Assure DeFi®

THE VERIFICATION GOLD STANDARD

Security Assessment

Gradient

Date: 29/05/2025

Audit Status: PASS

Audit Edition: Advanced

ASSURE DEFI®

THE VERIFICATION GOLD STANDARD

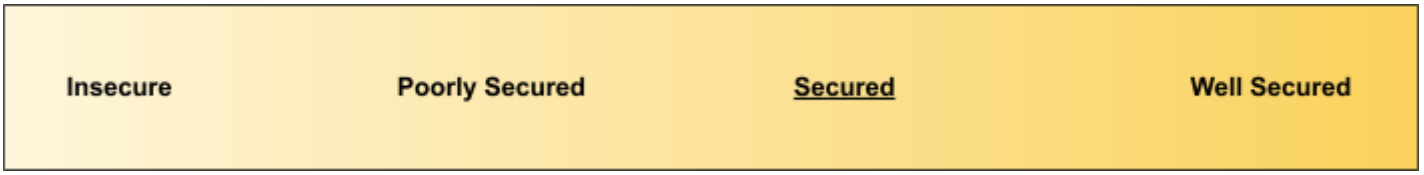
Risk Analysis

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Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.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 **Secured.**



Scope

Target Code And Revision

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

Target Code And Revision

Project	Assure
Language	Solidity
Codebase	TestToken.sol [SHA256]: f34aeb4afea7d5e19bdbbc1b7c5f54b53a4c64bfd4308f5a154afcafa389976f Fixed version: Token.sol [SHA256] 97ad837f864e5bff28a22d4b6ab33893c994a0ab6c480026124ea2357d893742
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

HIGH

1. Division-by-Zero in _handleLiquidity [Fixed ✅]

Function: _handleLiquidity

Issue: Division-by-Zero on combinedFee: combinedFee = totalTaxRate / 100; uses integer division. If totalTaxRate < 100, then combinedFee == 0, causing a revert on (amount * ethFee) / combinedFee.

Recommendation: Add a require(totalTaxRate >= 100, "Tax rate too low") in constructor and updateTaxRates, or compute combinedFee with full-precision (e.g. multiply before dividing) and guard against zero.

Fix: Constructor now has require(_initialTaxRate >= 100, "Tax rate is too low"); so totalTaxRate ≥ 100. _handleLiquidity also checks require(combinedFee > 0, "Invalid tax rate"); to prevent divide-by-zero.

MEDIUM

1. DoS via Gas-limited .transfer [Fixed ✅]

Issue: DoS via transfer to treasury: uses treasuryReceiver.transfer(). If the treasury is a contract whose fallback consumes >2 300 gas, the transfer and thus liquidity routine will revert, halting auto-liquidity forever.

Recommendation: Replace .transfer with (bool sent,) = treasuryReceiver.call{value:ethForTreasury}(""); require(sent, "Treasury transfer failed"); to forward all gas and allow recovery.

Fix: The code now uses payable(treasuryReceiver).call{value: ethForTreasury}("") with a require(success, ..), forwarding all gas and preventing the 2 300 gas limitation.

2. Unsafe ERC-20 Rescue in rescueTokens [Fixed ✅]

Issue: Unsafe ERC-20 call: invokes IERC20(token).transfer(owner(), bal) without checking return value. Tokens that don't return bool or that revert on transfer will block this rescue.

Recommendation: Use OpenZeppelin's SafeERC20 library: SafeERC20.safeTransfer(IERC20(token), owner(), bal); to handle non-standard tokens safely.

Fix: The contract imports SafeERC20 and now calls IERC20(token).safeTransfer(owner(), bal);, ensuring compatibility with tokens that do not return a boolean.

3. Imbalanced Fee Update in updateTaxRates

Issue: Insufficient bounds checking: allows newETHFee or newLiqFee = 0, potentially setting one fee to zero while the other equals combinedFee, but does not verify that ethFee + liqFee == combinedFee. This can unbalance the intended split or set both to zero, breaking liquidity logic.

Recommendation: Add require(newETHFee + newLiqFee == totalTaxRate/100, "Fees must sum to combinedFee"); so that liquidity/treasury shares cannot be mis-split or accidentally zeroed.

Fix: Now enforces uint256 newTotal = newETHFee + newLiqFee; require(newTotal > 0, "Total tax rate cannot be zero"); require(newTotal <= (totalTaxRate / 100), "Tax must be reduced"); totalTaxRate = newTotal * 100;, preventing zero or mis-split rates.

LOW

No low issues were found.

INFORMATIONAL

1. Tax-exemption logic uses OR instead of AND [Acknowledge ✅]

Issue: Tax-exemption logic uses OR instead of AND. The code uses

```
if (!_isTxLimitExempt[from] || !_isTxLimitExempt[to]) {
    // enforce trading/tax logic
}
```

This means that if either the sender or the recipient is not exempt, the contract still applies taxes and limits. As a result, addresses intended to be fully exempt (e.g., the owner or the contract itself) can still incur fees or hit transaction limits when interacting with non-exempt parties.

Recommendation: Change the conditional to require both parties to be non-exempt before applying tax or limits

2. Trading-enabled check overly broad [Acknowledge ✅]

Issue: Trading-enabled check overly broad. The code uses:

```
if (!_isTxLimitExempt[from] || !_isTxLimitExempt[to]) {
    require(tradingEnabled, "Trading is not yet enabled");
}
```

Because of the ||, trading is only allowed when both parties are exempt. This can block legitimate exempt-to-exempt transfers and unintentionally allow some non-exempt transfers before trading is activated.

Recommendation: Use && so the trading-enabled requirement only applies when both parties are non-exempt

Technical Findings Summary

Findings

Vulnerability Level	Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
High	1					1
Medium	3					3
Low	0					
Informational	2			2		

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

Following our comprehensive security audit of the token contract for the Gradient project, 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 adGradient 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 adGradient, 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 serGradients 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 serGradients, 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 serGradients may access, and depend upon, multiple layers of third parties.