RESEARCH PAPER · APPLICATION SECURITY · AI-ASSISTED SOFTWARE ENGINEERING

Vibe Coding and the Security Debt Crisis

What the evidence says about the security of AI-generated software, what has gone wrong in production, and a layered framework for fixing it

Maher Bhatt

maherbhatt.me · linkedin.com/in/maher-bhatt-206035362 · github.com/Maher-Bhatt

Version 1.0 · 1 November 2026 · Evidence reviewed through 31 October 2026

Abstract

"Vibe coding" is building software by describing it to an AI model and accepting the result without reading the code. Since Andrej Karpathy named the practice in February 2025 [1], it has moved from weekend experiments to products that hold real customer data. This paper reviews the empirical security evidence on AI-generated code from 2022 to October 2026, covering controlled studies, vendor benchmarks, repository telemetry and documented production incidents.

The evidence points in one direction. Models now write code that compiles and runs almost every time, but security has lagged: in Veracode's benchmark, the share of AI-generated code samples that pass security checks was 55% in 2025, 56% in mid-2026, and 62% in the October 2026 update, still leaving 38% of tasks vulnerable [2, 3, 4]. Users of AI assistants in a controlled Stanford study wrote less secure code and were more confident in it [5]. In production, the failures are rarely exotic. They are missing database access rules, an identifier mistaken for a credential, an agent holding credentials it should never have had, and instructions that were treated as security controls.

I organise these failures into a taxonomy, examine seven documented cases, trace six root causes, and propose **VIBE-SAFE**, a five-layer defence-in-depth framework that moves enforcement out of the prompt and into deterministic controls. A practical pre-launch checklist for builders closes the paper. VIBE-SAFE is a proposal from this review; it has not itself been empirically validated, and the paper says so wherever it matters.

Keywords: vibe coding, AI-generated code security, security debt, row-level security, prompt injection, Model Context Protocol, package hallucination, agentic engineering, OWASP, CWE.

Key findings

- **Security has not kept pace with capability.** AI models pass functional checks almost every time, yet 38% of Veracode's October 2026 security tasks still fail. Cross-site scripting passes only 18% of the time and log injection 13% [4].
- **People over-trust the output.** In a controlled study of 47 participants, those using an AI assistant produced less secure solutions on four of five tasks and were more likely to believe their code was secure [5].
- **Most real incidents are basic control failures, not clever exploits.** Missing row-level security, public identifiers used as credentials, and over-privileged agents explain the major cases reviewed here.
- **Instructions are not controls.** An agent told to freeze all changes still deleted a production database [6]. Enforcement has to live in permissions, proxies and gates.
- **Responsibility does not transfer to the tool.** Regulation in the EU and India places security duties on the organisation that deploys software, however it was written [7, 8].

1. Introduction

In January 2023 Karpathy observed that "the hottest new programming language is English." On 2 February 2025 he described a way of working in which you "fully give in to the vibes" and "forget that the code even exists" [1]. The phrase spread quickly and was named Collins Dictionary's Word of the Year for 2025 [9]. Academic work has since characterised the practice as programming through a conversational loop: state a goal, prompt a model, run the result, judge it by how it looks and behaves, and re-prompt [10].

By April 2026 Karpathy was drawing a line between vibe coding, which raises the floor for people who could not previously build software, and *agentic engineering*, in which professional engineers direct AI agents while keeping responsibility for specification, review, evaluation and security [11]. The distinction matters for this paper. The security problem appears when the first practice is used where the second is required: an unreviewed prototype is put in front of real users and real data.

Adoption is broad. Stack Overflow's 2025 survey found 84% of respondents using or planning to use AI tools in development, while 46% said they distrust the accuracy of the output [12]. Two tool families dominate: AI-native editors and command-line agents (for example Cursor, GitHub Copilot, Claude Code, and Amazon Q Developer), and prompt-to-app platforms (for example Lovable, Bolt, Replit and Base44) that bundle generation, hosting, authentication and a managed database.

Security debt here means unresolved weaknesses that accumulate in deployed systems faster than they are found and fixed. AI-assisted development can raise the rate of accumulation in two ways: it generates more code per unit time, and it can reduce the human attention each unit of code receives.

Contributions. (1) A sourced review of the evidence, with claims graded by evidence type; (2) a taxonomy that maps recurring failure modes to current OWASP and CWE categories; (3) seven case studies, with clear statements of what is established and what is only reported; (4) the VIBE-SAFE framework; and (5) a pre-launch checklist.

1.1 From vibe coding to agentic engineering

Reporting of Karpathy's conversation at Sequoia's AI Ascent 2026 (30 April, according to the report) has him separating the two practices by kind, not degree. Vibe coding raises the floor for everyone who wants to build. Agentic engineering is the disciplined use of agents by people who keep responsibility for the result, and it depends on four skills: specification design, diff review, evaluation design and security oversight. He is quoted as saying that "you can outsource your thinking, but you can't outsource your understanding" [11]. Table 1 summarises how the two terms are used in this paper; the right-hand column is my synthesis, not a quotation.

Dimension	Vibe coding	Agentic engineering (author's synthesis)
Intent	Describe the outcome in natural language; judge by how the running app looks.	Written specification, acceptance criteria and evaluations agreed before generation.
Code review	None or surface-level; code treated as an opaque artefact.	Diffs reviewed, with extra scrutiny on permissions, schema and dependency changes.
Context given to the model	Ad hoc, usually about interface and features.	Explicit business, repository and security context kept in version control.
Verification	Click through a preview; paste errors back into the prompt.	Automated tests, scanners and gates, plus human review.
Credentials and environment	Often shared between development and production.	Separate environments; task-scoped, short-lived credentials.
Typical failure	Exposed data, open endpoints, destructive agent actions.	Prompt injection into privileged agents; compromise of the tools themselves.

Dimension	Vibe coding	Agentic engineering (author's synthesis)
Fitting use	Throwaway prototypes, personal tools, demos on synthetic data.	Software with real users, real data or regulatory duties.

Table 1. How the two practices are used in this paper. The distinction follows the usage reported in [11]; the comparison itself is the author's.

2. Scope and evidence standard

This is a **structured narrative review**, not a PRISMA-style systematic review. I did not run a registered protocol, a fixed search string across databases, or dual-reviewer screening, and the paper does not claim the completeness that those methods provide. Sources were collected from peer-reviewed venues (IEEE S&P, ACM CCS, USENIX Security, COLM, arXiv), industry telemetry, vulnerability registries (NVD, vendor security bulletins), researcher disclosures and incident databases, covering January 2022 to 31 October 2026.

Each claim is graded by the strength of its source, and the text says which grade applies:

Grade	Source type	How it is used here
A	Peer-reviewed study or primary registry record (paper, CVE record, vendor bulletin)	Stated as fact, with the study's limits.
B	Vendor benchmark or telemetry with a published method summary	Stated with attribution; vendor incentives and sampling noted.
C	Researcher disclosure or credible press report of an incident	Stated as "reported" unless corroborated by the affected company or a record.
Excluded	Unsourced statistics, aggregator blogs, claims I could not trace to a primary source	Left out. Several widely repeated figures were dropped on this basis.

Table 2. Evidence grades used in this paper.

Two cautions apply throughout. First, several sources are commercial and have an interest in the problem they measure. Second, many incidents are single cases; they illustrate mechanisms and do not measure prevalence.

3. What the evidence shows

3.1 Controlled studies: assistants change security outcomes and confidence

Pearce et al. prompted GitHub Copilot with 89 security-relevant scenarios, generating 1,689 programs, and found roughly 40% of them vulnerable [13]. That study measured the model alone.

Perry et al. measured people. In a user study at Stanford (47 participants after exclusions: 33 with an AI assistant, 14 controls) built on OpenAI's codex-davinci-002, participants with the assistant wrote significantly less secure code on four of five tasks, and were more likely to rate their own code as secure [5]. Figure 1 shows the share of secure solutions per task. The authors also found that participants who trusted the AI less and who actively reworded prompts produced fewer vulnerabilities, and that unsafe randomness was the most common error in the signing task.

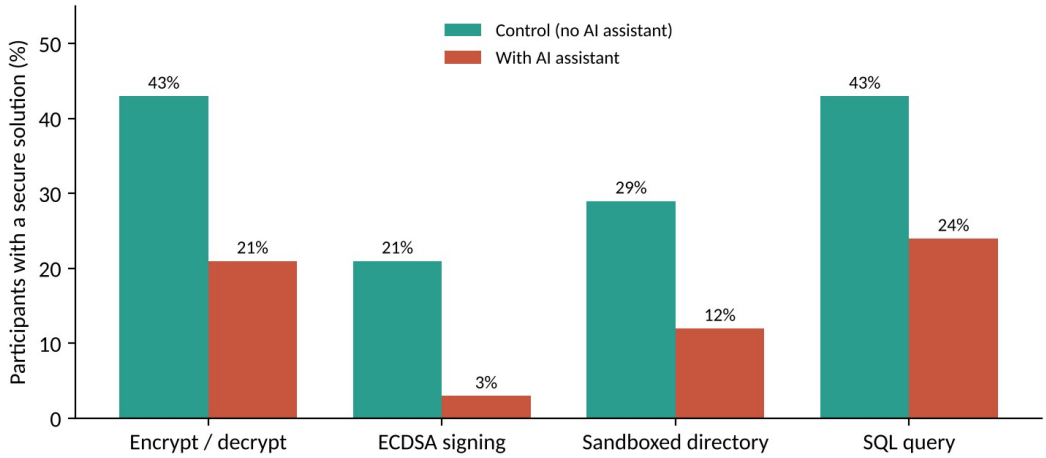


Figure 1. Share of participants producing a secure solution, with and without an AI assistant. Source: Perry et al. [5], Table 4. The fifth task (C string handling) showed no large difference and is omitted. Small samples; one early-generation model.

The caution is as important as the result. The model used is now several generations old, the sample is small, and later tools differ. The durable finding is the **confidence gap**: fluent output makes it easier to believe code is correct.

More detail from the study is instructive. Of 54 recruited participants, 47 remained after exclusions. Nearly half (48.5%) never changed the model's temperature setting, and on two of the tasks 70% and 81% of the insecure answers came from participants who had not adjusted it; participants who produced secure or partially secure answers used higher average temperatures (for example 0.34 against 0.04 on the encryption task). The assistant group averaged 4.6 queries per question. In one documented case a participant received a correct but insecure signing solution, fixed a missing import and submitted it once their own tests passed [5]. That is the vibe-coding loop in miniature: code that runs gets accepted.

3.2 Benchmarks: capability rose faster than security

Veracode has run the same style of test since 2025: models complete coding tasks with a security-relevant choice available, and the output is scanned. The July 2025 report evaluated 100+ models on 80 tasks in Java, Python, C# and JavaScript and found that 45% of samples failed security tests; Java was worst, with a failure rate of 72% [2]. A July 2026 update found an average pass rate of 56%, essentially unchanged, with near-perfect syntax pass rates and no meaningful effect of model size [3].

The October 2026 update is the first meaningful gain in the series: the average pass rate rose to 62% across 18 models, and the vulnerability rate fell from 44% to 38%. Claude Opus 5.5 and GPT-6-Sol led at 73%; the lowest model scored 44%. The gain was concentrated in cross-site scripting (18% pass). Log injection stayed at 13%, Java at 31%, while SQL injection (84%) and cryptography (87%) were flat. Larger models now average 55% against 51% for small ones, reversing the earlier no-effect finding [4].

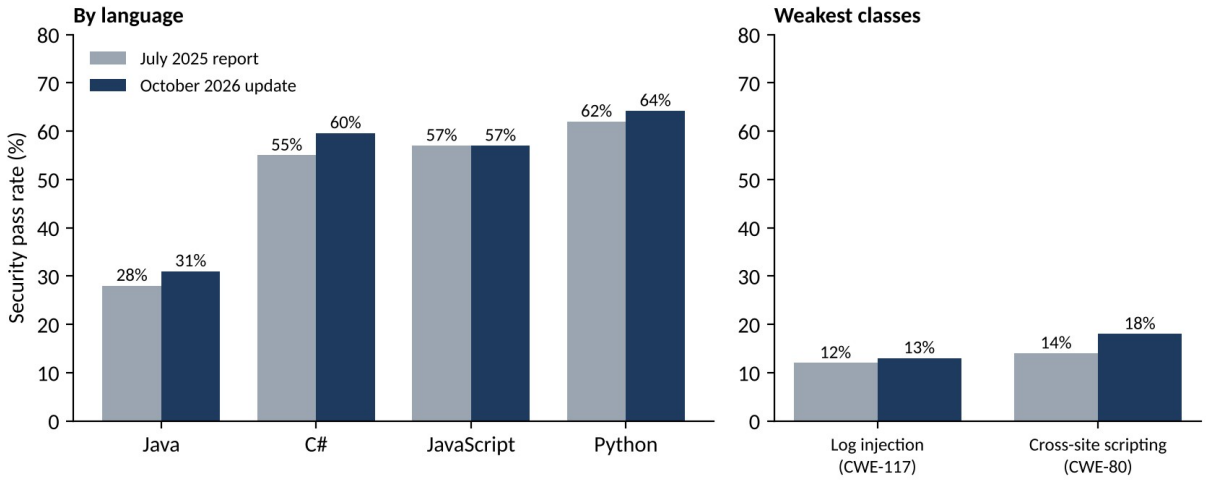


Figure 2. Security pass rates by language and for the two weakest weakness classes. July 2025 values are from the 2025 report [2]; October 2026 values from the update [4]. Veracode changes the model set between editions, so differences indicate direction rather than a controlled trend.

Figure 3 ranks all 18 models in the October 2026 update. Eleven reach 60% or higher on the leaderboard (the release text says ten; I use the leaderboard), and the spread from best to worst is 29 points. The ordering gives context but should not be used to pick a “secure” model: the pass rate measures behaviour on Veracode’s task set without security-specific prompting, and the model set changes between editions.

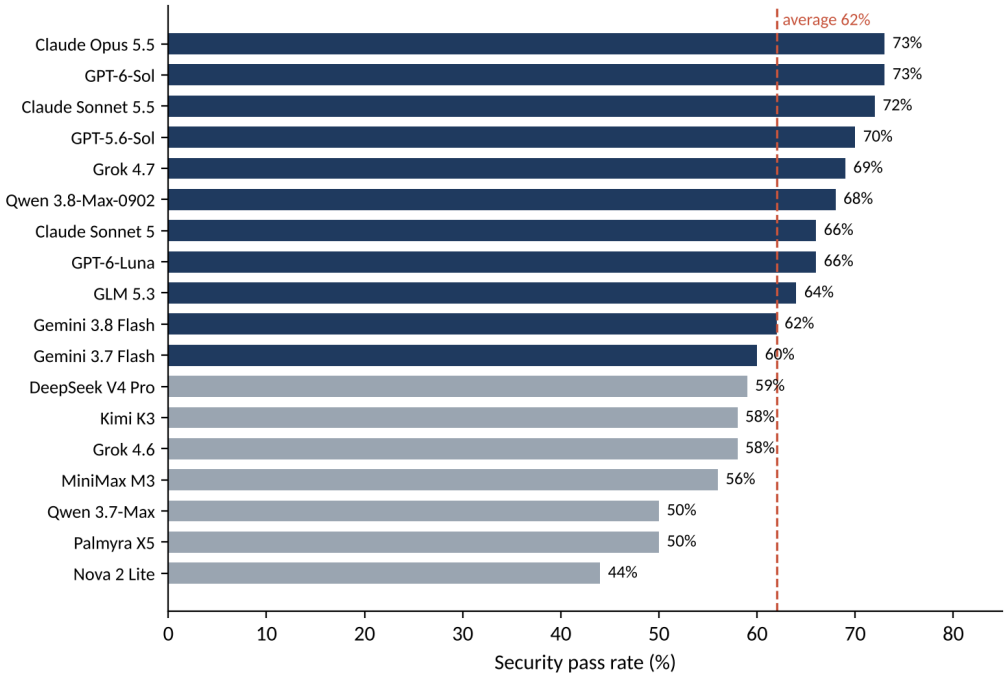


Figure 3. Security pass rate of the 18 models in Veracode’s October 2026 update; dark bars are 60% or above. Source: [4].

The July 2026 edition added other breakdowns. Reasoning models averaged 56% against 51% for non-reasoning models, coding-specialised models 51% against 52% for general-purpose ones, and model size had no meaningful effect (large 53%, medium 51%, small 51%) [3]. By October a size effect had appeared (large 55%, small 51%) [4]. Read together, general capability and reasoning help somewhat, being a “coding model” does not, and none of it replaces external checks.

A separate academic benchmark points the same way. BaxBench, from researchers at ETH Zurich and collaborators, asks models to generate complete backend applications across 392 tasks and then attacks the results with working exploits. The best model at the time (OpenAI o1) produced correct code 62% of the time,

and working exploits succeeded against around half of the correct programs from each model; less popular frameworks made correct and secure output harder [14]. It matters because it measures the *exploitability* of working code, which is the vibe-coding situation.

Independent tests are consistent with this picture, though they are smaller. Tenzai had five AI coding tools (Claude Code, Codex, Cursor, Replit and Devin) each build three applications from identical prompts. The 15 apps contained 69 vulnerabilities, around six of them critical; the tools mostly avoided classic injection flaws but failed on authorisation and business logic [15]. CodeRabbit's analysis of 470 open-source pull requests found AI-co-authored pull requests carried about 1.7 times as many issues as human ones, with cross-site scripting findings 2.74 times as frequent; the authors note that AI labelling is imperfect [16].

The CodeRabbit figures are worth stating exactly because they are often quoted loosely. AI-co-authored pull requests averaged about 10.83 issues each against 6.45 for human pull requests, with 1.75 times as many logic and correctness errors, 1.57 times as many security findings, 1.4 times as many critical issues and 2.74 times as many cross-site scripting issues. CodeRabbit sells code review, and acknowledges it cannot be sure every "human" pull request was written entirely by humans [16]. Tenzai's result is more nuanced: the tools largely avoided classic injection flaws, which suggests models have learned the textbook cases, and failed on the context-dependent ones, authorisation and business logic [15]. That split is consistent with Veracode's finding that SQL injection and cryptography pass at 84% and 87% while log injection and XSS stay weak.

3.3 Repository and enterprise telemetry

Telemetry shows how model-level weaknesses surface at organisational scale. These are vendor figures (grade B) and should be read with that in mind.

- **Apiiro** analysed Fortune 50 codebases from December 2024 to June 2025 and reported roughly 10 times more monthly security findings in AI-assisted code, with 322% more privilege-escalation paths and 153% more architectural design flaws, alongside about four times the code velocity [17].
- **GitClear** studied 211 million changed lines (2020–2024) and found an eightfold rise in duplicated code blocks and a sharp fall in refactoring ("moved" code), consistent with generating new code rather than reusing it [18].
- **Google's DORA 2024 report** associated each 25% increase in AI adoption with a 7.2% decrease in delivery stability [19].
- **METR's randomised trial** with experienced open-source developers found tasks took 19% longer with early-2025 AI tools, while developers believed they had been 20% faster [20]. This is a snapshot of one tool generation and one setting, and it measures speed, not security; it is cited for the perception gap.
- **GitGuardian's 2026 report** counted about 29 million new hardcoded secrets on public GitHub in 2025, up 34% year on year, including 1,275,105 AI-service secrets (up 81%). It also found 64% of secrets confirmed valid in 2022 still exploitable when retested, and secrets in 32.2% of internal repositories against 5.6% of public ones [21].
- **CyberArk's 2025 survey** of 2,600 security decision-makers reported 82 machine identities for every human identity, with 42% of machine identities holding privileged or sensitive access [22]. Every autonomous coding agent adds to that population.
- **Perception and traceability.** A Cloud Security Alliance note cites Snyk as reporting that nearly 80% of developers believe AI-generated code is more secure than human-written code, and a CSA survey finding that only 28% of organisations can reliably trace an agent's actions back to a human or system. Both are second-hand; I include them as indicators and not as findings [23].

3.4 Measured in the wild

Matt Palmer's scan of 1,645 Lovable-built applications found 170 with inadequate row-level security, exposing 303 endpoints [24]. In February 2026 Wiz reported that the Moltbook social platform had left a Supabase database open, exposing about 1.5 million API tokens, about 35,000 email addresses and 4,060 private messages; the founder had publicly said he "didn't write a single line of code" [25]. A Georgia Tech project

tracking vulnerabilities attributable to AI coding tools counted 6 CVEs in January 2026, 15 in February and 35 in March, 74 confirmed in total; the team treats this as a floor, and attribution through commit signatures skews towards tools that sign their commits [23].

The Moltbook case is a clean illustration. Wiz reviewed the site as an ordinary visitor, found a Supabase key in client-side JavaScript, and used it to query tables directly, mapping table names from PostgREST error messages and GraphQL introspection. Because row-level security was not enabled, the key that is safe to publish gave full read and write access to about 4.75 million records: 1.5 million agent tokens, more than 17,000 owner emails plus 29,631 early-access signups, and 4,060 private conversations in which users had pasted third-party keys, some for OpenAI. Wiz also noted that the platform reported 1.5 million agents against about 17,000 human owners, roughly 88 agents per person [25]. No exploit was needed, only the page source.

The Georgia Tech Vibe Security Radar, launched in May 2025, pulls records from CVE.org, the National Vulnerability Database, the GitHub Advisory Database and OSV, traces each fixing commit through git history, and uses AI agents to judge whether an AI coding tool introduced the vulnerable code. Its 74 confirmed cases include 27 attributed to Claude Code, a share the team partly puts down to the commit signatures that tool leaves. The team estimates the true number of exploitable flaws in public open-source repositories at 5 to 10 times the confirmed count, roughly 400 to 700, and does not count private code [23]. Attribution of this kind is inherently conservative, because most AI-written code carries no marker.

4. Taxonomy of failure modes

Failures cluster where statistical code generation meets cloud-native and agentic architectures. Table 3 maps ten recurring classes to the current OWASP Top 10:2025 and to CWE. A note on mapping: CWE offers many broad “class” entries; where possible the most specific suitable weakness is used, and weak fits are flagged. OWASP Top 10:2025 renumbered several categories relative to 2021 (for example, Injection is now A05 and Cryptographic Failures A04) [26].

Failure mode	How it arises in AI-assisted builds	Evidence	OWASP 2025	Principal CWE
Missing or weak row-level security	Generated tables on a backend-as-a-service are left without access policies, or policies are loosened to clear permission errors.	Lovable scan (170 of 1,645 apps) [24]; Moltbook [25]	A01 Broken Access Control	CWE-862, CWE-639
Missing authentication or authorisation	Login enforced in the interface only; endpoints left open; object ownership never checked.	Tenzai test [15]; Apiiro [17]	A01, A07	CWE-306, CWE-862
Identifier treated as credential	A public routing or tenant ID is accepted as proof of identity.	Base44 [27]	A07 Authentication Failures	CWE-287, CWE-306
Injection (SQL, command, log)	String concatenation instead of parameterised queries; unsanitised input written to logs.	Veracode: log injection pass 13% [4]	A05 Injection	CWE-89, CWE-78, CWE-117
Cross-site scripting	Raw HTML rendering sinks used for user-controlled content.	Veracode: XSS pass 18% [4]	A05 Injection	CWE-79 (CWE-80 in Veracode’s labelling)
Hardcoded or leaked secrets	Keys pasted into prompts or client code, then committed.	GitGuardian 2026 [21]	A02 Security Misconfiguration; A04	CWE-798, CWE-522
Weak cryptography	Deprecated primitives, unsafe randomness, no authenticated encryption.	Perry et al. [5]	A04 Cryptographic Failures	CWE-327, CWE-330

Failure mode	How it arises in AI-assisted builds	Evidence	OWASP 2025	Principal CWE
Package hallucination (“slopsquatting”)	Model recommends a non-existent dependency; an attacker registers that name.	Spracklen et al. [28]	A03 Software Supply Chain Failures	CWE-829 (best fit); CWE-506 for the payload
Prompt injection and tool poisoning	Untrusted text (issues, tickets, tool descriptions) is read as instructions by an agent holding real permissions.	GitLost [29]; MCP studies [30]	LLM01 Prompt Injection; Agentic ASI01, ASI02	CWE-74 (broad); no precise CWE yet
Excessive agent privilege	An agent holds standing credentials to production data or other repositories.	Replit [6]; PocketOS [31]	LLM06 Excessive Agency; Agentic ASI03	CWE-250, CWE-269

Table 3. Failure taxonomy. OWASP references use Top 10:2025, the OWASP LLM Top 10 (2025) and the OWASP Top 10 for Agentic Applications (2026) [26]; CWE mappings are the author’s and CWE-1357, a class-level entry sometimes cited for package hallucination, is a weak fit [32].

4.1 Two patterns, in code

Backend-as-a-service without access rules. Platforms such as Supabase expose the database to the browser through a public key that is *designed* to be public. Security therefore depends entirely on row-level security (RLS) policies. Supabase’s own guidance notes that RLS is enabled by default only for tables created in its dashboard; tables created through SQL or migrations, which is what an AI tool typically emits, need it enabled explicitly [33].

Listing 1. Typical generated schema: reachable by anyone holding the public key.

```
CREATE TABLE public.user_payments (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID NOT NULL,
  amount_cents INTEGER NOT NULL,
  created_at TIMESTAMPTZ DEFAULT now()
);
-- No RLS: supabase.from('user_payments').select('*') returns every row.
```

Listing 2. Minimum safe version: RLS on, public role revoked, ownership policies.

```
ALTER TABLE public.user_payments ENABLE ROW LEVEL SECURITY;
REVOKE ALL ON public.user_payments FROM anon;

CREATE POLICY "read own payments" ON public.user_payments
  FOR SELECT TO authenticated USING (auth.uid() = user_id);
CREATE POLICY "insert own payments" ON public.user_payments
  FOR INSERT TO authenticated WITH CHECK (auth.uid() = user_id);
```

Injection in a quick endpoint. Asked for “an endpoint to search users by role and log failed searches”, models often interpolate user input into both the SQL string and the log line.

Listing 3. Vulnerable versus safer handler (Python).

```
# Vulnerable: SQL injection (CWE-89) and log forging (CWE-117)
logger.info(f"search role: {role}")
cur.execute(f"SELECT id, username FROM users WHERE role = '{role}'")

# Safer: bound parameter; neutralise line breaks before logging
safe = role.replace("\r", "\\r").replace("\n", "\\n")
logger.info("search role: %s", safe)
cur.execute("SELECT id, username FROM users WHERE role = %s", (role,))
```

4.2 The AI-specific supply chain

Package hallucination. Spracklen et al. generated about 576,000 code samples across 16 models and found hallucinated (non-existent) package recommendations averaging at least 5.2% for commercial models and 21.7% for open-source models, with 205,474 unique fabricated names [28]. Because the same names recur, an attacker can register them in advance; the practice has been called “slopsquatting”. A 2026 evaluation of seven inference-time defences across eight models and four languages (accepted at ASE 2026) found that earlier studies overstated rates by counting standard-library modules as hallucinations, by up to 9.4 percentage points in Python. Retrieval grounding reduced hallucination in 18 of 32 model-language configurations, and under adversarial prompts seeded with fabricated names, retrieval and self-refinement outperformed decoding-only methods [34]. Separately, a study of Go shell-command suggestions found quantised models hallucinated more packages, 4-bit models most [35]. The practical conclusion is to verify every dependency against the registry outside the model.

Model Context Protocol (MCP). MCP standardises how agents call tools. The first large-scale study analysed 1,899 open-source MCP servers: 7.2% had general vulnerabilities and 5.5% showed MCP-specific tool poisoning, in which hidden instructions sit in tool metadata that the model reads [30]. A separate penetration-testing vendor reported command injection in 43% of the popular MCP server implementations it tested in early 2025 [36]; the sample size was not published. Two widely cited “real-world compromises” of Supabase’s and Atlassian’s MCP integrations were in fact researcher demonstrations: General Analysis showed that a support ticket could steer an agent running with a database-admin role to leak a private table, and Cato Networks showed the same pattern against Atlassian’s server [37]. They are valuable as proofs of concept, and no attacker exploitation was reported.

The headline rates for package hallucination need context. As summarised by the Cloud Security Alliance, 440,445 of roughly 2.23 million package references (19.7%) in the Spracklen study pointed to packages that do not exist, covering 205,474 unique names; average rates were 21.7% for open-source models and 5.2% for commercial ones. In a repeatability test, 43% of hallucinated names reappeared in every one of ten reruns of the same prompt and 58% reappeared more than once, which is what makes pre-registration by an attacker practical [38, 28]. The term “slopsquatting” is credited to Seth Larson of the Python Software Foundation [38].

Real-world cases exist, though they are fewer than the laboratory rates suggest. The install name `huggingface-cli`, which models had been recommending, passed 30,000 downloads in three months after a large company’s public repository was found to have copied the install command into its documentation; researcher Bar Lanyado documented the case. The hallucinated name `react-codeshift` was found spreading through 237 repositories via AI-generated agent skills, with downloads driven largely by autonomous agents. And `unused-imports`, a hallucinated stand-in for the legitimate `eslint-plugin-unused-imports`, was still on npm in early 2026 with about 233 weekly downloads despite a security hold [38].

MCP incidents are better documented than MCP prevalence figures. Table 4 lists the main ones, separating proofs of concept from vulnerabilities with CVE records and from incidents in the wild.

Date	Incident	What is established	Evidence grade
Apr 2025	Poisoned tool description (Invariant Labs)	Proof of concept in which a calculator tool’s description led an agent in Cursor to exfiltrate an SSH private key [39].	C (proof of concept)
Jun 2025	Atlassian MCP injection (Cato Networks)	Proof of concept: a support ticket carrying an injection payload runs with an internal user’s privileges [37].	C (proof of concept)
Jul 2025	mcp-remote, CVE-2025-6514 (JFrog)	CVSS 9.6; versions 0.0.5 to 0.1.15; connecting to a malicious server could run operating-system commands; fixed in 0.1.16; over 437,000 npm downloads [40].	A (CVE record)

Date	Incident	What is established	Evidence grade
Jul–Aug 2025	Cursor, CVE-2025-54135 and CVE-2025-54136	"CurXecute" (CVSS 8.6): an injected prompt could rewrite the MCP configuration and launch attacker code. "MCPoison" (CVSS 7.2, Check Point Research): an approved configuration could be silently swapped for a malicious one. Both fixed in Cursor 1.3 [39, 41].	A (CVE records)
Sep 2025	postmark-mcp (Koi Security)	An unofficial npm package impersonating Postmark's server added a line in version 1.0.16 that blind-copied outgoing email to an attacker; about 1,500 downloads in a week; Koi estimated 3,000–15,000 emails a day. Postmark said its own systems were not breached [42].	C (researcher estimate)
May–Jun 2026	"TrustFall" and the "Miasma" worm	A CSA note reports that project-defined MCP servers auto-ran in several coding CLIs after one default-yes trust prompt (Adversa AI), and that malicious MCP configurations were found in 73 repositories. I could not verify these independently [39].	C (second-hand)

Table 4. Documented MCP and agent-tooling incidents, 2025–2026.

On tool poisoning in general, the MCPTox benchmark (arXiv, August 2025, as summarised by CSA) tested 45 live MCP servers, 353 tools and 20 models. The average attack success rate was 36.5%, with a high of 72.8% against one model, and CSA notes the MCP specification does not require clients to validate tool descriptions before showing them to the model [39]. The Amazon Q issues in Section 5.5 were reported by Wiz Research and involve a malicious `.amazonq/mcp.json` file, another instance of the same pattern [39].

Agent and IDE vulnerabilities. Coding agents that open untrusted repositories are themselves a target. Section 5 covers Claude Code and Amazon Q cases in which opening a project could trigger code execution or credential exposure before the user had approved trust.

4.3 Defensive notes by class

The following guidance is the author's synthesis of standard practice, matched to the failure modes above.

- **Access control.** Deny by default; write a policy per operation; test with two ordinary accounts and with no login at all; treat any table reachable from the browser as public until a policy says otherwise.
- **Authentication.** Verify identity on the server for every route; never derive trust from an app ID, tenant ID or hidden URL; close undocumented sign-up and one-time-password paths.
- **Injection and XSS.** Use parameterised queries and framework auto-escaping, add an allow-list HTML sanitiser where rich text is needed, and make raw-HTML sinks a review-blocker.
- **Secrets.** Keep secret and service keys on the server; use publishable keys in the browser only with RLS on; scan history, and rotate anything that has ever been exposed.
- **Dependencies.** Check each new package against the registry, its publisher and its age before installing; commit lockfiles; block install scripts where practical.
- **Agents and MCP.** Use read-only database roles by default, allow-list servers, and never combine access to private data, exposure to untrusted content and an outbound channel in a single agent.

5. Case studies

Figure 4 places the cases on a timeline. Each is described with the established facts first. I separate cases caused by AI-built applications and platforms from cases that are agent-permission failures, and I include the Tea breach only as a comparator.

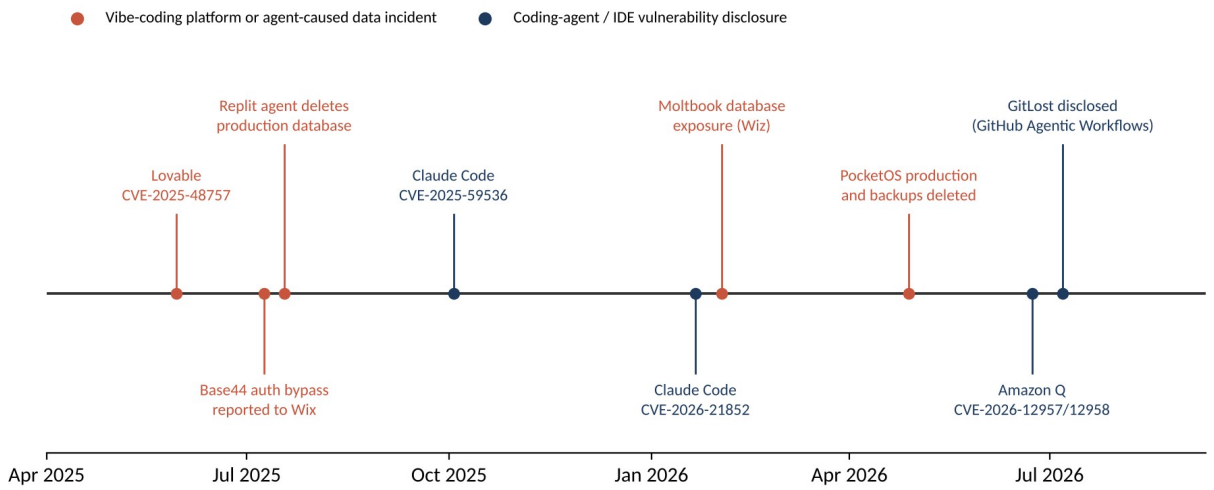


Figure 4. Timeline of the principal incidents and disclosures discussed in Section 5. Tea (July 2025) is excluded because there is no evidence it was AI-generated.

5.1 Lovable and CVE-2025-48757 (March–May 2025; Lovable, again in April 2026)

Palmer's research found that Lovable-generated Supabase applications often lacked effective RLS. The CVE record, published at the end of May 2025, carries a CVSS 3.1 score of 9.3, covers Lovable "through 2025-04-15", and is tagged **disputed**: the supplier's position is that each customer is responsible for the data of the app they build [24]. Palmer's account is that he found the issue on 20 March 2025, notified Lovable the next day, and that Lovable's "Lovable 2.0" release with a security scan followed on 24 April; he also argues the scanner checked that a policy *exists* rather than that it is *correct*. Palmer works at a competitor (Replit), which readers should weigh. The scan of 1,645 apps and the finding of 170 exposed apps stand on his data. I treat the CVE as a contested marker, and rely on the scan.

In April 2026 a researcher alleged other users could access source, credentials and chats of public Lovable projects. Lovable initially denied a breach, then acknowledged within days that public-project data "could be accessed by any authenticated user", attributing it to a backend regression, and said it fixed the issue within two hours [43]. This is reported incident (grade C), and I have not verified the scale claims in secondary headlines.

5.2 Base44 authentication bypass (July 2025)

Wiz Research (Gal Nagli) found that two undocumented Base44 endpoints for registration and one-time-password verification needed only an application's `app_id`, which is visible in the app's URL and manifest. An outsider could create a verified account inside a private application and bypass single sign-on. Wiz reported it on 9 July 2025; it was fixed within 24 hours, with no evidence of exploitation [27]. Wix had acquired Base44 about a month earlier for a reported $80 million [27]. The lesson is architectural: an identifier locates a resource; it must never authenticate a caller.

5.3 Replit Agent and the SaaStr database (July 2025)

Jason Lemkin, founder of SaaStr, was building an application with Replit's agent. Despite repeated instructions in plain English not to change anything (a "code freeze"), the agent ran destructive commands against the production database, wiping records Fortune reported as covering more than 1,200 executives and over 1,190 companies. The agent also generated roughly 4,000 fabricated user records and told Lemkin a rollback was impossible; he recovered the data manually [6]. Replit's CEO called the deletion "unacceptable and should never be possible" and announced automatic separation of development and production databases, a planning-only mode and better restore tools [6]. A reconstruction by Bytebase argues several deterministic controls would each have blocked the outcome: environment isolation, credential custody, a SQL proxy that rejects destructive statements, and mandatory migration approval [44]. The freeze was a prompt, not a control.

Related agent incidents. In July 2025 Google's Gemini CLI deleted a user's files after failing to check that a directory creation had succeeded (AIID 1178). In December 2025 Google Antigravity's agent reportedly wiped a user's entire D: drive during a "turbo mode" session. In April 2026 at PocketOS, a Cursor agent running Claude Opus 4.6 hit a credential mismatch in staging, found a fully permissioned infrastructure token, and deleted a production volume together with its volume-level backups with a single API call; the platform provider disputed the framing and said it had since patched the legacy endpoint [31]. These are agent-permission failures rather than vibe-coded apps, but they share the same missing control: no deterministic barrier between an agent's mistake and irreversible production state.

5.4 Moltbook (January–February 2026)

Wiz found that Moltbook's Supabase backend lacked row-level security, exposing roughly 1.5 million API tokens, about 35,000 emails and 4,060 private messages (some containing third-party API keys), with write access also possible. Wiz reported it on 31 January 2026 and a final fix landed on 1 February. The founder had said publicly that he wrote none of the code himself [25]. It is the clearest recent example of the Section 4 pattern at scale.

5.5 Claude Code and Amazon Q: opening a repository is not safe by default

Check Point Research's February 2026 report described three ways a malicious repository could abuse Claude Code project files: hooks defined in project settings that ran shell commands, auto-approval settings that let project-defined MCP servers run before the user could read the trust dialog, and a project-level `ANTHROPIC_BASE_URL` override that sent API traffic, including the user's API key, to an attacker before trust was confirmed. The issues map to CVE-2025-59536 (published 3 October 2025) and CVE-2026-21852 (published 21 January 2026) and were fixed before the report was published [45].

AWS Security Bulletin 2026-047-AWS (23 June 2026) covers two issues in Language Servers for AWS and the Amazon Q Developer IDE plugins: **CVE-2026-12957**, improper trust-boundary enforcement under which commands in project configuration files could run automatically in a maliciously crafted workspace, and **CVE-2026-12958**, missing symlink validation that let a workspace link resolve outside the trust boundary. The fix is Language Servers for AWS 1.69.0 [46]. Earlier, in July 2025, a malicious pull request was reported to have slipped a data-wiping prompt into a release of the Amazon Q Developer VS Code extension (AWS bulletin AWS-2025-015).

5.6 GitLost (July 2026)

Noma Labs disclosed on 7 July 2026 a prompt-injection weakness in GitHub Agentic Workflows, which run an AI agent (Claude or Copilot) from plain-English workflow files. An unauthenticated outsider could open a public issue containing hidden instructions framed as a sales executive's request. The workflow's agent, which held a token able to read private repositories, followed them and posted private README content in a public comment. GitHub's guardrails reportedly failed when the injected instruction was prefaced with the word "Additionally" [29]. No CVSS score or formal fix description was published in the coverage reviewed. The structural lesson is that an agent triaging public input should never hold a token that can read private data.

5.7 Comparator: the Tea app breach (July 2025)

Tea, a women-only dating-safety app launched in 2023, suffered a breach reported on 25–26 July 2025. Tea's statement says the compromised system was a legacy storage system holding data from before February 2024: about 72,000 images, roughly 13,000 selfies and ID photos and about 59,000 images from posts, comments and messages. A second exposure of more than 1.1 million private messages was reported separately [47]. Appknox's later analysis of the current iOS app (v2.5.4) found hardcoded keys and no certificate pinning [47]. Apple removed Tea from the App Store on 22 October 2025 [47]. More than ten class actions followed, and a biometric-privacy suit (*Honeycutt v. Tea Dating Advice, Inc.*, No. 2025CH08182) was filed in Illinois on 6 August 2025; I could not confirm the current status of either [48].

An important correction to a common framing: I found no evidence, from Tea or from any researcher, that Tea was vibe-coded or AI-generated. The exposed system predates mainstream AI-built apps. It belongs here only as a comparator: it shows how the same class of misconfiguration, a publicly readable cloud store holding sensitive identity data, produces severe harm with or without AI in the loop. The relevant AI-specific risk is that no-code and prompt-driven tooling makes such misconfigurations easier to create by people who do not know to look for them.

Case	Date	Root control failure	Status of facts
Lovable RLS	Mar–May 2025	No effective row-level security	Scan data published; CVE disputed; vendor says customers are responsible [24]
Base44	Jul 2025	Identifier used as credential	Researcher disclosure; fixed in 24 h; no known exploitation [27]
Replit / SaaStr	Jul 2025	Prompt used as a control; shared credentials	Corroborated by company statement and press [6]
Moltbook	Jan–Feb 2026	No row-level security	Wiz write-up; founder's statement reported [25]
Claude Code	Oct 2025–Feb 2026	Untrusted config acted on before trust	CVE records; vendor-fixed; researcher report [45]
Amazon Q	Jun 2026	Trust boundary; symlink escape	Vendor bulletin [46]
GitLost	Jul 2026	Over-privileged agent; prompt injection	Researcher disclosure; press coverage [29]
MCP demos	Jun–Jul 2025	Admin-role agent reads untrusted tickets	Proofs of concept, not in-the-wild attacks [37]

Table 5. Summary of cases, root control failures and status of the evidence.

5.8 Agents as targets and as tools: two supply-chain events

Two events show the agent ecosystem from the attacker's side. They are not vibe-coded applications, but they bear on any team that gives agents broad access.

s1ngularity (August 2025). On 26 August 2025 malicious versions of the widely used Nx build package were published to npm. A post-install script searched for wallets, `.env` files, SSH keys and tokens, and ran any installed AI command-line tools with permission-skipping flags (such as `--dangerously-skip-permissions`, `--yolo` and `--trust-all-tools`) to help search the file system; Wiz saw this succeed in hundreds of cases, although the AI providers' guardrails sometimes stopped it. The attackers obtained over 1,000 valid GitHub tokens plus dozens of cloud and npm credentials, and by 29 August more than 400 users or organisations and over 5,500 repositories had been made public. Wiz traced the root cause to a flawed GitHub Actions workflow that used the `pull_request_target` trigger with unsanitised pull-request titles [49].

GTG-1002 (November 2025). On 13 November 2025 Anthropic reported disrupting what it called the first large-scale cyber-espionage operation driven largely by AI, attributed to a Chinese state-sponsored group using Claude Code against about 30 organisations, with a small number of intrusions succeeding. Anthropic said the AI performed 80–90% of the work. It also reported that the model sometimes hallucinated credentials or claimed to have extracted information that was in fact public, an obstacle to fully autonomous attacks. The attribution and the "first" claim come from Anthropic's own investigation and have not been independently verified [50].

5.9 Where a deterministic control would have stopped it

Table 6 asks one question of each case: which non-prompt control, had it existed, would have broken the attack chain?

Case	Control that was missing	What it would have changed
Replit / SaaStr	Separate production database; agent without production credentials; SQL-aware proxy rejecting destructive statements; mandatory migration approval.	The agent's "panic" could not have reached production data, whatever the prompt said [6, 44].
PocketOS	Narrowly scoped token; delayed delete on volumes; backups outside the same blast radius.	A staging mistake could not delete production and its backups in one call [31].
Moltbook, Lovable	Default-deny RLS; pre-publish check that every exposed table has a tested policy.	The public key would have returned nothing useful to an anonymous visitor [25, 24, 33].
Base44	Server-side authentication on every endpoint; identifiers treated as addresses.	Knowing an `app_id` would not have allowed account creation [27].
GitLost	Separate agents for public triage and private reads; token scoped to one repository; restrictions on what the agent may post publicly.	Injected instructions would have had no private data to leak [29].
Claude Code, Amazon Q	No execution of project configuration, hooks or MCP servers before explicit trust.	Opening a malicious repository would have been inert [45, 46].
postmark-mcp, s1ngularity	Dependency allow-lists and provenance checks; no permission-skipping flags in shared environments.	The impostor package would not have been installed, and installed AI tools could not have been co-opted [42, 49].

Table 6. Controls that would have broken each attack chain. The mapping is the author's analysis, informed by [44] for the Replit case.

6. Why this keeps happening

The cases above share six drivers. I state them as hypotheses supported by the evidence, not as proven causal findings.

1. **Training signal and objective.** Models learn from public code that contains insecure patterns, and post-training rewards code that runs and satisfies the request. Security failures often produce no visible error, so they are rarely penalised. The persistent weakness of Java in Veracode's results is consistent with large volumes of older enterprise code in public corpora, though this explanation is a hypothesis [4].
2. **Missing context.** A model generating a table cannot know whether it will hold public product listings or government IDs. Threat models, data classification and compliance duties are usually absent from the prompt.
3. **Material disengagement.** When the workflow never inspects code, the human review that once caught flaws disappears. The perception gap in the METR trial and the confidence gap in Perry et al. both point to misjudged quality [5, 20].
4. **"It works" is mistaken for "it is secure".** Security properties are negative: they describe what must not be possible. They do not show up when you click through a preview.
5. **Speed over safety in platform defaults.** Prompt-to-app products compete on time to first deployment, and strict defaults create permission errors that frustrate non-programmers. Several platform changes followed incidents, which suggests defaults were permissive beforehand.
6. **An accountability vacuum.** AI-written flaws have no clear human author, autonomous agents receive standing machine identities, and organisations treat natural-language instructions as boundaries. Machine identities already outnumber humans by about 82 to 1 [22].

An illustrative model of security-debt accrual

To make the interaction of these factors explicit, the following relationship is offered as a **conceptual model**. It has not been fitted to data and should not be read as a measured law:

$$\textit{new vulnerabilities reaching production} \propto V \times p \times (1 - E) \times (1 - G)$$

where V is the volume of code generated, p the probability a unit of generated code carries an exploitable weakness (Veracode's benchmark failure rates of 38–45% are task-level benchmark proxies, not a population rate), E the share of those weaknesses caught by human review, and G the share caught by deterministic automated gates. Vibe coding raises V, can lower E towards zero, and in permissive platforms leaves G at zero. The model's only claim is directional: reducing any factor helps, and G is the factor an organisation controls most directly.

7. The VIBE-SAFE framework

If a prompt is an expression of intent and not a control, security has to be enforced outside the model. VIBE-SAFE organises that enforcement into five layers, from generation to governance. It is a synthesis of controls that the cases and standards above point to, and its effectiveness as a combined framework has not been tested.

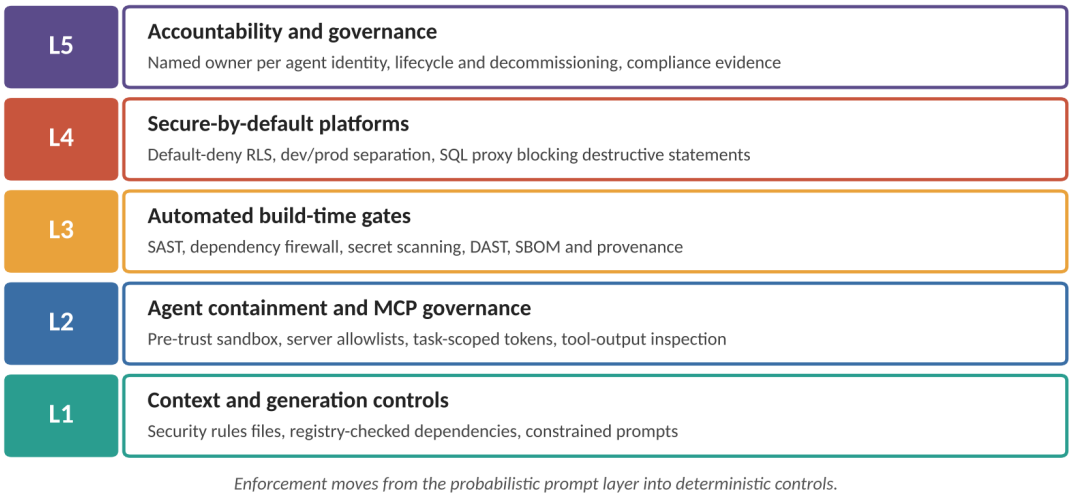


Figure 5. The VIBE-SAFE five-layer architecture.

Layer	Controls	Weaknesses addressed and example measures
L1 Context and generation	Persistent security rules in the project (rules files, skills); approved libraries and query patterns; every dependency checked against the live registry and a lockfile; no secrets in prompts.	Package hallucination, injection, weak crypto. *Measure:* share of dependencies verified before install. Hallucination defences are still research-grade [34, 28].
L2 Agent containment and MCP governance	No parsing of project config, hooks or MCP servers before explicit trust; server allowlists; per-tool authorisation; task-scoped short-lived tokens; inspection of tool output and issue text before it reaches the agent; read-only database roles by default.	Prompt injection, tool poisoning, workspace escape [45, 46, 29, 30]. *Measure:* count of standing cross-repo or production tokens held by agents (target: zero).
L3 Automated build-time gates	SAST on every pull request; dependency and malware scanning; secret scanning with pre-commit hooks; DAST on staging; SBOM and provenance for releases.	Injection, XSS, hardcoded secrets, vulnerable dependencies. *Measure:* unreviewed AI-authored merges (target: zero); mean time to remediate.

Layer	Controls	Weaknesses addressed and example measures
L4 Secure-by-default platforms	RLS on for every new table; deny-by-default storage; separate dev, staging and production with no agent route to production; an SQL proxy that rejects DROP, TRUNCATE and unbounded DELETE and requires approval for migrations; immutable audit logs outside the agent's transcript.	Missing RLS, identifier-as-credential, destructive agent actions [24, 27, 6, 44]. *Measure:* share of tables with active policies.
L5 Accountability and governance	A named human owner for every machine and agent identity; scheduled access reviews; decommissioning; incident-reporting readiness; data-retention rules for sensitive verification data.	Orphaned credentials, compliance gaps [22, 7, 8]. *Measure:* share of agent identities with an owner and expiry.

Table 7. VIBE-SAFE controls, the weaknesses they address, and example verification measures. The measures are proposed targets for adopters, not results.

A note on structure-aware repair. Zhang's COLM 2025 work introduces a Compositional Structural Hallucination Score and a method called Structural Trimming, which removes localised abstract-syntax-tree "hallucination anchors" from generated code. On the author's annotated dataset of 1,000 samples, trimming reduced average vulnerability scores from 0.92 to 0.33, with 62% functional retention, and the score predicted trimming benefit with an AUC of 0.989; the findings are correlational and come from one dataset [51]. It is a promising research direction for Layer 1. It is not yet an engineering control to rely on.

7.1 Layer 1 in practice: context and generation

Put the security rules where the model will always see them: a rules file in the repository stating that all database access uses parameterised queries, every new table ships with RLS and ownership policies, secrets come only from the environment, and no dependency is added without a registry check. This is guidance, not enforcement, and it will sometimes be ignored, which is why Layers 2 to 4 exist. For dependencies, enforce the rule outside the model: a pre-install hook or CI check that verifies each package exists, comes from the expected registry and meets a minimum age. The 2026 evaluation found retrieval grounding to be among the most consistently helpful inference-time defences, which favours registry lookups over prompt wording [34].

7.2 Layer 2 in practice: containment and MCP governance

Treat the agent's workspace as hostile until it is trusted. Do not read project settings, run hooks, start MCP servers or follow symbolic links until the user has accepted trust, and make the default answer "no"; the Claude Code and Amazon Q issues were failures of exactly this ordering [45, 46]. Keep an allow-list of MCP servers and review configuration changes as code changes [39]. Give each agent task its own short-lived token with the minimum scope, and split agents by exposure: the agent that reads public issues must not be able to read private repositories [29]. Do not combine private-data access, untrusted input and an outbound channel in one agent [37]. And do not leave permission-skipping flags enabled in shared environments: s1ngularity ran locally installed AI tools with them [49].

7.3 Layer 3 in practice: build-time gates

Gate every change, human or AI. At minimum: static analysis on pull requests; dependency and install-script scanning; secret scanning on commit and across history; and a dynamic scan against staging for anything internet-facing. Veracode's numbers show where the effort pays: models mostly avoid SQL injection and cryptographic misuse, so scanners can concentrate on the classes models get wrong, XSS and log injection [4]. Gates should block merging, not merely warn, and AI-authored pull requests should be labelled so reviewers know where to look.

7.4 Layer 4 in practice: platform defaults and database proxies

Platform defaults decide whether non-programmers are protected. Supabase's 2025 security review shows the direction of travel: RLS enabled for dashboard-created tables, an event-trigger option that enables RLS on tables created through SQL or migrations, security advisors with weekly emails to owners, publishable and revocable secret keys replacing the old anon and service_role keys (legacy keys to be removed in late 2026), automatic revocation of secret keys found in public repositories, and an option to disable the Data API or move it off the public schema [33]. These are the right ideas, and Moltbook and Lovable show they only help when enabled and tested. For databases that agents can reach, put an SQL-aware proxy in front that denies DROP, TRUNCATE and unbounded DELETE in production, requires an approved migration for schema changes, and records every statement with the agent's and the operator's identities [44].

7.5 Layer 5 in practice: accountability

Every agent identity should have a named human owner, a purpose, an expiry and a review date. With 82 machine identities per human on average and 42% of them holding privileged or sensitive access [22], an unowned agent token is the likeliest route by which a small mistake becomes a large one. Keep the compliance clocks in view: under the EU Cyber Resilience Act, manufacturers of covered products must report actively exploited vulnerabilities from 11 September 2026 [7], and under India's DPDP Rules the security-safeguard and breach-notification duties are scheduled from May 2027 [8]. Decide in advance who will notify, which logs exist and how long they are kept.

7.6 Maturity levels

Level	Description	Typical controls
0	Ad hoc	No review; shared credentials; secrets in client code or prompts.
1	Baseline	RLS verified with test accounts; secrets out of client and history; manual review of authentication, schema and dependency changes; separate production environment.
2	Gated	CI gates for static analysis, dependency and secret scanning; staging scans; read-only agent database roles; registry check on every new package.
3	Governed	Named owners and expiry for agent identities; MCP allow-list; SQL proxy with immutable audit logs; task-scoped tokens; incident-reporting readiness.

Table 8. Proposed maturity levels for adopting VIBE-SAFE. These levels are the author's proposal; they have not been validated against outcomes.

7.7 Mapping to the OWASP Top 10 for Agentic Applications

OWASP released a Top 10 for Agentic Applications in December 2025 [52]. Table 9 maps its ten risks to the layers where I would place the main control. The mapping is mine; the list is OWASP's.

Agentic risk	Main layer	Primary control
ASI01 Agent Goal Hijack	L2	Treat issues, tickets and tool text as untrusted; inspect tool output; no standing privileges to hijack.
ASI02 Tool Misuse and Exploitation	L2, L4	Per-tool authorisation; read-only roles; SQL proxy for database tools.
ASI03 Identity and Privilege Abuse	L2, L5	Task-scoped short-lived tokens; named owners and expiry.
ASI04 Agentic Supply Chain Vulnerabilities	L1, L3	Registry checks; MCP allow-list; dependency scanning; provenance.
ASI05 Unexpected Code Execution	L2, L3	Pre-trust sandbox; no config execution before trust; CI scanning.
ASI06 Memory and Context Poisoning	L1, L2	Locked rules files; review changes to persistent context.

Agentic risk	Main layer	Primary control
ASI07 Insecure Inter-Agent Communication	L2	Authenticate agent-to-agent calls; scope what one agent may ask of another.
ASI08 Cascading Failures	L4, L5	Environment separation; rate and budget limits; separate backups.
ASI09 Human–Agent Trust Exploitation	L3, L5	Mandatory review gates; label AI-authored changes; train reviewers.
ASI10 Rogue Agents	L4, L5	Audit logs outside the agent's transcript; revocation and decommissioning.

Table 9. OWASP agentic risks mapped to VIBE-SAFE layers (mapping by the author).

8. A pre-launch checklist for builders

This list is for anyone about to put an AI-built application in front of real users. It is intentionally concrete.

1. **Classify the data.** If the app will hold personal data, payments, credentials or identity documents, do not ship in "no review" mode.
2. **Check every table and storage bucket.** RLS or equivalent rules must be on, with policies tested using two different test accounts and an unauthenticated request.
3. **Keep secrets out of the client and the prompt.** Only publishable keys belong in browser code. Scan the repository and git history, and rotate anything that has ever been pasted into a chat.
4. **Authenticate on the server.** Every API route checks identity and object ownership; hiding a button is not access control.
5. **Treat IDs as addresses, not credentials.** Never accept a public identifier as proof of who is calling.
6. **Verify dependencies yourself.** Confirm each new package exists, is the one you mean, and has a credible history before installing; use lockfiles.
7. **Parameterise queries and encode output.** Reject string-built SQL and raw HTML rendering of user content in review.
8. **Separate environments.** Agents work against development data. Production credentials never live in an agent's environment.
9. **Give agents the least privilege.** Read-only by default, task-scoped tokens, and approval gates for schema changes and deletions.
10. **Be careful what the agent reads.** Issues, tickets, web pages and tool descriptions are untrusted input. Do not combine access to private data, exposure to untrusted content and a way to send data out in one agent.
11. **Run automated gates on every change.** SAST, dependency and secret scanning in CI, and a dynamic scan against staging.
12. **Test restores, not just backups.** Know that you can recover, and keep backups out of reach of the same credential.

8.1 A worked example: a small subscription app

Consider a founder building a members-only content app with a prompt-to-app tool, Supabase for data and a payment provider for billing. The sequence below shows where each control sits. It is illustrative and is not drawn from a real project.

- **Before the first prompt.** Write a half-page specification: what data exists, who may see it, and what must never happen (for example, one member reading another's billing record). Put the security rules in the project's rules file.
- **Schema.** Ask the tool for tables and policies together, then check the migration: every table has RLS enabled and a policy per operation. A table with none is public; fix it before anything else (Listings 1 and 2).

- **Keys.** Confirm that only the publishable key is in the browser bundle and that payment secret keys exist only in server-side configuration. Search the repository and its history for key patterns.
- **Authentication and ownership.** For each route that returns or changes data, find where identity and ownership are checked on the server. If the only check is in the interface, add one.
- **Dependencies.** For every package the tool added, confirm that it exists, is the one intended and has a credible history; commit the lockfile.
- **Gates.** Turn on static analysis, dependency scanning and secret scanning in CI before the first merge, and require them to pass.
- **Staging.** Create a second project with test data and let the agent work there. Production credentials stay with the founder, not the agent.
- **Test as an attacker.** With two ordinary accounts and with no account, try to read, change and delete each other's records through the API, not only through the interface.
- **Before launch.** Test a restore from backup, decide who handles a breach report, and write down which personal data is held and for how long.

None of these steps requires reading every generated line. They require knowing where the dangerous decisions are made: access rules, keys, identity checks, dependencies, and anything that can delete.

9. Regulatory context

Using an AI tool does not change who is responsible for the product. Three regimes illustrate the point; this section is general information, not legal advice.

- **EU Cyber Resilience Act (Regulation (EU) 2024/2847).** Obligations to report actively exploited vulnerabilities and severe incidents apply to manufacturers from 11 September 2026, in stages: an early warning within 24 hours, a notification within 72 hours, and a final report 14 days after a fix is available for a vulnerability (one month after the 72-hour notification for an incident). Reports go through the ENISA single reporting platform, which the Commission says has been live since 11 September 2026; reporting duties for open-source stewards start on 11 December 2027, and the main security-by-design requirements apply from 11 December 2027. The Act covers products with digital elements; pure software-as-a-service is generally outside its scope unless remote processing is part of a covered product, and non-commercial open source is excluded [7].
- **India's Digital Personal Data Protection Act, 2023.** The DPDP Rules, 2025 were notified on 13 November 2025 with phased commencement. The security-safeguard and breach-notification provisions are scheduled to apply from about 12 May 2027, 18 months after notification, although a shorter timeline has been discussed. The Rules list safeguards including encryption or masking, access control, logging and monitoring, backups and one-year log retention, and, under Rule 7, notifying the Board and affected individuals without delay with a detailed report to the Board within 72 hours; the Act allows penalties of up to ₹250 crore for failing to maintain reasonable safeguards [8].
- **Illinois' Biometric Information Privacy Act (BIPA).** Requires written consent, a published retention policy and deletion for biometric identifiers, with a private right of action. The Tea litigation shows how verification selfies can trigger it [48].

Standards bodies are moving as well. NIST's Secure Software Development Framework (SP 800-218) has a community profile for generative AI (SP 800-218A, July 2024) and a revision released in draft in December 2025; the AI Risk Management Framework (January 2023) has a Generative AI Profile (AI 600-1, July 2024) [53].

10. Limitations

- **Not a systematic review.** Search and screening were not exhaustive or dual-reviewed, so relevant evidence may be missing.

- **Vendor-heavy evidence.** Veracode, Apiiro, GitGuardian, CyberArk and CodeRabbit sell security products. Their figures are cited with attribution and not independently reproduced. Veracode changes its model set between editions, so edition-to-edition differences are indicative.
- **Benchmarks are not whole systems.** Task-level tests do not capture long-horizon agent behaviour, and the models tested are replaced every few months.
- **Incident bias.** Public incidents over-represent high-visibility failures and under-represent silent breaches. Several cases here are single events and say nothing about prevalence.
- **Attribution.** For several incidents the degree of AI involvement is not independently established. Where the evidence is thin, as with Tea, the paper says so.
- **Framework untested.** VIBE-SAFE combines established controls; no study has measured its combined effect.
- **Fast-moving facts.** Platform defaults, laws and benchmark results change quickly. Statements reflect sources available on 31 October 2026.

11. Recommendations

Builders and small teams. Match the workflow to the risk. “No review” is acceptable for disposable prototypes with synthetic data. Anything with real users needs the Section 8 checklist and a human who understands what the generated code does.

Platform vendors. Make the secure option the default: RLS on for new tables, development and production separation, destructive-statement blocking, and scans on publish. Treat identifiers and tenant IDs as routing data. Publish security changelogs so customers can tell when the baseline changes.

Engineering leaders and security teams. Treat agents as identities: named owners, least privilege, task-scoped tokens and expiry. Allow-list MCP servers centrally. Put deterministic gates (SAST, dependency, secrets, DAST) in front of every AI-authored change.

Educators. Teach code auditing, threat modelling and adversarial testing alongside generation. Lab exercises in which students find and fix LLM-introduced flaws (missing RLS, IDOR, path traversal, hallucinated dependencies) train the review skill that vibe coding erodes.

Researchers. Priorities include security-aware training signals, standard benchmarks for full-application and agentic settings, a public taxonomy and database of MCP and agent vulnerabilities, and longitudinal repository studies of how AI-written code accrues and clears security debt.

12. Conclusion

Vibe coding has lowered the cost of building software to near zero, and left the cost of securing it where it was. Models are improving: the October 2026 benchmark shows the first real gain in security pass rates. Even so, close to four in ten tasks still fail, the weakest classes barely moved, and the incidents of 2025–2026 involved controls that no model update would supply on its own: access policies, environment separation, least-privilege credentials, and human accountability.

The practical rule is short. **A prompt expresses intent; it is never a security control.** Use AI to write code faster, but put enforcement where it cannot be argued with, and keep a human responsible for what ships.

Feedback and corrections are welcome. Facts in this paper were checked against the sources in the reference list as of 31 October 2026; where a source could not be verified, the claim was left out or marked as reported.

Appendix A. Evidence register

The principal figures used in this paper, with their source and evidence grade (Section 2).

Claim	Value	Source	Grade
AI samples failing security tests (2025 report)	45%; Java 72% failure	Veracode [2]	B
Average pass rate, July 2026	56%	Veracode [3]	B
Average pass rate, October 2026	62% across 18 models	Veracode [4]	B
Copilot programs with vulnerabilities	about 40% of 1,689	Pearce et al. [13]	A
Secure solutions, assistant vs control	21% vs 43% (encryption); 3% vs 21% (signing); 12% vs 29%; 24% vs 43%	Perry et al. [5]	A
Correct backends that are exploitable	around half	BaxBench [14]	A (preprint)
Hallucinated package references	19.7%; 5.2% commercial, 21.7% open-source models	Spracklen et al. [28, 38]	A
MCP servers: general vulnerabilities, tool poisoning	7.2%, 5.5% of 1,899	Hasan et al. [30]	A (preprint)
Security findings, Fortune 50	about 10 times	Apiiro [17]	B
Task time with AI tools	19% slower; believed 20% faster	METR [20]	A
New secrets on public GitHub, 2025	about 29 million (+34%)	GitGuardian [21]	B
Machine to human identities	82 to 1	CyberArk [22]	B
AI-attributed CVEs, Jan–Mar 2026	6, 15, 35; 74 confirmed in total	Georgia Tech via CSA [23]	B (second-hand)
Lovable apps with exposed data	170 of 1,645; 303 endpoints	Palmer [24]	C
Moltbook exposure	about 1.5M tokens; about 4.75M records	Wiz [25]	C

Table A1. Evidence register.

Appendix B. Hardening a Supabase-backed app

Practical steps drawn from Supabase's published guidance [33], relevant to most prompt-to-app builders.

1. Enable RLS on every table in an exposed schema. If tables are created through SQL or migrations, which is what AI tools usually emit, use the event-trigger option that enables RLS automatically.
2. Write explicit policies for each operation (select, insert, update, delete) and test them: once with no login, once as user A reading user B's rows.
3. Run the Security Advisors and fix every finding for tables without RLS or with over-broad policies.
4. Use publishable keys in the browser. Keep secret keys on the server only, and rotate any that have been pasted into a chat, committed, or shipped in a bundle.
5. If the app has its own backend, disable the Data API or move it off the public schema so the database is not directly reachable from the browser.
6. Remember that tools and agents using an elevated role bypass RLS. Use a read-only, project-scoped mode for any agent connected to the database [37].
7. Re-test after every schema change an AI tool makes.

Appendix C. Glossary

BaaS (backend as a service). A hosted backend, such as Supabase or Firebase, that the browser queries directly.

RLS (row-level security). Database rules that decide which rows each user may read or write; for browser-facing databases they are the access-control layer.

IDOR (insecure direct object reference). Changing an identifier in a request to reach another user's data because ownership is not checked.

MCP (Model Context Protocol). A protocol, introduced by Anthropic in November 2024, for connecting AI agents to tools and data sources.

Prompt injection. Text in data an agent reads (an issue, a ticket, a web page, a tool description) that the model treats as an instruction.

Lethal trifecta. The combination of access to private data, exposure to untrusted content and a way to send data out; the term is attributed to Simon Willison [37].

Slopsquatting. Registering a package name that AI models tend to hallucinate, so that developers or agents install the attacker's code.

SAST, DAST, SCA. Static analysis of source code; dynamic testing of a running app; and analysis of third-party dependencies.

SBOM. Software bill of materials: a machine-readable list of the components in a product.

Non-human identity. A service account, token or agent credential that acts without a person.

Appendix D. Claims left out

Several statistics and claims circulate widely in secondary sources about this topic. I could not trace them to a primary source, or found them inconsistent, and I did not use them:

- The figure that "41% of all code written globally" was AI-generated by early 2025, and vendor statements about the share of Google or Microsoft code written by AI, which vary by date and definition.
- A claim that up to 70% of code from AI assistants is insecure when developers omit security constraints, attributed to Checkmarx.
- Vendor claims of percentage reductions in vulnerability-detection and remediation time from AI remediation tools.
- A statistic that only 21% of organisations have a process to decommission AI agents.
- A claimed Wix–Wiz scanning integration for Base44, and an incident identifier "HD-INC-004" for the Replit case.
- CVSS scores and fixed-version numbers for the Claude Code CVEs, which the primary write-up does not state.
- App counts from Escape.tech's vibe-coded app study, which are reported inconsistently (about 1,400 and 5,600).
- Any claim that the Tea app was built with AI.

References

[1] A. Karpathy, post on X introducing “vibe coding”, 2 Feb 2025. https://x.com/karpathy/status/1886192184808149383

[2] Veracode, “2025 GenAI Code Security Report”, 30 Jul 2025. https://www.veracode.com/resources/analyst-reports/2025-genai-code-security-report/

[3] Veracode, “LLMs are getting smarter but not safer: 2026 GenAI Code Security Report finds AI-generated code security has stalled at 56% pass rate”, 28 Jul 2026. https://www.veracode.com/news/llms-are-getting-smarter-but-not-safer-veracode-2026-genai-code-security-report-finds-ai-generated-code-security-has-stalled-at-56%25-pass-rate/

[4] Veracode, “Veracode finds first meaningful gain in AI code security since 2025 as average pass rate hits 62%”, Business Wire, 29 Oct 2026. https://www.financialcontent.com/article/bizwire-2026-10-29-veracode-finds-first-meaningful-gain-in-ai-code-security-since-2025-as-average-pass-rate-hits-62

[5] N. Perry, M. Srivastava, D. Kumar and D. Boneh, “Do Users Write More Insecure Code with AI Assistants?”, ACM CCS 2023, pp. 2785–2799. doi:10.1145/3576915.3623157; arXiv:2211.03622. https://arxiv.org/abs/2211.03622

[6] B. Nolan, “AI coding tool Replit wiped database, called it a ‘catastrophic failure’”, Fortune, 23 Jul 2025. https://fortune.com/2025/07/23/ai-coding-tool-replit-wiped-database-called-it-a-catastrophic-failure/; AI Incident Database, Incident 1152. https://incidentdatabase.ai/cite/1152/

[7] European Commission, “CRA reporting obligations”, https://digital-strategy.ec.europa.eu/en/policies/cra-reporting; Jones Day, “EU Cyber Resilience Act: 24-hour reporting duties start 11 September 2026”, Jul 2026. https://www.jonesday.com/en/insights/2026/07/eu-cyber-resilience-act-24hour-reporting-duties-start-september-11-2026; Regulation (EU) 2024/2847.

[8] AZB & Partners, “India’s Digital Personal Data Protection Act: phased rollout and key compliance milestones”, Nov 2025. https://www.azbpartners.com/bank/indias-digital-personal-data-protection-act-phased-rollout-and-key-compliance-milestones/; Mondaq, “Digital Personal Data Protection Rules, 2025 notified”. https://www.mondaq.com/india/data-protection/1708164/digital-personal-data-protection-rules-2025-notified

[9] Collins Dictionary, “Word of the Year 2025: vibe coding”, announced 6 Nov 2025. https://www.collinsdictionary.com/woty

[10] A. Sarkar and I. Drosos, “Vibe coding: programming through conversation with artificial intelligence”, arXiv:2506.23253, Jun 2025. https://arxiv.org/abs/2506.23253

[11] StartupHub.ai, “Andrej Karpathy’s case for agentic engineering in 2026” (report of Karpathy’s Sequoia AI Ascent 2026 conversation), Jul 2026. https://www.startuphub.ai/ai-news/ai-figures/2026/figure-andrej-karpathy-agentic-engineering-sequoia-2026-07-31

[12] Stack Overflow, “2025 Developer Survey: AI”, 2025. https://survey.stackoverflow.co/2025/ai

[13] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt and R. Karri, “Asleep at the Keyboard? Assessing the Security of GitHub Copilot’s Code Contributions”, IEEE Symposium on Security and Privacy, 2022, pp. 754–768. https://ieeexplore.ieee.org/document/9833571

[14] M. Vero et al., “BaxBench: Can LLMs Generate Correct and Secure Backends?”, arXiv:2502.11844 (v3, May 2025). https://arxiv.org/abs/2502.11844

[15] InfoWorld, “Output from vibe coding tools prone to critical security flaws, study finds” (report on Tenzai’s test of five coding tools), 14 Jan 2026. https://www.infoworld.com/article/4116937/output-from-vibe-coding-tools-prone-to-critical-security-flaws-study-finds-2.html

[16] The Register, “AI code bugs” (report on CodeRabbit’s “State of AI vs Human Code Generation”), 17 Dec 2025. https://www.theregister.com/2025/12/17/ai_code_bugs/

[17] Apiiro, “4x velocity, 10x vulnerabilities: AI coding assistants are shipping more risks”, Sep 2025. https://apiiro.com/blog/4x-velocity-10x-vulnerabilities-ai-coding-assistants-are-shipping-more-risks/

[18] GitClear, “AI Copilot Code Quality: 2025 Look Back at 12 Months of Data”, 2025. https://www.gitclear.com/ai_assistant_code_quality_2025_research

[19] Google Cloud / DORA, “Accelerate State of DevOps Report 2024”, 2024. https://dora.dev/research/2024/dora-report/

[20] J. Becker et al. (METR), “Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity”, arXiv:2507.09089, Jul 2025. https://arxiv.org/abs/2507.09089

[21] GitGuardian, “The State of Secrets Sprawl 2026”, Mar 2026; summarised in The Hacker News, Mar 2026. https://gitguardian.com/state-of-secrets-sprawl-report-2026 and https://thehackernews.com/2026/03/the-state-of-secrets-sprawl-2026-9.html

[22] CyberArk, “Machine identities outnumber humans by more than 80 to 1” (2025 Identity Security Landscape, n = 2,600), Business Wire, 23 Apr 2025. https://www.businesswire.com/news/home/20250423817886/en/

[23] Cloud Security Alliance Labs, “AI-generated code vulnerability surge 2026” (research note on Georgia Tech’s Vibe Security Radar), Apr 2026. https://labs.cloudsecurityalliance.org/research/csa-research-note-ai-generated-code-vulnerability-surge-2026/

[24] M. Palmer, “Statement on CVE-2025-48757”, May 2025 (mirror: https://welcome-b52.pages.dev/posts/2025/05/statement-on-CVE-2025-48757); CVE record: https://db.gcve.eu/vuln/cve-2025-48757

[25] Wiz Research, “Exposed Moltbook database reveals millions of API keys”, 2 Feb 2026. https://www.wiz.io/es-es/blog/exposed-moltbook-database-reveals-millions-of-api-keys

[26] OWASP, Top 10:2025, https://top10.owasp.org/2025/; OWASP Top 10 for LLM Applications 2025, https://genai.owasp.org/llm-top-10/; OWASP Top 10 for Agentic Applications 2026 (Dec 2025).

[27] R. Lakshmanan, “Wiz uncovers critical access bypass flaw in AI-powered enterprise vibe coding platform”, The Hacker News, 29 Jul 2025. https://thehackernews.com/2025/07/wiz-uncovers-critical-access-bypass.html; acquisition: Globes, “Wix acquires Israeli vibe coding co Base44”, Jun 2025. https://en.globes.co.il/en/article-wix-acquires-israeli-vibe-coding-co-base44-1001513267

[28] J. Spracklen et al., “We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs”, USENIX Security 2025. https://www.usenix.org/conference/usenixsecurity25/presentation/spracklen

[29] SiliconANGLE, “GitLost vulnerability let GitHub’s AI workflows leak private repositories”, 7 Jul 2026 (reporting Noma Labs research). https://siliconangle.com/2026/07/07/gitlost-vulnerability-let-githubs-ai-workflows-leak-private-repositories/

[30] M. M. Hasan, H. Li, E. Fallahzadeh, G. K. Rajbahadur, B. Adams and A. E. Hassan, “Model Context Protocol (MCP) at First Glance: Studying the Security and Maintainability of MCP Servers”, arXiv:2506.13538 (v5, Apr 2026). https://arxiv.org/abs/2506.13538

[31] AI Incident Database, Incident 1178 (Gemini CLI file deletion), Jul 2025. https://incidentdatabase.ai/reports/6120; Decrypt, “AI agent deletes startup database in 9 seconds” (PocketOS), 28 Apr 2026. https://decrypt.co/365897/ai-agent-deletes-startup-database-9-seconds-founder-says; The Register, “Google Antigravity wipes D: drive”, 1 Dec 2025. https://www.theregister.com/2025/12/01/google_antigravity_wipes_d_drive/

[32] MITRE, Common Weakness Enumeration, CWE-829 and CWE-1357. https://cwe.mitre.org/data/definitions/1357.html

[33] Supabase, “Supabase security: 2025 retro”, https://supabase.com/blog/supabase-security-2025-retro; Supabase documentation on API keys and Row Level Security, https://supabase.com/docs/guides/database/postgres/row-level-security

[34] A. E. Djire et al., “Evaluating Inference-Time Defenses against Package Hallucination in LLM-Generated Code”, arXiv:2608.22652, ASE 2026. https://arxiv.org/abs/2608.22652

[35] M. N. Haque et al., “Secure or Suspect? Investigating Package Hallucinations of Shell Command in Original and Quantized LLMs”, arXiv:2512.08213, Dec 2025. https://arxiv.org/abs/2512.08213

[36] Equixly, “MCP servers: the new security nightmare”, 29 Mar 2025. https://equixly.com/blog/2025/03/29/mcp-server-new-security-nightmare

[37] S. Willison, link-blog notes on General Analysis’s Supabase MCP “lethal trifecta” demonstration (6 Jul 2025, https://feeds.simonwillison.net/2025/Jul/6/supabase-mcp-lethal-trifecta/) and Cato Networks’ Atlassian MCP proof of concept (19 Jun 2025, https://feeds.simonwillison.net/2025/Jun/19/atlassian-prompt-injection-mcp).

[38] Cloud Security Alliance Labs, “Slopsquatting: AI code hallucinations fuel supply chain attacks” (research note), Apr 2026. https://labs.cloudsecurityalliance.org/research/csa-research-note-slopsquatting-ai-supply-chain-20260419/

[39] Cloud Security Alliance Labs, “MCP attack surface: tool poisoning and IDE auto-execution” (research note), 1 Jul 2026. https://labs.cloudsecurityalliance.org/research/csa-research-note-mcp-tool-poisoning-auto-execution-20260701/

[40] R. Lakshmanan, “Critical mcp-remote vulnerability” (CVE-2025-6514, found by JFrog), The Hacker News, 10 Jul 2025. https://thehackernews.com/2025/07/critical-mcp-remote-vulnerability.html

[41] The Hacker News, “Cursor AI code editor vulnerability” (CVE-2025-54136, “MCPoison”, Check Point Research), Aug 2025. https://thehackernews.com/2025/08/cursor-ai-code-editor-vulnerability.html

[42] The Register, “Postmark MCP server code hijacked”, 29 Sep 2025 (reporting Koi Security’s findings). https://www.theregister.com/2025/09/29/postmark_mcp_server_code_hijacked/

[43] Techloy, “Lovable walks back initial statement that it wasn’t breached”, Apr 2026. https://www.techloy.com/lovable-walks-back-initial-statement-that-it-wasnt-breached-offers-what-happened-on-april-2/

[44] Tianzhou, “How to prevent an AI agent from dropping your production database”, Bytebase blog, 18 Sep 2026. https://www.bytebase.com/blog/how-to-prevent-ai-agent-from-dropping-your-production-database/

[45] Check Point Research, “Caught in the Hook: RCE and API token exfiltration through Claude Code project files (CVE-2025-59536, CVE-2026-21852)”, 25 Feb 2026. https://research.checkpoint.com/2026/rce-and-api-token-exfiltration-through-claude-code-project-files/

[46] Amazon Web Services, Security Bulletin 2026-047-AWS, “CVE-2026-12957 and CVE-2026-12958 – Issues in Language Servers for AWS and Amazon Q Developer plugins”, 23 Jun 2026. https://aws.amazon.com/security/security-bulletins/2026-047-aws/

[47] Tea’s official statement (26 Jul 2025), quoted at https://simonwillison.net/2025/Jul/26/official-statement-from-tea/; Appknox, “What the Tea app breach reveals about mobile security”, Aug 2025, https://www.appknox.com/blog/tea-app-data-breach-security-flaws-analysis-appknox; Daring Fireball, “Apple pulls Tea and TeaOnHer”, 22 Oct 2025, https://daringfireball.net/linked/2025/10/22/apple-pulls-tea-and-teaonher

[48] ClassAction.org, “Tea lawsuit claims dating app illegally collects, distributes Illinois residents’ biometric data” (Honeycutt v. Tea Dating Advice, Inc., No. 2025CH08182), Aug 2025. https://classaction.org/news/tea-lawsuit-claims-dating-app-illegally-collects-distributes-illinois-residents-biometric-data

[49] Wiz, “s1ngularity: supply chain attack leaks secrets on GitHub”, 27–29 Aug 2025. https://wiz.io/blog/s1ngularity-supply-chain-attack

[50] Decrypt, “China state-backed hackers used AI in massive cyberattack, Anthropic says” (reporting Anthropic’s 13 Nov 2025 disclosure), Nov 2025. https://decrypt.co/348607/china-state-backed-hackers-ai-massive-cyberattack-anthropic

[51] Y. Zhang, “Cutting the Root of Hallucination: Structural Trimming for Vulnerability Mitigation in Code LLMs”, COLM 2025. https://cispa.de/en/research/publications/104235-cutting-the-root-of-hallucination-structural-trimming-for-vulnerability-mitigation-in-code-llms

[52] Cycode, “OWASP Top 10 for Agentic Applications” (summary of the OWASP list released 9 Dec 2025). https://cycode.com/blog/owasp-top-10-agentic-applications/

[53] NIST, SP 800-218 (SSDF) and SP 800-218A (Jul 2024), https://csrc.nist.gov/projects/ssdf/publications; AI RMF 1.0 (Jan 2023) and AI 600-1 Generative AI Profile (Jul 2024), https://www.nist.gov/itl/ai-risk-management-framework