SURFACE-ONLY RESEARCH MEMO

When the platform builds your app, the deployment topology becomes a security decision.

A passive, surface-only audit of 57,947 AI-built apps across the major vibe-coding platforms. We never authenticated, never POSTed a form, never requested a private record - everything below is visible from a single anonymous page load.

Create 27,375 | Lovable 19,015 | Replit 10,907 | Bolt 650

57,947
APPS SUCCESSFULLY RENDERED

10.22%
SHIP A SUPABASE ANON JWT TO THE BROWSER

41
LLM-VALIDATED CVE-2025-48757 PATTERN MATCHES

386
APPS WITH A CONCRETE SECRET IN CLIENT CODE

345
THIRD-PARTY API KEYS LEAKED IN JS

7
SERVICE_ROLE JWTS LEAKED (RLS BYPASS)

Generated 2026-05-07 - All findings derived from public homepages and the JS bundles linked from them. No customer hostnames, JWTs, secrets, or scrape excerpts appear in this report; counts and percentages only.

Findings memo

Authored locally by `nvidia/Llama-3.1-8B-Instruct-FP4` on Blackwell tensor cores via vLLM. The model only ever saw the structured indicator counts and (already anonymised) category histograms - never hostnames, JWTs, or scrape excerpts.

Corpus

Our analysis of the corpus reveals a significant number of hosts that were successfully rendered, with a render failure rate of 0.14%. Out of 57963 candidate hosts discovered, 57947 were successfully rendered, indicating a high level of success in our analysis.

Exposure pattern: anon JWT plus PII-named tables

The exposure pattern of shipping an anonymous JWT (anon JWT) to the browser alongside PII-named tables is a concerning one. Our analysis reveals that 10.22% of fetched hosts (5924) are shipping an anon JWT to the browser. Furthermore, when we look at the hosts that are shipping an anon JWT and have PII-named tables, we see a deterministic regex match of 3380 hosts, which is 5.83% of fetched hosts. However, when we apply a stricter validation using a Large Language Model (LLM), we find that only 41 hosts (0.07% of fetched hosts, 0.69% of anon-JWT hosts) meet this tighter criteria. This suggests that the LLM-validated count is a more defensible headline, and we will focus on this figure in our analysis.

Concrete leaks observed

Our analysis reveals that 0.67% of fetched hosts (386) have concrete secret leaks, including hosts with a third-party API key in client code, hosts shipping a service_role Supabase JWT, and hosts with any concrete leak. Specifically, 345 hosts have a third-party API key in client code, 7 hosts are shipping a service_role Supabase JWT, and 386 hosts have any concrete leak.

What this means for sovereign / on-prem AI

The findings of this analysis have significant implications for sovereign and on-prem AI. The exposure of PII-named tables alongside anon JWTs, as well as concrete secret leaks, can compromise the security and integrity of AI systems. The fact that 0.07% of fetched hosts (41) meet the stricter criteria for the anon JWT + PII tables pattern suggests that this is a relatively rare but still concerning issue. The concrete secret leaks observed in 0.67% of fetched hosts (386) are a clear indication of a security risk that needs to be addressed.

When the App Builds Itself, So Does the Breach

A field study of AI-built apps shipping live data to anyone who asks - and the case for keeping your AI on-prem.

> ***Note on numbers.*** *Every figure below is generated from* `reports/stats.json` *and traces back to a deterministic scan of public, rendered pages. We deliberately publish* ***counts and percentages only****. Specific apps, hostnames, project IDs, and excerpts are kept locally as our own audit trail and are not included in this article. If a number changes after a re-scan, this file is regenerated rather than edited by hand.*

TL;DR

- We surveyed **57963** candidate vibe-coding apps across **Lovable**, **Replit Agent**, **Bolt.new**, and **Create.xyz**, and successfully rendered **57947** of them.
- The same risk pattern appears across platforms - which is the point: this is a *deployment-topology* problem, not a Lovable-specific bug. See the per-platform breakdown table below.
- **10.22%** ship a Supabase anon JWT directly to the browser - expected if Row Level Security (RLS) is correct, catastrophic if it's not.
- An LLM-validated second pass over every Supabase-using host (running locally on Blackwell tensor cores at ~28 hosts/sec) found **41** apps shipping that anon JWT alongside client code that *concretely* names PII-bearing tables - the exact pattern that turned CVE-2025-48757 into a breach class.
- **0.67%** ship a concrete secret - service-role JWT, live third-party API key, or similar - to every visitor who loads the homepage. That's **345** apps with a leaked third-party API key and **7** apps shipping a `service_role` JWT that bypasses RLS entirely.
- We did not request a single user record. The risk surface is visible from the public homepage alone.

Why this matters

"Vibe coding" - describing an app and watching an LLM build it - is the most honest demo we have of where software is going. Lovable, v0, Bolt, Replit Agent and the rest are *good*. They produce real, shippable apps in minutes. The catch is the invisible part of the stack: the database, the auth model, the secrets boundary. Those are exactly the things an LLM is happy to hallucinate "into existence" with policies that compile, deploy, and look plausible - while leaving the front door open.

In March 2025, security researchers documented this directly: CVE-2025-48757 - a class of RLS misconfigurations in Lovable-built apps that left a measurable share of projects exposing user data through the public Supabase REST API. The pattern has continued: an educational app on Lovable's Discover page leaked **18,000+ users** in February 2026, and a BOLA-class flaw disclosed in April 2026 affected every project created before November 2025.

This study asks the simple follow-up: **a year later, how many apps are still shipping the same shape of risk?** And we answer it the way the original CVE did - with passive, surface-only observation.

Method

Everything here is reproducible from a single self-hosted box.

1. **Discovery.** Five passive sources are run *per platform* and merged: Certificate Transparency (`crt.sh`), Internet Archive Wayback CDX (paged to one million rows), recent Common Crawl monthly indexes, the public URLscan.io search index (`domain:<apex>`), and each platform's public showcase page (`lovable.dev/discover` , `create.xyz/explore` , etc.). Wildcard records and ephemeral preview subdomains are dropped.
2. **Rendering.** Each candidate is rendered through a self-hosted Firecrawl Simple instance with a 4-second post-load wait, so we observe what an anonymous visitor's browser actually sees - not the empty SPA shell.
3. **Detection (deterministic).** Regex / JWT decoding runs over the rendered HTML, the rendered markdown, and every JavaScript bundle the page links to. Indicators include: Supabase project URLs, anon vs service-role JWTs, Supabase storage URLs, client-side `from('table')` call sites, and a panel of third-party secret patterns (OpenAI, Anthropic, Stripe, Google, AWS, GitHub, SendGrid, Resend).
4. **Validation (local LLM).** A second pass classifies every Supabase-using host with `nvidia/Llama-3.1-8B-Instruct-FP4` running locally on a single DGX Spark (Blackwell GB10) under vLLM with native NVFP4 tensor-core kernels. The model receives only structured indicator counts and (already anonymised, already PII-keyword-filtered) table-name strings - never hostnames, JWTs, URLs, or excerpts. It returns a strict JSON `{category, confidence, rationale}` per host. This stage runs at roughly 28 hosts per second at concurrency 48, so the entire Supabase-using subset of the corpus is rescored in under three minutes.
5. **Aggregation.** Per-host evidence is kept local for our own audit and is never published. Only category counts and percentages cross the fence into this article. A publication-check script refuses to ship any output that contains a customer hostname, a JWT-shaped string, a third-party-secret shape, or a verbatim excerpt from the local evidence directory.

We never authenticate, never POST a form, never request a private database row. The entire study is what an unauthenticated browser already receives.

Findings

Reach of the corpus in our sample

- Hosts surveyed: **57947** (from **58028** scanned, **57963** candidates discovered across four platforms)
- Connected to a Supabase backend (URL visible in client): **10.53%**

Per-platform breakdown

Platform	Hosts fetched	Anon JWT exposed	Concrete leaks
Lovable	**19015**	**30.43%**	**246**
Replit	**10907**	**0.17%**	**102**
Bolt	**650**	**18.31%**	**16**
Create	**27375**	**0%**	**22**

The headline pattern: anon JWT + PII-named tables

The thing that turned CVE-2025-48757 into a real breach class was not the presence of an anon JWT - that's expected. It was the *combination* of an anon JWT shipped to the browser **and** client code that names tables holding personal data. If the table's RLS isn't correct, those rows are public.

We measured this two ways. The deterministic regex pass flags any anon JWT plus any client-side `from('table')` call where the table name contains a PII-suggestive keyword (`users` , `customers` , `payments` , `messages` , etc.). The LLM pass takes the same indicators, removes the hostname, and asks the local NVFP4 model whether the table names *actually* look like PII storage in context. The LLM is much stricter, which is exactly what we want for a publishable number.

Pattern	Hosts	% of fetched
anon JWT exposed	**5924**	**10.22%**
anon JWT + PII-named table (deterministic)	**3380**	**5.83%**
anon JWT + PII tables (LLM-validated)	**41**	*(see narrative below)*
service-role JWT shipped to browser	**7**	**0.01%**
concrete third-party secret in client code	**345**	**0.6%**
concrete secret of any kind in client code	**386**	**0.67%**

Every count above reflects a host whose risk surface we could verify *without ever asking for a user record*. We can see the shape of the exposure from the homepage alone.

Severity distribution (max severity per host)

Severity	Hosts
critical	**80**
high	**10**
medium	**262**
low	**283**
info-only	**47071**

Top indicator kinds

> *Counts are hosts affected, not raw matches.*

- `nextjs_build_tag` - 27431 hosts (47.34%)
- `create_build_tag` - 26876 hosts (46.38%)
- `lovable_build_tag` - 16364 hosts (28.24%)
- `vite_build_tag` - 15582 hosts (26.89%)
- `supabase_url` - 6001 hosts (10.36%)
- `supabase_anon_jwt` - 5924 hosts (10.22%)
- `supabase_table_hint` - 4939 hosts (8.52%)
- `replit_build_tag` - 2113 hosts (3.65%)
- `bolt_build_tag` - 637 hosts (1.1%)
- `google_api_key` - 265 hosts (0.46%)

Why on-prem / sovereign AI helps

The platforms we studied are not the villains. They are LLMs assembling systems faster than humans can audit them. The structural problem is that the *default deployment topology* sends raw user data through hosted services that the builder never directly configures, gated by access policies the LLM generates from a natural-language description.

Bringing the AI on-prem - same models, same builder UX, but on hardware you control, with your own data inside your own VPC - flips three things:

1. **The blast radius is yours.** A misconfigured policy still hurts, but it hurts inside an environment your security team can audit, isolate, and roll back. There is no public REST endpoint enumerating your tables.
2. **The LLM sees private context, not vice versa.** Sovereign-AI deployments make it cheap to feed the model the actual schema, RLS posture, and compliance constraints, instead of guessing from a prompt.
3. **The audit trail lives in your logs.** Every prompt, every generated policy, every deploy is recordable on your side of the wall - which is exactly what your auditors will ask for the morning after CVE-N+1 lands.

The point of this study is not "stop using AI builders". It is: **the moment those builders are pointed at production user data, the deployment topology matters as much as the model.** Putting the AI - and the data it touches - on your own infrastructure is how you keep the upside while taking the systemic risk off the public internet.

Reproducibility & data handling

The full pipeline is in this repository under `scripts/` and `agent/`. Three things to call out:

- **Evidence is local-only.** `reports/evidence/` and `reports/findings.jsonl` contain raw scrape output and per-host indicator excerpts. Both are `.gitignore`'d.
- **Stats are reproducible.** `reports/stats.json` is regenerated from the evidence directory by `scripts/aggregate.py` and contains only the numbers cited above.
- **A publication check refuses to leak.** `scripts/publication_check.py` walks `article/` and `reports/stats.json` and fails the build if it finds any reference to a host in our corpus, any JWT-shaped string, any secret-shaped string, or any verbatim excerpt from local evidence. The numbers above passed that check on **2026-05-07T01:54:59.603213+00:00**.

Acknowledgements

Methodology and prior reporting we built on:

- Matt Palmer, *Statement on CVE-2025-48757* (March 2025)
- VibeScan, *CVE-2025-48757: The Lovable Supabase RLS Vulnerability Explained*
- Bastion, *Lovable Data Breach April 2026* (BOLA disclosure)

We owe them the framing; this study refreshes the numbers a year on.

Appendix: numbers

All numbers below come from `reports/stats.json`, the only artifact this project lets out of the box. The local evidence directory (full per-host scrapes, indicators, hostnames) stays on the research machine.

Headline counts (all platforms combined)

Metric	Value
Successfully rendered hosts	57,947
Anon JWT exposed	5,924 (10.22%)
Anon JWT + PII tables (deterministic regex)	3,380 (5.83%)
Anon JWT + PII tables (LLM-validated, stricter)	41
service_role JWT shipped to browser	7 (0.01%)
Third-party secret in client code	345 (0.6%)
Any concrete leak	386 (0.67%)
Severity: critical (max per host)	80
Severity: high	10
Severity: medium	262
Severity: low	283
Severity: info	47,071

Per-platform breakdown

Each row is one vibe-coding platform's hosts, attributed by the apex domain of the scanned URL (e.g. `*.lovable.app`, `*.replit.app`) and falling back to a build-fingerprint match for hosts on custom domains.

Platform	Fetched	Anon JWT %	PII tables	3rd-party key	service_role	Any leak
Create	27,375	0.0%	0	18	0	22
Lovable	19,015	30.43%	3,307	216	7	246
Replit	10,907	0.17%	16	96	0	102
Bolt	650	18.31%	57	15	0	16

Exposure category distribution (all platforms, deterministic)

Category	Hosts	% of fetched
none	51,558	88.97%
supabase-anon-only	2,550	4.4%
supabase-anon-pii-tables	3,262	5.63%
supabase-service-leaked	7	0.01%
third-party-secret-leaked	345	0.6%
pii-form-only	225	0.39%
mixed	0	0.0%

LLM-validated second pass

A local LLM rescored every Supabase-using host using ONLY structured indicators. The model is much stricter than the keyword regex; this is the more defensible figure. Two independent runs at temperature 0 gave 37 and 42 hosts respectively, so the answer is stable within roughly +/-5 hosts (12% variance).

Field	Value
Backend / model	vllm / llm
Hosts classified	6,101
Throughput	28.01 hosts/s
Concurrency	48
Took	217.8 s

Hosts discovered per platform (seeds)

Platform	Hosts in seeds
create	27,639
lovable	18,643
replit	11,025
bolt	656

Provenance & safety

- Rendering: self-hosted Firecrawl Simple on the research box.
- Detection: deterministic regex / JWT decode over rendered HTML, markdown, and every JS bundle.
- LLM second pass: vLLM serving NVFP4 weights on Blackwell tensor cores; structured-only payloads.
- Publication gate: `scripts/publication_check.py` refuses any output containing a customer hostname, JWT-shaped string, third-party-secret shape, or verbatim excerpt from local evidence.
- This PDF was generated from `article/sovereign-ai-warning.rendered.md` + `reports/findings_memo.md` + `reports/stats.json`; it adds no narrative not already present in those publishable artefacts.