PUBLIC SAMPLE — REDACTED. A genuine Vari review with the client's identity and identifying scope details removed (shown as [redacted]). The redacted text is stripped from the file, not merely hidden. Findings, severities, locations, evidence and recommendations are unaltered.

Vari

FULL-STACK SECURITY REVIEW

Security Review

— Track 5 — App · Three production dApps · signing flows, API routes and address configuration

CLIENT
[redacted]

REPORT DATE
2026-08-28

STACK
TypeScript · React · SvelteKit · Next.js

TIER
Full

VERSION REVIEWED
[redacted]

02 Executive Summary

0	12	19	10	4
CRITICAL	HIGH	MEDIUM	LOW	INFO

12 findings at High severity or above. The most serious is **PROTO-P0008 — Wallet auth challenges are not bound to an origin, and the [redacted] dApp SIWE nonce is a 30-day static cookie value that is never consumed**. An attacker who can get the admin wallet to sign one string on any page they control obtains an admin Bearer session on the XRPL vault (full admin API: protocol config, payments, withdrawals) or an admin cookie pair on the [redacted] dApp (merkle root and strategy/action management). For ordinary users the same weakness lets a look-alike site mark a wallet 'verified' in the vault database.

In total 45 findings: 12 High · 19 Medium · 10 Low · 4 Info. Each is stated with its location in the source, the conditions under which it is reachable, and a recommended fix. Findings that were driven to a working proof

carry that proof beneath them.

03 Scope of Review

LAYER	STATUS
Frontend and dApp	Reviewed
Backend and API	Reviewed
Wallet signing flows	Reviewed

Version reviewed: Local snapshot at the provided source snapshot — no .git directory present in any of the three dapps, so no commit hash is recoverable. File mtimes: [redacted] dApp 2025-08-11, [redacted] dApp 2025-07-27, [redacted] dApp 2025-06-09.

04 Methodology

The review began with the architecture: every path along which value moves — minting, redemption, deposits, withdrawals, staking, bridging and settlement — was traced end to end, and every privileged role was mapped to the actions it can take and the key that holds it. Each module was then read in full against that map, looking for places where the code does something other than what its author intended, where a control is present but does not bind, or where value can be created or destroyed without a corresponding entry elsewhere.

That reading was checked against the defect classes this kind of system has failed on historically — drawn from the published record of audited protocols — so that a known failure mode is not missed because it did not occur to the reviewer on the day. Alongside it, a separate reading targeted the opposite risk: defects specific to this protocol's own logic, which by definition match no known pattern and which no checklist would surface.

Nothing was reported on suspicion. The contracts were compiled and deployed locally through their own initialization scripts, and each hypothesis was driven to a verdict by a test that either reproduces the problem or fails to. Where a finding below carries a proof, that test was executed and its output is quoted verbatim. Where a path could not be exercised in a local environment, it is stated as unverified rather than presented as sound. Findings were then ranked by what an attacker actually gains and what it costs them to try.

All work was performed against a local build. No transaction was sent, no production endpoint was contacted, and no client key was handled at any point during this review.

05 Findings

PROTO-P0008 · High

Wallet auth challenges are not bound to an origin, and the [redacted] dApp SIWE nonce is a 30-day static cookie value that is never consumed

LOCATION

[redacted] dApp/lib/xrpl-signature-verify.js:7; [redacted] dApp/lib/api/admin-router.js:105; [redacted] dApp/src/shared/utils/auth.ts:17; [redacted] dApp/src/app/api/admin/nonce/route.ts:10

Both apps authenticate a wallet by having it sign a server-issued challenge, and in both the signed material omits the field that would tie the signature to this application. The XRPL vault signs a raw string containing only a prefix, a nonce and an expiry: nothing in it names vaults.[redacted], so a signature harvested on any other page is a valid credential here. The [redacted] dApp does construct a full SIWE message client-side (domain = window.location.host) but the server verifier never checks that field, and the nonce it checks against is read from an x-nonce cookie that /api/admin/nonce hands back unchanged for 30 days and that no code path ever deletes - so one captured admin SIWE signature stays valid for the whole cookie lifetime and can be replayed from any client that sets the same cookie value.

Impact. An attacker who can get the admin wallet to sign one string on any page they control obtains an admin Bearer session on the XRPL vault (full admin API: protocol config, payments, withdrawals) or an admin cookie pair on the [redacted] dApp (merkle root and strategy/action management). For ordinary users the same weakness lets a look-alike site mark a wallet 'verified' in the vault database.

Recommendation. Bind the origin into the signed material: use full SIWE (or EIP-4361-style text) including domain, uri, issuedAt and expirationTime, and validate domain/scheme server-side against an allow-list. In [redacted] dApp, generate a fresh nonce per login attempt and delete it on first use (or on failure) instead of returning the cached cookie value; give the session its own short-lived, revocable token rather than re-verifying a stored signature on every request.

> PROOF OF CONCEPT — EXECUTED reasoned + code-read - not executed (LOCAL ONLY; no live app or API called)
>
> dapps/[redacted] dApp/lib/xrpl-signature-verify.js:7-10 buildAuthMessage(kind,nonce,expUnix) -> `${prefix}:v1:${nonce}:${expUnix}`; dapps/[redacted] dApp/lib/api/admin-router.js:105-170 (challenge issued unauthenticated, signature accepted, createSession(address)); dapps/[redacted] dApp/src/shared/utils/auth.ts:17-27 validateSiweMessage without domain; dapps/[redacted] dApp/src/app/api/admin/nonce/route.ts:10-14 returns the cached cookie nonce unchanged

PROTO-P0092 · **High**

POST /api/withdraw on the XRPL vault authorises on a permanent per-address flag, so any third party can force any depositor's position into the withdrawal queue

LOCATION [redacted] dApp/lib/api/public-router.js:384-427

The withdrawal endpoint parses {address, amount} and, after registering the user if needed, checks only isWalletVerifiedMerged(address) - a boolean set once, forever, when that address completed a wallet-verification signature at some point in the past. There is no Bearer session on this route, no per-request signature, no nonce, and nothing that binds the HTTP caller to the XRPL account named in the body. The browser is what enforces 'withdrawals require a verified wallet connection': index.html refuses when walletProvider === 'manual', and then issues an ordinary unauthenticated fetch that anybody can reproduce.

Impact. Every verified depositor's address is discoverable from the XRPL ledger (they all sent a tagged Payment to the vault's public deposit address), and GET /api/user/:address/verified confirms which ones are verified. An attacker can therefore call POST /api/withdraw for each of them, up to their full balance, forcing applyVaultWithdrawalReserve to lock their position and enqueuing a 7-day withdrawal they never asked for. Funds are paid to the rightful owner, so this is not theft, but it is an unauthorised, unlimited, un-cancellable-by-the-victim state change on other users' positions: yield accrual stops, the vault's liquidity buffer is drained on the attacker's schedule, and the treasury is forced to fund the withdrawal wallet. Rate

limiting does not contain it because the limiter key is the client-supplied X-Forwarded-For (see PROTO-N02).

Recommendation. Require a per-withdrawal signature over a server-issued nonce that names the address, the amount and the origin, or issue the user a short-lived session token at verify-wallet time and require it here. Make the 'verified' flag expire. Also allow the owner to cancel a pending request.

PROOF OF CONCEPT — EXECUTED reasoned + code-read - not executed (LOCAL ONLY)

```
dapps/[redacted] dApp/lib/api/public-router.js:384-427 (POST /withdraw, only
isWalletVerifiedMerged(address)); dapps/[redacted] dApp/src/app/api/[redacted]/route.ts:9,94 (isServerRequest
= presence-only, master token attached); dapps/[redacted] dApp/src/routes/api/crons/coin-
prices/+server.ts:11 (no auth); dapps/[redacted] dApp/src/app/api/send-email/route.ts:88-101; dapps/
[redacted] dApp/src/lib/sdk/services/transaction.ts:87-105 (receipt returned, status never read)
```

PROTO-P0171 · **High**

All three dApps ship dependencies with published High/Critical advisories, including a Critical node-tar and a runtime axios in the SvelteKit app

LOCATION

```
[redacted] dApp/package.json (axios ^1.17.0, package-lock.json
tar<=7.5.20); [redacted] dApp/package.json + pnpm-lock.yaml (next 15.5.18,
shell-quote, axios, hono); [redacted] dApp/package.json (next ^15.5.18)
```

Audits over the committed lockfiles show live advisories that the toolchain has not absorbed. The two that matter most for production behaviour are axios in the vulnerable 1.0.0-1.17.0 band as a direct runtime dependency of the SvelteKit app (its server routes proxy to the [redacted] backend, CoinMarketCap and the Discord notifier), and Next.js 15.5.18 in both Next apps, which sits inside the advisory range for SSRF in rewrites via an attacker-controlled destination hostname, SSRF in Server Actions, response-body cache confusion, and unauthenticated disclosure of internal Server Function endpoints. node-tar <=7.5.20 is Critical but reaches only the build/install path.

Impact. The Next advisories are directly relevant because both apps are the trust boundary in front of privileged upstreams (SUBSQUID_MASTER_TOKEN, SENDGRID_API_KEY, ADMIN_ADDRESS session verification, the XRPL vault's Express admin API). Cache confusion and Server Function disclosure on those hosts would leak or cross-wire exactly those responses.

Recommendation. Bump next to the fixed patch line in both Next apps, bump axios past 1.17.0 in the SvelteKit app, and add lockfile overrides/resolutions for tar, form-data, nanoid, postcss, brace-expansion and shell-quote. Wire `npm audit --audit-level=high` / `pnpm audit` into CI so the lockfile cannot drift back.

PROOF OF CONCEPT — EXECUTED npm audit --json ([redacted] dApp, node_modules present); npm audit --package-lock-only ([redacted] dApp); pnpm audit --json ([redacted] dApp). All local, no app or RPC contacted.

```
[redacted] dApp: {info:0,low:1,moderate:0,high:5,critical:0}; [redacted] dApp:
{low:65,moderate:134,high:20,critical:1}; [redacted] dApp: CRITICAL shell-quote + High
axios/undici/ws/hono/next/immutable/js-yaml/brace-expansion/sharp/postcss/socket.io-parser/nanoid.
Installed next = 15.5.18 (node -p require('next/package.json').version).
```

PROTO-R01 · High

Unauthenticated [redacted] email endpoint (brand-spoofed phishing / quota abuse)

LOCATION `none - the handler's only gate is a presence check on template/to/values`

`[redacted] dApp/src/app/api/send-email/route.ts:88-101` sends mail through [redacted] with no caller authentication. After two environment-variable presence checks (`SENDGRID_API_KEY`, `CLEARPOOL_EMAIL`) the only validation is `if (!template || !values || !to) return 400` -- no header, no cookie, no session, no rate limit. The `to` field from the request body is used verbatim as the recipient, and every `values.*` field is interpolated into the HTML body by `getHtml` with no escaping. The SvelteKit sibling at `[redacted] dApp/src/routes/api/send-email/+server.ts:16-20` does gate on a `token` header compared against `CLEARPOOL_TOKEN`, though with a non-constant-time `!==`. The in-app client only ever posts `vaultConfig.borrowerEmail`, but nothing server-side constrains the recipient to that.

Impact. Anyone on the internet can send arbitrary HTML from [redacted]'s [redacted] account to any recipient they name, under [redacted] branding and from [redacted]'s sending domain -- a ready-made phishing channel that inherits the domain's deliverability and reputation. Because `values.*` is interpolated unescaped, the attacker controls the rendered body, not merely the text substituted into a template. The secondary costs are quota exhaustion on the [redacted] account and sender-reputation damage to the domain if it is used at volume.

Recommendation. Authenticate the handler before it sends: gate `[redacted] dApp/src/app/api/send-email/route.ts` on a server-side shared secret compared with `crypto.timingSafeEqual` (or on an authenticated session), as the SvelteKit sibling already does, and add per-IP and per-recipient rate limiting. Stop letting the client choose the recipient -- resolve `to` server-side from the vault config keyed by an id in the body -- and HTML-escape every `values.*` field in `getHtml` instead of interpolating raw. The escaping is needed independently of the auth fix, since the rendered output goes out under [redacted] branding.

> PROOF OF CONCEPT — EXECUTED Read [redacted] dApp/src/app/api/send-email/route.ts end to end and compared it with the SvelteKit equivalent. Cross-referenced only; not re-proved, as instructed.

```
route.ts:88-101: after `if (!process.env.SENDGRID_API_KEY)` and `if (!process.env.CLEARPOOL_EMAIL)` the only check is `if (!template || !values || !to) return 400`. No header, no cookie, no session, no rate limit. `to` is used verbatim as the recipient and every values.* field is interpolated into the HTML body with no escaping (getHtml). SIBLING: [redacted] dApp/src/routes/api/send-email/+server.ts:16-20 does gate on a `token` header against CLEARPOOL_TOKEN (though with a non-constant-time !==). The client caller only ever sends to vaultConfig.borrowerEmail, but nothing server-side enforces that.
```

PROTO-R02 · High

Admin session secret fails open to an in-repo default in production

LOCATION `none - the production check only console.error()s and execution continues`

`[redacted] dApp/lib/auth/admin-session.js:11-22` reads `ADMIN_SESSION_SECRET` and, when it is unset in production, logs and continues: `if (!SESSION_SECRET_RAW && isProdRuntime()) { console.error(...) }` is followed by `crypto.createHash('sha256').update(SESSION_SECRET_RAW || 'dev-only-insecure-session-secret-change-me')`. The fallback literal is committed in the source tree, and the digest of it becomes the key that HMACs every admin Bearer token. There is no throw, no process exit and no runtime refusal, so a production deployment missing the variable starts cleanly and issues tokens

signed with a published constant. The same secret is the only thing standing behind the token `requireAuth()` accepts.

Impact. If `ADMIN_SESSION_SECRET` is unset in any production deployment, anyone who can read the repository can mint a valid `{address, iat, exp, jti}` session for the admin address and obtain full administrative access to the vault application. The failure is silent -- the only signal is a `console.error` line in the logs -- so the misconfiguration can persist indefinitely unnoticed. The precondition is an environment misconfiguration rather than an attack, but the code is written so that the safe outcome depends on that configuration being right and the unsafe outcome is what happens by default.

Recommendation. Make the missing secret fatal: in `[redacted] dApp/lib/auth/admin-session.js:11-22`, throw at module load when `ADMIN_SESSION_SECRET` is unset and `isProdRuntime()`, rather than logging and falling through to `'dev-only-insecure-session-secret-change-me'`. Remove the fallback literal from the source tree altogether -- generate a random value into `.env.local` for development -- so the string that HMACs every admin Bearer token is never a published constant. Rotate the secret on deploy and invalidate outstanding sessions, since any token minted under the default is currently valid.

> PROOF OF CONCEPT — EXECUTED Read [redacted] dApp/lib/auth/admin-session.js:11-22.
>
> `const SESSION_SECRET_RAW = cleanEnv('ADMIN_SESSION_SECRET'); if (!SESSION_SECRET_RAW && isProdRuntime()) { console.error('[security] ADMIN_SESSION_SECRET must be set in production...') } const SESSION_SECRET = crypto.createHash('sha256').update(SESSION_SECRET_RAW || 'dev-only-insecure-session-secret-change-me').digest('hex');` - the fallback string is in the source tree and is used to HMAC every admin Bearer token, so with the env var unset anyone can mint a valid `{address,iat,exp,jti}` session for the admin address. Unchanged since the prior review. Note the same secret is now also the only thing standing behind the token accepted by requireAuth().

PROTO-R05 · High

Spoofable referer-based access control on the [redacted] GraphQL proxy

LOCATION	none - and the prior mitigating reasoning does not hold

`[redacted] dApp/src/app/api/[redacted]/route.ts` authorises requests by substring-matching the `Referer` header -- four `referer.includes(...)` early returns, one of which matches any host containing `-[redacted].vercel.app` in preview. `Referer` is client-supplied and freely forged, so it is not a credential. The escalation comes from two helpers disagreeing about what a server request is: `isServerRequest(req)` is `!!req.headers.get('x-[redacted]-token')` -- presence, not value -- while `validateRequest` accepts a correct token in its first branch and otherwise falls through to the `Referer` check. A request carrying a bogus `x-[redacted]-token` therefore passes authorisation via the `Referer` branch, is then classified as a server request, and is forwarded upstream with `'x-[redacted]-token': process.env.SUBSQUID_MASTER_TOKEN`.

Impact. Any anonymous caller who sets two headers -- a made-up `x-[redacted]-token` and a `Referer` containing one of the matched strings -- reaches the upstream [redacted] API with the privileged master token attached. That is an escalation from anonymous relay to master-token-privileged access, so the proxy hands out an authority level it was never intended to expose. No user interaction, session or prior access is needed, and the preview wildcard means any host whose name contains `-[redacted].vercel.app` also satisfies the check.

Recommendation. Stop authorising on `Referer`: replace the four `referer.includes(...)` early returns in `[redacted] dApp/src/app/api/[redacted]/route.ts` with a verified credential, and close the escalation

first -- `isServerRequest(req)` tests only for the presence of `x-[redacted]-token` while `validateRequest` accepts a correct one or else falls through to the Referer branch, so a bogus token plus a spoofed Referer is forwarded upstream carrying `SUBSQUID_MASTER_TOKEN`. Make attachment of the master token conditional on the token's *value*, not on header presence, and delete the preview wildcard that matches any host containing `-[redacted].vercel.app`. If the proxy must remain reachable from browsers, give it its own scoped read-only [redacted] token and never attach the master one to a browser-originated request.

> PROOF OF CONCEPT — EXECUTED Read [redacted] dApp/src/app/api/[redacted]/route.ts end to end, tracing isServerRequest() and validateRequest() separately rather than assuming they agree.
>
> The Referer substring matching is unchanged (four `referer.includes(...)` early returns, one of which matches any host containing '-[redacted].vercel.app' in preview). The prior register concluded the impact was limited because 'the privileged master token is only attached to server requests'. That is not what the code does: isServerRequest(req) is `!!req.headers.get('x-[redacted]-token')` - presence, not value - while validateRequest only accepts a CORRECT token in its first branch and otherwise falls through to the Referer check. A request with a bogus x-[redacted]-token and a spoofed Referer therefore passes authorisation via the Referer branch and is then forwarded with `'x-[redacted]-token': process.env.SUBSQUID_MASTER_TOKEN`. This is an escalation from anonymous relay to master-token-privileged access. Reported in full as novel PROTO-N01.

PROTO-R07 · **High**

Test library (supertest) and full Express app in the production request path

LOCATION	none - and the prior recommendation's caveat about X-Forwarded-For is now a confirmed finding

`[redacted] dApp/app/api/[[...path]]/route.ts:3` imports `request from 'supertest'`, and `supertest` is declared in `dependencies` rather than `devDependencies`, so a testing library sits in the production request path. Every inbound request is replayed into the in-process Express app via `request(app)[methodName](fullPath)` with all client headers forwarded except `host` and `content-length`. That blanket forwarding is what makes the header-trust problem live: `lib/backend/express-app.js:115-121` keys every limiter on the leftmost value of the client-supplied `X-Forwarded-For`, with express-rate-limit's own protections disabled by `validate: { trustProxy: false, xForwardedForHeader: false }`. The attacker's own `X-Forwarded-For` therefore crosses the Next.js boundary intact and becomes the limiter bucket key.

Impact. A test harness executes on every production request, carrying its own dependency tree and error handling into the serving path, and the header forwarding it performs makes every rate limiter in the app bypassable by varying one header per request -- `adminAuthStrictLimiter` on `/admin/login` and the wallet-auth routes included. The immediate consequence is unthrottled brute force against the admin authentication routes and no containment on the unauthenticated withdrawal endpoint. The `supertest` dependency also widens the production attack surface and the patching burden for no functional benefit, since the Express app can be invoked directly.

Recommendation. Take `supertest` out of the request path: move it to `devDependencies` and have `[redacted] dApp/app/api/[[...path]]/route.ts` invoke the Express app directly (a `createServer(app)`-backed handler, or port the routes to native Next handlers) instead of replaying every request through `request(app)`. Fix the header trust independently -- `express-app.js:115-121` keys every limiter, `adminAuthStrictLimiter` included, on the leftmost client-supplied `X-Forwarded-For` with `validate: { trustProxy: false, xForwardedForHeader: false }`. Set `app.set('trust proxy', 1)` and key on the resolved

`req.ip` or the platform's own client-IP header, and stop blanket-forwarding the client's `X-Forwarded-For` across the boundary.

> PROOF OF CONCEPT — EXECUTED Read [redacted] dApp/app/api/[[...path]]/route.ts and followed the forwarded headers into lib/backend/express-app.js's limiter key generator.
>
> ```
> route.ts:3 still `import request from 'supertest'`, and supertest remains a production dependency in package.json (not devDependencies). Every request is replayed into the in-process Express app via `request(app)[methodName](fullPath)` with all client headers forwarded except host and content-length. The prior recommendation asked the team to 'ensure the app does not trust client X-Forwarded-For for rate-limit/IP logic'; it does. express-app.js:115-121 keys every limiter on the leftmost value of the client-supplied X-Forwarded-For, with express-rate-limit's own validators disabled (validate: { trustProxy: false, xForwardedForHeader: false }), so the blanket header forwarding makes every limiter - including adminAuthStrictLimiter on the admin login and wallet-auth routes - trivially bypassable. Reported in full as novel PROTO-N02.
> ```

PROTO-N01 · **High**

Presence-only check on x-[redacted]-token makes the [redacted] proxy attach the privileged master token to any unauthenticated request

LOCATION [redacted] dApp/src/app/api/[redacted]/route.ts:9 (isServerRequest), :12-39 (validateRequest), :94 (x-[redacted]-token)

The proxy has two separate notions of 'this is a trusted server request' and they disagree. validateRequest() authorises a request if EITHER the x-[redacted]-token header equals OZEAN_ACCESS_TOKEN OR the client-supplied Referer contains one of four substrings. isServerRequest() is a different, weaker test: `return !!req.headers.get('x-[redacted]-token')` - it checks only that the header EXISTS, never that its value is correct. The upstream call then does `'x-[redacted]-token': isServerRequest(req) ? process.env.SUBSQUID_MASTER_TOKEN ?? '' : ''`. So a request carrying a garbage x-[redacted]-token value fails the token branch of validateRequest, falls through to the Referer branch, passes on a header the client also controls, and is then forwarded to the [redacted] GraphQL endpoint WITH the master token attached. Two attacker-controlled headers - one whose value is never checked and one that is trivially spoofable - combine into a privileged upstream credential.

Impact. Any unauthenticated client can issue GraphQL operations against the [redacted] backend at whatever privilege level SUBSQUID_MASTER_TOKEN confers - the same credential the server-to-server path uses, and strictly more than the anonymous relay the prior review scoped this endpoint to. The prior register explicitly reasoned that this was safe ('The privileged master token is only attached to server requests, so this is limited to unauthenticated read-query relay/abuse'); that reasoning does not hold. The same master token is also used by /api/stats/vaults, so its blast radius is the whole indexer.

Recommendation. Make isServerRequest() a value comparison (constant-time) against OZEAN_ACCESS_TOKEN, not a presence test, and derive it from the same result validateRequest already computed rather than re-deriving it. Drop Referer authorisation entirely in favour of exact-host Origin allow-listing or a signed token, and rate-limit the anonymous path.

ANALYSIS Reasoned, not executed (LOCAL ONLY - no request was made to the live proxy or to [redacted]). Reproduction for the client to run in a staging environment: POST /api/[redacted] with headers `x-[redacted]-token: not-the-real-token` and `Referer: https://vaults.[redacted]/` and any GraphQL body; assert the outbound request to holdex.squids.live does NOT carry x-[redacted]-token.

```
isServerRequest(req) { return !!req.headers.get('x-[redacted]-token') }  // route.ts:9-11  ---
validateRequest: `if (isDev || req.headers.get('x-[redacted]-token') === process.env.OZEAN_ACCESS_TOKEN)
return` then four `referer.includes(...)` early returns  --- upstream fetch header: `'x-[redacted]-token':
isServerRequest(req) ? process.env.SUBSQUID_MASTER_TOKEN ?? '' : ''`
```

PROTO-N02 · **High**

Every rate limiter in the XRPL vault keys on the raw client-supplied X-Forwarded-For, which the Next.js entrypoint forwards verbatim

LOCATION

[redacted] dApp/lib/backend/express-app.js:115-121 (limiterKeyGen);
[redacted] dApp/app/api/[[...path]]/route.ts:20-36 (header forwarding)

limiterKeyGen reads req.headers['x-forwarded-for'] and, if present, returns its FIRST comma-separated element as the limiter bucket key, falling back to req.ip only when the header is absent. Every limiter in the app - the global limiter, adminPathLimiter, adminWriteLimiter, strictApiWriteLimiter, userAddressLimiter and the adminAuthStrictLimiter that guards /admin/login, /admin/wallet-auth and /admin/wallet-auth-[redacted] - uses that key generator, and express-rate-limit's own protections are explicitly switched off via validate: { trustProxy: false, xForwardedForHeader: false }. On Vercel the request never reaches Express directly: app/api/[[...path]]/route.ts copies every inbound client header except host and content-length into a supertest request against the in-process app, so the attacker's own X-Forwarded-For is the value Express sees, and the app's `app.set('trust proxy', 1)` does not help because the limiter bypasses req.ip entirely.

Impact. Rate limiting on this application is decorative: an attacker who varies `X-Forwarded-For` per request gets a fresh bucket every time, so every limiter is bypassed at the cost of one header. That removes the brute-force ceiling on `/admin/login` when `ALLOW_ADMIN_PASSWORD` is enabled and on the admin challenge and wallet-auth routes guarded by `adminAuthStrictLimiter`, and it removes any throughput containment on the unauthenticated `POST /api/withdraw`, so a mass forced-withdrawal against every depositor runs unthrottled. Audit records are also inconsistent with the limiter as a side effect: `clientIp()` resolves `req.ip` under `trust proxy 1` while the limiter keys on the leftmost header value, so the address logged and the address throttled can differ.

Recommendation. Delete limiterKeyGen's header parsing and use req.ip with a correctly configured `trust proxy` hop count for the actual deployment, or take the platform-verified client IP (x-vercel-forwarded-for / the request's connection info) inside the Next.js route and pass it in a header the client cannot set, stripping any client-supplied x-forwarded-for at the entrypoint. Re-enable express-rate-limit's validators.

ANALYSIS Reasoned, not executed. Client-side reproduction in staging: send N+1 requests to POST /api/admin/login with a distinct X-Forwarded-For per request and assert the (N+1)th is 429.

```
const limiterKeyGen = (req) => { const xf = req.headers['x-forwarded-for']; if (typeof xf === 'string' && xf.trim()) { return xf.split(',')[0].trim(); } return req.ip || req.socket?.remoteAddress || 'unknown'; }  --- const rlValidate = { trustProxy: false, xForwardedForHeader: false }  --- route.ts: `for (const [k, v] of Object.entries(inHeaders)) { if (k === 'host' || k === 'content-length') continue; sreq = sreq.set(k, v); }`
```

PROTO-01 · **High**

Unauthenticated email-send endpoint with HTML injection lets anyone send arbitrary mail from [redacted]'s verified sender

LOCATION [redacted] dApp/src/app/api/send-email/route.ts:86-138

POST /api/send-email takes `{ template, to, values }` straight from the request body and calls [redacted] with the server-held SENDGRID_API_KEY and the server-held CLEARPOOL_EMAIL as the `from` and `reply_to` address. There is no authentication, no session check, no origin/referer check, no recipient allowlist and no rate limit — the only validation is `if (!template || !values || !to) return 400`. Unlike the sibling /api/[redacted] route, which at least attempts a referer check, and unlike /api/admin/* and /api/merkle, which call `authenticate()`, this route has no gate at all. There is also no middleware.ts in the project to add one. Separately, every field of `values` is interpolated raw into the HTML body (`${values.amount}`, `${values.currency}`, `${values.withdrawalDeadline}`, `${values.noticePeriod}`, `${values.requestTime}`, `${values.totalAvailableToWithdraw}`) and `values.vaultName` is interpolated raw into the Subject, with no escaping. The `default` branch echoes `JSON.stringify(values)` into the body. So the caller controls the recipient, the subject and — through any `values` field — arbitrary HTML including anchors and images inside a message that genuinely originates from [redacted]'s authenticated sending domain and passes SPF/DKIM/DMARC.

Impact. A third party can send unlimited, fully attacker-authored HTML email that is cryptographically attributable to [redacted], addressed to any inbox. The two canned subjects ('Withdrawal Request Created', 'Deposit Confirmed') are exactly the pretexts a wallet-drainer campaign against [redacted] depositors would use, and the injected HTML supplies the malicious link. Secondary effects: the protocol's [redacted] quota can be exhausted and its sending-domain reputation destroyed by a third party at no cost.

Recommendation. Gate the route: require the SIWE session cookie the admin routes already use, or bind it to the depositing wallet, and confirm server-side that `to` is the address that wallet registered rather than accepting it from the body. HTML-escape every `values.*` interpolation (or move to [redacted] dynamic templates so the body is not assembled from request data at all). Restrict `template` to the Templates enum and delete the `default` branch that reflects `JSON.stringify(values)`. Add per-IP and per-address rate limiting.

ANALYSIS Static: route.ts has no auth call. Reachability is provable from the file alone — POST handler body begins `const jsonData = await req.json(); const { template, to, values } = jsonData; if (!template || !values || !to) return 400` and proceeds directly to the [redacted] fetch. Shape of an abusive request (NOT executed — local-only rule): POST /api/send-email {"template":"withdrawalRequest","to":"<victim>","values":{"vaultName":"X-Pool","amount":"Verify your withdrawal","currency":"USDC"}}. Compare with src/app/api/merkle/route.ts:24 `await authenticate()` and src/app/api/admin/actions/route.ts:24 `await authenticate()`, which show the project's own gating pattern is simply absent here.

Credentials used by the route are read from process.env (SENDGRID_API_KEY, CLEARPOOL_EMAIL); neither value is present in the snapshot and neither is printed here.

PROTO-02 · **High**

Withdraw UI states a 'Minimum Received … after slippage (0.5%)' guarantee that the signed transaction does not encode, and the configured slippage is 0

LOCATION [redacted] dApp/src/features/port/components/vault/user/modals/WithdrawModal/T
45 vs [redacted] dApp/src/lib/sdk/services/port.ts:186-225

The withdraw modal renders a row labelled 'Minimum Received:' with the tooltip `Minimum ${vault?.baseToken.symbol} you'll receive after slippage (0.5%)`, whose value is computed purely in the browser as `receiveAmountValue * (1 - (vaultConfig?.withdrawalSlippage ?? 0))`. Two separate problems. (a) The figure is never sent anywhere. The transaction the user actually signs is `AtomicQueue.updateAtomicRequest(offer, want, deadline, offerAmount)`. Byte-level inspection of src/lib/sdk/contracts/abis/AtomicQueue.ts confirms this overload takes exactly four arguments — `contract ERC20 offer, contract ERC20 want, uint64 deadline, uint96 offerAmount` — and carries no price, rate, or minimum-out field of any kind. The user therefore authorises the queue to take `offerAmount` vault shares and be filled by the solver at whatever rate applies at solve time, with the deadline set to `now + (noticePeriod + withdrawalDeadline) * 24 * 3600` — up to 37 days out for the configured vaults. Nothing on-chain enforces the number the UI called a minimum. (b) `withdrawalSlippage` is the only input to that number and it is `0` in all five entries of `portVaultsConstants` (lines 51, 93, 131, 169, 207 of src/features/port/constants.ts). A grep for `withdrawalSlippage` across the whole of src returns exactly the type declaration, those five zeros, and the single display line — it reaches no transaction path. So `minimumReceived === receiveAmount` exactly, while the tooltip asserts a 0.5% buffer that is not applied anywhere.

Impact. The user is shown an explicit floor on what they will receive and signs a request that contains no floor. If the accountant exchange rate moves down during the notice period — including via the manager's own `updateExchangeRate` call, which this same dapp exposes at port.ts:236 — the request still fills, at the worse rate, with no revert and no user re-consent. The user's only accurate signal, the tooltip, states a protection level (0.5%) that differs from the value actually used (0%).

Recommendation. Either stop presenting an unenforced number as a guarantee — relabel to 'Estimated proceeds at the current rate; the final rate is set when your request is solved' and drop the tooltip's '(0.5%)' — or move the protection on-chain. If the AtomicQueue deployment has an overload carrying `atomicPrice`, use it and derive the price from the displayed minimum. Also delete or wire up `withdrawalSlippage`: a field that is 0 in every config and read by one display line is worse than no field.

PROOF OF CONCEPT — EXECUTED Executed greps and ABI parse. (1) `grep -rn withdrawalSlippage src` → 7 hits: 1 type decl, 5 x `withdrawalSlippage: 0`, 1 display-only use at TransactionDetails.tsx:22. Zero hits in any file under src/lib/sdk. (2) `grep -rn '0.5%' src --include=*.tsx` → 1 hit, the tooltip at TransactionDetails.tsx:40. (3) python3 substring extract of AtomicQueue.ts around 'updateAtomicRequest' → inputs are exactly [offer address, want address, deadline uint64, offerAmount uint96].

```
UI string: `<h1>Minimum Received:</h1>` with `<Tooltip tip={`Minimum ${vault?.baseToken.symbol} you'll receive after slippage (0.5%)`}>` and body `{minimumReceived} {vault?.baseToken.symbol}`, where `minimumReceived = formatNumber(receiveAmountValue * (1 - (vaultConfig?.withdrawalSlippage ?? 0)))` and every configured `withdrawalSlippage` is 0. Constructed payload: `simulateContract({ abi: AtomicQueue_ABI, address: atomicQueueAddress, functionName: 'updateAtomicRequest', args: [vaultAddress, wantToken, BigInt(deadline), offerAmount] })` — four args, no minimum, no price.
```

PROTO-03 · **High**

XRPL vault signing pages load wallet SDKs from third-party CDNs with no SRI and an unpinned major-version range

LOCATION [redacted] dApp/public/index.html:548-555,847,1394-1395 and public/admin.html:592,674-675,739,1257-1258; policy at [redacted] dApp/lib/http/csp.js:7

Both the depositor UI (public/index.html) and the admin UI (public/admin.html) are plain HTML shells that pull their wallet SDKs at runtime from unpkg.com, cdn.jsdelivr.net and esm.sh. No `integrity=` attribute appears anywhere in either file (grep for `integrity=` across all HTML/TSX returns zero hits). Three of the four loads are version-pinned but unhashed (`@gemwallet/api@3.8.0`, `xumm@1.8.0`, `@crossmarkio/sdk@0.4.0`), which stops an upstream author republishing but does not stop a CDN or registry compromise. The fourth is worse: `import('https://esm.sh/@walletconnect/sign-client@2')` and `import('https://esm.sh/@walletconnect/modal@2')` specify only a major version, so esm.sh resolves whatever 2.x exists at page load — the served bytes are not fixed by anything in this repository. These are exactly the modules that construct and dispatch the XRPL `Payment` transactions at index.html:1820-1895, and in admin.html the ones that authorise treasury operations. The CSP in lib/http/csp.js explicitly whitelists all three CDNs in `script-src` and additionally permits `'unsafe-inline'` and `'unsafe-eval'`, so it provides no containment for this class.

Impact. Whoever controls the served bytes for any of those four URLs controls the page that builds and signs XRPL payments. The most direct abuse is to rewrite the `Destination` field before the wallet is asked to sign — the deposit destination is already a mutable page variable (`depositAddress`, set from the API at index.html:2067) — so every subsequent deposit routes to an attacker-held XRPL account while the confirmation toast still reads 'Deposit sent!'. On admin.html the equivalent reach is over treasury and withdrawal signing. Exploitation requires compromising a third party (CDN, registry account, or the unpinned 2.x release channel), which is why this is High rather than Critical.

Recommendation. Pin every remote script to an exact version AND an `integrity="sha384-..."` hash with `crossorigin="anonymous"`. For the two esm.sh dynamic imports, either pin the exact patch version or — better, given these run in a signing context — vendor them into public/ and serve them same-origin, as the repo already does for xumm-browser.min.js in public/vendor/. Then tighten script-src to `'self'` and drop `'unsafe-eval'`.

PROOF OF CONCEPT — EXECUTED Executed: `grep -rn 'unpkg.com|jsdelivr|esm.sh' --include=*.html --include=*.tsx . | grep -v /public/vendor/` → 14 hits across index.html and admin.html. `grep -rn 'integrity=' --include=*.html --include=*.tsx .` → 0 hits.

```
index.html:1394-1395 `import('https://esm.sh/@walletconnect/sign-client@2'),
import('https://esm.sh/@walletconnect/modal@2')` — major-version-only specifier. csp.js:7 `script-src
'self' 'unsafe-inline' 'unsafe-eval' https://unpkg.com https://cdn.jsdelivr.net https://[redacted].app
https://esm.sh;`
```

PROTO-R03 · **Medium**

No CSP or security headers on the two Next.js apps

LOCATION	partial and mis-targeted - [redacted] dApp now has a full policy, but it is installed on the wrong surface; [redacted] dApp has nothing

`[redacted] dApp/next.config.ts` sets `images`, `sassOptions`, `eslint` and `typescript` and no `headers()`, and the project has no `vercel.json`, so it serves no security headers at all. `[redacted] dApp` does define a complete policy in `lib/http/csp.js` and applies it alongside `X-Content-Type-Options: nosniff`, `X-Frame-Options: DENY`, `Referrer-Policy`, `Permissions-Policy` and conditional HSTS at `lib/backend/express-app.js:207-236` -- but that middleware runs only for requests routed through `app/api/[[...path]]/route.ts`, i.e. `/api/*`. The browser-facing documents are static files in `public/` reached by `redirect()` from `app/page.tsx`, `app/admin/page.tsx` and `app/how-it-works/page.tsx`, served by Vercel's static layer, and neither `next.config.ts` nor `vercel.json` sets headers there. The policy therefore lands on JSON responses and is absent from the HTML it was written to protect. The policy text itself also permits `'unsafe-inline'` and `'unsafe-eval'` and allows script origins `unpkg`, `jsdelivr` and `esm.sh`.

Impact. Neither application's HTML is covered by a Content-Security-Policy, so any injection reaching a rendered page executes with no second line of defence, and no clickjacking, MIME-sniffing or referrer controls apply to the admin document either. On `[redacted] dApp` the gap is easy to miss operationally, because the headers do appear on `/api/*` responses -- exactly where a spot check would look. Even once the policy is moved onto the HTML surface, `'unsafe-inline'`, `'unsafe-eval'` and the three third-party script origins leave it with little containment value as written.

Recommendation. Move the policy onto the surface that serves HTML: set `headers()` in `[redacted] dApp/next.config.ts` (or `vercel.json`) so the CSP and companion headers from `lib/http/csp.js` also cover the static documents in `public/` reached by the `redirect()` stubs, not only requests routed through `app/api/[[...path]]/route.ts`. Add the same block to `[redacted] dApp/next.config.ts`, which sets no headers at all. Then tighten the policy itself: drop `'unsafe-inline'` and `'unsafe-eval'` by moving the inline scripts in `admin.html`/`index.html` to nonced or bundled files, and remove the `unpkg`/`jsdelivr`/`esm.sh` origins by vendoring those dependencies.

PROOF OF CONCEPT — EXECUTED Read [redacted] dApp/next.config.ts (and confirmed no vercel.json exists), [redacted] dApp/next.config.ts, vercel.json, lib/http/csp.js, lib/backend/express-app.js and the three app/*/page.tsx redirect stubs.

[redacted] dApp: next.config.ts contains images/sassOptions/eslint/typescript only - no headers() - and `ls vercel.json` fails. Zero security headers, unchanged. [redacted] dApp: lib/http/csp.js now defines a complete CSP and express-app.js:207-236 sets it together with nosniff, X-Frame-Options: DENY, Referrer-Policy, Permissions-Policy and conditional HSTS - but that middleware only runs for requests routed through app/api/[[...path]]/route.ts, i.e. /api/*. The browser-facing documents are static files in public/ reached by redirect() from app/page.tsx, app/admin/page.tsx and app/how-it-works/page.tsx, served by Vercel's static layer, and neither next.config.ts nor vercel.json sets headers. So the policy lands on JSON responses and is absent from the HTML. See novel PROTO-N05. The CSP text itself also permits 'unsafe-inline' and 'unsafe-eval' plus unpkg/jsdelivr/esm.sh.

PROTO-R04 · **Medium**

Dynamic HTML-injection sinks on non-static data

LOCATION [redacted] dApp half is now unreachable (no caller); SvelteKit half is unguarded and backend-fed

`[redacted] dApp/src/routes/(products)/borrowers/[slug]/+page.svelte:317` renders `{@html borrowerInfo?.description}`. `borrowerInfo` comes from `getDefaultBorrowerInfo(borrowerData?.info, ...)` (`lib/components/__v2/BorrowerOverview/index.ts:104-112`), and `info` is taken straight off the borrowers API and store -- backend-sourced data, not a repo literal -- so the markup rendered into the page is whatever that service returns, unsanitized. `[redacted]Banner/index.svelte:42` and `__v2/Tooltip/index.svelte:97` are prop-driven raw-HTML sinks whose current callers happen to pass literals, so they are safe by convention rather than by construction. The SvelteKit CSP is `frame-ancestors 'none'` only, with no `script-src`, so there is no mitigation behind the sink.

Impact. Anything able to influence a borrower's `info.description` -- the borrowers API, its data store, or an operator with write access to it -- gets script execution in the browser of every visitor to that borrower page, in the origin holding the user's wallet connection and session state. Exploitation requires control of that upstream content rather than an anonymous request, so the exposure tracks the trust placed in the borrowers backend and whoever can edit its records. The two prop-driven sinks are not exploitable today; they become so the first time a caller passes dynamic data, with nothing inside the components to prevent it.

Recommendation. Sanitize the one backend-fed sink: run `borrowerInfo?.description` through DOMPurify -- or render it as text -- at `[redacted] dApp/src/routes/(products)/borrowers/[slug]/+page.svelte:317`, since `info` arrives from the borrowers API rather than a repo literal. Add a `script-src` directive to the SvelteKit CSP, which is currently only `frame-ancestors 'none'` and so offers no second line of defence. `[redacted]Banner/index.svelte:42` and `__v2/Tooltip/index.svelte:97` are safe only by virtue of their present callers; sanitize inside those components too, so a future caller cannot introduce the sink by passing dynamic data.

PROOF OF CONCEPT — EXECUTED Enumerated every {@html}, dangerouslySetInnerHTML and innerHTML assignment across the three repos, then traced each one's data provenance and each sink's callers.

STILL PRESENT: [redacted] dApp/src/routes/(products)/borrowers/[slug]/+page.svelte:317 `{@html borrowerInfo?.description}` where borrowerInfo comes from getDefaultBorrowerInfo(borrowerData?.info, ...) (lib/components/__v2/BorrowerOverview/index.ts:104-112), and `info` is taken straight off the borrowers API/store - backend-sourced, not a repo literal. [redacted]Banner/index.svelte:42 and __v2/Tooltip/index.svelte:97 remain prop-driven raw-HTML sinks (current callers pass literals). The SvelteKit CSP is only frame-ancestors 'none', so there is no script-src mitigation. REFUTED for [redacted] dApp: SideBySideText's `{icon && <div dangerouslySetInnerHTML={{__html: icon}} />}` is dead code - no caller anywhere in [redacted] dApp/src passes an `icon` prop (verified by grepping every <SideBySideText usage). NEW BUT CLEAN: [redacted] dApp's admin.html and index.html build HTML by concatenation, but every interpolation goes through escHtml() (admin.html:1646) and the single href goes through encodeURIComponent (txLink, admin.html:1679); no unescaped sink found.

PROTO-R08 · **Medium**

security.txt absent; SIWE domain binding omitted; build-time type/lint checks disabled

LOCATION	mixed - one sub-item partially fixed, one unchanged, one materially worse than rated

Three separate hardening gaps sit under this heading. (a) No `/.well-known/security.txt` exists in any of the three repositories -- a `find` for it, excluding `node_modules`, returns nothing -- so there is no published channel for vulnerability reports. (b) `[redacted] dApp/src/shared/utils/auth.ts:17-27` calls `validateSiweMessage({ address, message, nonce, time })` with no `domain` and no `scheme`, so the domain the wallet displayed to the user at signing time is never checked against the domain performing verification; the nonce that would otherwise compensate is not fresh, because `/api/admin/nonce/route.ts:10-14` re-serves the same value out of the `x-nonce` cookie for the whole `MAX_AGE` (default 60*60*24*30, thirty days) and no code path ever deletes it. (c) `[redacted] dApp/next.config.ts` sets both `eslint.ignoreDuringBuilds = true` and `typescript.ignoreBuildErrors = true`, with the '!! WARN !!' comment intact; `[redacted] dApp/next.config.ts` sets `typescript.ignoreBuildErrors = false` but still ignores eslint.

Impact. The SIWE item is the material one: with no domain binding and a nonce that is neither fresh nor single-use, an admin signature captured on any site the admin is lured to remains a valid bearer credential for up to thirty days and is accepted from any origin -- and the attacker can fetch their own nonce cookie to build the phishing page around. The missing `security.txt` costs the project reports it would otherwise receive through a defined channel, and disabling build-time type and lint checking lets whole classes of defect reach production without failing a build. None of the three requires an attacker to breach anything on their own; together they lower the bar for everything else.

Recommendation. Publish `/.well-known/security.txt` with a contact and an expiry in all three repos. Pass `domain` and `scheme` into `validateSiweMessage` at `[redacted] dApp/src/shared/utils/auth.ts:17-27`, and make the nonce single-use: `/api/admin/nonce/route.ts:10-14` re-serves the same cookie value for the full `MAX_AGE` and nothing ever deletes it, so generate a fresh nonce per challenge, store it server-side against the address, and delete it on first verification -- otherwise one captured admin signature stays a 30-day bearer credential accepted from any origin. Set `eslint.ignoreDuringBuilds = false` and `typescript.ignoreBuildErrors = false` in `[redacted] dApp/next.config.ts` (and re-enable eslint in `[redacted] dApp/next.config.ts`), then fix the resulting errors rather than shipping with the checks disabled.

PROOF OF CONCEPT — EXECUTED Checked each of the three sub-items separately: searched all three repos for security.txt; read [redacted] dApp/src/shared/utils/auth.ts and src/app/api/admin/nonce/route.ts; read both next.config.ts files.

(a) security.txt - STILL ABSENT: `find . -name security.txt` (node_modules excluded) returns nothing in any of the three repos. (b) SIWE domain binding - STILL OMITTED, and the prior INFO rating rested on a mitigation that does not exist. auth.ts:17-27 calls validateSiweMessage({ address, message, nonce, time }) with no `domain` and no `scheme`, so the domain the wallet displayed is never checked; the prior note said this was 'mitigated by nonce + admin-address + signature', but /api/admin/nonce/route.ts:10-14 returns the SAME nonce out of the x-nonce cookie for MAX_AGE (default 60*60*24*30 = 30 days) and no code path ever deletes it, so the nonce is neither fresh nor single-use, and an attacker can obtain their own nonce cookie to phish against. That makes one captured admin signature a 30-day bearer credential accepted from any domain. Reclassified from Info to Medium and reported under sweep row PAT-0008. (c) Build checks - PARTIALLY FIXED: [redacted] dApp/next.config.ts now sets `typescript: { ignoreBuildErrors: false }` (eslint still ignored); [redacted] dApp/next.config.ts still sets both eslint.ignoreDuringBuilds=true and typescript.ignoreBuildErrors=true, with the '!! WARN !!' comment intact.

PROTO-N03 · Medium

[redacted] dApp treats any mined transaction as a success: receipt.status is never read, and the Safe path ignores isSuccessful

LOCATION [redacted] dApp/src/lib/sdk/services/transaction.ts:87-105; [redacted] dApp/src/lib/sdk/services/safe.ts:249-257

TransactionsService.waitForTransactionReceipt awaits viem's waitForTransactionReceipt and returns the receipt. viem resolves - it does not throw - when a transaction is mined with status 'reverted'. No file in [redacted] dApp/src ever reads receipt.status. The Safe branch has the same shape one level up: waitForTransactionExecution polls until transaction.isExecuted and returns, without consulting isSuccessful, so an executed-and-reverted Safe transaction is also treated as a success. Every port write funnels through these two functions, so react-query's mutation resolves, isSuccess flips true, the modal renders 'Deposit successful!' / 'Withdrawal Request Submitted!', the query cache is invalidated as though state had changed, and EmailService fires a deposit or withdrawal notification to the borrower. simulateContract and estimateContractGas run first, so this is not reachable for a deterministic revert - it is reachable exactly when state changes between simulation and inclusion, which for this product means the manager pausing the Teller, raising or lowering the NAV, or another depositor consuming the remaining deposit cap in the intervening block.

Impact. A user is told their deposit or withdrawal request succeeded when it reverted. Their funds are untouched but their mental model is wrong at the worst moment: they believe a 30-day notice clock has started when no request exists on chain, or that they hold shares they do not hold. The borrower simultaneously receives an email announcing a deposit or a withdrawal request that never happened, which is the input to that borrower's liquidity planning.

Recommendation. Read the status: `if (receipt.status !== 'success') throw new TransactionRevertedError(hash)` in waitForTransactionReceipt, and `if (!transaction.isSuccessful) throw ...` in waitForTransactionExecution. Move the email send behind the same assertion.

ANALYSIS Reasoned, not executed. Deterministic reproduction: fork the vault chain, submit a deposit, and pause the Teller in the same block before inclusion; assert the modal shows an error rather than 'Deposit successful!'.

transaction.ts:96-99 `const receipt = await client.waitForTransactionReceipt({ hash })` then returns it; grep for receipt.status / status === 'success' across [redacted] dApp/src returns nothing. safe.ts:250-255 `while (true) { const transaction = await this.getSafeTransactionBySafeTxHash(safeTxHash); if (transaction.isExecuted) return transaction; ... }`. SIBLING: [redacted] dApp/src/lib/components/WalletProvider/utils.ts:293-303 uses ethers v5 tx.wait(1) inside a 180 s timeout with normalised TRANSACTION_REVERTED handling.

PROTO-N04 · **Medium**

WalletConnect deposit discards the wallet's response and announces 'signed & submitted' unconditionally, on a hardcoded xrpl:1 chain

LOCATION [redacted] dApp/public/index.html:1875-1899 and 1421-1427

The WalletConnect/Girin deposit branch awaits wcSignClient.request({ method: 'xrpl_signTransaction', ... }) without assigning the result, then unconditionally calls showToast('Deposit signed & submitted via WalletConnect!'), clears the amount field, zeroes the USD readout and schedules a balance refresh. Nothing checks whether the wallet signed or rejected, nothing reads a hash or an engine result, and the file contains no submit step for a signed blob - so on a wallet that implements xrpl_signTransaction as sign-only (which is what the method name means; submission is an optional behaviour, not a guarantee) the Payment is never broadcast and the user is told it was. Compounding it, both the session namespace and the request pin the CAIP chain to the literal 'xrpl:1' (XRPL mainnet), while the vault's network is a server-side XRPL_NETWORK setting whose shipped default is 'testnet', and index.html never reads or displays the network - so there is nothing on the page for a user or a reviewer to compare the signed chain against.

Impact. A WalletConnect user can believe they deposited when no Payment reached the ledger, and will then see a zero balance and open a support ticket - or worse, deposit again. If the deployment is ever on testnet while the WC request says mainnet, the wallet is asked to sign a mainnet Payment to a destination the vault controls only on testnet: real XRP to an address with no operator behind it.

Recommendation. Assign and inspect the response: require a signed result (and, if the wallet returns a blob rather than submitting, submit it via the XRPL client and check engine_result) before showing success; treat a rejection as an error toast. Derive the CAIP chain from the same XRPL_NETWORK the backend reports through /api/vault/info instead of hardcoding xrpl:1, and show the active network on the deposit card.

ANALYSIS Reasoned, not executed (LOCAL ONLY - no wallet, no RPC, no live page).

`await wcSignClient.request({ topic: wcSession.topic, chainId: 'xrpl:1', request: { method: 'xrpl_signTransaction', params: { tx_json: { TransactionType: 'Payment', ... } } } });` showToast('Deposit signed & submitted via WalletConnect!');` - the return value is never bound. Namespace: `xrpl: { chains: ['xrpl:1'], methods: ['xrpl_signTransaction','xrpl_signMessage'], ... }`. XRPL_NETWORK appears in admin.html but never in index.html.

PROTO-N05 · **Medium**

The XRPL vault's CSP and security headers are set only on /api responses; the HTML documents they were written

for are static files that never pass through the middleware

LOCATION [redacted] dApp/lib/http/csp.js; [redacted] dApp/lib/backend/express-app.js:207-236; [redacted] dApp/app/page.tsx; [redacted] dApp/next.config.ts; [redacted] dApp/vercel.json

Since the prior review this app has grown a genuine, carefully written policy: a full Content-Security-Policy string with frame-ancestors 'none' and an explicit allow-list for the [redacted]/WalletConnect frames, plus nosniff, X-Frame-Options: DENY, Referrer-Policy, Permissions-Policy and conditional HSTS. All of it is installed as Express middleware. But on Vercel the Express app is mounted only at app/api/[[...path]]/route.ts, so it handles /api/* and nothing else. The pages are static HTML in public/ - app/page.tsx, app/admin/page.tsx and app/how-it-works/page.tsx each just redirect() to /index.html, /admin.html and /how-it-works.html, which Vercel serves from its static layer. next.config.ts defines no headers() block and vercel.json contains only functions and crons. The result is that the CSP is delivered on JSON API responses, where a browser has nothing to apply it to, and is absent on the three documents that load wallet SDKs and hold the admin Bearer token.

Impact. The admin console at /admin.html - which pulls @gemwallet/api from unpkg over a plain script tag with no SRI and dynamically imports @walletconnect/* from esm.sh - runs with no CSP, no frame-ancestors, no nosniff and no Referrer-Policy. A compromised or hijacked CDN response, or any injected script, executes unconstrained in the page holding the admin session token, and the page can be framed. The prior finding PROTO-03 therefore remains open for this app despite a fix having been written, because it was attached to the wrong surface.

Recommendation. Move the header set into next.config.ts headers() (source '/(.*)') or a vercel.json headers block so it applies to static documents, keeping lib/http/csp.js as the single source of the policy string. While doing so, drop 'unsafe-eval' and tighten 'unsafe-inline' with hashes or a nonce, and pin the unpkg script with an integrity attribute.

> ANALYSIS Reasoned, not executed. Client-side verification: `curl -I https://vaults.[redacted]/index.html` and compare the header set against `curl -I https://vaults.[redacted]/api/health`.
>
> app/page.tsx: `export default function HomePage() { redirect('/index.html'); }`. express-app.js:236 `res.setHeader('Content-Security-Policy', CONTENT_SECURITY_POLICY)` inside `app.use((req,res,next)=> {...})`, reached only via app/api/[[...path]]/route.ts. next.config.ts contains eslint/typescript/serverExternalPackages only. vercel.json contains functions + crons only.

PROTO-N06 · **Medium**

The withdrawal notice period and slippage are front-end-only constants keyed by name+chain+address; a miss produces an already-expired on-chain request while the UI reports success

LOCATION [redacted] dApp/src/features/port/constants.ts:38-233;
[redacted] dApp/src/state/client/port/usePortUserWithdraw.ts:47;
[redacted] dApp/src/lib/sdk/services/strategy.ts:264-266;
[redacted] dApp/src/state/server/port/useGetPortVaults.ts:28

Everything the withdrawal screen tells the user about timing and price protection comes from a hardcoded table of five vaults in the front-end bundle, not from any contract: noticePeriod, withdrawalDeadline and withdrawalSlippage. Two consequences. (1) Representation versus enforcement: withdrawalSlippage is 0

for all five entries while the tooltip beside the number hardcodes the string '(0.5%)', so the 'Minimum Received' figure is the unreduced estimate and the on-chain AtomicRequest carries no price field at all - three layers claiming a protection that exists at none of them. (2) Reachability: the deadline actually signed is derived from the same table, `((noticePeriod ?? 0) + (withdrawalDeadline ?? 0)) * 24` hours, and because 0 is then passed EXPLICITLY into withdrawERC20 the deadlineHours = 888 default parameter is bypassed, so a table miss yields deadline = now. A miss is reachable: useGetVault() resolves through useGetPortVaults(), which filters only on hideOnProd and not on hasVaultConstants (unlike useGetPortAllVaults), and the vault is chosen from an unvalidated ?address= query parameter, so any indexed Port vault not in the five hardcoded entries - a newly deployed one, or one whose indexed `name` no longer matches the key - renders a full withdraw modal. The same table feeds strategy.ts:264, so the strategist's 'Revert Funds' from a [redacted] strategy vault has the identical failure.

Impact. The user signs an ERC20 allowance over their shares to the AtomicQueue and an AtomicRequest that can never be solved (AtomicQueue__RequestDeadlineExceeded), sees 'Withdrawal Request Submitted!', and receives an email stating 'Notice Period: 0 days'. Their capital is neither withdrawn nor visibly stuck; they will discover it only when the request quietly expires. On the strategist path the vault's own funds stay parked in the strategy while the manager console reports the reversion succeeded.

Recommendation. Read noticePeriod/withdrawalDeadline from chain (or from the indexer alongside the vault) rather than a bundled table; refuse to build a withdrawal when the configuration is missing instead of defaulting to zero; apply the hasVaultConstants filter in useGetPortVaults as it is already applied in useGetPortAllVaults; and either implement the 0.5% slippage claim or delete it from the tooltip.

> ANALYSIS Reasoned, not executed. Reproduction: deploy or index a Port vault absent from portVaultsConstants, open /vault?address=<it>&chainId=<its chain>, submit a withdrawal, and read back AtomicQueue.getUserAtomicRequest - deadline equals the signing timestamp.

```
constants.ts: `withdrawalSlippage: 0` appears 5 times (every entry). TransactionDetails.tsx:40 tooltip literal "after slippage (0.5%)"; :35 `minimumReceivedRaw = receiveAmountValue * (1 - (vaultConfig?.withdrawalSlippage ?? 0))`. usePortUserWithdraw.ts:47 `((vaultConfig?.noticePeriod ?? 0) + (vaultConfig?.withdrawalDeadline ?? 0)) * 24`. port.ts:214-216 `const deadline = isCancelled ? now+10 : now + deadlineHours*3600` with `deadlineHours = 888` as a default parameter only. useGetPortVaults.ts:28 filters on hideOnProd only; useGetPortAllVaults.ts:26 filters on hasVaultConstants.
```

PROTO-N07 · **Medium**

Every [redacted] dApp value-moving call hardcodes its slippage bound to zero, on both the depositor and the solver side

LOCATION

```
[redacted] dApp/src/lib/sdk/services/port.ts:172 (deposit _minimumMint 0), :425 (redeemSolve minimumAssetsOut 0), :368 (p2pSolve minOfferReceived 0, maxOfferReceived maxUint256)
```

The Teller, the AtomicSolver and the AtomicQueue all expose a bound, and the app passes the degenerate value to every one of them. Teller.deposit is called with _minimumMint = 0n, so the 'You Receive: N portUSD' figure on the deposit screen - which is computed client-side from a currentNav read out of the [redacted] indexer, not from the accountant at signing time - is backed by nothing; the vault may mint one wei of shares and the transaction still succeeds. AtomicSolverV3.redeemSolve is called with minimumAssetsOut = 0 and p2pSolve with minOfferReceived = 0 and maxOfferReceived = 2^256-1. The contracts define TellerWithMultiAssetSupport__MinimumMintNotMet and

TellerWithMultiAssetSupport__MinimumAssetsNotMet errors, so the protection is designed in and declined at the call site.

Impact. A depositor has no protection against an exchange-rate move between quote and inclusion, and the manager can move that rate at will (Accountant.updateExchangeRate is a one-click 'Update NAV' in the same app, and is also issued silently as part of 'Claim'). A NAV update landing in the block before a deposit costs the depositor shares with no revert to protect them. On the solver side the manager can pay out assets and receive nothing back for them. Because the quote is read from an indexer rather than from the accountant, ordinary indexer lag - not an attack - is enough to make the displayed 'You Receive' wrong.

Recommendation. Compute the minimum from the accountant's live getRate at signing time, apply an explicit tolerance the user can see and change, and pass it as _minimumMint / minimumAssetsOut / minOfferReceived. Bound maxOfferReceived by the allowance rather than by maxUint256. Show the tolerance on the screen next to 'You Receive'.

> ANALYSIS Reasoned, not executed. Reproduction: on a fork, submit a deposit, then in the same block call Accountant.updateExchangeRate to halve the rate; the deposit still succeeds and mints half the advertised shares.

port.ts:170-174 `args: [asset, amount, 0n], functionName: 'deposit'`; :419-430 `functionName: 'redeemSolve', args: [..., 0, approvalCap, tellerAddress]` with the inline comment `// uint256 minimumAssetsOut`; :363-372 `functionName: 'p2pSolve', args: [..., 0, maxUint256]` with `// uint256 minOfferReceived`. ABI: deposit(ERC20 _depositAsset, uint256 _depositAmount, uint256 _minimumMint).

PROTO-N08 · **Medium**

The SvelteKit app substitutes MaxUint256 for the user's typed amount whenever it equals the maximum, granting unlimited allowances behind fixed-amount screens

LOCATION

[redacted] dApp/src/lib/layouts/lending/pool-details/__components/modal/Deposit.svelte:79-90; .../Redeem.svelte:75-85; src/routes/(products)/lending/[redacted]/__components/modals/RepayPool.svelte:47,88-95; RepayBulletPool.svelte:54,90-95; RepayLender.svelte:73-98; RepayInterest.svelte:107-114; src/routes/(products)/bridge/+page.svelte:255; bridge-v1/+page.svelte:224

Two related substitutions run through this app. First, the [redacted] repay modals and both bridge pages approve constants.MaxUint256 unconditionally - RepayPool.svelte carries the comment '//approving maxUint' directly above `approveTransfer(pool.asset.address, pool.id, constants.MaxUint256, 0)` - while the screen shows a finite 'Total Due' and a button reading 'Repay Pool'. Second, the vault Deposit and Redeem modals silently swap the typed amount for MaxUint256 when it equals the maximum, and then use that same sentinel for BOTH the approval and the value-moving call. In Deposit that means a 100% chip yields approve(token, pool, 2^256-1) followed by supply(pool, 2^256-1); because availableToDeposit is min(wallet balance, depositCap - currentSize), a user whose deposit is limited by the CAP is shown the capped figure and signs 'everything', so if the pool reads the sentinel as 'take my whole balance' the amount authorised exceeds the amount displayed. In Redeem the 'is this the maximum?' test is performed on raw wei integers pushed through Number() - values far above 2^53 - so the decision to escalate to 'redeem everything' is made on rounded doubles.

Impact. Standing unlimited allowances from borrower and lender wallets to pool and bridge contracts, created without disclosure, are the single most-exploited artefact in DeFi front-ends: any later compromise of the approved contract, or of its upgrade authority, drains the wallet's entire balance of that asset rather than the amount that was actually being transacted. The screen never shows the allowance, so there is nothing for the user to decline.

Recommendation. Approve the exact amount required (the app already does this correctly on the Stake and PlaceBid paths, so the pattern exists in-repo). Where a 'max' semantic is genuinely wanted, show it as such on screen ('Approve unlimited' / 'Deposit entire balance') and make it an explicit user choice. Replace the Number()-based equality in Redeem with a BigNumber comparison, as Deposit's checkEqualValue already does.

> ANALYSIS Reasoned, not executed (the @[redacted]-finance/* SDKs are private packages and node_modules is absent for this app, so the sentinel's contract-side meaning is stated as requiring confirmation).

Deposit.svelte:77-90 `if (checkEqualValue(value, availableToDeposit, pool.decimals)) { value = constants.MaxUint256.toString() }` then requiredAllowance = MaxUint256 and `depositFn(_sdk, actions, pool, value)` -> `_sdk.supply(pool.id, amount)`. RepayPool.svelte:45-47 `//approving maxUint` + `.approveTransfer(pool.asset.address, pool.id, constants.MaxUint256, 0)`. COUNTER-EXAMPLE in the same repo: Stake.svelte:131 `_sdk.approve(amount)` and PlaceBid.svelte:99 `_sdk.approve(pool.asset.address, _sdk.addresses.auction, amount)`.

PROTO-N09 · **Medium**

'Pause Deposits & Withdrawal Processing' pauses only the Teller, while 'Resume' unpauses two contracts from one click

LOCATION [redacted] dApp/src/state/client/port/useUpdateVaultStatus.ts:35-59; src/state/client/port/useIsVaultPaused.ts:9; src/features/port/components/vault/manager/modals/ManageStatusModal.tsx:37-43

The vault presents one status - a pill reading 'Active' or 'Paused' and one button - over two independently pausable contracts. The mutation is asymmetric. On 'pause', when neither contract is currently paused, control falls through to `return sdk.port.updateTellerContract(vault.teller, 'pause')`: a single transaction that stops deposits and leaves the Accountant running, so NAV updates and fee accrual continue on a vault the console now labels 'Paused'. On 'resume', in the mirror-image state, the mutation issues `Promise.all([updateTellerContract(unpause), updateAccountantContract(unpause)])` - two wallet prompts fired concurrently from one button press, with no ordering, no atomicity and no rollback if one is rejected. And because useIsVaultPaused computes `isPaused = tellerIsPaused || accountantState?.isPaused`, the pill flips to 'Active' as soon as either transaction lands.

Impact. A manager who pauses in an incident believes the vault is frozen while the pricing contract is still live and still writable. A manager who pauses and later resumes also unpauses an Accountant that may have been paused deliberately and separately beforehand - the resume path does not restore the prior state, it forces both to running. The concurrent double prompt is also a signing hazard: two simultaneous requests from one click is exactly the shape a user is trained to approve without reading.

Recommendation. Make the control state-symmetric: pause both contracts, or expose them as two controls with two status indicators. Issue the transactions sequentially and await each, and derive the

displayed status from both contracts explicitly ('Deposits paused / Accounting live') rather than from a boolean OR.

> ANALYSIS Reasoned, not executed. Reproduction on a fork: pause the Accountant directly, then use the console's Pause and Resume, and read isPaused on both contracts - the Accountant ends unpaused.

useUpdateVaultStatus.ts:35-44 pause branch ends `return sdk.port.updateTellerContract(vault.teller, 'pause')`; :55-58 resume branch `return await Promise.all([sdk.port.updateTellerContract(vault.teller,'unpause'), sdk.port.updateAccountantContract(vault.accountant,'unpause')])`. useIsVaultPaused.ts:9 `isPaused: tellerIsPaused || accountantState?.isPaused`. Button label from ManageStatusModal.tsx:43.

PROTO-N10 · **Medium**

Safe support map omits Flare and Base, so Safe users on the live Flare vault get a silently hung transaction flow

LOCATION [redacted] dApp/src/lib/sdk/constants/safe.ts:3-8, :22-27; src/lib/sdk/services/safe.ts:26-46; src/lib/sdk/services/transaction.ts:88-91

SupportedNetworkIds declares five chains (1, 11155111, 98866, 8453, 14) but SAFE_TRANSACTION_SERVICE_URL_CHAIN_MAP and SAFE_TRANSACTIONS_URL_CHAIN_MAP cover only three (1, 11155111, 98866). portVaultsConstants nevertheless contains a live X-Pool vault on FLARE_MAINNET (14), and Context.isMainnet lists 8453. On those chains SafeApiKit is constructed with `txServiceUrl: undefined`; isSafeWallet() wraps its lookup in a bare `catch { return false }`, so the failure is converted into the assertion 'this is not a Safe wallet'. transaction.waitForTransactionReceipt then takes the EOA branch and calls client.waitForTransactionReceipt({ hash: safeTxHash }) - waiting on a Safe transaction hash that will never appear on chain.

Impact. For a Safe-connected manager or depositor on the Flare vault, every write hangs forever: the mutation never settles, isPending stays true, and the modal's close handler is guarded by `if (!isPending)` so the user cannot even dismiss it. The ButtonWithSafeStatus 'pending in Safe' badge and its 'open in Safe' link, which are the intended escape hatch, are driven by the same missing map and resolve against an undefined base URL. Vault managers are very likely to be multisigs, which is precisely the population this breaks.

Recommendation. Add entries for chains 14 and 8453 to both Safe maps, or gate Safe support explicitly and tell the user when the connected chain is unsupported. Replace the blanket `catch { return false }` in isSafeWallet with a distinction between 'not a Safe' and 'could not determine', and fail loudly on the latter. Give waitForTransactionReceipt a timeout.

> ANALYSIS Reasoned, not executed. Verification: connect a Safe on chain 14 to /vault?address=[redacted]&chainId=14 and attempt a deposit.

constants/safe.ts SAFE_TRANSACTION_SERVICE_URL_CHAIN_MAP keys: ETHEREUM_MAINNET, ETHEREUM_SEPOLIA, PLUME_MAINNET. types/common.ts SupportedNetworkIds also declares BASE_MAINNET = 8453 and FLARE_MAINNET = 14. constants.ts registers an X-Pool vault at networkId FLARE_MAINNET, address [redacted]. safe.ts:38-45 `try { const safeInfo = await this.apiKit.getSafeInfo(walletAddress); return safeInfo.owners.length > 0 } catch { return false }`.

PROTO-N11 · **Medium**

'Claim' on accrued fees also writes the vault's exchange rate and approves the accountant for the vault's entire fee-asset balance

LOCATION [redacted] dApp/src/lib/sdk/services/port.ts:435-525; src/state/client/port/usePortClaimFee.ts:45-53; src/features/port/components/vault/manager/AccruedFees.tsx:80

One button labelled 'Claim', beside a single fee figure, produces up to three transactions. The first is BoringVault.manage(feeAsset, approve(accountant, vaultFeeBalance), 0) - the vault approves the accountant for its WHOLE balance of the fee asset, not for the fee that is owed (on mainnet USDT this is preceded by a further approve-to-zero manage call, so four in total). The second, whenever usePortClaimFee decides needUpdateNav, is Accountant.updateExchangeRate(newExchangeRate) - a write to the number that prices every depositor's position, with the new rate computed client-side as exchangeRate - feesOwedInBase/totalSharesLastUpdate. The third is the actual claimFees. The screen shows the fee amount and the word 'Claim'; it does not show the approval amount, does not say a NAV update is part of the operation, and does not show the old and new exchange rate - even though the same app has a dedicated 'Update NAV' modal that does show exactly that before asking for a signature.

Impact. A manager signing 'Claim' unknowingly authorises a rate change that every depositor's position is marked to, and unknowingly grants an allowance far larger than the claim. If the intermediate transactions succeed and a later one fails, the vault is left with a standing full-balance allowance to the accountant and a NAV that was moved for a claim that never completed.

Recommendation. Show the operation as what it is: list the steps, the approval amount and the before/after exchange rate in the modal, and require confirmation of the NAV change with the same disclosure the dedicated Update NAV modal already provides. Approve the fee amount rather than the balance.

> ANALYSIS Reasoned, not executed.
>
> usePortClaimFee.ts:45-53 `const newExchangeRate = ...; const needUpdateNav = !isPayFiVault && newExchangeRate !== accountantState.exchangeRate; await sdk.port.claimFees(vault.address, vault.accountant, feeAsset, newExchangeRate, needUpdateNav)`. port.ts:465-476 `runFeeAssetApprovalManage(vaultFeeBalance)` where vaultFeeBalance = the vault's full balance of the fee asset. port.ts:493-500 `functionName: 'updateExchangeRate'`. AccruedFees.tsx:80 the button's only label is 'Claim'.

PROTO-04 · **Medium**

[redacted] dApp reports a reverted transaction as success — success UI shown and a confirmation email dispatched

LOCATION [redacted] dApp/src/lib/sdk/services/transaction.ts:87-103; consumers at src/state/client/port/usePortUserDeposit.ts:37-79 and usePortUserWithdraw.ts:33-75

`TransactionsService.waitForTransactionReceipt` awaits viem's `client.waitForTransactionReceipt({ hash })` and returns the receipt unconditionally. viem resolves that promise for a mined transaction whether it succeeded or reverted — a revert is reported as `receipt.status === 'reverted'`, not as a rejection. A grep for `receipt.status`, `'reverted'`, `status === 1` and `status !== 1` across all three dapps returns zero hits, so nothing in [redacted] dApp ever inspects that field. Every one of the 16 call sites in port.ts /

token.ts / strategy.ts / merkle.ts therefore treats 'mined' as 'succeeded'. The two user-facing hooks build directly on that: `mutationFn` awaits `sdk.port.depositERC20(...)` / `withdrawERC20(...)` and returns normally, so react-query sets `isSuccess: true`, `ActionButtons.tsx:42` flips the button to 'Done', and `onSuccess` fires `EmailService.sendDepositNotification` / `sendWithdrawalRequestNotification`. This is a divergence from the sibling dapp: [redacted] dApp routes everything through ethers v5 `tx.wait()` (src/lib/components/WalletProvider/utils.ts:256-266), which does throw CALL_EXCEPTION on revert, so that codebase is not affected.

Impact. A user whose deposit reverts on-chain — Teller paused (`getTellerPauseState` is queried but is not a precondition of the write), deposit cap hit, asset not accepted, allowance race — is shown a completed deposit and receives an email headed 'Deposit Confirmed' stating an amount that never entered the vault. The same holds for a reverted withdrawal request. Beyond the direct confusion, react-query invalidates and refetches on this path, so the UI reconciles to the true balance moments later, contradicting the success state and the email.

Recommendation. In `waitForTransactionReceipt`, after resolving, `if (receipt.status === 'reverted') throw new TransactionRevertedError(hash)`. This is a single-point fix that corrects all 16 call sites at once. Do the same for the Safe branch, which returns `this.sdk.safe.waitForTransactionExecution(hash)` without a status check either.

> PROOF OF CONCEPT — EXECUTED Executed: `grep -rn "receipt\.status|status === 'reverted'|status === 1|status !== 1|'reverted'" --include=*.ts --include=*.tsx --include=*.svelte .` across all three dapps → zero results. `grep -rn 'waitForTransactionReceipt|\.wait\(' ...` → 16 call sites in [redacted] dApp, all discarding or returning the receipt unexamined; 2 in [redacted] dApp, both ethers `tx.wait()`.
>
> ```
> transaction.ts:94-102 `const receipt = await client.waitForTransactionReceipt({ hash }); if (waitSubsquidIsSynced && this.maxSubsquidSyncRetries) { await this.waitUntilSubsquidSynced(Number(receipt.blockNumber), chainId) } return receipt` — no status branch. usePortUserDeposit.ts:61-79 `onSuccess: async (data) => { ...; await EmailService.sendDepositNotification({...}); onSuccess?.() }`.
> ```

PROTO-05 · **Medium**

[redacted] proxy attaches the master token on mere presence of an x-[redacted]-token header, behind a substring referer check

LOCATION [redacted] dApp/src/app/api/[redacted]/route.ts:9-40,89-97

Two defects compound. First, `isServerRequest(req)` is `return !!req.headers.get('x-[redacted]-token')` — presence only, no comparison to the expected value. `validateRequest` does compare properly (`=== process.env.OZEAN_ACCESS_TOKEN`) but returns early on success and otherwise falls through to the referer checks. So a request carrying `x-[redacted]-token: anything-at-all` plus an acceptable referer passes `validateRequest` via the referer branch, and then at line 94 `isServerRequest(req)` is true, causing the proxy to forward `x-[redacted]-token: process.env.SUBSQUID_MASTER_TOKEN` upstream. A browser-originated request thereby obtains master-token-privileged access to the indexer. Second, the referer check is `referer.includes('port-[redacted].vercel.app')` / `.includes('[redacted].[redacted]')` / `.includes('vaults.[redacted]')` / `.includes('-[redacted].vercel.app')` — substring matching on an attacker-suppliable header, so

`https://attacker.example/[redacted].[redacted]` satisfies it, as does any hostname ending in `-[redacted].vercel.app` (the preview branch, when VERCEL_ENV is preview).

Impact. Whatever the [redacted] master token unlocks beyond the anonymous read path — additional queries, higher rate limits, mutations, or private fields — becomes reachable from an ordinary browser or curl, without possessing OZEAN_ACCESS_TOKEN. The referer weakness independently means the origin gate is decorative. I could not determine what the master token actually authorises upstream without calling the live indexer, which is out of scope; the privilege escalation from 'anonymous' to 'master' is nevertheless established from this file alone.

Recommendation. Make `isServerRequest` compare the header to OZEAN_ACCESS_TOKEN with a constant-time comparison and use that single result for both the auth decision and the master-token decision — never presence. Replace the referer substring tests with exact origin equality against an allowlist, read from the `Origin` header, and treat a missing/mismatched origin as unauthorised.

> ANALYSIS Trace from source. `isServerRequest` (line 9-11) checks presence only. `validateRequest` (13-40) returns early only when the token matches exactly; otherwise it evaluates referer substrings and returns. Line 94: `'x-[redacted]-token': isServerRequest(req) ? process.env.SUBSQUID_MASTER_TOKEN ?? '' : ''`. A request with header `x-[redacted]-token: x` and `Referer: https://evil.example/[redacted].[redacted]` satisfies validateRequest via line 31 and then satisfies isServerRequest at line 94. Not executed — sending it would require contacting the live deployment.

```
SUBSQUID_MASTER_TOKEN and OZEAN_ACCESS_TOKEN are read from process.env and are not present in the snapshot; no value is reproduced here.
```

PROTO-06 · **Medium**

[redacted] dApp silently signs MaxUint256 approvals — and a MaxUint256 deposit amount on MAX — behind UI that shows a specific figure

LOCATION

`[redacted] dApp/src/lib/layouts/lending/pool-details/__components/modal/Deposit.svelte:76-95,236 and src/routes/(products)/lending/[redacted]/__components/modals/RepayPool.svelte:45-48,86-93,159,177`

Two distinct instances of the same pattern. (a) Credit-vault deposit. `handleDeposit` reads the typed amount, then `if (checkEqualValue(value, availableToDeposit, pool.decimals)) { value = constants.MaxUint256.toString() }`. That rewritten `value` is used both for the approval (`approveFn` → `_sdk.approve(pool.asset.address, pool.id, value)`) and for the deposit itself (`depositFn` → `_sdk.supply(pool.id, value)`). So when the user clicks a MAX chip, or types a figure that happens to equal their available balance, the amount field still displays e.g. '10,000' and the only button reads 'Approve & Deposit', while the two transactions actually carry 2^256-1 in both the allowance slot and the deposit-amount slot. (b) [redacted] pool repayment. `RepayPool.svelte` renders a 'Total Due' figure and a single button labelled 'Repay Pool'. `handleRepayPool` first calls `approveFn`, which is `sdk.approveTransfer(pool.asset.address, pool.id, constants.MaxUint256, 0)` — an unconditional unlimited approval, commented in the source as `//approving maxUint`. The UI never names an approval step, never states an approval amount, and never mentions the word 'unlimited'. The same construction appears in RepayBulletPool.svelte:54 and RepayLender.svelte:82,98. The `supply(MaxUint256)` in (a) is presumably a contract-side sentinel meaning 'deposit my whole balance'; I cannot confirm that on-chain

from this snapshot and flag it for human verification. Either way the displayed number is not the number encoded.

Impact. For (a): if the sentinel reading is right, the amount actually deposited is the balance at mining time rather than the balance shown at signing time, so an incoming transfer landing in between silently increases the deposit past what the user reviewed; if the sentinel reading is wrong, the call reverts. In both readings the residual allowance to the pool is unlimited and outlives the deposit. For (b): the borrower grants an unlimited, permanent allowance on the pool's asset to the pool contract having been shown only 'Total Due' and a 'Repay Pool' button — any future compromise or upgrade of that pool contract can drain the borrower's full token balance rather than the amount they intended to repay. Bounded by the fact that the spender is in every case the pool the user chose to interact with, which is why this is Medium and not High.

Recommendation. Approve the exact required amount, as the sibling __v2/Modals/Deposit.svelte:98 already does (`sdk.approve(_poolAddress, amount)` with the parsed amount). If an unlimited allowance is a deliberate UX choice, surface it: a distinct 'Approve' step naming the spender and stating 'unlimited', with an exact-amount alternative. For the MAX path, either pass the exact parsed balance or, if the MaxUint256 sentinel is genuinely required by the contract, change the displayed copy to 'Deposit entire balance' so the figure shown and the figure signed agree.

> ANALYSIS Source trace, verified by reading both modals end to end. Cannot be executed without a funded wallet and a live RPC, both out of scope.
>
> (a) UI: `<Button type="submit" ... text="Approve & Deposit" />` (Deposit.svelte:236) over an input bound to the user's typed amount. Payload: `value = constants.MaxUint256.toString()` (line 79) → `_sdk.approve(pool.asset.address, pool.id, value)` (line 133) and `_sdk.supply(pool.id, value)` (line 109). (b) UI: `title="Total Due"` and `text="Repay Pool"` (RepayPool.svelte:159,177). Payload: `sdk.approveTransfer(pool.asset.address, pool.id, constants.MaxUint256, 0)` (line 47), reached whenever `!parseUnits(allowance).eq(constants.MaxUint256.toString())` (line 88).

PROTO-07 · **Medium**

[redacted] dApp deposit signs minimumMint = 0 while the modal displays an exact 'You Receive' share amount

LOCATION [redacted] dApp/src/lib/sdk/services/port.ts:165-176 vs src/features/port/components/vault/user/modals/DepositModal/{ReceiveSectio 20,AmountInput.tsx:157-184}

The deposit modal computes `receiveAmount = amountBN * 10^decimals / navBN` from the client-side `vault.currentNav` and renders it under the heading 'You Receive' as a concrete figure next to the vault symbol. `depositERC20` then calls `TellerWithMultiAssetSupport.deposit` with `args: [asset, amount, 0n]`. Inspection of the ABI confirms the third parameter is `uint256 _minimumMint`, so the hardcoded `0n` tells the contract to accept any number of shares, including one. The NAV used for the preview is a value fetched from the indexer, and the same dapp exposes `updateExchangeRate` on the accountant (port.ts:236) as a manager action, so the rate can legitimately change between the moment the figure is rendered and the moment the transaction is mined.

Impact. The share count the user is shown carries no on-chain floor. Any downward rate movement, manager rate update, or stale indexer NAV between preview and execution mints fewer shares than displayed, silently and without revert. Not third-party-exploitable in an open market sense (the Teller is

permissioned), which caps this at Medium, but it is a promised-versus-signed divergence on the primary user action.

Recommendation. Derive `_minimumMint` from the displayed `receiveAmount` less an explicit, user-visible tolerance, and pass it. If a zero minimum is intentional, remove the concrete 'You Receive' figure or label it clearly as an unguaranteed estimate. Note the same `0` appears as `minimumAssetsOut` in `redeemSolve` (port.ts:424) and `minOfferReceived` in `p2pSolve` (port.ts:365) — those are manager-only paths, but they deserve the same review.

> PROOF OF CONCEPT — EXECUTED Executed python3 substring extract of TellerWithMultiAssetSupport.ts around `name: 'deposit'` → inputs `[_depositAsset address, _depositAmount uint256, _minimumMint uint256]`, confirming the third positional argument is the minimum mint.
>
> UI: `<p className="text-b3 text-gray-30">You Receive</p>` followed by `<span>{receiveAmount?.formatted || '0'}</span> <span>{vault?.symbol}</span>`. Payload: `simulateContract({ abi: TellerWithMultiAssetSupport_ABI, address: tellerAddress, args: [asset, amount, 0n], functionName: 'deposit' })`.

PROTO-08 · Medium

XRPL deposit UI announces success without submitting or validating the transaction result

LOCATION [redacted] dApp/public/index.html:1874-1900 (WalletConnect), :1805-1836 (Crossmark), :1858-1867 ([redacted])

None of the four deposit branches checks the ledger outcome — `grep -rn 'tesSUCCESS|TransactionResult|engine_result'` across the whole dapp returns five hits, all inside lib/services/xrpl-service.js on the server, and zero in either HTML shell. Three specific problems, in descending severity. (a) WalletConnect/Girin: the code awaits `wcSignClient.request({ ..., request: { method: 'xrpl_signTransaction', params: { tx_json: {...} } } })`, discards the return value entirely, and then calls `showToast('Deposit signed & submitted via WalletConnect!')`. `xrpl_signTransaction` is the sign-only method in the XRPL WalletConnect namespace — it returns a signed blob for the dapp to submit; the counterpart that submits is a different method. Nothing in this file submits the returned blob and nothing reads it, so with a spec-conformant wallet the payment never reaches the ledger while the user is told it was submitted. (b) Crossmark: `const txHash = resp?.response?.data?.resp?.result?.hash; if (txHash) { showToast('Deposit sent! ...') } else { showToast('Deposit submitted via Crossmark') }` — the else branch reports success precisely in the case where no transaction hash came back, i.e. where something went wrong. (c) All branches: even when a hash is returned, `meta.TransactionResult` is never inspected, so a mined-but-failed payment (tecUNFUNDED_PAYMENT, tecNO_DST, tecPATH_DRY, tecDST_TAG_NEEDED) still produces 'Deposit sent!'. Each success branch then schedules `refreshUser()` and `loadHistory()` 5 seconds later, so the balance quietly fails to move.

Impact. Users are told deposits completed when they did not. In case (a) the funds never left the user's account, so the loss is confusion and support burden rather than money; but users who believe a deposit landed will not retry, and will report a missing balance against a vault that never received anything. Case (c) is the same outcome for genuinely failed payments.

Recommendation. For WalletConnect, either switch to the submitting method for the connected namespace or take the signed blob from the response and submit it via the app's own XRPL client, then

wait for validation. For every branch, resolve the transaction on-ledger and require `meta.TransactionResult === 'tesSUCCESS'` before showing a success toast — the server already applies exactly this check at lib/services/xrpl-service.js:302 (`if (tr !== "tesSUCCESS") throw new Error("Transaction was not successful")`), so reuse that standard on the client. Replace the Crossmark else-branch with an error toast.

> PROOF OF CONCEPT — EXECUTED Executed: `grep -rn 'tesSUCCESS|TransactionResult|engine_result' --include=*.html --include=*.js --include=*.ts . | grep -v /public/vendor/` → 5 hits, all in lib/services/xrpl-service.js (lines 212, 259, 261, 300, 302); 0 hits in public/index.html or public/admin.html.
>
> index.html:1876-1897 `await wcSignClient.request({ topic: wcSession.topic, chainId: 'xrpl:1', request: { method: 'xrpl_signTransaction', params: { tx_json: { TransactionType: 'Payment', Account: userAddress, Destination: depositAddress, DestinationTag: destinationTag, Amount: amountDrops } } } });
> showToast('Deposit signed & submitted via WalletConnect!');` — return value never bound.
> index.html:1832-1836 `if (txHash) { showToast('Deposit sent! TX: ' + txHash.slice(0,12) + '...') } else { showToast('Deposit submitted via Crossmark') }`.

PROTO-09 · **Medium**

Withdrawal request deadline collapses to the current second for any vault missing a hardcoded constants entry

LOCATION [redacted] dApp/src/state/client/port/usePortUserWithdraw.ts:25,47-56 with src/features/port/constants.ts:38-40,234-244 and src/lib/sdk/services/port.ts:214-216

The withdraw hook computes `totalDeadlineHours = ((vaultConfig?.noticePeriod ?? 0) + (vaultConfig?.withdrawalDeadline ?? 0)) * 24` and passes it explicitly to `withdrawERC20`, overriding that function's sensible `deadlineHours = 888` default. `vaultConfig` comes from `getVaultConstants(vault)`, which looks the vault up in a hardcoded map keyed by ``` `${name}-${networkId}-${address.toLowerCase()}` ```. When that lookup misses, both `??` fallbacks fire, `totalDeadlineHours` is 0, and port.ts computes `deadline = Math.floor(Date.now()/1000) + 0` — a deadline equal to the moment of construction, already in the past by the time the transaction is mined. The lookup misses for any vault the indexer returns that is not one of the five entries in `portVaultsConstants`, and the vault list is not filtered on membership: `hasVaultConstants` is exported but a grep shows it is called from nowhere, and `useGetPortVaults.ts:30` filters only on `!vaultConfig?.hideOnProd`, which is `!undefined === true` for exactly the unknown vaults. The key is also brittle in a way that makes a miss likely over time — it embeds the vault's display `name`, so an upstream rename from e.g. 'X-Pool' silently breaks the match for a vault that is otherwise correctly configured. Two vaults already present in the sibling dapp's pool list (oval-fi `0x615132b8...`, liquid-credit `0xa44e379d...`) have no entry in this map.

Impact. The user approves their vault shares to the AtomicQueue and submits a withdrawal request that is dead on arrival and can never be solved. The modal simultaneously displays 'Request Deadline: 0D', which is the one visible symptom, but nothing blocks submission. The user pays gas for two transactions, believes a withdrawal is pending, and waits on a request that will never fill. Funds are not lost — the shares are not transferred until a solve — but the core exit path silently fails for any vault outside the hardcoded five.

Recommendation. Do not let a missing config produce a zero deadline. Either refuse to render the withdraw action when `hasVaultConstants(vault)` is false (the helper already exists for this and is

unused), or fall back to the SDK's 888-hour default by passing `undefined` rather than a computed 0. Better still, source `noticePeriod` and `withdrawalDeadline` from the indexer alongside the vault rather than from a name-keyed local map, and drop `name` from the key so a display rename cannot break the lookup.

> PROOF OF CONCEPT — EXECUTED Executed: `grep -rn 'hasVaultConstants|getVaultConstants' src` → `hasVaultConstants` is defined at constants.ts:234 and imported nowhere; all 20 `getVaultConstants` call sites are presentational components plus this withdraw hook. `grep -rn 'hideOnProd' src | grep -v constants.ts` → 2 hits, both `!vaultConfig?.hideOnProd`, which admits unknown vaults.
>
> ```
> usePortUserWithdraw.ts:47 `const totalDeadlineHours = ((vaultConfig?.noticePeriod ?? 0) +
> (vaultConfig?.withdrawalDeadline ?? 0)) * 24` → port.ts:214-216 `const deadline = isCancelled ?
> Math.floor(Date.now()/1000) + 10 : Math.floor(Date.now()/1000) + deadlineHours * 60 * 60`.
> constants.ts:38-40 `return `${name}-${networkId}-${address.toLowerCase()}``.
> ```

PROTO-10 · **Medium**

XRPL withdrawal requests are authorised by a persistent per-address flag rather than an authenticated session

LOCATION [redacted] dApp/lib/api/public-router.js:384-427 with lib/vault/wallet-verification.js:7-14

`POST /api/withdraw` accepts `{ address, amount }` from the body and gates on `if (!(await isWalletVerifiedMerged(address)))`. That helper is `db.isWalletVerified(address) || await wstore.isWalletVerifiedNeon(address)` — a durable boolean recorded once, when that address completed the wallet-challenge flow at some point in the past. It carries no notion of the current caller. The request itself bears no session cookie, no bearer token, and no signature: the client at public/index.html:2029 sends `headers: { 'Content-Type': 'application/json' }` and nothing else. The client-side check that refuses `walletProvider === 'manual'` (index.html:2019) is cosmetic and trivially bypassed by calling the endpoint directly. XRPL classic addresses are public on-ledger data, and the app's own `/api/user/:address/deposits` and `/withdrawals` endpoints make depositor addresses enumerable.

Impact. Any third party who knows a verified depositor's XRPL address can force a withdrawal request against that user's position. Because the vault is custodial and pays out to the address on file, this is not direct theft — the funds return to the rightful owner — but it lets an attacker forcibly exit any user's yield-bearing position at will, repeatedly, and lets them spam the withdrawal queue and the treasury's payout schedule. The route's own `auditLargeWithdrawalRequest(address, amt, clientIp(req))` acknowledges the sensitivity of the action while the authorisation check does not.

Recommendation. Authorise the caller, not the address. The codebase already has the pieces: `lib/auth/admin-session.js` issues HMAC bearer sessions and `generateNonceRecord('user')` / `consumeNonce` implement a user-purpose challenge. Require a valid user session bound to `address`, or require a fresh signed challenge on the withdrawal request itself, and reject when the session subject and the body's `address` disagree.

PROOF OF CONCEPT — EXECUTED Executed source trace. public-router.js:404 is the only authorisation check on the route and its argument is the body-supplied `address`. wallet-verification.js:7-14 shows the helper reads a stored flag with no request context. Confirmed the client sends no auth header: `grep -n 'Authorization|userToken|sessionToken' public/index.html` → 0 hits. Not exercised against any live endpoint.

```
public-router.js:384 `r.post("/withdraw", async (req, res) => { const parsed =
withdrawBodySchema.safeParse(req.body ?? {}); const { address, amount: amt } = parsed.data; ... if (!
(await isWalletVerifiedMerged(address))) return res.status(403)... const result = await
getVault().requestWithdrawal(address, amt)`.
```

PROTO-R06 · **Low**

Unauthenticated cron GET endpoints on the SvelteKit app

LOCATION	none

The SvelteKit cron routes are plain unauthenticated GETs. `[redacted] dApp/src/routes/api/crons/coin-prices/+server.ts:11` is `export const GET: RequestHandler = async ({ fetch }) => {` with no authorization check before calling `pro-api.coinmarketcap.com` with `config.cmc_api_key` and then writing MongoDB through `coinPrices.create`/`updateOne`. `protocol-kpi` has the same shape and fans out to twelve internal KPI routes before writing the database. Across the app only 3 of 29 `+server.ts` endpoints reference any credential at all, and one of those is a false positive. The pattern that would fix it already exists in the same scope: `[redacted] dApp` gates every `/api/cron/*` route through `lib/cron-auth.ts` with a constant-time `CRON_SECRET` compare.

Impact. Any anonymous caller can trigger the scheduled jobs on demand: `coin-prices` burns CoinMarketCap quota on [redacted]'s paid key, and both routes drive MongoDB writes at whatever rate the caller chooses, with `protocol-kpi` fanning out across twelve internal routes per call. The realistic outcomes are quota exhaustion -- and price data going stale once the quota is gone -- plus database write load and needless internal traffic. No authentication bypass or data disclosure is involved, so the exposure is cost and availability rather than confidentiality.

Recommendation. Gate every route under `[redacted] dApp/src/routes/api/crons/*` on a `CRON_SECRET` compared with `crypto.timingSafeEqual`, reusing the pattern already implemented in `[redacted] dApp/lib/cron-auth.ts`. Enforce it centrally in `hooks.server.ts` by path prefix rather than per route, so the remaining uncredentialed `+server.ts` endpoints are covered by default and a new one cannot ship open. As it stands, `coin-prices` and `protocol-kpi` let any anonymous caller burn the CoinMarketCap quota and drive MongoDB writes on demand.

PROOF OF CONCEPT — EXECUTED Read [redacted] dApp/src/routes/api/crons/coin-prices/+server.ts and .../protocol-kpi/+server.ts, and grepped all 29 +server.ts files for any credential reference.

```
coin-prices/+server.ts:11 is `export const GET: RequestHandler = async ({ fetch }) => {` with no
authorization check before calling pro-api.coinmarketcap.com with config.cmc_api_key and then writing
MongoDB via coinPrices.create/updateOne. protocol-kpi has the same shape and fans out to 12 internal KPI
routes before writing the DB. Only 3 of 29 endpoints reference any credential at all (send-email,
discord-callback, bonds - and the last is a false positive). SIBLING: [redacted] dApp gates every
/api/cron/* route through lib/cron-auth.ts with a constant-time CRON_SECRET compare, so the correct
pattern exists inside the same engagement scope.
```

PROTO-N12 · Low

The withdrawal notification email formats a vault-share amount using the base token's decimals

LOCATION [redacted] dApp/src/lib/sdk/services/email.ts:88-102; src/state/client/port/usePortUserWithdraw.ts:57-65

usePortUserWithdraw passes { amount: <shares, parsed with vault.decimals>, token: vault.baseToken } to EmailService.sendWithdrawalRequestNotification, which formats it as `Number(amount.toString()) / Math.pow(10, token.decimals)`. The amount is denominated in vault shares; the divisor is the base token's decimals. The two coincide only if the BoringVault happens to have been deployed with the same decimals as its base asset, which nothing in the app checks or asserts. The deposit notification does not have this problem because it passes the pre-formatted amount.formatted string.

Impact. The borrower's 'Withdrawal Request Created' email - which is what a borrower uses to plan liquidity for the notice period - can state an amount wrong by a factor of 10^(vault.decimals - token.decimals). For an 18-decimal share against a 6-decimal stablecoin that is a factor of one trillion. Number() on a raw 18-decimal integer is also already beyond the safe-integer range, so the value is additionally rounded.

Recommendation. Format from the same DisplayBN the modal renders (amount.formatted), or divide by vault.decimals and label the figure in the vault symbol rather than the base token symbol. Assert vault.decimals === baseToken.decimals somewhere if the code intends to rely on it.

> ANALYSIS Reasoned, not executed.
>
> email.ts:97 `const formattedAmount = (Number(amount.toString()) / Math.pow(10, token.decimals)).toLocaleString(...)`; usePortUserWithdraw.ts:64 `amount: data.amount` where data.amount originates from AmountInput's `utils.parseUnits(cleanValue, vault?.decimals || 18)` and token = vault.baseToken. Compare email.ts:69 (deposit) which uses `amount: amount.formatted`.

PROTO-N13 · Low

Safe transaction polling is an unbounded while(true) with no timeout, so a Safe transaction that is never executed locks the modal permanently

LOCATION [redacted] dApp/src/lib/sdk/services/safe.ts:249-257

waitForTransactionExecution loops `while (true)`, polling the Safe transaction service every 5 seconds and returning only when isExecuted becomes true. There is no timeout, no maximum iteration count, no abort signal and no rejection path. A Safe transaction that never reaches its owner threshold, or that an owner rejects, never satisfies the condition.

Impact. The react-query mutation never settles, so isPending stays true forever. Both the deposit and withdraw modals pass loading={isPending} to Modal and guard their close handler with `if (!isPending)`, so the user cannot dismiss the dialog; the only recovery is a page reload. In the background the tab polls the Safe API every 5 seconds indefinitely.

Recommendation. Bound the loop with a deadline and an iteration cap, reject on a rejected/replaced Safe transaction, accept an AbortSignal so closing the modal cancels it, and always allow the modal to be closed.

ANALYSIS Reasoned, not executed.

`async waitForTransactionExecution(safeTxHash: string) { while (true) { const transaction = await this.getSafeTransactionBySafeTxHash(safeTxHash); if (transaction.isExecuted) { return transaction } await new Promise(resolve => setTimeout(resolve, SAFE_TRANSACTION_POLLING_INTERVAL)) } }`

PROTO-N14 · Low

[redacted] dApp's .env.example omits the two variables that decide which chains the app talks to, so a correctly-followed template silently defaults it to Plume

LOCATION [redacted] dApp/.env.example; [redacted] dApp/src/shared/utils/init.ts:3-13; [redacted] dApp/src/lib/sdk/context.ts:26

getSupportedChains() reads NEXT_PUBLIC_SUPPORTED_NETWORKS, falls back to NEXT_PUBLIC_DEFAULT_CHAIN_ID, and finally falls back to the hardcoded [PLUME_MAINNET]. Context's constructor sets `this.chainId = Number(process.env.NEXT_PUBLIC_DEFAULT_CHAIN_ID)`, which is NaN when the variable is absent. Neither variable appears in .env.example, which lists only NEXT_PUBLIC_ENV, NEXT_PUBLIC_PROJECT_ID, NEXT_PUBLIC_VERCEL_URL, NEXT_PUBLIC_SUPPORTED_MARKETS and the private keys. A separate `export const supportedNetworks = [SupportedNetworkIds.ETHEREUM_SEPOLIA]` sits unused at the top of features/port/constants.ts, contradicting the mainnet/Flare vaults defined below it.

Impact. An operator who provisions from the documented template gets an app whose supported-chain set is the silent Plume fallback and whose pre-connection chainId is NaN. Since the production vault filter in useGetPortAllVaults is `supportedChains.includes(vault.networkId)`, the visible vault list depends on an undocumented variable - vaults appear or vanish based on configuration nobody was told to set. The default serves the last deployment target rather than the operator.

Recommendation. Add both variables to .env.example with the production values; make the final fallback throw at boot rather than silently choosing a chain; delete the unused supportedNetworks export.

ANALYSIS Reasoned, not executed.

init.ts:3-13 (three-level fallback ending in `[SupportedNetworkIds.PLUME_MAINNET]`); context.ts:26 `this.chainId = Number(process.env.NEXT_PUBLIC_DEFAULT_CHAIN_ID) as SupportedNetworkIds`; .env.example contains neither name; features/port/constants.ts:6 `export const supportedNetworks = [SupportedNetworkIds.ETHEREUM_SEPOLIA]` with no reader.

PROTO-N15 · Low

Unauthenticated GETs publish the full merkle authority map for every vault

LOCATION [redacted] dApp/src/app/api/admin/trees/route.ts (no authenticate()); src/app/api/merkle/route.ts GET; src/app/api/admin/actions/route.ts GET; src/app/api/admin/strategy/route.ts GET

The POST/PUT/DELETE handlers on these routes all call authenticate(); the GET handlers do not, on any of them. Anyone can therefore retrieve, for any vault and chain, the complete set of merkle trees (GET /api/admin/trees?vault=&chainId=), the leaves and root for a given manager/strategist pair (GET /api/merkle), and the full action definitions including target, decoder, selector, ABI, input schema and placeholder data (GET /api/admin/actions). The path is also injectable: `id`/`slug` is interpolated unvalidated into the upstream URL by fetchApi.

Impact. This is the authority map for the vault: exactly which targets each strategist may call, with which decoder and which selector, and what the current root is. It is not a secret in the cryptographic sense - the root is on chain - but publishing the pre-image set turns a targeted-guessing problem into a lookup, and it gives an attacker who later obtains a strategist key or an admin session the complete inventory of what that key can reach without a single on-chain probe. Combined with PROTO-N01's master-token escalation and the SIWE weaknesses above, it materially shortens the reconnaissance phase.

Recommendation. Require authenticate() on the GET handlers too, or scope them to the data the connected manager already controls; validate `id`/`slug` against an id format before interpolating it into the upstream path.

> ANALYSIS Reasoned, not executed (LOCAL ONLY).
>
> admin/trees/route.ts has no import of authenticate at all. merkle/route.ts imports authenticate and calls it only in POST. admin/actions/route.ts and admin/strategy/route.ts call `await authenticate()` in POST/PUT/DELETE and not in GET. shared/utils/api.ts:10 `fetch(`${process.env.API_URL}${url}`, ...)` with url built as `/api/actions/${id}`.

PROTO-11 · Low

Admin SIWE verification does not pin domain or uri, and the nonce is reusable for up to 30 days

LOCATION [redacted] dApp/src/shared/utils/auth.ts:10,12-51 and src/app/api/admin/nonce/route.ts:9-23

`verifyAdminSiwe` calls `validateSiweMessage({ address, message: parsed, nonce, time })`. viem validates only the fields supplied, so with `domain` and `uri` omitted the SIWE message's own domain claim is never compared against the server's expected origin — the EIP-4361 domain binding that distinguishes a SIWE signature from a bare personal_sign is not enforced. The client does populate `domain: window.location.host` and `uri: window.location.origin` (useAdminLogin.tsx:45-47), but those are client-side values the server never checks. Separately, `GET /api/admin/nonce` returns the existing `x-nonce` cookie if one is present and only mints a new one otherwise, and the cookie's `maxAge` is `MAX_AGE`, defaulting to `60*60*24*30`. The nonce is therefore long-lived and re-usable rather than single-use, and it is never consumed on successful verification — `authenticate()` re-verifies the same stored message/signature/nonce triple on every subsequent request. The cookies are `httpOnly` and `secure` in production, and no `sameSite` is set so Next's `lax` default applies.

Impact. A SIWE signature produced for one origin would be accepted by this server, so a phishing page that reproduced the exact statement and a valid nonce could obtain an admin-panel session. That requires the attacker to hold a currently-valid nonce, which is httpOnly and not readable cross-origin — which is what keeps this at Low — but the 30-day validity window materially widens the opportunity for a nonce leaked through a HAR file, a shared debug session, or proxy logs. Blast radius on success is the admin panel: merkle trees, strategies, and action definitions.

Recommendation. Pass `domain` and `uri` to `validateSiweMessage`, sourced from server configuration rather than from the message. Make the nonce single-use — delete it from the cookie jar the moment `/api/admin/verify` succeeds — and shorten its lifetime to minutes. Set `sameSite: 'strict'` explicitly on `x-nonce`, `x-signature` and `x-message`.

ANALYSIS Source review of viem's validateSiweMessage contract (fields not supplied are not validated) against the call at auth.ts:18-23, plus the nonce lifecycle across nonce/route.ts and verify/route.ts. Not executed — would require signing with a real key.

auth.ts:18-23 `validateSiweMessage({ address: parsed.address, message: parsed, nonce, time: new Date() })` — no `domain`, no `uri`. auth.ts:10 `MAX_AGE = process.env.ADMIN_NONCE_MAX_AGE ? ... : 60 * 60 * 24 * 30`. nonce/route.ts:11-13 `const nonce = cookieStore.get('x-nonce')?.value; if (nonce) return NextResponse.json({ nonce })`.

PROTO-12 · Low

Wallet authentication signature uses a static, nonce-less, domain-unbound message and is logged to the console

LOCATION [redacted] dApp/src/lib/utils/authUtils.ts:10-22 and src/routes/(products)/notifications-v2/+page.svelte:116-125

`requestWalletSignature` asks the user to sign the fixed string 'Please sign this message to authenticate with our service.' — no nonce, no timestamp, no domain, no chain id, no address binding, and not EIP-4361 formatted. A signature over that string is identical every time for a given wallet and is indistinguishable from one solicited by any other site that asks for the same words, so it is replayable forever and provides no origin binding. In the one place it is called, the result is not sent anywhere: the caller does `const { signature, address } = await requestWalletSignature(); console.log(signature, address); authCookie = getAuthCookie()` — the signature is written to the browser console and then dropped, and the code re-reads a cookie that nothing in this flow sets.

Impact. Low today, because the signature is never transmitted or verified, so no session actually depends on it — this reads as an unfinished feature. The concrete harms are that the notifications page trains users to approve an unbound signature request (eroding the habit of scrutinising what a site asks them to sign) and that a wallet signature is printed to the console where any browser extension with page access can read it. If the server side is completed later against this same message, it becomes a replayable authentication bypass.

Recommendation. Delete the dead call and the `console.log`, or complete the flow using EIP-4361 with a server-issued single-use nonce, an explicit `domain`, `uri`, `chainId` and issuance time — the pattern the sibling [redacted] dApp already uses via `viem/siwe`. Never log a signature.

PROOF OF CONCEPT — EXECUTED Executed: `grep -rn 'requestWalletSignature|getAuthCookie' src` → defined in authUtils.ts, called from exactly one site, notifications-v2/+page.svelte:119. The signature variable is referenced only by the adjacent console.log; no fetch or api call consumes it.

authUtils.ts:14 `const message = 'Please sign this message to authenticate with our service.'` — constant. +page.svelte:119-121 `const { signature, address } = await requestWalletSignature(); console.log(signature, address); authCookie = getAuthCookie()`.

PROTO-13 · Low

[redacted] dApp SDK never refreshes its wallet client after a chain switch, leaving signer chain id and gas estimation bound to the old chain

LOCATION [redacted] dApp/src/shared/hooks/useWalletConnection.ts:101-105 with src/lib/sdk/context.ts:42-47,99-103 and src/lib/sdk/services/transaction.ts:48-51,62-85

`useEffect(() => { if (!sdk.context.hasWalletClient() && walletClient) { sdk.setWalletClient(walletClient) } }, [walletClient])` installs the wagmi wallet client exactly once. wagmi returns a new `walletClient` instance bound to the new chain whenever the user switches networks, but the guard `!hasWalletClient()` is false from then on, so the SDK keeps the original client forever. Consequences: `context.signerChainId` (which reads `walletClient.chain.id`) reports the chain the user first connected on; `signerPublicClient` resolves to that chain's public client; `writeContract`'s guard `if (targetChain && targetChain !== this.sdk.context.signerChainId)` compares against a stale value; and `withGasEstimationForContract` picks its client from the stale `walletClient.chain.id` and estimates gas on the wrong chain. Note `context.chainId` is separately kept current by the appKit effect at line 95-99, so the SDK holds two chain notions that disagree after any switch. Most port.ts writes call `writeContract(request)` with no `targetChain` at all, so nothing in the SDK re-checks the chain.

Impact. Bounded to failure rather than mis-execution. The `request` object returned by viem's `simulateContract` carries the simulating chain, and viem's `writeContract` fetches the provider's live chain id and asserts a match before sending, so a mismatched transaction throws instead of being broadcast to the wrong chain — that assertion is what keeps this at Low. The observable effect is that gas estimation and writes fail after a chain switch until the page is reloaded, with errors that do not point at the cause.

Recommendation. Drop the `!hasWalletClient()` guard and set the wallet client whenever the wagmi client identity or its chain changes: `useEffect(() => { if (walletClient) sdk.setWalletClient(walletClient) }, [walletClient])`. Then collapse `context.chainId` and `signerChainId` to a single source of truth so the two cannot diverge. Separately, `TransactionsService.isERC20Approve` (transaction.ts:29-46) is dead code — nothing calls it — and should be removed or wired up.

> ANALYSIS Source trace plus viem's documented writeContract behaviour (it resolves the provider chain id and asserts against `request.chain`). Not executed — reproducing requires a live wallet and two chains.

useWalletConnection.ts:101-105 `useEffect(() => { if (!sdk.context.hasWalletClient() && walletClient) { sdk.setWalletClient(walletClient) } }, [walletClient])`. context.ts:44-46 `get signerChainId() { ... return this.context.walletClient.chain.id }`. transaction.ts:71 `const walletChainId = this.sdk.context.walletClient.chain.id`.

PROTO-14 · Low

[redacted] dApp and [redacted] dApp ship no security response headers; the XRPL CSP permits unsafe-eval and unsafe-inline

LOCATION [redacted] dApp/next.config.ts (no headers()); [redacted] dApp/vercel.json (functions and crons only) and [redacted] dApp/lib/http/csp.js:5-22

[redacted] dApp sets a reasonable baseline in vercel.json — `Content-Security-Policy: frame-ancestors 'none'`, `X-Frame-Options: same-origin`, `X-Content-Type-Options: nosniff`, `Referrer-Policy: same-origin` on `/(.*)`. Neither Next.js app does anything equivalent: [redacted] dApp's next.config.ts has no `headers()` function and there is no vercel.json, and [redacted] dApp's vercel.json contains only `functions` and `crons`. So the [redacted] dApp vault UI — the page that constructs deposit, withdrawal and manager transactions — is served with no framing protection, no nosniff and no CSP at all, and is embeddable in a hostile frame. [redacted] dApp does define a CSP string in lib/http/csp.js applied to its HTML shells, and that one does set `frame-ancestors 'none'`, but it permits `'unsafe-inline'` and `'unsafe-eval'` in `script-src` alongside three CDNs (see PROTO-03) and opens `connect-src` to `wss://*`, so it constrains very little.

Impact. No clickjacking or framing protection on a transaction-signing UI, and no CSP to blunt an injected-script incident should one arise. Low on its own because no reachable injection sink was found in [redacted] dApp, but these headers are the mitigation that would matter if one appeared, and their absence is the reason PROTO-03's CDN exposure has no second line of defence.

Recommendation. Add a `headers()` block to [redacted] dApp/next.config.ts mirroring what [redacted] dApp already ships, and add the same to [redacted] dApp/vercel.json so the policy covers Next-rendered routes and not just the static shells. For the XRPL CSP, remove `'unsafe-eval'`, work toward removing `'unsafe-inline'` via nonces, and replace `connect-src wss://*` with the specific relay hosts already listed.

> PROOF OF CONCEPT — EXECUTED Executed: read all three deploy configs in full. [redacted] dApp/vercel.json contains a `/(.*)` headers entry with CSP/Referrer-Policy/X-Content-Type-Options/X-Frame-Options/X-XSS-Protection. [redacted] dApp/next.config.ts contains only `images`, `sassOptions`, `eslint` and `typescript` keys. [redacted] dApp/vercel.json contains only `functions` and `crons`.
>
> csp.js:7 `script-src 'self' 'unsafe-inline' 'unsafe-eval' https://unpkg.com https://cdn.jsdelivr.net https://[redacted].app https://esm.sh;` and csp.js:16 `wss://*;`

PROTO-15 · Low

Dependency vulnerabilities: 192 in [redacted] dApp (1 critical, 11 high), 6 in [redacted] dApp (5 high); [redacted] dApp not auditable offline

LOCATION	[redacted] dApp/package-lock.json; [redacted] dApp/package-lock.json; [redacted] dApp/pnpm-lock.yaml

`npm audit --package-lock-only` was executed against both npm-locked projects. [redacted] dApp: 192 advisories — 1 critical (node-tar <=7.5.20, PAX header interpretation differential enabling file smuggling), 11 high, 114 moderate, 66 low. Direct dependencies among the highs: axios 1.0.0-1.17.0 (DoS via formDataToJSON recursion), postcss <=8.5.22 (sourceMappingURL path traversal disclosing arbitrary .map files), eslint, sass, @storybook/svelte. Transitives of note: ip-address <=10.3.0 (leading-zero octet parsing differential enabling SSRF), fast-uri (host confusion via backslash authority), form-data (CRLF injection), nanoid, js-yaml, brace-expansion, shell-quote, socket.io-parser. [redacted] dApp: 6 advisories — 5 high (next, sharp/libvips CVE-2026-33327/33328/35590/35591, form-data, nanoid, postcss) and 1 low (body-parser). The vast majority in both projects sit in build and test tooling rather than the shipped bundle, which is why this is Low rather than higher; the `next` and `sharp` advisories in [redacted] dApp

are the exceptions that affect the running server and warrant priority. [redacted] dApp uses pnpm-lock.yaml and no pnpm binary was available offline, so it was NOT audited — an explicit gap.

Impact. Mostly developer-machine and CI exposure. The `next` and `sharp` advisories in [redacted] dApp affect production runtime. No advisory was traced to a directly exploitable path in the shipped dapp bundles during this review.

Recommendation. Run `npm audit fix` on both npm projects and prioritise the runtime-affecting `next` and `sharp` upgrades in [redacted] dApp and the node-tar critical in [redacted] dApp's toolchain. Run `pnpm audit` against [redacted] dApp — it is the only one of the three that was not covered here. [redacted] dApp already has renovate.json and dependabot.yml configured; confirm they are actually enabled, since the lockfile suggests they are not landing updates.

> PROOF OF CONCEPT — EXECUTED Executed `npm audit --package-lock-only` in [redacted] dApp → '192 vulnerabilities (66 low, 114 moderate, 11 high, 1 critical)'. Executed the same in [redacted] dApp → '6 vulnerabilities (1 low, 5 high)'. High/critical entries enumerated via a python3 filter over the JSON report. [redacted] dApp: no pnpm available, not audited.
>
> Counts and advisory titles as reported by npm audit; no secret material involved.

PROTO-N16 · **Info**

Unauthenticated bond-metadata endpoint reflects an attacker-chosen name into a JSON document served from [redacted] and cached for 30 days

LOCATION [redacted] dApp/src/routes/api/bonds/[bondToken]/[bondId]/+server.ts:5-45

The route builds NFT metadata from six unvalidated query parameters plus two path parameters, with no authentication and no allow-list, and returns it with `Cache-Control: public, max-age=2592000, s-maxage=2592000`. `name` is `symbol + ' Bond'` where symbol is the caller's string; tokenId, contract address, credit vault, APR, penalty APR and withdrawal date are likewise reflected into the attributes array. The description and image_url are fixed literals, which bounds the damage.

Impact. Any URL of this shape is a [redacted]-hosted, 30-day-edge-cached JSON document whose displayed name and attribute values the requester chose. Pointed at by a third-party NFT's tokenURI, or simply linked, it presents attacker-authored labelling under [redacted]'s domain - the same brand-borrowing primitive as the known unauthenticated email route, at lower intensity.

Recommendation. Validate the parameters (symbol against a known token set, addresses with a checksum check, rates and dates as bounded numbers) and reject anything else; or derive the metadata from the indexer using only the path parameters.

> ANALYSIS Reasoned, not executed.
>
> `const symbol = url.searchParams.get('symbol') as string` ... `generateJSON(symbol, params.bondId, params.bondToken, pool, interestRate, penaltyRate, endDate, decimals || '6')` -> `{ name: symbol + ' Bond', created_by: '[redacted]', ... }`, returned with `'cache-control': 'public, max-age=2592000, s-maxage=2592000, stale-if-error=3600'`.

PROTO-16 · **Info**

[redacted] dApp production builds ignore all TypeScript and ESLint errors

LOCATION [redacted] dApp/next.config.ts:12-21

`eslint: { ignoreDuringBuilds: true }` and `typescript: { ignoreBuildErrors: true }`, the latter carrying the Next.js scaffold's own '!! WARN !!' comment. The repository also ships a 678 KB eslint-report.html and a 1.1 MB tsconfig.tsbuildinfo, suggesting a known and unresolved backlog. This is Info rather than a vulnerability, but it is directly relevant to the findings above: a type system that is not enforced at build time is why divergences like the `?? 0` deadline collapse in PROTO-09 and the never-supplied `icon` prop in PROTO-17 can ship unnoticed.

Impact. Type and lint regressions reach production silently. No direct security impact on its own.

Recommendation. Fix the backlog and remove both flags, or at minimum re-enable `typescript.ignoreBuildErrors: false` (as [redacted] dApp already sets) so type errors block a release. Remove the committed eslint-report.html and tsconfig.tsbuildinfo build artefacts from version control.

> PROOF OF CONCEPT — EXECUTED Read [redacted] dApp/next.config.ts in full; confirmed by contrast that [redacted] dApp/next.config.ts sets `typescript: { ignoreBuildErrors: false }`.
>
> next.config.ts:12-21.

PROTO-17 · **Info**

Latent dangerouslySetInnerHTML sink in a shared component (currently unreachable)

LOCATION [redacted] dApp/src/shared/components/SideBySideText/index.tsx:17

`{icon && <div dangerouslySetInnerHTML={{ __html: icon }} />}` renders the `icon` prop as raw HTML with no sanitisation. This is the only `dangerouslySetInnerHTML` in all three dapps. It is presently unreachable: an exhaustive check of all `<SideBySideText` usages (deposit TransactionDetails, UpdateNavModal, UpdateAPRModal, InitializeAPRModal, UpdateDepositCapModal and siblings) shows every call site passes only `title`, `primary` and `secondary` — `icon` is never supplied, so it defaults to `''` and the branch never renders. Reported as hygiene because the component sits in `shared/` and the next caller that wires up an icon from indexer or user data turns it into a live XSS sink on a transaction-signing page.

Impact. None today. The sink is a rendering hazard waiting on its first caller.

Recommendation. Replace the prop with a typed icon-name lookup (the codebase already has an `<Icon name=... />` component used elsewhere in the same modals) or a `ReactNode`, and delete the raw-HTML path.

> PROOF OF CONCEPT — EXECUTED Executed: `grep -rn 'dangerouslySetInnerHTML' --include=*.tsx --include=*.ts .` across all three dapps → 1 hit. Enumerated every `<SideBySideText` call site with a context grep and confirmed none passes `icon`.
>
> index.tsx:12 `icon = '',` and :17 `{icon && <div dangerouslySetInnerHTML={{ __html: icon }} />}`.

PROTO-18 · **Info**

Inverted feature-flag comparison and undeclared environment variables

LOCATION [redacted] dApp/src/lib/config/index.ts:66; [redacted] dApp/src/shared/utils/init.ts:3-12 with [redacted] dApp/.env.example

Two unrelated configuration defects found while auditing config sources. (a) `showRFLRCampaignBanner: PUB_SHOW_RFLR_CAMPAIGN === 'false'` — the flag is true precisely when the environment variable says 'false'. The sibling line 65 gets it right (`PUB_SHOW_OPTIMISM_CAMPAIGN === 'true'`), and .env.example documents `PUB_SHOW_RFLR_CAMPAIGN=true`, so the documented setting hides the banner and vice versa. (b) `getSupportedChains()` reads `NEXT_PUBLIC_SUPPORTED_NETWORKS` and `NEXT_PUBLIC_DEFAULT_CHAIN_ID`, neither of which appears in [redacted] dApp/.env.example — that file declares `NEXT_PUBLIC_SUPPORTED_MARKETS`, which nothing reads. When both are unset the function silently falls back to `[SupportedNetworkIds.PLUME_MAINNET]`, a mainnet, while `src/features/port/constants.ts:6` independently declares `supportedNetworks = [SupportedNetworkIds.ETHEREUM_SEPOLIA]`. Relatedly, `context.chainId = Number(process.env.NEXT_PUBLIC_DEFAULT_CHAIN_ID)` yields NaN when unset.

Impact. (a) A campaign banner displays inversely to its documented configuration — cosmetic. (b) A deployment that follows .env.example configures nothing, and the app silently defaults to a mainnet chain list rather than failing loudly. No direct security impact, but silent mainnet defaults on a money-moving app are the kind of drift that produces a wrong-chain incident later.

Recommendation. (a) Change to `=== 'true'`. (b) Reconcile .env.example with the variables the code reads, delete the unused `NEXT_PUBLIC_SUPPORTED_MARKETS`, and make `getSupportedChains()` throw on missing configuration instead of defaulting to a mainnet.

> PROOF OF CONCEPT — EXECUTED Read both config modules and both .env.example files; confirmed by grep that `NEXT_PUBLIC_SUPPORTED_MARKETS` appears only in .env.example and `NEXT_PUBLIC_SUPPORTED_NETWORKS` only in init.ts.
>
> config/index.ts:65-66 `showOptimismCampaignBanner: PUB_SHOW_OPTIMISM_CAMPAIGN === 'true', showRFLRCampaignBanner: PUB_SHOW_RFLR_CAMPAIGN === 'false',`. init.ts:12 `return [SupportedNetworkIds.PLUME_MAINNET]`.

01 Wallet Signing Review — what the screen says versus what is signed

For every action in the application that asks a user to sign, we recorded what the interface promised and what the wallet was actually asked to authorise, then compared the two. **29 actions reviewed — 17 gap · 3 mismatch · 7 match · 2 undisclosed**. Actions where the two agree are listed as well, so the table shows the whole surface rather than only the problems.

ACTION	WHAT THE SCREEN PROMISED	WHAT THE WALLET WAS ASKED TO SIGN	VERDICT
[redacted] dApp / Vault / user Deposit - 'Deposit' button	Modal heading 'DEPOSIT'. 'Available to Deposit: <wallet balance> <symbol>'. Token selector. Percentage chips 25/50/75/100. 'You Receive: <N> <vault symbol>'.	Up to two eth_sendTransaction. (1) conditional ERC20 approve; (2) Teller deposit.	GAP

	'Exchange Rate: 1 <vault symbol> -> <currentNav> <token>'. No fee row, no slippage row, no minimum row, no share-lock warning. Success banner 'Deposit successful!'.		
[redacted] dApp / Vault / user Withdraw request - 'Withdraw' button	Modal heading 'WITHDRAW'. 'Available to Withdraw: <balance> <vault symbol>'. 'Burn <symbol> <amount>'. 'You Receive: <Y> <base token>'. 'Exchange Rate: 1 <symbol> -> <nav>'. 'Minimum Received: <Y> <base token>' with tooltip "Minimum <token> you'll receive after slippage (0.5%)". 'Request Deadline: 37D'. Success banner 'Withdrawal Request Submitted!'.	Up to two eth_sendTransaction. (1) conditional ERC20 approve of the vault share token to the AtomicQueue; (2) AtomicQueue updateAtomicRequest.	GAP
[redacted] dApp / Vault / user Withdraw on a vault with no front-end constants entry	Same withdraw modal, except 'Request Deadline: 0D' and 'Minimum Received' computed with an undefined slippage. Success banner 'Withdrawal Request Submitted!' and a borrower email stating 'Notice Period: 0 days'.	approve(atomicQueue, offerAmount) + updateAtomicRequest(...) with an already-expired deadline.	MISMATCH
[redacted] dApp / Vault / user Cancel withdrawal - 'Cancel' button	Modal heading 'CANCEL WITHDRAWAL'. 'Pending Withdrawal: <offerAmount> <vault symbol>'. Success banner 'Withdrawal cancelled successfully!'.	One eth_sendTransaction: AtomicQueue updateAtomicRequest.	MATCH
[redacted] dApp / Vault / user 'Update Request'	Same withdraw modal plus a 'Pending Withdrawal: <X>' panel. Button 'Update Request'. 'Request Deadline: 37D'.	approve (if needed) + updateAtomicRequest with the new offerAmount.	GAP
[redacted] dApp / Vault / manager 'Withdraw' (manager withdrawal of vault assets)	Manager withdraw modal: token selector, amount, 'Withdraw'.	One eth_sendTransaction to the ManagerWithMerkleVerification contract.	GAP
[redacted] dApp / Vault / manager 'Process Withdrawals' (redeemSolve)	Withdrawal-requests table with the selected users and amounts, and a rates tab; 'Process' action.	Up to two eth_sendTransaction: an ERC20 approve to the solver, then AtomicSolverV3 redeemSolve.	GAP
[redacted] dApp / Vault / manager p2pSolve	Withdrawal-processing screen showing the users and the want amount.	approve(solver, wantAmount) + AtomicSolverV3 p2pSolve.	GAP

[redacted] dApp / Vault / manager 'Claim' (accrued fees)	AccruedFees panel showing the fee figure and a single button labelled 'Claim'. Success alert 'Claimed fees successfully'.	Up to THREE eth_sendTransaction from one button press.	UNDISCLOSED
[redacted] dApp / Vault / manager 'Pause Deposits & Withdrawal Processing' / 'Resume ...'	One status pill reading 'Active' or 'Paused' and one button labelled 'Pause Deposits & Withdrawal Processing' (or 'Resume ...').	PAUSE: one eth_sendTransaction. RESUME: two, issued concurrently.	GAP
[redacted] dApp / Vault / manager 'Update NAV' / 'Update APR' / 'Update Deposit Cap'	Each modal shows the current value and the new value side by side ('New Exchange Rate:', 'New Borrowing Rate:', 'New Deposit Cap:') before the confirm button.	One eth_sendTransaction each: Accountant.updateExchangeRate(uint96), Accountant.setLendingRate(...), Teller.setDepositCap(uint256).	MATCH
[redacted] dApp / Vault / manager 'Divert Funds' / 'Revert Funds' (Aave / Compound / [redacted] strategies)	Two-step modal: pick a strategy, enter an amount; the screen names the protocol and the amount.	One or two eth_sendTransaction, each wrapped as ManagerWithMerkleVerification.manageVaultWithMerkleVerification.	GAP
[redacted] dApp / Admin panel login	Admin login screen: connect wallet, then sign to authenticate. Statement rendered in the wallet: 'Authenticate to Administration Panel'.	One personal_sign (viem walletClient.signMessage) of a SIWE-formatted string.	GAP
[redacted] dApp / any action with a Safe (multisig) wallet on Flare or Base	Same modals; a 'pending in Safe' badge and an 'open in Safe' link are promised by ButtonWithSafeStatus.	The wallet returns a safeTxHash rather than a transaction hash.	MISMATCH
[redacted] dApp / Vaults / Deposit - 'Approve & Deposit'	Section title 'Deposit USDC'. 'Available to deposit: <min(wallet balance, cap headroom)> <symbol>'. Amount field with 25/50/75/100% chips. Button 'Approve & Deposit'. Validation errors 'Min deposit: X' and 'Exceed available amount'.	Up to two eth_sendTransaction via the private @[redacted]-finance/vaults-sdk: approve then supply.	GAP
[redacted] dApp / Vaults / Redeem	'Pool Position', 'Available to redeem', an amount field with chips, button 'Redeem'.	One eth_sendTransaction: redeem(pool, amount).	GAP
[redacted] dApp / [redacted] / 'Repay Pool' (and Repay Bullet Pool)	'Total Due: <X>' and 'Wallet Balance: <Y>' panels; button 'Repay Pool'. Nothing about an allowance.	Two eth_sendTransaction: approveTransfer then repayAll.	GAP

[redacted] dApp / Bridge - approve step	Bridge page with a source chain, destination chain and amount.	One eth_sendTransaction: approveTransfer(tokenAddr, MaxUint256).	GAP
[redacted] dApp / Staking / Stake	Stake modal with an amount field and a Stake button.	approve(amount) then stake(...) via the staking SDK.	MATCH
[redacted] dApp / Auction / Place Bid (and Increase Bid)	Bid modal with the bid amount and the auction details.	approve(<pool.asset>, <auction contract>, amount) then bid(<pool.id>, amount).	MATCH
[redacted] dApp / wrong-network state (all modals)	When the wallet is on the wrong chain the submit button is replaced by 'Change network' and an alert reads 'Change wallet network to <chain> to deposit'.	No transaction is constructed until the chain matches.	MATCH
[redacted] dApp / Connect + verify wallet (GemWallet / Crossmark / WalletConnect)	Wallet picker, then a toast asking the user to sign to verify the wallet; the site is vaults.[redacted].	One signMessage / xrpl_signMessage of a bare UTF-8 string.	GAP
[redacted] dApp / Admin sign-in	Admin login page, 'Connect the admin wallet to authenticate'.	One signMessage of 'VAULT_ADMIN_AUTH:v1:<nonce>:<exp>'.	GAP
[redacted] dApp / Verify wallet via [redacted]	'[redacted] - approve the SignIn in your app to verify this wallet...'	A [redacted] payload of TransactionType 'SignIn' carrying the challenge in Memos.	MATCH
[redacted] dApp / Deposit via GemWallet	'You Deposit <N> XRP', a USD equivalent, and the deposit address + destination tag panel.	GemWalletApi.sendPayment -> an XRPL Payment.	MATCH
[redacted] dApp / Deposit via Crossmark	'You Deposit <N> XRP'. Toast on completion.	crossmark.async.signAndSubmitAndWait -> an XRPL Payment.	GAP
[redacted] dApp / Deposit via WalletConnect / Girin	'You Deposit <N> XRP'. On completion: 'Deposit signed & submitted via WalletConnect!', the amount field is cleared and the balance is refreshed after 5 s.	wcSignClient.request({ method: 'xrpl_signTransaction', chainId: 'xrpl:1', params: { tx_json: Payment{...} } }).	MISMATCH
[redacted] dApp / Withdraw	Tab switches to 'You Withdraw'; the amount field is validated against the on-screen balance; a manual/read-only connection is refused with 'Withdrawals require a verified wallet connection (GemWallet, Crossmark, [redacted], or WalletConnect)'. Success toast: 'Withdrawal requested: <N> XRP - processable after <date>'.	NOTHING. No wallet method is invoked on this path at all.	UNDISCLOSED

[redacted] dApp / Admin 'On-Chain' protocol config save	Per-field 'On-Chain' button beside each protocol setting; helper text 'For production changes: use the on-chain button per field (GemWallet).'	A GemWallet payment carrying a hex-encoded 'config_update' memo.	GAP

06 Action Points

Findings at Medium severity and above, with the change we recommend. The status column is for your team to track remediation; we re-check these on a follow-up review.

ID	SEVERITY	POTENTIAL SOLUTION	STATUS
PROTO-P0008	High	Bind the origin into the signed material: use full SIWE (or EIP-4361-style text) including domain, uri, issuedAt and expirationTime, and validate domain/scheme server-side against an allow-list. In [redacted] dApp, generate a fresh nonce per login attempt and delete it on first use (or on failure) instead of returning the cached cookie value; give the session its own short-lived, revocable token rather than re-verifying a stored signature on every request.	
PROTO-P0092	High	Require a per-withdrawal signature over a server-issued nonce that names the address, the amount and the origin, or issue the user a short-lived session token at verify-wallet time and require it here. Make the 'verified' flag expire. Also allow the owner to cancel a pending request.	
PROTO-P0171	High	Bump next to the fixed patch line in both Next apps, bump axios past 1.17.0 in the SvelteKit app, and add lockfile overrides/resolutions for tar, form-data, nanoid, postcss, brace-expansion and shell-quote. Wire `npm audit --audit-level=high` / `pnpm audit` into CI so the lockfile cannot drift back.	
PROTO-R01	High	Authenticate the handler before it sends: gate `[redacted] dApp/src/app/api/send-email/route.ts` on a server-side shared secret compared with `crypto.timingSafeEqual` (or on an authenticated session), as the SvelteKit sibling already does, and add per-IP and per-recipient rate limiting. Stop letting the client choose the recipient -- resolve `to` server-side from the vault config keyed by an id in the body -- and HTML-escape every `values.*` field in `getHtml` instead of interpolating raw. The escaping is needed independently of the auth fix, since the rendered output goes out under [redacted] branding.	
PROTO-R02	High	Make the missing secret fatal: in `[redacted] dApp/lib/auth/admin-session.js:11-22`, throw at module load when `ADMIN_SESSION_SECRET` is unset and `isProdRuntime()`, rather than logging and falling through to `'dev-only-insecure-session-secret-change-me'`. Remove the fallback literal from the source tree altogether -- generate a random value into `.env.local` for	

		development -- so the string that HMACs every admin Bearer token is never a published constant. Rotate the secret on deploy and invalidate outstanding sessions, since any token minted under the default is currently valid.
PROTO-R05	High	Stop authorising on `Referer`: replace the four `referer.includes(...)` early returns in `[redacted] dApp/src/app/api/[redacted]/route.ts` with a verified credential, and close the escalation first -- `isServerRequest(req)` tests only for the presence of `x-[redacted]-token` while `validateRequest` accepts a correct one or else falls through to the Referer branch, so a bogus token plus a spoofed Referer is forwarded upstream carrying `SUBSQUID_MASTER_TOKEN`. Make attachment of the master token conditional on the token's *value*, not on header presence, and delete the preview wildcard that matches any host containing `-[redacted].vercel.app`. If the proxy must remain reachable from browsers, give it its own scoped read-only [redacted] token and never attach the master one to a browser-originated request.
PROTO-R07	High	Take `supertest` out of the request path: move it to `devDependencies` and have `[redacted] dApp/app/api/[[...path]]/route.ts` invoke the Express app directly (a `createServer(app)`-backed handler, or port the routes to native Next handlers) instead of replaying every request through `request(app)`. Fix the header trust independently -- `express-app.js:115-121` keys every limiter, `adminAuthStrictLimiter` included, on the leftmost client-supplied `X-Forwarded-For` with `validate: { trustProxy: false, xForwardedForHeader: false }`. Set `app.set('trust proxy', 1)` and key on the resolved `req.ip` or the platform's own client-IP header, and stop blanket-forwarding the client's `X-Forwarded-For` across the boundary.
PROTO-N01	High	Make isServerRequest() a value comparison (constant-time) against OZEAN_ACCESS_TOKEN, not a presence test, and derive it from the same result validateRequest already computed rather than re-deriving it. Drop Referer authorisation entirely in favour of exact-host Origin allow-listing or a signed token, and rate-limit the anonymous path.
PROTO-N02	High	Delete limiterKeyGen's header parsing and use req.ip with a correctly configured `trust proxy` hop count for the actual deployment, or take the platform-verified client IP (x-vercel-forwarded-for / the request's connection info) inside the Next.js route and pass it in a header the client cannot set, stripping any client-supplied x-forwarded-for at the entrypoint. Re-enable express-rate-limit's validators.
PROTO-01	High	Gate the route: require the SIWE session cookie the admin routes already use, or bind it to the depositing wallet, and confirm server-side that `to` is the address that wallet registered rather than accepting it from the body. HTML-escape every `values.*` interpolation (or move to [redacted] dynamic templates so the body is not assembled from request data at all). Restrict `template` to the Templates enum and

		delete the `default` branch that reflects `JSON.stringify(values)`. Add per-IP and per-address rate limiting.
PROTO-02	High	Either stop presenting an unenforced number as a guarantee — relabel to 'Estimated proceeds at the current rate; the final rate is set when your request is solved' and drop the tooltip's '(0.5%)' — or move the protection on-chain. If the AtomicQueue deployment has an overload carrying `atomicPrice`, use it and derive the price from the displayed minimum. Also delete or wire up `withdrawalSlippage`: a field that is 0 in every config and read by one display line is worse than no field.
PROTO-03	High	Pin every remote script to an exact version AND an `integrity="sha384-..."` hash with `crossorigin="anonymous"`. For the two esm.sh dynamic imports, either pin the exact patch version or — better, given these run in a signing context — vendor them into public/ and serve them same-origin, as the repo already does for xumm-browser.min.js in public/vendor/. Then tighten script-src to `'self'` and drop `'unsafe-eval'`.
PROTO-R03	Medium	Move the policy onto the surface that serves HTML: set `headers()` in `[redacted]dApp/next.config.ts` (or `vercel.json`) so the CSP and companion headers from `lib/http/csp.js` also cover the static documents in `public/` reached by the `redirect()` stubs, not only requests routed through `app/api/[[...path]]/route.ts`. Add the same block to `[redacted]dApp/next.config.ts`, which sets no headers at all. Then tighten the policy itself: drop `'unsafe-inline'` and `'unsafe-eval'` by moving the inline scripts in `admin.html`/`index.html` to nonced or bundled files, and remove the `unpkg`/`jsdelivr`/`esm.sh` origins by vendoring those dependencies.
PROTO-R04	Medium	Sanitize the one backend-fed sink: run `borrowerInfo?.description` through DOMPurify -- or render it as text -- at `[redacted]dApp/src/routes/(products)/borrowers/[slug]/+page.svelte:317`, since `info` arrives from the borrowers API rather than a repo literal. Add a `script-src` directive to the SvelteKit CSP, which is currently only `frame-ancestors 'none'` and so offers no second line of defence. `[redacted]Banner/index.svelte:42` and `__v2/Tooltip/index.svelte:97` are safe only by virtue of their present callers; sanitize inside those components too, so a future caller cannot introduce the sink by passing dynamic data.
PROTO-R08	Medium	Publish `/.well-known/security.txt` with a contact and an expiry in all three repos. Pass `domain` and `scheme` into `validateSiweMessage` at `[redacted]dApp/src/shared/utils/auth.ts:17-27`, and make the nonce single-use: `/api/admin/nonce/route.ts:10-14` re-serves the same cookie value for the full `MAX_AGE` and nothing ever deletes it, so generate a fresh nonce per challenge, store it server-side against the address, and delete it on first verification -- otherwise one captured admin signature stays a 30-day bearer credential accepted from any origin. Set `eslint.ignoreDuringBuilds = false` and

		`typescript.ignoreBuildErrors = false` in `[redacted] dApp/next.config.ts` (and re-enable eslint in `[redacted] dApp/next.config.ts`), then fix the resulting errors rather than shipping with the checks disabled.
PROTO-N03	Medium	Read the status: `if (receipt.status !== 'success') throw new TransactionRevertedError(hash)` in waitForTransactionReceipt, and `if (!transaction.isSuccessful) throw ...` in waitForTransactionExecution. Move the email send behind the same assertion.
PROTO-N04	Medium	Assign and inspect the response: require a signed result (and, if the wallet returns a blob rather than submitting, submit it via the XRPL client and check engine_result) before showing success; treat a rejection as an error toast. Derive the CAIP chain from the same XRPL_NETWORK the backend reports through /api/vault/info instead of hardcoding xrpl:1, and show the active network on the deposit card.
PROTO-N05	Medium	Move the header set into next.config.ts headers() (source '/(.*)') or a vercel.json headers block so it applies to static documents, keeping lib/http/csp.js as the single source of the policy string. While doing so, drop 'unsafe-eval' and tighten 'unsafe-inline' with hashes or a nonce, and pin the unpkg script with an integrity attribute.
PROTO-N06	Medium	Read noticePeriod/withdrawalDeadline from chain (or from the indexer alongside the vault) rather than a bundled table; refuse to build a withdrawal when the configuration is missing instead of defaulting to zero; apply the hasVaultConstants filter in useGetPortVaults as it is already applied in useGetPortAllVaults; and either implement the 0.5% slippage claim or delete it from the tooltip.
PROTO-N07	Medium	Compute the minimum from the accountant's live getRate at signing time, apply an explicit tolerance the user can see and change, and pass it as _minimumMint / minimumAssetsOut / minOfferReceived. Bound maxOfferReceived by the allowance rather than by maxUint256. Show the tolerance on the screen next to 'You Receive'.
PROTO-N08	Medium	Approve the exact amount required (the app already does this correctly on the Stake and PlaceBid paths, so the pattern exists in-repo). Where a 'max' semantic is genuinely wanted, show it as such on screen ('Approve unlimited' / 'Deposit entire balance') and make it an explicit user choice. Replace the Number()-based equality in Redeem with a BigNumber comparison, as Deposit's checkEqualValue already does.
PROTO-N09	Medium	Make the control state-symmetric: pause both contracts, or expose them as two controls with two status indicators. Issue the transactions sequentially and await each, and derive the displayed status from both contracts explicitly ('Deposits paused / Accounting live') rather than from a boolean OR.
PROTO-N10	Medium	Add entries for chains 14 and 8453 to both Safe maps, or gate Safe support explicitly and tell the user when the connected chain is unsupported. Replace the blanket `catch { return false }` in

		isSafeWallet with a distinction between 'not a Safe' and 'could not determine', and fail loudly on the latter. Give waitForTransactionReceipt a timeout.
PROTO-N11	**Medium**	Show the operation as what it is: list the steps, the approval amount and the before/after exchange rate in the modal, and require confirmation of the NAV change with the same disclosure the dedicated Update NAV modal already provides. Approve the fee amount rather than the balance.
PROTO-04	**Medium**	In `waitForTransactionReceipt`, after resolving, `if (receipt.status === 'reverted') throw new TransactionRevertedError(hash)`. This is a single-point fix that corrects all 16 call sites at once. Do the same for the Safe branch, which returns `this.sdk.safe.waitForTransactionExecution(hash)` without a status check either.
PROTO-05	**Medium**	Make `isServerRequest` compare the header to OZEAN_ACCESS_TOKEN with a constant-time comparison and use that single result for both the auth decision and the master-token decision — never presence. Replace the referer substring tests with exact origin equality against an allowlist, read from the `Origin` header, and treat a missing/mismatched origin as unauthorised.
PROTO-06	**Medium**	Approve the exact required amount, as the sibling __v2/Modals/Deposit.svelte:98 already does (`sdk.approve(_poolAddress, amount)` with the parsed amount). If an unlimited allowance is a deliberate UX choice, surface it: a distinct 'Approve' step naming the spender and stating 'unlimited', with an exact-amount alternative. For the MAX path, either pass the exact parsed balance or, if the MaxUint256 sentinel is genuinely required by the contract, change the displayed copy to 'Deposit entire balance' so the figure shown and the figure signed agree.
PROTO-07	**Medium**	Derive `_minimumMint` from the displayed `receiveAmount` less an explicit, user-visible tolerance, and pass it. If a zero minimum is intentional, remove the concrete 'You Receive' figure or label it clearly as an unguaranteed estimate. Note the same `0` appears as `minimumAssetsOut` in `redeemSolve` (port.ts:424) and `minOfferReceived` in `p2pSolve` (port.ts:365) — those are manager-only paths, but they deserve the same review.
PROTO-08	**Medium**	For WalletConnect, either switch to the submitting method for the connected namespace or take the signed blob from the response and submit it via the app's own XRPL client, then wait for validation. For every branch, resolve the transaction on-ledger and require `meta.TransactionResult === 'tesSUCCESS'` before showing a success toast — the server already applies exactly this check at lib/services/xrpl-service.js:302 (`if (tr !== "tesSUCCESS") throw new Error("Transaction was not successful")`), so reuse that standard on the client. Replace the Crossmark else-branch with an error toast.

PROTO-09	**Medium**	Do not let a missing config produce a zero deadline. Either refuse to render the withdraw action when `hasVaultConstants(vault)` is false (the helper already exists for this and is unused), or fall back to the SDK's 888-hour default by passing `undefined` rather than a computed 0. Better still, source `noticePeriod` and `withdrawalDeadline` from the indexer alongside the vault rather than from a name-keyed local map, and drop `name` from the key so a display rename cannot break the lookup.
PROTO-10	**Medium**	Authorise the caller, not the address. The codebase already has the pieces: `lib/auth/admin-session.js` issues HMAC bearer sessions and `generateNonceRecord('user')` / `consumeNonce` implement a user-purpose challenge. Require a valid user session bound to `address`, or require a fresh signed challenge on the withdrawal request itself, and reject when the session subject and the body's `address` disagree.
