KALP

KALP NETWORK GINI VESTING SMART CONTRACTS SECURITY AUDIT REPORT

OXORIO

JANUARY 30, 2025

1 EXECUTIVE SUMMARY

OXORIO

1.1 EXECUTIVE SUMMARY

This document presents the smart contracts security audit conducted by Oxorio for Kalp Network's Gini Vesting Smart Contracts.

Kalp Network is a permissioned, cross-chain blockchain ecosystem designed to integrate regulatory compliance directly into its architecture. It offers a modular infrastructure that supports both public and private sub-networks, ensuring scalability and interoperability across various platforms. The network emphasizes adherence to data privacy laws such as GDPR and incorporates KYC and KYB protocols to maintain a secure and compliant environment. Kalp Network provides tools like Kalp Studio for streamlined decentralized application development and the Kalp Wallet for managing digital assets within its ecosystem.

Kalp Network's Gini Vesting smart contract is a token distribution management system deployed on the Kalp Network that controls the release of GINI tokens to various stakeholders through predefined vesting schedules. The contract handles 14 distinct allocation groups, implements time-based vesting mechanisms with configurable parameters, and ensures secure token distribution through automated claims processing and role-based access control. Built using the Kalp SDK, the system maintains the integrity of token distribution through comprehensive state management and event logging while providing a transparent and automated approach to token vesting.

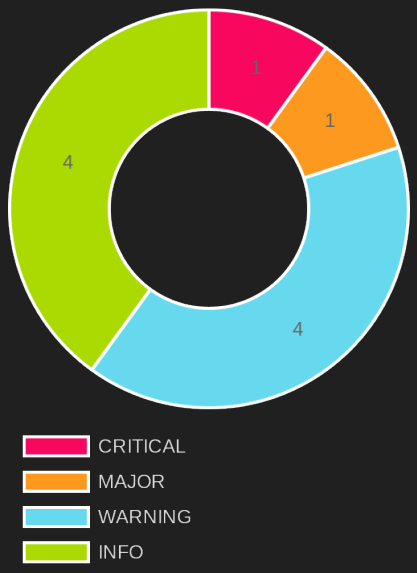
The audit process involved a comprehensive approach, including manual code review, automated analysis, and extensive testing and simulations of the smart contracts to assess the project's security and functionality. The audit covered a total of 8 files, encompassing 1112 lines of code. The codebase was thoroughly examined, with the audit team collaborating closely with Kalp Network 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.

1.2 SUMMARY OF FINDINGS

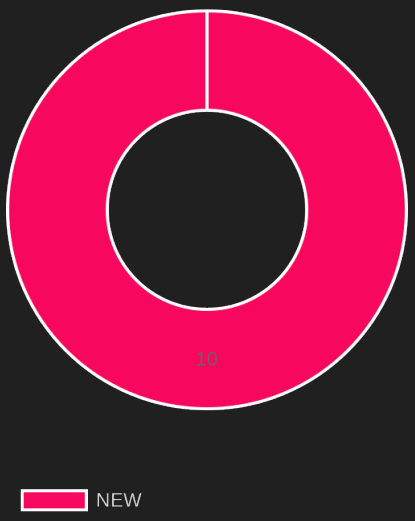
The table below provides a comprehensive summary of the audit findings, categorizing each by status and severity level. For a detailed description of the severity levels and statuses of findings, see the Findings Classification Reference section.

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

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	**1**	1	0	0	0
MAJOR	**1**	1	0	0	0
WARNING	**4**	4	0	0	0
INFO	**4**	4	0	0	0
TOTAL	**10**	**10**	**0**	**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	Kalp Network
Project name	Kalp Gini Vesting Smart Contracts
Category	Vesting
Website	www.kalp.network
Documentation	kalp-network.gitbook.io/gini-smartcontracts-documentation
Repository	github.com/p2eengineering/gini-vesting-contract
Initial Commit	`0129335c2067f7b836c69e84653efc6957b4f7d5`
Platform	L1
Network	Kalp Network
Languages	Go
Lead Auditor	Alexander Mazaletskiy - am@oxor.io
Project Manager	Nataly Demidova - nataly@oxor.io

2.3 PROJECT TIMELINE

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

Date	Event
December 26, 2024	Client engaged Oxorio requesting an audit.
January 15, 2025	The audit team initiated work on the project.
January 30, 2025	Submission of the comprehensive audit report.

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	vesting/constants.go	60	6	1	**53**	0
2	vesting/errors.go	93	19	0	**74**	3
3	vesting/events.go	115	20	0	**95**	17
4	vesting/helpers.go	102	23	0	**79**	16
5	vesting/internal.go	201	47	0	**154**	27
6	vesting/models.go	256	52	0	**204**	28
7	vesting/smartcontract.go	596	144	0	**452**	31
8	vesting/variables.go	1	0	0	**1**	0
	Total	**1424**	**311**	**1**	**1112**	**24**

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 Gini Vesting smart contract implements a token vesting mechanism for the GINI token distribution on the Kalp Network. The contract manages various vesting schedules for different stakeholder groups including team members, foundation, advisors, and investors.

The core functionality includes:

- Token distribution management across 14 distinct allocation groups
- Configurable vesting periods with cliff periods and TGE (Token Generation Event) percentages
- Automated vesting schedule calculations and token release mechanisms
- Beneficiary management system with claim verification
- Integration with the GINI token contract for transfer operations

The contract utilizes the Kalp SDK for blockchain interactions and implements key security features such as:

- Role-based access control with Kalp Foundation as the administrator
- Address validation for both users and contracts
- State management safeguards
- Event emission for tracking vesting activities

Key components of the vesting mechanism are implemented in the main contract file `vesting/smartcontract.go` which handles initialization, beneficiary management, and claim processing.

The vesting schedules are enforced through a time-based calculation system `vesting/internal.go` that determines claimable amounts based on elapsed intervals and initial unlock percentages.

The project follows a modular architecture with separate components for:

- Core vesting logic
- State management
- Event handling
- Helper functions
- Access control

The contract is designed to operate within the Kalp Network ecosystem, utilizing its native transaction context interface for blockchain interactions and state management.

This implementation serves as the foundation for GINI token's distribution strategy, ensuring transparent and automated token release according to predefined schedules while maintaining security and auditability through comprehensive event logging.

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 project's codebase implements a robust access control mechanism with multiple differentiated roles to manage system functionalities efficiently. However, a critical vulnerability was identified due to the lack of proper authorization checks for token claims, as highlighted in issue C-01. Addressing this flaw is essential to prevent unauthorized access and ensure system security.	**Fair**
Arithmetic	The project has no identified issues related to inadequate handling of arithmetic operations. All arithmetic operations are executed and verified correctly.	**Excellent**
Complexity	The contract appears well-structured; however, attention should be paid to removing hardcoded values. (W-04)	**Good**
Data Validation	The project performs data validation across many components; however, there are gaps in validation under certain conditions. Detailed attention is required to address issue W-02.	**Good**
Decentralization	Contract management is role-based; a decentralized approach is not applicable here.	**Not Applicable**
Documentation	Documentation regarding functionality and limitations was provided, and it is highly helpful in understanding the codebase and its functionality effectively.	**Excellent**
External Dependencies	The project does not interact with any external smart contracts in its logic; therefore, this metric is not applicable in this context.	**Not Applicable**

Category	Assessment	Result
Error Handling	The project demonstrates robust exception handling throughout the codebase, utilizing custom errors with clear naming and descriptions. However, a few minor issues related to error handling (I-04) have been identified.	**Good**
Logging and Monitoring	The project exhibits excellent logging capabilities, recording all important events within the system.	**Excellent**
Low-Level Calls	The project is free from low-level calls, ensuring a higher level of security by avoiding potential pitfalls associated with direct, low-level interactions with the blockchain.	**Not Applicable**
Testing and Verification	Working tests were provided for the codebase, with a coverage of 80%, which is generally sufficient. However, not all edge cases are thoroughly tested, as indicated by the identified issues. Expanding test cases to cover these scenarios would enhance the robustness and reliability of the system.	**Good**

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
vesting/smartcontract.go	**7**	1	1	3	2
vesting/internal.go	**2**	0	0	1	1
vesting/models.go	**2**	0	0	0	2
vesting/helpers.go	**1**	0	0	0	1

2.8 CONCLUSION

A comprehensive audit was conducted on the vesting contract codebase, identifying 1 critical and 1 major issue, along with numerous warnings and informational notes. The audit revealed potential security risks and logical flaws, including the ability for unauthorized parties to claim tokens on behalf of a beneficiary and the possibility of exceeding the `vestingTotalSupply`, leading to improper token distribution. Additional concerns were identified in documentation mismatches, missing validation checks, incorrect event logging, and the presence of hardcoded values, which could hinder the maintainability and security of the smart contract.

The proposed changes focus on enforcing strict validation for claim destination addresses, preventing excess allocations, aligning documentation with code behavior, and consolidating contract parameters for better maintainability. Implementing these recommendations is crucial to ensure the integrity of the vesting mechanism and to enhance the overall security and reliability of the smart contract. Addressing these issues will mitigate potential risks and improve compliance with industry best practices.

3 FINDINGS REPORT

OXORIO

3.1 CRITICAL

C-01	Missing Check of Claim Destination Address in `smartcontract.go`
Severity	**CRITICAL**
Status	• NEW

Location

File	Location	Line
smartcontract.go	function `ClaimAll`	275

Description

In the function `ClaimAll`, tokens are transferred to a specific address from all allocations associated with that beneficiary:

```
func (s *SmartContract) ClaimAll(ctx kalpsdk.TransactionContextInterface, beneficiary
string) error {
  if !IsUserAddressValid(beneficiary) {
    return ErrInvalidUserAddress(beneficiary)
  }

  signer, err := GetUserId(ctx)

  userVestingList, err := GetUserVesting(ctx, beneficiary)

  // ...

  err = TransferGiniTokens(ctx, signer, totalClaimAmount.String())

  return err
}
```

After calculating all tokens available for claim, they are transferred to the `signer` address. However, there is no check to ensure that the `signer` address matches the `beneficiary`

address or that the `beneficiary` has given any approval for the `signer` to withdraw their tokens. This allows anyone to withdraw tokens belonging to the `beneficiary` to their own address.

Recommendation

We recommend refactoring the `ClaimAll` function logic to ensure that the claim destination address either matches the `beneficiary` address or is explicitly approved by the `beneficiary`.

3.2 MAJOR

M-01	`VestingTotalSupply` can be exceeded in `smartcontract.go`
Severity	**MAJOR**
Status	• NEW

Location

File	Location	Line
smartcontract.go	function `AddBeneficiaries`	130

Description

In the function `AddBeneficiaries`, new beneficiaries are added to the vesting contract:

```
func (s *SmartContract) AddBeneficiaries(...) error {

  // ...

  vestingTotalSupply, ok := new(big.Int).SetString(vestingPeriod.TotalSupply, 10)

  if vestingTotalSupply.Cmp(totalAllocations) < 0 {
    return ErrTotalSupplyReached(vestingID)
  }

  vestingTotalSupply.Sub(vestingTotalSupply, totalAllocations)

  EmitBeneficiariesAdded(ctx, vestingID, totalAllocations.String())

  return nil
  }
```

At the last step of the `AddBeneficiaries` function, there is a check to ensure that `totalAllocations` is less than or equal to `vestingTotalSupply`, which is set during initialization for each vesting ID. However, after this check, the `vestingTotalSupply` value

is reduced by `totalAllocations` but not saved back to the contract's storage. This allows the `AddBeneficiaries` function to be called again for the same vesting ID, passing the `vestingTotalSupply.Cmp(totalAllocations) < 0` check each time, potentially resulting in `totalAllocations` across multiple calls exceeding `vestingTotalSupply`. This can lead to token distribution errors, where beneficiaries of a particular vesting ID receive more tokens than allocated, while other vesting IDs are left short of tokens.

Additionally, in the `Initialize` function, when a beneficiary is added for the `EcosystemReserve`, the `vestingTotalSupply` storage variable is not updated. This may result in a similar issue for the `EcosystemReserve` vesting ID.

Recommendation

We recommend refactoring the `AddBeneficiaries` function to ensure that repeated calls cannot exceed the `vestingTotalSupply` set during the `Initialize` function for each vesting ID.

3.3 WARNING

W-01	Documentation mismatch in `internal.go`
Severity	**WARNING**
Status	• NEW

Location

File	Location	Line
internal.go	function `validateNSetVesting`	48

Description

In the function `validateNSetVesting`, a variable of the `VestingPeriod` struct type is initialized:

```
vestingPeriod := &VestingPeriod{
  TotalSupply:         totalSupply,
  CliffStartTimestamp: startTimestamp,
  StartTimestamp:      startTimestamp + cliffDuration,
  EndTimestamp:        startTimestamp + duration + cliffDuration,
  Duration:            duration,
  TGE:                 tge,
}
```

The `TGE` field in the `VestingPeriod` struct type is used in the code to represent the percentage of tokens unlocked at the Token Generation Event (TGE) moment. However, the documentation describes the `TGE` field in the `VestingPeriod` struct as: "The timestamp of the Token Generation Event (TGE)." This discrepancy between the implementation and documentation creates confusion about the purpose and behavior of the `TGE` field.

Recommendation

We recommend resolving the mismatch between the documentation and the code implementation to ensure consistency and clarity in the project. Update either the code or the documentation to reflect the accurate meaning of the `TGE` field in the `VestingPeriod` struct.

W-02	Missing check in `smartcontract.go`
Severity	**WARNING**
Status	• NEW

Location

File	Location	Line
smartcontract.go	function `Initialize`	19

Description

In the function `Initialize`, there is a check ensuring that the `startTimestamp` parameter value is not zero:

```
func (s *SmartContract) Initialize(ctx kalpsdk.TransactionContextInterface, startTimestamp
uint64) error {
  logger := kalpsdk.NewLogger()
  logger.Infoln("Initialize Invoked.... with arguments ", startTimestamp)

  if startTimestamp == 0 {
    return ErrCannotBeZero
  }

  // ...
}
```

However, the `startTimestamp` parameter value can still be less than `currentTimestamp`. This creates a potential issue where `startTimestamp` can be set in the past, leading to tokens being partially or fully unlocked at the time of vesting initialization.

Recommendation

We recommend adding a check to ensure that `startTimestamp >= currentTimestamp` to prevent premature token unlocking during the initialization of vesting.

W-03	Incorrect event message in `smartcontract.go`
Severity	**WARNING**
Status	• NEW

Location

File	Location	Line
smartcontract.go	function `ClaimAll`	269

Description

In the function `ClaimAll`, the logger creates a log message about the invocation of `ClaimAll`:

```
func (s *SmartContract) ClaimAll(ctx kalpsdk.TransactionContextInterface, beneficiary
string) error {
  logger := kalpsdk.NewLogger()
  logger.Infoln("GetVestingData Invoked.... with arguments ", beneficiary)

  // ...
}
```

However, the log message is incorrect and refers to a different function, leading to confusion and inaccurate event tracking.

Recommendation

We recommend updating the log message to accurately reflect the `ClaimAll` function invocation, ensuring clarity and consistency in event logging.

W-04	Hardcoded values in `smartcontract.go`
Severity	**WARNING**
Status	• NEW

Location

File	Location	Line
smartcontract.go	function `Initialize`	48-59

Description

In the function `Initialize`, multiple vesting IDs are initialized with hardcoded values:

```
validateNSetVesting(ctx, Team.String(), 30*12*24*60*60, startTimestamp, 30*24*24*60*60,
ConvertGiniToWei(300000000), 0)
validateNSetVesting(ctx, Foundation.String(), 0, startTimestamp, 30*12*24*60*60,
ConvertGiniToWei(220000000), 0)
validateNSetVesting(ctx, PrivateRound1.String(), 30*12*24*60*60, startTimestamp,
30*12*24*60*60, ConvertGiniToWei(200000000), 0)
validateNSetVesting(ctx, PrivateRound2.String(), 30*6*24*60*60, startTimestamp,
30*12*24*60*60, ConvertGiniToWei(60000000), 0)
validateNSetVesting(ctx, Advisors.String(), 30*9*24*60*60, startTimestamp, 30*12*24*60*60,
ConvertGiniToWei(30000000), 0)
validateNSetVesting(ctx, KOLRound.String(), 30*3*24*60*60, startTimestamp, 30*6*24*60*60,
ConvertGiniToWei(30000000), 25)
validateNSetVesting(ctx, Marketing.String(), 30*1*24*60*60, startTimestamp, 30*18*24*60*60,
ConvertGiniToWei(80000000), 10)
validateNSetVesting(ctx, StakingRewards.String(), 30*3*24*60*60, startTimestamp,
30*24*24*60*60, ConvertGiniToWei(180000000), 0)
validateNSetVesting(ctx, EcosystemReserve.String(), 0, startTimestamp, 30*150*24*60*60,
ConvertGiniToWei(560000000), 2)
validateNSetVesting(ctx, Airdrop.String(), 30*6*24*60*60, startTimestamp, 30*9*24*60*60,
ConvertGiniToWei(80000000), 10)
validateNSetVesting(ctx, LiquidityPool.String(), 0, startTimestamp, 30*6*24*60*60,
ConvertGiniToWei(200000000), 25)
validateNSetVesting(ctx, PublicAllocation.String(), 30*3*24*60*60, startTimestamp,
30*6*24*60*60, ConvertGiniToWei(60000000), 25)
```

These values should be stored in a dedicated `constants.go` file along with other parameters of the vesting contract. Hardcoded values scattered across different project files make auditing and maintenance challenging, leading to potential errors when updating parameters, as it becomes difficult to locate and modify all instances.

Recommendation

We recommend consolidating all parameters of the vesting contract into a `constants.go` file to enhance readability, maintainability, and adherence to good coding practices.

3.4 INFO

I-01	`CompositeKey` not used in `models.go`
Severity	**INFO**
Status	• NEW

Location

File	Location	Line
models.go	function `SetVestingPeriod`	112

Description

In the function `SetVestingPeriod` and various other locations within the `models.go` file, the `fmt.Sprintf` method is used to create keys for storing values in contract storage:

```
func SetVestingPeriod(ctx kalpsdk.TransactionContextInterface, vestingID string, vesting
*VestingPeriod) error {
  vestingKey := fmt.Sprintf("vestingperiod_%s", vestingID)
  // ...
```

However, the `ctx.CreateCompositeKey` method is specifically designed for generating keys in a structured and consistent manner. `CreateCompositeKey` combines the provided `attributes` to form a composite key that can be directly used with `PutState()` and related methods.

Recommendation

We recommend replacing `fmt.Sprintf` with `ctx.CreateCompositeKey` to generate keys. This approach ensures consistent key creation, improves readability, and aligns with best practices for using the framework's built-in methods.

I-02	Unrecommended method usage in `models.go`
Severity	**INFO**
Status	• NEW

Location

File	Location	Line
models.go	function `SetVestingPeriod`	118

Description

In the function `SetVestingPeriod` and various other locations within the `models.go` file, the `PutStateWithoutKYC` method is used to store data in the contract storage:

```
func SetVestingPeriod(...) error {
  // ...

  err = ctx.PutStateWithoutKYC(vestingKey, vestingAsBytes)
  if err != nil {
    return NewCustomError(http.StatusInternalServerError, "failed to set vesting", err)
  }

  // ...
}
```

However, according to the documentation, it is recommended to use `PutStateWithKYC`, as this method enforces KYC restrictions, adding an extra layer of security to contract operations. The current usage of `PutStateWithoutKYC` bypasses this important security measure.

Recommendation

We recommend replacing `PutStateWithoutKYC` with `PutStateWithKYC` to enforce KYC restrictions and ensure enhanced security in contract operations.

I-03	Potential documentation mismatch in `smartcontract.go`
Severity	**INFO**
Status	• NEW

Location

File	Location	Line
smartcontract.go	function `CalculateClaimAmount`	206

Description

In the function `CalculateClaimAmount`, the claimable token amount is calculated based on the current timestamp:

```
func (s *SmartContract) CalculateClaimAmount(...) (string, error) {
  // ...

  if uint64(currentTime.Seconds) <= vestingPeriod.CliffStartTimestamp {
    return "0", nil
  }

  // ...

  claimAmount := new(big.Int)
  claimAmount.Add(claimableAmount, initialUnlock)
  claimAmount.Sub(claimAmount, beneficiaryClaimedAmount)

  // ...

  return claimAmount.String(), nil
}
```

In the current implementation, `initialUnlock` tokens are distributed to the beneficiary immediately at the start of the cliff period (without waiting for the end of the cliff period). However, the documentation states: "Time-Based Validation: If the current time is before the cliff period, return 0" — which could imply either the start or the end of the cliff period. If the intention is the start of the cliff period, the implementation is correct; otherwise, the tokens should only be distributed after the cliff period ends.

Recommendation

We recommend verifying the documentation's intended meaning and either updating the documentation or modifying the implementation to align with the expected behavior.

I-04	Error ignored
Severity	**INFO**
Status	• NEW

Location

File	Location	Line
smartcontract.go	function `CalculateClaimAmount`	204
smartcontract.go	function `Claim`	553
internal.go	function `addBeneficiary`	84
internal.go	function `TransferGiniTokens`	194
helpers.go	function `IsContractAddressValid`	41
helpers.go	function `IsUserAddressValid`	51

Description

In the mentioned locations, the returned error is ignored and not processed. While this does not currently lead to issues, ignoring errors is a poor programming practice and could result in undesirable consequences.

Recommendation

We recommend processing all returned errors to improve the security and stability of the codebase.

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