

Postmortem: Railway’s Build Escape Reached AI Keys for 1,852 Users, and Five Trust Boundaries Had to Fail First

The attacker who broke into Railway in October 2026 was not after customer data. They were after AI provider keys, and Railway’s own forensics say that is exactly what the queries went looking for.

THE SHORT VERSION

Railway’s incident report of 10 November 2026 describes an intrusion that started 20 October at 07:28 UTC with a filesystem escape in multi-tenant build infrastructure and ended 21 October at 20:19 UTC with every session and token invalidated. In between: a misconfigured OIDC trust, an SSH key, 353 build histories read, and linked-account tokens decrypted for 1,852 users. Forensics showed AI-provider credentials were the primary target.

Credential theft has new economics. A stolen database row has to be sold, laundered, and matched to a buyer. A stolen inference key can be consumed directly by whoever holds it, and the cost appears on someone else's invoice. That changes what an attacker searches for once inside, and it should change what you assume about the blast radius of your own secret store.

Railway published a detailed timeline with hours, counts, and the names of the cloud providers whose refund policies decided who ate the loss. The useful part of this story is the chain. Five separate trust boundaries gave way in sequence before the keys were reachable, and each one individually looked like an acceptable design decision.

How the attacker got from a build container to 1,852 users' AI keys

The sequence as published in Railway's incident report, with timestamps where Railway gave them.

20 Oct, 07:28 UTC. A previously unknown vulnerability in Railway's multi-tenant build infrastructure allowed partial escape of filesystem isolation, enough to read files on the host VM.

21 Oct, 03:26 UTC. A misconfigured OpenID Connect trust relationship exposed a short-lived cloud credential intended for CI/CD. Short-lived does not mean low-value when the attacker is live on the box.

Next came the pivot. That credential led to further internal credentials and an SSH private key, giving access to a machine coordinating customer workloads. The boundary between build runner and control plane was gone at that point.

Per The Pragmatic Engineer's write-up, the attacker then located Railway's internal monorepo stored as a build history, downloaded it, and used the code to map internal routes and trust relationships. The platform built itself on itself, so its source code was sitting in its own build artifacts.

Using that map, the attacker reached a Temporal cluster holding build histories and read

stored histories from 353 builds, including variables and secrets passed to those builds. Linked-account tokens for 1,852 users were decrypted, including BYOK agent keys and GitHub OAuth tokens.

Containment ran through 21 October: SSH disruption from 08:14 UTC, GitHub-login disruption at 09:22, token invalidation starting 11:05, all sessions and tokens invalidated at 20:19. Builds were restored for all plans on 22 October at 22:49 UTC after infrastructure hardening. Some static-egress SSH services stayed affected until 4 November, 17:30 UTC.

Roughly 37 hours from first entry to full token invalidation. Railway also reported that up to 1.85% of workspaces could have had environment variables potentially compromised, and that no logins to Railway accounts were made using credentials stolen in the incident.

Why the monorepo in the build history is the detail that matters

Every step above is defensible in isolation. Multi-tenant build isolation with a container boundary is standard. OIDC federation for CI/CD is the recommended modern alternative to long-lived static keys. Storing build histories in a durable workflow engine is sensible engineering. Dogfooding your own platform for your own builds is normal and, in most respects, good practice.

The compounding is what did the damage. The OIDC misconfiguration mattered because the escape put the attacker in a position to use it. The SSH key mattered because the OIDC credential led to it. The monorepo converted a blind lateral position into a guided one: the attacker stopped probing and started working from a map of internal routes and trust relationships. And the build histories retained the secrets that had been passed into builds, which is a retention decision rather than a security vulnerability.

That last point deserves emphasis. Nothing about keeping 353 build histories was a bug. It was a feature behaving exactly as designed, and it became the payload.

My read: the highest-leverage fix in this chain is secret retention in build history, not the escape vulnerability, which is the hardest and most expensive thing to guarantee. If the Temporal cluster had held build metadata without the resolved variable values, the

attacker would have reached step five and found nothing worth decrypting. Treat secret lifetime in build artifacts as a security control with its own owner, not as a side effect of your observability choices.

Who paid for the stolen inference, and why the answer differed by provider

This is the part of the report I find most instructive for anyone running a BYOK product. Financial exposure from stolen inference keys turns out to be a function of the individual provider's fraud posture.

According to Railway's report and secondary coverage: AWS Bedrock and Google Gemini covered the fraudulent usage under existing fraud-refund policies. OpenAI directed affected customers to file support requests, which puts the burden and the outcome on each customer individually. OpenRouter reportedly held a no-refund position, and Railway covered that cost.

So the same stolen credential, used for the same unauthorized inference, in the same incident, produced different financial outcomes depending on which logo was on the key. If you are building a product that holds customer AI keys, that distribution is your actual risk model. The hyperscalers absorb fraud because they have the fraud machinery and the balance sheet for it. Aggregators and smaller providers may not.

Railway's forensic review of the attacker's queries indicated AI-provider credentials were the primary target, apparently for unauthorized usage or distillation attacks. Distillation is the quieter half of that sentence: using someone else's paid model access to generate training data for your own model. That motive makes a stolen key valuable even to an actor who has no intention of reselling anything, and it means the spend pattern may look like legitimate heavy usage rather than obvious abuse.

Railway broke SSH and GitHub login rather than wait for a

clean cutover

The containment is worth separating from the breach. Entry at 07:28 on 20 October, first customer-visible disruption at 08:14 on 21 October, full invalidation of all sessions and tokens at 20:19 the same day. Railway reset affected linked-account tokens, invalidated API and project tokens and all user sessions, rotated storage-bucket credentials, and redeployed affected services.

> ✓ **The thing that worked**
> Railway's token and session architecture supported a global invalidation it could actually execute, and the report states no logins to Railway accounts were made with stolen credentials. Mass revocation is a capability you either have as a tested button or do not have at all, and most platforms find out which on the day they need it.

The honest counterweight: 37 hours from entry to full invalidation is fast response after detection, not fast detection. The report does not establish when Railway first noticed, only when disruption began. Reading the gap between the 07:28 entry on day one and the 03:26 OIDC pivot on day two, the attacker appears to have had roughly 20 hours of unobserved working time inside build infrastructure. I could be wrong about that window, since the report does not say. But it is the thing worth attacking in your own environment: how long a process can read host files from inside a build container before anything complains.

Four changes worth making before your next build runs

These follow from the chain above, not from a generic hardening checklist.

Strip resolved secrets from build history. Store references instead of values. If your workflow engine, CI system, or build cache retains the environment a build ran with, you have a secondary secret store that nobody reviews and nobody rotates. Railway's attacker got variables and secrets from 353 builds out of a place whose job was durability.

Scope OIDC trust to the narrowest subject you can express. Federated short-lived credentials beat static keys, but the trust policy is the entire security boundary. If any workload in a shared build fleet can assume the role, escaping into that fleet is equivalent to holding the role. Ask specifically: can a customer's build container satisfy the subject

condition on our CI role? If the answer takes more than a minute to determine, that is the finding.

Decide now who pays for stolen inference, provider by provider. Go read the fraud policy for every AI provider whose keys your product holds. The Railway outcome shows the spread: Bedrock and Gemini refunded, OpenAI routed to per-customer support tickets, OpenRouter did not refund. If you hold BYOK keys, that spread is a line item in your risk register, and the customers whose provider does not refund are the ones who will be angry at you rather than at the provider.

Alert on anomalous inference spend, not only on anomalous logins. The attacker’s goal was consumption, and consumption generates a signal before anyone notices a breach. Rate and cost anomalies against a per-key baseline are cheaper to build than host intrusion detection, and they fire during the spend, which is the only window where the loss is still shrinking.

One thing I cannot tell you from the published material: whether the escape vulnerability was in a widely used container or build component or was specific to Railway’s own stack. The report calls it previously unknown and does not name the component. If it turns out to be upstream, the set of platforms with the same first link in the chain is considerably larger than one.



WRITTEN BY

Artur Markus

Software engineer in Basel building AI infrastructure and agentic systems, with a background in pharmaceutical automation, GxP compliance, and validation. I help teams turn AI from a chatbot into a reliable operational layer.

Book a meeting →