

Assure DeFi[®]

THE VERIFICATION **GOLD STANDARD**



Security Assessment

Numpay

Date: 25/07/2025

Audit Status: FAIL





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
Language	Solidity
Codebase	Cash Tide Backend.zip [SHA256] Commit: b6afed08ab354dc0aff4c783221c834e2947ec81abe4357cfd03fddd744bde32
Audit Methodology	Static, Manual

Detailed Technical Report



1. No authentication enforcement on protected routes

Location: src/server.ts lines 14–22 (all routes mounted without auth) for example user.route.ts line 27 (router.get("/get/:id"...)) and all other routes in src/routes/*.route.ts

Issue: You import and initialize @privy-io/server-auth (in services/privy/index.ts) but never actually use it to validate sessions or tokens. As a result any client can call “protected” endpoints (for example when fetching user data, sending transactions, approving requests) with no credential check.

Remediation: Add Privy authentication middleware (like app.use(privy.authenticate()) or per-route privy.authorize()) before your protected routes.

Enforce role-based or resource-based checks inside controllers (for example verify that req.user.privyDID === targetUser.privyDID for user-specific operations).

2. Overly permissive CORS policy

Location: src/server.ts, line 14

Issue: app.use(cors()) without any options sets Access-Control-Allow-Origin: *, exposing your API to requests from any origin.

Remediation: Configure CORS with an explicit allow-list:

```
app.use(cors({
  origin: ["https://your-frontend.com", "https://admin.your-frontend.com"]
}));
```

3. No rate limiting on OTP/SMS endpoints

Location: src/controllers/verification/verification.controller.ts, lines 17 (POST /api/verification/generate) and 18 (POST /api/verification/verify)

src/controllers/sms/sms.controller.ts, line 5 (POST /api/send-text)

Issue: Without a rate-limiter or CAPTCHA, attackers can brute-force verification codes or flood SMS endpoints.

Remediation: Use express-rate-limit (or similar) to cap requests per IP and per phone number. Optionally integrate CAPTCHA on the “generate” step or device fingerprinting to slow abuse.

4. Missing critical HTTP security headers

Location: src/server.ts

Issue: You never set headers like Content-Security-Policy, X-Content-Type-Options: nosniff, X-Frame-Options, or HSTS.

Remediation: Install and configure helmet



1. Missing critical HTTP security headers

Location: .gitignore

Issue: Your .gitignore omits any rule for .env, risking accidental commits of DB_URL, TWILIO_* secrets, Privy credentials, etc.

Remediation: Add at least:

```
# Environment
.env
```

2. Insufficient input validation/sanitization

Location: Various controllers, for example user.controller.ts line 8 (const userData = req.body), transaction.controller.ts, etc.

Issue: You only check for existence of parameters, never enforce types, formats, lengths, or whitelist fields leaving you open to malformed input or injection.

Remediation: Adopt a schema validation library (like Zod, Joi) at the route boundary to strictly validate and sanitize all incoming data.

3. Excessive logging of sensitive data

Location: src/controllers/users/user.controller.ts line 15: console.log("privy no", userData.phoneNumber)

src/services/twilio/index.ts line 12: console.log(sendOTP)

Issue: Logging PII (phone numbers) and OTP objects can leak data in logs.

Remediation: Remove or redact sensitive fields from logs; use structured logging with levels and redaction in production.

4. Missing required-env-var validation

Location: src/config/enviroment.ts lines 3–5 (requiredEnvVars declared but never checked)

Issue: If any of DATABASE_URL, TWILIO_SID, etc. is undefined, the app may crash or run with invalid config.

Remediation: Add a loop at startup to assert non-empty process.env values and throw if missing,



1. Potential over-fetching in Prisma queries

Location: src/db/services/userService.ts, e.g. lines 10–12 (prisma.user.create({ data })), lines 16–18 (prisma.user.findUnique({ where..}))

Issue: By default, Prisma returns every column, even ones you don't need. If you later add sensitive fields, they'll be exposed.

Remediation: Use the select or include options to fetch only required fields.



No informational issues were found.

Technical Findings Summary

Findings

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

Assessment Results

Score Results

Review	Score
Global Score	50/100
Assure KYC	Not completed
Audit Score	50/100

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 FAIL

The current state of the codebase contains critical security gaps that must be remediated before it can be considered secure.

Add Privy middleware at the top of your src/server.ts (or per-route) to lock down every endpoint.

Harden your public-facing routes with rate-limits, CORS origin-whitelisting, and input schemas.

Improve logging and secret handling by ignoring .env, sanitizing logs, and validating required configs at startup.

Layer on security headers via Helmet.

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.