LTV protocol

LTV VAULT CRAFT SMART CONTRACTS SECURITY AUDIT REPORT

OXORIO

NOVEMBER 4, 2025

1 EXECUTIVE SUMMARY

OXORIO

1.1 EXECUTIVE SUMMARY

This document presents the smart contracts security audit conducted by Oxorio for LTV Protocol's LTV Vault Craft.

The LTV Protocol is a revolutionary Curatorless Leveraged Tokenized Vault that maintains a constant target Loan-To-Value (LTV) ratio without requiring a central curator. Built on the foundation of two interconnected EIP-4626 vaults, it enables users to deposit and withdraw funds while receiving tokenized shares representing their leveraged positions. The protocol's core innovation lies in its auction-based stimulus system that incentivizes users to participate in rebalancing actions through rewards or fees. This mechanism ensures alignment with the target LTV while providing basic MEV protection against frontrunning.

Vault Craft is a collection of Solidity helper contracts that provide safe, slippage-protected operations for ERC-4626 tokenized vaults and collateral vaults.

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 17 smart contracts, encompassing 612 lines of code. The codebase was thoroughly examined, with the audit team collaborating closely with LTV Protocol 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 LTV Protocol to address all concerns identified within the audit's scope. Each issue has been either resolved or formally acknowledged by LTV Protocol, contributing to the robustness of the project.

As a result, following a comprehensive review, our auditors have verified that the LTV Vault Craft, as of audited commit 2a784193bb8bdc6714d161d68f3953ff2f07097c, 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.

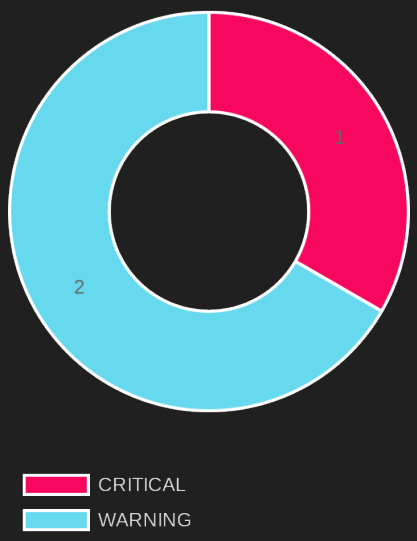
1.2 SUMMARY OF FINDINGS

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.

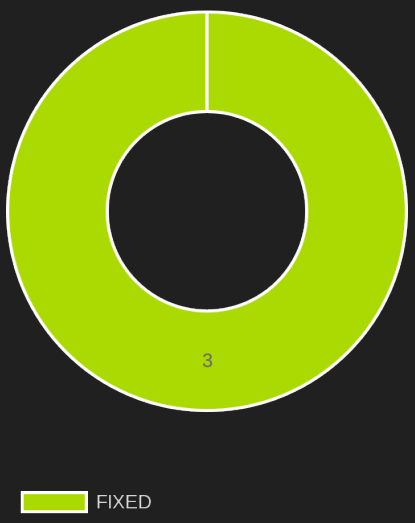
Detailed technical information on the audit findings, along with our recommendations for addressing them, is provided in the Findings Report section for further reference.

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

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



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	LTV Protocol
Project name	LTV Vault Craft
Category	Leveraged Tokenized Vault
Website	https://ltv.finance
Repository	https://github.com/ltvprotocol/vault_craft/
Documentation	LTV_Curatorless_Leveraged_Tokenized_Vault_with_a_Constant_Target_Loan-To-Value_Ratio.pdf
Initial Commit	6f82ee6d4a3fd75edbb385229e6e538d41849461
Final Commit	2a784193bb8bdc6714d161d68f3953ff2f07097c
Platform	L1
Network	Ethereum
Languages	Solidity
Lead Auditor	Alexander Mazaletskiy - am@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
October 27, 2025	Client requested audit.
October 28, 2025	The audit team initiated work on the project.
October 29, 2025	Submission of the preliminary audit report #1.
October 29, 2025	Submission of the preliminary audit report #2.
November 4, 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/CommonFlashLoanHelper.sol	52	11	1	**40**	10%
2	src/FlashLoanMintHelperWstethAndWeth.sol	78	21	8	**49**	12%
3	src/FlashLoanRedeemHelperWstethAndWeth.sol	81	18	8	**55**	9%
4	src/interfaces/balancer/IBalancerVault.sol	14	2	1	**11**	0%
5	src/interfaces/balancer/IFlashLoanRecipient.sol	13	2	1	**10**	0%
6	src/interfaces/ICurve.sol	9	2	2	**5**	0%
7	src/interfaces/IERC4626.sol	124	28	60	**36**	0%
8	src/interfaces/IERC4626Collateral.sol	34	14	1	**19**	0%
9	src/interfaces/IFlashLoanHelperErrors.sol	10	1	1	**8**	0%
10	src/interfaces/IFlashLoanHelperEvents.sol	7	1	1	**5**	0%
11	src/interfaces/IFlashLoanMintHelper.sol	10	2	1	**7**	0%
12	src/interfaces/ILowLevelVault.sol	14	2	1	**11**	0%
13	src/interfaces/tokens/IstEth.sol	12	2	1	**9**	0%
14	src/interfaces/tokens/IWETH.sol	9	2	1	**6**	0%
15	src/interfaces/tokens/IwstEth.sol	9	2	1	**6**	0%
16	src/Safe4626CollateralHelper.sol	71	10	1	**60**	7%
17	src/Safe4626Helper.sol	65	10	1	**54**	7%
	Total	**612**	**130**	**91**	**391**	**6%**

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

Vault Craft is a collection of Solidity helper contracts that provide safe, slippage-protected operations for ERC-4626 tokenized vaults and collateral vaults.

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	Access control mechanisms in helper contracts were reviewed. A **critical issue C-1** was identified: missing `approve` for token transfers to the `vault` from `Safe4626CollateralHelper` and `Safe4626Helper`. The issue was resolved.	**Excellent**
Arithmetic	Arithmetic operations in the contracts were correct, with overflow/underflow prevented via SafeMath.	**Excellent**
Complexity	The logic in helper contracts was moderately complex but readable.	**Excellent**
Data Validation	Input data in deposit and mint functions was properly validated (zero checks, sufficient balance).	**Excellent**
Decentralization	The contracts contained no centralized control elements or owner privileges with critical impact.	**Not Applicable**
Documentation	NatSpec comments were present on key functions. Full external interface documentation was not required for helper contracts.	**Not Applicable**
External Dependencies	Well-audited OpenZeppelin libraries (ERC20, SafeERC20) were used.	**Excellent**
Error Handling	Errors were handled via `require` and `revert` with informative messages.	**Excellent**
Logging and Monitoring	Events for minting and transfers were emitted correctly. Additional monitoring was not required.	**Excellent**
Low-Level Calls	Low-level calls were absent. All interactions used safe methods from SafeERC20. No issues were present.	**Excellent**

Category	Assessment	Result
Testing and Verification	Tests covered main scenarios using mocked `vault` contracts. The critical `approve` bug was not caught due to mocks, but the code was retested thoroughly after the fix.	**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/CommonFlashLoanHelper.sol	**1**	0	0	1	0
src/FlashLoanMintHelperWstethAndWeth.sol	**1**	0	0	1	0
src/Safe4626CollateralHelper.sol	**1**	1	0	0	0
src/Safe4626Helper.sol	**1**	1	0	0	0

2.8 CONCLUSION

A comprehensive audit was conducted on 17 smart contracts, initially revealing 1 critical and 0 major issues, along with numerous warnings notes.

The audit identified a **critical vulnerability** in token transfer logic within helper contracts (`Safe4626CollateralHelper` and `Safe4626Helper`), where missing `approve` calls prevented the `vault` from pulling assets, despite direct `safeTransferFrom` usage — a flaw masked by mocked vaults in testing. Additionally, **warning-level issues** included an unnecessary `fallback` function introducing phishing risks and inefficient `wstETH` wrapping via redundant computations instead of direct Lido submission. No major or informational issues were found beyond these. All identified problems were acknowledged, with the critical issue resolved and warnings recommended for optimization and risk reduction. Overall, the codebase demonstrated strong security practices in access control, arithmetic safety, error handling, and dependency management, achieving **excellent** post-remediation status across applicable categories.

Following our initial audit, LTV Protocol worked closely with our team to address the identified issues.

The proposed changes focused on adding missing `approve` calls in `Safe4626CollateralHelper` and `Safe4626Helper` to enable secure token transfers to the `vault`, removing the unnecessary `fallback` function in `CommonFlashLoanHelper` to eliminate phishing risks, and optimizing `wstETH` wrapping in `FlashLoanMintHelperWstethAndWeth` by leveraging Lido's direct submission mechanism to avoid redundant and potentially inaccurate computations.

All identified issues have been successfully addressed or formally acknowledged.

As a result, the project has passed our audit. Our auditors have verified that the LTV Vault Craft, as of audited commit 2a784193bb8bdc6714d161d68f3953ff2f07097c, 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.

3 AUDIT OBJECTIVES

The audit focused on identifying vulnerabilities, ensuring compliance with standards, and verifying the correctness of the implementation for the LTV Protocol's **Safe4626 helper contracts** (`Safe4626CollateralHelper`, `Safe4626Helper`, `CommonFlashLoanHelper`, and `FlashLoanMintHelperWstethAndWeth`). The key areas of focus included:

3.1 TOKEN HANDLING AND APPROVALS

- Review of ERC20 interactions in deposit and mint workflows, with emphasis on `safeTransferFrom` usage and the necessity of prior `approve` calls when the helper acts as an intermediary between user and `vault`.
- Detection of approval bypass risks where the `vault` expected to pull assets from the helper, but no approval was granted — a flaw masked by mocked vault behavior in testing.

3.2 INTEGRATION WITH EXTERNAL PROTOCOLS

- Validation of interactions with ERC4626-compliant vaults, ensuring correct asset flow in `deposit` / `mint` operations via helper contracts.
- Assessment of Lido `wstETH` wrapping logic in `FlashLoanMintHelperWstethAndWeth`, identifying inefficient on-chain computation paths versus direct payable submission to the `wstETH` contract.

3.3 ACCESS CONTROL AND CALL SAFETY

- Examination of function call patterns in helper contracts, including direct vault interactions and flash loan callbacks.
- Identification of unnecessary `fallback` functions that increased contract attack surface and introduced phishing risks via unintended call forwarding.

3.4 REENTRANCY AND LOW-LEVEL SAFETY

- Analysis of state changes during helper operations, confirming absence of reentrancy vectors despite external calls to `vault` and token contracts.
- Verification that all external interactions used `SafeERC20` and avoided raw `call` or `delegatecall`.

3.5 DATA FLOW AND VALIDATION

- Checks on input validation in helper functions (e.g., zero-amount guards, address sanity).
- Review of flash loan payload handling and return value enforcement in Balancer-style and Morpho-style callbacks.

3.6 TESTING AND VERIFICATION GAPS

- Evaluation of test coverage, particularly the use of mocked `ERC4626` vaults that bypassed real `safeTransferFrom` enforcement, leading to undetected approval logic flaws in production-like conditions.

3.7 CODE CLARITY AND MAINTAINABILITY

- Assessment of inline documentation, function naming, and modularity in helper contracts.
- Identification of optimization opportunities that also reduced complexity and potential error surfaces.

3.8 OUT OF SCOPE

The following areas were explicitly excluded from the audit:

- Core vault logic, leverage mechanics, or rebalancing modules outside the helper contracts.
- Full protocol-level economic risks, liquidation paths, or LTV parameter safety.
- Gas optimization beyond security-critical inefficiencies.
- Frontend, off-chain automation, or monitoring systems.

4. FINDINGS REPORT

OXORIO

4.1 CRITICAL

C-01	Missing `approve` for token transfers to the `vault` in `Safe4626CollateralHelper`, `Safe4626Helper`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
Safe4626CollateralHelper.sol	contract `Safe4626CollateralHelper` > function `safeDepositCollateral`	16
Safe4626CollateralHelper.sol	contract `Safe4626CollateralHelper` > function `safeMintCollateral`	30
Safe4626Helper.sol	contract `Safe4626Helper` > function `safeDeposit`	16
Safe4626Helper.sol	contract `Safe4626Helper` > function `safeMint`	30

Description

In the mentioned locations, functions of the `vault` are called, where token transfers are performed:

```
<token>.safeTransferFrom(msg.sender, address(this), assets);
```

However, within the `vault`, the `msg.sender` will be the helper contract (`Safe4626CollateralHelper` or `Safe4626Helper`). The helper does not receive tokens from the user and does not provide `approve` for the `vault` to transfer tokens.

It should be noted that the helper tests use mocked `vault` contracts, which do not perform real `safeTransferFrom` calls:

- #L85
- #L108

Recommendation

We recommend adding token transfer and `approve` functionality to the helper contracts before performing deposits into the `vault`.

Update

Client's response

Fixed at 2a784193bb8bdc6714d161d68f3953ff2f07097c.

4.2 MAJOR

No major issues found.

4.3 WARNING

W-01	Unnecessary `fallback` function in `CommonFlashLoanHelper`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
CommonFlashLoanHelper.sol	contract `CommonFlashLoanHelper` > `receive/fallback function`	51

Description

In the contract `CommonFlashLoanHelper`, the `fallback` function is unnecessary and may introduce phishing risks, as it allows calls to non-existing functions on the contract.

Recommendation

We recommend removing the `fallback` function.

Update

Client's response

Fixed at 2a784193bb8bdc6714d161d68f3953ff2f07097c.

W-02	Excessive computations during conversion to `wstETH` in `FlashLoanMintHelperWstethAndWeth`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
FlashLoanMintHelperWstethAndWeth....	contract `FlashLoanMintHelperWstethAndWeth` > function `_wrapShares`	71

Description

In the function `_wrapShares` of contract `FlashLoanMintHelperWstethAndWeth`, shares of `stETH` are converted into `wstETH`.

However, this conversion can be performed directly in Lido by sending `ETH` to the `wstETH` contract:

```
/**

* @notice Shortcut to stake ETH and auto-wrap returned stETH
*/
receive() external payable {
    uint256 shares = stETH.submit{value: msg.value}(address(0));
    _mint(msg.sender, shares);
}
```

Recommendation

We recommend sending the flash loan `ETH` directly to the `wstETH` contract for conversion, avoiding extra and potentially inaccurate computations.

Update

Client's response

Fixed at 2a784193bb8bdc6714d161d68f3953ff2f07097c.

4.4 INFO

No info issues found.

5 APPENDIX

OXORIO

5.1 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.

5.2 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.

5.2.1 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.

5.3 FINDINGS CLASSIFICATION REFERENCE

5.3.1 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.

5.3.2 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.

5.4 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