

Assure DeFi[®]

THE VERIFICATION **GOLD STANDARD**



Security Assessment

ONCHAIN BATTLES

Date: 29/07/2025

Audit Status: PASS





Audit Edition: Code audit



ASSURE DEFI[®]
THE VERIFICATION **GOLD STANDARD**

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/MetaDeckTrade/onchain-battles-backend Commit: 34feecdb21f2f2301ef3980140fbd453bec48c77 Fixes commit: 41873f316b4917fc2205412f7962b240d21b365a
Audit Methodology	Static, Manual

Detailed Technical Report



1. No per-user authorization - users can modify or join pools on behalf of any wallet [Fixed]



Location: controllers/user/UpdateProfile.js & controllers/user/JoinPool.js (invoked via /user/profile/update and /pool/join routes)

Issue: Both the profile-update and pool-join endpoints accept an arbitrary wallet parameter in the JSON body and use only the global BEARER key for authorization. There is no check that the incoming request is on behalf of the wallet in question. An attacker who obtains the BEARER token can update any user's profile, upload arbitrary avatars to other users, or add any wallet to a pool.

Remediation: Introduce a user-scoped authentication mechanism (like JWTs or sessions) that binds each request to a specific wallet address.

In the middleware, verify that the authenticated user's wallet matches the wallet in the request body.

Remove reliance on a single global BEARER token for user-level operations.

Fix: A ValidateJwt middleware now reads a JWT from a secure, HTTP-only cookie (req.cookies.token), verifies it, and populates req.wallet.

All protected user routes (like /user/profile/update, /pool/join) use ValidateJwt and pass only req.wallet into their controllers, so you can no longer spoof another user wallet in the request body.

2. Unrestricted file upload & static serving [malicious files can be uploaded and served]

[Fixed]

Location: utils/Multer.js & static serving in index.js (app.use("/avatars", express.static("avatars")))

Issue: Multer is configured without any file-type or size restrictions and stores uploads under ./avatars with the original filename. Coupled with express.static("avatars"), an attacker can upload executable scripts or HTML pages, then access them via /avatars/..., leading to XSS or remote code execution depending on your hosting environment.

Remediation: Restrict uploads to image MIME types (for example image/png, image/jpeg) and enforce a maximum file size.

Sanitize file names (for example you can use a UUID and strip any paths).

Consider scanning uploads for malware.

Serve uploads from a separate subdomain or with strict Content-Security-Policy headers to isolate them.

Fix: The Multer setup (middlewares/Multer.js) now enforces a fileFilter permitting only JPG/PNG/WEBP and a 10 MB size limit.

Uploaded filenames are prefixed with a UUID (uuidv4()), preventing path-traversal or predictable names.

3 Committed Google Cloud service account key in source control [Fixed ✓]

Location: Repository root: skyspacecloud-c4a472655352.json

Issue: A JSON key file for a GCP service account is checked into version control. If leaked, an attacker gains full programmatic access to your Google Cloud resources under that service account's permissions.

Remediation: Immediately revoke the exposed key in Google Cloud IAM.

Remove the JSON file from the repo history (for example with git-filter-repo) and add it to .gitignore.

Switch to a secrets manager (for example GCP Secret Manager) or environment-injected credentials that never touch source control.

Fix: The skyspacecloud-c4a472655352.json file has been removed from source control. (It no longer appears in the extracted directory.)



1. Insecure admin authentication via bearer token in request body [Fixed ✓]

Location: index.js — admin routes: /pool/register, /pool/status/update, /pools/clear, /interval/start, /interval/stop

Issue: For “admin” operations you rely on a bearer field in the JSON body compared against PRIVATE_BEARER. Transmitting sensitive tokens in the request body is non-standard, may be logged, and is more easily leaked (ex, via referer headers).

Remediation:

Move all bearer secrets into the standard Authorization: Bearer <token> HTTP header.

Enforce HTTPS to protect token in transit.

Use a more robust authentication scheme (ex JWTs with expiration + refresh).

Fix: Admin routes now use an AuthAdmin middleware that reads from the standard Authorization: Bearer <token> header and compares against process.env.PRIVATE_BEARER.

2. Custom Bearer header bypasses standard Authorization mechanisms [Fixed ✓]

Location: utils/Authorization.js

Issue: The middleware reads from req.get("Bearer") instead of the standard Authorization header. This non-standard header is easier to spoof and not supported by many libraries (for example automatic Swagger/OpenAPI tooling).

Remediation: Switch to reading req.get("Authorization"), parse out the Bearer <token> prefix, and verify the token.

Reject requests missing or malformed Authorization headers.

Fix: The Authorization middleware now does `req.get("Authorization")` and checks for "Bearer <token>" rather than a bespoke header.

3. No input validation or sanitization [risk of injection and malformed data] [Fixed

Location: `controllers/*` (all controllers)

Issue: Controllers accept arbitrary string/number inputs (name, limit, status, wallet, etc.) without enforcing type or format. This can lead to MongoDB operator injection (e.g. passing `{ $gt: "" }`), stored XSS (via unescaped pool names), or invalid data.

Remediation: Introduce a validation layer (use something like Joi or express-validator).

Define strict schemas: something like wallet must match `/^0x[a-fA-F0-9]{40}$/`, limit must be positive integer, status must be one of `['ACTIVE', 'INACTIVE']`, etc.

Reject requests with invalid or malicious payloads before hitting your controllers.

Fix: A suite of express-validator middlewares (`idValidator`, `nameValidator`, `limitValidator`, etc.) is applied to both user and admin endpoints, and invalid payloads are rejected before reaching controllers.

4. Uncontrolled scheduling [potential resource exhaustion and duplicate intervals] [Partially fixed

Location: `controllers/pnl/PnlInterval.js`

Issue: `StartInterval()` launches an interval and writes to the module-scoped interval variable, but does not prevent multiple calls; each call will spawn a new loop.

Inside each interval, you iterate pools and users sequentially; if a Moralis call hangs or scales up, intervals may stack or starve the event loop.

`PnlInterval()` does not return on success, so the initial `await PnlInterval()` in `StartInterval()` may mask failures.

Remediation: Guard `StartInterval()` to only allow one active interval at a time for example check if (interval) return error).

Return a consistent status object from `PnlInterval()` on success vs. error.

Introduce concurrency limits (`Promise.allSettled()` with batching) and timeouts for external API calls.

Consider a more robust scheduler (you could use BullMQ, Agenda) that persists job state and retry policies.

Fix: Added: A guard in `StartInterval()` prevents spawning multiple intervals.

Still missing: Concurrency limits (for example batching or `Promise.allSettled()`) and timeouts on each Moralis call, so long-running or hung requests could still pile up.



LOW

1. No rate limiting or brute-force protection [Acknowledge]

Location: index.js (entire application)

Issue: There is no middleware to throttle requests. An attacker could spam endpoints especially the PnL or registration routes to overload your service or brute-force tokens.

Remediation: Add express-rate-limit or an API gateway with per-IP and per-user throttling.

Block IPs exhibiting abnormal request patterns.

Fix: A global express-rate-limit instance (middlewares/RateLimiter.js) is applied via `app.use(rateLimiter)`, throttling all incoming requests.

2. Loose Mongoose schemas, no required, no stricter typing [Partially fixed ✓]

Location: models/UserModel.js & models/PoolModel.js

Issue: Schemas define all fields as bare types (for example String, Number) with no required, trim, or validations. This allows incomplete or malformed documents.

Remediation: Strengthen schemas with required: true, enum, min/max, validate, and trim where appropriate. For example you could do:

```
wallet: { type: String, required: true, match: /^0x[a-fA-F0-9]{40}$/ },
username: { type: String, required: true, minlength: 3, maxlength: 30, trim: true },
```

Fix: UserModel now has required: true and a match regex on the wallet field.

PoolModel has been marked required in some fields but lacks enum/match constraints and trim so malformed pool docs can still slip through.



INFORMATIONAL

No informational issues were found.

Technical Findings Summary

Findings

Vulnerability Level	Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
<div><div></div>High</div>	3					3
<div><div></div>Medium</div>	4				1	3
<div><div></div>Low</div>	2			1	1	
<div><div></div>Informational</div>	0					

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 ONCHAIN BATTLES presently fails to mitigate several critical vulnerabilities. Until these high-risk issues are fully addressed, the ONCHAIN BATTLES cannot be considered secure for production use.

[After the fixes, we inform you that the project has met the necessary security standards.](#)

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 AIToken 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 AIToken, 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 AITokens 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 AITokens, 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 AIToken may access, and depend upon, multiple layers of third parties.

