Tenant isolation review

Teamdesk

Can one Teamdesk customer read or change another customer's data? This review tested every table and access rule in the database to find out, fixed what it found, and tested again.

CLIENT	VERSION	DATED	REVIEWER
Teamdesk (fictional)	1.1, after retest	30 November 2026	Mudit Atrey, Pradite
SCOPE	SOURCE	DATABASE	RESULT
9 tables, 19 policies, 2 functions	Migrations, read-only	PostgreSQL 17	4 found, 4 closed

Sample report. Teamdesk is a fictional app that Pradite wrote to show what a review contains. Each mistake in it copies one we found in public Supabase repositories in November 2026. No real company, user or data is involved.

1 SUMMARY

Before the fix, any signed-in user could make themselves owner of any organisation and read its documents and invoices. After the fix, all four findings are closed, sign-up and invitations still work, and a test in CI fails if any of them returns.

ID	FINDING	SEVERITY	STATUS
TDK-01	Any signed-in user can join any organisation as owner	Critical	Fixed, verified
TDK-02	Every organisation is visible to anonymous visitors and new users	High	Fixed, verified
TDK-03	Any signed-in user can read every pending invitation and its token	High	Fixed, verified
TDK-04	Any signed-in user can write audit entries in any organisation, in anyone's name	Medium	Fixed, verified

2 METHOD

- Every migration was replayed into a private PostgreSQL 17 database with Supabase's auth schema and roles. Fake data only; production was never contacted.
- Two organisations were seeded with realistic rows. Every read, insert, update and delete on every table was then tried as an anonymous visitor, a member of organisation A, an outsider and an invited user.
- On each membership table, the outsider tried to join organisation A. Where that worked, every table was checked again to see what had become readable. TDK-01 was found this way.
- Four open-source RLS scanners were run on the same database and every flag they raised was read by hand (section 4).
- Each finding was reproduced with a script before it was written up, and again after the fix.

3 FINDINGS

TDK-01

Any signed-in user can join any organisation as owner

SEVERITY	AFFECTED	CLASS	STATUS
Critical	`organization_members`	CWE-863, OWASP A01	Fixed, verified

WHAT HAPPENS

The insert rule checks that users add themselves. It never checks which organisation they join or which role they take. Anyone with an account can add a row that makes them owner of another organisation, and from then on every rule that asks "is this person a member?" says yes.

IMPACT

Read access to any organisation's projects and documents, and, because the attacker chooses the role, to owner-only data such as invoices. Organisation IDs are easy to obtain from links, invitations and TDK-02.

POLICY

```
create policy "join as yourself" on public.organization_members for insert
  with check (user_id = auth.uid());
```

EVIDENCE, REPRO/01, AS THE OUTSIDER

```
 step   | victim_docs | victim_invoices
--------+-------------+-----------------
 before |           0 |               0
 after  |           1 |               1
```

FIX

Three narrow rules replace the open one: admins add members, the creator of a new organisation becomes its first owner, and an invited email joins as a plain member. The "new organisation" and "invited" checks run in `security definer` functions. A plain `not exists (...)` inside a policy runs under the caller's own row level security, sees nothing, and always passes. We found that second mistake in public repositories as well.

```
create policy "admins add members" on public.organization_members for insert
  with check (public.is_admin(org_id));
create policy "creator becomes first owner" on public.organization_members for insert
  with check (user_id = auth.uid() and role = 'owner' and public.can_claim_new_org(org_id));
create policy "invitee joins as member" on public.organization_members for insert
  with check (user_id = auth.uid() and role = 'member' and public.has_invite(org_id));
```

RETEST

```
ERROR:  new row violates row-level security policy for table "organization_members"
```

TDK-02

Every organisation is visible to anonymous visitors and new users

SEVERITY	AFFECTED	CLASS	STATUS
High	`organizations`	CWE-200, OWASP A01	Fixed, verified

WHAT HAPPENS

The read rule includes an onboarding clause: "or the user has no memberships yet". An anonymous visitor has no memberships, and neither does anyone who has just signed

up, so both see every organisation.

IMPACT

The customer list is public, and it supplies the organisation IDs that TDK-01 needs.

POLICY

```
using (public.is_member(id) or not exists (
  select 1 from public.organization_members m where m.user_id = auth.uid()))
```

EVIDENCE, REPRO/02, NO LOGIN

```
 who               | organisations_visible
-------------------+-----------------------
 anonymous visitor |                     2
```

FIX

```
using (public.is_member(id) or created_by = auth.uid())
```

A new `created_by` column, defaulting to `auth.uid()`, lets creators see the organisation they have just made before their membership row exists.

RETEST

```
 anonymous visitor |                     0
```

TDK-03

Any signed-in user can read every pending invitation and its token

SEVERITY	AFFECTED	CLASS	STATUS
High	`invitations`	CWE-200, OWASP A01	Fixed, verified

WHAT HAPPENS

The rule written so that "the invited person can look up their invite" only checks that the reader is logged in.

IMPACT

Anyone can read every invitee's email address and token, and can use a token to join an organisation they were never invited to.

POLICY

```
create policy "look up invites" on public.invitations for select using (auth.uid() is not
null);
```

EVIDENCE, REPRO/03, AS THE OUTSIDER

```
 who      | email               | token_prefix
----------+---------------------+--------------
 outsider | invitee@victim.test | 6a60f3...
```

FIX

```
using (public.is_admin(org_id) or lower(email) = lower(coalesce(auth.jwt() ->> 'email', '')))
```

RETEST

```
(0 rows) for the outsider; the invitee still sees their own invitation (check 9)
```

TDK-04

Any signed-in user can write audit entries in any organisation, in anyone's name

SEVERITY	AFFECTED	CLASS	STATUS
Medium	audit_log	CWE-863, OWASP A01	Fixed, verified

WHAT HAPPENS

The insert rule only checks that the writer is logged in. It checks neither the organisation nor the `actor_id`.

IMPACT

An outsider can plant "Alice deleted all projects" in another organisation's audit trail. An audit log that can be forged is a liability when a customer or auditor asks for it.

EVIDENCE, REPRO/04

```
 result              | rows
---------------------+------
 forged entry stored |    1
```

FIX

```
with check (actor_id = auth.uid() and public.is_member(org_id))
```

RETEST

```
ERROR:  new row violates row-level security policy for table "audit_log"
```

4 OPEN-SOURCE SCANNERS

We run the open-source RLS scanners on every review and pass on what they find. On Teamdesk they raised 79 flags (pgrls), 55 (rls-doctor) and 40 (rlsgrid), mostly general hardening advice. This table shows which of them pointed at the four real findings.

ID	PGRLS 0.57	RLSGRID 0.5	RLSAUTOTEST 0.7	RLS-DOCTOR 0.5
TDK-01	Not flagged	Not flagged	Marked unreliable	Not flagged
TDK-02	Not flagged	Flagged	Not flagged	Not flagged
TDK-03	Warning	Flagged	Not flagged	Not flagged
TDK-04	Warning	Not flagged	Not flagged	Not flagged

In 16 public multi-tenant Supabase repositories drawn at random in November 2026, 4 had the TDK-01 pattern. Across the 9 cases we have found so far, none of these four scanners flagged it. The samples are small, and a repository's migrations can differ from its live database.

5 FLAGS REVIEWED AND SET ASIDE

FLAG	DECISION
`plans` readable by everyone, `using (true)`	Intended. It is the public price list.
Signed-in users can create organisations	Intended. That is sign-up. After the fix they are also recorded as its creator.
`security definer` helpers `is_member`, `is_admin`	Supabase's recommended pattern. Both pin `search_path` and only read the caller's own memberships.
FORCE ROW LEVEL SECURITY missing on every table	Applies only to connections made as the table owner. Supabase clients connect as `anon` or `authenticated`.

6 NOT IN SCOPE

- The live database. Policies changed in the dashboard and not in migrations are invisible to a migrations review; a schema-only dump closes this gap.
- Edge functions, API routes and anything that uses the service role key, which bypasses row level security.
- Storage bucket policies, auth settings and the front end.
- Denial of service, rate limits, and anything that needs the running app.

This is a review of database access rules and their configuration. It is not a certified penetration test.

7 DELIVERABLES AND RETEST

FILE	CONTENTS
`repro/01..04_*.sql`	One script per finding. It creates its own fake rows on a staging copy, prints the result, and rolls everything back.
`fix/20261130000001_fix_tenant_isolation.sql`	A migration that closes all four findings.
`tests/tenant_isolation_test.sql`	Ten checks: seven that must be blocked and three that must still work. It fails before the fix and passes after. Run it in CI with `psql "$STAGING_DB_URL" -v ON_ERROR_STOP=1 -f tests/tenant_isolation_test.sql`

RETEST, 30 NOVEMBER 2026

```
before fix   ERROR:  FAIL: an outsider joined another organisation as owner   exit 3
after fix    ALL 10 TENANT ISOLATION CHECKS PASSED                            exit 0
full review rerun on the fixed schema: 0 findings
```