consortium. Evaluating web accessibility overview. W3C Web Accessibility Initiative, 2026. URL <https://www.w3.org/WAI/test-evaluate/>. Accessed 22 October 2026.
- Tianyi Wu, Chao Ge, Peng Chen, Fengji Zhang, Yongxing Wu, Yuchong Xie, Jian Li, Chaoyuan Peng, Kaikai Zhang, Dong Huang, Mingzhe Du, Yukai Huang, Jiaheng Zhang, Terry Yue Zhuo, and See-Kiong Ng. Last secure code benchmark: Is the code you vibe secure? arXiv:2610.14766v1, 2026. URL <https://arxiv.org/abs/2610.14766v1>. Preprint, 10 October 2026. Accessed 22 October 2026.
- ZAP Development Team. ZAP - Baseline Scan. ZAP documentation, n.d. URL <https://www.zaproxy.org/docs/docker/baseline-scan/>. Accessed 22 October 2026.
- Songwen Zhao, Danqing Wang, Kexun Zhang, Jiaxuan Luo, Zhuo Li, and Lei Li. Is vibe coding safe? benchmarking vulnerability of agent-generated code in real-world tasks. In *International Conference on Machine Learning*, 2026. URL <https://arxiv.org/abs/2512.03262v4>. Accepted paper; arXiv version 4, 21 September 2026. Accessed 22 October 2026.

A The Pre-Launch Checklist

This standalone release worksheet combines 36 documented requirements and the lab's recommended verification steps for applications built with AI assistance. It is not a certification scheme. No number of checked boxes proves security, correctness, accessibility or compliance.

Rank by consequence. CRITICAL findings block launch because they can expose privileged access, other users' data, payment authority or irreplaceable state. HIGH findings block the affected flow until repaired or removed. MEDIUM findings require an owner, target date and accepted operating limit. These are contextual engineering priorities, not measured frequencies or standardized vulnerability scores. Raise severity when the affected data, users or business dependency warrant it. Item numbers are stable lookup identifiers, not an empirical ordering within a tier.

Prepare and record. Use owned local or staging environments, synthetic accounts, disposable data and provider sandboxes. Production smoke checks must be nondestructive. These methods use free tools or manual inspection without paid scanning services; remote infrastructure may still incur charges. Never print real credentials. For every item record pass, fail or not applicable, the artifact and configuration version, date, owner and redacted evidence. Explain every not-applicable decision. Repeat affected checks after dependency, schema, permission or deployment changes.

A.1 Security and trust boundaries

01. Secrets in history and deployed artifacts

CRITICAL

Check. Privileged credentials are absent from repository history, browser bundles, source maps and public configuration. Distinguish intended publishable keys from secrets.

Verify. Run Gitleaks on Git history and exported build directories with redacted output; manually inspect client configuration and deployment variables. Record scanned paths and exclusions, and triage findings. Pass only when exposed credentials have been revoked or rotated at their provider and rebuilt public artifacts contain no privileged key ([Gitleaks contributors, n.d.](#); [OWASP Foundation, n.d.f](#); [Supabase, n.d.a](#)).

02. Authentication, sessions and recovery

HIGH

Check. Login, logout, expiry and account recovery enforce server-side identity rules. A visible login screen does not constitute protection.

Verify. Use synthetic accounts to complete recovery, expire a session and log out, then exercise protected actions through the application's test harness. Check that recovery links are bounded and invalidated sessions cannot authorize new actions. Pass requires explicit denial where identity is missing or invalid ([OWASP Foundation, n.d.a](#)).

03. Tenant and object authorization

CRITICAL

Check. Every read, write, export and deletion enforces the intended owner or tenant boundary on the server.

Verify. Create two synthetic users with separate records and a role, resource and action matrix. Exercise permitted and forbidden combinations with integration tests or a manual harness, including list and export paths. Pass when permitted operations succeed and every forbidden case returns no protected data and causes no state change ([OWASP Foundation, n.d.b](#)).

04. Administrative privileges

CRITICAL

Check. Ordinary users cannot perform administrative actions, assign stronger roles or retain authority after role revocation.

Verify. Inventory privileged routes and background actions, including settings, invitations, exports and account management. In staging, test each with an administrator, ordinary account and revoked administrator. Inspect server-side role checks rather than relying on hidden buttons. Pass requires the documented role matrix to hold for direct service calls and the interface ([OWASP Foundation, 2020](#)).

05. Database grants, row policies and storage

CRITICAL

Check. Exposed tables and stored objects have intentional grants and row-level security (RLS) rules. Privileged server credentials never enter a client.

Verify. Inspect schema exposure, grants, row-level policies and storage permissions with database tools or the service dashboard. Test allowed and denied operations with anonymous and low-privilege synthetic users. Pass requires the combined grants and policies to enforce the intended boundary; merely enabling RLS is insufficient ([Supabase, n.d.b,n](#)).

06. Query construction

HIGH

Check. Untrusted values cannot become executable query structure, including in raw queries beneath an object-relational mapper (ORM).

Verify. Review database call sites and trace form, URL and imported-data fields to queries. Require parameter binding for values and allowlists where identifiers cannot be bound. Use boundary and invalid-type fixtures to check rejection and error handling. Pass requires reviewed query construction and passing tests, not merely a clean scan ([OWASP Foundation, n.d.e](#)).

07. Browser rendering and model output

HIGH

Check. User content, imported content and runtime model responses are rendered safely for their destination context.

Verify. Review template escapes, rich-text paths and APIs interpreting HTML or script. Use harmless markup-character fixtures and confirm they remain text where appropriate. When HTML is required, verify a maintained sanitizer and its configuration. Inspect error and preview states too. Pass requires context-appropriate handling; a content-security policy alone does not establish it ([OWASP Foundation, n.d.i](#)).

08. File uploads and retrieval

HIGH

Check. Uploads have justified type and size limits, independent content checks, authorization and safe storage and serving behavior.

Verify. Submit small valid fixtures, mismatched types and boundary-sized files. Inspect generated names, storage location and retrieval permissions with two synthetic accounts. Pass when rejected files remain unavailable, accepted files cannot become executable application content, and another user cannot retrieve private uploads. Keep limits and outcomes, not sensitive test content ([OWASP Foundation, n.d.g](#)).

09. Cross-origin policy

HIGH

Check. Credentialed browser access is limited to intended origins, while deliberately public resources have an explicit policy.

Verify. Inspect cross-origin resource sharing (CORS) middleware and headers, then use local pages at allowed and unapproved origins to confirm the policy. Verify endpoint authorization separately because CORS governs browser access, not general API identity. Pass when credentialed responses use approved origins and no broad exception exposes private data ([MDN contributors, n.d.](#); [OWASP Foundation, n.d.d](#)).

10. Package identity and provenance

HIGH

Check. Each new dependency is the intended maintained project, with a recorded origin and resolved version.

Verify. Before installation, compare registry listings, official project links, maintainers and release history. Review the lockfile and inspect registry signatures or provenance where supported, for example with `npm audit signatures`. Justify unfamiliar packages. Pass requires resolved identity and reviewed dependency changes; a plausible generated name or clean advisory scan is insufficient ([Spracklen et al., 2025](#); [npm contributors, n.d.](#)).

11. Known advisories and deployed fixes

HIGH

Check. The release contains reviewed fixes for relevant known vulnerabilities, including transitive dependencies and builder frameworks.

Verify. Run OSV-Scanner or the ecosystem's free advisory tool against resolved dependencies. Triage reachability and severity, apply reviewed upgrades, rebuild and redeploy, then repeat core flows. Pass requires the deployed artifact's version or manifest to confirm the fix; editing a local manifest alone does not patch production ([OSV-Scanner contributors, n.d.](#); [Linsley, 2026](#)).

12. Coding-agent and runtime-tool authority

CRITICAL

Check. Coding agents and runtime large language model (LLM) tools have only necessary permissions, with server-enforced limits on sensitive actions.

Verify. Inspect tools, credentials, writable locations and approval rules. Keep development agents away from production authority. In a disposable fixture, verify that an untrusted document cannot authorize broader tool actions and sensitive configuration changes require approval. Pass requires an enforced boundary and current patches; a prompt promising restraint is insufficient ([Cursor, 2025](#); [OWASP GenAI Security Project, 2026](#)).

A.2 Reliability and recoverable state

13. Complete user journeys

HIGH

Check. Promised journeys include real persistence and subsequent retrieval, including account and purchase flows where present.

Verify. With a fresh synthetic account, create, read, edit and delete representative records, reload the application and inspect backend state. Automate repeatable paths in Playwright or document manual screenshots and state checks. Pass when visible success agrees with stored results and every required step works without developer intervention ([Microsoft, n.d.d](#); [Apple, n.d.b](#)).

14. Boundary and unusual inputs

HIGH

Check. Declared input limits, empty states and supported locales behave consistently without corruption or misleading success.

Verify. Derive fixtures and expected results from the specification, including missing values, boundary lengths, Unicode, numeric limits, empty results and relevant dates or timezones. Exercise client and server validation with a local test runner or manual forms. Pass when each case is handled deliberately and persisted values match the specification ([OWASP Foundation, n.d.d](#); [Microsoft, n.d.c](#)).

15. Dependency failures and retry bounds

HIGH

Check. Failed or slow dependencies produce truthful, recoverable outcomes without unbounded waits or retry amplification.

Verify. Use Playwright response mocks or local fixtures for timeouts, unavailability and invalid responses. Inspect timeout and retry settings, including software development kit (SDK) defaults and retries at multiple layers. Pass when the interface preserves recoverable work, reports failure, stops within the declared bound and avoids duplicate side effects. Measure stopping times ([Microsoft, n.d.d](#); [Amazon Web Services, n.d.b](#)).

16. Concurrent and repeated actions

HIGH

Check. Repeated submissions and overlapping updates preserve stated invariants, including uniqueness and balances where relevant.

Verify. Use two browser sessions or a local fixture to submit the same action twice and update one record concurrently. Inspect stored outcomes after refresh and handler restart. Pass when duplication is prevented or safely reconciled and conflicts are visible to the user. Verify the invariant, not only the returned HTTP status ([Amazon Web Services, n.d.b](#)).

17. Supported browsers and devices

HIGH

Check. Core journeys work across the declared browser, viewport and mobile platform combinations.

Verify. Run Playwright projects for Chromium, Firefox and WebKit with relevant device, touch and permission settings. Manually repeat critical mobile flows on physical devices, including login, file selection and keyboard use. Document tested versions and coverage gaps. Pass requires the declared support matrix to work; emulation alone does not establish behavior on every device ([Microsoft, n.d.b,n](#)).

18. Schema migration rehearsal

CRITICAL

Check. Schema changes preserve required records, relationships and deployment compatibility. Destructive conversions are explicitly reviewed.

Verify. Migrate a disposable database containing representative existing data. Compare important values, counts, constraints and application behavior before and after. Where versions overlap, test both against the intermediate schema and retain old structures until unused. Pass requires preservation checks and a feasible recovery plan, not merely successful migration execution ([Prisma, n.d.](#)).

19. Restoration from backup

CRITICAL

Check. The service can recover required data, attachments and configuration within declared downtime and data-loss limits.

Verify. Restore a recent backup separately using free database tools or a manual provider workflow. Open representative records, validate relationships and verify external-object coverage. Record elapsed time and the newest recovered state. Pass requires usable restored data within declared objectives; an existing backup file or isolated database dump is insufficient ([PostgreSQL Global Development Group, n.d.](#); [Amazon Web Services, n.d.a](#)).

20. Rollback and configuration recovery

HIGH

Check. A failed release has a recovery path covering application code, schema compatibility, environment variables and external integrations.

Verify. In staging, redeploy the retained previous artifact or rehearse the intended forward fix, then test core journeys and integration callbacks. Compare required configuration without exposing secret values. Pass when the documented procedure restores service and identifies changes that cannot safely be reversed. Assign a recovery decision owner ([Eriksson, 2025](#)).

A.3 Payments, privacy and distribution

21. Signed payment events

CRITICAL

Check. Payment state changes require authentic provider events verified with the correct endpoint secret and exact signed payload.

Verify. Use the provider's official verification library and sandbox tooling. Confirm middleware passes raw request bytes to verification. Run valid, missing-signature and invalid-signature fixtures against the owned endpoint. Pass when only valid events reach business processing and rejected fixtures leave orders and entitlements unchanged. Keep event identifiers, never signing secrets ([Stripe, n.d.b](#); [OWASP Foundation, n.d.h](#)).

22. Duplicate and reordered events

HIGH

Check. Repeated, delayed or reordered payment events cannot duplicate fulfillment or regress payment state.

Verify. Use provider sandbox redelivery and local fixtures to exercise the same event before and after handler restart, plus relevant ordering changes. Inspect durable processing records and entitlements. Pass requires the intended transition exactly once where appropriate, with recoverable unfinished work. Arrival order or timestamps alone cannot establish correct processing ([Stripe, n.d.b](#)).

23. Authoritative payment and entitlement state

CRITICAL

Check. The server links customer, product, amount and currency to verified payment state before granting access.

Verify. Compare successful, failed and delayed sandbox payments against order expectations. Close checkout before redirect and confirm server-side reconciliation. Exercise promised cancellation or refund behavior. Pass when entitlements match verified orders and payment outcomes, including repeated checks, rather than a client flag or success-page visit ([Stripe, n.d.a](#)).

24. Necessary data collection

HIGH

Check. Collected fields, requested permissions and external transfers serve specified features with appropriate scope.

Verify. Inventory forms, schema, SDKs, analytics and model requests. Remove unnecessary collection, test alternatives when permissions are declined, and review whether accounts are needed. Pass when the inventory matches observed behavior and every retained category has a documented purpose, access rule and retention decision ([Apple, 2026](#); [Google, n.d.c](#)).

25. Accurate disclosures and SDK inventory

HIGH

Check. Disclosures and store declarations match actual data handling, including third-party SDKs and services.

Verify. Compare test-flow network destinations and data categories with the privacy policy, Apple privacy details and Google Data safety answers. Consult SDK documentation for hidden behavior. Pass when discrepancies are resolved and an owner can update declarations after changes. A generated policy template alone is insufficient ([Apple, n.d.a](#); [Google, n.d.a](#)).

26. Deletion and retention behavior

HIGH

Check. Deletion and retention controls match disclosed behavior and applicable distribution requirements.

Verify. Delete a synthetic account through the published path. Check primary records, connected services and access revocation. Document justified retention exceptions and backup expiry. Test any required external deletion-request route. Pass when results match the disclosed lifecycle and users receive an accurate explanation of remaining retention ([Google, n.d.c](#); [Apple, 2026](#)).

27. Store submission readiness

MEDIUM

Check. The submitted build is complete, functional and reviewable under current product and regional distribution rules.

Verify. Install the release build, check every link, remove placeholders, and verify screenshots, descriptions, support contacts and privacy links. Supply working reviewer access where needed and check applicable payment rules. Pass when reviewers can exercise promised flows without undocumented setup. Store approval is distinct from technical assurance ([Apple, n.d.b](#); [Google, n.d.b](#); [Apple, 2026](#)).

A.4 Usage costs and operations**28. Request and resource limits**

HIGH

Check. Endpoints bound costly work, including uploads, page sizes, batches and repeated requests.

Verify. Inspect server limits and seed local counters near thresholds. Send small synthetic requests below and above each bound, confirming rejection before expensive processing. Check anonymous and signed-in use where both exist. Pass when ceilings produce predictable errors and ordinary alternate routes cannot bypass them ([OWASP Foundation, 2023](#)).

29. Model and agent execution budgets

HIGH

Check. Runtime features bound input and output tokens, tool calls, concurrency, duration and retries per request and user.

Verify. Review provider and application limits, then use mocked model responses and tool results to test stopping conditions without a large bill. Check accounting across retries and resumed jobs. Pass when execution stops at declared limits with a useful outcome. Natural-language stop instructions are not enforcement ([OWASP GenAI Security Project, 2026](#)).

30. Billing alerts and enforcement semantics

HIGH

Check. Owners distinguish notifications from enforced limits, including scope and delays.

Verify. Inspect current provider documentation and thresholds. Document covered services, reset behavior, excluded charges and recipients. Where supported, test enforcement with a small sandbox or local accounting fixture. Pass requires an actionable response and bounded exposure. Google Cloud alerts-only budgets do not cap spend; service-scoped spend caps retain documented exceptions and delays ([Google Cloud, 2026a,b](#)).

31. Useful and redacted telemetry

HIGH

Check. Logs and metrics reveal failures and usage without unnecessary credentials, payment details or sensitive content.

Verify. Trigger staging successes and failures. Inspect event fields, correlation identifiers, access controls and retention. Check visibility into latency, traffic, errors, saturation and relevant billable usage. Pass when operators can identify the affected flow and version from redacted evidence without recovering private request contents or secrets ([OWASP Foundation, n.d.c](#); [Ewaschuk, 2016](#)).

32. Delivered alerts and emergency controls

HIGH

Check. A named owner receives actionable alerts and can suspend failing or costly features.

Verify. Trigger a harmless staging error or threshold fixture and confirm delivery. Rehearse feature disabling, verify new work stops safely, and test recovery. Measure alert receipt time and observed behavior. Pass requires a completed drill with a documented control; an unwatched dashboard or untested notification setting is insufficient ([Ewaschuk, 2016](#); [Eriksson, 2025](#)).

A.5 Accessibility and performance**33. Automated accessibility checks**

HIGH

Check. Key pages and dynamic states are checked for detectable contrast, labeling, naming and structural problems.

Verify. Run free axe-core with Playwright or Lighthouse on menus, dialogs and form-error states. Fix confirmed failures and manually inspect text alternatives and undecided findings. Pass when confirmed barriers are fixed and uncertain findings assessed; a clean automated report does not establish full accessibility conformance ([Microsoft, n.d.a](#); [Google, 2025](#); [World Wide Web Consortium, 2026](#)).

34. Keyboard, focus and screen-reader journeys

HIGH

Check. Critical tasks work without a pointer, with understandable controls, errors and state changes.

Verify. Check keyboard navigation, visible focus, logical order and dialog entry and exit. Repeat the journey with VoiceOver or NVDA, inspecting labels and announcements. Pass when tasks remain operable and understandable without trapped focus or inaccessible required actions. Document tested devices and assistive technologies ([World Wide Web Consortium, 2024, 2026](#)).

35. Performance under constrained conditions

MEDIUM

Check. Representative paths meet explicit loading and interaction budgets under slower device and network conditions.

Verify. Use Lighthouse and browser performance tools with CPU/network constraints, realistic data and authenticated states. Inspect oversized assets, long tasks and repeated requests. Pass when recorded measurements meet budgets and regressions are reviewed. Document conditions; a fast development machine or empty database can hide bottlenecks ([Google, 2025](#)).

36. Field measurement and regression readiness

MEDIUM

Check. The release can measure real-user performance and detect regressions without unnecessary personal data.

Verify. Review metrics, sampling and privacy treatment; verify collection with synthetic visits. Retain lab baselines and future-build budgets. Pass when measurements have an owner and action threshold. Plan separate mobile and desktop 75th-percentile Core Web Vitals evaluation; pre-launch synthetic runs cannot establish field percentiles ([Walton, 2024](#); [Apple, n.d.a](#)).

B Search Log

The search and source-verification date was **22 October 2026**. This appendix records all 47 query strings in 33 logged search batches, including searches with no retained evidence. S1 and S2 denote the two search-service engines, `system1_search_query` and `system2_search_query`. Queries within an academic batch were submitted together; the retained set is associated with the batch, not attributed to a particular query. The log makes a targeted narrative survey reproducible at the level of search strings and selection decisions. Ranked results can change, and no exhaustive database search or reconstructed hit count is claimed.

B.1 Searches and screening decisions

The literal search strings appear in monospaced text. A date filter is recorded only when used. Retained items were opened before inclusion. Social posts and derivative summaries served only as leads; selected sources and rejected claims are identified below.

Table 7: Full query log. All rows were searched on 22 October 2026.

ID / engine	Exact query strings, retained sources and screening decisions
A1 S2	AI generated code security empirical study Pearce Perry Sandoval Copilot "We Have a Package for You" package hallucinations authors vibe coding Karpathy February 2 2025 definition IBM AI generated code security empirical study 2026 survey October Retained. (Pearce et al., 2022; Sandoval et al., 2023; Spracklen et al., 2025; Perry et al., 2023; Fu et al., 2025; Belozero- rov et al., 2026). IBM definition opened but not selected because academic Sarkar source better supports exact origin date. News, Reddit and encyclopedia results were leads only.
A2 S1	site.arxiv.org 2026 security generated code empirical "vibe" "Vibe coding: programming through conversation with artificial intelligence" Sarkar "Is Your Code Generated by ChatGPT Really Correct" NeurIPS 2023 "Secure coding with AI" "from detection to repair" First query filter: previous 365 days. Retained. (Sarkar and Drosos, 2025; Liu et al., 2023; Belozero- v et al., 2026; Deng et al., 2026; Wu et al., 2026). Current-date coverage found June and October 2026 primary preprints. Opened full HTML for sampling and limitations. Unverified social claims excluded.
A3 S2	site.github.blog "Introducing GitHub Copilot" "June 29, 2021" site.lovable.dev "November" "2024" "launch" Lovable site.anthropic.com "Claude Code" "Feb 24, 2025" site.replit.com "Introducing Replit Agent" "September 5, 2024" Retained. (Friedman, 2021; Anthropic, 2025). Exact Replit September 5 date not established by official page, so not used. Lovable history unnecessary; app-builder example uses Replit official announcement.
A4 S2	site.blog.replit.com agent September 2024 introducing site.stackblitz.com bolt.new launch October 3 2024 "Assessing the Security of GitHub Copilot Generated Code" "Targeted Replication Study" "SWE-bench" "security" "61%" "10.5%" Retained. (The Replit Team, 2024; Majdinasab et al., 2023). Opened Replit official announcement and replication arXiv. Bolt exact launch not verified in official source; omitted.
A5 S2	"BaxBench" "secure" arxiv "SusVibes" arxiv "61%" "10.5%" "secure" agents Retained. (Vero et al., 2025; Zhao et al., 2026). Opened PMLR and latest arXiv v4. Rejected stale v1 statistics (61%/10.5%, 200 tasks) in secondary summaries.
SEC-Q01 S1	site:owasp.org Top 10 2025 A01 A10 2026 Filter: previous 730 days. Retained. (OWASP Foundation, 2025b). Current 2025 web edition verified directly; did not use 2021 numbering.
SEC-Q02 S1	site:genai.owasp.org LLM Top 10 2025 2026 risk Filter: previous 730 days. Retained. (OWASP GenAI Security Project, 2026). Archive still displays 2025 but official 2026 resource/PDF and Sep 2026 announcement verified. New numbering used.

Continued on next page

Table 7 continued

ID / engine	Exact query strings, retained sources and screening decisions
SEC-Q03 S1	site:wiz.io Base44 critical vulnerability July 2025 authentication Retained. (Nagli, 2025). Primary discoverer report selected; no breach prevalence inferred.
SEC-Q04 S1	site:cursor.com security advisory CVE 2025 prompt injection Retained. None from this search. Forum leads not cited; followed later exact-CVE search to official vendor advisory.
SEC-Q05 S1	site:msrc.microsoft.com CVE-2025-53773 Copilot Retained. None from this search. MSRC page returned JavaScript shell; NVD pages returned no substantive text. Case omitted in favor of readable Cursor vendor advisory.
SEC-Q06 S1	site:lovable.dev security incident row level security 2025 Retained. (Eriksson, 2025; Lovable, 2026; Moyal, 2026). Followed official blog index to dated postmortems and 2026 framework disclosure. Marketing prevalence numbers were not retained.
SEC-Q07 S1	site:nvd.nist.gov CVE-2025-54135 Cursor Retained. None from this search. NVD returned empty/down page; no claims taken from it. Replaced with vendor GHSA.
SEC-Q08 S2	"CVE-2025-54135" "github.com" Retained. (Cursor, 2025). Mirrors used only to locate vendor GHSA, then vendor source opened.
SEC-Q09 S2	"CVE-2025-53773" Microsoft Security Response Center GitHub advisory Retained. None from this search. Unreviewed mirror/aggregator results not retained; accessible vendor Cursor case preferred.
SEC-Q10 S2	site:cursor.com "CVE-2025" security Retained. None from this search. Forum results not included as evidence.
O01 S1	site:docs.stripe.com webhooks signature verify duplicate events retry Retained. (Stripe, n.d.b). Third-party webhook tutorials replaced with Stripe documentation.
O02 S1	site:developer.apple.com app review guidelines completeness privacy data collection Retained. (Apple, 2026, n.d.b,n). Third-party review-advice blogs not used.
O03 S1	site.support.google.com googleplay android developer user data privacy policy data safety Retained. (Google, n.d.c,n). Privacy-policy vendors and social posts not used.
O04 S1	site.w3.org WCAG 2.2 Recommendation 2024 Retained. (World Wide Web Consortium, 2024). Wikipedia and translations omitted because the normative English standard was available.
O05 S1	site.webaim.org projects million 2026 95.9 Retained. (WebAIM, 2026). Derivative 2026 accessibility posts and LinkedIn/Facebook commentary not used.
O06 S1	site.web.dev articles vitals LCP 2.5 INP 200 CLS 0.1 Retained. (Walton, 2024). SEO/marketing explainers not used; one gave an incorrect 3-second LCP threshold.
O07 S1	site.w3.org WAI evaluation tools cannot determine accessibility Retained. (World Wide Web Consortium, 2026). W3C Selecting Web Accessibility Evaluation Tools was opened as corroboration but duplicate for bibliography.
O08 S1	site.playwright.dev docs emulation devices browsers API mocking Retained. (Microsoft, n.d.c,n,n,n). Third-party emulation tutorials not used; browser/accessibility guides were followed directly from official documentation.
O09 S1	site.prisma.io docs expand contract migrations data migration Retained. (Prisma, n.d.). Forum questions and ORM-version-specific instructions excluded; current page explicitly labels ORM 8 as release candidate.

Continued on next page

Table 7 continued

ID / engine	Exact query strings, retained sources and screening decisions
O10 S1	site.postgresql.org docs backup dump restore pg_dump Retained. (PostgreSQL Global Development Group, n.d.). Unsupported version 8.1 docs and Stack Overflow tutorials not used. Backup and Restore chapter opened for context.
O11 S1	site.aws.amazon.com builders library timeouts retries backoff jitter Retained. (Amazon Web Services, n.d.b). Original Builders' Library article was opened but newer official Reliability Pillar is the selected concise source.
O12 S1	site.cloud.google.com billing budgets does not cap usage spending Retained. (Google Cloud, 2026a,b). Old blanket no-cap advice rejected after current official docs showed spend-cap budgets.
O13 S1	site.sre.google sre book monitoring distributed systems four golden signals Retained. (Ewaschuk, 2016). Vendor summaries of SRE golden signals omitted in favor of authored original chapter.
O14 S1	site.developer.chrome.com docs lighthouse overview open source Retained. (Google, 2025). Third-party Lighthouse introductions omitted.
O15 S1	site.docs.stripe.com checkout fulfillment webhooks can't rely landing page Retained. (Stripe, n.d.a). Base page initially returned UI option links; checkout-form variant was opened to read full text.
O16 S1	site.docs.aws.amazon.com wellarchitected test data recovery backups regularly Retained. (Amazon Web Services, n.d.a). Marketing backup pages and general disaster-recovery tutorials not needed.
O17 S1	site.support.google.com/googleplay/android-developer broken functionality limited functionality policy Retained. (Google, n.d.b). Community support threads and app-submission agency advice omitted.
O18 S1	site.sre.google/sre-book 2016 O'Reilly copyright monitoring Ewaschuk Retained. (Ewaschuk, 2016). Second-edition announcement (12 October 2026) opened, but no claims imported from marketing announcement; citation remains to original authored 2016 chapter.

B.2 Source verification register

For each retained entry, the title, authors or corporate author, year, and venue or URL were checked against the opened source. The bibliography supplies the full title and canonical URL. Table 8 records source type and discovery route; every listed source was opened. “Direct” means navigation to official documentation rather than a new search query. The accompanying audit/source-audit.json preserves exact queries, metadata, opened URLs, version notes and claim limitations without reproducing source text.

Undated living documentation is cited as n.d. Explicit revision years are distinguished from original publication years. The taxonomy is frozen to OWASP web Top 10:2025 and LLM Top 10:2026. Other version anchors are ASVS 5.0.0, WSTG 4.2 and the 12 December 2024 WCAG 2.2 Recommendation. ArXiv versions are identified where results changed. Subsequent direct re-openings for citation checks are verification repeats, not additional searches.

Table 8: Opened-source register. Bibliography entries provide the verified titles and URLs.

Record	Source and evidence type	Discovery route
V01	Pearce et al. (2022). Research publication.	A1
V02	Perry et al. (2023). Research publication.	A1
V03	Sandoval et al. (2023). Research publication.	A1
V04	Fu et al. (2025). Research publication.	A1
V05	Spracklen et al. (2025). Research publication.	A1
V06	Liu et al. (2023). Research publication.	A2
V07	Sarkar and Drosos (2025). Research publication.	A2
V08	Deng et al. (2026). Research preprint.	A2

Continued on next page

Table 8 continued

Record	Source and evidence type	Discovery route
V09	Wu et al. (2026) . Research preprint.	A2
V10	Belozarov et al. (2026) . Research publication.	A1, A2
V11	Vero et al. (2025) . Research publication.	A5
V12	Zhao et al. (2026) . Research publication.	A5
V13	Majdinasab et al. (2023) . Research preprint.	A4
V14	Friedman (2021) . Vendor announcement.	A3
V15	The Replit Team (2024) . Vendor announcement.	A4
V16	Anthropic (2025) . Vendor announcement.	A3
V17	OWASP Foundation (2025b) . Awareness taxonomy.	SEC-Q01
V18	OWASP GenAI Security Project (2026) . Awareness taxonomy.	SEC-Q02
V19	Nagli (2025) . Primary researcher disclosure.	SEC-Q03
V20	Lovable (2026) . Vendor postmortem.	SEC-Q06
V21	Linsley (2026) . Vendor advisory.	Direct
V22	TanStack (2026) . Vendor advisory.	Direct
V23	Moyal (2026) . Vendor disclosure.	SEC-Q06
V24	Cursor (2025) . Vendor advisory.	SEC-Q08
V25	Eriksson (2025) . Vendor postmortem.	SEC-Q06
V26	Supabase (n.d.b) . Official documentation.	Direct
V27	Supabase (n.d.a) . Official documentation.	Direct
V28	OWASP Foundation (n.d.f) . Official security guidance.	Direct
V29	OWASP Foundation (n.d.b) . Official security guidance.	Direct
V30	OWASP Foundation (n.d.a) . Official security guidance.	Direct
V31	OWASP Foundation (n.d.e) . Official security guidance.	Direct
V32	OWASP Foundation (n.d.i) . Official security guidance.	Direct
V33	OWASP Foundation (n.d.g) . Official security guidance.	Direct
V34	MDN contributors (n.d.) . Browser documentation.	Direct
V35	OWASP Foundation (n.d.d) . Official security guidance.	Direct
V36	OWASP Foundation (2023) . Awareness taxonomy.	Direct
V37	OWASP Foundation (2025a) . Verification standard.	Direct
V38	OWASP Foundation (2020) . Testing guidance.	Direct
V39	Gitleaks contributors (n.d.) . Official tool documentation.	Direct
V40	ZAP Development Team (n.d.) . Official tool documentation.	Direct
V41	npm contributors (n.d.) . Official tool documentation.	Direct
V42	OSV-Scanner contributors (n.d.) . Official tool documentation.	Direct
V43	OWASP Foundation (n.d.c) . Official security guidance.	Direct
V44	OWASP Foundation (n.d.h) . Official security guidance.	Direct
V45	Stripe (n.d.b) . Official documentation.	O01
V46	Stripe (n.d.a) . Official documentation.	O15
V47	Apple (2026) . Official documentation.	O02
V48	Apple (n.d.b) . Official documentation.	O02
V49	Apple (n.d.a) . Official documentation.	O02
V50	Google (n.d.c) . Official documentation.	O03
V51	Google (n.d.a) . Official documentation.	O03
V52	Google (n.d.b) . Official documentation.	O17
V53	World Wide Web Consortium (2024) . Normative accessibility standard.	O04
V54	World Wide Web Consortium (2026) . Official documentation.	O07
V55	WebAIM (2026) . Practitioner measurement report.	O05

Continued on next page

Table 8 continued

Record	Source and evidence type	Discovery route
V56	Walton (2024) . Official documentation.	O06
V57	Google (2025) . Official documentation.	O14
V58	Microsoft (n.d.c) . Official documentation.	O08
V59	Microsoft (n.d.b) . Official documentation.	O08
V60	Microsoft (n.d.d) . Official documentation.	O08
V61	Microsoft (n.d.a) . Official documentation.	O08
V62	Prisma (n.d.) . Official documentation.	O09
V63	PostgreSQL Global Development Group (n.d.) . Official documentation.	O10
V64	Amazon Web Services (n.d.b) . Official documentation.	O11
V65	Amazon Web Services (n.d.a) . Official documentation.	O16
V66	Google Cloud (2026a) . Official documentation.	O12
V67	Google Cloud (2026b) . Official documentation.	O12
V68	Ewaschuk (2016) . Practitioner book chapter.	O13, O18

B.3 Corroborating reads and retrieval limits

The source register includes retained references only. The following additional official pages were opened to check metadata or resolve ambiguity; they are not counted as new query strings.

Table 9: Additional direct reads used for corroboration.

Official page	Purpose
Apple Developer news W3C, Selecting Web Accessibility Evaluation Tools	Checked the 16 October 2026 App Review Guidelines revision. Corroborated the need for human evaluation; the overview is the retained reference.
AWS Builders' Library, Timeouts, retries, and backoff with jitter	Original practitioner article inspected; current Reliability Pillar guidance retained.
PostgreSQL 18, Backup and Restore	Checked backup families and context for the retained <code>pg_dump</code> documentation.
Google Research, Site Reliability Engineering book record	Verified the 2016 book, editors and publisher.
Google Cloud, SRE book second-edition announcement	Checked the 12 October 2026 announcement; retained claims remain tied to the original authored chapter.
OWASP, LLM Top 10 2026 download and release announcement	Read the linked PDF and corroborated the 2026 edition. The PDF revision-history date was a placeholder; the dated resource page supplies the citation year.
OWASP, WSTG release-versions page	Verified that version 4.2 was released on 3 December 2020.
Lovable blog index	Located dated disclosures and verified article bylines.

NVD records for CVE-2025-54135 and CVE-2025-53773 returned no substantive content. The Microsoft Security Response Center page for CVE-2025-53773 returned a JavaScript application shell. No claims were taken from those failed reads. The Cursor case instead uses its opened vendor advisory. The ASVS PDF could not be parsed through either attempted download URL. Its official landing page supports the version, release date and purpose cited here; no unseen numbered requirement is quoted.

The log rejects universal vulnerability percentages, causal claims that AI authored a platform or framework flaw, unsupported loss totals, stale benchmark values and claims that a clean scanner result or a prompt guarantees security. It also rejects blanket statements that every visible Supabase key is secret, every wildcard CORS policy is unsafe, all cloud budgets are alerts only, or automated accessibility checks establish conformance. These decisions preserve the scope and limitations of the retained sources.