

# Assure DeFi<sup>®</sup>

THE VERIFICATION **GOLD STANDARD**



## Security Assessment

### SHARBI FUN API

Date: 28/07/2025

Audit Status: PASS





Audit Edition: Code audit



ASSURE DEFI<sup>®</sup>  
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	<a href="https://github.com/dappgenie/sharbi.fun-api">https://github.com/dappgenie/sharbi.fun-api</a> Commit: <a href="#">616a3f2d92863ff1ad358b24abe1dcdb8a50a195</a>  Commit: <a href="#">abcfb611aab061d723ff6221dbd5f9cb8d062ab1</a>
Audit Methodology	Static, Manual

# Detailed Technical Report



## **1. Missing JWT SECRET assertion / fallback key [Fixed ✓]**

**Location:** src/auth/strategies/jwt.strategy.ts line 17

src/auth/auth.module.ts line 12

**Issue:** The JWT strategy falls back to a hard-coded secret ('your-secret-key') if process.env.JWT\_SECRET isn't defined. An attacker who discovers this fallback can forge arbitrary tokens, impersonating any user.

### **Remediation:**

```
// In bootstrap or your auth module
if (!process.env.JWT_SECRET) {
  throw new Error('Missing JWT_SECRET');
}
// Remove any `|| 'your-secret-key'` fallback
```

**Fix:** Added in the auth flow.

## **2. No Global Validation Pipe [Fixed ✓]**

**Location:** Missing in src/main.ts bootstrap (around lines 9–12)

**Issue:** Without ValidationPipe applied globally, incoming DTOs aren't stripped of unexpected properties or type-checked opening the door to mass assignment, prototype pollution, and malformed payload attacks.

### **Remediation:**

```
// src/main.ts
app.useGlobalPipes(new ValidationPipe({
  whitelist: true,
  forbidNonWhitelisted: true,
  transform: true,
}));
```

**Fix:** src/main.ts adds app.useGlobalPipes(new ValidationPipe({ whitelist:true, forbidNonWhitelisted:true, transform:true, forbidUnknownValues:true })).



### **3. EIP-191 Timestamp Replay Window (No Nonce) [Fixed ✓]**

**Location:** src/auth/auth.service.ts (lines 51–60, validateTimeStamp)

src/auth/auth.controller.ts (line 30, challenge message)

**Issue:** Relying solely on a timestamp window (for example 15 minutes) allows replay of a valid signature within that window. Without a one-time nonce, recorded challenges can be reused to impersonate a user.

**Remediation:** Include a unique, server-stored nonce in each challenge.

On verify, ensure the nonce hasn't been seen before, then mark it as used.

**Fix:** server-stored nonces implemented: src/auth/schemas/auth-nonce.schema.ts (unique nonce per address TTL 15 min).

src/auth/auth.service.ts has generateNonce(), extractNonce(), and validateNonce() that rejects reused nonces and stores the nonce on first successful verify (marking it as used).

Controller code for the challenge string is redacted in your file, but the service clearly expects a Nonce: line ensure your challenge includes it.

### **4. No Helmet / HSTS / Secure Headers [Fixed ✓]**

**Location:** Missing in src/main.ts bootstrap (around lines 9–12)

**Issue:** Missing security headers (CSP, HSTS, X-Frame-Options, etc.) leaves you vulnerable to clickjacking, MIME-sniffing, downgrade attacks, and XSS.

**Remediation:**

```
// src/main.ts
import helmet from 'helmet';
app.use(helmet());
```

**Fix:** src/main.ts imports helmet and calls app.use(helmet()).



### **1. No Rate-Limiting [Fixed ✓]**

**Location:** Missing ThrottlerModule import in src/app.module.ts (around lines 12–19)

**Issue:** Without throttling, attackers can brute-force login endpoints or overwhelm your API (DoS).

**Remediation:**

```
// src/app.module.ts
import { ThrottlerModule } from '@nestjs/throttler';

@Module({
  imports: [
    ThrottlerModule.forRoot({ ttl: 60, limit: 10 }),
```

```

    // ...
  ],
})
export class AppModule {}

```

**Fix:** src/app.module.ts imports ThrottlerModule.forRoot([{ ttl:60, limit:60 }]).

## 2. Weak File Upload Filtering (Malware Risk) [Fixed

**Location:** src/uploads/uploads.controller.ts (line 65)

**Issue:** Using FilesInterceptor('file') without fileFilter, size limits, or malware scanning permits large or malicious uploads risking storage exhaustion or hosting dangerous payloads.

**Remediation:**

```

@UseInterceptors(FilesInterceptor('file', 1, {
  limits: { fileSize: 5 * 1024 * 1024 }, // 5 MB max
  fileFilter: (req, file, cb) => {
    // e.g. allow only images: check file.mimetype
    const allowed = ['image/jpeg', 'image/png'];
    cb(null, allowed.includes(file.mimetype));
  },
}))

```

And integrate a virus-scan before persisting.

**Fix:** src/uploads/uploads.controller.ts now uses FilesInterceptor('files', 1, { limits:{ fileSize: 5\*1024\*1024 }, fileFilter: ... }) and re-checks allowed MIME types before uploading. Still no antivirus/malware scanning step present (no ClamAV/Snyk file scan..). If needed, add a scan before persisting/forwarding.

## 3. No Config Validation Schema (Env Var Risk) [Fixed

**Location:** src/app.module.ts (line 12)

**Issue:** Loading env vars without validation lets typos or missing values slip through (for example undefined secrets, wrong types) and only surface as runtime errors.

**Remediation:**

```

// src/app.module.ts
import * as Joi from 'joi';

ConfigModule.forRoot({
  isGlobal: true,
  validationSchema: Joi.object({
    NODE_ENV: Joi.string().valid('development', 'production').required(),
    JWT_SECRET: Joi.string().required(),
    PORT: Joi.number().default(3000),
    // ...other vars
  }),

```

```
});
```

**Fix:** `src/app.module.ts` uses `ConfigModule.forRoot({ isGlobal:true, validationSchema: Joi.object({ ... JWT_SECRET: Joi.string().required(), ... }) })`.

#### **4. Lack of Automated SCA [Fixed ✓]**

**Location:** No `.github/workflows/ci.yml` with SCA steps in project root

**Issue:** No CI step runs `npm audit` or `Dependabot/renovate`; unpatched dependencies may harbor known CVEs or supply-chain attacks.

**Remediation:** Add an `npm audit` (or `yarn audit`) stage to your CI pipeline.

Enable `Dependabot` (or `Renovate`) in `.github/dependabot.yml`.

Consider `Snyk/GitHub Advanced Security` integration.

**Fix:** `.github/dependabot.yml` exists (weekly `npm` updates).



#### **1. Insufficient Logging & Error Sanitation [Fixed ✓]**

**Location:** `src/main.ts` (lines 39–40)

`src/auth/auth.service.ts` (lines 42 & 97)

`src/uploads/uploads.controller.ts` (line 94)

**Issue:** Raw `console.log/console.error` calls leak stack traces and sensitive details to clients and unstructured logs hamper monitoring.

**Remediation:** Replace with `Logger` from `Nest` (for example `this.logger.error(.)`).

Sanitize error responses (no stack traces to clients).

Structure logs (JSON) with timestamps, levels, and request IDs.

**Fix:** Many places moved to `Nest Logger` and there's a `MongoExceptionFilter` that logs internally and sanitizes client errors. As a recommendation you are still using raw `console.*` in several files (like `src/filters/request-logging-middleware.ts`, `src/referral/referral.controller.ts`, and some watcher files). Replace those with `Logger` and keep client responses sanitized.



**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>	4					4
<div><div></div>Medium</div>	4					4
<div><div></div>Low</div>	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 SHARBI FUN API presently fails to mitigate several critical vulnerabilities. Until these high-risk issues are fully addressed, the SHARBI FUN API 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.

