Assure DeFi®

THE VERIFICATION GOLD STANDARD

Security Assessment

Sharbi

Date: 24/07/2025

Audit Status: FAIL

Audit Edition: Advanced

ASSURE DEFI®

THE VERIFICATION GOLD STANDARD

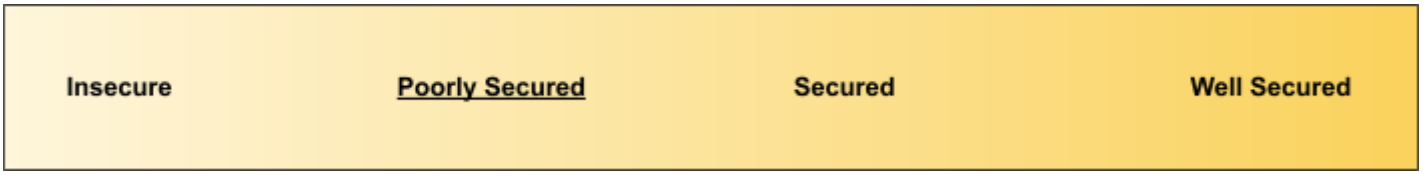
Risk Analysis

Vulnerability summary

Classification	Description
High	High-level vulnerabilities can result in the loss of assets or manipulation of data.
Medium	Medium-level vulnerabilities can be challenging to exploit, but they still have a considerable impact on smart contract execution, such as allowing public access to critical functions.
Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.
Informational	Informational vulnerabilities, code style violations, and informational statements do not affect smart contract execution and can typically be disregarded.

Executive Summary

According to the Assure assessment, the Customer's smart contract is **Poorly Secured.**



Scope

Target Code And Revision

For this audit, we performed research, investigation, and review of the Sharbi contracts followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

Target Code And Revision

Project	Assure
Language	Solidity
Codebase	https://github.com/dappgenie/sharbi.fun-contracts/blob/main/contracts [cf9db20bfb6b64057c15c4af560511b7c6c7b908]
Audit Methodology	Static, Manual

Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

Category	Item
Code review & Functional Review	• Compiler warnings. • Race conditions and Reentrancy. Cross-function race conditions. • Possible delays in data delivery. • Oracle calls. • Front running. • Timestamp dependence. • Integer Overflow and Underflow. • DoS with Revert. • DoS with block gas limit. • Methods execution permissions. • Economy model. • Private user data leaks. • Malicious Event log. • Scoping and Declarations. • Uninitialized storage pointers. • Arithmetic accuracy. • Design Logic. • Cross-function race conditions. • Safe Zeppelin module. • Fallback function security. • Overpowered functions / Owner privileges

AUDIT OVERVIEW

1. Single-address freeze [Denial-of-Service]

Contract: FunToken.sol

Issue: The owner can point blockedPairAddress at any address be it a hot wallet, bridge, DEX pair, or zero address and thereby globally block every transfer to or from that address.

Recommendation: The contract needs either:

Multiple address blocking with reasonable limits

Governance/timelock for blocking decisions

Whitelist approach instead of blacklist

Remove the blocking mechanism entirely

2. Reentrancy in launch

Contract: SharbiFun.sol

Issue: Multiple external calls (to feeTo, donationAddress, Uniswap factory, new FunToken, etc.) occur before any of the contract's own state is updated, and launch is not protected by nonReentrant. If an attacker (via a malicious feeTo or donationAddress contract) re-enters launch(), they can:

Create multiple token sales from one deposit

Duplicate entries in tokenList

Bypass or steal launch fees

Corrupt internal accounting like tokenLaunchBlock or totalTokenCount

Recommendation: Add nonReentrant to launch.

3. Broken Ownership Initialization

Contract: SharbiFun.sol

Issue: OpenZeppelin's OwnableUpgradeable defines __Ownable_init() without parameters. Calling __Ownable_init(_msgSender()) will fail to compile or if it compiles via a fallback overload, it won't correctly set the owner.

Recommendation: Change to __Ownable_init(); (no arguments).

Ensure owner is set to the intended admin in initialize.

4. Uniswap V3 Pool Creation DoS

Contract: SharbiFun.sol

Issue: No check for an existing pool.

If anyone pre-deploys a V3 pool for (tokenAddress, WETH, poolFee), createPool will revert with "POOL_EXISTS" and break both:

launch() (which calls _createTokenPair), so token sales cannot start

graduate() (if re-creating pair), preventing pooled liquidity

Recommendation: First call getPool(token0, token1, poolFee).

If non-zero, skip creation and return the existing pool; else create.

5. Zero-Min Slippage on Pool Mint

Contract: SharbiFun.sol

Issue: By specifying amount0Min = amount1Min = 0, any price movement between block.timestamp and execution (front-running, MEV) can steal value from your liquidity deposit. An attacker can sandwich your mint TX and capture most of the ETH or token leg.

Recommendation: Require sensible minimums or pass them in from the operator to tune slippage.

6. Zero-Min Slippage on Pool Mint

Contract: SharbiFun.sol

Issue: By specifying amount0Min = amount1Min = 0, any price movement between block.timestamp and execution (front-running, MEV) can steal value from your liquidity deposit. An attacker can sandwich your mint TX and capture most of the ETH or token leg.

Recommendation: Require sensible minimums or pass them in from the operator to tune slippage.

1. Mis-calculated & Block-based Timing

Contract: SharbiFun.sol

Issue: Wrong arithmetic: 4 × 24 × 60 × 30 = 172 800 blocks, not the intended 30 days × 24 × 60 × 30 = 1 296 000.

Block-number timing is insecure (miners can speed up/slow down +1 block).

Recommendation: Fix to 30 * 24 * 60 * BLOCKS_PER_MINUTE or switch to block.timestamp + 30 days.

Use block.timestamp for real-world durations.

2. No Domain Separation in Signature Scheme

Contract: SharbiReward.sol

Issue:

```
bytes32 messageHash = keccak256(
    abi.encodePacked(_user, _token, _amount, _nonce)
```

```
);
bytes32 ethSignedMessageHash = MessageHashUtils.toEthSignedMessageHash(messageHash);
address recoveredSigner = ECDSA.recover(ethSignedMessageHash, _claimProof);
```

You are signing raw packed data without any context (chain ID, contract address, function identifier)

Recommendation: Migrate to an EIP-712 domain separator (including chainId, address(this), and a custom SharbiFunDrop name/version). This binds signatures unambiguously to your contract and chain.

3. Bounding curve duration

Contract: SharbiFunDrop.sol

Issue:

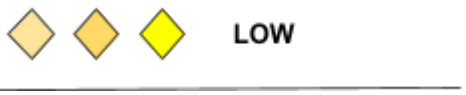
```
uint256 public constant BONDING_CURVE_DURATION = 4 * 24 * 60 * 30; // 30 days in
blocks (30 block per minute)
```

The comment says “30 days in blocks (30 blocks per minute)”, but the actual value is computed as

```
4 (days) × 24 (hours/day) × 60 (minutes/hour) × 30 (blocks/minute)
  = 4 × 24 × 60 × 30 = 172 800 blocks
```

At 30 blocks per minute, 172 800 blocks = 96 hours (4 days), not 30 days. The intended calculation 30 × 24 × 60 × 30 = 1 296 000 blocks is off by a factor of 7.5.

Recommendation: Fix the constant to match 30 days at 30 blocks/min



1. Missing Zero-Address Check on setRewardsContract

Contract: SharbiFun.sol

Issue: Owner can set rewardsContract = address(0). Later, when handleTokenExpire triggers, any ETH meant for rewards is sent to the zero address and irretrievably burned.

Recommendation:

```
function setRewardsContract(address newRewardsContract) public onlyOwner {
    if (newRewardsContract == address(0)) revert InvalidAddress("newRewardsContract",
newRewardsContract);
    rewardsContract = newRewardsContract;
}
```

2. Use abi.encode Instead of abi.encodePacked

Contract: SharbiFunDrop.sol

Issue: encodePacked concatenates fields without padding; although here all types are fixed-length, best practice is to avoid any future collision risk.

Recommendation: Switch to

```
keccak256(abi.encode(_user, _token, _amount, _nonce))
```

which unambiguously encodes each field with its full 32-byte word.

3. Hardcoded Fee Tier in Path Encoding

Contract: SharbiFunDrop.sol

Issue: The function hardcodes a 0.3% fee tier (3000) without checking which pools actually have liquidity.

Recommendation: Implement logic to check available fee tiers and select the best one.

4. Missing Token State Validation in Key Functions

Contract: SharbiFunDrop.sol

Issue: Some functions that should check token state don't have the onlyWhenStateIs modifier, potentially allowing operations on tokens in incorrect states.

Recommendation: Add state validation to all functions that interact with specific tokens.

5. Launch Fee and Amount Values

Contract: SharbiFunDrop.sol

Issue: Several values have TODOs indicating they should be changed to higher values (for example, from 400 * 10^12 to 4000).

Recommendation: Ensure all values are properly set before production deployment.

4. No msg.sender Check on claimReward

Contract: SharbiReward.sol

Issue:

```
function claimReward(
    address _user,
    uint256 _amount,
    uint256 _nonce,
    bytes memory _claimProof
) public onlyValidNonce(_user, _nonce) { … }
```

Anyone can submit a claim on behalf of _user. While funds still go to _user, this allows:

Front-running a genuine user's transaction (wasting their gas).

Potential griefing by calling many small claims on behalf of users.

Recommendation: Require msg.sender == _user (or implement an optional "relayer" whitelist).

5. Missing Zero-Address Check on Operator

Contract: SharbiReward.sol

Issue:

```
constructor(address _operator) Ownable(_msgSender()) {
    operator = _operator;
}
```

Issue: If _operator is passed as address(0), no check reverts, so:

No one can ever produce a valid signature (operator==0), locking all future claims.

Recommendation: if (_operator == address(0)) revert InvalidAddress("_operator", _operator);

6. No Event on Owner Withdrawal

Contract: SharbiReward.sol

Issue:

```
function withdraw(uint256 _amount) public onlyOwner { ... }
```

Owner's withdraw emits no event, making it harder to trace when funds leave the contract.

Recommendation:

```
event Withdrawn(address indexed owner, uint256 amount);
...
function withdraw(uint256 _amount) public onlyOwner {
    ...
    emit Withdrawn(owner(), _amount);
}
```

1. Commented contract

Contract: SharbiToken.sol

Issue: The entire contract is commented.

Recommendation: Uncomment or delete the contract if it is not needed.

Technical Findings Summary

Findings

Vulnerability Level	Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
High	6					
Medium	3					
Low	6					
Informational	1					

Assessment Results

Score Results

Review	Score
Global Score	**40/100**
Assure KYC	Not completed
Audit Score	40/100

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

Following our comprehensive security audit of the token contract for the Sharbi project, the project did not fulfill the necessary criteria required to pass the security audit.

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.