LIDO

LIDO EARN SMART CONTRACTS SECURITY AUDIT REPORT

OXORIO

JANUARY 15, 2026

1 EXECUTIVE SUMMARY

OXORIO

1.1 EXECUTIVE SUMMARY

This document presents the smart contracts security audit conducted by Oxorio for Lido Earn project.

Lido is a leading decentralized liquid staking provider that enables users to stake digital assets while retaining liquidity via tokenized staking positions.

Lido Earn is an ERC4626-compliant vault infrastructure that enables integration of arbitrary ERC4626 yield strategies into wallets and applications. The system provides a foundation for yield aggregation with comprehensive security controls, fee management, and reward distribution.

The audit process involved a comprehensive approach, including manual code review, automated analysis, and extensive testing and simulations of the smart contracts to assess the project's security and functionality. The audit covered a total of 4 smart contracts, encompassing 618 lines of code. The codebase was thoroughly examined, with the audit team collaborating closely with Lido and referencing the provided documentation to address any questions regarding the expected behavior. For an in-depth explanation of used the smart contract security audit methodology, please refer to the Security Assessment Methodology section of this document.

Throughout the audit, a collaborative approach was maintained with Lido to address all concerns identified within the audit's scope. Each issue has been either resolved or formally acknowledged by Lido, contributing to the robustness of the project.

As a result, following a comprehensive review, our auditors have verified that the Lido Earn, as of audited commit fc15b104d3859955ed341e2785059e2806c6aa36, has met the security and functionality requirements established for this audit, based on the code and documentation provided, and operates as intended within the defined scope.

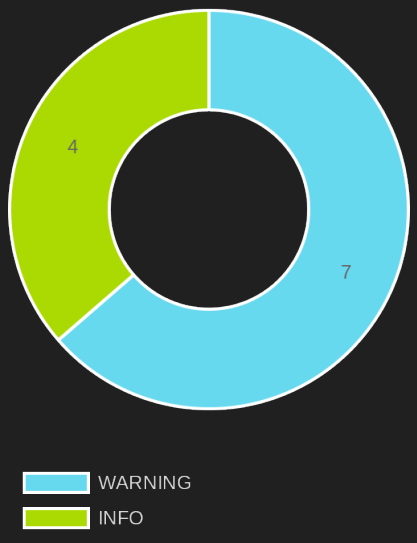
1.2 SUMMARY OF FINDINGS

The table below provides a comprehensive summary of the audit findings, categorizing each by status and severity level. For a detailed description of the severity levels and statuses of findings, see the Findings Classification Reference section.

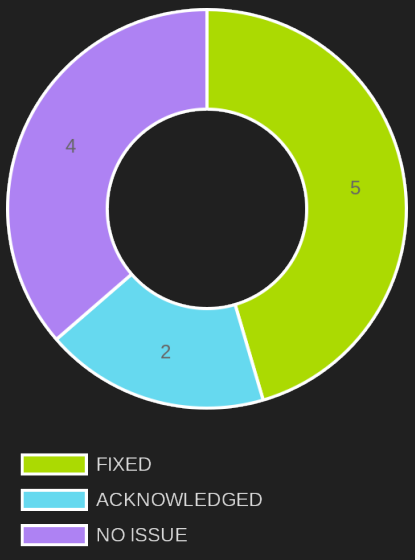
Detailed technical information on the audit findings, along with our recommendations for addressing them, is provided in the Findings Report section for further reference.

All identified issues have been addressed, with Lido fixing them or formally acknowledging their status.

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	**0**	0	0	0	0
MAJOR	**0**	0	0	0	0
WARNING	**7**	0	3	1	3
INFO	**4**	0	2	1	1
TOTAL	**11**	**0**	**5**	**2**	**4**



Issue distribution by severity



Issue distribution by status

2 AUDIT OVERVIEW

CONTENTS

2.1 DISCLAIMER

At the request of the client, Oxorio consents to the public release of this audit report. The information contained herein is provided "as is" without any representations or warranties of any kind. Oxorio disclaims all liability for any damages arising from or related to the use of this audit report. Oxorio retains copyright over the contents of this report.

This report is based on the scope of materials and documentation provided to Oxorio for the security audit as detailed in the Executive Summary and Audited Files sections. The findings presented in this report may not encompass all potential vulnerabilities. Oxorio delivers this report and its findings on an as-is basis, and any reliance on this report is undertaken at the user's sole risk. It is important to recognize that blockchain technology remains in a developmental stage and is subject to inherent risks and flaws.

This audit does not extend beyond the programming language of smart contracts to include areas such as the compiler layer or other components that may introduce security risks. Consequently, this report should not be interpreted as an endorsement of any project or team, nor does it guarantee the security of the project under review.

THE CONTENT OF THIS REPORT, INCLUDING ITS ACCESS AND/OR USE, AS WELL AS ANY ASSOCIATED SERVICES OR MATERIALS, MUST NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE. Third parties should not rely on this report for making any decisions, including the purchase or sale of any product, service, or asset. Oxorio expressly disclaims any liability related to the report, its contents, and any associated services, including, but not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Oxorio does not warrant, endorse, or take responsibility for any product or service referenced or linked within this report.

For any decisions related to financial, legal, regulatory, or other professional advice, users are strongly encouraged to consult with qualified professionals.

2.2 PROJECT BRIEF

Title	Description
Client	Lido
Project name	Lido Earn
Category	Yield Vault Infrastructure
Website	https://lido.fi/
Documentation	https://github.com/lidofinance/defi-interface/tree/main/docs
Repository	https://github.com/lidofinance/defi-interface/
Initial Commit	`831fb72cb3a5a9ec0ae6d5b19848ac4e15e0311a`
Final Commit	`fc15b104d3859955ed341e2785059e2806c6aa36`
Platform	L1
Network	Ethereum
Languages	Solidity
Lead Auditor	Artem Belozerov - artem@oxor.io
Project Manager	Elena Kozmiryuk - elena@oxor.io

2.3 PROJECT TIMELINE

The key events and milestones of the project are outlined below.

Date	Event
November 24, 2025	Client engaged Oxorio requesting an audit.
December 3, 2025	The audit team initiated work on the project.
December 8, 2025	Submission of the initial audit report.
December 9, 2025	Client's feedback on the initial audit report was received.
December 11, 2025	The audit team commenced work on a re-audit of the project.
December 12, 2025	Submission of the final audit report incorporating client's verified fixes.

2.4 AUDITED FILES

The following table contains a list of the audited files. The scc tool was used to count the number of lines and assess complexity of the files.

	File	Lines	Blanks	Comments	Code	Complexity
1	src/adapters/ERC4626Adapter.sol	228	32	109	**87**	18%
2	src/EmergencyVault.sol	372	47	209	**116**	25%
3	src/RewardDistributor.sol	281	56	115	**110**	23%
4	src/Vault.sol	722	128	289	**305**	30%
	Total	**1603**	**263**	**722**	**618**	**26%**

Lines: The total number of lines in each file. This provides a quick overview of the file size and its contents.

Blanks: The count of blank lines in the file.

Comments: This column shows the number of lines that are comments.

Code: The count of lines that actually contain executable code. This metric is essential for understanding how much of the file is dedicated to operational elements rather than comments or whitespace.

Complexity: This column shows the file complexity per line of code. It is calculated by dividing the file's total complexity (an approximation of cyclomatic complexity that estimates logical depth and decision points like loops and conditional branches) by the number of executable lines of code. A higher value suggests greater complexity per line, indicating areas with concentrated logic.

2.5 PROJECT OVERVIEW

Lido Earn provides an ERC4626-compliant middleware layer that wraps external ERC4626 vaults (e.g., Morpho strategies) and adds fee harvesting, reward distribution, inflation-attack defenses, and a robust emergency withdrawal/recovery flow. The system is structured as a reusable Vault core, an EmergencyVault extension, and concrete adapters (today focused on ERC4626 targets) so new strategies can reuse the same risk controls while exposing a standard interface to wallets, custodians, and integrators.

2.6 CODEBASE QUALITY ASSESSMENT

The Codebase Quality Assessment table offers a comprehensive assessment of various code metrics, as evaluated by our team during the audit, to gauge the overall quality and maturity of the project's codebase. By evaluating factors such as complexity, documentation and testing coverage to best practices, this table highlights areas where the project excels and identifies potential improvement opportunities. Each metric receives an individual rating, offering a clear snapshot of the project's current state, guiding prioritization for refactoring efforts, and providing insights into its maintainability, security, and scalability. For a detailed description of the categories and ratings, see the Codebase Quality Assessment Reference section.

Category	Assessment	Result
Access Control	The contracts implement well-defined role-based access controls with appropriate modifiers. Minor issues were noted in `I-01`, which may require additional attention to streamline role interactions.	**Good**
Arithmetic	Arithmetic operations are generally safe and correctly implemented.	**Excellent**
Complexity	The code is well-structured, with clear modularization and maintainable logic. Overall complexity is low, making the contracts readable and straightforward to audit.	**Excellent**
Data Validation	Input and parameter validations are largely sufficient.	**Excellent**
Decentralization	Contracts are administratively controlled by an account set at deployment. No decentralized governance mechanisms are implemented, and emergency functions rely on the administrator.	**Not Applicable**
Documentation	Documentation is generally clear and comprehensive.	**Excellent**
External Dependencies	The system demonstrates a generally robust reliance on ERC-4626-compliant external vaults, and most dependency-related observations do not introduce meaningful risks under the assumed standard-compliant integrations. The only notable concern is highlighted in `W-04`, where emergency withdrawals fully depend on the target vault's ability to execute `redeem`; aside from this edge case, no significant external-dependency risks were identified.	**Good**

Category	Assessment	Result
Error Handling	Error handling is implemented correctly, with descriptive revert messages and appropriate safeguards for all standard operations.	**Excellent**
Logging and Monitoring	Event logging is implemented for critical operations.	**Excellent**
Low-Level Calls	Low-level calls are not used in the contracts, and no delegatecall or assembly patterns are present.	**Not Applicable**
Testing and Verification	The contracts are well-covered by automated tests, providing confidence in the correctness of core functionalities and edge case handling.	**Excellent**

2.7 FINDINGS BREAKDOWN BY FILE

This table provides an overview of the findings across the audited files, categorized by severity level. It serves as a useful tool for identifying areas that may require attention, helping to prioritize remediation efforts, and provides a clear summary of the audit results.

File	TOTAL	CRITICAL	MAJOR	WARNING	INFO
src/RewardDistributor.sol	**4**	0	0	2	2
src/Vault.sol	**4**	0	0	3	1
src/adapters/ERC4626Adapter.sol	**4**	0	0	3	1
src/EmergencyVault.sol	**3**	0	0	2	1

2.8 CONCLUSION

A comprehensive audit was conducted on 4 smart contracts, revealing no critical or major vulnerabilities, while identifying several warnings and informational observations. The review highlighted a number of correctness and robustness concerns, primarily related to reward-distribution edge cases, emergency-flow reliability, fee-calculation accuracy, interaction safety with external ERC-4626 vaults, and consistency of internal accounting across protocol operations.

Following our initial audit, Lido worked closely with our team to address the identified issues. The proposed changes focused on improving validation of user- and admin-provided parameters, reinforcing role-management integrity, ensuring that reward- and fee-distribution mechanisms behave predictably under all conditions, and enhancing the protocol's resilience to unusual target-vault behavior. Additional improvements were introduced to align documentation with the implemented logic and to tighten safety around emergency and operational flows. Through multiple iterations of review and clarification, all issues have either been successfully fixed, mitigated, or formally acknowledged.

As a result, the project has passed our audit. Our auditors have verified that the Lido Earn, as of audited commit `fc15b104d3859955ed341e2785059e2806c6aa36`, operates as intended within the defined scope, based on the information and code provided at the time of evaluation. The robustness of the codebase has been significantly improved, meeting the necessary security and functionality requirements established for this audit.

2.9 AUDIT OBJECTIVES

The audit focused on identifying vulnerabilities, ensuring protocol correctness, and verifying the secure and reliable implementation of the `Lido Earn` smart contracts. Key areas of focus included:

2.9.1 Architecture and Inheritance

- Examination of the vault architecture, including the `Vault` base, `EmergencyVault` extension, and concrete adapter, to ensure proper use of inheritance and overrides across all critical functions.
- Analysis of why certain functions were not overridden at all levels of the inheritance hierarchy, ensuring consistent behavior and avoiding unintended side effects.

2.9.2 Access Control and Permissions

- Verification of role-based access control to confirm correct enforcement of permissions.
- Assessment of potential conflicts or overlaps between roles, ensuring that administrative and operational tasks (e.g., reward distribution, recipient updates, emergency withdrawals) are properly segregated.

2.9.3 Integration with External Protocols

- Evaluation of interactions with `TARGET_VAULT` (external ERC4626 vaults), including deposits, withdrawals, and emergency recovery flows.
- Assessment of potential failures or manipulations in target vaults, including cases such as third-party donations or externally triggered profit/loss events.

2.9.4 Token Handling and Arithmetic

- Examination of ERC20 operations, including deposits, withdrawals, and fee harvesting, for correctness and safety.
- Analysis of rounding, division, and precision issues that could lead to share dilution, fee miscalculations, or cumulative errors across operations.
- Verification of behavior with tokens having different decimal values to ensure consistent accounting.

2.9.5 Data Validation and Parameter Updates

- Checks on constructor inputs, setters, and critical functions to prevent invalid configurations, zero addresses, or inconsistent state between `TARGET_VAULT` and `Vault`.
- Validation of internal accounting variables, such as `lastTotalAssets`, ensuring they remain consistent with actual `totalAssets` in the target vault.

2.9.6 Emergency and Recovery Mechanisms

- Review of emergency withdrawal and recovery flows, including pausing/unpausing, partial liquidity scenarios, and late recovery processes.
- Assessment of potential issues arising when users attempt interactions during emergency mode or when assets are temporarily locked in external protocols.

2.9.7 Reentrancy and Attack Vectors

- Analysis of potential attack vectors, including inflation attacks, fee manipulation, desynchronization of accounting, and improper handling of donations to the vaults.
- Evaluation of the protocol's resistance to rounding errors, edge-case deposits/withdrawals, and other unexpected scenarios that could compromise user shares or protocol integrity.

2.9.8 Edge Cases and Risk Scenarios

- Simulation of unusual scenarios such as extreme profit/loss sequences, partial withdrawals, or misaligned asset accounting, to assess protocol resilience.
- Exploration of failures in input parameters, including extreme or out-of-bounds values, to ensure proper safeguards are in place.

2.9.9 Out of Scope

The following areas were explicitly excluded from the audit:

- Off-chain components, such as frontend integrations or external monitoring services, unless directly impacting on-chain logic.
- Detailed implementation of external dependencies like Morpho internals (e.g., full protocol exploits beyond integration points).
- Gas optimization beyond identifying critical inefficiencies tied to security risks.

3 FINDINGS REPORT

OXORIO

3.1 CRITICAL

No critical issues found.

3.2 MAJOR

No major issues found.

3.3 WARNING

W-01	User share dilution due to fees even in the absence of actual profit in `Vault`
Severity	**WARNING**
Status	• NO ISSUE

Location

File	Location	Line
Vault.sol	contract `Vault` > function `_calculateFeeShares`	`445`

Description

In the function `_calculateFeeShares` of the contract `Vault`, a percentage of profit is collected as a protocol fee. In this case, the profit is calculated as the difference between the current and the last recorded value of `totalAssets`:

```
uint256 profit = currentTotal - lastTotalAssets;
uint256 feeAmount = profit.mulDiv(rewardFee, MAX_BASIS_POINTS, Math.Rounding.Ceil);
```

When the value of `totalAssets` increases due to yield in `TARGET_VAULT`, the protocol takes its fee from this profit. On the other hand, if `TARGET_VAULT` incurs losses, the protocol does not collect any fees because there is no profit. However, the value of `lastTotalAssets` is still overwritten.

Assume that a profit of `X` is generated in `TARGET_VAULT`. The vault then incurs a loss of `Y` and later generates a profit of `Y` again. The `_harvestFees` function is called on each of these events. As a result, fee shares are minted to the `TREASURY` address both for `X` and for `Y`, even though the net profit of `TARGET_VAULT` is only `X`.

This can lead to a situation where `TARGET_VAULT` provides little or no real profit to users, but due to intermediate profit and loss events, the users' share is gradually diluted in favor of the `TREASURY`.

Recommendation

We recommend considering a fairer profit-fee collection mechanism for users, for example by not updating `lastTotalAssets` in the event of losses, i.e., by charging fees only when a new “high water mark” is exceeded.

It is important to note that this approach involves a trade-off: new deposits made during a drawdown would not generate performance fees until the historical share value is fully recovered, potentially creating a “free ride” effect for new investors. However, this trade-off may be considered acceptable, as it helps avoid direct losses for long-term depositors that could otherwise arise from charging fees on recovery.

Update

Client's response

The protocol collects its fee whenever rewards are accrued. A decrease in the share rate may indicate that the vault has incurred losses, which must then be socialized among all depositors. Once the losses are realized, the vault continues operating in its normal mode.

W-02	Distributed reward amount does not match the transferred amount in `RewardDistributor`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
RewardDistributor.sol	contract `RewardDistributor` > function `distribute`	`238`

Description

In the function `distribute` of the contract `RewardDistributor`, the balance of the specified token is distributed:

```
for (uint256 i = 0; i < recipientsLength; i++) {
    Recipient memory recipient = recipients[i];

    uint256 amount = (balance * recipient.basisPoints) / MAX_BASIS_POINTS;

    if (amount > 0) {
        tokenContract.safeTransfer(recipient.account, amount);
        emit RecipientPaid(recipient.account, token, amount);
    }
}

emit RewardsDistributed(token, balance);
```

However, a situation is possible where, for a small value of `balance`, the product `balance * recipient.basisPoints` is less than `MAX_BASIS_POINTS`, as a result of which `amount` will be equal to `0` due to rounding.

In this case, no actual token transfer will occur, while the `RewardsDistributed` event will still report that the full `balance` has been distributed. This may lead to incorrect behavior of monitoring and logging systems.

This is especially relevant, for example, for tokens with a small number of `decimals`. In addition, a small value of `recipient.basisPoints` may further exacerbate the issue.

Recommendation

We recommend explicitly emitting the actual amount of tokens transferred to recipients in the `RewardsDistributed` event instead of reporting the full `balance`.

Update

Client's response

Fixed at 766eefdd6e1c5c9fd17e26aec01cf2ccd3b392f5.

W-03	Fee Manipulation via `totalAssets` in `Vault`, `ERC4626Adapter`
Severity	**WARNING**
Status	• NO ISSUE

Location

File	Location	Line
Vault.sol	contract `Vault` > function `_harvestFees`	`413`
ERC4626Adapter.sol	contract `ERC4626Adapter` > function `totalAssets`	`102`

Description

The `_harvestFees` function calculates performance fee based on changes in `totalAssets`. In the adapter, `totalAssets` relies on `TARGET_VAULT.convertToAssets`. If the target vault is susceptible to manipulation (e.g., donation attack or read-only reentrancy), an attacker can temporarily "inflate" the asset value.

Attack scenario:

1. Attacker manipulates `TARGET_VAULT`, increasing the share price.
2. Calls a function that triggers `_harvestFees` (e.g., `deposit`).
3. `Vault` records false "profit" and mints a huge amount of shares to `TREASURY`.
4. The price normalizes, but user shares are diluted in favor of the protocol.

Recommendation

We recommend using a TWAP (time-weighted average price) or validating the `TARGET_VAULT` exchange rate deviation against a secure oracle or previous values, instead of relying on the instantaneous `convertToAssets` value for fee calculations.

Update

Client's response

This finding is planned to be addressed operationally by interacting only with immutable contracts that cannot manipulate the share price and that fully comply with the ERC-4626 standard.

Oxorio's response

Since the system is expected to interact only with immutable ERC-4626 vaults that cannot manipulate the share price, we agree that this issue does not apply. However, we recommend clearly documenting this requirement so that it is taken into account in future integrations.

W-04	Funds Lock in Emergency Mode in `ERC4626Adapter`, `EmergencyVault`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
ERC4626Adapter.sol	contract `ERC4626Adapter` > function `_emergencyWithdrawFromProtocol`	`183`
EmergencyVault.sol	contract `EmergencyVault` > function `emergencyWithdraw`	`188`

Description

The emergency withdrawal mechanism strictly depends on the target protocol's `redeem` function. Emergency Mode is often activated precisely because the target protocol is broken or paused. If `TARGET_VAULT` is paused, the `redeem` call will revert. User funds may become permanently locked in the `Vault`, as even the admin won't be able to extract them.

Recommendation

We recommend adding an alternative emergency path to the adapter which, instead of attempting redemption via `TARGET_VAULT.redeem`, transfers all available target vault shares to a controlled address (via `TARGET_VAULT.transfer`) and processes them either through a dedicated “rescue” contract or via an off-chain recovery procedure.

Update

Client's response

There is a strong dependency on the target vault’s ERC-4626 standard implementation, so compliance with this standard is assumed for any integrated vault. In addition, transferring shares to another account may carry legal implications.

W-05	Late Recovery Issues in `EmergencyVault`
Severity	**WARNING**
Status	• NO ISSUE

Location

File	Location	Line
EmergencyVault.sol	contract `EmergencyVault` > function `activateRecovery`	`236`
EmergencyVault.sol	contract `EmergencyVault` > function `_emergencyRedeem`	`335`

Description

When Recovery Mode is activated, the `recoveryAssets` variable is fixed (balance snapshot).

1. If funds arrive in the protocol *after* activation (e.g., unlock in an external protocol), they won't be accounted for in the user redemption formula.
2. `emergencyWithdraw` is blocked after Recovery activation, so these "late" funds cannot be retrieved.

Recommendation

We recommend adding a separate function to `EmergencyVault` (e.g., `recoverExcessAssets`), accessible by the `EMERGENCY_ROLE` and callable only in `recoveryMode`, which calculates the amount of assets required to fully satisfy current shareholders based on the fixed snapshot and transfers any excess to a specified admin address.

Update

Client's response

This finding is almost valid; however, in the case of ERC-4626 vaults, withdrawals are synchronous: the tokens are either received immediately (possibly over several transactions due to insufficient liquidity) or they will not arrive at all.

W-06	Desynchronization of `lastTotalAssets` accounting with the actual `totalAssets` balance in `Vault`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Vault.sol	contract `Vault` > function `deposit`	`237`
Vault.sol	contract `Vault` > function `mint`	`281`
Vault.sol	contract `Vault` > function `withdraw`	`320`
Vault.sol	contract `Vault` > function `redeem`	`361`

Description

At the specified locations, deposits to and withdrawals from the external protocol are performed. After that, the current value of `totalAssets` of the external `TARGET_VAULT` protocol is stored in the `lastTotalAssets` variable.

However, the update logic of `lastTotalAssets` differs between deposits and withdrawals. In the case of a deposit, `lastTotalAssets` is increased by the amount of assets provided by the user via `assetsToDeposit`:

```
uint256 protocolSharesReceived = _depositToProtocol(assetsToDeposit);
// ...

lastTotalAssets += assetsToDeposit;
```

In the case of a withdrawal, `lastTotalAssets` does not depend on the user-specified `assetsToWithdraw` amount and is instead set to the value returned by `TARGET_VAULT.totalAssets`:

```
uint256 assetsWithdrawn = _withdrawFromProtocol(assetsToWithdraw, assetReceiver);
// ...

lastTotalAssets = totalAssets();
```

It should also be noted that in both cases there are no checks ensuring that the actual change in `TARGET_VAULT.totalAssets` corresponds to the amount deposited or withdrawn.

For example, when using fee-on-transfer tokens, when an entry fee is charged by the external protocol, or in the case of rounding, the amount provided by the user may not match the actual amount deposited into `TARGET_VAULT.totalAssets`. The same applies to withdrawals.

Additionally, the `Deposited` and `Withdrawn` events may mislead logging and monitoring systems, as they record the user-specified values `assetsToDeposit` and `assetsWithdrawn`, respectively, rather than the actual amounts deposited into or withdrawn from the external protocol.

Recommendation

We recommend verifying that the user-specified deposit or withdrawal amount corresponds to the actual change in `totalAssets` on the external protocol after the deposit or withdrawal is executed, and then using this verified value when updating `lastTotalAssets`.

Update

Client's response

Partially fixed. The remaining issues are planned to be handled operationally by avoiding integration with fee-on-transfer tokens and vaults that apply fee-on-deposit or fee-on-withdraw mechanisms.

Fixed at 3dbef1424817cba2877f831f3a40809e07334992.

W-07	Insufficient validations in `RewardDistributor`, `ERC4626Adapter`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
ERC4626Adapter.sol	contract `ERC4626Adapter` > `constructor`	`88`
RewardDistributor.sol	contract `RewardDistributor` > `constructor`	`154`
RewardDistributor.sol	contract `RewardDistributor` > `constructor`	`128`

Description

At the specified locations, insufficient validation is performed.

In the `constructor` of the contract `ERC4626Adapter`, the addresses of the contracts `TARGET_VAULT` and `ASSET` are set. However, no validation is performed to ensure the compatibility between `TARGET_VAULT.asset` and `ASSET`. As a result, it is possible to mistakenly set an arbitrary `TARGET_VAULT`, which would not cause an error during deployment, but would later lead to incorrect contract behavior due to a mismatch between `TARGET_VAULT.asset` and `ASSET`.

In the `constructor` of the contract `RewardDistributor`, the list of recipients is initialized. However, it is possible to include the address of the `RewardDistributor` contract itself in this list. As a result, after each reward distribution, a portion of the rewards would be transferred back to the contract balance.

It is also possible to provide the arrays `recipients_` and `basisPoints_` with an unbounded size, which will inevitably lead to a revert if the total exceeds `MAX_BASIS_POINTS`. However, this will occur only after the entire `for` loop has been executed, resulting in a misleading `InvalidBasisPointsSum` error.

Additionally, a deployment error could permanently lock the contract, as there is no `admin_ != address(0)` check in the constructor.

Recommendation

We recommend adding validation to prevent the address of the contract itself from being included in the recipient list, as well as validating the sizes of the `recipients_` and `basisPoints_` arrays before entering the `for` loop to improve gas efficiency and provide clearer error reporting. Additionally, we recommend adding asset compatibility checks when setting the `TARGET_VAULT` address.

Update

Client's response

Partially fixed at d6f7a6a2381da706b17c7f2164e297bf0e6fdf1d.

3.4 INFO

I-01	Role misalignment in reward distribution in `RewardDistributor`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
RewardDistributor.sol	contract `RewardDistributor` > function `replaceRecipient`	`195`

Description

In the function `replaceRecipient` of the contract `RewardDistributor`, the reward recipient addresses are updated while the `basisPoints` values remain unchanged.

Access to `replaceRecipient` is limited to the `RECIPIENTS_MANAGER_ROLE`, while the ability to distribute rewards belongs to the `MANAGER_ROLE`. Because these roles are separated, the account responsible for updating recipients may not know when a reward distribution is about to occur, and the account responsible for distributing rewards may not know whether the recipient list is up to date.

Moreover, in some cases, `RECIPIENTS_MANAGER_ROLE` may need to change the `basisPoints` values themselves, for example, if the initial `recipients_` array provided in the constructor was incorrect. For instance, swapping allocations between `Alice` (20%) and `Bob` (15%) to `Bob` (20%) and `Alice` (15%) requires two successive calls to `replaceRecipient` using a temporary intermediate address.

If `MANAGER_ROLE` performs a reward distribution during this intermediate state, part of the funds may be sent to an incorrect address.

Recommendation

We recommend implementing a more coordinated interaction model between the roles `RECIPIENTS_MANAGER_ROLE` and `MANAGER_ROLE` to prevent their parallel operation at the

same time. For example, this could include introducing flags that block reward distribution until the recipient update process is finished.

I-02	Contract descriptions do not match the code in `EmergencyVault.sol`, `RewardDistributor.sol`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
EmergencyVault.sol	-	24
RewardDistributor.sol	-	17

Description

At the specified locations, the contract behavior is described.

In the documentation for `EmergencyVault`, it is stated that calling the function `emergencyWithdraw` also pauses the vault. However, in the protocol, the pause can only be triggered explicitly by calling the function `pause` with the role `PAUSER_ROLE`.

```
 *      Emergency Flow:
 *      1. Admin calls emergencyWithdraw() as many times as needed to recover funds from
protocol
 *         - First call snapshots emergencyTotalAssets and pauses vault
```

In the documentation of the contract `RewardDistributor`, it is stated that recipient configuration is immutable after deployment. This is only partially correct, as recipient addresses can be modified via the function `replaceRecipient`.

```
* Recipient configuration is immutable after deployment.
```

Recommendation

We recommend aligning the contract documentation with the actual implemented behavior of the contracts.

Update

Client's response

Fixed at f4c398f877aca5fd0308b3a531efd560ecbd27b2.

I-03	No sweep of excessive tokens in `ERC4626Adapter`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
ERC4626Adapter.sol	contract `ERC4626Adapter`	`25`

Description

The contract `ERC4626Adapter` lacks a function for withdrawing tokens that might have been mistakenly sent to the contract. As a result, any user can send tokens to the `ERC4626Adapter` contract, causing these tokens to become permanently locked within it.

Recommendation

We recommend adding a `sweep` function to withdraw tokens that were sent by mistake.

Update

Client's response

Fixed at d81dbdf61ec6002d87ad0fa51ec8ec2631770346.

I-04	Lack of Slippage Protection in `Vault`
Severity	**INFO**
Status	• NO ISSUE

Location

File	Location	Line
Vault.sol	contract `Vault` > function `deposit`	`211`

Description

The `deposit` function of `Vault` — consistent with the ERC-4626 interface — does not accept a `minShares` argument. If a front-runner changes the price in `TARGET_VAULT` before a user's large deposit (e.g., through rebalancing or `donation`), `previewDeposit` may show one share amount while the actual `deposit` call mints significantly fewer; without a `minShares` parameter, the user cannot lock in a minimum acceptable result and is forced to accept any rate, including a knowingly unfavorable one.

Recommendation

We recommend adding a `depositWithSlippage` function to allow users to enforce minimum share constraints during deposits.

Update

Client's response

The vault is designed to work with ERC-4626 vaults, which means that new deposits do not change the share price. If any rewards are pending, the vault recalculates the share price upon deposit so that the user does not lose any funds.

4. APPENDIX

4.1 SECURITY ASSESSMENT METHODOLOGY

Oxorio's smart contract security audit methodology is designed to ensure the security, reliability, and compliance of smart contracts throughout their development lifecycle. Our process integrates the Smart Contract Security Verification Standard (SCSVS) with our advanced techniques to address complex security challenges. For a detailed look at our approach, please refer to the full version of our methodology. Here is a concise overview of our auditing process:

1. Project Architecture Review

All necessary information about the smart contract is gathered, including its intended functionality and dependencies. This stage sets the foundation by reviewing documentation, business logic, and initial code analysis.

2. Vulnerability Assessment

This phase involves a deep dive into the smart contract's code to identify security vulnerabilities. Rigorous testing and review processes are applied to ensure robustness against potential attacks.

This stage is focused on identifying specific vulnerabilities within the smart contract code. It involves scanning and testing the code for known security weaknesses and patterns that could potentially be exploited by malicious actors.

3. Security Model Evaluation

The smart contract’s architecture is assessed to ensure it aligns with security best practices and does not introduce potential vulnerabilities. This includes reviewing how the contract integrates with external systems, its compliance with security best practices, and whether the overall design supports a secure operational environment.

This phase involves a analysis of the project's documentation, the consistency of business logic as documented versus implemented in the code, and any assumptions made during the design and development phases. It assesses if the contract's architectural design adequately addresses potential threats and integrates necessary security controls.

4. Cross-Verification by Multiple Auditors

Typically, the project is assessed by multiple auditors to ensure a diverse range of insights and thorough coverage. Findings from individual auditors are cross-checked to verify accuracy and completeness.

5. Report Consolidation

Findings from all auditors are consolidated into a single, comprehensive audit report. This report outlines potential vulnerabilities, areas for improvement, and an overall assessment of the smart contract's security posture.

6. Reaudit of Revised Submissions

Post-review modifications made by the client are reassessed to ensure that all previously identified issues have been adequately addressed. This stage helps validate the effectiveness of the fixes applied.

7. Final Audit Report Publication

The final version of the audit report is delivered to the client and published on Oxorio's official website. This report includes detailed findings, recommendations for improvement, and an executive summary of the smart contract's security status.

4.2 CODEBASE QUALITY ASSESSMENT REFERENCE

The tables below describe the codebase quality assessment categories and rating criteria used in this report.

Category	Description
Access Control	Evaluates the effectiveness of mechanisms controlling access to ensure only authorized entities can execute specific actions, critical for maintaining system integrity and preventing unauthorized use.
Arithmetic	Focuses on the correct implementation of arithmetic operations to prevent vulnerabilities like overflows and underflows, ensuring that mathematical operations are both logically and semantically accurate.
Complexity	Assesses code organization and function clarity to confirm that functions and modules are organized for ease of understanding and maintenance, thereby reducing unnecessary complexity and enhancing readability.
Data Validation	Assesses the robustness of input validation to prevent common vulnerabilities like overflow, invalid addresses, and other malicious input exploits.
Decentralization	Reviews the implementation of decentralized governance structures to mitigate insider threats and ensure effective risk management during contract upgrades.
Documentation	Reviews the comprehensiveness and clarity of code documentation to ensure that it provides adequate guidance for understanding, maintaining, and securely operating the codebase.
External Dependencies	Evaluates the extent to which the codebase depends on external protocols, oracles, or services. It identifies risks posed by these dependencies, such as compromised data integrity, cascading failures, or reliance on centralized entities. The assessment checks if these external integrations have appropriate fallback mechanisms or redundancy to mitigate risks and protect the protocol's functionality.
Error Handling	Reviews the methods used to handle exceptions and errors, ensuring that failures are managed gracefully and securely.
Logging and Monitoring	Evaluates the use of event auditing and logging to ensure effective tracking of critical system interactions and detect potential anomalies.
Low-Level Calls	Reviews the use of low-level constructs like inline assembly, raw `call` or `delegatecall`, ensuring they are justified, carefully implemented, and do not compromise contract security.

Category	Description
Testing and Verification	Reviews the implementation of unit tests and integration tests to verify that codebase has comprehensive test coverage and reliable mechanisms to catch potential issues.

4.2.1 Rating Criteria

Rating	Description
Excellent	The system is flawless and surpasses standard industry best practices.
Good	Only minor issues were detected; overall, the system adheres to established best practices.
Fair	Issues were identified that could potentially compromise system integrity.
Poor	Numerous issues were identified that compromise system integrity.
Absent	A critical component is absent, severely compromising system safety.
Not Applicable	This category does not apply to the current evaluation.

4.3 FINDINGS CLASSIFICATION REFERENCE

4.3.1 Severity Level Reference

The following severity levels were assigned to the issues described in the report:

Title	Description
CRITICAL	Issues that pose immediate and significant risks, potentially leading to asset theft, inaccessible funds, unauthorized transactions, or other substantial financial losses. These vulnerabilities represent serious flaws that could be exploited to compromise or control the entire contract. They require immediate attention and remediation to secure the system and prevent further exploitation.
MAJOR	Issues that could cause a significant failure in the contract's functionality, potentially necessitating manual intervention to modify or replace the contract. These vulnerabilities may result in data corruption, malfunctioning logic, or prolonged downtime, requiring substantial operational changes to restore normal performance. While these issues do not immediately lead to financial losses, they compromise the reliability and security of the contract, demanding prioritized attention and remediation.
WARNING	Issues that might disrupt the contract's intended logic, affecting its correct functioning or making it vulnerable to Denial of Service (DDoS) attacks. These problems may result in the unintended triggering of conditions, edge cases, or interactions that could degrade the user experience or impede specific operations. While they do not pose immediate critical risks, they could impact contract reliability and require attention to prevent future vulnerabilities or disruptions.
INFO	Issues that do not impact the security of the project but are reported to the client's team for improvement. They include recommendations related to code quality, gas optimization, and other minor adjustments that could enhance the project's overall performance and maintainability.

4.3.2 Status Level Reference

Based on the feedback received from the client's team regarding the list of findings discovered by the contractor, the following statuses were assigned to the findings:

Title	Description
NEW	Waiting for the project team's feedback.

Title	Description
FIXED	Recommended fixes have been applied to the project code and the identified issue no longer affects the project's security.
ACKNOWLEDGED	The project team is aware of this finding and acknowledges the associated risks. This finding may affect the overall security of the project; however, based on the risk assessment, the team will decide whether to address it or leave it unchanged.
NO ISSUE	Finding does not affect the overall security of the project and does not violate the logic of its work.

4.4 ABOUT OXORIO

OXORIO is a blockchain security firm that specializes in smart contracts, zk-SNARK solutions, and security consulting. With a decade of blockchain development and five years in smart contract auditing, our expert team delivers premier security services for projects at any stage of maturity and development.

Since 2021, we've conducted key security audits for notable DeFi projects like Lido, 1Inch, Rarible, and deBridge, prioritizing excellence and long-term client relationships. Our co-founders, recognized by the Ethereum and Web3 Foundations, lead our continuous research to address new threats in the blockchain industry. Committed to the industry's trust and advancement, we contribute significantly to security standards and practices through our research and education work.

Our contacts:

- oxor.io
- ping@oxor.io
- Github
- Linkedin
- Twitter

THANK YOU FOR CHOOSING