Assure DeFi®

THE VERIFICATION GOLD STANDARD

Security Assessment

Gradient

Date: 06/07/2025

Audit Status: PASS

Audit Edition: Advanced+

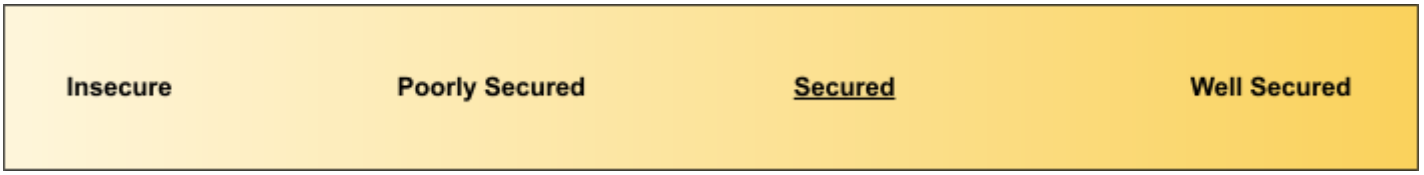
Risk Analysis

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Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.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	FallbackExecutor.sol [SHA256] - 175ae6ada483faa8639caef701306b741b4862392c523bac46eab290d60561d7 GradientMarketMakerPool.sol [SHA256] - ae63e4808a7f74647cb34255b02076ec064c3629372c1698f6353b8f5d55e1fb GradientOrderbook.sol [SHA256] - 8db61daebe5a75b03f6433c401690239b36d3f4cc72a7631f737f4e33c9f4008 GradientRegistry [SHA256] - 22b500a1a5defd6f1e22cfd78040906cd67c4aaf9fd81b37b86443d52463c31e — Fixed version - gradient-contracts.zip [SHA256] e7e16e2023e6019ae2772d336f891f54246f36a87b9d7e556619088901e1fe71
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. Incorrect Fee Calculation and ETH Handling in Buy Orders [Fixed - But Protocol recommendations ✅]

Function: GradientOrderbook.createOrder

Issue: In the createOrder function, when creating a buy order, the contract calculates the fee based on the total cost but does not properly account for this in the ETH value check. The fee is added to the required ETH, but then immediately added to totalFeesCollected, which could lead to incorrect accounting.

Recommendation: Defer buyer fee collection until the moment of matching (just like you do with the seller fee), for example move the totalFeesCollected += buyerFee into your _fulfill logic and deduct it from the ETH owed to the buyer.

Else, dont charge buyers a fee at order creation at all, just hold their principal (totalCost) and charge the fee when the trade executes or on cancellation/expiration.

Fix: _collectFee is only ever called in your fulfillment functions (_fulfillLimitOrdersMatching, _fulfillMarketOrdersMatching, MM and AMM flows), and never in createOrder.

Protocol recommendation: Currently if an order fails to fill (for example no matching orders), the ETH remains locked without any fee compensation for the protocol, as the contract accepts and holds the full totalCost amount immediately so if the order is never filled, this ETH remains locked until cancellation/expiration

2. Incorrect Order Queue Management [Fixed ✅]

Function: GradientOrderbook, Queue management functions

Issue: The order queue implementation using orderQueues mapping doesn't properly handle order removal, leading to potential ghost orders in the queue that can't be efficiently removed.

Recommendation: Implement a proper queue structure with efficient removal, or use a bitmap to mark inactive orders rather than trying to remove them from the array.

Fix: Replaced the old array-based queues (which couldn't delete in-place) with a proper doubly-linked list keyed by (token, orderType, executionType), and every removal now call:

```
_removeOrderFromLinkedQueue(queueKey, orderId)
```

which updates its prev and next pointers (or the headOrder/tailOrder) and then deletes that node.

3. Price Precision Issues [Fixed ✅]

Function: GradientOrderbook, all price related calculations

Issue: The contract uses fixed-point arithmetic with 1e18 precision, but there is no validation that token decimals align with this assumption. This could lead to incorrect price calculations.

Recommendation: Normalize amounts to 18 decimals or restrict tokens to 18 decimals only

Fix: Addressed by normalizeTo18Decimals/denormalizeFrom18Decimals helpers

4. Price Manipulation in Market Orders [Fixed ✅]

Function: GradientOrderbook.fullFillMarketOrders()

Issue: Market orders can be executed at any price within the bounds, but there's no validation that the execution price is fair or close to market rates.

Recommendation: Implement price validation against oracle or recent trade prices.

Fix: Enforced that market fills stay within a configurable band around the live Uniswap rate.

5. Unauthorized Fund Access via Registry [Acknowledge]

Function: GradientOrderbook.fullFillMarketOrders()

Issue: The owner can change the registry at any time, potentially redirecting funds to malicious contracts since the registry controls authorized fulfillers and market maker pools

Recommendation: Implement timelock or multi-sig for registry changes.

6. Incorrect Reward Calculation in _claimTokenPoolFee [N/A, fees will be distributed in eth ✅]

Function: GradientMarketMakerPool.claimTokenPoolFee

Issue: The _claimTokenPoolFee function incorrectly sends ETH rewards instead of token rewards. The function's logic and event emission suggest it should handle token rewards, but it transfers ETH instead.

Recommendation: Replace the call{value:…} with an ERC-20 transfer, this aligns the on-chain behavior with the event and your intended token pool semantics.

7. Excessive Funds Withdrawal Logic Flaw [Acknowledge]

Function: GradientMarketMakerPool.withdrawExcesiveFunds()

Issue: The calculation doesn't account for pending rewards or accumulated fees that should belong to LPs. This could allow the owner to withdraw funds that rightfully belong to liquidity providers.

Recommendation: Before letting the owner pull excess ETH, you need to reserve:

All unclaimed ETH rewards and any other owed balances (for example ending fees, if this pool holds fee distributions). Account for all pending rewards and fees in the calculation

8. Epoch Increment Logic Manipulation [Fixed ✅]

Function: GradientMarketMakerPool._checkAndIncrementETHEpoch() and _checkAndIncrementTokenEpoch()

Issue: Epoch increments are based solely on pool being empty, which can be manipulated by an attacker who drains the pool and then immediately adds liquidity to a new epoch, potentially avoiding rewards or manipulating the reward distribution.

Recommendation: Implement more sophisticated epoch management with time-based or

governance-based transitions. Consider moving epoch transitions to an explicit admin/governance function that can only be called after a fixed delay

Fix: Included Cooldown Period and Minimum Liquidity Requirement

9.Lack of Zero Address Checks in setMainContracts [Fixed ✅]

Function: GradientRegistry.setMainContracts

Issue: The setMainContracts function doesn not validate that the input addresses are non-zero. This could lead to protocol components being set to zero addresses, effectively breaking the system.

Recommendation: Before emitting and assigning, validate each input is non-zero

Fix: In setMainContracts now immediately require each of _marketMakerPool, _gradientToken, _orderbook, _fallbackExecutor, and _router to be non-zero before doing anything els

10. Price Manipulation Vulnerability [Partially Fixed ✅]

Function: FallbackExecutor._isDEXSuitable()

Issue: The liquidity check only verifies if reserves exist but doesn't protect against price manipulation attacks through flash loans or large trades.

Recommendation: Implement price oracles, TWAP checks, or minimum liquidity thresholds.

Fix: A minLiquidityThreshold (default 10 ETH) was implemented to prevent using pools with extremely low liquidity and a 10% safety margin on reserve checks was added. It is still possible to add more improvements implementing TWAP checks, max trade size limits, cross-DEX price validation, and deeper liquidity verification to prevent manipulation.

11. Market Order Price Validation Bypass [Acknowledge]

Function: GradientOrderbook.fullFillMarketOrders()

Issue: In _fulfillMarketOrdersMatching, the price validation for market orders is not properly enforced when one order is a market order and the other is a limit order. The validation only checks the market order side's price constraints

Recommendation: Replace the two if(...) checks with un-conditional requires that each order check its own constraint, regardless of whether it’s limit or market.

```
// Example:
if (buyOrder.executionType == OrderExecutionType.Limit) {
    require(executionPrice <= buyOrder.price, "Execution price exceeds buyer's limit price");
}
if (sellOrder.executionType == OrderExecutionType.Limit) {
    require(executionPrice >= sellOrder.price, "Execution price below seller's limit price");
}
```

MEDIUM

1. Inefficient Order Queue Processing [Fixed ✅ - Possible improvements]

Function: GradientOrderbook.createOrder

Issue: The order queue processing is O(n) and could become gas-intensive with many orders.

Recommendation: Use a priority heap / sorted linked-list, Shard your queues by price 'bins' or use Off-chain matching, on-chain settlement

Fix: The code now uses a linked list structure with headOrder, tailOrder, and linkedOrders mapping, which allows for O(1) insertion and removal operations and efficient queue operations. Additionally, the following improvements can be made:

1. Shard orders by price ranges to limit search space
2. Maintain price-sorted order for efficient best-price retrieval
3. Move complex matching logic off-chain, only settle on-chain
4. Implement batch processing with gas limits
5. Use heaps for maintaining top N orders efficiently

2. Unbounded Array Growth in User Epoch Tracking [Acknowledge]

Function: GradientMarketMakerPool._addUserToParticipatedEpochs

Issue: The userParticipatedETHEpochs and userParticipatedTokenEpochs arrays can grow indefinitely as users participate in more epochs. This could lead to gas limits being hit when processing these arrays.

Recommendation: Cap the array size, Use a mapping of epoch - bool , Prune old epochs

3. Missing Minimum Amount Checks in Order Execution [Acknowledge]

Function: GradientMarketMakerPool.executeBuyOrder and executeSellOrder

Issue: The executeBuyOrder and executeSellOrder functions don't verify that the received amounts meet any minimum requirements, which could lead to sandwich attacks or other MEV exploitation.

Recommendation: Add an on-chain sanity check against your Uniswap reserves (or your own internal reserves) to enforce that the pool is giving the right amount. **Do the mirror check in executeSellOrder to protect LPs and ensure fair execution*.

4. Inefficient Gas Usage in Batch Operations [Fixed - Escalability ✅]

Function: GradientMarketMakerPool.removeAllEpochsLiquidity(), removeAllETHLiquidity()

Issue: Batch operations iterate through arrays without gas limit considerations, potentially causing transactions to fail due to block gas limits.

Recommendation: Implement gas limit checks and allow partial processing.

Fix: The recommended improvements have been correctly implemented, ensuring:

Batch operations respect gas limits.

Users can process partial withdrawals.

The contract prevents out-of-gas failures.

5. No Way to Remove Reward Distributors [Fixed ✅]

Function: GradientRegistry.setRewardDistributor

Issue: The setRewardDistributor function only allows adding reward distributors but doesn't provide a way to remove them if they become compromised or need to be rotated.

Recommendation: Add a function to remove reward distributors

Fix: Now, administrators can both add and remove reward distributorse.

6. Centralization Risk - Single Point of Failure [Fixed ✅]

Function: GradientRegistry

Issue: The contract relies entirely on a single owner for all administrative functions. This creates significant centralization risks.

Recommendation: Implement multi-signature wallet as owner

Consider role-based access control (AccessControl from OpenZeppelin)

Add timelock mechanisms for critical changes

Fix: Timelock and role based access control.

7. Unbounded ActiveDEXes Array [Acknowledge]

Function: FallbackExecutor.sortDEXesByPriority

Issue: The activeDEXes array can grow without bound as new DEXes are added. When sorting (in _sortDEXesByPriority), this could lead to gas limits being hit and making the contract unusable if too many DEXes are added.

Recommendation: Implement a maximum number of active DEXes

Consider using a mapping with a separate sorted list for more efficient management

Or implement a more efficient sorting algorithm (current bubble sort is $O(n^2)$)

8. Insufficient Access Control on DEX Management [Acknowledge]

Function: FallbackExecutor.addDEX() and removeDEX() functions

Issue: Owner can add malicious DEX routers that could steal funds or return incorrect data.

Recommendation: Implement multi-sig governance or time-locked changes for DEX modifications.

9. Liquidity Check Mismatch [Acknowledge]

Function: FallbackExecutor._isDEXSuitable()

Issue: Liquidity check compares reserves against trade amount but doesn't account for slippage impact.

Recommendation: Calculate expected output amount and verify against minAmountOut.

LOW

1. Inconsistent Authorization Patterns [Acknowledge]

Function: GradientRegistry.authorizedContracts, isRewardDistributor, and authorizedFulfillers

Issue: The contract has three different authorization mappings (authorizedContracts, isRewardDistributor, and authorizedFulfillers) with similar but separate functionality. This could lead to confusion and inconsistent access control.

Recommendation: Consider consolidating these into a more unified access control system, possibly using OpenZeppelin's AccessControl.

2. String Comparison Vulnerability [Acknowledge]

Function: GradientRegistry.setContractAddress

Issue: The setContractAddress function compares string hashes which, while not directly vulnerable, is an unusual pattern that could lead to maintenance issues or potential collisions.

Recommendation: Consider using an enum for contract names instead of string comparison

3. Missing Access Control on Key Functions [Acknowledge]

Function: GradientRegistry.onlyAuthorized

Issue: The contract has an onlyAuthorized modifier defined but it's never used in any functions.

Recommendation: Apply onlyAuthorized modifier to appropriate functions

Consider which functions should be callable by authorized contracts vs. only owner

INFORMATIONAL

1. Incorrect Partial Fill Handling in Market Maker Fulfillment [Acknowledge]

Contract: GradientOrderbook

Issue: In _fulfillOrderWithMarketMaker, when handling partial fills of sell orders, the contract doesn't properly account for the actual amount of ETH received from the market maker pool.

Recommendation: Capture and use the actual returned ETH [example]:

```
// Instead of ignoring the return value:
uint256 ethReceived = IGradientMarketMakerPool(marketMakerPool)
    .executeSellOrder(order.token, paymentAmount, actualFillAmount);

// Now compute fee on real proceeds
uint256 fee = _collectFee(ethReceived);
uint256 payout = ethReceived - fee;
```

```
// Distribute portion to pool as before

// Finally, send the correct amount
(bool success, ) = order.owner.call{value: payout}("");
require(success, "ETH transfer to seller failed");
```

Testing coverage

During the testing phase, custom use cases were written to cover all the logic of contracts. *Check "Annexes" to see the testing code.*

```
tests/test_orderbook.py::test_create_order RUNNING
Transaction sent: 0xe1ff63ff59524a7abed77049f2267cd7b5068698d62e4d65c6253184363940a0
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  ERC20Mock.constructor confirmed   Block: 1   Gas used: 523822 (4.37%)
  ERC20Mock deployed at: 0x3194cBDC3dbcd3E11a07892e7bA5c3394048Cc87

Transaction sent: 0xd50dd7eb989b4b9c0ed63a0c62b52530ea223e7248ec9c81887ff7b0c084d662
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  GradientOrderbook.constructor confirmed   Block: 2   Gas used: 2201820 (18.35%)
  GradientOrderbook deployed at: 0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6

Transaction sent: 0x632fd8cdbc2ca674a0b17ecc5a782f496f5ae0d9f1f2ebbee7aaca6a716fb902
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  ERC20Mock.mint confirmed   Block: 3   Gas used: 65821 (0.55%)

Transaction sent: 0x4825644c1325c53aeb237dd63ec1e3d211ba051a8556923632af155cce96736d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  ERC20Mock.approve confirmed   Block: 4   Gas used: 44283 (0.37%)

Transaction sent: 0xc53440ed8c693f42fb7d7eef69432b6b3bafdbe828682afd72321e8375954a23
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  ERC20Mock.mint confirmed   Block: 5   Gas used: 50809 (0.42%)

Transaction sent: 0xe9ea835997dc3c06e70b09f6c8813d54b7e76fa8d48946e4add57caa54d93be7
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  ERC20Mock.approve confirmed   Block: 6   Gas used: 44283 (0.37%)

Transaction sent: 0xed960cafe4980a64ad52ff5b7c760d31496aa7d45baac9924a86225dc15ac69b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  GradientOrderbook.createOrder confirmed (Invalid token)   Block: 7   Gas used: 28345 (0.24%)

Transaction sent: 0xbd71f061f10053898d4b96029e24dd698e7ba418549099bc1c96c4749d0aec59
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  GradientOrderbook.createOrder confirmed (Amount must be greater than 0)   Block: 8   Gas used: 28590 (0.24%)

Transaction sent: 0x246f8a683ae16182e882797b1dfd2e985461593dd898d2bc0f61cb7b34926653
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  GradientOrderbook.createOrder confirmed (Invalid price range)   Block: 9   Gas used: 28571 (0.24%)

Transaction sent: 0x2d250ddbe7612ca7a484720a020319741c10b397e7d9b5820d7f19961c0b7ed9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  GradientOrderbook.createOrder confirmed (TTL must be greater than 0)   Block: 10   Gas used: 28612 (0.24%)

Transaction sent: 0xcb02cb90d001bbfa8f753fa25fcc1d2f5b2180eb3cfb81f7cc228b4601f61e3e
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  GradientOrderbook.createOrder confirmed (Insufficient ETH sent)   Block: 11   Gas used: 29866 (0.25%)

Transaction sent: 0xfa4414734a55fba51b5715a4943b856baf9273f9a29490f1919d9873e0cb4d03
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 6
  GradientOrderbook.createOrder confirmed   Block: 12   Gas used: 207761 (1.73%)

Transaction sent: 0x9859c43e990d8b2c9aa56b23d783361f1d0692767e29a02df9def57c06157cf1
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  GradientOrderbook.createOrder confirmed   Block: 13   Gas used: 239440 (2.00%)

tests/test_orderbook.py::test_create_order PASSED
```

```
tests/test_orderbook.py::test_cancel_order RUNNING
Transaction sent: 0x65d4d295d008530607748c25c3f2a4873c11c995b32f6b0bb3edee3876502ca4
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  ERC20Mock.constructor confirmed   Block: 14   Gas used: 523822 (4.37%)
  ERC20Mock deployed at: 0xe0aA552A10d7EC8760Fc6c246D391E698a82dDf9

Transaction sent: 0xb96e65d9462e1406b42c7d46cc265d026d97948344ac794ef2460bcd115b40f8
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  GradientOrderbook.constructor confirmed   Block: 15   Gas used: 2201820 (18.35%)
  GradientOrderbook deployed at: 0x6b4BDe1086912A6Cb24ce3dB43b3466e6c72AFd3

Transaction sent: 0xb7ed23510d5212a16769de8eea4fa4c863b67525b3bd5b50e330a275b595de44
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 6
  ERC20Mock.mint confirmed   Block: 16   Gas used: 65821 (0.55%)

Transaction sent: 0xb75ab4c8b403a86ccdfb990fc9cd9d1d87c816a6240a8b0473c503ad3212c9c0
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  ERC20Mock.approve confirmed   Block: 17   Gas used: 44283 (0.37%)

Transaction sent: 0x2963ad1baa640822cab9b3aae96ba2dcc45de974c0fc0f1263ad16c2dbdae0b7
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  ERC20Mock.mint confirmed   Block: 18   Gas used: 50809 (0.42%)

Transaction sent: 0xdd4c67522d37c4d803148ea97396e073dd4696070c12be0f0cf6241f97a436bc
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  ERC20Mock.approve confirmed   Block: 19   Gas used: 44283 (0.37%)

Transaction sent: 0x1e071c209ef311e8381c57b21fca21fd23c75245f366a3d7e9028a0f71a52074
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  GradientOrderbook.createOrder confirmed   Block: 20   Gas used: 207761 (1.73%)

Transaction sent: 0x2db2c48a64c2b3811fafe7c9746eb5de6d06c58119ecbfbd17c986229ab9fd23
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  GradientOrderbook.createOrder confirmed   Block: 21   Gas used: 239440 (2.00%)

Transaction sent: 0x976600c45f9e7e18b3f178dd51be1bc627ca493752674b088171f8d3f9d304ea
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  GradientOrderbook.cancelOrder confirmed (Order does not exist)   Block: 22   Gas used: 28348 (0.24%)

Transaction sent: 0x44abbd921151c24fa1042ae630ec1b39bb71c9068d84783265f7e06bf9bf57e6
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  GradientOrderbook.cancelOrder confirmed (Not order owner)   Block: 23   Gas used: 29233 (0.24%)

Transaction sent: 0xd8d62b3992133a2538c27f4c13001a19e6c1c7b66d2343f5797ccdf7c1efd7a3
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  GradientOrderbook.cancelOrder confirmed   Block: 24   Gas used: 63420 (0.53%)

Transaction sent: 0xe0e17b1ed348d3c628d44abb5011023013ede638a5436c4bdd22a391beafee3c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 11
  GradientOrderbook.cancelOrder confirmed (Order not active)   Block: 25   Gas used: 30163 (0.25%)

Transaction sent: 0x3218cb443ee4aa9662f988a399ce4db9da2433bd271415411fdb25e988f35ed0
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  GradientOrderbook.cancelOrder confirmed   Block: 26   Gas used: 55308 (0.46%)

Transaction sent: 0x3ea08da148ac9b0c41365ce5f6143f759e9658f482994308df46f46ce61c8eee
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  GradientOrderbook.cleanupExpiredOrder confirmed (Order not active)   Block: 27   Gas used: 29323 (0.24%)

Transaction sent: 0xf703c191bfc141fc95f24fcc3d9b7fdefc26035850d67d38ccc28a9a4885fc9f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 12
  GradientOrderbook.createOrder confirmed   Block: 28   Gas used: 201161 (1.68%)

Transaction sent: 0xc3d7f8c1ec0b74835c19328f729601103a2cf10668e2d50f13ae56b669c8fa06
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  GradientOrderbook.cancelOrder confirmed (Order expired)   Block: 30   Gas used: 32019 (0.27%)

Transaction sent: 0xfafcde961977e00827948ee950726953fb6d43dff88f5329bb83fa146cb87b42
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  GradientOrderbook.createOrder confirmed   Block: 31   Gas used: 224440 (1.87%)

Transaction sent: 0x9f8b4bef23b133a30418f76c6d82ddeacf57f01c562e85f87e6583cd1541ba60
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  GradientOrderbook.cleanupExpiredOrder confirmed   Block: 33   Gas used: 54438 (0.45%)
```

```
tests/test_orderbook.py::test_fulfill_matched_orders RUNNING
Transaction sent: 0xcaae7574140b242b748996b62499a1322c5348c8b434123edc30c7d842c63ab1
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 11
  ERC20Mock.constructor confirmed   Block: 34   Gas used: 523822 (4.37%)
  ERC20Mock deployed at: 0x7a3d735ee6873f17Dbdcab1d51B604928dc10d92

Transaction sent: 0x2afa7230790de6df64e30b7771ad026bc74656909a717871eb6a3c220e8acfc7
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 12
  GradientOrderbook.constructor confirmed   Block: 35   Gas used: 2201820 (18.35%)
  GradientOrderbook deployed at: 0x2c15A315610Bfa5248E4CbCbd693320e9D8E03Cc

Transaction sent: 0x015dc7f59c272cbdca9cddc3ab5dc525512cc70bf720c4df92c04676d8cb2b95
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  ERC20Mock.mint confirmed   Block: 36   Gas used: 65821 (0.55%)

Transaction sent: 0x53b2b2499afaad26a491db8bafa25e9007f59680e0d3c9bd8b5274a456998613
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 14
  ERC20Mock.approve confirmed   Block: 37   Gas used: 44283 (0.37%)

Transaction sent: 0x988950f8eadab535d27a63ffd538aa4bf9423eb5abff191c01cbea1b0f034489
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 14
  ERC20Mock.mint confirmed   Block: 38   Gas used: 50809 (0.42%)

Transaction sent: 0xcc21710de993940a60054675e2f4c49fe328c779dfe91e946011964f094504b2
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 6
  ERC20Mock.approve confirmed   Block: 39   Gas used: 44283 (0.37%)

Transaction sent: 0x98c54e120307b1772c2dcbde2d5530dc9479bcf3625cb79058d1dd3c2a86a5e8
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 15
  GradientOrderbook.fulfillMatchedOrders confirmed (Caller is not whitelisted)   Block: 40   Gas used: 28664 (0.24%)

Transaction sent: 0x018c4e19737653bc65803fd067b6bba7ff48017abb9747dbe2c5578aaea9edf7
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 15
  GradientOrderbook.fulfillMatchedOrders confirmed (No order matches to fulfill)   Block: 41   Gas used: 28681 (0.24%)

Transaction sent: 0xd471c6ab4779717492f25581c101d684aa87d4e17d878c73e23e33b45165bd02
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 16
  GradientOrderbook.fulfillMatchedOrders confirmed (Order does not exist)   Block: 42   Gas used: 32279 (0.27%)

Transaction sent: 0x6f5ffb149e7dfbe7feddda0291a00a98bf1e92cfa26f3efb75c825cdc3840839
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 16
  GradientOrderbook.createOrder confirmed   Block: 43   Gas used: 207773 (1.73%)

Transaction sent: 0x5b429cedb138f749b4f4ee08c02dcc1fbee0cef30aef09a47f941664cdb9c145
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  GradientOrderbook.createOrder confirmed   Block: 44   Gas used: 239440 (2.00%)

Transaction sent: 0x24d4f10c37839b71c4534a722778a70027549a1943f079bbbcf302a959c58b52
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 17
  GradientOrderbook.fulfillMatchedOrders confirmed (Invalid fill amount)   Block: 45   Gas used: 43542 (0.36%)

Transaction sent: 0x0d201597ae947f403e0123b7f990212c9ac966f15a50ef29da6c771001665567
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 18
  GradientOrderbook.fulfillMatchedOrders confirmed (Fill amount exceeds available)   Block: 46   Gas used: 43603 (0.36%)

Transaction sent: 0x2851a9d93501c306602625d1b7b12620fe6502c247525d0252fdda996e758269
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 19
  GradientOrderbook.fulfillMatchedOrders confirmed   Block: 47   Gas used: 157305 (1.31%)

Transaction sent: 0x77585e06abfba1e53f35725f15cf08e629000125081be243015c8f300ae2bd1f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  GradientOrderbook.cancelOrder confirmed   Block: 48   Gas used: 55308 (0.46%)

Transaction sent: 0x4c6ebedf089e456fa63a833a23f1bca1bc76bf5d6101321a1d49083deb347ded
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 20
  GradientOrderbook.fulfillMatchedOrders confirmed (Orders must be active)   Block: 49   Gas used: 30484 (0.25%)

tests/test_orderbook.py::test_fulfill_matched_orders PASSED
```

Annexes

Testing code:

```
from brownie import (
    reverts,
)


from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    DAY_TIMESTAMP,
    get_account,
    get_timestamp,
    get_chain_number,
    increase_timestamp
)


from scripts.deploy import (
    deploy_erc,
    deploy_orderbook
)


'''
    enum OrderType {
        Buy,
        Sell
    }
'''
```

```
def test_create_order(only_local):
    # Arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)

    mock_token = deploy_erc(owner, "Mock", "MCK")
    orderbook = deploy_orderbook(owner)

    # mint some tokens
    mock_token.mint(other, 10e18)
    mock_token.approve(orderbook.address, 100e18, {"from": other})
    mock_token.mint(extra, 10e18)
    mock_token.approve(orderbook.address, 100e18, {"from": extra})

    priceXToken = 1e16

    with reverts("Invalid token"):
        orderbook.createOrder(
            0, ZERO_ADDRESS, 5,
            priceXToken, DAY_TIMESTAMP * 7,
            {"from": other})
    with reverts("Amount must be greater than 0"):
        orderbook.createOrder(
            0, mock_token.address, 0,
            priceXToken, DAY_TIMESTAMP * 7,
            {"from": other})
    with reverts("Invalid price range"):
        orderbook.createOrder(
```

```
            0, mock_token.address, 5,
            0, DAY_TIMESTAMP * 7,
            {"from": other})
    with reverts("TTL must be greater than 0"):
        orderbook.createOrder(
            0, mock_token.address, 5,
            priceXToken, 0,
            {"from": other})
    with reverts("Insufficient ETH sent"):
        orderbook.createOrder(
            0, mock_token.address, 5,
            priceXToken, DAY_TIMESTAMP * 7,
            {"from": other})


    tx = orderbook.createOrder(
            0, mock_token.address, 5,
            priceXToken, DAY_TIMESTAMP * 7,
            {"from": other, "value": priceXToken * 6})
    assert tx.events['OrderCreated'][0]['orderId'] == 0
    assert tx.events['OrderCreated'][0]['owner'] == other
    assert tx.events['OrderCreated'][0]['orderType'] == 0
    assert tx.events['OrderCreated'][0]['token'] == mock_token.address
    assert tx.events['OrderCreated'][0]['amount'] == 5
    assert tx.events['OrderCreated'][0]['price'] == priceXToken


    tx = orderbook.createOrder(
            1, mock_token.address, 10,
            priceXToken, DAY_TIMESTAMP * 7,
            {"from": extra})
```

```
    assert tx.events['OrderCreated'][0]['orderId'] == 1
    assert tx.events['OrderCreated'][0]['owner'] == extra
    assert tx.events['OrderCreated'][0]['orderType'] == 1
    assert tx.events['OrderCreated'][0]['token'] == mock_token.address
    assert tx.events['OrderCreated'][0]['amount'] == 10
    assert tx.events['OrderCreated'][0]['price'] == priceXToken

    # 0,05

def test_cancel_order(only_local):
    # Arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)

    mock_token = deploy_erc(owner, "Mock", "MCK")
    orderbook = deploy_orderbook(owner)

    # mint some tokens
    mock_token.mint(other, 10e18)
    mock_token.approve(orderbook.address, 100e18, {"from": other})
    mock_token.mint(extra, 10e18)
    mock_token.approve(orderbook.address, 100e18, {"from": extra})

    priceXToken = 1e16

    tx = orderbook.createOrder(
            0, mock_token.address, 5,
            priceXToken, DAY_TIMESTAMP * 7,
```

```
        {"from": other, "value": priceXToken * 6})
order_id_1 = tx.events['OrderCreated'][0]['orderId']


tx = orderbook.createOrder(
        1, mock_token.address, 10,
        priceXToken, DAY_TIMESTAMP * 7,
        {"from": extra})
order_id_2 = tx.events['OrderCreated'][0]['orderId']


with reverts("Order does not exist"):
    orderbook.cancelOrder(5, {"from": other})
with reverts("Not order owner"):
    orderbook.cancelOrder(order_id_1, {"from": owner})


tx = orderbook.cancelOrder(order_id_1, {"from": other})
assert tx.events['OrderCancelled'][0]['orderId'] == order_id_1


with reverts("Order not active"):
    orderbook.cancelOrder(order_id_1, {"from": other})


tx = orderbook.cancelOrder(order_id_2, {"from": extra})
assert tx.events['OrderCancelled'][0]['orderId'] == order_id_2


with reverts("Order not active"):
    orderbook.cleanupExpiredOrder(order_id_2)


tx = orderbook.createOrder(
        0, mock_token.address, 5,
        priceXToken, DAY_TIMESTAMP * 7,
```

```
            {"from": other, "value": priceXToken * 6})
    order_id = tx.events['OrderCreated'][0]['orderId']
    increase_timestamp(DAY_TIMESTAMP * 8)

    with reverts("Order expired"):
        orderbook.cancelOrder(order_id, {"from": other})

    tx = orderbook.createOrder(
            1, mock_token.address, 10,
            priceXToken, DAY_TIMESTAMP * 1,
            {"from": extra})
    order_id = tx.events['OrderCreated'][0]['orderId']
    increase_timestamp(DAY_TIMESTAMP * 5)

    tx = orderbook.cleanupExpiredOrder(order_id)
    assert tx.events['Transfer'][0]['from'] == orderbook.address
    assert tx.events['Transfer'][0]['to'] == extra
    assert tx.events['Transfer'][0]['value'] == 10

def test_fulfill_matched_orders(only_local):
    # Arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)

    mock_token = deploy_erc(owner, "Mock", "MCK")
    orderbook = deploy_orderbook(owner)

    # mint some tokens
```

```
mock_token.mint(other, 10e18)
mock_token.approve(orderbook.address, 100e18, {"from": other})
mock_token.mint(extra, 10e18)
mock_token.approve(orderbook.address, 100e18, {"from": extra})


priceXToken = 1e16


with reverts("Caller is not whitelisted"):
    orderbook.fulfillMatchedOrders(
        [], {"from": other})
with reverts("No order matches to fulfill"):
    orderbook.fulfillMatchedOrders(
        [], {"from": owner})
with reverts("Order does not exist"):
    orderbook.fulfillMatchedOrders(
        [[0,0,10]], {"from": owner})


tx = orderbook.createOrder(
        0, mock_token.address, 5,
        1.5e16, DAY_TIMESTAMP * 7,
        {"from": other, "value": priceXToken * 10})
order_id_1 = tx.events['OrderCreated'][0]['orderId']


tx = orderbook.createOrder(
        1, mock_token.address, 10,
        priceXToken, DAY_TIMESTAMP * 7,
        {"from": extra})
order_id_2 = tx.events['OrderCreated'][0]['orderId']
```

```
    with reverts("Invalid fill amount"):
        orderbook.fulfillMatchedOrders(
            [[order_id_1,order_id_2,0]], {"from": owner})
    with reverts("Fill amount exceeds available"):
        orderbook.fulfillMatchedOrders(
            [[order_id_1,order_id_2,15]], {"from": owner})


    tx = orderbook.fulfillMatchedOrders(
            [[order_id_1,order_id_2,5]], {"from": owner})
    assert tx.events['Transfer'][0]['from'] == orderbook.address
    assert tx.events['Transfer'][0]['to'] == other
    assert tx.events['Transfer'][0]['value'] == 5
    assert tx.events['OrderFulfilled'][0]['orderId'] == order_id_1
    assert tx.events['OrderFulfilled'][0]['amount'] == 5


    orderbook.cancelOrder(order_id_2, {"from": extra})


    with reverts("Orders must be active"):
        orderbook.fulfillMatchedOrders(
            [[order_id_1,order_id_2,5]], {"from": owner})
```

Technical Findings Summary

Findings

Vulnerability Level		Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
	HIGH	11		1	3		7
	MEDIUM	9			5		4
	LOW	3			3		
	INFORMATIONAL	1			1		

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

Following our comprehensive security audit of the token contract for the Gradient project, The smart contract security audit has been successfully completed. The development team has addressed and resolved the majority of the critical findings. The remaining issues were acknowledged by the team, who have opted not to take further action at this stage, having accepted the associated residual risks.
Based on the fixes implemented and the team's response, the audit is considered passed. We recommend ongoing security after the deployment and continuous monitoring in mainnet.

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.