ASSURE DEFI®

THE VERIFICATION GOLD STANDARD

SECURITY ASSESSMENT REPORT

NAME:	RIFTS PROTOCOL
STATUS:	PASS
DATE:	04/01/2025

®

Risk Analysis

Vulnerability summary

Classification	Description
High	Vulnerabilities that lead to direct compromise of critical assets, large-scale data exposure, unauthorized fund transfers, or full system takeover.
Medium	Flaws that weaken security posture or privacy but do not immediately enable catastrophic failures.
Low	Issues that have minimal direct impact, often involving best-practice deviations or potential future risks.
Informational	Observations, style concerns, or suggestions that do not constitute vulnerabilities but may improve security hygiene.

SCOPE

Target Code And Revision

Project	Assure
Codebase	https://github.com/riftsprotocol/rifts-protocol-v2 Commit: 756ef8054bf719cd4c19b0d808f35a9034be75e9 Commit fix: 2dff7680ea57d6f48119da1dc84ab357adf8a483
Audit Methodology	Static, Manual

Detailed Technical Report

1. Hardcoded Supabase service-role key in source [Fixed ✅]

Location:

src/app/api/arb-bot/route.ts:12

Issue:

SUPABASE_SERVICE_KEY falls back to a literal JWT string (service-role style) embedded in the repo.

Remediation:

Remove hardcoded secrets immediately. Rotate the Supabase service-role key in Supabase now. Enforce “no-default-secret” pattern: fail startup if the env var is missing. Add secret scanning (GitHub secret scanning / gitleaks) in CI.

Fix: removed hardcoded fallback; now reads from env and fails in supabaseFetch if missing.

2. Unauthenticated write endpoint using Supabase service role [Partially Fixed ✅]

Location:

src/app/api/save-rifts/route.ts:30–46

Issue:

POST /api/save-rifts upserts arbitrary rifts using service role (upsert(..., { onConflict: 'id' })) with no authn/authz.

Remediation:

Require strong authentication (admin token / OIDC / signed admin JWT). Authorize per action. Consider moving this functionality to a private admin job/cron, not a public route. If it must exist, enforce mTLS or a high-entropy shared secret in an Authorization header and rate-limit.

Fix: Added an auth gate, but it still accepts legacy admin auth based only on a caller-supplied wallet == ADMIN_WALLET (no signature proof). Where:

src/lib/middleware/api-auth.ts → legacy wallet-only branch still returns authenticated

src/app/api/save-rifts/route.ts

src/app/api/update-rift-pool/route.ts

src/pages/api/backfill-metrics.ts

3. Auth bypass in arb-lp-sync when wallet omitted [Fixed ✅]

Location:

src/app/api/update-rift-pool/route.ts:30+

Issue:

Updates pool address in rifts by reading and rewriting raw_data with service role, no auth.

Remediation:

Same as 2: authenticate + authorize + validate inputs (base58 pubkeys, expected pool types).

Fix: now requires authentication, missing wallet no longer grants access.

4. Unauthenticated admin maintenance endpoint using service role [Partially Fixed ✅]

Location:

src/pages/api/backfill-metrics.ts:5–8, 14+]

Issue:

POST handler performs destructive cleanup/fixes with service role client and no protection.

Remediation:

Restrict to internal execution only (cron with secret, Vercel protection, or remove from public surface). Add mandatory auth middleware.

Fix: Added isAuthenticatedForAdminOp() gate (403 if not authenticated). Still bypassable via legacy "wallet == ADMIN_WALLET" because api-auth.ts keeps admin-legacy auth without signature.

5. Auth bypass: wallet-based "auth" allows unauthenticated access [Fixed ✅]

Location:

src/app/api/arb-lp-sync/route.ts:292–301

Issue:

POST rejects only when wallet && wallet !== ADMIN_WALLET. If wallet is omitted, the request is allowed.

Remediation:

Require authentication unconditionally. Do not use "presence/absence of wallet" as auth.

Fix: Replaced the "wallet optional" logic with isAuthenticatedForAdminOp() check. Missing wallet no longer grants access; request is rejected unless cron secret / auth header / admin auth is present.

6. Admin impersonation via wallet string (no signature verification) [Fixed ✅]

Location:

src/app/api/arb-bot/route.ts:969–976

Issue:

checkRiftPermission() grants admin privileges if walletAddress === ADMIN_WALLET, but wallet ownership is never proven (no signed message / SIWS).

Remediation:

Replace wallet-string checks with cryptographic proof: Require a signed nonce (for example Sign-In With Solana pattern) and verify signature server-side. Bind session/token to wallet after verification, then enforce RBAC.

Fix: If caller claims ADMIN_WALLET, endpoint now requires signature + timestamp and verifies via verifyAdminAuth(). Wallet-string impersonation no longer works for admin actions on this route.

7. Admin actions protected only by caller-supplied wallet string [Fixed ✅]

Location:

src/app/api/arb-profits/route.ts:534–536, 788–795

Issue:

Admin view and admin actions (including reset/update-config/distribution flows) are gated by wallet === ADMIN_WALLET from query/body.

Remediation:

Proper admin authentication (server-validated token/OIDC) + authorization. Do not trust user-supplied wallet as identity.

Fix: Moved to centralized auth (isAuthenticatedForAdminOp()), but still allows legacy admin wallet-only auth when wallet == ADMIN_WALLET and no signature is provided.

8. CRON secret can be bypassed when unset [Fixed ✅]

Location:

src/app/api/lp-earnings-sync/route.ts:20–22, 121–127 and similar patterns

Issue:

CRON_SECRET defaults to ''. Code treats authHeader === \Bearer ${CRON_SECRET}`as “Vercel cron”. If secret is empty,Authorization: Bearer ` can satisfy the check.

Remediation:

Fail hard if CRON_SECRET is missing/empty. Don’t accept “Bearer ${secret}” checks unless secret.length >= 32 and validated in constant-time compare.

Fix: Added MIN_CRON_SECRET_LENGTH and isCronSecretValid() checks so empty/short CRON_SECRET can’t authenticate (Bearer/param checks fail closed).

Note: endpoints that accept wallet can still be reached via legacy admin wallet-only auth unless you remove that fallback.

9. Hardcoded Helius API key exposed to clients [Fixed ✅]

Location:

src/components/liquidity/LiquidityModal.tsx:1056

Issue:

Client code instantiates Connection('https://mainnet.helius-rpc.com/?api-key=...').

Remediation:

Remove from client bundle. Route RPC through your server proxy (/api/rpc / /api/rpc-http) and keep provider keys server-side. If you must expose a key, use a restricted/low-privilege key with strict allowlists and rate limits (vendor permitting).

Fix: No hardcoded helius-rpc API key found in client components anymore; current code builds RPC URLs from env/server config instead.

10. PostgREST query injection / parameter smuggling via unencoded interpolation [Fixed ✅]

Location:

src/app/api/referrals/route.ts:62–85

Issue:

User-controlled wallet is concatenated into Supabase REST URL without encodeURIComponent or strict validation: ...referrer_wallet=eq.${wallet}&select=*

Remediation:

Validate wallet format strictly (base58 + expected length) and always encodeURIComponent(wallet) when building URLs. Prefer Supabase client query builders where possible.

Fix: Added wallet format validation (isValidWalletAddress) and encodeURIComponent(wallet) before embedding into Supabase REST URLs.

11. Unauthenticated bulk stats endpoint (data exposure + DoS risk) [Partially Fixed ✅]

Location:

src/app/api/arb-bot/stats/route.ts:75+

Issue:

Endpoint aggregates large datasets (noted “7000+ rows”) without authentication/authorization or strict pagination, enabling scraping and resource exhaustion.

Remediation:

Require auth (admin or scoped user). Add pagination + hard upper bounds, caching, and rate limiting (per IP + per identity). Return minimal fields by default.

Fix: Added basic in-memory rate limiting + capped limit to 100. Still unauthenticated (data still publicly accessible), so the main exposure risk remains.

12. Public cache-warmer endpoint can be abused for DoS [Partially Fixed ✅]

Location:

src/app/api/warm-cache/route.ts:16+

Issue:

Triggers expensive upstream calls/cache population without auth, allowing repeated requests to consume compute/RPC quota.

Remediation:

Restrict to internal/cron-only (strong secret, fail closed if missing). Add rate limits, idempotency, and max-work caps (for example, only warm N items per invocation).

Fix: Added isAuthenticatedForAdminOp() guard. Still bypassable via legacy admin wallet-only auth (no signature required).

13. CSP allows unsafe script execution paths [Acknowledge]

Location:

next.config.js:145–150

Issue:

script-src includes 'unsafe-inline' and 'unsafe-eval', and CSP includes http: in some directives, reducing XSS mitigation strength and increasing mixed-content exposure.

Remediation:

Remove unsafe-eval/unsafe-inline where possible (use nonces/hashes). Restrict to https: only. Minimize allowed domains to strict necessities. Validate with CSP evaluator and test in Report-Only first.

Note: CSP still includes 'unsafe-inline', 'unsafe-eval', and allows some http: sources. No hardening change detected.

14. Install command bypasses dependency safety checks [Partially Fixed ✅]

Location:

[vercel.json (installCommand)]

Issue:

npm install --legacy-peer-deps --force can pull unexpected dependency trees and suppress install-time safeguards, increasing supply-chain risk and reducing reproducibility.

Remediation:

Remove --force, resolve peer conflicts properly, and enforce lockfile-based installs in CI like, npm ci / pnpm --frozen-lockfile / yarn --immutable).

Fix: --force was removed. Install still uses --legacy-peer-deps, so some supply-chain/reproducibility risk remains.

15. Dual lockfiles increase build inconsistency risk [Fixed ✅]

Location:

[repo root: yarn.lock + package-lock.json]

Issue:

Having multiple lockfiles can result in different dependency graphs across environments (CI vs local vs Vercel), complicating patching and vulnerability management.

Remediation:

Standardize on one package manager, delete the other lockfile, and enforce via CI checks.

Fix: Only package-lock.json is present; yarn.lock is not in the root anymore.

No medium issues were found.

1. Origin/Referer-based CSRF checks are defense-in-depth only [Acknowledge]

Location:

src/lib/middleware/csrf-protection.ts

Issue:

Reliance on Origin/Referer allowlists is helpful for browser CSRF mitigation but should not be treated as primary protection for privileged or server-to-server routes (non-browser clients can bypass browser CSRF assumptions).

Remediation:

Keep as defense-in-depth, but ensure privileged routes rely on real authentication + authorization. For cookie-based sessions, prefer CSRF tokens (double-submit or synchronizer token) where appropriate.

2. Rate limiting may be vulnerable to IP spoofing depending on proxy trust [Acknowledge]

Location:

rate limit / request IP extraction codepath — deployment-dependent

Issue:

If rate limiting depends on x-forwarded-for without strict trust boundaries, clients can spoof IPs to evade throttling. Whether this is exploitable depends on your hosting/proxy behavior.

Remediation:

Use platform-provided “trusted client IP” headers (Vercel/edge runtime) and treat forwarded headers as trusted **only** when set by your reverse proxy. Prefer identity-based limits (API key/user/session) in addition to IP.

INFORMATIONAL

No informational issues were found.

Technical Findings Summary

Findings

Vulnerability Level	Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
High	15			1	4	9
Medium						
Low	2			2		
Informational						

Assessment Results

Score Results

Review	Score
Global Score	**85/100**
Assure KYC	Not completed
Audit Score	85/100

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 84 Points for a higher standard, if a project does not attain 85% is an automatic failure. Read our notes and final assessment below. The Global Score is a combination of the evaluations obtained between having or not having KYC and the type of contract audited together with its manual audit.

Audit PASS

The RIFTS DAPP presently fails to mitigate several critical vulnerabilities.
Until these high-risk issues are fully addressed, the RIFTS DAPP cannot be considered secure for production use. **After the development team's review, all critical vulnerabilities have been addressed/reviewed, and the audit results are satisfactory.**

Disclaimer

Assure Defi has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocating for the Project, and users relying on this report should not consider this as having any merit for financial Token in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies. All information provided in this report does not constitute financial or investment in Token, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided ‘as is, and Assure Defi is under no covenant to audit completeness, accuracy, or solidity of the contracts. In no event will Assure Defi or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report. The assessment of Tokens provided by Assure Defi are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any Tokens, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The Token may access, and depend upon, multiple layers of third parties.