SYNTHETIC EVIDENCE SAMPLE

Supabase RLS sample audit report

A compact example of how an authorization finding can be reproduced, explained, remediated and verified without using client or production data.

This document uses only a deliberately vulnerable public fixture. It does not describe a real company, Supabase project, customer record or discovered production vulnerability.

01 / Executive summary

The fixture models a **public.notes** table with two synthetic users. On the broken branch, the authenticated database role has table privileges while Row Level Security is not enabled. The same five-test suite shows that one user can read, insert, update and delete rows belonging to another user.

SEVERITY	FINDING	STATUS
HIGH	Cross-user row access because the table has no enforced RLS boundary	Reproduced on broken; remediated on fixed
INFO	Regression coverage for SELECT, INSERT, UPDATE and DELETE	Five tests pass on fixed

Fixed-branch verification

```
Test Files 1 passed (1)
Tests 5 passed (5)
```

02 / Finding RLS-001

Cross-user row access

High severity in this fixture / public.notes

Security property

An authenticated user must only access rows they own.

Evidence

The suite seeds one synthetic row for user A and one for user B, switches to the fixture's authenticated role, and sets user B's synthetic subject claim. On the broken branch:

- user B can read user A's row;
- user B can insert a row owned by user A;
- user B can modify user A's row; and
- user B can delete user A's row.

The raw test command exits non-zero with four failed isolation assertions and one passing self-access assertion. The self-access check prevents a blanket revoke from creating a misleading pass.

Root cause

The authenticated role has table privileges, but the broken branch does not apply an RLS policy file. PostgreSQL therefore applies no row filter to this role. Real-world reachability would also depend on grants, exposed schemas and API paths.

Remediation demonstrated

- Enable RLS on public.notes.
- Apply ownership conditions to reads and deletes.
- Apply ownership checks to inserts and updates.
- Verify legitimate self-access beside forbidden paths.

These policies are fixture-specific. Production remediation must follow the application's real roles, tenant model and privileged paths.

03 / Scope and handoff

What this sample proves - and what it does not

Included in the synthetic review

- public.notes and the authenticated database role
- synthetic subject claims used by the local test harness
- SELECT, INSERT, UPDATE and DELETE ownership paths
- the fixed branch policy file and five regression checks

Not included

- a live Supabase project, production data or customer records
- Supabase Auth sign-in or PostgREST/Data API exposure
- Storage, Realtime, Edge Functions, RPCs or network controls
- secret scanning, penetration testing or compliance certification

Example production follow-up

- Inventory tables and views reachable through the public API.
- Test anonymous, authenticated, cross-user and cross-tenant paths.
- Review membership removal, invitations, admin and service-role boundaries.
- Inspect SECURITY DEFINER functions, callable RPCs and relevant storage policies.
- Keep actor, target, expected denial and observed evidence in the handoff.

Handling and limitations

Do not send production passwords or service-role keys in chat. Share only the minimum anonymized schema, policy and test context agreed for the work. Findings are limited to the reviewed scope and evidence available at review time.

A scoped audit reduces uncertainty. It is not a security, compliance, uptime or breach-prevention guarantee.

Public proof: github.com/cekuu35/supabase-rls-leak-demo