

SAMPLE REPORT · Example app, not a real customer. "PlantPal" is fictional; every key, ID and email below is made up and redacted.

# Security Report: PlantPal

Full Report (\$29) · Prepared by LaunchSafeAI · Sample dated Oct 23, 2026

App  
plantpal.example (fictional)

Built with  
Lovable → Cursor, React + Vite

Backend  
Supabase (Postgres, Auth), Stripe Checkout

Scope  
Live app + read-only GitHub repo

Result  
2 critical · 2 high · 1 medium

Re-test  
5 / 5 fixed and verified

## Summary

PlantPal is a subscription plant-care tracker. Any visitor could read every user's profile and garden data, and the Supabase service key shipped in the public JavaScript, giving full database control to anyone who opened DevTools. The Stripe webhook accepted forged events, so anyone could grant themselves a Pro plan for free. All five issues were fixed with the prompts below and re-tested with the same exploit requests.

| # | Finding                                          | Severity | Re-test         |
|---|--------------------------------------------------|----------|-----------------|
| 1 | Row Level Security off on profiles and gardens   | Critical | Fixed           |
| 2 | Supabase service_role key in the frontend bundle | Critical | Fixed + rotated |
| 3 | Stripe webhook doesn't verify signatures         | High     | Fixed           |
| 4 | Admin API has no role check                      | High     | Fixed           |
| 5 | Missing security headers                         | Medium   | Fixed           |

## Critical

# 1. Row Level Security is off on profiles and gardens

## EVIDENCE (REDACTED)

```
$ curl 'https://xxxxxxx.supabase.co/rest/v1/profiles?
select=email,full_name,stripe_customer_id&limit=3' \
-H 'apikey: eyJhbGciOiI...[anon key, redacted]'
```

HTTP/2 200

```
[{"email":"j***@gmail.com","full_name":"J***
D***","stripe_customer_id":"cus_****"}, ...]
```

# 1,284 rows returned without logging in. gardens: 3,910 rows, incl. home addresses.

## WHY IT MATTERS

The anon key is public by design; it's in every visitor's browser. With RLS off, it can read (and here, also update) every row. That's a reportable data breach of emails, names and addresses, and anyone can edit other users' data.

## FIX PROMPT (PASTE INTO CURSOR)

```
Enable Row Level Security on the Supabase tables profiles and gardens. Write a
new SQL migration that:
1) runs ALTER TABLE ... ENABLE ROW LEVEL SECURITY on both tables;
2) adds policies so a signed-in user can SELECT/UPDATE only rows where
profiles.id = auth.uid() and gardens.user_id = auth.uid(), and INSERT gardens
only with user_id = auth.uid();
3) adds no policy for the anon role.
Don't change app code unless a query breaks; if one does, make it filter by the
current user's id. Show me the migration before applying it.
```

## RE-TEST

1. Re-run the exact curl above with only the anon key → expect [] or 401.
2. Sign in as test user A, request user B's garden by id → expect [].
3. Confirm the app still loads your own garden. **Result: 0 rows for anon, 0 rows cross-user. Fixed.**

Critical

## 2. Supabase service\_role key shipped in the frontend bundle

### EVIDENCE (REDACTED)

```
$ curl -s https://plantpal.example/assets/index-8f3a1c.js | grep -o 'eyJ[^\"]*' |  
jwt-decode  
{"iss":"supabase","ref":"xxxxxxx","role":"service_role", ...}  
# Source: src/lib/admin.ts uses import.meta.env.VITE_SUPABASE_SERVICE_KEY  
# Any VITE_ variable is compiled into public JS. Also present in git history  
(commit 3f9c**).
```

### WHY IT MATTERS

The service\_role key bypasses all RLS. Even after fixing #1, anyone with this key can read, change or delete the entire database and every user account. Removing it from the code isn't enough; it's already public and must be rotated.

### FIX PROMPT (PASTE INTO CURSOR)

The Supabase service\_role key is exposed via VITE\_SUPABASE\_SERVICE\_KEY. Fix it:

- 1) Remove every use of the service key from client code (src/). Any admin-only operation must move to a Supabase Edge Function (or server route) that reads SUPABASE\_SERVICE\_ROLE\_KEY from server env, never a VITE\_ var.
- 2) Delete VITE\_SUPABASE\_SERVICE\_KEY from .env files and .env.example, and add .env\* to .gitignore.
- 3) Make the client use only the anon key.
- 4) List the exact steps for me to roll the service key in Supabase (Settings → API) and set the new one as an Edge Function secret.

### RE-TEST

1. Rebuild and grep every deployed JS file for service\_role and eyJ → expect no service-role JWT.
2. Call the REST API with the *old* key → expect 401 Invalid API key (proves rotation).
3. **Result: key absent from bundle, old key rejected. Fixed.**

High

### 3. Stripe webhook doesn't verify signatures

#### EVIDENCE (REDACTED)

```
$ curl -X POST https://plantpal.example/api/stripe-webhook -H 'Content-Type: application/json' \
  -d '{"type":"checkout.session.completed","data":{"object":{"customer_email":"attacker@example.com","metadata":{"plan":"pro"}}}}'
HTTP/2 200 {"received":true}
# Test account attacker@example.com was upgraded to Pro with no payment.
# Handler: supabase/functions/stripe-webhook/index.ts does JSON.parse(body); no constructEvent().
```

#### WHY IT MATTERS

Anyone who finds the webhook URL can forge "payment succeeded" events and unlock paid plans for free, or downgrade real customers. This is direct revenue loss.

#### FIX PROMPT (PASTE INTO CURSOR)

```
Secure the Stripe webhook in supabase/functions/stripe-webhook/index.ts:
1) Read the raw request body as text (not JSON) and verify it with
   stripe.webhooks.constructEventAsync(body, req.headers.get('stripe-signature'),
   STRIPE_WEBHOOK_SECRET) using Stripe's SubtleCryptoProvider for Deno.
2) Return 400 if the signature is missing or invalid.
3) Don't trust metadata.plan from the event; retrieve the Checkout Session from
   Stripe and map its price ID to the plan.
4) Make processing idempotent by storing event.id and skipping duplicates.
```

#### RE-TEST

1. Re-send the forged request above → expect 400 and no plan change.
2. Run `stripe trigger checkout.session.completed` via Stripe CLI → expect 200 and the correct plan.
3. **Result: forged event rejected, real event processed. Fixed.**

High

## 4. Admin API has no role check

### EVIDENCE (REDACTED)

```
$ curl https://plantpal.example/api/admin/users -H 'Authorization: Bearer eyJ...  
[ordinary free-user session]'  
HTTP/2 200 [{"id":"b1e2****","email":"m***@outlook.com","plan":"pro"}, ...]  
$ curl -X DELETE https://plantpal.example/api/admin/users/b1e2**** -H  
'Authorization: Bearer eyJ...'  
HTTP/2 204 # (run against our own test account only)  
# The route checks that a user is logged in, not that they're an admin. /admin  
page is only hidden in the UI.
```

### WHY IT MATTERS

Hiding the admin page in the UI isn't access control. Any signed-up user can list all customers and delete accounts.

### FIX PROMPT (PASTE INTO CURSOR)

```
Add server-side authorization to every route under /api/admin:  
1) Create a requireAdmin middleware that verifies the Supabase JWT server-side  
(supabase.auth.getUser(token)), then checks the user's role in a profiles.role  
column (or app_metadata.role) equals 'admin'. Return 403 otherwise.  
2) Apply it to all /api/admin/* handlers, including DELETE.  
3) Make sure users can't edit their own role: add an RLS policy or trigger that  
blocks updates to profiles.role except by the service role.  
4) Add a test that a non-admin token gets 403 on GET /api/admin/users.
```

### RE-TEST

1. Repeat both requests with a normal user token → expect 403.
2. Try PATCH profiles set role='admin' as that user → expect it to be blocked.
3. Repeat with an admin token → 200. **Result: 403 for non-admins. Fixed.**

Medium

## 5. Missing security headers

### EVIDENCE

```
$ curl -sI https://plantpal.example | grep -iE 'strict-transport|content-security|x-frame|x-content-type|referrer'
(no output: none of the 5 headers are set)
```

### WHY IT MATTERS

Without these, the app can be framed for clickjacking, any injected script runs with full power (no CSP), and browsers may downgrade to HTTP. Cheap to fix, and it limits the blast radius of future bugs.

### FIX PROMPT (PASTE INTO CURSOR)

```
Add security headers for all routes in our host config (vercel.json headers, or public/_headers on Netlify/Cloudflare):
Strict-Transport-Security: max-age=31536000; includeSubDomains
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Referrer-Policy: strict-origin-when-cross-origin
Content-Security-Policy: default-src 'self'; script-src 'self'
https://js.stripe.com; connect-src 'self' https://*.supabase.co
https://api.stripe.com; frame-src https://js.stripe.com
https://checkout.stripe.com; img-src 'self' data: https:; style-src 'self'
'unsafe-inline'
Start CSP in Content-Security-Policy-Report-Only if unsure, and tell me which
existing scripts would be blocked.
```

### RE-TEST

1. Re-run the curl above → all 5 headers present.
2. Load the app and complete a test Stripe checkout → no CSP errors in the console.

**Result: headers present, checkout works. Fixed.**

## What wasn't found

No exposed Stripe secret key, no open storage buckets, no SQL injection in the search endpoint, and password reset links expire correctly. We report what we checked, not just what broke.

**Want this for your app?**

Start with a free scan, or get the full report with exploit proof, fix prompts and a verified re-test, delivered the same day.

This is a sample. PlantPal and all data shown are fictional. Exploit requests in real reports are run only against apps the customer confirms they own, using test accounts.