the BLOCKCHAIN team

APPLICATION SECURITY ASSESSMENT

Waldrop

Web application, API surface, zkLogin & Seal integration, transform engine

0	1	4	5	4
CRITICAL	HIGH	MEDIUM	LOW	INFO

Audited by **The Blockchain Team** · https://theblockchain.team

TARGETS	`waldrop-app` (Next.js / Vercel) · `waldrop-transform-engine` (Rust / AWS EC2)
OBJECTIVES	(A) auth & API · (B) zkLogin · (C) Seal · (D) rate limiting & DDoS · (E) XSS/CSRF
METHOD	White-box source review - static only, no dynamic verification
SURFACE	28 API routes, 14 gRPC RPCs, 2,565 LoC of route handlers
REPORT DATE	14 August 2026
STATUS	Final - static source assessment

Contents

1 Executive Summary

This assessment reviewed the Waldrop web application and the Rust transform engine behind it, against five objectives: platform authentication and API endpoints, the zkLogin workflow, Seal encryption and key handling, rate limiting and DDoS posture, and XSS/CSRF exposure.

The security posture is strong. This is an unusually well-defended codebase, and the areas that most often fail were the areas that held up best. Authentication replay protection is textbook-correct. zkLogin is delegated to Sui's canonical verifier rather than reimplemented, with the claimed address bound and on-chain JWKs enforced. There are zero XSS sinks in the application. SSRF defence includes IP pinning and per-hop redirect revalidation - the two controls almost every implementation omits. OAuth CSRF, cookie flags, open-redirect handling and transport security are all correct.

No critical issues were found, and no path was identified by which an attacker authenticates as another user, reads another user's data, forges a session, or executes script in a victim's browser.

The findings cluster in a single theme: **controls that exist and are correct are not applied uniformly, and one route that handles a secret has no gate at all.**

- **The Seal proxy is an unauthenticated confused deputy** (H-01). `/api/seal/[...path]` attaches the server's `ENOKI_SEAL_API_KEY` to any request that reaches it, with no authentication, no rate limit and no path allowlist. Anyone on the internet can use Waldrop's paid Enoki credential as a free authenticated proxy to Mysten, and bill Waldrop for the Vercel invocations while doing it. This is the only place a real secret is exposed to unauthenticated callers, and it is the one finding that should be fixed before anything else.

- **The authentication path has no rate limit** (M-01). `/api/auth/challenge` and `/api/auth/session` are unmetered, and the engine's own `GetChallenge` and `CreateSession` RPCs are reachable directly at `:443`, bypassing Vercel entirely.
- **A declared rate limit is never enforced** (M-02). `RateLimit { requests_per_minute, requests_per_hour }` is constructed, cloned, and returned to clients in the session response - but no code ever compares a request count against it.
- **The front-door limiter is per-instance on Vercel** (M-03), so concurrent lambdas multiply the effective limit. The code states this plainly rather than hiding it.

> **PRIORITY**
>
> Fix H-01 first and independently - it is a live credential exposed to the internet, it is a two-line fix, and it does not depend on any other decision. The rate-limiting findings (M-01 to M-03) are one piece of work and should be resolved together, most cheaply by configuring Vercel Firewall rules plus a limiter at Caddy for the engine's unauthenticated RPCs.

> **SCOPE OF THIS REPORT**
>
> This is a **static assessment**. Every finding was derived by reading source; none has been confirmed by issuing a request against a running instance. Findings are stated with the evidence that supports them, and the two that depend on runtime configuration (M-01, M-03) say so explicitly. A dynamic phase would confirm exploitability, establish whether Vercel Firewall rules are configured, and cover the one class of issue static review cannot reach - horizontal access control between two real accounts.

2 Scope, Method & Limitations

2.1 In scope

COMPONENT	STACK	HOSTING	REVIEWED
`waldrop-app`	Next.js App Router, TypeScript	Vercel	28 API routes, auth library, rate limiter, redirect and cookie handling, security headers, client bundle exposure
`waldrop-transform-engine`	Rust, tonic gRPC	AWS EC2 + Caddy	Auth module, signature verification, zkLogin verifier, per-RPC permission gates, SSRF validation, deployment configuration

2.2 Out of scope

Mysten's zkLogin circuit, Seal key servers and Walrus storage nodes; Google and Dropbox OAuth infrastructure; the Move contracts (covered separately in the Waldrop Protocol audit); the SDK, CLI and landing site; and the Vercel and AWS account configurations themselves, which are not visible from source.

2.3 Method

Manual white-box review. Rather than scanning for patterns, the review traced each trust boundary and asked what each side is entitled to assume:

- **Guard-set differencing.** Every route and RPC was enumerated and its guards recorded - authentication, identity binding, rate limit, input validation. Sibling handlers that should carry identical guards but do not were treated as findings. H-01 and M-01 were both found this way.
- **Secret-flow tracing.** Every server-side secret was followed to each point it is attached to an outbound request, and the gate protecting that point was identified. This is what surfaced H-01.
- **Replay and state-machine analysis** of the challenge/signature/session flow, including the Redis-unavailable fallback path.
- **Sink enumeration** for XSS, open redirect and SSRF, followed by review of each guard found - including the cases guards usually miss (protocol-relative URLs, DNS rebinding, redirect chains).
- **Declared-versus-enforced comparison.** Configuration surfaced to clients was checked for a corresponding enforcement site. M-02 is the result.

2.4 Limitations

Stated plainly, because they bound what the findings can claim:

- **No dynamic testing was performed.** No request was issued to any Waldrop system. All findings are static inferences from source.

- **Runtime configuration is invisible.** Whether Vercel Firewall rate-limiting rules are configured, and on which plan tier, cannot be determined from the repository. M-01 and M-03 are rated on the assumption that no such rules exist; if they do, both drop.
- **Horizontal access control is untested.** Confirming that user A cannot reach user B's data requires two live accounts. This is the single largest residual gap.
- **Infrastructure was not assessed.** Security groups, IAM policies, Redis network exposure and secret storage were read from committed deployment scripts only.
- **Dependencies were not audited.** No SCA or supply-chain review was performed.

3 Severity Classification

SEVERITY	DEFINITION
CRITICAL	Unauthenticated compromise of user data, accounts, or funds; remote code execution; full authentication bypass.
HIGH	Exposure of a live secret, or unauthenticated access to a privileged capability. Exploitable today by anyone, with no precondition.
MEDIUM	A missing control that enables abuse, cost amplification, or degradation, but not direct data compromise; or a control that is declared but not enforced.
LOW	Defence-in-depth gaps, hardening opportunities, and accepted risks worth restating.
INFO	Observations, architectural notes, and items requiring confirmation from runtime configuration.

4 Findings Summary

ID	SEVERITY	TITLE	COMPONENT	OBJ.
H-01	HIGH	Seal proxy is an unauthenticated confused deputy for the Enoki API key	app	A, C
M-01	MEDIUM	Authentication path has no rate limit, and the engine is reachable independently of Vercel	app, engine	D
M-02	MEDIUM	Per-identity request rate limit is declared and advertised but never enforced	engine	D
M-03	MEDIUM	Front-door limiter is per-instance; Vercel's scaling multiplies the effective limit	app	D
M-04	MEDIUM	`requireSession` establishes credential validity, never caller identity	app	A
L-01	LOW	Session-verification probe depends on an unrelated RPC keeping its permission gate	app	A
L-02	LOW	No `script-src` Content-Security-Policy	app	E
L-03	LOW	Drive OAuth access token is handed to client-side JavaScript	app	A
L-04	LOW	Long-lived cloud credentials transit the request body; logging unverified	app, engine	A
L-05	LOW	Unimplemented `TransformStream` RPC remains in the public service definition	engine	A
I-01	INFO	Vercel Firewall configuration must be confirmed to close out M-01 and M-03	infra	D
I-02	INFO	Engine is a second internet-facing entry point with a different control set	infra	A, D
I-03	INFO	Horizontal access control between accounts remains untested	–	A
I-04	INFO	Redis availability is a soft dependency of the authentication path	engine	A, D

5 Detailed Findings

H-01 HIGH Seal proxy is an unauthenticated confused deputy

Location: `src/app/api/seal/[...path]/route.ts`
Objectives: (A) API endpoints, (C) Seal key management · **Status:** Static - not dynamically confirmed

The route proxies to Mysten's Seal aggregator, injecting a server-held API key:

```
const SEAL_UPSTREAM = "https://seal-aggregator-mainnet.mystenlabs.com";
const ENOKI_SEAL_API_KEY = process.env.ENOKI_SEAL_API_KEY ?? "";

async function proxy(req, ctx) {
  // ... header stripping ...
  headers.set("x-api-key", ENOKI_SEAL_API_KEY);   // server credential attached
  const upstream = await fetch(url, init);
  return new Response(upstream.body, { /* ... */ });
}

export const GET = proxy;
export const POST = proxy;   // no auth check anywhere in this file
```

There is no session check, no rate limit, and no allowlist constraining `path`. Any unauthenticated caller can issue a request to `/api/seal/<anything>` and have Waldrop's paid Enoki credential attached to it upstream. The consequences are:

- **Free use of a paid credential.** The endpoint is a public, authenticated gateway to Mysten's aggregator on Waldrop's account. Quota consumption and any associated billing accrue to Waldrop.
- **Quota exhaustion as denial of service.** If the Enoki key is rate-limited or quota-capped upstream, an attacker can exhaust it and break decryption for every legitimate user - the product's core function.
- **Vercel invocation cost.** Each request is a billed function invocation plus egress, so the endpoint amplifies attacker cost into platform cost.

Two things this is *not*, both checked:

- **Not an SSRF.** `SEAL_UPSTREAM` is a constant and path segments are appended after it, so a caller cannot redirect the request to another host.
- **Not an unbounded memory DoS.** The handler buffers the full body via `req.arrayBuffer()`, but Vercel caps serverless request bodies at 4.5 MB, which bounds it. Worth noting if the app ever moves off Vercel or onto an runtime without that cap.

> **CREDIT WHERE DUE**
>
> The header handling in this file is exemplary. It explicitly strips `cookie` on the outbound leg, with a comment noting that a same-origin proxy would otherwise forward users' `gdrive_access_token` and `dropbox_access_token` to Mysten on every decrypt. That is a subtle leak most implementations ship. It also strips caller-supplied `x-api-key` and `authorization`, and handles `content-encoding` correctly on the response. The problem is narrowly that nothing gates *who* may invoke it.

RECOMMENDATION

Apply the session check the connector routes already use, and add a rate limit:

```
import { requireSession, unauthorizedResponse } from "@/lib/api-auth";
import { checkRateLimit, clientKey, rateLimitResponse } from "@/lib/rate-limit";

async function proxy(req, ctx) {
  const rl = checkRateLimit(`seal:${clientKey(req)}`, 60, 60);
  if (!rl.ok) return rateLimitResponse(rl);
  try { await requireSession(await readAuthFrom(req)); }
  catch (e) { return unauthorizedResponse(e); }
  /* ... existing proxy logic ... */
}
```

Seal decryption is only meaningful for a user who owns or has been granted access to a blob, so requiring a session costs legitimate users nothing. Note the credential is read from the request body elsewhere in the app; for a streaming proxy an `Authorization` header is the better carrier. Additionally, constrain `path` to the specific Seal endpoints the client actually calls, so the proxy cannot be repurposed as a general gateway. Rotate `ENOKI_SEAL_API_KEY` after deploying the fix, on the assumption it may already have been used.

M-01 MEDIUM Authentication path has no rate limit

> **Location:** `src/app/api/auth/challenge|session/route.ts`, `engine GetChallenge / CreateSession`
> **Objective:** (D) · **Status:** Rated assuming no Vercel Firewall rules - see I-01

`guardAiRoute` is applied to three of twenty-eight routes - the AI endpoints only. The authentication routes have no limiter, and neither do `/api/upload`, `/api/walrus/token` or the Seal proxy.

The authentication path is the one that matters most, for two reasons. First, it is unauthenticated by definition, so there is no identity to meter against once the request is accepted. Second, every call reaches the engine and touches Redis: `GetChallenge` writes a challenge entry, and `CreateSession` performs a signature verification, which for zkLogin means an outbound gRPC call to a Sui fullnode. An attacker can therefore convert cheap unauthenticated requests into Redis writes and third-party network calls at Waldrop's expense.

Compounding this, the engine is **directly reachable** at `:443` via Caddy, independently of Vercel. Any limiter added at the Vercel layer protects only the browser path; a caller who addresses the engine's gRPC endpoint bypasses it entirely (see I-02).

RECOMMENDATION

Rate-limit at both entry points, since they are independently reachable. At Vercel, a Firewall rule on `/api/auth/*` is the cheapest option and survives cold starts. At the engine, add a limiter in Caddy for the unauthenticated RPCs, keyed on source IP. Because both paths must be covered, prefer configuration at the edge over application code.

M-02 MEDIUM Declared per-identity rate limit is never enforced

> **Location:** `src/auth/types.rs`, `src/auth/mod.rs`
> **Objective:** (D) · **Status:** Static - confirmed by absence of any comparison site

The engine defines a per-identity rate limit and threads it through the authentication result:

```
pub struct RateLimit {
    requests_per_minute: u32,
    requests_per_hour: u32,
}
```

It is constructed for API keys, sessions and wallet identities, cloned into `AuthResult`, and **returned to the client in the session response**. No code anywhere compares a request count against either field. The values are surfaced as though they were a policy in force; they are inert.

This is distinct from the quota controls that *are* enforced. AI credit consumption is metered per identity in Redis and fails closed, and the transform and upload paths record tier limit hits. Those are volume and spend quotas. A request-rate limit is a different control and does not exist.

The risk is primarily one of misplaced confidence: an operator reading the configuration, or a client reading the session response, would reasonably conclude that per-identity request throttling is active. It is not, so an authenticated user can issue requests as fast as they like, bounded only by the quota controls on the specific expensive paths.

RECOMMENDATION

Either enforce it or remove it. If enforcing, a Redis counter keyed on identity and window matches the existing AI-credit pattern and can reuse the same failure semantics. If not, delete the struct and stop returning the fields to clients, so nothing downstream depends on a guarantee that is not made. Leaving inert policy in place is the worst of the three options.

M-03 MEDIUM Front-door limiter is per-instance on a horizontally scaled platform

> **Location:** `src/lib/rate-limit.ts`
> **Objective:** (D) · **Status:** Rated assuming no Vercel Firewall rules - see I-01

The limiter stores counters in a module-level `Map`. On Vercel each concurrent serverless instance holds its own copy, so a caller who triggers N warm instances receives roughly N times the intended limit, and counters reset on every cold start.

The file documents this precisely, and the surrounding implementation is careful in ways that matter:

```
// `x-forwarded-for` is client-controllable in general, but on Vercel the
// platform overwrites it, so the FIRST entry is the real peer. Reading the
// last entry instead — a common mistake — would let a caller pick their own
// bucket by sending a header.
const first = xff.split(",")[0]?.trim();
```

That reasoning is correct for Vercel specifically, and the fallback to a shared strict bucket rather than an exemption is the right default. The weakness is architectural, not a coding error: an in-process counter cannot bound a distributed request rate. It raises the cost of casual abuse and does not constitute a guarantee - which is exactly what the file claims for it.

RECOMMENDATION

Treat this as a defence-in-depth layer and place the actual guarantee at the edge, via Vercel Firewall rules. If an application-level guarantee is wanted, `checkRateLimit` is already the correct seam - swapping its body for Vercel KV or Upstash changes nothing else.

M-04 MEDIUM `requireSession` establishes validity, never identity

Location: `src/lib/api-auth.ts`
Objective: (A) · **Status:** Static - no exploitable instance identified; see I-03

The helper returns `Promise<void>`. It proves that the supplied credential is one the engine accepts, and tells the caller nothing about whose it is. A route that calls it knows only that *some* valid user is present.

This is safe today because the five routes using it pass the same credential straight to the engine, which resolves identity itself and gates per-RPC. But the shape invites a specific future bug: any handler that calls `requireSession` and then acts on an identifier taken from the request body - a wallet address, an account id, a job id - would be authenticated but not authorised, and the mistake would look correct at the call site. No current route does this; the risk is that the helper's signature makes it easy.

RECOMMENDATION

Change the return type to the resolved identity rather than `void`, so that using it without binding to a subject becomes awkward:

```
-export async function requireSession(auth: unknown): Promise<void>
+export async function requireSession(auth: unknown): Promise<{ address: string }>
```

This requires an engine RPC that returns the caller's identity - which L-01 recommends adding for a separate reason, so the two fixes share the work.

Low severity

L-01 LOW Session probe depends on an unrelated RPC's permission gate

`requireSession` has no dedicated validation RPC to call, so it invokes `GetJobStatus` with a deliberately impossible job id and infers the answer from the gRPC status code - `UNAUTHENTICATED` or `PERMISSION_DENIED` means reject, `NOT_FOUND` or `INVALID_ARGUMENT` means the credential cleared the gate. It fails closed on transport errors, which is correct.

The coupling is the issue, and the file states it: the check is only as strong as `GetJobStatus` keeping its `Permission::Transform` gate. If that gate is ever relaxed, this silently degrades to accepting any credential, with no failing test in the app to signal it. Add a `ValidateSession` RPC that returns the caller's identity; it costs less, says what it means, and resolves M-04 at the same time.

L-02 LOW No `script-src` Content-Security-Policy

`next.config.ts` sets `frame-ancestors 'none'`, `X-Frame-Options: DENY`, `nosniff`, HSTS with preload, `Referrer-Policy` and a restrictive `Permissions-Policy` - a strong set. There is deliberately no `script-src` directive, and the reasoning given is sound: the wallet SDK, Enoki and the Google Picker all inject at runtime, and a CSP that breaks wallet connect gets disabled under pressure, leaving a worse posture than starting narrow.

Given zero XSS sinks were found (section 6), the residual risk is low and this is a reasonable accepted risk. It is recorded here so it stays a decision rather than becoming an oversight. Deploy `Content-Security-Policy-Report-Only` with a `report-uri` first - that costs nothing, breaks nothing, and turns the eventual enforcing policy into a data-driven change. The one directive worth adding now regardless is `object-src 'none'`.

L-03 LOW Drive OAuth token is handed to client-side JavaScript

`/api/auth/gdrive/token` reads the `httpOnly` cookie and returns the access token in a JSON body, because the Google Picker runs in the browser and needs the token directly. This is a deliberate, documented trade, bounded by the `drive.file` scope - per-file access to what the user explicitly opened, so a stolen token cannot enumerate their Drive. The response is `no-store` and the endpoint is not cross-origin readable.

The reasoning holds. Two hardening steps: return the token only when the Picker is actually being opened rather than on any GET, and confirm the `drive.file` scope is what is requested in `/start` - the bound depends entirely on that, and a future scope widening would silently remove it.

L-04 LOW Long-lived cloud credentials transit the request body

The S3 connector accepts `accessKeyId` and `secretAccessKey` in the POST body, and the Azure and GCS connectors follow the same shape. These are long-lived credentials to storage the user controls, sent to Waldrop over TLS and used server-side.

The design is defensible for an ETL product. The risk is incidental capture: any request-body logging - in the route, in the engine, at the Vercel platform layer, or in an error path that serialises the request on failure - converts a transient credential into a stored one. **This was not verified and should be.** Confirm no logger touches these fields, including in exception handlers, and document that credentials are never persisted. Where the provider supports it, prefer scoped temporary credentials or a role assumption handshake over long-lived keys.

L-05 LOW Unimplemented RPC remains in the service definition

`TransformStream` is advertised in the proto and reachable, and its handler returns `unimplemented` before doing anything - so it carries no authentication check. That is harmless as written, and it is the correct order for a stub. It is recorded only because the absence of an auth call in a handler that is exposed on the public service is precisely the shape a reviewer must not become accustomed to skipping. Remove it from the proto until it is built, or add the permission gate now so the guard is present when the body is filled in.

Informational

I-01 INFO Vercel Firewall configuration must be confirmed

Vercel provides automatic L3/L4 volumetric DDoS mitigation on all plans, so the *network-layer* half of objective (D) is covered by the platform. Vercel does **not** apply application-layer rate limiting by default; that requires explicit Firewall rules, and availability varies by plan tier.

M-01 and M-03 are rated assuming no such rules exist, because the repository cannot show otherwise. Confirm in the Vercel dashboard whether Firewall rules cover `/api/auth/*` and `/api/seal/*`, and on which plan. If they do, both findings drop to informational. This is the single cheapest item to resolve in this report and it changes two severities.

I-02 INFO The engine is a second internet-facing entry point

The deployment places Caddy in front of the engine on `:443`, terminating TLS via Let's Encrypt and proxying to h2c on internal `:50052`, which is not exposed in the security group. Transport security is correct and session tokens are protected in transit.

The architectural consequence deserves stating: **the engine is reachable without going through Vercel.** Vercel's DDoS mitigation, any Firewall rules, and every application-layer control in the Next.js app apply only to the browser path. A caller addressing the engine directly is governed solely by the engine's own per-RPC permission gates - which are sound for authenticated RPCs, and absent for `GetChallenge`, `CreateSession`, `HealthCheck` and `ListTransforms`. Any control intended as a platform-wide guarantee must be implemented at both entry points or at a layer common to both.

I-03 INFO Horizontal access control untested

No IDOR or cross-tenant testing was performed, because it requires two live accounts and a running target. Static review found no route that acts on a body-supplied identifier after only a validity check, but that is an argument from absence: the engine resolves identity from gRPC metadata internally, and confirming it never honours a caller-supplied subject requires exercising it. This is the largest residual gap in the assessment and the primary reason to run a dynamic phase.

I-04 INFO Redis is a soft dependency of authentication

Challenges and consumed nonces are written to both Redis and an in-process map, with Redis preferred and the map used when it is unavailable. The fallback repeats the full discipline of the primary path - challenge binding, nonce-before-verification, bounded map with TTL pruning - and the analysis in section 6 finds it correct.

One property worth recording: with multiple engine replicas and Redis down, a challenge issued by one replica is unknown to the others, so authentication fails rather than falling open. That is the right failure direction. The operational consequence is that a Redis outage degrades authentication availability across replicas, which is a capacity and alerting concern rather than a security one.

6 Controls Verified Effective

This section records what was tested and found correct. It is not filler: several of these are controls that commonly fail, and their correctness here is the reason the finding count is low.

6.1 Authentication and replay protection - no findings

The challenge/signature/session flow is correct in every respect examined:

- The challenge is stored keyed by wallet address, so it is bound to the identity it was issued for and cannot be redeemed by another.
- A three-minute timestamp window bounds signature reuse independently of the nonce.
- The nonce is consumed atomically via Redis `SET NX`, which the code correctly identifies as closing the check-then-act race that a `GET`-then-`SET` would leave open.
- **The nonce is marked used before the signature is verified.** This is the detail most implementations get wrong: verifying first and consuming after lets an attacker retry a captured challenge with different candidate signatures. Both the Redis and in-memory paths do it in the correct order.
- The signed message binds wallet, timestamp and nonce together, so none can be substituted independently.
- The challenge is deleted on use, and the in-memory copy is deleted alongside the Redis copy specifically so the fallback path cannot re-accept it.

6.2 zkLogin integration - no findings (objective B)

zkLogin is not reimplemented. Scheme byte `0x05` is delegated to Sui's `SignatureVerificationService` over gRPC - the same verifier validators run. Three properties were checked specifically:

- **The claimed address is passed to the verifier** (`address: Some(wallet.to_string())`), so the check is "this signature is valid *for this address*", not merely "this signature is well-formed". Omitting this binding is the canonical zkLogin integration failure and it is not present here.
- **JWKs are left empty**, so the verifier uses on-chain keys rather than any supplied by the caller. Accepting client-supplied JWKs would allow an attacker to present a self-signed JWT with a matching key and authenticate as any address.
- **It fails closed.** `is_valid` is read with `unwrap_or(false)`; network errors, malformed responses and an unconfigured `SUI_GRPC_URL` all deny.

Native schemes 0x00 Ed25519, 0x01 Secp256k1, 0x02 Secp256r1 and 0x06 Passkey/WebAuthn are verified in-process against the correct Sui signing domain (`Blake2b-256(intent || BCS(message))`).

6.3 XSS and CSRF - no findings (objective E)

- **Zero XSS sinks.** No `dangerouslySetInnerHTML`, `innerHTML`, `eval`, `new Function` or `document.write` anywhere in the application source. React's default escaping is therefore the only rendering path.

- **OAuth CSRF is correctly prevented.** Both the Drive and Dropbox flows issue a state value, store it in an `httpOnly` cookie, compare it against the returned parameter, and delete it on use.
- **Cookies are correctly flagged** - `httpOnly`, `secure` in production, `sameSite=lax`, scoped `maxAge` mirroring token expiry.
- **Open redirect is properly handled.** `safe-redirect.ts` resolves the candidate against a sentinel origin and compares the result, rather than pattern-matching the string. This catches protocol-relative (`//evil.com`) and backslash-normalised (`/\evil.com`) payloads that defeat a `startsWith("/")` check, and it returns a path rather than a URL so callers cannot reintroduce the problem.
- **State-changing routes are POST with JSON bodies** carrying credentials, so they are not reachable by simple form-based cross-site submission.

6.4 SSRF - no findings

The engine validates user-supplied source URLs through a dedicated module implementing: private and loopback IP rejection, **IP pinning** between validation and connection, and **independent revalidation of every redirect hop** under a maximum hop count. The first defeats DNS rebinding; the second defeats redirect-to-internal. A private-IP check alone - which is what most implementations ship - loses to both. Bucket-name and region validators are applied to the S3 and GCS paths.

6.5 Authorization and transport

- Every engine RPC that performs work is gated on an explicit permission. The ungated ones are the login pair, health, and a static list - appropriate in each case.
- gRPC is TLS-terminated at Caddy on `:443`; the plaintext h2c listener on `:50052` is internal and not exposed.
- The metrics endpoint refuses to bind a non-loopback address without `METRICS_TOKEN` - a fail-closed default on an endpoint that is routinely left open.
- **No secrets in the client bundle.** Every `NEXT_PUBLIC_*` variable is legitimately public: the Enoki *public* key, a Google client id and API key, analytics id, and public RPC and aggregator URLs.
- `.env` files are not tracked in git.

7 Coverage Against Objectives A–E

OBJ	OBJECTIVE	RESULT	FINDINGS
A	Platform authentication & API endpoints	Authentication core is correct and the engine gates every working RPC. One route exposes a server credential with no gate (H-01). The app-layer session helper proves validity but not identity (M-04, L-01). Cross-account access control remains untested (I-03).	H-01, M-04, L-01, L-03, L-04, L-05, I-02, I-03
B	zkLogin workflow & key handling	**No findings.** Delegated to Sui's canonical verifier with the address bound, on-chain JWKs enforced, and deny-by-default on every error path. Ephemeral key material is never handled server-side.	-
C	Seal encryption usage & key management	Seal usage is correct - the server never sees plaintext or user key material, and the proxy's header hygiene actively prevents leaking users' third-party OAuth tokens upstream. The finding is access control on the proxy, not cryptographic handling.	H-01
D	Rate limiting & DDoS protection	Weakest area. Volumetric DDoS is covered by Vercel at the network layer. Application rate limiting covers 3 of 28 routes, is per-instance, omits the authentication path, and the engine's declared per-identity limit is not enforced. The engine is separately reachable, so Vercel-layer controls do not cover it.	M-01, M-02, M-03, I-01, I-02
E	XSS / CSRF	**No findings.** No injection sinks exist. OAuth CSRF, cookie flags, open-redirect handling and framing protection are all correct. The absent `script-src` CSP is a deliberate, documented trade.	L-02

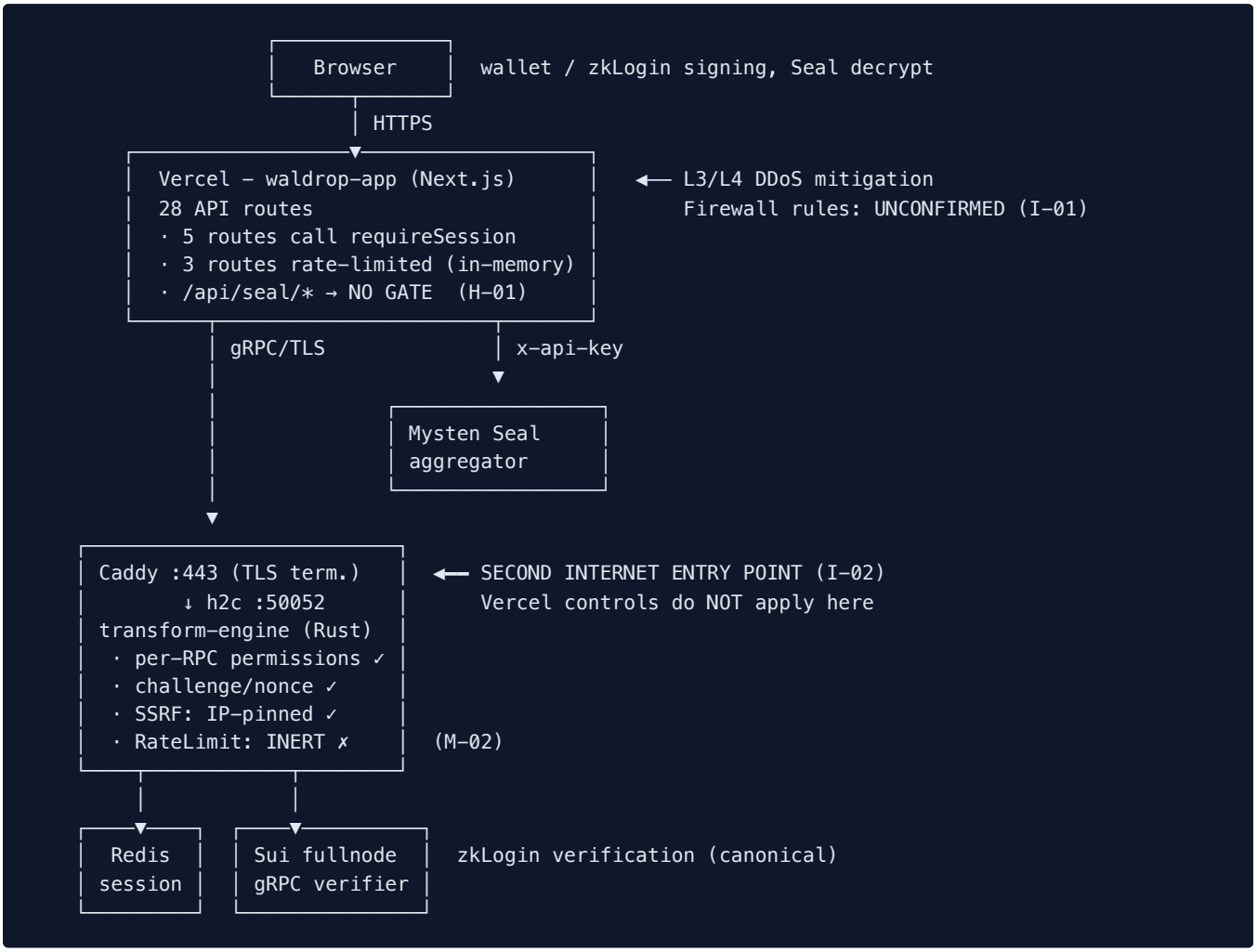
> **READING THIS TABLE**
>
> "No findings" against B and E means no issue was identified by static review of the relevant code, not that the areas are proven secure. For B that conclusion is strong: the integration is small, and the three properties that determine its safety were each checked directly. For E it is strong for injection, since sinks were enumerated exhaustively, and weaker for CSRF, where confirming that no state-changing route accepts a cross-site request is better established dynamically than by reading handlers.

8 Remediation Checklist

ID	PRIORITY	ACTION
H-01	IMMEDIATE	Add session check + rate limit to the Seal proxy; constrain the forwarded path; rotate `ENOKI_SEAL_API_KEY` after deploying.

I-01	IMMEDIATE	Confirm Vercel Firewall rules and plan tier. Resolves or re-rates M-01 and M-03 at no engineering cost.
L-04	IMMEDIATE	Verify cloud credentials in request bodies never reach any log, including error paths.
M-01	THIS SPRINT	Rate-limit `/api/auth/*` at Vercel **and** the unauthenticated RPCs at Caddy - both entry points, since they are independently reachable.
M-02	THIS SPRINT	Enforce `RateLimit` with a Redis counter, or delete it and stop returning it to clients.
M-04, L-01	THIS SPRINT	Add a `ValidateSession` RPC returning caller identity; change `requireSession` to return it. One change closes both.
M-03	NEXT	Move the guarantee to the edge; keep the in-memory limiter as defence in depth.
L-02	NEXT	Ship `CSP-Report-Only` with a report endpoint; add `object-src 'none'` now.
L-03, L-05	NEXT	Narrow the Drive token endpoint; verify the `drive.file` scope; gate or remove `TransformStream`.
I-03	DYNAMIC PHASE	Cross-account access control testing with two live accounts.

Appendix A Architecture & Trust Boundaries



TRUST BOUNDARIES

BOUNDARY	CROSSING	CONTROL
Browser → Vercel	Untrusted input	Per-route; inconsistent (H-01, M-01)
Vercel → engine	Credential forwarded as gRPC metadata	Engine per-RPC permission gate ✓
Internet → engine	**Direct, bypasses Vercel**	Engine gates only; unauthenticated RPCs unmetered (I-02)
Engine → user sources	User-supplied URLs and credentials	SSRF validation with IP pinning ✓
App → Mysten Seal	Server credential attached	**None** (H-01)

Engine → Sui fullnode	Signature verification	TLS, fails closed ✓

Appendix B Route & RPC Inventory

API ROUTES BY PROTECTION STATE

STATE	COUNT	ROUTES
Session-checked	5	`cloud/s3, cloud/gcs, cloud/azure, cloud/gdrive, cloud/dropbox`
Rate-limited	3	`ai/pipeline, ai/policy, ai/preview` (in-memory, per-instance)
Engine-enforced	7	`transform, transforms, upload, profile, walrus, walrus/token, session`
Login flow (unauth by design)	3	`auth/challenge, challenge, auth/session`
Cookie-authenticated	7	`auth/gdrive/{start,callback,token,disconnect}, auth/dropbox/{start,callback,disconnect}`
Public	2	`health, seal-check`
Ungated with a secret	1	`seal/[...path]` H-01

GRPC RPCS

RPC	PERMISSION	NOTE
`Transform`	`Transform`	Tier quota enforced
`Upload`	`Transform`	Tier quota enforced
`RequestUploadToken`	`Transform`	-
`PreviewTransform`	`Transform`	-
`GetJobStatus`	`Transform`	Also used as the session probe (L-01)
`ConsumeAiCredit`	`Transform`	Redis-metered, fails closed
`ProfileData`	`Profile`	-
`CancelJob`	`ManageJobs`	-
`GetChallenge`	none - by design	Unmetered (M-01)

`CreateSession`	none - by design	Unmetered; triggers signature verification (M-01)
`VerifySignature`	none	Stateless verification
`HealthCheck` / `ListTransforms`	none	Static response
`TransformStream`	none	Returns `unimplemented` (L-05)

Disclaimer

This assessment was performed by **The Blockchain Team** (https://theblockchain.team). This assessment covers the components listed in section 2.1 as they existed on 14 August 2026. It is a **static source review**: no request was issued against any running Waldrop system, and no finding has been confirmed by dynamic exploitation. Severity ratings for M-01 and M-03 depend on runtime configuration not visible in the repository and are stated on the assumption recorded in I-01. A security review is a best-effort exercise under time constraints and cannot prove the absence of vulnerabilities; "no findings" against an objective means no issue was identified by the methods described, not that the area is proven secure. Third-party services were assessed only through their integration points. Any change to the code after this date invalidates these results.

ATTESTATION

The Blockchain Team · https://theblockchain.team
Issued 17 August 2026 · Report version 1.0

- End of report -