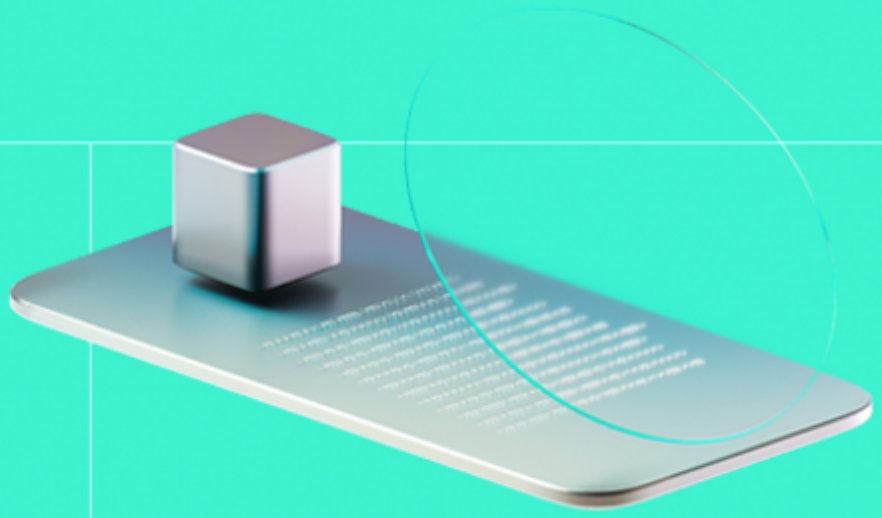
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Rebalance

Date: 27/06/2024

We express our gratitude to the Rebalance team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

The Rebalance Lending System is a set of smart contracts designed for managing and generating yield on assets deposited by users. It provides a vault-like structure where users can deposit assets, receive tokenized shares representing their stake, and earn interest from various yield-generating providers.

Document

Name	Smart Contract Code Review and Security Analysis Report for Rebalance
Audited By	Kaan Caglan
Approved By	Ataberk Yavuzer
Website	https://www.rebalance.finance/
Changelog	25/06/2024 - Preliminary Report
	27/06/2024 - Final Report
Platform	EVM
Language	Solidity
Tags	Lending, ERC20
Methodology	https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/REBALANCE-Finance/lending-contracts
Commit	dd4033d

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

1	0	1	0
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	0
Medium	1
Low	0

Vulnerability	Status
F-2024-3983 - Fee-on-Transfer Accounting-Related Issues	Accepted

Documentation quality

- Functional requirements are missed.
- Technical description is not provided.

Code quality

- Some of the best practices are missing.

Test coverage

Code coverage of the project is **90.96%** (branch coverage),.

- Not all branches are covered with tests.

Table of Contents

System Overview

The Rebalance Lending System is a set of smart contracts designed for managing and generating yield on assets deposited by users. It provides a vault-like structure where users can deposit assets, receive tokenized shares representing their stake, and earn interest from various yield-generating providers. The system also includes functionality for locking tokens, managing vaults, and interacting with yield providers.

Contracts

InterestVaultV1

- **Definition**: An abstract ERC4626-compliant vault contract defining common functions and interfaces for all vault types.
- **Functions**:
  - `deposit`, `mint`, `withdraw`, `redeem`: Basic ERC4626 functions for managing user deposits and withdrawals.
  - `initializeVaultShares`: Initializes vault shares with a specified amount of assets.
  - `setActiveProvider`, `setDepositLimits`, `setTreasury`, `setWithdrawFee`, `setMinAmount`: Admin functions for setting various parameters.
  - `rebalance`: Rebalances assets across providers.
- **Attributes**:
  - `_asset`: The main ERC20 asset managed by this vault.
  - `_underlyingDecimals`: Decimals of the underlying asset.
  - `_providers`: Array of yield providers.
  - `activeProvider`: Currently active provider for yield generation.
  - `minAmount`, `vaultDepositLimit`, `userDepositLimit`, `withdrawFeePercent`, `treasury`: Configuration parameters for deposits and withdrawals.
  - `initialized`: Boolean indicating if the vault has been initialized.
- **Privileged Roles**:
  - `DEFAULT_ADMIN_ROLE`: Full administrative access.
  - `REBALANCER_ROLE`: Role allowed to perform rebalancing operations.

InterestLocker

- **Definition**: A contract to lock and unlock ERC20 tokens, intended for locking rebalancer tokens.
- **Functions**:
  - `lockTokens`: Allows users to lock tokens for a specified duration.
  - `unlockTokens`: Allows users to unlock tokens after the lock duration has passed.
  - `setTokens`: Admin function to set supported tokens.
- **Attributes**:
  - `MIN_DURATION`: Minimum duration for locking tokens (30 days).
  - `nextLockId`: Incremental ID for tracking locks.
  - `_tokens`: Array of supported tokens.
  - `lockInfo`: Mapping of lock ID to lock information.
  - `_beneficiaries`: Mapping of lock ID to beneficiary address.
  - `_totalLocked`: Mapping of token address to total locked amount.
- **Privileged Roles**:

- `owner`: Only the contract owner can set supported tokens.

VaultManager

- **Definition**: Manages the rebalancing of vaults.
- **Functions**:
  - `rebalanceVault`: Rebalances assets across providers within a vault.
- **Privileged Roles**:
  - `DEFAULT_ADMIN_ROLE`: Full administrative access.
  - `EXECUTOR_ROLE`: Role allowed to execute rebalancing operations.

VaultRebalancerV1

- **Definition**: An implementation vault that handles pooled single-sided asset lending strategies for yield generation.
- **Functions**:
  - `rebalance`: Rebalances assets across providers.
- **Attributes**:
  - Inherits attributes and functions from `InterestVaultV1`.

Attributes

- **Tokens**:
  - `name`: The name of the token-shares managed in the vault.
  - `symbol`: The symbol of the token-shares managed in the vault.
  - `decimals`: The number of decimals used to get user representation.
  - `totalSupply`: The total supply of token-shares in the vault.
  - `balanceOf`: The balance of token-shares held by a user.
- **Providers**: Various yield-generating providers like AaveV3Arbitrum.
  - Functions to interact with providers include `deposit`, `withdraw`, `getDepositBalance`, `getDepositRateFor`, `getOperator`, and `getProviderName`.

Privileged Roles

- **DEFAULT_ADMIN_ROLE**: Complete control over the system, including setting active providers, deposit limits, treasury, and withdrawal fees.
- **REBALANCER_ROLE**: Authorized to perform rebalancing of assets across providers.
- **EXECUTOR_ROLE**: Allowed to execute rebalancing operations.
- **Owner**: In `InterestLocker`, the owner can set supported tokens for locking.

This structured system ensures secure and efficient management of user deposits, yield generation through multiple providers, and controlled access for administrative tasks.

Risks

- `block.number` means different things on different L2s.
- **Scope Definition and Security Guarantees**: The audit does not cover all code in the repository. Contracts outside the audit scope may introduce vulnerabilities, potentially impacting the overall security due to the interconnected nature of smart contracts.
- **Dependency on External Logic for Implemented Logic**: The implemented `InterestVaultV1` logic highly depends on external contracts not covered by the audit. This reliance introduces risks if these external contracts are compromised or contain vulnerabilities, affecting the audited project's integrity.
- **Interactions with External DeFi Protocols**: Dependence on external DeFi protocols inherits their risks and vulnerabilities. This might lead to direct financial losses if these protocols are exploited, indirectly affecting the audited project.

Findings

Vulnerability Details

F-2024-3983 - Fee-on-Transfer Accounting-Related Issues - Medium

Description: The function below transfer funds from the caller to the receiver via `safeTransferFrom()`, but do not ensure that the actual number of tokens received is the same as the input amount to the transfer. If the token is a fee-on-transfer token, the balance after the transfer will be smaller than expected, leading to accounting issues. Even if there are checks later, related to a secondary transfer, an attacker may be able to use latent funds (e.g. mistakenly sent by another user) in order to get a free credit. One way to address this problem is to measure the balance before and after the transfer, and use the difference as the amount, rather than the stated amount.

```
Path: ./contracts/InterestLocker.sol

106:        IERC20(token).safeTransferFrom(msg.sender, address(this), amount);
```

Assets:

- contracts/InterestLocker.sol [https://github.com/REBALANCE-Finance/lending-contracts]

Status: Accepted

Classification

Impact: 3/5

Likelihood: 3/5

Exploitability: Independent

Complexity: Simple

Severity: Medium

Recommendations

Remediation: To mitigate potential vulnerabilities and ensure accurate accounting with fee-on-transfer tokens, modify your contract's token transfer logic to measure the recipient's balance before and after the transfer. Use this

observed difference as the actual transferred amount for any further logic or calculations.

Resolution: This issue is acknowledged because the Rebalance team commented that they won't use any Fee-on-Transfer (FoT) tokens in the `InterestLocker` contract.

Observation Details

F-2024-3984 - Missing checks for `address(0)` - Info

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "zero address". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address. The Missing zero address control issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.
For instance, a contract might allow tokens to be sent to the zero address without any checks, which essentially burns those tokens as they become irretrievable. While sometimes this is intentional, without proper control or checks, accidental transfers could occur.

```
_providerManager = IProviderManager(providerManager_);
```

Assets:

- contracts/providers/arbitrum/CompoundV3Arbitrum.sol [https://github.com/REBALANCE-Finance/lending-contracts]
- contracts/providers/arbitrum/LodestarArbitrum.sol [https://github.com/REBALANCE-Finance/lending-contracts]

Status: Fixed

Recommendations

Remediation: It is strongly recommended to implement checks to prevent the zero address from being set during the initialization of contracts. This can be achieved by adding require statements that ensure address parameters are not the zero address.

Resolution: The finding was fixed in the commit `1c51d62` by the Rebalance team after zero address checks were implemented.

F-2024-3985 - Use `Ownable2Step` rather than `Ownable` - Info

Description: Ownable2Step and Ownable2StepUpgradeable prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner permissions actively accept via a contract call of its own.

```
contract InterestLocker is Ownable {
```

Assets:

- contracts/InterestLocker.sol [https://github.com/REBALANCE-Finance/lending-contracts]

Status: Fixed

Recommendations

Remediation: Consider using `Ownable2Step` or `Ownable2StepUpgradeable` from OpenZeppelin Contracts to enhance the security of your contract ownership management. These contracts prevent the accidental transfer of ownership to an address that cannot handle it, such as due to a typo, by requiring the recipient of owner permissions to actively accept ownership via a contract call. This two-step ownership transfer process adds an additional layer of security to your contract's ownership management

Resolution: The finding was fixed in the commit `f782d87` by the Rebalance team with using `Ownable2Step`.

F-2024-3986 - Cache State Variable Array Length In For Loop - Info

Description:

Failing to cache the array length when iterating through arrays in Solidity can have significant performance and gas cost implications. In Solidity, array lengths can change during execution due to external calls or storage modifications. When the array length is not cached before entering a loop, it is recomputed with each iteration, leading to unnecessary gas consumption.

```
for (uint i = 0; i < _tokens.length; i++) {

for (uint256 i = 0; i < _providers.length; i++) {
```

Assets:

- contracts/InterestLocker.sol [https://github.com/REBALANCE-Finance/lending-contracts]
- contracts/abstracts/InterestVaultV1.sol [https://github.com/REBALANCE-Finance/lending-contracts]

Status: Fixed

Recommendations

Remediation:

To enhance performance and reduce gas costs, cache the array length before entering a for loop in Solidity. This approach prevents repeated computation of the array length and mitigates the risk of reentrancy attacks due to array length changes during loop execution.

Resolution:

The finding was fixed in the commit `1e50eb0` by the Rebalance team with caching array length.

F-2024-3987 - Custom Errors in Solidity for Gas Efficiency - Info

Description:

Starting from Solidity version 0.8.4, the language introduced a feature known as "custom errors". These custom errors provide a way for developers to define more descriptive and semantically meaningful error conditions without relying on string messages. Prior to this version, developers often used the `require` statement with string error messages to handle specific conditions or validations. However, every unique string used as a revert reason consumes gas, making transactions more expensive.

Custom errors, on the other hand, are identified by their name and the types of their parameters only, and they do not have the overhead of string storage. This means that, when using custom errors instead of `require` statements with string messages, the gas consumption can be significantly reduced, leading to more gas-efficient contracts.

```
require(borrowRateMantissa <= 0.0005e16, "RATE_TOO_HIGH")
```

Assets:

- contracts/libraries/LibCompoundV2.sol [https://github.com/REBALANCE-Finance/lending-contracts]

Status: Fixed

Recommendations

Remediation: It is recommended to use custom errors instead of revert strings to reduce gas costs, especially during contract deployment. Custom errors can be defined using the error keyword and can include dynamic information.

Resolution: The finding was fixed in the commit `de7f83f` by the Rebalance team by using custom errors instead of `require` statements.

F-2024-3988 - Unused Error Definition - Info

Description: The error `error SiloArbitrum__AssetsZero();` is declared, but never used.

This leaves redundant logic in code.

Assets:

- contracts/providers/arbitrum/SiloArbitrum.sol [https://github.com/REBALANCE-Finance/lending-contracts]

Status: Fixed

Recommendations

Remediation: Unused `error` definitions should be removed from the contract, and if needed, consolidated into a separate file to avoid duplication.

Resolution: The finding was fixed in the commit `19d2247` by the Rebalance team with removing unused error definitions.

F-2024-3989 - Optimization of Loop Control for Early Termination - Info

Description:

The identified issue is a common inefficiency in programming related to loop control during search operations. In the current implementation, loops are designed to traverse through the entire dataset (like arrays or lists) even after the required condition has been met or the target element has been found. This lack of early termination results in unnecessary processing and can lead to increased execution times, particularly in large datasets. The issue is not about the correctness of the function but its efficiency, as continuing the loop after meeting the required condition is redundant and wastes computational resources.

```
        for (uint i = 0; i < _tokens.length; i++) {
            if (_tokens[i] == token) {
                isValid = true;
```

Assets:

- contracts/InterestLocker.sol [https://github.com/REBALANCE-Finance/lending-contracts]
- contracts/abstracts/InterestVaultV1.sol [https://github.com/REBALANCE-Finance/lending-contracts]

Status: Fixed

Recommendations

Remediation:

To optimize such functions, it is recommended to incorporate an early exit mechanism within the loop. This can be achieved by introducing a `break` statement (or an equivalent control structure) immediately after the condition is satisfied or the target element is located.

Resolution:

The finding was fixed in the commit `bda7d1a` by the Rebalance team with a breaking loop after the condition is satisfied.

F-2024-3990 - Avoid Using State Variables Directly in `emit` for Gas Efficiency - Info

Description:

In Solidity, emitting events is a common way to log contract activity and changes, especially for off-chain monitoring and interfacing. However, using state variables directly in `emit` statements can lead to increased gas costs. Each access to a state variable incurs gas due to storage reading operations. When these variables are used directly in `emit` statements, especially within functions that perform multiple operations, the cumulative gas cost can become significant. Instead, caching state variables in memory and using these local copies in `emit` statements can optimize gas usage.

```
emit FeesCharged(treasury, assets
```

`treasury` can be cached here.

Assets:

- contracts/VaultRebalancerV1.sol [https://github.com/REBALANCE-Finance/lending-contracts]
- contracts/abstracts/InterestVaultV1.sol [https://github.com/REBALANCE-Finance/lending-contracts]

Status: Fixed

Recommendations

Remediation: To optimize gas efficiency, cache state variables in memory when they are used multiple times within a function, including in `emit` statements.

Resolution: The finding was fixed in the commit `6eeb30a` by the Rebalance team with caching state variables before using them in an emit statement.

F-2024-3991 - Unneeded initializations of uint256 and bool variable to 0/false - Info

Description: In Solidity, it is common practice to initialize variables with default values when declaring them. However, initializing `uint256` variables to `0` and `bool` variables to `false` when they are not subsequently used in the code can lead to unnecessary gas consumption and code clutter. This issue points out instances where such initializations are present but serve no functional purpose.

```
for (uint256 i = 0;
```

Assets:

- contracts/InterestLocker.sol [https://github.com/REBALANCE-Finance/lending-contracts]
- contracts/abstracts/InterestVaultV1.sol [https://github.com/REBALANCE-Finance/lending-contracts]

Status: Fixed

Recommendations

Remediation: It is recommended not to initialize integer variables to `0` to and boolean variables to `false` to save some Gas.

Resolution: The finding was fixed in the commit `cb49d6e` by the Rebalance team with not initializing integer variables to `0`.

F-2024-3992 - Event is not properly `indexed` - Info

Description: Index event fields make the field more quickly accessible to off-chain tools that parse events. This is especially useful when it comes to filtering based on an address. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Where applicable, each `event` should use three `indexed` fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three applicable fields, all of the applicable fields should be `indexed`.

```
Path: ./contracts/InterestLocker.sol

45:    event TokensLocked(
46:        uint256 lockId,
47:        address user,
48:        address token,
49:        uint256 amount,
50:        uint256 duration
51:    );

52:    event TokensUnlocked(
53:        uint256 lockId,
54:        address user,
55:        address token,
56:        uint256 amount
57:    );
```

Assets:

- contracts/InterestLocker.sol [https://github.com/REBALANCE-Finance/lending-contracts]

Status: Fixed

Recommendations

Remediation: Enhance smart contract efficiency post-deployment by utilizing indexed events. This approach aids in efficiently tracking contract activities, significantly contributing to the reduction of gas costs.

Resolution: The finding was fixed in the commit `eff5f8f` by the Rebalance team with indexing the event parameters.

F-2024-3993 - Inefficient Use of String Parameter in Internal Function - Info

Description: The `_checkProvidersBalance` function is designed to accept a string parameter `method`, which it uses to dynamically generate a function signature for querying the balance of multiple providers. However, this function is only called by the `totalAssets` function with the hardcoded value `"getDepositBalance"`. This design incurs unnecessary gas costs due to the overhead of encoding and decoding the method string at runtime.

```
function _checkProvidersBalance(
    string memory method
) internal view returns (uint256 assets) {
    bytes memory data = abi.encodeWithSignature(
        string(abi.encodePacked(method, "(address,address)")),
```

Assets:

- contracts/abstracts/InterestVaultV1.sol [https://github.com/REBALANCE-Finance/lending-contracts]

Status: Fixed

Recommendations

Remediation: Refactor the `_checkProvidersBalance` function to remove the string parameter and directly use the `"getDepositBalance"` method within the function. This will optimize gas usage by eliminating the need for dynamic string handling.

```
function _checkProvidersBalance() internal view returns (uint256 assets) {
        bytes memory data = abi.encodeWithSignature(
            string(abi.encodePacked("getDepositBalance(address,address)")),
```

Resolution: The finding was fixed in the commit `d9ac96c` by the Rebalance team after the code was refactored regarding the given recommendation.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details	
Repository	https://github.com/REBALANCE-Finance/lending-contracts
Commit	dd4033dad8d77595565f8fdfd510d83831bb7377
Whitepaper	N/A
Requirements	N/A
Technical Requirements	N/A

Contracts in Scope
./contracts/VaultRebalancerV1.sol
./contracts/VaultManager.sol
./contracts/InterestLocker.sol
./contracts/abstracts/VaultPermit.sol
./contracts/abstracts/VaultPausable.sol
./contracts/abstracts/InterestVaultV1.sol
./contracts/libraries/LibCompoundV2.sol
./contracts/providers/ProviderManager.sol
./contracts/providers/arbitrum/DolomiteArbitrum.sol
./contracts/providers/arbitrum/AaveV3Arbitrum.sol
./contracts/providers/arbitrum/LodestarArbitrum.sol
./contracts/providers/arbitrum/RadiantV2Arbitrum.sol
./contracts/providers/arbitrum/SiloArbitrum.sol
./contracts/providers/arbitrum/CompoundV3Arbitrum.sol