OpenZeppelin | security

EIP-4844 Support Audit

April 15, 2024

Table of Contents

Table of Contents 2

Summary 3

Scope 4

System Overview 5

ScrollChain Contract 5

MultipleVersionRollupVerifier Contract 5

BatchHeaderV1Codec and ChunkCodecV1 Libraries 5

Security Model and Trust Assumptions 6

Medium Severity 7

M-01 Batch Commitments Can Make Use of Arbitrary Library 7

Low Severity 8

L-01 Unchecked Blob-Proof Parameter 8

L-02 Incomplete Docstrings 8

L-03 Missing Docstrings 9

Notes & Additional Information 10

N-01 Unused Named Return Variables 10

N-02 State Variable Visibility Not Explicitly Declared 10

N-03 Unused Function With Internal Visibility 11

N-04 Lack of Security Contact 11

N-05 Lack of Indexed Event Parameter 12

N-06 Misleading Comments 12

Client Reported 12

CR-01 Incorrect Calculation of Non-Skipped L1 Messages 12

Conclusion 14

Summary

Type	L2
Timeline	From 2024-03-25 To 2024-04-09
Languages	Solidity

Total Issues	11 (4 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (0 resolved)
Low Severity Issues	3 (2 resolved)
Notes & Additional Information	6 (1 resolved, 1 partially resolved)
Client Reported Issues	1 (1 resolved)

Scope

We audited the changes to the scroll-tech/scroll repository between head commit 8bd4277 and base commit 02415a6. Newly introduced contracts were fully audited, whereas for the modified contracts, only the diff between the previous and new versions was audited.

In scope were the following modified files:

```
contracts/src
├── L1/rollup
│   ├── IScrollChain.sol
│   ├── MultipleVersionRollupVerifier.sol
│   └── ScrollChain.sol
└── libraries
    ├── verifier/IRollupVerifier.sol
    └── codec
        ├── BatchHeaderV0Codec.sol
        └── ChunkCodecV0.sol
```

In addition to the newly added files:

```
contracts/src/libraries/codec
    ├── BatchHeaderV1Codec.sol
    └── ChunkCodecV1.sol
```

System Overview

The system architecture is described in our previous audit reports (1, 2). Here we only describe the relevant changes.

This system upgrade has been executed to support proto-danksharding and utilizing blob-carrying transactions defined in EIP-4844. With proto-danksharding, the protocol can use less expensive L1 storage to handle L2 transactions. This makes it possible to compress more L2 transactions into each batch, lowering the transaction costs. In addition to processing L2 transactions via blob-carrying transactions, the protocol still provides the processing of L2 transactions via calldata.

This upgrade mostly modifies the `ScrollChain` L1 rollup contract. Two additional codec libraries were introduced to accommodate those modifications. Below is an explanation of the main changes in the existing contracts, followed by the newly added libraries.

`ScrollChain` Contract

The `ScrollChain` contract upgrade now allows the `commitBatch` function to accept new batches of version 1 in addition to version 0 while rejecting other versions. Furthermore, a new `finalizeBatchWithProof4844` function has been introduced to finalize committed batches with blob data on L1.

`MultipleVersionRollupVerifier` Contract

An additional function was added to allow for the verification of different versions of aggregated ZK proofs.

`BatchHeaderV1Codec` and `ChunkCodecV1` Libraries

The `BatchHeaderV1Codec` and `ChunkCodecV1` libraries are nearly identical copies of the `BatchHeaderV0Codec` and `ChunkCodecV0` (previously `ChunkCodec`) libraries, with minor adjustments made to facilitate the encoding and decoding processes specific to each data

type. The modifications in both V1 libraries were made to accommodate the inclusion of the `blobVersionedHash` in `BatchHeaderV1Codec`'s `BatchHeader` structure as well as the removal of `l2Transactions` from `ChunkCodecV1`'s `Chunk` data structure.

Security Model and Trust Assumptions

The `ScrollChain` contract still has an `initialize` function defined, which sets the `maxNumTxInChunk` variable (`__verifier` and `__messageQueue` have been deprecated). In case the admin wants to change this variable during an upgrade (for instance, by calling the `upgradeToAndCall` function in the proxy), calling `initialize` will revert since the `_initialized` flag in the `Initialize` contract has already been set in previous deployments. If updating this variable is desired when upgrading, the `updateMaxNumTxInChunk` function should be called instead.

Medium Severity

M-01 Batch Commitments Can Make Use of Arbitrary Library

In the `commitBatch` function of the `ScrollChain` contract, the `_version` parameter is used to define whether the version of the batch to commit is 0 or 1, as any other values will cause the `commitBatch` function to revert. If the `_version` is 0, the `_commitChunksV0` function, as well as the `BatchHeaderV0Codec` library, will be used to handle the data. Otherwise, if the `_version` is 1, the function will use the `_commitChunksV1` function, as well as the `BatchHeaderV1Codec` library.

However, the sequencer can arbitrarily define the `_version` value. This means a version 0 batch commitment can be forced to follow a version 1 commitment path and vice versa. For instance, if a version 1 batch was committed, but the `_version` parameter is set to 0, the `commitBatch` function will gracefully pass without throwing any error.

Note that this scenario has a low likelihood since, at the time of this audit, the `commitBatch` function is guarded by the `OnlySequencer` modifier, which allows access only to the Scroll relayer EOAs. However, the severity of this issue could increase if additional parties are granted the sequencer role in the future.

Consider validating the `_version` parameter to match the version of the committed batch.

Update: *Acknowledged, will resolve. The Scroll team added PR 1264 at commit c03cdad explaining the rationale of addressing this potential risk in the future:*

> *We initially excluded the KZG commitment by assuming lack of presence of malicious Sequencer entities that collude with malicious Provers in the current threat model. Upon considering such a scenario (which is ruled out at present, but could eventually be possible in a decentralized setting), and as per cryptographic hygiene, we decided to include the KZG commitment (in the form of the blob's versioned hash, i.e. a hash of the commitment) while computing the Fiat-Shamir challenge. Since the blob's versioned hash is accepted as private witness to our circuits, we also include it in the preimage of the batch's public input hash (the public instance to our circuits).*

Low Severity

L-01 Unchecked Blob-Proof Parameter

In the `finalizeBatchWithProof4844` function of the `ScrollChain` contract, the `_blobDataProof` parameter should have a fixed length of 160 bytes, according to the function's documentation.

However, this length is not being checked, opening the possibility of injecting arbitrary bytes. While the likelihood of this scenario is low, it could introduce an unforeseen vulnerability if the client executing the `finalizeBatchWithProof4844` function contains a bug in its implementation of the point evaluation precompile.

Consider reverting when the length of `_blobDataProof` does not match its specifications.

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

> *Acknowledged, not resolved. The length is already checked by `ErrorCallPointEvaluationPrecompileFailed`. In the case of a wrong precompile implementation, such a vulnerability in L1 clients is not in the scope of this audit, since they would lead to major issues (erroneous hard fork) on L1, so they wouldn't just affect Scroll.*

L-02 Incomplete Docstrings

Throughout the codebase, there are several instances of incomplete docstrings.

- The `lastFinalizedBatchIndex`, `committedBatches`, `finalizedStateRoots`, `withdrawRoots`, and `isBatchFinalized` functions in `IScrollChain.sol` explain what they return in the `@notice` tag. However, this should be specified under the `@return` tag.
- In the `legacyVerifiersLength` function in `MultipleVersionRollupVerifier.sol`, the `_version` parameter and the return value are not documented.
- In the `getVerifier` function in `MultipleVersionRollupVerifier.sol`, the return value is not documented.

- In the `updateVerifier` function in `MultipleVersionRollupVerifier.sol`, the `_version` parameter is not documented.
- In the `importGenesisBatch` function in `ScrollChain.sol`, the `_batchHeader` and `_stateRoot` parameters are not documented.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #1256 at commit 5425ce7.*

L-03 Missing Docstrings

Throughout the codebase, there are several parts that do not have docstrings.

- The `IRollupVerifier` interface in `IRollupVerifier.sol`
- The `IScrollChain` interface in `IScrollChain.sol`
- The `MultipleVersionRollupVerifier` contract in `MultipleVersionRollupVerifier.sol`

Note that, for example, the `ScrollChain` contract can inherit the docstrings from the `IScrollChain` interface using the `@inheritdoc` tag.

Consider thoroughly documenting all contracts, interfaces, events, and functions 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: *Resolved in pull request #1256 at commit 5425ce7.*

Notes & Additional Information

N-01 Unused Named Return Variables

Named return variables are a way to declare variables that are meant to be used within a function body for the purpose of being returned as the function's output. They are an alternative to explicit in-line `return` statements.

Within `ChunkCodecV1.sol`, there are unused named return variables. For instance:

- The `_numBlocks` return variable in the `getNumBlocks` function
- The `_numTransactions` return variable in the `getNumTransactions` function
- The `_numL1Messages` return variable in the `getNumL1Messages` function

Consider either using or removing any unused named return variables.

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

> *Acknowledged, not resolved. This is not a priority. The naming convention is kept the same between versions for more readability in further code review.*

N-02 State Variable Visibility Not Explicitly Declared

Throughout the codebase, there are state variables that lack an explicitly declared visibility:

- The `scrollChain` state variable in `MultipleVersionRollupVerifier.sol`
- The `POINT_EVALUATION_PRECOMPILE_ADDR` state variable in `ScrollChain.sol`
- The `BLS_MODULUS` state variable in `ScrollChain.sol`

For clarity, consider always explicitly declaring the visibility of variables, even when the default visibility matches the intended visibility.

Update: *Resolved in pull request #1256 at commit 5425ce7.*

N-03 Unused Function With Internal Visibility

In both `BatchHeaderV0Codec` and `BatchHeaderV1Codec` libraries, the internal `getSkippedBitmap` function [1] [2] is not being used.

Consider removing any currently unused functions to improve the codebase's overall clarity, intentionality, and readability.

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

> *Acknowledged. Not fixed. This is not a priority.*

N-04 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 proves 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. Additionally, if the contract incorporates third-party libraries and a bug surfaces in these, it becomes easier for the maintainers of those libraries to make contact with the appropriate person about the problem and provide mitigation instructions.

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

- The `BatchHeaderV0Codec` library
- The `BatchHeaderV1Codec` library
- The `ChunkCodecV0` library
- The `ChunkCodecV1` library
- The `IRollupVerifier` interface
- The `IScrollChain` interface
- The `MultipleVersionRollupVerifier` contract
- The `ScrollChain` contract

Consider adding a NatSpec comment containing a security contact on top of the contracts 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 fixed. This is not a priority.*

N-05 Lack of Indexed Event Parameter

Consider indexing the `version` and `startBatchIndex` parameters in the `UpdateVerifier` event of the `MultipleVersionRollupVerifier` contract to enhance the ability of off-chain services to search and filter by version and batch interval.

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

> *Acknowledged. Not fixed. This is not a priority.*

N-06 Misleading Comments

The following misleading and inconsistent comments have been identified in the codebase:

- In line 49 of `ChunkCodecV1.sol`, "should contain" should be "should be equal".
- In line 61 of `MultipleVersionRollupVerifier.sol`, "lastest" should be "latest" or "last".

Consider revising the comments to improve consistency and more accurately reflect the implemented logic.

Update: *Partially resolved in pull request #1256 at commit 5425ce7. The fix did not address the first bullet point.*

Client Reported

CR-01 Incorrect Calculation of Non-Skipped L1 Messages

In the `_commitChunkV1` function, the `_totalTransactionsInChunk` is designated to hold the number of actual transactions in one chunk. To calculate the final value of `_totalTransactionsInChunk`, the number of non-skipped L1 messages is added to the number of L2 transactions.

To calculate the number of non-skipped L1 messages, on line 880, the value of the subtraction of `startPtr` from `dataPtr` is added to the result. The values of `startPtr` and `dataPtr` are computed as follows:

1. First, `startPtr` is set to `dataPtr`.
2. Then, the L1 message hashes are loaded to set the new value of `dataPtr`. Within the `_loadL1MessageHashes` function, the pointer value is increased by 32 bytes for every non-skipped L1 message.

However, the subtraction on line 880 is not taking into account the 32 bytes per message, resulting in a larger number of `_totalTransactionsInChunk`. This can cause the commit function to either fail for exceeding `maxNumTxInChunk` or return a wrong data hash if `_totalTransactionsInChunk` is still smaller than `maxNumTxInChunk`.

Update: *Resolved in pull request #1232 at commit cbb65d7.*

Conclusion

The system upgrade introduces the use of blob-carrying transactions to support EIP-4844. This will allow for cheaper transactions on L1 as well as L2. The added contracts and modifications set the foundation for future data availability solutions.

The codebase is well-written, very straightforward to follow, and well-documented. The Scroll team was very responsive throughout the engagement, answered all our questions, and provided us with the needed documentation and technical explanations regarding their test suite setup.

Appendix

Testing Coverage Recommendations

The audit revealed certain testing-related concerns within the current system. The following is an overview of the overall state of testing regarding this codebase, alongside recommendations to improve the system's soundness.

While not a specific vulnerability, insufficient testing implies a high probability of additional missed vulnerabilities and bugs. It also exacerbates multiple interrelated risk factors in a complex codebase with novel functionality. This includes a lack of full implicit specification of the functionality and the expected behaviors that tests normally provide, which increases the chances that issues will be missed. It also requires more effort to establish basic correctness and reduces the effort spent exploring edge cases, thereby increasing the chances of missing complex issues.

This system upgrade relies primarily on the EIP-4844, which introduces a new precompile contract, a new `OPCODE`, and a new type of blob transaction. All these innovations have yet to be battle-tested.

To address these issues, we encourage extending the tests to increase coverage to 95% - 100%. Crucially, the test suite should cover the newly implemented version-dependent branch when committing new chunks, as well as the new finalization method implemented and the underlying libraries that help handle the corresponding chunks and batches.

However, at the time of this audit, there is a lack of support from testing tools to handle these blob-carrying transactions, which pushes the testing ability to its limits.

Monitoring Recommendations

While audits help in identifying potential security risks, the Scroll team is encouraged to also incorporate automated monitoring of on-chain contract activity into their operations. Ongoing monitoring of deployed contracts helps in identifying potential threats and issues affecting the production environment. The following is a list of actions that are recommended to be monitored:

- Monitor L2 to L1 commitment and finalization transactions based on blob-carrying transactions to ensure blocks are being properly submitted to L1.
- Monitor the operating gas costs of transactions for committing and verifying. Since the EIP-4844 has been recently added and has yet to be battle-tested, an attack to increase gas costs may occur.
- Monitor that the previously deprecated verifiers cannot be used in a proof process.
- Monitor if a chunk not meant to pass the commitment passes in the wrong method, e.g., a chunk of version 0 passes through version 1 verification methods or vice versa.
- Monitor that no other types of batches can be submitted.

General Recommendations

Given the recent implementation of the EIP-4844 on Ethereum (L1) and the absence of support for blob-carrying transactions from testing tools, it is advised to conduct live testing on a testnet first, alongside a bounty program to identify potential vulnerabilities.

Meanwhile, prepare draft scenarios for when the testing tools add support to handle these blob-carrying transactions effectively. It is important to inform users about this and advise against using real money until the system is thoroughly tested and deemed secure.