Lunaro (lunaroco.ai) — Security & Site Audit

Prepared by: Matt Gambone, MackTechs LLC **Date:** April 24, 2026 **Site:** https://lunaroco.ai **Scope:** Public-facing landing page + signup API + admin panel

Executive Summary

Lunaro is a pre-launch landing page for a cycle-synced fitness app, built with vanilla HTML/JS and hosted on Vercel. The site collects personal information (name, email, phone) via a signup API and displays it in a hidden admin dashboard.

This audit found 19 issues across 4 severity levels. The most critical findings involve an exposed admin panel that displays all signup data in plain text, zero input validation on the signup API, no payload size limits, unlimited duplicate signups, and a complete lack of legal compliance (no privacy policy while collecting personal and eventually health data).

Severity	Count
Critical	6
High	5
Medium	5
Low	3

Tech Stack

Component	Detail
Frontend	Vanilla HTML/CSS/JS (no framework)
Hosting	Vercel (free tier possible)
Backend	Vercel serverless function (`/api/signup` , `/api/admin`)
Domain registrar	Squarespace Domains

Component	Detail
DNS	Google Cloud DNS
Email	Google Workspace (MX records)
Database	Unknown (likely Vercel KV, Postgres, or flat file)
Analytics	None
CDN	Vercel Edge (built-in)

Critical Findings

C-1: Admin Dashboard Exposes All Signup Data in Plain Text

Location: `/admin` and `/api/admin`

The site has a hidden admin panel at `/admin` that renders a live table of every signup — **name, email, phone number, and timestamp** — in plain HTML.

- The full admin page HTML (structure, JS, styling) is served to **anyone** who visits `/admin`, even without authentication — it returns HTTP 200
- The data itself is gated behind a `?key=` query parameter on `/api/admin`
- There is no rate limiting on key attempts — the key can be brute-forced
- The admin key is passed in the URL, meaning it leaks via browser history, bookmarks, server logs, referrer headers, and screenshots

Additionally, the admin page source code contains the developer's name:

```
"Use the full link Colton gave you."
```

This exposes internal team information to the public.

Risk: If the key is weak or leaked, all collected personal data (names, emails, phone numbers) is immediately exposed with no audit trail.

Recommendation: - Move the admin panel behind real authentication (session-based login with password or OAuth) - Do not serve the admin HTML to unauthenticated visitors — return 401 for the page itself - Add rate limiting and lockout on failed key attempts - Remove developer name from client-side code

C-2: Zero Input Validation on Signup API

Location: `POST /api/signup`

The signup endpoint accepts any string for all three fields. The following all return `200 OK` with a valid founder code:

Test	Input	Result
XSS payload	`name: <script>alert(1)</script>`	Accepted
Invalid email	`email: notanemail`	Accepted
Invalid phone	`phone: abc`	Accepted
SQL injection	`email: test@test.com OR 1=1--`	Accepted
Mass assignment	`admin: true, role: admin` (extra fields)	Accepted

The only validation: all three fields must be present (empty `{}` returns 400).

Risk: Stored XSS if data is rendered in the admin panel or emails. SQL injection if a relational database is used without parameterized queries. Garbage data pollution. Potential mass assignment depending on ORM/database design.

Recommendation: - Validate email format with regex or a library - Validate phone format (digits only, length check) - Sanitize name field (strip HTML tags at minimum) - Reject unexpected fields (allowlist `name`, `email`, `phone` only)

C-3: No Rate Limiting Anywhere

Location: `POST /api/signup`, `GET /api/admin`

10 rapid-fire requests to `/api/signup` all succeeded with HTTP 200. The `/api/admin` endpoint also has no rate limiting on key attempts.

Risk: - Signup endpoint can be flooded by bots (database pollution, resource exhaustion) - If the backend sends confirmation emails or SMS, this becomes an email/SMS bombing vector - Admin key can be brute-forced without lockout

Recommendation: - Add rate limiting via Vercel middleware, Upstash, or Cloudflare - Suggested limits: 5 signups per IP per hour, 10 admin key attempts per IP per hour with exponential backoff - Consider adding CAPTCHA (reCAPTCHA, Turnstile) to the signup form

---

C-4: No Privacy Policy or Terms of Service

Location: `/privacy` (404), `/terms` (404)

The site collects personally identifiable information (name, email, phone) with no data handling disclosures. The product will eventually collect menstrual cycle data, which is classified as **sensitive health data** under GDPR, CCPA, and various state health privacy laws.

Risk: Legal liability. App store rejection if/when a mobile app is submitted. Loss of user trust. Potential regulatory action depending on user geography.

Recommendation: - Publish a privacy policy covering: what data is collected, how it's stored, who has access, retention period, user rights (deletion, export), third-party sharing - Publish terms of service covering: refund policy, liability, account termination, data ownership - If health data will be collected, consult a lawyer on HIPAA (if US-based), GDPR Article 9 (if serving EU users), or state-level health privacy laws - Link both documents from the signup form (ideally with a checkbox acknowledgment)

C-5: No Payload Size Limits — Database Stuffing Possible

Location: `POST /api/signup`

The API accepts arbitrarily large payloads with no field-level size limits. Testing confirmed:

Payload Size	Result
10 KB name field	200 OK — accepted
100 KB name field	200 OK — accepted
1 MB name field	200 OK — accepted
3 MB name field	200 OK — accepted
5 MB name field	413 — Vercel's default limit finally kicks in

This means a single signup request can write up to ~4 MB of junk data into the database. Combined with no rate limiting, an attacker could fill the database with gigabytes of garbage data in minutes,

potentially exhausting storage and incurring costs.

Risk: Storage exhaustion, database performance degradation, potential cost explosion if using a metered database (e.g. Vercel Postgres, PlanetScale).

Recommendation: - Enforce field-level max lengths: name (100 chars), email (254 chars), phone (15 chars) - Set an explicit body size limit in the API handler (e.g. 1 KB max for this endpoint)

C-6: No Duplicate Signup Prevention

Location: `POST /api/signup`

The same exact name, email, and phone combination can be submitted unlimited times — each one is accepted with `200 OK`. There is no deduplication by email, no “already signed up” response, and no unique constraint.

Risk: - Database pollution with duplicate records - Inflated signup counts (the “72% claimed” metric is meaningless if one person can sign up 10,000 times) - If confirmation emails are sent, a single email address can be spammed indefinitely

Recommendation: - Add a unique constraint on the email field - Return a friendly “you’re already signed up” message for duplicates - Consider email verification before counting the signup as valid

---

High Findings

H-1: No Security Headers

Location: All pages

The site is missing every standard security header:

Header	Status	Risk
`Content-Security-Policy`	Missing	XSS, code injection
`X-Frame-Options`	Missing	Clickjacking
`X-Content-Type-Options`	Missing	MIME sniffing
`Referrer-Policy`	Missing	Data leakage via referrer
`Permissions-Policy`	Missing	Unrestricted browser APIs

Recommendation: Add headers via `vercel.json`:

```
{
  "headers": [
    {
      "source": "/(.*)",
      "headers": [
        { "key": "X-Frame-Options", "value": "DENY" },
        { "key": "X-Content-Type-Options", "value": "nosniff" },
        { "key": "Referrer-Policy", "value": "strict-origin-when-cross-origin" },
        { "key": "Permissions-Policy", "value": "camera=(), microphone=(), geolocation=()" }
        { "key": "Content-Security-Policy", "value": "default-src 'self'; script-src 'self'
      ]
    }
  ]
}
```

H-2: CORS Wide Open

Location: All responses

The server returns `access-control-allow-origin: *`, meaning any website can make API requests to the signup and admin endpoints from client-side JavaScript.

Risk: A malicious third-party site could embed a hidden form that submits fake signups to `/api/signup`, or attempt to access `/api/admin` if the key is known.

Recommendation: Restrict CORS to the actual domain:

```
access-control-allow-origin: https://lunaroco.ai
```

H-3: Founder Code is Static and Identical for Everyone

Location: `POST /api/signup` response

Every signup — regardless of input — returns the same code: `FoundersLaunch42426`. This code is: - Not unique per user - Not tied to an email or account - Not actually expiring (the “24-hour expiry” appears to be client-side only)

Risk: The code can be freely shared, undermining the “non-transferable” and “founding pricing” claims. Anyone who sees the code once can distribute it. The urgency mechanics (countdown, expiry,

capacity meter) appear to be purely cosmetic.

Recommendation: - Generate unique codes per signup tied to the user's email - Validate the code against the email at redemption - Store expiry server-side and enforce it

---

H-4: No CSRF Protection

Location: `POST /api/signup`

No CSRF token, no `SameSite` cookie, no origin header validation. Combined with the open CORS policy, cross-site request forgery is trivial.

Risk: Any website can programmatically submit signups on behalf of unsuspecting visitors.

Recommendation: Validate the `Origin` or `Referer` header on POST requests. If session cookies are introduced later, set `SameSite=Strict`.

H-5: Malformed Input Causes Silent Failures

Location: `POST /api/signup`

Sending a non-JSON `Content-Type` (e.g. `text/plain`) or malformed JSON body returns a `400` error, but with the generic `"name, email, phone required"` message — the same error as missing fields. There is no distinction between "bad request format" and "missing fields," making debugging difficult and potentially masking attacks.

The API also does not enforce `Content-Type: application/json` — it tries to parse any body regardless of the declared content type.

Recommendation: Reject requests that aren't `application/json` with a clear error. Return distinct error messages for malformed JSON vs. missing fields.

Medium Findings

M-1: No Email Deliverability Records

Only one TXT record exists (Google site verification). Missing: - **SPF** — no sender policy, emails may be flagged as spam - **DKIM** — no domain key signing - **DMARC** — no policy for failed authentication

If founder codes are emailed, they may land in spam.

Recommendation: Add SPF, DKIM, and DMARC TXT records via Google Workspace admin.

M-2: No Analytics or Monitoring

Zero analytics, error tracking, or uptime monitoring detected. There's no way to know: - How many people visit vs. sign up (conversion rate) - If the API is throwing errors - If someone is abusing the endpoints

Recommendation: Add at minimum GA4 or a privacy-friendly alternative (Plausible, Fathom). Add error tracking (Sentry free tier). Monitor uptime.

M-3: No Open Graph / Social Meta Tags

No `og:title`, `og:description`, `og:image`, or `twitter:card` tags. Links shared on social media (including the Instagram bio link driving traffic) show no preview.

Recommendation: Add OG meta tags with a branded preview image.

M-4: No Favicon or Structured Data

No favicon (generic browser tab icon). No JSON-LD schema markup. Minor but contributes to an unfinished/untrustworthy appearance.

M-5: Admin Page XSS is Mitigated (Partial Credit)

The admin dashboard does use an `esc()` function to escape HTML entities before rendering signup data into the table. This means the stored XSS from the signup form (`<script>alert(1)</script>` in the name field) would not execute in the admin panel specifically.

However, this only protects the admin dashboard. If signup data is rendered **anywhere else** — confirmation emails, exports, future app interfaces — without the same escaping, the XSS payload would execute. The fix should be server-side sanitization at the point of input, not client-side escaping at the point of display.

Low Findings

L-1: No robots.txt or sitemap.xml

Both return 404. Not critical for a single-page site but indicates incomplete setup.

L-2: Minimal Founder Identity

Only “Sydney” — no last name, credentials, or verifiable identity. For a fitness/health product, users expect to see qualifications. This is a trust issue more than a security issue.

L-3: Urgency Mechanics Appear Cosmetic

The “72% claimed” capacity bar, 24-hour code expiry, and countdown timer all appear to be client-side only. The founder code is static (`FoundersLaunch42426`), identical for all users, and has no server-enforced expiry. If users discover this, it damages trust in the brand.

BWS Hosting Comparison

Would hosting Lunaro on Beacon Web Services (Matt’s managed hosting) improve or worsen the security posture?

What BWS fixes automatically

Issue	Vercel (Current)	BWS (Caddy + Docker)
Security headers	None — must manually configure `vercel.json`	Caddy adds `X-Frame-Options` , `X-Content-Type-Options` , `Strict-Transport-Security` by default
SSL/TLS	Automatic (Vercel)	Automatic (Caddy ACME)
CORS	Wide open `*`	Not set by default (secure) — must explicitly open
Rate limiting	None built-in	Can add via Caddy `rate_limit` directive
DDoS protection	Vercel Edge has basic protection	Would need Cloudflare in front (BWS already uses CF)

Issue	Vercel (Current)	BWS (Caddy + Docker)
Admin panel exposure	Served publicly	Can restrict via Caddy IP allowlist or Cloudflare Access

What BWS does NOT fix

These are application-level issues that exist regardless of hosting: - Input validation (code problem, not infra) - Static founder code (code problem) - CSRF protection (code problem) - Privacy policy / legal (content problem) - Admin authentication model (code problem) - Email deliverability (DNS problem)

Verdict

BWS would be a moderate improvement for infrastructure-level security — Caddy's defaults are more secure than bare Vercel, and Cloudflare in front adds DDoS protection and the option for Access (email OTP gating for the admin panel, like you already use for server.macktechs.com). But the majority of the issues are in the application code itself, which would need to be fixed regardless of where it's hosted.

The better play: If Sydney/Colton are open to it, keep the landing page on Vercel (it's fast, free, and fine for static content) but fix the application-layer issues. If they want managed hosting with security handled for them, BWS + Cloudflare Access would be a step up — and that's a service you could offer.

Recommended Fix Priority

Priority	Issue	Effort
1	Secure admin panel (real auth, don't serve HTML to public, remove "Colton")	Medium
2	Add input validation + field length limits to signup API	Low
3	Add duplicate signup prevention (unique constraint on email)	Low

Priority	Issue	Effort
4	Add rate limiting (signups + admin key attempts)	Low-Medium
5	Add privacy policy + terms of service	Low (template + customize)
6	Add security headers via `vercel.json`	Low
7	Restrict CORS to `lunaroco.ai` only	Low
8	Enforce request body size limit (1 KB for signup)	Low
9	Generate unique per-user founder codes with server-side expiry	Medium
10	Add SPF/DKIM/DMARC records	Low
11	Add analytics + error tracking	Low
12	Add OG meta tags + favicon	Low
13	Enforce `Content-Type: application/json`	Low

This audit was performed using publicly accessible information only. No authentication was bypassed, no data was accessed, and no destructive testing was performed. All API tests used dummy/test data.

Prepared by MackTechs LLC — macktechs.com