Realize

SMART CONTRACTS REVIEW

zokyo

November 13th 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING REALIZE

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 1 Medium issue: 1 resolved = 0 points deducted
- 0 Low issues: 0 points deducted
- 3 Informational issues: 3 resolved = 0 points deducted

Thus, the score is 100

TECHNICAL SUMMARY

This document outlines the overall security of the Realize smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Realize smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Realize team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Realize repository:
Repo: https://github.com/realizeassets/realize-protocol

Last commit - 21814aea9f1fff1681e7f0dca61d8b2e9c663ced

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- ./Hub.sol
- ./Token.sol
- ./TokenManager.sol
- ./IHubToken.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Nakama smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contract/s by industry leaders.

03 Thorough manual review of the codebase line by line.

Executive Summary

The `Token, TokenManager,` and `Hub` contracts collectively establish a permissioned token system with batch processing, restricted transfers, and role-based access for multi-chain applications. The `Token` contract extends LayerZero's `OFT` (Omnichain Fungible Token), allowing batch minting, transferring, and burning of tokens. It stores batches of tokens by user and tracks the original minter, updating records on each transfer or burn. Transfer actions emit events, and the contract has (`isTrackOnChainPaused`) to control if batch tracking should occur. To restrict transfers, the contract uses a `kycList` from the `IList` interface to only allow pre-approved addresses as recipients.

The `TokenManager` contract, inheriting from `Hub`, maps various operation types (defined by `OperationType` enums) to permission lists using the `IList` interface, so different actions can have different restrictions. It runs validation checks for each `OperationType` before processing, ensuring that only addresses allowed by the corresponding list can perform certain actions. Initialization sets up lists for each operation type, ensuring a structured permissioned environment. The `Hub` contract, serving as a foundational layer, manages the central token contract, recipient addresses for collateral, and operator roles. It uses AccessControl for granular role-based access to administrative and operational functions. Together, these contracts create a scalable system for multi-chain environments, allowing controlled minting, burning, and transferring of tokens.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Realize team and the Realize team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Unchecked Return Value of transferFrom in requestRedemption	Medium	Resolved
2	Unused variable	Informational	Resolved
3	There is no reason for mint(), burn(), and burnFrom() functions to be payable - can lead to fund loss	Informational	Resolved
4	Use of abi.encodeWithSignature Instead of abi.encodeCall	Informational	Resolved

MEDIUM-1 | RESOLVED

Unchecked Return Value of `transferFrom` in `requestRedemption`

The `transferFrom` function on `Hub.sol` is called to transfer tokens from the user to the `operatorBurner` address. However, this function's return value is not checked to confirm that the transfer was successful and since the function returns a boolean, the return value must be checked.

Recommendation:
Check the return value of `transferFrom` to ensure the transfer was successful.

INFORMATIONAL-1 | RESOLVED

Unused variable

Location: Token.sol

The contract declares the destinationEIds variable but the variable is not used in the contract.

Recommendation:
Remove the unused variable.

INFORMATIONAL-2 | RESOLVED

There is no reason for mint(), burn(), and burnFrom() functions to be payable - can lead to fund loss

Location: Token.sol

In the current implementation, the mint(), burn(), and burnFrom() functions are payable, which means that users can send ETH along with the transaction call, however the functions has no operations to do with ETH, meaning that if a user accidentally sends ETH along with their function call, the ETH will be trapped in the contract which leads to fund loss for the user.

Recommendation:
Remove payable keyword.

INFORMATIONAL-3 | RESOLVED

Use of `abi.encodeWithSignature` Instead of `abi.encodeCall`

The `transferTokenOwnership` and `acceptOwnershipOfToken` functions in Hub.sol use `abi.encodeWithSignature` to encode function calls. While this is a valid encoding approach, it is less type-safe than `abi.encodeCall`, which allows for compile-time type checking of function signatures. By using `abi.encodeCall`, any mismatch in argument types or numbers would trigger a compile-time error, improving the safety and reliability of function calls.

Recommendation:
Replace `abi.encodeWithSignature` with `abi.encodeCall` for improved type safety

	./Hub.sol ./Token.sol ./TokenManager.sol ./IHubToken.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the Realize team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Realize team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo