



Security assessment / Confidential

Cadence application security assessment

Client	Northwind Labs GmbH
Product	Cadence (https://cadence-app.example)
Engagement	Focused Assessment
Testing window	2026-02-10 to 2026-02-13, Europe/Zurich
Report reference	GIN-2026-007
Report status	Final with retest
Author	Simonas Zabulionis, Gintar
Classification	Confidential

Executive summary

Cadence is a scheduling and booking product built quickly on Next.js, Supabase, and Stripe, hosted on Vercel. Over four days I reviewed the public web app, its API, and the staff dashboard against the scope agreed in the Rules of Engagement.

The headline issue was serious. The Supabase database was reachable with the public anonymous key and row-level security was switched off on the customer tables, so anyone with the key embedded in the frontend could read every customer's profile and booking. This has since been fixed and I verified the fix on retest. Beyond that, one account-boundary flaw in the booking API remains open and should be the next priority, along with a stored scripting issue in booking notes that affects your own staff.

This was a point-in-time, sampling-based review. It is not a guarantee that Cadence is free of vulnerabilities. The findings below are ordered by severity, with plain-English impact, reproduction steps, and specific fixes.

Severity counts: 1 Critical, 1 High, 2 Medium, 1 Low, 1 Informational

ID	Finding	Severity	Status
F-01	Row-level security disabled on customer tables	Critical	Fixed
F-02	Booking API returns other customers' bookings	High	Open
F-03	Stored script injection in booking notes	Medium	Open
F-04	No rate limiting on authentication endpoints	Medium	Mitigated
F-05	Missing HTTP security headers	Low	Risk accepted
F-06	Framework and version disclosure	Informational	Open

F-01

Critical

Row-level security disabled on customer tables

Status

Fixed

Asset

Supabase project cadence-prod, tables profiles and bookings (production)

CVSS 3.1

9.1 (Critical), AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:L/A:N

Category

Broken access control (CWE-284)

Impact

The Supabase anonymous key is shipped in the browser bundle, which is normal. What is not normal is that row-level security was disabled on profiles and bookings, so that key could read every row in both tables. Anyone who opened the site could pull the full customer list and every booking, including names, emails, and appointment details. For a booking product this is a full customer-data breach with no authentication required.

Preconditions

None. The anonymous key is readable by any visitor in the frontend JavaScript.

Steps to reproduce

1. Read the anonymous key SUPABASE_ANON_KEY from the page bundle at `/_next/static/...`
2. Query the REST endpoint directly:

```
GET /rest/v1/profiles?select=* HTTP/1.1
Host: cadence-prod.supabase.co
apikey: <anon-key>
Authorization: Bearer <anon-key>
```

3. The response returns all rows rather than an empty set or an authorization error.

Evidence

Synthetic excerpt, values redacted:

```
[
  { "id": "a1b2...", "email": "r...@example.com", "full_name": "R- K-", "plan":
    "pro" },
  { "id": "c3d4...", "email": "j...@example.com", "full_name": "J- M-", "plan":
    "free" }
]
```

The query returned 4,182 rows during testing. No data was retained beyond the two redacted rows above, in line with the evidence limit in the Rules of Engagement.

Remediation

1. Enable row-level security on every table exposed through PostgREST: `ALTER TABLE profiles ENABLE ROW LEVEL SECURITY;` and the same for bookings.
2. Add explicit policies so a user can only read their own rows, for example a policy `USING (auth.uid() = user_id)`.
3. Confirm no table is left with RLS off. See the Supabase guide on [Row Level Security](#).

Verification (retest)

Fixed. On 2026-02-13 I confirmed RLS is enabled on both tables and that the anonymous key now returns an empty set for `profiles` and only the authenticated user's own rows for `bookings`. Status changed from Open to Fixed.

F-02 High

Booking API returns other customers' bookings

Status	Open
Asset	GET /api/bookings/{id} (production)
CVSS 3.1	7.7 (High), AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N
Category	Insecure direct object reference (CWE-639)

Impact

Any logged-in user can read any other user's booking by changing the numeric ID in the URL. Booking records include the customer name, email, notes, and appointment time. One ordinary account is enough to enumerate the whole booking table.

Preconditions

A single valid account. Booking IDs are sequential integers, so they are trivial to guess.

Steps to reproduce

1. Sign in as account A and open one of your own bookings, noting the ID, for example 1042.
2. Request a different ID that does not belong to you:

```
GET /api/bookings/1041 HTTP/1.1
Host: cadence-app.example
Cookie: sb-access-token=<session-for-account-A>
```

3. The response returns account B's booking instead of 403 Forbidden.

Evidence

The endpoint reads the row by id alone and never checks that the booking belongs to the caller. Decrementing the ID walked backwards through other customers' bookings.

Remediation

Enforce ownership on the server for every booking lookup. In the route handler, filter by the authenticated user, not only by ID:

- Query with both keys: `where id = $1 and user_id = auth.uid()`.
- If you rely on Supabase for this table, the RLS policy from F-01 also closes this path, but the API must use the user's token, not the service role key, for the policy to apply.
- Return 404 rather than 403 so you do not confirm that an ID exists.

Verification (retest)

Pending retest.

F-03

Medium

Stored script injection in booking notes

Status	<div>Open</div>
Asset	notes field on POST /api/bookings, rendered in the staff dashboard
Category	Stored cross-site scripting (CWE-79)

Impact

The customer-supplied booking note is rendered in the staff dashboard without escaping. A customer can store markup that runs script in a staff member's browser when they open the booking. That runs in the context of your own admin interface, so it can act as the staff user. The blast radius is limited to staff who view the booking, which is why this is Medium rather than High, but it targets your most privileged users.

Preconditions

Ability to create a booking, which any customer has. A staff member must then open the booking in the dashboard.

Steps to reproduce

1. Create a booking with the note set to a scripting payload, for example `<img src=x onerror="alert(document.domain)">`.
2. Open the staff dashboard as a staff account and view that booking.
3. The payload executes rather than showing as text.

Evidence

The dashboard renders the note with a raw HTML sink rather than as text. Payload stored and executed in a controlled staff session during testing. No real staff account was targeted.

Remediation

- Render the note as text, not HTML. In React, output `{note}` directly rather than using `dangerouslySetInnerHTML`.
- If any formatting is required, sanitise with a maintained library such as DOMPurify before rendering.
- Add a Content-Security-Policy (see F-05) as defence in depth.

Verification (retest)

Pending retest.

F-04

Medium

No rate limiting on authentication endpoints

Status	Mitigated
Asset	POST /api/auth/login, POST /api/auth/reset (production)
Category	Improper restriction of excessive authentication attempts (CWE-307)

Impact

Neither the login nor the password-reset endpoint limited how many requests a client could make. That allows password guessing against known emails and lets an attacker send a large volume of reset emails to any address, which is both an account-security risk and a way to abuse your email sender's reputation.

Preconditions

None for the reset abuse. Password guessing also needs a target email, which is easy to obtain given F-01.

Steps to reproduce

1. Send 500 POST /api/auth/login requests for one email with different passwords.
2. Observe that all are processed with no throttling, lockout, or challenge.

Remediation

- Add per-IP and per-account rate limiting on both endpoints.
- Add a short lockout or an exponential backoff after repeated failures.
- Consider a CAPTCHA or proof-of-work only after a threshold, to avoid hurting normal users.

Verification (retest)

Mitigated. On retest, Vercel platform rate limiting had been enabled in front of both routes, which blunts high-volume abuse. Application-level per-account backoff is still recommended, so the status is Mitigated rather than Fixed.

F-05

Low

Missing HTTP security headers

Status

Risk accepted

Asset

All responses from https://cadence-app.example

Category

Security misconfiguration (CWE-693)

Impact

Responses omit several standard hardening headers. On their own these do not create a direct compromise, but they remove defence-in-depth that would reduce the impact of other issues such as F-03.

Steps to reproduce

- 1. Request any page and inspect the response headers.
- 2. Note the absence of the headers listed below.

Remediation

Add the following, tuned to the app:

Header	Suggested value
Content-Security-Policy	Start in report-only mode, then enforce a tight policy.
Strict-Transport-Security	max-age=63072000; includeSubDomains; preload
X-Content-Type-Options	nosniff
Referrer-Policy	strict-origin-when-cross-origin

These can be set once in next.config.js under headers().

Verification (retest)

Risk accepted. The team chose to defer this to the next release. Recorded as accepted by the client on 2026-02-13.

F-06

Informational

Framework and version disclosure

Status	Open
Asset	Response headers and default error pages
Category	Information exposure (CWE-200)

Impact

Responses advertise the framework and a specific version, and unhandled errors return a default stack trace in some cases. This does not create a direct vulnerability, but it hands an attacker a head start on which known issues to try.

Steps to reproduce

1. Inspect the `x-powered-by` response header.
2. Trigger a handled-poorly route and observe the verbose error output.

Remediation

- Set `poweredByHeader: false` in `next.config.js`.
- Ensure production builds do not return stack traces to the client.

Verification (retest)

Pending retest.

Appendix A: Scope

In scope, per the signed Rules of Engagement dated 2026-02-09:

Asset	Identifier	Environment
Web app	https://cadence-app.example	Production
API	https://cadence-app.example/api/*	Production
Staff dashboard	https://cadence-app.example/staff	Production
Database	Supabase project cadence-prod	Production

Out of scope: the Stripe integration and any payment flow, the marketing site, all third-party infrastructure, and any account other than the two controlled test accounts supplied by the client.

Appendix B: Methodology

Manual testing against the OWASP Web Security Testing Guide, focused on access control, authentication, injection, and configuration, supported by limited rate-controlled tooling within the agreed limits. Findings were validated by hand. Risk ratings use the definitions in Appendix C and are prioritisation guidance, not a prediction of loss.

Appendix C: Severity definitions

Severity	Meaning	Suggested response
Critical	Account takeover, mass data exposure, or full compromise is likely.	Fix immediately.
High	Serious impact is plausible, such as bypassing access controls or exposing protected data.	Fix urgently.
Medium	A real weakness that usually needs conditions, chaining, or user interaction.	Plan a fix soon.
Low	Limited impact or a defence-in-depth gap.	Fix when practical.
Informational	Hardening or improvement, not a direct vulnerability.	Optional.

Appendix D: Limitations

Testing ran across four days against production, within the agreed rate limits and testing hours. Payment flows and third-party infrastructure were not tested. A clean result for an area in scope is not proof that the area is free of all issues. This report reflects the state of the application during the testing window only.