OpenZeppelin | security

# Scroll Phase 2 Audit


**July 21, 2023**

This security assessment was prepared by OpenZeppelin.

# Table of Contents

# Summary

| | | | |
|---|---|---|---|
| Type | zkEVM-based zkRollup, Bridge & Rollup | Total Issues | 22 (11 resolved) |
| Timeline | From 2023-06-05<br>To 2023-06-30 | Critical Severity Issues | 0 (0 resolved) |
| Languages | Solidity | High Severity Issues | 1 (1 resolved) |
| | | Medium Severity Issues | 5 (4 resolved) |
| | | Low Severity Issues | 7 (6 resolved) |
| | | Notes & Additional Information | 9 (0 resolved) |

# Scope

We audited the `scroll-tech/scroll` repository at the `3bc8a3f` commit.

```
contracts
└── src
    ├── L1
    │   ├── gateways
    │   │   ├── IL1ERC1155Gateway.sol
    │   │   ├── IL1ERC20Gateway.sol
    │   │   ├── IL1ERC721Gateway.sol
    │   │   ├── IL1ETHGateway.sol
    │   │   └── IL1GatewayRouter.sol
    ├── L2
    │   ├── gateways
    │   │   ├── L2CustomERC20Gateway.sol
    │   │   ├── L2ERC1155Gateway.sol
    │   │   ├── L2ERC20Gateway.sol
    │   │   ├── L2ERC721Gateway.sol
    │   │   ├── L2ETHGateway.sol
    │   │   ├── L2GatewayRouter.sol
    │   │   ├── L2StandardERC20Gateway.sol
    │   │   └── L2WETHGateway.sol
    │   ├── predeploys
    │   │   ├── IL1BlockContainer.sol
    │   │   ├── IL1GasPriceOracle.sol
    │   │   ├── L1BlockContainer.sol
    │   │   ├── L1GasPriceOracle.sol
    │   │   ├── L2MessageQueue.sol
    │   │   ├── L2TxFeeVault.sol
    │   │   ├── WETH9.sol
    │   │   └── Whitelist.sol
    │   ├── IL2ScrollMessenger.sol
    │   └── L2ScrollMessenger.sol
    └── libraries
        ├── common
        │   ├── AppendOnlyMerkleTree.sol
        │   └── OwnableBase.sol
        ├── constants
        │   └── ScrollPredeploy.sol
        ├── token
        │   ├── IScrollERC1155.sol
        │   └── IScrollERC721.sol
        └── FeeVault.sol
```

Scroll's architecture and code structure draw inspiration from other Layer 2 solutions such as Arbitrum and Optimism, particularly in the design of their gateways, predeploys, and

messaging contracts. Notably, a lot of code structure from Arbitrum's gateways and the `AddressAliasHelper.sol` contract is reused with minor modifications.

The primary focus of this audit was on the Scroll L2 Bridge Contracts. In this audit, we aimed to verify the correctness and security of the contracts, focusing on aspects like block finalization, message passing, and the process of depositing and withdrawing.

***Update:*** *It is important to note that the* `develop` *branch changed the codebase between the audit's start and the fix review. Hence, we only reviewed the fixes in their respective context and cannot guarantee other implications that were introduced in the meantime.*

# 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 zkEVM and ConsenSys' Linea.

This report presents our findings and recommendations for 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.

## Architecture

The system's architecture is split into three main components:

- Scroll Node: This constructs Layer 2 (L2) blocks from user transactions, commits these transactions to the Ethereum base layer, and handles message passing between L1 and L2.
- Roller Network: This component is responsible for generating the zkEVM validity proofs, which are used to prove that transactions are executed correctly.
- Rollup and Gateway contracts: These contracts provide data availability for Scroll transactions, verify zkEVM validity proofs, and allow users to move assets between Ethereum and Scroll. Users can pass arbitrary messages between L1 and L2, and can bridge assets in either direction due to the Gateway contracts.

## Rollup and Bridging

The Scroll system connects to Ethereum primarily through its Rollup and Messenger contracts. The Rollup contract is responsible for receiving L2 state roots and blocks from the Sequencer, and finalizing blocks on Scroll once their validity is established.

The Messenger Contracts enable users to pass arbitrary messages between L1 and L2, as well as bridge assets in both directions. The gateway contracts make use of the messages to operate on the appropriate layer.

The standard ERC-20 token gateway automatically deploys tokens on L2, using a standardized token implementation. There is also a custom token gateway which enables users to deploy their L1 token on L2, in more sophisticated cases. In such scenarios, the Scroll team would need to manually set the mapping for these tokens. This could potentially lead to double-minting on L2 (two tokens being created, one through each method). To prevent such a scenario, it is recommended to use the GatewayRouter, which will route the token to the correct gateway. These custom gateways are also required for ERC-721 and ERC-1155 tokens, which currently do not have a standard gateway provided. However, the GatewayRouter does not currently support ERC-721 or ERC-1155 custom gateways.

## State of Refunds

When communicating/bridging from L1 to L2, values are handled in two ways on the L1 side:

1. If a token is bridged, the token will be transferred into the gateway contract. If ETH is transferred, the value is kept in the L1 messenger contract. In the case of WETH, the assets will be first unwrapped to ETH and forwarded to the L1 messenger contract.
2. The user has to specify a gas limit that will be used for the L2 transaction. The relayer accounts for this gas limit through a fee that is deducted on the L1 call.

In the audited version of the protocol, there is no refund mechanism for (1) if the L1 initialized message is not provable (or censored) and hence not executed and removed from the L1 message queue. This means assets can potentially get stuck in the Gateway or L1 messenger contracts. Regarding (2), any excessive gas limit over the required amount is paid as an extra fee into the fee vault. It is therefore crucial for users to make their best estimations through the l2geth API.

# 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, so the sequencer has the ability to censor L2 messages and transactions. L1 to L2 messages are appended into a message queue that is checked against during finalization. The sequencer can choose to skip any message from the queue during finalization to allow 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 the message skipping mechanism post-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.
- Control over L1BlockContainer: The L1BlockContainer has an owner that can initialize the starting block hash, block height, block timestamp, block base fee, and state root.
- Control over L1GasPriceOracle: The L1GasPriceOracle has an owner that can update the gas price.
- Control over L2MessageQueue: The L2MessageQueue has an owner that can update the address of the messenger.
- Control over L2TxFeeVault: The L2TxFeeVault has an owner that can update the address of the messenger, recipient, and minimum withdrawal amount.

# Privileged Roles

Certain privileged roles within the Scroll protocol were identified during the audit. These roles possess special permissions that could potentially impact the system's operation:

- **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
  - `L2ScrollMessenger`

- **Implementation Owners:** Most contracts are also ownable. The following actions describe what the owner can do in each contract.
  - `L2ScrollMessenger`: Pause relaying of L1 to L2 messages and L2 to L1 message requests.
  - `L2{CustomERC20|ERC721|ERC1155}Gateway`: Change the token mapping of 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.
  - `ScrollMessengerBase`: Change the fee vault address which collects fees for message relaying.
- **Sequencer**: The sequencer role can interact with the `ScrollChain` contract to commit to new batches that bundle multiple L2 blocks in chunks that can then be finalized along with a proof.
- **Whitelist**: Accounts can be whitelisted to change the L2 base fee on L1 as well as the intrinsic gas parameters. They can also change the parameters used in gas calculations by the sequencer.

# High Severity

## H-01 Incorrect Storage Slot Calculations

In the `L2ScrollMessenger` contract, the functions `verifyMessageInclusionStatus` and `verifyMessageExecutionStatus` use assembly code to manually calculate the storage slot for mappings `isL1MessageSent` and `isL2MessageExecuted` respectively. However, both calculations are incorrect, as they assume `ScrollMessengerBase` uses 4 storage slots instead of 3.

This would make both functions `verifyMessageInclusionStatus` and `verifyMessageExecutionStatus` unusable, which would prevent the contract from verifying inclusion or execution status for messages. Furthermore, the function `retryMessageWithProof` relies on `verifyMessageInclusionStatus`, and therefore failed messages on L2 could not be retried.

Consider fixing the calculation and thoroughly documenting any changes to the storage layout of `ScrollMessengerBase` and `L1ScrollMessenger`. Furthermore, consider moving these hard-coded values in the assembly code into a separate "constants" file to reduce the number of failure points in the case of future changes.

***Update:*** *Resolved in pull request #558 at commit 94db3ab and in pull request #618 at commit 2395883. The assembly code was updated with the correct magic constants for storage slot lookups. The values were not placed into a separate `constants` file, but the Scroll team stated:*

> *We decided to remove the `retry` in the Layer 2 logic. The code related to the storage slot will also be deleted.*

# Medium Severity

## M-01 `L2MessageQueue` Stores Incorrect Value if not Initialized Before Appending

In `L2MessageQueue` it is expected that `initialize` is called before `appendMessage`. This initializes the `zeroHashes` array. If however, it is not, `appendMessage` can still be called. On the 5th message sent, the wrong Merkle tree will be calculated (since at that point the calculation starts using the `zeroHashes[1]` value). After that point, even if `initialize` is called, it is too late, since messages cannot be removed and the contract is not upgradeable.

Fortunately, `appendMessage` can only be called by the `messenger`, which makes this scenario unlikely.

Consider adding a safeguard so that `appendMessage` cannot be called until `initialize` is called first.

***Update:*** *Resolved in pull request #630 at commit 89814bd.*

## M-02 WETH9 Approval Can Be Front-Run

The `WETH9` contract has an `approve` function which allows the `msg.sender` (the approver) to authorize an address (the spender) to spend a determined amount of tokens on their behalf. However, if the approver wants to change the authorized amount of the spender, the approver needs to call the `approve` function again with the new amount. A malicious spender could front-run this second `approve` transaction, by using the `transferFrom` to spend all the previously authorized tokens before being authorized the amount specified in the second `approve` transaction.

This is possible because L2 transactions are ordered by the L2 node based on gas price. This means that the malicious spender could transfer the original amount, and once this second approval transaction is confirmed, the spender could also transfer the second amount as well, rather than the approver's intent, which was only the second amount that was set. This is a well-known attack that is properly documented here.

While it is possible to avoid this problem by having every approver submit an `approve` transaction with the amount set to 0 prior to submitting another `approve` transaction with the

newly desired amount, this is error-prone and gas-inefficient. Consider adding `increaseAllowance` and `decreaseAllowance` functions to atomically increase and decrease the allowance granted to the spender.

***Update:*** *Resolved in pull request #632 at commit 85850f1. The `WETH9.sol` file was removed and was instead replaced by a custom-made wrapper (`WrappedEther.sol`) that inherits from OpenZeppelin's `ERC20Permit` contract, which already has the `increaseAllowance` and `decreaseAllowance` methods implemented.*

## M-03 Lack of Storage Gaps

The contract `L2ERC20Gateway` is being inherited by multiple upgradeable contracts.

Without adding a storage gap, new storage variables cannot be added to `L2ERC20Gateway` without causing a storage collision in all the contracts that inherit from it. This would cause contracts to malfunction and compromise their functionalities.

Consider adding a gap variable to future-proof base contract storage changes and be safe against storage collisions.

***Update:*** *Resolved in pull request #618 at commit 2395883.*

## M-04 Smart Contract Wallets Cannot Withdraw WETH

The predeployed WETH9 implementation uses transfer to unwrap WETH to ETH for a `msg.sender`. If users attempt to `withdraw` funds using a Smart Wallet that has any extra logic on the `receive` method, the transaction will run out of gas and fail. Users would then need to transfer the WETH to an EOA in order to unwrap their funds.

Consider using `address.call{value: amount}("")` or the `sendValue` function of the OpenZeppelin `Address` library to provide them with enough gas to handle additional logic, as this is the recommended method to use. Thereby make sure to protect against reentrancy by following the check-effects-interactions pattern or adding a reentrancy guard.

***Update:*** *Resolved in pull request #558 at commit 0df7531 and in pull request #632 at commit 85850f1 by replacing `WETH9.sol` with `WrappedEther.sol`, which uses the `call` method.*

## M-05 Lack Of Expiration For Retrying Transactions

The `L2ScrollMessenger` contract provides a mechanism to retry failed transactions from L1 to L2 at any time. Therefore, if a user creates an L1 to L2 transaction and it fails, it can be retried indefinitely at a later time. A scenario could occur where a user erroneously sends an L1 to L2 transaction (such as transferring more L2 ETH than the user had intended), which fails at the L2 level. The user may be inclined to resend the transaction with the correct amount of L2 ETH, which would then succeed. However, because the first transaction has still persisted, a malicious recipient could retry the first transaction again with a higher gas limit, which would succeed, causing the sender to send more L2 ETH than intended.

More generally, if a transaction is sent from L1 to L2 and fails, it can be retried indefinitely at a later time. This may not align with the intention of a user. Consider adding a timeout mechanism for failed transactions, limiting the timeframe in which a failed transaction can be retried.

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

> *It is related to the refund feature, so we don't support it for now.*

# Low Severity

## L-01 Lack of Validation When Updating Maximum Failed Execution Tries

The `L2ScrollMessenger contract` provides a mechanism to limit the number of failed execution tries that a transaction may have. This limit can be changed with the function `updateMaxFailedExecutionTimes`. However, it is possible to set `maxFailedExecutionTimes` to zero, which could lead to a situation where no transactions can ever succeed, as they would immediately fail this requirement.

Consider enforcing that `maxFailedExecutionTimes` cannot be set to zero in the `updateMaxFailedExecutionTimes` function.

***Update:*** *Resolved in pull request #649 at commit 58ee807.*

## L-02 Missing Error Messages in `require` Statements

Within `WETH9.sol` there are multiple `require` statements that lack error messages. For instance:

- The `require` statement on line 48
- The `require` statement on line 80
- The `require` statement on line 83

Consider including specific, informative error messages in `require` statements to improve overall code clarity and facilitate troubleshooting whenever a requirement is not satisfied.

***Update:*** *Resolved in pull request #632 at commit 85850f1. The `WETH9.sol` file was removed and was instead replaced by a custom-made wrapper (`WrappedEther.sol`) that inherits from OpenZeppelin's `ERC20Permit` contract, which does have such `revert` messages.*

## L-03 Unsafe ABI Encoding

Throughout the codebase there are several occurrences of unsafe ABI encodings through the use of `abi.encodeWithSelector`. It is not an uncommon practice to use either `abi.encodeWithSignature` or `abi.encodeWithSelector` to generate this calldata. However, the first option is not typo-safe and the second option is not type-safe. The result is that both of these methods are error-prone and should be considered unsafe. These occurrences are outlined below:

- On line 132 of `L2CustomERC20Gateway.sol`
- On line 173 of `L2ERC1155Gateway.sol`
- On line 216 of `L2ERC1155Gateway.sol`
- On line 166 of `L2ERC721Gateway.sol`
- On line 205 of `L2ERC721Gateway.sol`
- On line 100 of `L2ETHGateway.sol`
- On line 144 of `L2StandardERC20Gateway.sol`
- On line 124 of `L2WETHGateway.sol`

Consider replacing all the occurrences of unsafe ABI encodings with `abi.encodeCall`, which checks whether the supplied values actually match the types expected by the called function and also avoids errors caused by typos. Note that `abi.encodeCall` was not introduced until Solidity `0.8.11`. However, it is recommended to use Solidity `0.8.13` or above since there was a bug detected regarding fixed-length bytes literals. Using an updated

version will remove the possibility of future errors. In order to perform this safe encoding, consider upgrading all contracts from `^0.8.0` to Solidity version `0.8.13` at a minimum, but ideally to the latest version.

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

> *This issue was also reported in the Layer 1 report (N-02 Error-Prone Call Encoding). This is not a priority at the time. It will be addressed later on.*

## L-04 Initialization Not Disabled for Implementation Contracts

Throughout the codebase, implementation contracts are used behind proxies for upgradeability. Hence, many contracts have an `initialize` function that sets up the proxy. It is a good practice to not leave implementation contracts uninitialized, hence, consider calling `_disableInitializers` of the inherited `Initializable` contract from the `constructor` to prevent initialization of the implementation contract.

***Update:** Resolved in pull request #639 at commit d1b7719. It is worth noting that the `SimpleGasOracle` contract has not been fixed. The Scroll team stated:*

> *The `SimpleGasOracle` contract is not being used anymore, we are considerting its removal at a later time.*

## L-05 Lack of Event Emissions

Throughout the codebase, several instances were identified where events should be emitted.

- When `Whitelist` is deployed, the `owner` is set by the constructor. Because `owner` is not set using `_transferOwnership`, the `OwnershipTransferred` event is not emitted.
- The `updateMessenger` function does not emit an event upon the successful update of the `L2ScrollMessenger` address.
- The `L2GatewayRouter` does not emit an event when setting `defaultERC20Gateway` in the `initialize` function.

Consider emitting the `OwnershipTransferred` event, as well as an event when the address of the `L2ScrollMessenger` gets updated on the `L2MessageQueue` contract. Furthermore,

consider emitting the `SetERC20Gateway` event when the `L2GatewayRouter` is initialized. Emitting events provides a way for external integrations to track changes being made to the contract configuration by external integrations. This is especially important for monitoring operations and proper incident response, particularly with regard to trusted entities with special privileges.

***Update:*** *Resolved in pull request #650 at commit 90cf7b7.*

## L-06 Initialization Performed Outside of Initialization Function

The `updateMessenger` function in `L2MessageQueue` can only be called before any message is appended, otherwise it will revert since `nextMessageIndex` is sequentially increased. Therefore, the `updateMessenger` logic aligns closer with the purpose of the `initialize` function, rather than a standalone function.

Consider moving the logic of `updateMessenger` into the `initialize` function with the `onlyOwner` modifier. This can be done since the predeployed contracts should exist before `L2ScrollMessenger`.

***Update:*** *Resolved in pull request #652 at commit dd9d880.*

## L-07 Block Container Does Not Enforce Whitelist

In the `L1BlockContainer` contract, the `importBlockHeader` function can be called by anyone if the whitelist address has not been initialized.

The block container contract is used to check the state root when doing an inclusion proof in the `verifyMessage{Inclusion|Execution}Status` function. Hence, an attacker can determine which messages are seen as sent or executed on L1. Although the attacker cannot relay or retry any message on L2 because they are not the L1ScrollMessenger address, they can overwrite the state root to make retry messages fail.

Consider preventing the `importBlockHeader` function to be called if the `whitelist` address is zero.

***Update:*** *Resolved in pull request #651 at commit 6959d82.*

# Notes & Additional Information

## N-01 Typos In Comments

Throughout the codebase, there are some instances of typos in comments and docstrings. Some examples are:

- line 11 of IL2ERC1155Gateway.sol `transfered` should be `transferred`.
- line 11 of IL2ERC721Gateway.sol `transfered` should be `transferred`.
- line 19 of L2ERC721Gateway.sol `transfered` should be `transferred`.
- line 27 of L2GatewayRouter.sol `addess` should be `address`.
- line 22 of L2StandardERC20Gateway.sol `transfered` should be `transferred`.
- line 20 of L2WETHGateway.sol `transfered` should be `transferred`.
- line 251 of L1BlockContainer.sol `vaules` should be `values`.

Consider updating the lines identified above. Furthermore consider applying an automated spelling and grammar checker to your codebase to identify further instances.

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

> *This is not a priority at the time. It will be addressed later on.*

## N-02 Unused Imports

Throughout the codebase the following imports are unused and could be removed:

- Import `IL1GasPriceOracle` of `L2ScrollMessenger.sol`
- Import `IERC20Upgradeable` of `L2CustomERC20Gateway.sol`
- Import `IL2ERC20Gateway` of `L2CustomERC20Gateway.sol`
- Import `IScrollGateway` of `L2CustomERC20Gateway.sol`
- Import `IERC1155Upgradeable` of `L2ERC1155Gateway.sol`
- Import `IScrollGateway` of `L2ERC1155Gateway.sol`
- Import `IERC721Upgradeable` of `L2ERC721Gateway.sol`
- Import `IScrollGateway` of `L2ERC721Gateway.sol`
- Import `IL2ScrollMessenger` of `L2GatewayRouter.sol`
- Import `IL1ETHGateway` of `L2GatewayRouter.sol`

- Import `IScrollGateway` of `L2GatewayRouter.sol`
- Import `IL2ERC20Gateway` of `L2StandardERC20Gateway.sol`
- Import `IScrollGateway` of `L2StandardERC20Gateway.sol`
- Import `IScrollGateway` of `L2WETHGateway.sol`
- Import `IL1BlockContainer` of `L1GasPriceOracle.sol`

Consider removing unused imports to improve the overall clarity and readability of the codebase.

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

> *This is not a priority at the time. It will be addressed later on.*

## N-03 Unused or Unnecessary Code

Through the codebase, there are instances of variables that are not used or code that is unnecessary:

- In ScrollGatewayBase it states that the address of router could be zero, if this contract is `GatewayRouter`. However the `L2GatewayRouter` contract doesn't inherit from `ScrollGatewayBase.` Therefore, it would be possible to remove this line and this line and move the `require` statement to `ScrollGatewayBase`.
- In `L2ScrollMessenger`, the immutable variable `gasOracle` is never used.

Consider removing unnecessary code and unused variables to have a cleaner and more readable codebase. This helps keep the code readable which can help avoid bugs and reduce the attack surface in any future development.

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

> *This is not a priority at the time. It will be addressed later on.*

## N-04 Use Custom Errors

Throughout the codebase, the code makes use of `require` statements with error strings (i.e., this line of code) to describe the reason of a reversion.

However, since Solidity version 0.8.4, custom errors provide a cleaner and more cost-effective way to explain to users why an operation failed versus using require and revert statements with custom error strings.

To improve conciseness, consistency, and obtain gas savings, consider replacing hard-coded `require` and `revert` messages with custom errors.

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

> *This is not a priority at the time. It will be addressed later on.*

## N-05 Events Split Between Contracts and Interfaces

Throughout the codebase, events are placed both in contracts and interfaces. It is using the pattern where events on authorized actions are placed in the contract while user-relevant events are placed in the interface, which harms the readability. Consider moving the events from the contracts to their respective interfaces for clarity. Furthermore, to facilitate monitoring capabilities, which rely on checking events, it is easier to compile the interfaces and obtain their ABI when they are all located in a single place.

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

> *This is not a priority at the time. It will be addressed later on.*

## N-06 Incorrect Documentation

The following instances of incorrect documentation have been identified:

- The comment on line 43 in the `L2ScrollMessenger` contract incorrectly states that the `gasOracle` address variable contains the address for the contract `L2MessageQueue`.
- The comment on line 94 in `L2ETHGateway` should say `L2GatewayRouter` instead of `L1GatewayRouter`.
- The comment on line 112 in `L2WETHGateway` should say `L2GatewayRouter` instead of `L1GatewayRouter`.
- The comment on line 28 in `L2ERC721Gateway` should say `_l2Token` instead of `_l1Token`.
- The comment on line 9 in `L2MessageQueue` contains a broken link.

- The require message on line 144 in `L2GatewayRouter` should say "no eth gateway available".

Consider resolving these instances of incorrect documentation to improve the clarity and readability of the codebase.

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

> *This is not a priority at the time. It will be addressed later on.*

## N-07 Unused Return Value

The `L2ScrollMessenger` contract calls the function `appendMessage` in `L2MessageQueue` which returns `_currentRoot`. However, the return value is not used, and this function is only used by `L2ScrollMessenger`.

Consider removing the return value of the function `appendMessage` if it is not intended to be used.

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

> *This is not a priority at the time. It will be addressed later on.*

## N-08 Gas Optimizations

Throughout the codebase there are multiple instances where gas costs can be optimized:

- In line 30 of `Whitelist`, `_accounts.length` is calculated for every iteration of the loop. Consider using a local variable to store the value of `_accounts.length` and use that in the for loop instead.
- In line 203 of `L2GatewayRouter`, `_tokens.length` is calculated for every iteration of the loop. Consider using a local variable to store the value of `_tokens.length` and use that in the for loop instead.
- In line 51 of `L2StandardERC20Gateway`, the `require` statement checks if the address of `_tokenFactory` is equal to 0. This is done after the call to `ScrollGatewayBase._initialize`. Consider moving this `require` check prior to the `_initialize` call.
- In the `retryMessageWithProof` function of `L2ScrollMessenger`, it checks to see if the number of failed times has exceeded the maximum amount. In the event of

successful execution, consider setting the storage slot of `l1MessageFailedTimes[_xDomainCalldataHash]` to 0 after for a gas refund.

- In `L2ScrollMessenger`, the `_lock_status` is a state variable. The modifier `nonReentrant` uses the value of `_ENTERED` and `_NOT_ENTERED` as 1 and 2, but the `_lock_status` value is never set to `_NOT_ENTERED` in the `initializer`. Consider setting this `_lock_status` to the value of `_NOT_ENTERED` in the `initializer` to save on gas on the call to the modifier.
- In a previous version of the code, some users had to pay a fee on L2 to relay their message to L1. The architecture has now moved to Merkle Proofs for the user to prove message inclusion without relying on the relayer. However, there is still some code left from before that isn't used anymore. Consider removing the remaining `feeVault` and several `_gasLimit` parameters on L2 to save on gas.
- During the initialization of the `L2ScrollMessenger` contract, the `xDomainMessageSender` variable is set a second time after the `_initialize` function of the base contract. Consider setting this variable only once in order to save gas.

Consider applying the above changes to improve gas consumption.

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

> *This is not a priority at the time. It will be addressed later on.*

## N-09 Naming Suggestions

To favor explicitness and readability, the following locations in the contracts may benefit from better naming:

- The interface file `IL1BlockContainer` has functions `getStateRoot` and `getBlockTimestamp` which take in a parameter named `blockHash`, while the corresponding functions in the implementation of this interface, `L1BlockContainer`, accept a parameter named `_blockHash`. Consider updating this to be consistent.
- The interface file `IL1GasOracle` has functions `getL1Fee` and `getL1GasUsed` which take in a parameter named `data`, while the corresponding functions in the implementation of this interface, `L1GasPriceOracle`, accept a parameter named `_data`. Consider updating this to be consistent.
- The contract `L1BlockContainer` contains a function named `initialize` which is a name typically reserved for an initialization function for implementation contracts that sit behind a proxy. Consider changing this name to prevent confusion as to whether or not this contract is upgradeable.

- The contract `L2MessageQueue` contains a function named `initialize` which is a name typically reserved for an initialization function for implementation contracts that sit behind a proxy. Consider changing this name to prevent confusion as to whether or not this contract is upgradeable.
- In `Whitelist`, there is a private state variable named `isWhitelisted`. Consider changing this name to `_isWhitelisted` to match the naming convention of a private variable.
- In `L2StandardERC20Gateway`, there is a private state variable named `tokenMapping`. Consider changing this name to `_tokenMapping` to match the naming convention of a private variable.

Consider renaming as suggested above to improve the consistency and readability of the codebase.

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

> *This is not a priority at the time. It will be addressed later on.*

# Recommendations

## ERC-20 Factory Design

Tokens can be bridged in a custom and standard way. For the latter, the `ScrollStandardERC20` is the default implementation that will represent the L1 token on L2. This is realized with the `Clones` library and the EIP-1167 standard. It works by deploying a minimal proxy that delegates its calls into the token implementation and that is initialized as the token instance.

These standard tokens are not upgradeable, which comes with a trade-off. On the one hand, it is more secure since the logic cannot be changed. On the other hand, it is less future-proof meaning that standards like ERC-677 - which is not a finalized EIP - might at some point be overruled by a new standard that finds mass adoption.

An alternative factory design that is future-proof would be the Beacon proxy pattern. In a similar approach the `BeaconProxy` will be the token instance, but then fetches the implementation contract to delegate into from a single `UpgradeableBeacon` contract. This allows upgrading all tokens in one transaction.

Regarding the security implications of upgradeable contracts it is crucial to have the `UpgradeableBeacon` secured through a timelock, multisig, and cold wallets.

***Update:*** *Acknowledged. The Scroll team stated:*

> *For safety concerns, we prefer the contract to be non-upgradeable.*

## ERC-165 Support

While most of the codebase is comprised of custom contracts which do not implement a specific standard, the `ScrollStandardERC20` contract is implementing the ERC-20 and ERC-677 standards. As such, it makes sense to also add ERC-165 support to enable other parties to identify its interface and the standard it implements.

***Update:*** *Acknowledged. The Scroll team stated:*

> *Makes sense, will support it if we have time.*

## Testing Coverage

Due to the complex nature of the system, we believe this audit would have benefitted from more complete testing coverage.

While insufficient testing is not necessarily a vulnerability, it implies a high probability of additional hidden vulnerabilities and bugs. Given the complexity of this codebase and the numerous interrelated risk factors, this probability is further increased. Testing provides a full implicit specification along with the expected behaviors of the codebase, which is especially important when adding novel functionalities. A lack thereof increases the chances that correctness issues will be missed. It also results in more effort to establish basic correctness and reduces the effort spent exploring edge cases, thereby increasing the chances of missing complex issues.

Moreover, the lack of repeated automated testing of the full specification increases the chances of introducing breaking changes and new vulnerabilities. This applies to both previously audited code and future changes to current code. This is particularly true in this project due to the pace, extent, and complexity of ongoing and planned changes across all parts of the stack (L1, L2, relayer, and zkEVM). Underspecified interfaces and assumptions increase the risk of subtle integration issues, which testing could reduce by enforcing an exhaustive specification.

We recommend implementing a comprehensive multi-level test suite consisting of contract-level tests with >90% coverage, per-layer deployment and integration tests that test the deployment scripts as well as the system as a whole, per-layer fork tests for planned upgrades and cross-chain full integration tests of the entire system. Crucially, the test suite should be documented in a way so that a reviewer can set up and run all these test layers independently of the development team. Some existing examples of such setups can be suggested for use as reference in a follow-up conversation. Implementing such a test suite should be a very high priority to ensure the system's robustness and reduce the risk of vulnerabilities and bugs.

***Update:*** *Acknowledged. The Scroll team stated:*

> *More tests will be added later.*

## Custom Gateway Contracts

Developers who need to use custom gateway contracts should ensure that their contracts are designed to allow for the burning of tokens and the creation of the same tokens with the same

id. This is particularly relevant for non-fungible tokens (NFTs) and other unique asset types that rely on the token id for maintaining uniqueness.

The burning of tokens on one layer (L1 or L2) and the subsequent creation of the same tokens on the other layer is a crucial feature for token bridging in layer 2 solutions like Scroll. However, most NFT contracts are designed to create new tokens with a sequential counter, which makes them incompatible with this requirement.

Ensure providing adequate documentation, and if possible code examples, so that developers in the Scroll ecosystem can properly implement these requirements.

***Update:*** *Acknowledged. The Scroll team stated:*

> *We have refactored the interface required for ERC721/ERC1155 gateway. And also more examples will be added. Hopefully, it will be enough for developers.*

# 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.

### Governance

**Critical:** There are several important contracts that use the Proxy Pattern and can be arbitrarily upgraded by the proxy owner. Consider monitoring for upgrade events on at least the following contracts:

- `L2ScrollMessenger`
- Gateway contracts

### Access Control

**Critical:** `Ownable` allows implementing access control to prevent unauthorized parties from making unintended changes, but it is important to monitor for events where the owner changes. Consider monitoring for the `OwnershipTransferred` event on all ownable

contracts such as `L1BlockContainer`, `L1GasPriceOracle`, `L2MessageQueue`, `L2TxFeeVault`, and `Whitelist`.

#### Technical

**Medium:** The `L2ScrollMessenger` contract includes a mechanism for pausing in case of an incident. Consider monitoring for the `Paused` since an unexpected pause may cause a disruption in the system.

#### Financial

**Medium:** Consider monitoring the size, cadence and token type of bridge transfers during normal operations to establish a baseline of healthy properties. Any large deviation, such as an unexpectedly large withdrawal, may indicate unusual behavior of the contracts or an ongoing attack.

***Update:*** *Acknowledged. The Scroll team stated:*

> *It is on our roadmap. Will have one before the mainnet launch.*

# Conclusion

Over the course of this 3.5-week audit, the core of the Scroll protocol smart contracts were reviewed. The codebase of the Scroll protocol is well-documented and organized, making it easier to reason about its functionality and potential vulnerabilities. The audit was conducted smoothly, and the Scroll team provided invaluable insights, which facilitated the understanding and review of the protocol.

However, during the audit, several issues were identified that cast some doubts on the protocol's readiness for production. In particular, the issues identified suggest that more exhaustive and rigorous testing of the protocol is required. The lack of refund mechanisms, an important feature for safeguarding user assets when bridging assets, also points towards the need for further development before the protocol can be considered ready for live deployment.

It is strongly recommended for the Scroll team to expand their test suite to ensure all functionalities and edge cases are thoroughly tested. Following the implementation of the refund features, another round of auditing should be conducted to ensure that the protocol is secure and reliable.