TRIUMPH TOKEN SMART CONTRACTS SECURITY AUDIT REPORT

OXORIO

MARCH 10, 2026

1 EXECUTIVE SUMMARY

OXORIO

1.1 EXECUTIVE SUMMARY

This document presents the smart contracts security audit conducted by Oxorio for Triumph Games' Token.

Triumph Games is a company engaged in the development and publication of digital games with blockchain integration. The studio focuses on building an ecosystem in which in-game assets and player progress have characteristics of digital ownership and can be utilized across multiple products within the company's portfolio. Triumph Games aims to establish a platform that connects several games and related digital services based on distributed ledger technology.

Triumph Token (TRI) is the native utility token used within the Triumph Games ecosystem. It functions as an internal medium of exchange and interaction across the company's gaming and related products, including asset transfers, staking, in-game operations, and other platform functionalities. TRI is designed to support interoperability and operational functionality between ecosystem components and serves as a core element of the economic model of the projects developed by Triumph Games.

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 1 smart contracts, encompassing 271 lines of code. The codebase was thoroughly examined, with the audit team collaborating closely with Triumph Games 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 Triumph Games to address all concerns identified within the audit's scope. Each issue has been either resolved or formally acknowledged by Triumph Games, contributing to the robustness of the project.

As a result, following a comprehensive review, our auditors have verified that the Triumph Token, as of audited commit `74105cc851fcea7e6e93852ae7cdb88a0510183f`, 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.

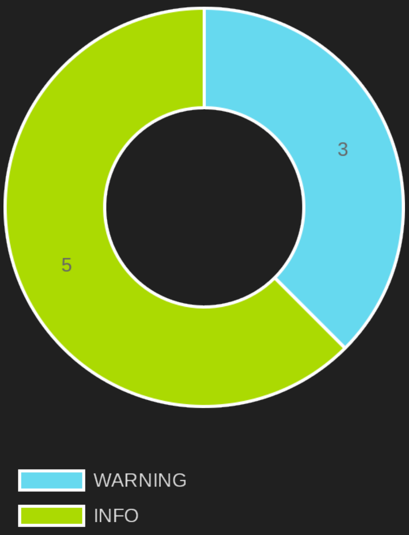
1.2 SUMMARY OF FINDINGS

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.

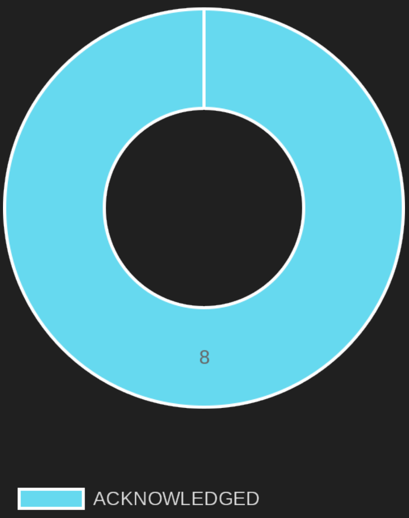
Detailed technical information on the audit findings, along with our recommendations for addressing them, is provided in the Findings Report section for further reference.

All identified issues have been addressed, with Triumph Games 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	**3**	0	0	3	0
INFO	**5**	0	0	5	0
TOTAL	**8**	**0**	**0**	**8**	**0**



Issue distribution by severity



Issue distribution by status

2 AUDIT OVERVIEW

OXORIO

CONTENTS

1. EXECUTIVE SUMMARY 2

1.1. EXECUTIVE SUMMARY 3

1.2. SUMMARY OF FINDINGS 4

2. AUDIT OVERVIEW 5

2.1. DISCLAIMER 8

2.2. PROJECT BRIEF 9

2.3. PROJECT TIMELINE 10

2.4. AUDITED FILES 11

2.5. PROJECT OVERVIEW 12

2.6. CODEBASE QUALITY ASSESSMENT 13

2.7. FINDINGS BREAKDOWN BY FILE 15

2.8. CONCLUSION 16

3. FINDINGS REPORT 17

3.1. CRITICAL 18

3.2. MAJOR 19

3.3. WARNING 20

W-01 Underflow in case of large fee values in DropERC20 20

W-02 Ability to mint more tokens than maxTotalSupply in DropERC20 22

W-03 Validation of _defaultAdmin for zero address in DropERC20 24

3.4. INFO 25

I-01 Replay attack in Drop 25

I-02 Zero address check in PlatformFee 26

I-03 Inconsistency of MAX_BPS in DropERC20, PlatformFee 27

I-04 Incorrect error description for small totalPrice in DropERC20 28

I-05 Transfers of tokens can be blocked for legitimate claimers in DropERC20 29

4. APPENDIX 30
4.1. SECURITY ASSESSMENT METHODOLOGY 31
4.2. CODEBASE QUALITY ASSESSMENT REFERENCE 33
Rating Criteria 34
4.3. FINDINGS CLASSIFICATION REFERENCE 35
Severity Level Reference 35
Status Level Reference 35
4.4. ABOUT OXORIO 37

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	Triumph Games
Project name	Triumph Token
Category	Token
Website	https://www.triumphgames.io/
Documentation	https://triumph-games.gitbook.io/litepaper/
Repository	https://github.com/thirdweb-dev/contracts/
Initial Commit	`74105cc851fcea7e6e93852ae7cdb88a0510183f`
Final Commit	`74105cc851fcea7e6e93852ae7cdb88a0510183f`
Platform	L2
Network	Arbitrum
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
January 7, 2026	Client engaged Oxorio requesting an audit.
February 10, 2026	The audit team initiated work on the project.
February 11, 2026	Submission of the initial audit report.
March 10, 2026	Submission of the final audit report with all findings acknowledged by the client.

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	contracts/prebuilts/drop/DropERC20.sol	271	51	56	**164**	11%
	Total	**271**	**51**	**56**	**164**	**11%**

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

The project's token is built using the Thirdweb framework, leveraging its standardized smart contract architecture and modular extensions for token distribution and management.

`DropERC20` is a smart contract from the Thirdweb library designed for controlled distribution of ERC-20 tokens through configurable claim conditions. It extends standard ERC-20 functionality with a drop mechanism, allowing tokens to be issued in batches across different phases with customizable conditions for each phase, such as:

- limiting the total number of tokens available for claim,
- per-wallet token limits,
- token price and payment currency,
- an allowlist (Merkle list) of addresses permitted to claim tokens under specific conditions.

The contract implements a claim mechanism - users can call the **claim** function when the current claim condition is met. The claim conditions are structured so that only one distribution phase is active at any given time (based on start time and order), and the conditions are stored in structures including `maxClaimableSupply`, `supplyClaimed`, `quantityLimitPerWallet`, `pricePerToken`, and other parameters.

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 contract implements role-based access control for administrative and transfer functions. Proper checks exist for `_defaultAdmin` and `transferRole`, though zero-address validation should be added to prevent misconfiguration.	**Excellent**
Arithmetic	Basic arithmetic operations are used with care. Underflow and overflow risks are minimal, but independent fee calculations could trigger underflow if not validated.	**Good**
Complexity	The code structure is modular and well-organized, using inheritance and extensions to maintain clarity. Functions have clear responsibilities, and the overall architecture is straightforward.	**Excellent**
Data Validation	Some input parameters, such as `_defaultAdmin`, `_pricePerToken`, and `platformFeeRecipient`, lack full validation, which could lead to unexpected behavior if misconfigured.	**Fair**
Decentralization	The contract relies on role-based administration and centralized fee management. Decentralization is not a primary design goal.	**Not Applicable**
Documentation	Functions, parameters, and error messages are adequately documented, and the code is clear and readable for auditors and developers.	**Excellent**
External Dependencies	The contract uses well-audited Thirdweb libraries and extensions. External dependencies are minimal and clearly defined.	**Excellent**

Category	Assessment	Result
Error Handling	The contract uses `require` statements for input and state validation. Error messages are mostly informative but could be improved for clarity in some cases.	**Good**
Logging and Monitoring	Events are emitted for critical actions such as token claims and role changes, enabling monitoring of key operations on-chain.	**Excellent**
Low-Level Calls	The contract does not perform unsafe low-level calls.	**Not Applicable**
Testing and Verification	The codebase includes comprehensive unit tests. Core functionalities, such as claiming and fee handling, are well-tested.	**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
contracts/prebuilts/drop/DropERC20.sol	**6**	0	0	3	3
contracts/extension/PlatformFee.sol	**2**	0	0	0	2
contracts/extension/Drop.sol	**1**	0	0	0	1

2.8 CONCLUSION

A comprehensive audit was conducted on 1 smart contract, initially revealing 0 critical and 0 major issues, along with numerous warnings and informational notes. The audit highlighted various attack vectors and potential vulnerabilities, with significant findings related to fee calculations, replay attacks, role management, zero-address validations, and error message clarity.

Following our initial audit, Triumph Games worked closely with our team to review the identified findings. The proposed changes focused on reinforcing role management integrity, ensuring accurate administrative permission enforcement, improving validation of input parameters, and enhancing clarity in error reporting and code documentation to strengthen the overall security and reliability of the smart contract. Through multiple rounds of interaction, all identified issues have been successfully addressed or formally acknowledged.

As a result, the project has passed our audit. Our auditors have verified that Triumph Token, as of audited commit `74105cc851fcea7e6e93852ae7cdb88a0510183f`, 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.

To enhance the project's security and readiness for production, we recommend conducting additional security reviews after each significant contract change. Additionally, increasing test coverage is advised to ensure thorough validation and reliability of the codebase.

3 FINDINGS REPORT

OXORIO

3.1 CRITICAL

No issues found

3.2 MAJOR

No issues found

3.3 WARNING

W-01	Underflow in case of large fee values in `DropERC20`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
DropERC20.sol	contract `DropERC20` > function `_collectPriceOnClaim`	163

Description

In the function `_collectPriceOnClaim` of contract `DropERC20`, fees are calculated relative to `totalPrice`:

```
uint256 platformFeesTw = (totalPrice * DEFAULT_FEE_BPS) / MAX_BPS;
uint256 platformFees = (totalPrice * platformFeeBps) / MAX_BPS;
```

However, both values are calculated independently, and their sum can exceed the original value of `totalPrice`. For example, if `platformFeeBps` equals `MAX_BPS`, then `platformFees` will equal the entire `totalPrice`.

This can cause an underflow revert in the transfer calculation, as `totalPrice` will be less than `platformFees + platformFeesTw`:

```
CurrencyTransferLib.transferCurrency(
    _currency,
    _msgSender(),
    saleRecipient,
    totalPrice - platformFees - platformFeesTw
);
```

Recommendation

We recommend checking the calculated fee amounts and returning a clear error message to the user in case of an issue.

W-02	Ability to mint more tokens than `maxTotalSupply` in `DropERC20`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
DropERC20.sol	contract `DropERC20` > function `_collectPriceOnClaim`	174

Description

In the function `_collectPriceOnClaim` of contract `DropERC20`, `_currency` tokens are transferred from the user's address to the protocol:

```
CurrencyTransferLib.transferCurrency(_currency, _msgSender(), DEFAULT_FEE_RECIPIENT,
platformFeesTw);
CurrencyTransferLib.transferCurrency(_currency, _msgSender(), platformFeeRecipient,
platformFees);
CurrencyTransferLib.transferCurrency(
    _currency,
    _msgSender(),
    saleRecipient,
    totalPrice - platformFees - platformFeesTw
);
```

However, neither the `_collectPriceOnClaim` function nor the calling `claim` function implements a reentrancy guard (e.g., `nonReentrant`).

Assume an attacker discovers a vulnerability in the `_currency` contract used for payment and is able to perform a reentrancy attack. In this case, by calling the `claim` function of contract `Drop`, and during execution of `CurrencyTransferLib.transferCurrency`, the attacker could reenter the `claim` function.

As a result, each time the attacker reenters the `claim` function, the check in `_beforeClaim` may be performed incorrectly, since `totalSupply` does not change between reentrant calls:

```
uint256 _maxTotalSupply = maxTotalSupply;
require(_maxTotalSupply == 0 || totalSupply() + _quantity <= _maxTotalSupply, "exceed max
total supply.");
```

The `totalSupply` value will only change after the reentrancy attack completes, when `_transferTokensOnClaim` is called to mint the tokens:

```
function claim(
    // ...
) public payable virtual override {
    _beforeClaim(_receiver, _quantity, _currency, _pricePerToken, _allowlistProof, _data);

    // ...

    // Function within which reentrancy occurs
    _collectPriceOnClaim(address(0), _quantity, _currency, _pricePerToken);

    // Final minting of tokens
    uint256 startTokenId = _transferTokensOnClaim(_receiver, _quantity);
```

As a result, an attacker may mint more tokens than the defined `maxTotalSupply`.

Recommendation

We recommend considering the addition of reentrancy protection mechanisms to prevent reentrancy attacks.

W-03	Validation of `_defaultAdmin` for zero address in `DropERC20`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
DropERC20.sol	contract `DropERC20` > function `initialize`	94

Description

In the function `initialize` of contract `DropERC20` `_defaultAdmin` is not validated to be non-zero. If a zero address is passed, the contract can be left without a usable admin, which locks all admin-gated functionality such as setting claim conditions, platform fee info and so on.

Recommendation

We recommend adding validation for `_defaultAdmin` to be non-zero address.

3.4 INFO

I-01	Replay attack in `Drop`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
Drop.sol	contract `Drop` > function `verifyClaim`	167

Description

In the function `verifyClaim` of contract `Drop`, the Merkle leaf is built only from `claimer`, `quantityLimitPerWallet`, `pricePerToken`, and `currency`. The leaf does not include any domain-separating data, such as `address(this)`, `block.chainid`, or a `campaign id`/ `condition id`. If the same merkleRoot is reused across campaigns, contracts, or networks, the same proof can be replayed to claim tokens in unintended way.

Recommendation

Include domain-separating fields in the leaf, such as `address(this)`, `block.chainid`, and `conditionId` or a campaign id.

I-02	Zero address check in `PlatformFee`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
PlatformFee.sol	contract `PlatformFee` > function `_setupFlatPlatformFeeInfo`	85

Description

In the function `_setupFlatPlatformFeeInfo` of contract `PlatformFee` the `platformFeeRecipient` variable is assigned without checking for `address(0)`. This can accidentally route fees to the zero address if misconfigured, leading to loss of funds or unexpected fee accounting.

Recommendation

We recommend adding `require(_platformFeeRecipient != address(0))` in `_setupFlatPlatformFeeInfo`, consistent with the checks in `_setupPlatformFeeInfo`.

I-03	Inconsistency of `MAX_BPS` in `DropERC20`, `PlatformFee`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
PlatformFee.sol	contract `PlatformFee` > function `_setupPlatformFeeInfo`	70
DropERC20.sol	contract `DropERC20` > function `_collectPriceOnClaim`	164

Description

In the mentioned locations, the maximum possible BPS value, equal to `10_000`, is used.

However, in the `DropERC20` contract it is defined as the constant `MAX_BPS`, while in the `PlatformFee` contract, a separate, independent value of `10_000` is used.

Thus, these values are not linked, which can lead to errors in case of refactoring or changing one of them.

Recommendation

We recommend linking these values under a single `MAX_BPS` constant to maintain code consistency.

I-04	Incorrect error description for small `totalPrice` in `DropERC20`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	**Location**	**Line**
DropERC20.sol	contract `DropERC20` > function `_collectPriceOnClaim`	161

Description

In the function `_collectPriceOnClaim` of contract `DropERC20`, `totalPrice` is calculated and subsequently checked:

```
uint256 totalPrice = (_quantityToClaim * _pricePerToken) / 1 ether;
require(totalPrice > 0, "quantity too low");
```

However, `totalPrice` can be `0` not only due to a small `quantity` but also due to a small `_pricePerToken`.

Recommendation

We recommend updating the error message, for example to "quantity or price too low", to avoid user misunderstanding.

I-05	Transfers of tokens can be blocked for legitimate claimers in `DropERC20`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
DropERC20.sol	contract `DropERC20` > function `_beforeTokenTransfer`	234

Description

In the function `_beforeTokenTransfer` of contract `DropERC20`, when transfer restrictions are active (`transferRole` revoked from `address(0)`), every non-mint/non-burn transfer requires the sender or receiver to hold `transferRole`:

```
if (!hasRole(transferRole, address(0)) && from != address(0) && to != address(0)) {
    require(hasRole(transferRole, from) || hasRole(transferRole, to), "transfers
restricted.");
}
```

The `claim` function mints tokens to `_receiver` via `_mint`, bypassing this check (`from == address(0)`). However, after receiving tokens, the claimer cannot transfer them without `transferRole`. The contract silently collects payment and distributes tokens that are immediately locked for the recipient.

Furthermore, if the admin revokes `transferRole` from `address(0)` after users have already claimed, all previously claimed tokens held by users without the `transferRole` become effectively frozen.

Recommendation

We recommend considering the addition of restrictions on the duration of the `transferRole` for already claimed tokens, or making the transfer lock mechanism more transparent for users to avoid misunderstanding when setting or revoking the `transferRole` for the `address(0)`.

4. APPENDIX

OXORIO

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

OXORIO