RAILGUN_

SMART CONTRACT AUDIT

zokyo

February 3rd 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

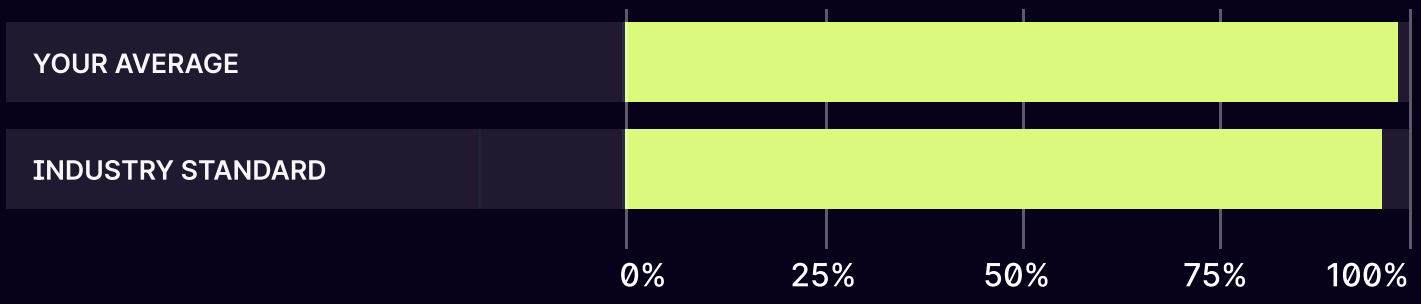
This document outlines the overall security of the Railgun smart contracts evaluated by the Zokyo Security team.

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

Contract Status

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

Testable Code



100% of the code is testable, 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 contracts 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 Railgun team put in place a bug bounty program to encourage further active analysis of the smart contracts.

Table of Contents

Auditing Strategy and Techniques Applied 3

Executive Summary 4

Structure and Organization of the Document 5

Complete Analysis 6

Code Coverage and Test Results for all files written by Zokyo Security 14

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Railgun repository: https://github.com/Railgun-Privacy/contract

Last commit: a49a82d1589426ece9f6b463c51635002f542e65

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

- Sender.sol
- Executor.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 resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Railgun smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Hardhat 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.	03	Testing contracts logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contracts by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

Contracts are well written and structured. There were no critical issues found during the audit, but few issues with medium severity, some of low severity and informational issues are spotted. All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner and the investors interacting with it. They are described in detail in the “Complete Analysis” section.

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 Railgun team and the Railgun team is aware of it, but they have chosen to not solved 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

System Overview

The main use case of the contracts audited by the Zokyo team is to provide governance functionality across layers 1-2 (Ethereum-Arbitrum). Users create tasks which include one or more actions aggregated together in layer-2 then in layer one admin enables the tasks to be executed. In that sense, tasks are executed in layer-2 with less gas cost while the trigger is securely being registered at layer-1 without overloading this layer with the costly computation of the tasks. During the manual and testing stages of the contracts audit, multiple issues were found. All those can be found in the following sections. Beside these findings, there are also remarks that have to be made about the overall security of the contracts which are submitted for this audit.

During the initial assessment of the contracts, it has been discovered that the admin can perform a couple of owner related actions that can affect the ecosystem significantly. The admin actions include triggering the tasks and also setting the address of the executor at layer-2. After further inspection, the Zokyo team established that these actions constitute a centralization risk for the security of the contracts within the audit scope. It is also worth mentioning that there are minor issues present like unnecessary long revert messages, redundant ownership transfer call, and important return values are unused. Zokyo advises the team to acknowledge these design decisions and take extra care while operating the contracts in their current design.

As for later stages, the developers informed Zokyo team that they intend to have the contracts taken control by governance, hence this omits the risk of centralization. Other issues are also addressed by the developers until we pass the assessment.

Findings Summary

#	Title	Risk	Status
1	Centralization risk	Medium	Resolved
2	canExecute causes ambiguity	Medium	Resolved
3	Return value not checked	Low	Resolved
4	Unnecessary transferOwnership	Low	Resolved
5	Error messages are long	Low	Acknowledged
6	Lock solidity version	Informational	Acknowledged
7	No validation of input address	Informational	Resolved
8	Better return uint256 taskID created	Informational	Resolved

MEDIUM | RESOLVED

Centralization risk

In Sender.sol - Admin enjoys much authority. The general theme is that admin has power to call several functions like setting the `executorL2` and enable the execution of tasks. Some functions can be more highly severe to be left out controlled by one wallet more than other functions.

Recommendation:
Apply governance methods / use multisig wallets.

Fix#1:
No message or comment from partner about this, stating that multisig are considered to be used (this is necessary to mitigate the effect of that risk).

Fix#2:
Partner stated that contracts ownership are intended to be overtaken by governance module. There is a centralization risk if this contract is owned by an EOA, care should be taken to verify that the owner is the governance delegator after deployment to prevent potential for exploits.

MEDIUM | RESOLVED

canExecute causes ambiguity

In Executor.sol - `executeTask` method executes task that result on marking `canExecute=false`. It is also the case from the perspective of layer-1 that a failed call to `Executor::readyTask` after a successful `Sender::readyTask` that ends up having `canExecute=false`.

Recommendation:
Apply an enum instead of bool `canExecute` that represents 3 states of execution
`NotReady`
`Ready`
`Executed`
Then apply the required checks that validate the states of execution in the relevant methods.

LOW | RESOLVED

Return value not checked

In Sender.sol - In body of `readyTask` there is unchecked return value of an external interaction

```
ARBITRUM_INBOX.createRetryableTicket{ value: submissionFee }(
  executorL2,
  0,
  submissionFee,
  // solhint-disable-next-line avoid-tx-origin
  tx.origin,
  // solhint-disable-next-line avoid-tx-origin
  tx.origin,
  0,
  0,
  data
);
```

Also, in Executor.sol - In body of `redeem(uint256)` there is unchecked return value of an external interaction that should return `bytes32`.

```
function redeem(uint256 _ticket) external {
  ARB_RETRYABLE_TX.redeem(bytes32(_ticket));
}
```

Recommendation:
check the returned unit of `createRetryableTicket`, apply needed validation on that returned value. Also, same with the returned `bytes32` of `redeem(uint256)` or pass the returned value to the caller.

Fix:
Methods are returning those returned values

LOW | RESOLVED

Unnecessary transferOwnership

In Sender.sol - In body of constructor, transferring ownership to `msg.sender` is already carried out by base contract's `Ownable` constructor.

```
constructor(address _admin, address _executorL2, IInboxPatched _arbitrumInbox) {
  Ownable.transferOwnership(msg.sender);
```

Recommendation:
remove that line of code.

LOW | ACKNOWLEDGED

Error messages are long

In Executor.sol - Error messages in the two require statements are long and considered costly in terms of gas.

Recommendation:

- Write shorter error messages
- Use custom errors which is the goto choice for developers since solidity v0.8.4 details about this shown here: soliditylang.

Fix#1 :
No fix to address this, there exist more occurrences of long revert messages in Executor.sol. And one occurrence in Sender.sol.

INFORMATIONAL | ACKNOWLEDGED

Lock solidity version

All contracts, Lock the pragma to a specific version, since not all the EVM compiler versions support all the features, especially the latest ones which are kind of beta versions, So the intended behavior written in code might not be executed as expected. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler, which may have higher risks of undiscovered bugs.

Recommendation:
fix version to 0.8.17 (the version stated in hardhat.config.ts)

Fix#1:.
No change occurred to fix this. Recommendation is to remove the caret to fix the compiler to a specified version.

INFORMATIONAL | RESOLVED

No validation of input address

In Sender.sol - `setExecutor(address)` does not validate the input to ensure it is non-zero.

Recommendation:
add require statement to ensure input address is not equal to a`ddress(0)`.

Fix:
Method setExecutorL2 is now validating the input address.

INFORMATIONAL | RESOLVED

Better return uint256 taskID created

In Executor.sol - `createTask` does create a new task and emit an event showing info about the task id just created to be referenced later. This might be tiresome for contracts, calling this method `createTask` to try and deduce the `taskID` . Contract using that method can know the id by knowing the length of `tasks`, but this is not straightforward and obvious.

Recommendation:
Have the method `createTask` return the `taskID` at the end of the method call.

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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting Railgun in verifying the correctness of their contracts code, our team was responsible for writing integration tests using the Hardhat testing framework.

The tests were based on the functionality of the code, as well as a review of the Railgun contracts requirements for details about issuance amounts and how the system handles these.

Test Arbitrum executor
- ✓ should create tasks
- ✓ should mark tasks as executable
- ✓ Call redeem function
- ✓ Should allow ready task be callable by L1 sender
- ✓ should check for timeout values
- ✓ should test for failed tasks

Test Sender
- ✓ Should send ready task message

Test Sender Owner
- ✓ Should allow only owner to call setExecutorL2
- ✓ Should not allow non-owner to call

9 passing (4s)

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	% UNCOVERED LINES
Executor.sol	100	100	100	100	
Sender.sol	100	100	100	100	
All files	**100**	**100**	**100**	**100**	

We are grateful for the opportunity to work with the Railgun 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 Railgun team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo