OpenZeppelin | security

Batch Token Bridge Audit

May 17, 2024

Table of Contents

Summary

Type	Layer 2
Timeline	From 2024-04-26 To 2024-05-02
Languages	Solidity

Total Issues	14 (4 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	5 (1 resolved)
Notes & Additional Information	8 (2 resolved)

Scope

We audited the scroll-tech/scroll repository at commit 84f73c7.

In scope were the following files:

```
contracts/src/batch-bridge
├── BatchBridgeCodec.sol
├── L1BatchBridgeGateway.sol
└── L2BatchBridgeGateway.sol
```

System Overview

We have previously covered Scroll's bridge architecture in the following reports: 1, 2, 3, 4, 5. Below, we shall provide a concise explanation of the bridge's core architecture in addition to an extensive description of the recently implemented batch bridge functionality.

Scroll's native bridge is capable of bridging messages, ETH, and different tokens like ERC-20, ERC-721, and ERC-1155 tokens. The bridge operates bidirectionally between L1 and L2. The newly implemented batch bridge gateway is limited to unidirectionally batch bridging ETH and ERC-20 tokens from L1 to L2. A deposit made on L1 can only be distributed to the same depositor address on L2. In addition, there is no option for sending data alongside the deposit.

`L1BatchBridgeGateway` Contract

The `L1BatchBridgeGateway` contract is the entry point of the batch bridging functionality. For users to deposit ETH and ERC-20 tokens, the respective assets must be configured. To avoid congestion or protracted delays, a batch of deposits will be finalized (ready to be bridged) once either `maxTxsPerBatch` or `maxDelayPerBatch` is reached.

Finalizing a batch implies a separate transaction which sends two L1->L2 messages: the first bridges the batch assets to L2 while the second calls `finalizeBatchDeposit` to relay a hash that represents the depositors and their corresponding amounts. In contrast to other bridge gateways on L1, `L1BatchBridgeGateway` does not implement the `onDropMessage` callback.

`L2BatchBridgeGateway` Contract

The `L2BatchBridgeGateway` contract is responsible for distributing the bridged assets. It first receives the assets from the `L2ScrollMessenger` and then finalizes the batch by storing the batch's hash. Once a batch is finalized on L2, the funds can be distributed to the corresponding receivers. If any of the transfers fail, they will be accounted for and can be withdrawn and manually refunded.

`BatchBridgeCodec` Library

The `BatchBridgeCodec` library is used to encode and decode the nodes for each batch bridging. The initial node holds information about the token and the batch index. Each following node holds the depositor's address and their deposited amount.

Security Model and Trust Assumptions

Both `L1BatchBridgeGateway` and `L2BatchBridgeGateway` contracts are upgradeable. Upon deployment, it is assumed that both contracts will be initialized and configured accordingly.

When bridging a batch to L2, the system sends two consecutive messages to the `L1ScrollMessenger`. These messages are expected to arrive in order and always cost roughly the same amount of gas. Hence, it is assumed that the messenger will never drop any of them. This not only makes the process atomic but also allows for not implementing the `onDropMessage` callback.

Both `SAFE_BATCH_BRIDGE_GAS_LIMIT` and `SAFE_ETH_TRANSFER_GAS_LIMIT` constants are pre-defined values that are used to avoid out-of-gas errors in their respective function calls. It is assumed that the primary users of the batch bridge will be EOAs and multisig wallets. Should the `SAFE_ETH_TRANSFER_GAS_LIMIT` not be enough while distributing ETH to a certain recipient, the ETH will be retrieved using `withdrawFailedAmount` and manually transferred to their address.

Privileged Roles

`DEFAULT_ADMIN_ROLE`

This role will be assigned to `ScrollOwner`. On L1, this role is responsible for adding and updating the batch bridge configuration for a given token address. When setting the batch configuration, the caller should ensure that `safeBridgeGasLimit` is enough for batch bridging the respective token. Moreover, the caller should set `maxTxsPerBatch` to a reasonable value to avoid out-of-gas errors when distributing on L2. Last but not least, the

value of `maxDelayPerBatch` should also be set to a reasonable value to avoid long waiting times for the users when bridging. On L2, this role is responsible for withdrawing failed amounts while distributing and manually redistributing the assets to the corresponding depositors.

`KEEPER_ROLE`

This role is responsible for executing batch deposits on L1 and distributing deposited assets to corresponding receivers on L2. In order to do so, the `KEEPER_ROLE` will have to be incentivized to spend gas, or be managed by the Scroll team.

Medium Severity

M-01 Malicious Actor Can Steal Deposits of Tokens With Sender Hooks or Cause Lock Of Funds

The `depositERC20` function can be used to deposit ERC-20 tokens. It first transfers the `_msgSender()`'s tokens to the contract and then calls the `_deposit` function. If the token implements sender hooks, a malicious attacker can leverage this to trick the smart contract into believing that they have deposited more funds than they actually had. Following are the steps to achieve this:

1. A malicious contract calls the `depositERC20` function for a token with sender hooks and deposits an amount of 50 tokens.
2. During the token transfer, the caller reenters the `depositERC20` function and deposits 50 tokens again.
3. The reentrancy check will not trigger as none of the calls has yet reached the `_deposit` function.
4. The second execution increases the user's balance by 50 and toggles the reentrancy guard on and then off.
5. The first execution increases the user's balance by 100, resulting in a total deposited balance of 150 whereas only 100 tokens made it to the contract.

There are two effects of this: if the smart contract contains enough tokens for sending the inflated amount of funds to L2, the batch deposit can be finalized on L2 and 150 tokens will be distributed to the malicious attacker. Alternatively, if the smart contract does not contain enough tokens, the `executeBatchDeposit` call would fail, without possibility of withdrawal and hence locking the funds.

Consider moving the reentrancy guard to both the `depositERC20` and `depositETH` functions in order to prevent any reentrancy into these functions.

Update: *Resolved in pull request #1334 at commit 3d08e40.*

Low Severity

L-01 Failed Funds Can Be Locked Inside `L2BatchBridgeGateway`

After a batch deposit has been finalized on L2, the `KEEPER_ROLE` can distribute the funds. If distributing to a party fails, the failed amounts are accounted for in order to rescue them later through the `withdrawFailedAmount` function. Note that the `_transferToken` function does not revert on failure but returns a `success` value which is not checked. As such, if the ETH or token transfer to the receiver fails, the `failedAmount[token]` is still set to 0, locking the funds.

Consider checking the success value of `_transferTokens` and reverting if the transfer fails.

Update: *Acknowledged, not resolved. The Scroll team stated:*

> *Acknowledged. Not a priority.*

L-02 Tokens Can Get Stuck While Finalizing Deposit

In the `L1BatchBridgeGateway` contract, `IL1ERC20Gateway.getL2ERC20Address` retrieves the corresponding `l2Token` address of the `l1Token`. In the `L2BatchBridgeGateway` contract, a `tokenMapping` is used to set and get the corresponding `l1Token` in order to minimize gas cost, as opposed to using the `IL2ERC20Gateway.getL1ERC20Address` function. The token mapping is used by the `distribute` function, which needs to know the corresponding L1 address of the token to distribute.

In the `L2BatchBridgeGateway` contract, the `finalizeBatchDeposit` function relies on the messenger to accurately map the `l2Token` address to the associated `l1Token` when a batch deposit of a token is executed and finalized for the first time. The function will revert in a subsequent call if the `l1Token` differs from the `storedL1Token`.

In the `L2CustomERC20Gateway` contract, the token mapping from L2 to L1 is updatable. In a scenario where the tokens being bridged are handled by the `L2CustomERC20Gateway`, a change of address of the token on L1 cannot be replicated in the `L2BatchBridgeGateway`. When batch bridging the new `l1Token` with the changed address to L2, the

`finalizeBatchDeposit` function will retrieve the old `l1Token` address from its mapping. Ultimately, the function will revert since the updated address of the `l1Token` will not match the address that is recorded in the mapping.

Consider removing the duplicated token mapping and supplying the `l1Token` as a parameter to the `distribute` function. Alternatively, consider either leveraging the `IL2ERC20Gateway.getL1ERC20Address` function or providing a function that enables the `L2BatchBridgeGateway` contract's token mapping to be updated.

Update: *Acknowledged, not resolved. The Scroll team stated:*

> *Acknowledged. Not a priority at the moment.*

L-03 Multiple Usages of Obsolete `safeApprove`

In the `executeBatchDeposit` function of `L1BatchBridgeGateway`, the function `safeApprove` is used to approve an amount of 0 and once again to approve the desired value. While this ensures compatibility with tokens that require the approval to be set to zero before setting it to a non-zero value(such as USDT), it hinders readability and is not the best practice. Furthermore, `safeApprove` will not be supported in future OpenZeppelin contracts.

Consider implementing best practices by using the `SafeERC20` contract's `forceApprove` function.

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

> *Acknowledged. Not a priority. We will document for better readability and update to `forceApprover` later.*

L-04 Missing Event Emission After Configuration Change

The `setBatchConfig` function is used to add or update the batch bridge config for a given token. However, this function does not emit any event.

Consider emitting an event to be able to efficiently track configuration changes off-chain.

Update: *Acknowledged, not resolved. The Scroll team stated:*

> *Acknowledged. Not a priority due to gas.*

L-05 Gas Inefficiencies

Across the codebase, there are some instances in which the code can be refactored to be more gas efficient:

- The `newConfig` parameter of the `setBatchConfig` function can be made read-only. Consider changing its location from `memory` to `calldata` to save gas.
- In order to improve code intentionality and reduce the gas cost in case of a revert, consider switching the order of the following instructions to prioritize the `if` statement.
- The `feeVault` state variable of the `L1BatchBridgeGateway` contract is not changeable, consider declaring it as `immutable`.

Update: *Resolved in pull request #1334 at commit b7cc5c2.*

Notes & Additional Information

N-01 Unused Event

In the `L2BatchBridgeGateway` contract, the `UpdateTokenMapping` event is unused.

To improve the overall clarity, intentionality, and readability of the codebase, consider removing it.

Update: *Acknowledged, not resolved. The Scroll team stated:*

> *Acknowledged. Not a priority.*

N-02 Unnecessary Usage of Upgradeable Interfaces

Using upgradeable interfaces does not provide significant benefits and can introduce unnecessary complexity to the codebase. Throughout the codebase, there are a few instances where upgradeable interfaces are being used:

- `SafeERC20Upgradeable` and `IERC20Upgradeable` in the `L1BatchBridgeGateway` contract

- `IERC20Upgradeable` in the `L2BatchBridgeGateway` contract

Moreover, upgradeable interfaces are no longer part of the newer releases of the OpenZeppelin Contracts Upgradable library.

Consider switching to non-upgradeable interfaces and libraries for better code compatibility.

Update: *Acknowledged, not resolved. The Scroll team stated:*

> *Acknowledged. Not a priority.*

N-03 Typos in Comments

In lines 86, 169, and 187 of `L1BatchBridgeGateway.sol`, consider replacing `L2BatchDepositGateway` by `L2BatchBridgeGateway` and `L1BatchDepositGateway` by `L1BatchBridgeGateway`.

Update: *Resolved in pull request #1334 at commit 7a21e29.*

N-04 Naming Suggestions

In the `L1BatchBridgeGateway` contract, consider replacing the `tokens` variable name with `tokenStates` to favor explicitness and readability. In addition, consider renaming the `pending` struct member to `pendingAmount`.

Update: *Acknowledged, not resolved. The Scroll team stated:*

> *Acknowledged. Not a priority.*

N-05 Unused Import

The `AddressUpgradeable` import inside the `L1BatchBridgeGateway` is unused. Consider removing it.

Update: *Resolved in pull request #1334 at commit 900ed4f.*

N-06 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, there are contracts that do not have a security contact:

- The `BatchBridgeCodec` library
- The `L1BatchBridgeGateway` contract
- The `L2BatchBridgeGateway` contract

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Acknowledged, not resolved. The Scroll team stated:*

> *Acknowledged. Not a priority.*

N-07 Overly Permissive Function Visibility

The `_deposit` and `_tryFinalizeCurrentBatch` functions in the `L1BatchBridgeGateway` contract have unnecessarily permissive visibility.

To better convey their intended use, consider changing their visibility from `internal` to `private`.

Update: *Acknowledged, not resolved. The Scroll team stated:*

> *Acknowledged. Not a priority.*

N-08 Missing Docstrings

Consider adding docstrings to the `BatchBridgeCodec` library. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Acknowledged, not resolved. The Scroll team stated:*

> *Acknowledged. Not a priority.*

Conclusion

The newly implemented batch bridge gateway offers users an interface to batch bridge ETH or whitelisted ERC-20 tokens.

The codebase is well-written, straightforward to follow, and well-documented. Only one medium-severity issue and some low-severity issues were discovered, while recommendations have been made to improve the overall quality of the codebase.

The Scroll team was very responsive throughout the engagement and answered all our questions. The test suite was easy to set up and had great branch coverage.