LeadBox — Supabase Data Leak Check

Prepared for: LeadBox **Date:** 28 September 2026 **Prepared by:** Sixthgear — Larik S **Project checked:** `your-project-ref.supabase.co` *(project reference redacted for this sample report)*

Summary

We checked whether LeadBox's customer lead data — names, emails, deal values — can be read, changed, or deleted by people who shouldn't have access to it. We did this the same way a stranger with your app's public website key could: by sending the same kind of requests your own app sends, from outside, with no password and no special access. We did **not** log into your admin panel, did **not** need your database password for the parts that matter most, and did **not** view or keep any of your customers' actual data — only whether a request succeeded, and how many rows came back.

Bottom line: right now, anyone on the internet — no account, no login, nothing more than your site's public key — can read, edit, and permanently delete every lead in your `leads_open` table, and any signed-up user can impersonate another user's leads. We found the same table, `leads_fixed`, correctly protected when set up the way we recommend below — proof that this is fixable in an afternoon, not a rebuild.

Findings

Severity	What's exposed	Who can exploit it	Proof
🔴 Critical	The entire leads table — every customer's name, email, and deal value — can be downloaded by anyone	Anyone on the internet, no account needed	A plain web request using only your site's public key returned data and revealed the column names (`id`, `user_id`, `name`, `email`, `deal_value`, `created_at`). Row count and contents redacted for this report.
🔴 Critical	Any visitor or logged-in user can edit any lead — including handing it to a different user	Anonymous visitors **and** any signed-up user	An edit request sent as an unrelated visitor, and separately as a different logged-in user, both succeeded (`200 OK`) against a lead they didn't own. Values redacted.
🔴 Critical	Any visitor or logged-in user can permanently delete any lead	Anonymous visitors **and** any signed-up user	A delete request sent as an anonymous visitor, and separately as a different logged-in user, both succeeded (`200 OK`, 1 row removed).
🔴 Critical	A logged-in user can create a fake lead and assign it to someone else's account (identity spoofing)	Any signed-up user	An insert request naming another user's account as the owner succeeded (`201 Created`).
🔴 Critical	A logged-in user can take an existing lead away from its real owner	Any signed-up user	An edit request changing only the "owner" field on someone else's lead succeeded — the lead's ownership changed.
🟠 High	Junk or fake data can be inserted directly into the	Anonymous visitors	The insert rule protecting this table has no restriction at all; a test insert with no login

	leads table, bypassing your app entirely		succeeded.

Every check above was confirmed twice: once by reading the table's security rules directly, and once by actually attempting the action (as a fake visitor and two throwaway test accounts we created and deleted immediately after). Both methods agreed.

What this means for your business

Data exposure. Every name, email address and deal value your sales team has entered is downloadable by anyone who opens your browser's network tab on your own public website — no hacking skill required, just curiosity. If any of those leads are EU or UK residents, this is a reportable personal-data exposure under GDPR/UK GDPR.

Data integrity. Because anyone can edit or delete a lead, your pipeline numbers, deal values, and follow-up history can't be trusted until this is fixed — a competitor, a disgruntled user, or an automated scraper could quietly corrupt or wipe your data without leaving a normal trace.

Account takeover of your data, not your login. The "assign to another user" and "reassign ownership" issues mean a logged-in user (a real customer of yours) can make your data say a lead belongs to them, without ever guessing anyone's password.

The fix

The underlying problem is the same in every row above: the table's Row Level Security (RLS) policies either don't check who's asking, or don't specify a role at all (so they apply to literally everyone, including anonymous visitors — Supabase calls this having "no `TO` clause"). The fix is to replace those policies with ones that check the request's caller against the row's owner.

Run this in the Supabase SQL editor, against `leads_open`:

```
-- Remove the policies that let anyone read, edit, delete or insert without restriction
drop policy if exists "public read" on public.leads_open;
drop policy if exists "Service role full access" on public.leads_open;
drop policy if exists "Service role full delete" on public.leads_open;
drop policy if exists "anyone can insert" on public.leads_open;

-- Replace them with ownership-checked policies, explicitly for signed-in users only
create policy "read own leads"
  on public.leads_open
  for select
  to authenticated
  using (auth.uid() = user_id);

create policy "insert own leads"
  on public.leads_open
  for insert
  to authenticated
  with check (auth.uid() = user_id);

create policy "update own leads"
  on public.leads_open
```

```
  for update
  to authenticated
  using (auth.uid() = user_id)
  with check (auth.uid() = user_id);

create policy "delete own leads"
  on public.leads_open
  for delete
  to authenticated
  using (auth.uid() = user_id);
```

The `with check (auth.uid() = user_id)` clause on both `insert` and `update` is what closes the identity-spoofing and ownership-reassignment issues — it stops a user from writing a row that claims to belong to somebody else, whether that's a brand-new row or one they already own.

If some of your leads genuinely need to come in from an anonymous contact form (public website visitors who aren't logged in), that's a legitimate exception — just say so and we'll write a narrower insert-only policy for that specific case instead of opening the whole table.

How to verify the fix

1. Open your browser's dev tools on your own live site, find a request to `/rest/v1/leads_open`, and repeat it without being logged in — it should now fail.
2. Or, the fast way: ask us to re-run this same check against `leads_open` after you've applied the SQL above. It's included free as part of the fix pack (see pricing below).

After the fix: the same checks on `leads_fixed`

`leads_fixed` is a second table in your project, already set up the way we recommend above. We ran every check against it too, so you can see exactly what "fixed" looks like — this isn't a hypothetical, it's the same test suite, same project, same day:

Check	Result on `leads_open` (before)	Result on `leads_fixed` (control)
Anonymous read (public key only)	**Returns data**	Blocked — 0 rows returned
Read another user's lead	**Succeeds**	Blocked (`403`)
Read as an anonymous visitor	**Succeeds**	Blocked (`403`)
Edit another user's lead	**Succeeds**	Blocked (`403`)
Edit as an anonymous visitor	**Succeeds**	Blocked (`403`)
Delete another user's lead	**Succeeds**	Blocked (`403`)
Delete as an anonymous visitor	**Succeeds**	Blocked (`403`)
Create a lead owned by someone else	**Succeeds**	Blocked (`403`)
Reassign an existing lead to someone else	**Succeeds**	Blocked (`403`)

Nine for nine, correctly blocked. Zero findings from the policy review, zero from the live test. This is the standard we bring every table in your project up to.

Scope and limits — what we did NOT test

We're explicit about this because a clean result is easy to oversell:

- **Only the tables named above.** `leads_open` and `leads_fixed` were checked in detail. Any other table in your project was not examined as part of this report.
- **Only direct-ownership access.** Both tables use a simple "one column holds the owner's user ID" pattern. If any part of your app uses team, organization, or shared-access permissions, those need a separate, tailored review.
- **Not covered at all:** your application code, Edge Functions or other backend routes, sign-up/login settings (open registration, email verification, session length), API keys that may already be exposed in your frontend's published code, and any Supabase schema other than the one reviewed here.
- **A point-in-time check.** This reflects your project's configuration on the day we ran it. A schema change, a new table, or a policy edited next week isn't covered by this report — that's what the monthly re-check below is for.
- **No customer data was viewed, copied, or retained.** Every check either counted rows or read column names; the two temporary test accounts and their test rows were deleted immediately after each check and independently verified as gone.

Pricing

Service	Price
Audit (this report)	**$149** (launch price)
Fix pack — we apply the SQL fixes above and re-verify	**$390**
Monthly re-check — catches new tables and policy drift going forward	**$49/month**

Invoicing by bank transfer.