SMART CONTRACTS REVIEW

January 9th 2025 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING EVOQ FINANCE

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 6 Medium issues: 3 resolved and 3 acknowledged = - 9 points deducted
- 2 Low issues: 1 resolved and 1 acknowledged = - 1 points deducted
- 2 Informational issues: 2 acknowledged = 0 points deducted

Thus, 100 - 9 - 1 = 90

TECHNICAL SUMMARY

This document outlines the overall security of the Evoq Finance smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Evoq Finance smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Evoq Finance team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Evoq Finance repository:
Repo: https://github.com/evoqfinance/evoq-finance-contracts

Last commit - 8508e40726f0d2a7b6817b5cc070a8f84070dee6

Contracts under the scope:

- Evoq.sol
- EvoqGovernance.sol
- EvoqStorage.sol
- EvoqUtils.sol
- MatchingEngine.sol
- PositionsManager.sol
- InterestRatesManager.sol
- InterestRatesModel.sol
- RewardsManager.sol
- Treasury.sol
- WBNBGateway.sol
- Lens.sol
- LensExtension.sol
- LensStorage.sol
- MarketLens.sol
- IndexesLens.sol
- RateLens.sol
- RewardLens.sol
- UsersLens.sol
- DataLens.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Evoq Finance smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contract/s by industry leaders.
03	Thorough manual review of the codebase line by line.

Executive Summary

The **Codebase** is inspired by the Morpho optimizer and serves as a p2p-based lending optimizer that operates alongside existing pool-based lending protocols, such as Aave, Compound, and Venus, to provide better rates for both suppliers and borrowers through peer-to-peer (p2p) matching. It is designed to enhance performance tailored to the network environment while maintaining compatibility with existing infrastructure.

Evoq is the primary contract that inherits from **EvoqGovernance** and contains the entrypoints to supply, borrow, repay and withdraw funds. **EvoqGovernance** contains governance functions such as setter functions and the createMarket function and in turn inherits from **EvoqUtils** which is an upgradeable contract inheriting from **EvoqStorage** which contains the storage variables. **EvoqUtils** contains utility functions and modifiers and uses a Doubly Linked List data structure in its operations of storing and managing user positions. The **InterestRatesManager** contract handles the computation of indexes used for peer-to-peer interactions and inherits from EvoqStorage so that Evoq can delegate calls to this contract. The **MatchingEngine** is responsible for matching as well as unmatching borrowers and supplier from and to P2P.

The **PositionsManager** contains the main logic and implementation of the supply, borrow, withdraw, repay and liquidate functionalities. **RewardsManager** is an upgradeable contract that manages user rewards in corresponding token (here XVS from Venus protocol). And the treasury is where the revenue of the Evoq protocol gets stored. Lens contracts are utility/ helper contracts that provide view/read-only functions to query protocol data such as market indexes, rates, in a more organized way. The **LensExtension** is a custom contract that extends this Lens functionality and has getter functions to get unclaimed XVS rewards,accrued XVS rewards, etc. In addition to this, the **WBNBGateway** contract interacts with evoq contract and allows wrapping and unwrapping of BNB when interacting with evoq.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed.Acknowledged means that the issue was sent to the Evoq Finance team and the Evoq Finance team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Unrestricted “On-Behalf” Borrowing	Medium	Resolved
2	Unrestricted “On-Behalf” Withdrawal	Medium	Resolved
3	Missing Address Validation in Setters	Medium	Resolved
4	Deprecated Markets May Prevent Liquidations When isLiquidateBorrowPaused Is True	Medium	Acknowledged
5	ERC20 with Transfer Fees Not Handled in PositionsManager	Medium	Acknowledged
6	Cached Borrow Index in Liquidation Logic Leads to Discrepancy with Underlying Pool	Medium	Acknowledged
7	Redundant checks in EvoqUtils.sol	Low	Resolved
8	Ownership can be lost if transferred to an incorrect address	Low	Acknowledged
9	Funds will be stuck if the receiver is the contract itself	Informational	Acknowledged
10	Missing Validation for Zero toRepay in repayLogic	Informational	Acknowledged

Unrestricted "On-Behalf" Borrowing

Description

- **Evoq.sol →** `borrow()` / internal call `_borrow()`
- **PositionsManager.sol →** `borrowLogic()`

These functions allow specifying arbitrary `_borrower` and `_receiver` addresses without verifying that `msg.sender` is actually authorized to borrow on behalf of `_borrower`. As a result, an attacker can borrow against another user's collateral, sending the borrowed tokens to themselves.

Scenario:

1. The attacker sees that User A (victim) has a large collateral.
2. Attacker calls `borrow(...)` with `_borrower = User A` and `_receiver = Attacker`.
3. Protocol sees User A's collateral, deems the borrow safe, and transfers tokens to the attacker.
4. User A is left with the borrowed debt and no tokens.

PoC:

```
function testBorrow2Attack() public {
        uint256 amount = 10_000 ether;

        address bob = address(0x1234);
        address alice = address(0xdeff);
        console.log("Bob address:", bob);
        console.log("Alice address:", alice);
        deal(usdc, address(alice), INITIAL_BALANCE * WAD);
        uint256 aliceBalance = ERC20(usdc).balanceOf(address(alice));
        console.log(aliceBalance);
        vm.startPrank(alice);
        ERC20(usdc).approve(address(evoq), 2 * amount);
        evoq.supply(vUsdc, 2 * amount);

        vm.startPrank(bob);

        evoq.borrow(vUsdt, amount, alice, bob);
    }
```

In other words, Alice can supply funds to the protocol, and Bob can borrow funds on behalf of her(bob need not put any collateral into the protocol!). This will also mean that Alice's debt position increases instead of Bob.

Recommendation:

- Require `msg.sender == _borrower` or implement a robust delegation (e.g., signed approval) for on-behalf borrowing.
- Disallow free-form `_borrower` unless proper authorization is enforced.

MEDIUM-2 | RESOLVED

Unrestricted "On-Behalf" Withdrawal

Description (Contracts & Functions):

- **Evoq.sol →** `withdraw()` / internal call `_withdraw()`
- **PositionsManager.sol →** `withdrawLogic()`

These functions let the caller specify `_supplier` and `_receiver` with no checks confirming that `msg.sender` is `_supplier` or has `_supplier's` consent. This can allow an attacker to withdraw another user's supplied collateral to themselves.

Scenario:

1. Attacker calls `withdraw(...)` with `_supplier = Victim` and `_receiver = Attacker`.
2. The protocol sees Victim's collateral, processes withdrawal, and sends tokens to the attacker.
3. Victim's collateral is stolen with no action on Victim's part.

PoC:

In other words, if Alice supplies funds to the smart contract as a supplier, an unauthorized user Bob can withdraw her funds or collateral from the smart contract.

Recommendation:

- Enforce `msg.sender == _supplier` or require an explicit authorization mechanism for on-behalf withdrawals.
- Alternatively, provide distinct “withdraw self” vs. “withdraw on behalf” methods with strong permission checks.

Missing Address Validation in Setters

Description (Contracts & Functions):

- **EvoqGovernance.sol** (e.g., `setPositionsManager()`, `setTreasuryVault()`, etc.)

Certain governance or admin functions allow setting critical addresses (like `positionsManager`, `treasuryVault`) to the zero address. This breaks protocol logic if a call to address(0) silently no-ops or otherwise fails unexpectedly, yet returns success.

Scenario:

1. An owner or multisig calls `setPositionsManager(address(0))` by mistake.
2. The system's subsequent calls to `positionsManager` become calls to `address(0)`, effectively doing nothing or reverting unpredictably.
3. Operations dependent on `positionsManager` fail until corrected.

Recommendation:

- Add `require(_newAddress != address(0), "Zero address not allowed")` in all setters for critical addresses.
- Implement a timelock or thorough review step before finalizing critical parameter changes.

Deprecated Markets May Prevent Liquidations When isLiquidateBorrowPaused Is True

Context:

- **Contracts Involved:**
  - **EvoqGovernance.sol**
    - `setIsDeprecated(address _poolToken, bool _isDeprecated)`
  - **EvoqGovernance.sol**
    - `setIsLiquidateBorrowPaused(address _poolToken, bool _isPaused)`

Description: Currently, in the **EvoqGovernance** contract, when deprecating a market using the `setIsDeprecated` function, the contract ensures that borrowing is paused by checking `isBorrowPaused` before applying the deprecation flag. However, there is no corresponding check or logic to handle the `isLiquidateBorrowPaused` flag when a market is deprecated. This omission can lead to scenarios where a market is deprecated, but the `isLiquidateBorrowPaused` flag remains true, thereby preventing liquidators from liquidating borrowers in that deprecated market.

Scenario:

1. **Deprecating a Market:**
   - The contract owner calls `setIsDeprecated(_poolToken, true)` to deprecate a specific market.
   - The function checks that `isBorrowPaused` is true before allowing the deprecation.
   - The market is marked as deprecated by setting `marketPauseStatus[_poolToken].isDeprecated = true`.
2. **Impact on Liquidations:**
   - If `isLiquidateBorrowPaused` is already true for the deprecated market, liquidators are unable to liquidate borrowers in that market.
   - This situation undermines the protocol's risk management by allowing borrowers to maintain positions in a deprecated market without the ability to be liquidated, potentially leading to increased systemic risk.

Recommendation:

Modify the `setIsDeprecated` function to ensure that `isLiquidateBorrowPaused` is set to `false` when deprecating a market. This ensures that liquidations remain possible even after deprecation.

Client comment: This logic is intentional. Markets can be deprecated regardless of isLiquidateBorrowPaused is True or False. This gives operator more flexibility to pause/unpause liquidation borrow. For this reason, we prefer to leave things as it is.

MEDIUM-5 | ACKNOWLEDGED

ERC20 with Transfer Fees Not Handled in PositionsManager

Description:
The current implementation of `PositionsManager` does not account for ERC20 tokens with transfer fees. These tokens deduct a percentage of the transferred amount as a fee, causing the actual amount received by the contract to be less than the amount specified in the transfer call. As a result, functions like `supplyLogic`, `borrowLogic`, `repayLogic`, and `withdrawLogic` assume the full amount has been transferred or received, potentially leading to incorrect accounting and vulnerabilities.

For example:

- In `supplyLogic`, the contract updates the user's supply balance with the full amount specified by the user, but the actual transferred amount could be less due to fees.
- Similarly, in `repayLogic`, the amount credited towards repayment might be incorrect, leaving some debt unpaid.

This discrepancy can lead to mismatched balances, calculation errors, and a false sense of security in the protocol's state.

Recommendation:
the contract should dynamically calculate the actual amount received during token transfers. This can be done by comparing the contract's token balance before and after the transfer operation

Client comment: This logic is intentional. Markets can be deprecated regardless of whether isLiquidateBorrowPaused is True or False.

Cached Borrow Index in Liquidation Logic Leads to Discrepancy with Underlying Pool

Description:
The `_isLiquidatable` function in EvoqUtils uses cached borrow indexes `(lastPoolIndexes[_poolToken].lastBorrowPoolIndex)` to calculate user debt. These indexes are updated only during interactions with the market but may become outdated over time. As a result:

- Users who are liquidatable on the underlying pool may not be flagged as liquidatable in Evoq.
- Liquidators must first interact with the market (e.g., supply, borrow, or repay) to update the cached indexes before executing liquidations, adding friction to the process.

This discrepancy introduces inconsistency with the liquidation mechanics of the underlying pool (e.g. Venus) and reduces the efficacy of liquidators.

Recommendation:
Ensure that `_isLiquidatable` explicitly calls `_updateP2PIndexes` for all user-entered markets to align with real-time data.

implement an off-chain or on-chain mechanism to periodically update `lastPoolIndexes` for all markets to minimize reliance on liquidator-triggered updates.

Client comment: Under normal use case, this does not pose any issue. `_isLiquidatable` function is always called after `updateP2PIndexes` in the contract logic.

LOW-1 | RESOLVED

Redundant checks in EvoqUtils.sol

In **EvoqUtils.sol,** there are redundant checks as follows:
Redundant check on **line: 211** in **_supplyAllowed()-**

```
require(marketStatus[_poolToken].isCreated, "market not created");
```

The same redundant code exists on line: 240 in _borrowAllowed()

```
require(marketStatus[_poolToken].isCreated, "market not created");
```

These checks are not required and are redundant as these checks are already done on **line: 252** of **supplyLogic()** and on **line: 342** in **borrowLogic()** in **PositionsManager.sol** .

Recommendation:
It is advised to remove the redundant checks and code as mentioned above.

LOW-2 | ACKNOWLEDGED

Ownership can be lost if transferred to an incorrect address

The **EvoqStorage.sol** and **Treasury** contract does not implement a two-step process for transferring ownership. In its current state, ownership can be transferred in a single step, which can be risky as it could lead to accidental or malicious transfers of ownership without proper verification.

Recommendation:
It is advised to use Openzeppelin's Ownable2StepUpgradeable contract instead of OwnableUpgradeable to mitigate this issue.
Comment: The client said that under normal use case, this does not pose any issue and that this would be refactored at some point.

INFORMATIONAL-1 | ACKNOWLEDGED

Funds will be stuck if the receiver is the contract itself

In **PositionsManager.sol**, if the receiver in **borrowLogic()** or **withdrawLogic()** is set as address(this), then it can lead to funds being stuck forever. It may also lead to undiscovered accounting errors in the smart contract.

Recommendation:
It is advised add checks to disallow setting the receiver as address(this).

Comment: The client says that in normal case this won't pose any issue.

INFORMATIONAL-2 | ACKNOWLEDGED

Missing Validation for Zero `toRepay` in `repayLogic`

Description:
The `repayLogic` function calculates the `toRepay` value as the minimum of `_getUserBorrowBalanceInOf(_poolToken, _onBehalf)` and `_amount`. However, the function does not check whether `toRepay` is zero after this calculation. If `toRepay` is zero, the function continues to execute and incurs unnecessary gas costs without performing any meaningful repayment.

This issue arises when:
1. The user's debt for the specified `_poolToken` is already fully repaid (`_getUserBorrowBalanceInOf(_poolToken, _onBehalf) == 0`).
2. The `_amount` provided by the caller is greater than zero.

In such cases, the contract performs operations that ultimately have no effect, leading to wasted gas.

Recommendation:
Introduce a validation check after calculating `toRepay` to ensure it is greater than zero before proceeding. If `toRepay` is zero, revert the transaction to prevent unnecessary execution

	Evoq.sol EvoqGovernance.sol EvoqStorage.sol EvoqUtils.sol MatchingEngine.sol PositionsManager.sol InterestRatesManager.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	InterestRatesModel.sol RewardsManager.sol Treasury.sol WBNBGateway.sol Lens.sol LensExtension.sol LensStorage.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	MarketLens.sol IndexesLens.sol RateLens.sol RewardLens.sol UsersLens.sol DataLens.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the Evoq Finance team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Evoq Finance team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo