RAILGUN_

SMART CONTRACT AUDIT

zokyo

Nov 23rd, 2021 | v. 1.0

Security Audit Score

PASS

Zokyo Security Team has concluded that this smart contract passes security qualifications and bear no security or operational risk

TECHNICAL SUMMARY

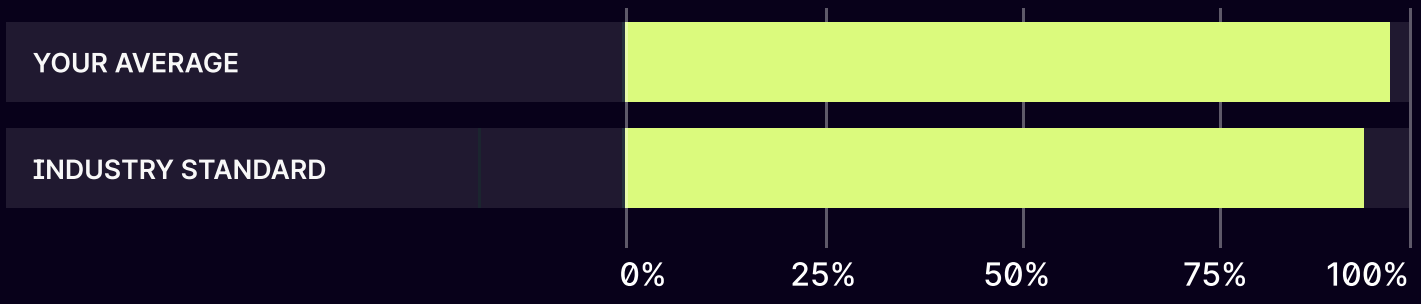
This document outlines the overall security of the RAILGUN smart contracts, evaluated by Zokyo's Blockchain Security team.

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

Contract Status

There were no critical issues found during the audit.

Testable Code



The testable code is 98.2%, which is above the industry standard of 95%.

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

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the RAILGUN repository:

Repository: https://github.com/Railgun-Privacy/contract/commit/0418a0f1bf0e58e5b3bab8870112b7648ff20aca

Last commit: 97af307fa1d737d4526323acc3d0ef372c703417

Contracts under the scope:

- Commitments;
- Globals;
- RailgunLogic.

Additional requirements to check:

- For the logic system:
  - Users can spend all their funds and only their funds, there is no hidden mint or burn in the contract.
    Check result – confirmed.
  - Assuming high volume + a relayer system to pay gas, a user's actions can't be linked to them (exception: deposits and withdraws can be linked to the depositing and withdrawing addresses respectively).
    Check result – confirmed.
- For governance:
  - Only a DAO vote has the ability to make the governance contract call any function on any contract.
    Check result – confirmed.
  - A DAO vote can delegate the ability to call certain other contracts or functions to another address (EOA or Contract).
    Check result – confirmed.
  - AA DAO vote can remove the ability to call certain other contracts or functions from another address (EOA or Contract).
    Check result – confirmed.
  - Only the governance process can change the deployed code (Proxy pattern).
    Check result – confirmed.

Throughout the review process, care was taken to ensure that the token contract:

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of RAILGUN smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. 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 contracts by industry leaders.
03	Testing contract logic against common and uncommon attack vectors.
04	Thorough, manual review of the codebase, line-by-line.

Executive Summary

The Zokyo team has conducted a security audit of the given codebase. The contracts provided for an audit are well written and structured. All the findings within the auditing process are presented in this document.

During the auditing process, our auditor's team found 2 issues which were successfully resolved by the RAILGUN contributors.

Taking into consideration the audit results, we can state that the audited smart contracts are safe to deploy and bear no security or operational risk for neither contract owner, nor end user. Hence, the score of the audit is set to 100 to the provided codebase.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. 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 ability of the contract to compile or operate in a significant way.

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

Unsafe transfer usage

LOW | RESOLVED

In contract RailgunLogic.sol in functions transact() and generateDeposit() is used transfer() function for transferring fee to the treasury contract. Since transfers have fixed 2,300 gas for executing it is unsafe to use it with new versions of solidity.

Recommendation:
Consider to use call() instead of transfer().

Re-audit:
Fixed. Replaced transfer() with call().

Misleading NatSpec comment

INFORMATIONAL | RESOLVED

In Commitments.sol contracts for function insertLeaves() provided misleading comments in lines 138-142. In section @notice and @dev descriptions in not valid for the functionality.

Recommendation:
Consider fixing the comment.

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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests are written by the RAILGUN contributors

Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	Uncovered Lines
logic\	96.40	61.11	88.89	97.30	245, 250, 253
Commitments.sol	94.59	100	75	92.11	
Globals.sol	100	100	100	100	
RailgunLogic.sol	97.30	56.23	100	100	
All files	**96.40**	**61.11**	**88.89**	**97.30**	

Test Results

Logic/Commitments

- ✓ Should initialize the tree correctly
- ✓ Should update the tree correctly (60203ms)

Logic/RailgunLogic

- ✓ Should verify proofs (19092ms)
- ✓ Should deposit token correctly (19962ms)
- ✓ Should collect treasury fees correctly (44755ms)
- ✓ Should deposit with 2 outputs correctly (22064ms)
- ✓ Should deposit with 3 outputs correctly (22560ms)
- ✓ Should deposit, do an internal transaction, and withdraw (66361ms)
- ✓ Should transact with large circuit (125395ms)
- ✓ Should deposit and generate commitments correctly (935ms)
- ✓ Should be able to spend from generated commitment (23790ms)
- ✓ Should do batch transactions(67603ms)

13 passing (9m)

Tests written by Zokyo Security team

As part of our work assisting RAILGUN in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the RAILGUN contract requirements for details about issuance amounts and how the system handles these.

Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	Uncovered Lines
logic\	98.20	79.63	100	100	
Commitments.sol	100	100	100	100	
Globals.sol	100	100	100	100	
RailgunLogic.sol	97.63	77.08	100	100	
All files	**98.20**	**79.63**	**100**	**100**	

Test Results

Contract: Commitments

- ✓ should initialize merkle tree correct (48ms)
- ✓ should update the tree correctly (1156ms)
- ✓ should create a new tree correct (93ms)

Contract: RailgunLogic

- ✓ Should initialize railgun logic correct (144ms)
- ✓ Should change treasury correct (211ms)
- ✓ Should revert change treasury by not owner(92ms)
- ✓ Should change fee correct (117ms)
- ✓ Should revert change fee by not owner
- ✓ Should verify proofs (19469ms)
- ✓ Should deposit whitelisted token correct (20339ms)
- ✓ Should deposit and withdraw whitelisted token correct (47679ms)
- ✓ Should revert deposit when adaptID address is incorrect (21244ms)
- ✓ Should revert deposit when the merkle root is incorrect(21037ms)

```
✓ Should revert deposit not whitelisted token (21518ms)
✓ Should collect treasury fees correctly (46725ms)
✓ Should generateDeposit and withdraw token from the generated commitments(24076ms)
✓ Should revert generateDeposit if fee not paid (77ms)
✓ Should revert generateDeposit if deposited amount is zero
✓ Should revert generateDeposit if deposit not whitelisted token (89ms)

19 passing (4m)
```

We are grateful to have been given the opportunity to work with the RAILGUN contributors.

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

Zokyo's Security Team recommends that the RAILGUN contributors put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo