Assure DeFi®

THE VERIFICATION **GOLD STANDARD**

VERIFIED BY ASSURE DEFI
INTEGRITY ★ TRUST ★ CREDIBILITY

Security Assessment

HedgeFi

Date: 16/06/2025

Audit Status: PASS

Audit Edition: Code audit

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
Language	Solidity
Codebase	https://github.com/degenomics2000/dex Commit: a4b4db9a593f4c1e6b902157d586897b52c50933 Fixed version Commit - 7688c23f8d6a64874389e1cd081462f35b40ed06
Audit Methodology	Static, Manual

Detailed Technical Report

1. Core Cryptographic Review

1.1 TOPT Generation [✅]

Location: src/utils/totp.ts

Issue: Uses crypto.getRandomValues and a BASE32 charset which is compliant with RFC 4226/6238.

Status: No issues.

1.2 JWT Signing & Verification [HIGH ISSUE - Fixed ✅]

Location: netlify/functions/2fa-secure.js

Issue: Retrieves PRIVY_APP_ID and MASTER_2FA_KEY from process.env, but falls back to an insecure default ('secure.master.key.here.change.this') if not set.

Risk: Attackers can bypass 2FA verification by guessing the default key.

Severity: High

Remediation:

Remove fallback: require MASTER_2FA_KEY be explicitly set for example:

```
if (!process.env.MASTER_2FA_KEY) {
  throw new Error('MASTER_2FA_KEY must be defined');
}
```

Rotate keys: ensure strong, high-entropy secrets (at least 256 bits).

Fix: The default fallback key was removed; the function now throws an error if MASTER_2FA_KEY is not explicitly set in the environment.

1.3 Algorithm Selection & IV Management [✅]

Location: inspect all uses of crypto.createCipher or manual AES constructions.

Findings: No use of deprecated MD5 or ECB mode. All symmetric ops use Node's default aes-256-cbc via standard libraries. IVs are randomly generated per encryption.

Status: No deprecated algorithms detected.

2. Smart-Contract Integration

2.1 ABI Encoding & Decimal Scaling [✅]

Location: src/services/ostium-protected.ts

Issue: Uses ethers.js parseUnits(amount, token.decimals) and formatUnits correctly. No manual string manipulations confirmed.

Status: Correct use of ABI encoding.

2.2 Slippage Protection [MEDIUM ISSUE 🟠 - Acknowledge]

Location: trade submission logic in src/components/SendModal.tsx

Issue: Slippage tolerance is hard-coded at 0.5% with no user override.

Risk: Market volatility could lead to unexpected execution or front-running losses.

Severity: Medium

Remediation: Expose a UI control for users to set maximum slippage; enforce constraints on-chain via maxSlippage parameter in the contract call.

2.3 Replay & Front-Running Mitigation [✅]

Observation: Transactions include nonces from signer; no custom replay protection needed beyond Ethereum mechanism.

Status: Adequate on-chain protection

3. Authentication & Secrets Management

3.1 VITE_ Prefixed Environment Variables [HIGH ISSUE - Fixed ✅]

Location: vite.config.ts and front-end code referencing import.meta.env.VITE_*

Issue: Any VITE_ variable is embedded in the client bundle. Inspect for secrets like VITE_PRIVY_APP_ID.

Risk: Secrets leak to end-user, enabling API misuse.

Severity: High

Remediation: Remove all sensitive keys from VITE_ namespace.

Proxy calls through serverless functions or Supabase with service-role keys.

Fix: No VITE_-prefixed secrets remain in the client bundle. All sensitive keys are loaded server-side via the new app-config endpoint.

3.2 Supabase RLS Policies [MEDIUM ISSUE 🟠 - Acknowledge]

Location: Supabase dashboard (not in repo)

Issue: Default policies allow the public role to read trades and users tables.

Risk: Unauthorized data exposure.

Severity: Medium

Remediation: Enforce row-level policies & restrict insert/update/delete similarly.

4. Serverless Proxies & APIs

4.1 CORS Configuration [HIGH ISSUE - Fixed ✅]

Location: netlify/functions/*.js

Issue: Uses Access-Control-Allow-Origin: '*'.

Risk: Any origin can interact, facilitating CSRF and data exfiltration.

Severity: High

Remediation [example]:

```
const ALLOWED_ORIGINS = ['https://app.hedgefi.com'];
const origin = event.headers.origin;
if (ALLOWED_ORIGINS.includes(origin)) {
  headers['Access-Control-Allow-Origin'] = origin;
}
```

Fix: All Netlify Functions now use a strict ALLOWED_ORIGINS list instead of '*', and respond with the incoming origin only if it matches the whitelist.

4.2 Rate Limiting [MEDIUM ISSUE - Fixed ✅]

Issue: No throttle on 2FA issuance or verification endpoints.

Risk: Brute-force OTP codes or exhaust SMS quotas.

Severity: Medium

Remediation:

Implement per-IP/user counters (for example in-memory or Redis) & return 429 Too Many Requests when thresholds exceeded.

Fix: Export-security.js now implements a in-memory per-user rate limiter.

4.3 Input Sanitization & CSRF [MEDIUM ISSUE - Acknowledge 🟠]

Findings:

All POST handlers parse JSON without schema validation (e.g. using AJV).

No CSRF tokens on API routes; relies solely on JWT in header.

Risk: Malformed requests may cause logic errors; CSRF if JWT is stored in a cookie.

Severity: Medium

Remediation:

Validate inputs against strict JSON schemas.

Use double-submit CSRF tokens or ensure JWT is sent via Authorization: Bearer.

5. Front-End Security Controls

5.1 Content Security Policy (CSP) [LOW - FIXED ✅]

Location: absent in index.html or server headers.

Risk: Permits arbitrary inline scripts/styles; XSS pods.

Severity: Low

Remediation:

Add CSP meta or Netlify header:

```
Content-Security-Policy:
  default-src 'self';
  script-src 'self';
  style-src 'self' 'unsafe-inline';
  connect-src 'self' https://api.hedgefi.com ws://localhost:3000;
```

Fix: public/_headers now ships a strict CSP (plus X-Frame-Options, Referrer-Policy...).

5.2 Dependency Audit

Next Steps:

Run npm audit and address all High/Moderate advisories and pin transitive dependencies or use npm audit fix.

Technical Findings Summary

Findings

Vulnerability Level	Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
🔴 High	3					3
🟠 Medium	4			3		1
🟡 Low	1					1
🟢 Informational	0					

Assessment Results

Score Results

Review	Score
Global Score	**90/100**
Assure KYC	Not completed
Audit Score	90/100

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 security audit did not pass. Multiple high-risk issues were identified across the decentralized exchange front-end and supporting infrastructure. Notably, insecure CORS configurations, hard-coded fallback secrets for 2FA, and the exposure of environment secrets in the client-side bundle pose serious security risks. Additionally, the lack of rate limiting on authentication endpoints, permissive Supabase Row-Level Security (RLS) policies, and absent CSRF protections leave the application vulnerable to abuse and unauthorized access. Several dependencies also include known CVEs, and the trading logic lacks adequate user-configurable slippage protection. These findings require remediation before the system can be considered production-grade secure.
Update: After review by the development team, all high vulnerabilities found are resolved. The audit is considered **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 adMetabot 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 adMetabot, 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 serMetabots 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 serMetabots, 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 serMetabots may access, and depend upon, multiple layers of third parties.