ScrollOwner and Rate Limiter Audit

OPENZEPPELIN SECURITY | OCTOBER 16, 2023

Security Audits

Table of Contents

- Table of Contents
- Summary
- Scope
- System Overview
- Trust Assumptions
- Privileged Roles
- High Severity
  - Abuse of Rate Limiter Mechanism to Perform DOS Attack
- Low Severity
  - Missing Docstrings
  - Utilizing Deprecated Function From Library
- Notes & Additional Information
  - Variable Cast is Unnecessary
  - Incorrect Function Visibility
  - Inconsistent Usage of msg.sender
  - Implicit Type Casting
  - Lack of Logs on Sensitive Actions
  - Unpinned Compiler Version
  - Naming Issue
- Recommendations

Summary

Type
zkEVM-based ZK-rollup, Bridge & Rollup
Timeline
From 2023-08-16
To 2023-08-23
Languages
Solidity

Total Issues
10 (8 resolved, 1 partially resolved)
Critical Severity Issues
0 (0 resolved)
High Severity Issues
1 (1 resolved)
Medium Severity Issues
0 (0 resolved)
Low Severity Issues
2 (1 resolved)
Notes & Additional Information
7 (6 resolved, 1 partially resolved)

Scope

We audited the Scroll Owner pull request and the Rate Limiter pull request from the scroll-tech/scroll repository at commit ae2e010.

```
contracts
└── src
    ├── misc
    │   └── ScrollOwner.sol
    └── rate-limiter
        ├── ETHRateLimiter.sol
        ├── IETHRateLimiter.sol
        ├── ITokenRateLimiter.sol
        └── TokenRateLimiter.sol
```

```
contracts
└── src
    ├── L1
    │   ├── gateways
    │   │   ├── L1ERC20Gateway.sol
    │   │   └── L1ETHGateway.sol
    │   └── L1ScrollMessenger.sol
    ├── L2
    │   ├── gateways
    │   │   ├── L2CustomERC20Gateway.sol
    │   │   ├── L2StandardERC20Gateway.sol
    │   │   ├── L2WETHGateway.sol
    │   │   └── L2ETHGateway.sol
    │   └── L2ScrollMessenger.sol
    ├── gas-swap
    │   └── GasSwap.sol
    └── libraries
        ├── gateway
        │   └── ScrollGatewayBase.sol
        └── ScrollMessengerBase.sol
```

System Overview

Scroll is an EVM-equivalent ZK-rollup designed to be a scaling solution for Ethereum. It achieves this by interpreting EVM bytecode directly at the bytecode level, following a similar path to projects like Polygon's zkEVM and Consensys' Linea.

Scroll's architecture and code structure draw inspiration from other Layer 2 solutions like Arbitrum and Optimism, particularly in the design of their gateways, predeploys, and messaging contracts. Notably, a lot of code structures from Arbitrum's gateways and the `AddressAliasHelper.sol` contract are reused with minor modifications.

This audit reviewed two new additions to the protocol. `ScrollOwner` implements a role-based model for access control. This will allow administrators' addresses to execute sensitive

This report presents our findings and recommendations for the additions to the Scroll ZK-rollup protocol. In the following sections, we will discuss these aspects in detail. We urge the Scroll team to consider these findings in their ongoing efforts to provide a secure and efficient Layer 2 solution for Ethereum.

Trust Assumptions

During the course of the audit, several assumptions about the Scroll protocol were considered to be inherently trusted. These assumptions and the context surrounding them include:

- **EVM node and relayer implementation**: It is assumed that the EVM node implementation will work as described in the Scroll documentation, particularly the opcodes and their expected behavior. The relayer implementation is trusted to act in the best interest of the users.
- **Censoring**: The protocol is centralized as is, as the sequencer and prover have the ability to censor L2 messages and transactions. L1 to L2 messages are appended into a message queue that is checked against when finalizing, but the sequencer can currently choose to skip any message from this queue during finalization. This allows the chain to finalize even if a message is not provable. Therefore, it is worth noting that L1 to L2 messages from the `L1ScrollMessenger` or `EnforcedTxGateway` can be ignored and skipped. There are plans to remove this message-skipping mechanism after the mainnet launch, once the prover is more capable.
- **No escape hatch**: The Scroll protocol does not feature an escape hatch mechanism. This, combined with the potential for transaction censorship by the relayer, introduces a trust assumption in the protocol. In the event of the network going offline, users would not be able to recover their funds.
- **Whitelist ownership**: The whitelist contract has an owner who can update the whitelist status of different addresses. This implies trust in the owner of the whitelist to manage this list correctly and in the best interest of the system and its users.

Privileged Roles

- **Access Control:** The access control manager is a contract in which there is an address with default admin privileges that can perform the critical administrative action of giving and revoking roles for different addresses. This mechanism is used in the following contracts:
  - `ScrollOwner`: The default admin role can grant roles for addresses to execute functions through the contract. Every role will be associated with a designated set of functions tied to specific addresses permissible to execute within that role. Moreover, existing roles will come with execution delays ranging from 0 days for instant execution to 1 day, 7 days, and 14 days. This provides a dynamic control mechanism over the timing of function execution based on their respective impact levels.
  - `TokenRateLimiter`: The default admin role can update the total token amount limit. The admin can also grant a token spender role for the Scroll gateways and messengers to ensure a rate limit when depositing or withdrawing funds.
- **Proxy Admins:** Most of the contracts are upgradeable. Hence, most of the logic can be changed by the proxy admin. The following contracts are upgradeable:
  - The gateway contracts
  - `L1MessageQueue`
  - `L1ScrollMessenger`
  - `L2GasPriceOracle`
  - `ScrollChain`
  - `L2ScrollMessenger`
- **Implementation Owners:** Most contracts are also ownable. The following actions describe what the owner can do in each contract.
  - `L1ScrollMessenger`: Pause the relay of L2 to L1 messages and L1 to L2 message requests.
  - `EnforcedTxGateway`: Pause L1 to L2 transaction requests and change the fee vault.
  - `L1{CustomERC20|ERC721|ERC1155}Gateway`: Change the token mapping containing which L1 token is bridged to which L2 token.
  - `L1GatewayRouter`: Set the respective gateway for ETH, custom ERC-20s and default ERC-20s.
  - `L1MessageQueue`: Update the maximum allowed gas limit for L2 transactions, the gas price oracle to calculate the L2 fee, and the `EnforcedTxGateway` address that

- `ScrollChain`: Revert previously committed batches that have not yet been finalized, set addresses as sequencers, change the verifier, and update the maximum amount of L2 transactions that are allowed in a chunk (bundle of blocks).
- `FeeVault`: Change the messenger address that is used to withdraw the funds from L2 to L1, the recipient address of the collected fees, and update the minimum amount of funds to withdraw.
- `ScrollMessengerBase`: Change the fee vault address which collects fees for message relaying.
- `ScrollStandardERC20Factory`: Use the factory to deploy another instance of a standard ERC-20 token on L2.
- `L2ScrollMessenger`: Pause the relay of L1 to L2 messages and L2 to L1 message requests.
- `L2{CustomERC20|ERC721|ERC1155}Gateway`: Change the token mapping containing which L2 token is bridged to which L1 token.
- `L2GatewayRouter`: Set the respective gateway for ETH, custom ERC-20s and default ERC-20s.
- `L2MessageQueue`: Update the address of the messenger.
- `L2TxFeeVault`: Change the messenger address that is used to withdraw the funds from L1 to L2, the recipient address of the collected fees, and update the minimum amount of funds to withdraw.
- `L1BlockContainer`: Initialize the starting block hash, block height, block timestamp, block base fee, and state root.
- `L1GasPriceOracle`: Update the gas price and whitelist.
- `Fallback`: Withdraw ERC-20 tokens and ETH as well as execute arbitrary messages.
- `Whitelist`: Accounts can be whitelisted to change the L2 base fee on L1 as well as the intrinsic gas parameters.
- `GasSwap`: Withdraw stuck ERC-20 tokens and ETH, update the fees, and set the approved targets to call.
- `MultipleVersionRollupVerifier`: Set the new verifier and their starting batch index.
- `ETHRateLimiter`: Update the total ETH amount limit.

- **Prover**: The prover role can interact with the `ScrollChain` contract to prove batches that have already been committed by the sequencer, thus finalizing them.

Each of these roles presents a unique set of permissions within the Scroll protocol. The potential implications of these permissions warrant further consideration and mitigation to ensure the system's security and robustness.

Abuse of Rate Limiter Mechanism to Perform DOS Attack

The Rate Limiting mechanism is an excellent safety mechanism to add to a bridge. However, in its current implementation, it can be abused by a malicious attacker to perform a denial-of-service on the system.

The rate limiter contracts `ETHRateLimiter` and `TokenRateLimiter` keep track of the amount of `ETH` or tokens that have been deposited in a certain time period. Every time a user makes an asset deposit or withdrawal, the total amount of their transaction is added to the total amount of the current time period. If the total amount of the current time period exceeds the maximum amount allowed, the transaction is reverted.

A malicious user can make a large deposit, withdraw, and repeat the process multiple times until the rate limiter reaches its total amount limit. This will prevent other users from making deposits or withdrawals from the protocol. The only cost for the malicious user would be the gas fees incurred. Once the amount limit is reached, it can only be updated by admins with the `updateTotalLimit` functions. Based on the responses to our inquiries with the Scroll team, it was communicated that the rate limiter `updateTotalLimit` functions will have a full-day delay before being executable. This would allow a malicious user to perform a DOS attack for 24 hours before the rate limiter can be updated and even then, the malicious user could keep repeating the DOS attack.

Consider adding an emergency 0-day execution of the rate limiter's `updateTotalLimit` function. This would allow the Scroll team to update the rate limiter in case there is a DOS attempt. We also recommend using short time periods for the rate limiter and adding the capability to turn on withdrawal fees that can be enabled during an attack to discourage malicious deposit-withdrawal cycling.

Update: *Resolved, the Scroll team removed the delay for updating the limit total amount in pull request #838 at commit fd5dcda.*

Low Severity

Missing Docstrings

- Line 13 in `ETHRateLimiter.sol`
- Line 5 in `IETHRateLimiter.sol`
- Line 5 in `ITokenRateLimiter.sol`
- Line 13 in `TokenRateLimiter.sol`
- Line 96 in `TokenRateLimiter.sol`
- Line 13 in `ITokenRateLimiter.sol`

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Acknowledged, will resolve. The Scroll team stated:*

> *It will be fixed later on when we have more time.*

Utilizing Deprecated Function From Library

The `TokenRateLimiter` contract is setting up its default admin role in the constructor using the `_setupRole` function. However, this function has been deprecated.

To avoid potential future incompatibilities with OpenZeppelin's contracts library, consider using the `_grantRole` method instead.

Update: *Resolved in pull request #902 at commit daae8c4.*

Notes & Additional Information

Variable Cast is Unnecessary

Within the `ScrollOwner` contract, the `_target` variable in the `_execute` function was unnecessarily cast.

To improve the overall clarity, intent, and readability of the codebase, consider removing this unnecessary cast.

In the `ScrollOwner` contract, the `execute` function is marked as `public`, while the `_execute` function has `internal` visibility.

The `execute` function should be marked as `external` because it is not called internally by any of the functions in the `ScrollOwner` contract.

The `_execute` function should be marked as `private` because it is implemented in the top-level `ScrollOwner` contract and is not intended to be called by the derived contracts.

To improve the overall clarity, intent, and readability of the codebase, consider fixing the visibility for these functions.

Update: *Resolved in pull request #904 at commit 72d484c.*

Inconsistent Usage of `msg.sender`

The `ScrollOwner` contract inherits from OpenZeppelin's `AccessControl` contract through its inheritance chain. This contract uses the `_msgSender` function.

Likewise, the `ETHRateLimiter` contract inherits from OpenZeppelin's `Ownable` contract. This contract also uses the `_msgSender` function.

The `_msgSender` function from OpenZeppelin's `Context` contract is used to allow the usage of metadata execution.

However, the `ScrollOwner` and `ETHRateLimiter` contracts directly use `msg.sender` [1], [2] instead of the `_msgSender` function. Even though there is no security risk per se, having some methods use `msg.sender` while others use the `_msgSender` function can be confusing.

Consider standardizing the usage of the `_msgSender` function throughout the codebase to increase clarity and consistency.

Update: *Resolved in pull request #906 at commit d0780e9.*

`_currentPeriodStart` are being compared. However, `_currentPeriod.lastUpdateTs`'s type is `uint48` and `_currentPeriodStart`'s type is `uint256`.

Consider explicitly type-casting variables of different types when they are compared or used in the same calculation. This ensures consistency in computations and reduces the risk of errors resulting from data loss.

Update: *Resolved in pull request #907 at commit d0780e9.*

Lack of Logs on Sensitive Actions

In the `ScrollOwner` contract, the `updateAccess` function can modify a role's access to calling specific function selectors on a target contract. Although this function will not interfere with users' actions, it is recommended to emit events to allow users to track changes to their permissions and monitor for any anomalies.

In the `constructor` of the `ETHRateLimiter` contract, the storage variable `currentPeriod.limit` is set without emitting the respective `UpdateTotalLimit` event.

Consider emitting events when changing state variables. Moreover, consider replacing the manual variable declarations in constructors with the corresponding function to update those variables.

Update: *Resolved in pull request #908 at commit 0bb41be.*

Unpinned Compiler Version

`ScrollOwner.sol`, `IETHRateLimiter.sol` and `ITokenRateLimiter.sol` use an unpinned Solidity version.

Consider pinning the version of the Solidity compiler to match the pinned version in the other contracts. This should help prevent the introduction of unexpected bugs due to incompatible future releases.

Update: *Partially resolved in pull request #909 at commit 0c5fbb7.* `IETHRateLimiter.sol` *and* `ITokenRateLimiter.sol` *still use an unpinned Solidity version.*

to favor explicitness and readability.

Update: *Resolved in pull request #910 at commit 86556dd.*

Monitoring Recommendations

While audits help in identifying code-level issues in the current implementation and potentially the code deployed in production, the Scroll team is encouraged to consider incorporating monitoring activities in the production environment. Ongoing monitoring of deployed contracts helps identify potential threats and issues affecting production environments. With the goal of providing a complete security assessment, the monitoring recommendations section raises several actions addressing trust assumptions and out-of-scope components that can benefit from on-chain monitoring.

Scroll Owner

Monitoring any role changes from the Scroll Owner will help detect any unauthorized changes. This can help detect malicious activity or a compromised account.

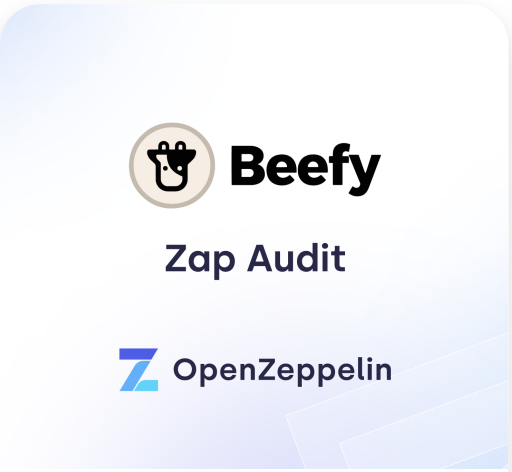
Rate Limiter

Monitoring the current rate limits against the total rate limit should be done to ensure that protocol withdrawals are not being paused because of rate limit breaches. It can also help detect DOS attacks against the rate limiter or suspicious activity. It is also recommended to monitor any updates to the rate limiter limit.

Over the course of this 8-day audit, we reviewed the Scroll Owner and the Rate Limiter codebases. We identified a single high-severity issue, as well as a few low-severity issues and additional notes. Overall, we commend the quality and thoughtful design of both codebases. The auditing process was seamless, and we appreciate the Scroll team's prompt responses to our inquiries throughout the process.

The implementation of the Scroll Owner for governance and the integration of the Rate Limiter for the bridge serve as significant enhancements to the protocol.

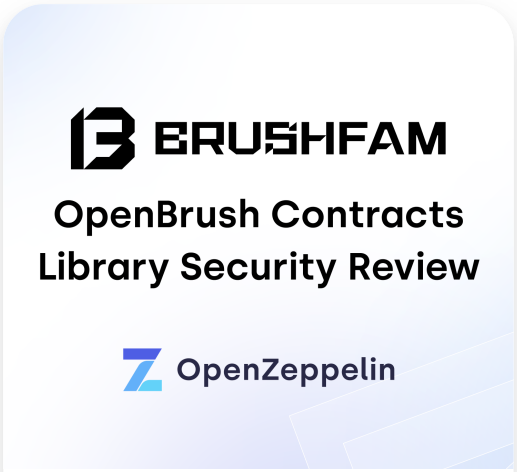
Related Posts



Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

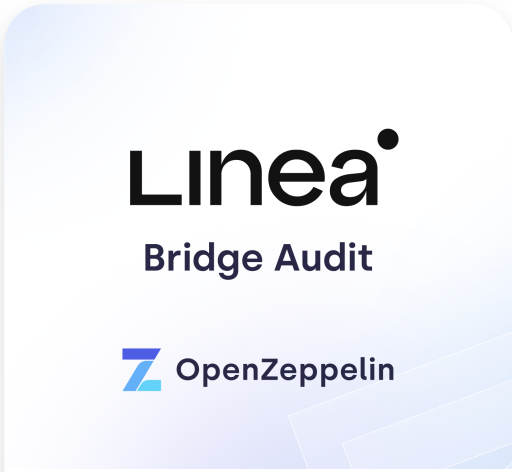
Security Audits



OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits



Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

OpenZeppelin