zkSync | Layer 1 Diff Audit

(February 2023)

OPENZEPPELIN SECURITY | MARCH 8, 2023

Security Audits

March 8, 2023

This security assessment was prepared by **OpenZeppelin**.

Table of Contents

Summary

Type

Rollups

Timeline

From 2023-02-06

To 2023-02-17

Languages

Solidity

Total Issues

0 (0 resolved)

High Severity Issues

2 (1 resolved)

Medium Severity Issues

6 (3 resolved, 3 partially resolved)

Low Severity Issues

5 (2 resolved)

Notes & Additional Information

9 (4 resolved)

Scope

We audited changes to the matter-labs/zksync-2-contracts repository at the 3f345ce commit, and conducted full audits of `AddressAliasHelper.sol` and `L2ContractHelper.sol`.

In scope were the following contracts:

```
contracts
├── zksync2-contracts
│   ├── Allowlist.sol
│   ├── IAllowList.sol
│   ├── Config.sol
│   ├── Storage.sol
│   ├── Mailbox.sol
│   ├── IMailbox.sol
│   ├── Governance.sol
│   ├── IGovernance.sol
│   ├── Getters.sol
```

```
|   ├── L2ContractHelper.sol
```

System Overview

zkSync Era is a permissionless general-purpose ZK rollup that operates similarly to L1 blockchains and sidechains. It enables deployment and interaction with Turing-complete smart contracts, which are executed on a specialized virtual machine called the zkEVM. It's important to note that the bytecode for the zkEVM is different from the L1 EVM, but there are Solidity and Vyper compilers available for developing L2 smart contracts. A strength of zkSync Era is its protocol for passing messages between L1 and L2. This provides a standard way for users to interact with smart contracts across both layers.

Privileged Roles and Trust Assumptions

The governor currently possesses a superpower to upgrade contracts instantaneously and indefinitely freeze them, but this power is only temporary. In the future, the upgrade process will be subject to time locks and the duration of freezing will be limited, thus preventing the governor from exploiting their power.

Testing Coverage Recommendations

Several concerns regarding the testing of the current system were identified during the audit. While there is a separate finding highlighting these concerns for the specific contracts within the scope of this audit, it is important to note the overall systematic risks here.

Insufficient testing, while not a specific vulnerability, implies the high probability of additional unfound vulnerabilities and bugs. It also exacerbates multiple interrelated risk factors in a complex code base with novel functionality. This includes a lack of full implicit specification of the functionality and exact expected behaviors that tests normally provide, which increases the chances that correctness 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.

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

bootloader and system contracts, compiler and VM). Underspecified interfaces and assumptions increase the risk of subtle integration issues, which testing could reduce by enforcing an exhaustive specification.

To address these issues, we recommend implementing a comprehensive multi-level test suite before the next expected audits. Such a test suite should comprise contract-level tests with 95%-100% 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 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.

High Severity

Deposit cap not updated on withdrawals

User deposits are capped by increasing the `totalDepositedAmountPerUser` counter. However, the counter is not decreased by withdrawals.

As the counter can only be increased, after sufficient usage, all withdrawing users will be locked out of depositing again. This can possibly happen right after the first deposit and withdrawal, if the initial amount is larger than half the cap.

Consider decreasing the per-user cap during withdrawals to allow users to return to the rollup.

Update: *Acknowledged, not resolved. The Matter Labs team stated:*

> *The deposit limitations are only enabled in Fair Onboarding Alpha, while only approved partners may deposit funds. This will be removed at Full Launch Alpha, so we treat this issue as an accepted risk.*

ETH withdrawal within allowed limit could fail

logic flaw that could cause an ETH withdrawal within the limit to fail.

When withdrawal validations occur within the same 1-day window, the function checks the limit in line 215 of Mailbox, as follows:

```
require(
                _amount + s.withdrawnAmountInWindow <= (limitData.w
                "w4"
            );
```

However, `address(this).balance` has already changed due to the previous withdrawal within the same day, causing the allowance to be less than the allowed 10%. This could cause any planned withdrawal to fail due to previous withdrawals. Note that a similar issue could also exist in the ERC-20 bridge .

Consider recording `address(this).balance` when updating `s.lastWithdrawalLimitReset`, and using it as the base when calculating the daily withdraw limit.

Update: *Resolved in pull request #60 at commit 6365a8b. The Matter Labs team decided to completely remove the withdrawal limitation.*

Medium Severity

Refund recipient defaults to inaccessible address for contracts

In `_requestL2Transaction`, if `address(0)` is specified as the refund recipient, `msg.sender` is used by default. However, the `msg.sender` address will not be controllable by contracts on L2, so any refund, or the bridged ETH amount in case of a failed transaction, will be lost.

Consider disallowing unspecified refund recipients for any ETH transfers, or reverting in the case of an unspecified recipient if the sender is not an EOA.

The formula for `overheadForPublicData` uses `Tm` which is defined as the maximal transaction ergs limit here.

This appears to correspond in code to `L2_TX_MAX_GAS_LIMIT`:

> /// @dev The maximum number of L2 gas that a user can request for an L2 transaction

However, the calculation in code uses the `MAX_PUBDATA_PER_BLOCK` constant instead, which refers to:

> */// @dev The maximum number of the pubdata an L2 operation should be allowed to use.*

This corresponds to `Pm` in the documentation.

These appear to be different quantities, measured in different units, of different magnitudes (`80000000` vs `110000`). As a result, a denial of service may occur if the overhead is calculated incorrectly (underestimated), which will result in `l2GasForTxBody` being overestimated, and possibly reverting in `_writePriorityOp` despite having legitimate values passed as inputs.

Alternatively, as the overhead is underestimated, a larger-than-limit `l2GasForTxBody` may be submitted, which will cause failures on L2.

Consider adding test cases to the documentation with concrete example values, and implementing these test cases in the codebase test suite to ensure basic compatibility. Additionally, consider documenting in code both the correspondence of the constants to the documentation's notation, and the derivation and logic of the formulas implemented in comments in the same file, so that access to external documentation would not prevent the reader from reviewing the code.

Update: *Partially resolved in pull request #34 at commit 19c7b81. The Matter Labs team stated:*

> *Acknowledged. For now, we have decided to temporarily remove the overhead. The issue will be fixed once we introduce the block overhead back to our users.*

Unchecked subtraction underflow

depending on their values can still cause an underflow.

For example, the calculation of the memory overhead can result in arbitrarily large values depending on the value passed in `_encodingLength` and the constant `BOOTLOADER_TX_ENCODING_SPACE`, since both values' ranges are not validated.

Note that it is likely that there are additional ways by which the combination of different possible values of constants and inputs could cause the resulting overhead to be higher than the total gas limit.

This may result in an underflow of the unchecked subtraction. In turn, it will likely cause a revert due to subsequent `l2GasForTxBody` checks.

Consider not using the `unchecked` subtraction to prevent the underflow, and adding an explicit check to validate the overhead.

Update: *Resolved in pull request #54 at commit 94bc1a6.*

User could deposit more ETH than allowed

zkSync has implemented a limit on the amount of ETH that can be deposited to L2 per account. However, the code currently has a design flaw that allows users to bypass this limit by exploiting a gas refund scheme. Specifically, in the `requestL2Transaction` function, the deposited ETH amount is verified using the `_l2Value` parameter, while the `valueToMint` parameter is set to `msg.value` when composing a priority queue transaction.

This means that a user can mint ETH without triggering the limit by setting `_l2Value = 0` and using the gas refund when requesting any L2 transactions. Furthermore, even if `msg.value` was set as the limit amount, the user could run into issues when trying to request L2 transactions from L1 after reaching their limit.

To mitigate this issue, a system design change is needed around L2 gas refunding or ETH bridging limits. However, the specifics of the solution will depend on the overall design and goals of the system. Careful consideration and testing will be needed to ensure that the solution effectively mitigates this issue while also preserving the intended functionality of the system.

might not be able to request L2 transactions after reaching their limits.

Block overhead limits may be exceeded

In `_getOverheadForTransaction` some overhead values can go over their maximum values if the transaction data or the public data posted (e.g., state changes) are large:

- `_encodingLength` can take more memory than allowed by the `BOOTLOADER_TX_ENCODING_SPACE`.
- `overheadForPublicData` for the transaction can be larger than the `MAX_PUBDATA_PER_BLOCK` constant.
- `overheadForComputation` can be larger than the `L2_TX_MAX_GAS_LIMIT`.

Exceeding these values may violate invariants that are important for accurate L2 gas metering.

Consider checking that the encoded transaction length is in the expected range to prevent going over the maximum expected values. Additionally, consider checking that at no point the resulting calculated overhead is larger than the maximum overhead.

Update: *Partially resolved in pull request #34 at commit 19c7b81. The Matter Labs team stated:*

> *Acknowledged. We temporarily removed the block overhead, but a fix will be applied when restoring it.*

Lack of tests

There are very few tests for most functionalities.

For example, `Mailbox.sol` (particularly `MailboxFacet`) is a key contract for the L1 bridge, and has 360+ lines of code and a large dependency tree of aggregated thousands of lines of non-library solidity code (specific to this codebase). The code implements sensitive functionality with many important details and execution branches. However, there are only four basic tests in `mailbox_test.ts`:

- One test for non-reverting execution of an expected valid input. This tests nothing about the successful execution results.

Furthermore, as some system-level integration tests exist in another repository, there too, most of the functionality of the Mailbox remains untested:

- There are no invocations for the `finalizeEthWithdrawal` mutative method, or the views `proveL2LogInclusion`, `proveL1ToL2TransactionStatus`, and `serializeL2Transaction`.
- There is a single file for L1 functionality that invokes the main `requestL2Transaction` method. This test file totals around 300 lines of testing code, for which the majority of tests only assert either a revert or a lack of revert.

This leads to several potential issues:

- The correctness of the code can only be assessed based on partial, and changing documentation. This is because intended and unintended behavior is not captured in tests.
- Introduction of new vulnerabilities for established code in future code changes, since known positive and negative behaviors are not checked automatically.
- Higher likelihood of missed vulnerabilities in current and future development and review.

Consider adding contract-level testing to test all branches of execution. Additionally, consider implementing an ongoing measurement of testing coverage as a way to ensure at least 95% coverage.

Update: *Partially resolved in pull request #36, pull request #42, pull request #43, pull request #45, pull request #46, pull request #48 and pull request #51. The Matter Labs team stated:*

> *We are working on improving the test coverage over the entire codebase.*

Low Severity

Missing documentation

Docstrings and inline comments are missing in several parts of the codebase with sensitive functionality. For example:

- `_getMinimalPriorityTransactionGasLimit`: it is possible that this method overestimates / underestimates, or is implemented incorrectly. However, relevant documentation was insufficient to validate this.

Consider including thorough docstrings and inline explanations with references to relevant source files or documentation, allowing readers or maintainers to verify the implementations and their correct usage.

Update: *Resolved in pull request #55 at commits 41946cc and 57f702d.*

Revert messages are not informative

This issue has been reported in the previous Layer 1 Diff Audit (L01 – Missing error messages in require statements). Reverts are important logical components, and a lack of revert messages makes them confusing and increases the chance of missing vulnerabilities during a review.

The codebase as a whole has revert messages that consist of two letters and convey no information. Additionally, the two-letter combinations collide (for example “po”) for different contracts. Crucially, no resource is available to translate the codes into meaningful error messages. Although something is typically mentioned in comments next to the revert, a meaningful error message (or a custom error) is expected since comments can be outdated and cannot be tested.

Consider using informative error messages or custom errors throughout the codebase.

Update: *Acknowledged, not resolved. The Matter Labs team stated:*

> *We agree that custom errors will be much more understandable and convenient. At this time, we have no capacity for such a large refactoring, but we have planned it for the next milestone.*

L2 transaction hash is not emitted

The L2 transaction hash is a needed input during the L1 to L2 transaction flow on chain (in `claimFailedDeposit`), and for keeping track of L2 inclusion success off chain. However, it is not emitted in events by the callers in `_requestDeployTransaction()` or during ERC-

Consider emitting the resulting transaction hash in an event after submitting the request in the `MailboxFacet`.

Update: *Resolved. We later found that this was not an issue because the hash was emitted in event NewPriorityRequest as stated by the Matter Labs team:*

> `_requestDeploy` *is only used for the bridge initialization, so we do not think it may affect off-chain infrastructure (indexers, UIs, analytics dashboards). Moreover, the transaction hashes are emitted in the* `Mailbox` *itself.*

Diamond proxy holding large bridged ETH balance

In the current design, ERC-20 bridging uses a separate contract to hold token balances, but `MailboxFacet` holds and operates the ETH balance. However, since it is implemented as part of the Diamond Proxy, this means that the contract balance is common to all current and future facets of the proxy. This poses several risks:

- It exposes the bridged and locked funds to an additional risk of exploit by introducing vulnerabilities in the other facets.
- The locked balance is possibly co-mingled with other ETH that may be in use by the other facets.
- By adding functionality to handle outbound ETH transfers, it introduces a site for a call to a user-defined destination out of the main proxy that can potentially be leveraged for other future attacks.

Considering that the ETH balance of the bridge may become very large, it may be better to design a system that reduces these risks.

Consider handling ETH deposits and withdrawals by converting them to WETH and using the ERC-20 bridge. This will have the additional benefit of avoiding balance handling code duplication between the `MailboxFacet` and the ERC-20 bridge, and will also remove the need to make a dangerous ETH transfer call out of the contract.

Update: *Acknowledged, not resolved. The Matter Labs team stated:*

> *diamond proxy).*

Cross-chain system contract invocation is allowed and undocumented

System contract addresses can be specified in `requestL2Transaction`. However, this can cause potential unexpected side effects when executing the transactions on L2 since system contracts are documented to not be intended for direct invocation by users.

For example, if the destination address is set to the system `ContractDeployer`, it appears that the bootloader will execute it in `isSystem` mode. This code path is used for the deployment of the ERC-20 bridge in `L1ERC20Bridge.initialize`.

It is possible that allowing this invocation path from L1 may introduce vulnerabilities and side effects, depending on each specific system contract's access control. This is because L1 and L2 invocation paths in the bootloader are treated differently and may encode different assumptions which may be violated in one path but not in the other. If calling most system contracts is not an expected usage pattern, allowing the users to make these cross-chain calls creates an unnecessary attack surface area.

Consider restricting the addresses allowed to be called from L1. This can be done by checking that the destination address doesn't fall into the system contracts' address space. Additionally, consider documenting these usage patterns.

Update: *Acknowledged, not resolved. The Matter Labs team stated:*

> *We agree that calling system contracts can be dangerous in general. However, due to the design of L1-to-L2 transactions, we do not see any potential problems with calling system contracts. The same call may be done via L2 by directly calling the system contracts.*

Notes & Additional Information

Contract poorly named and structured

The `L2ContractHelper` file and contract present several issues that impact readability:

- `sendMessageToL1` is unused along with its dependencies (`L2_MESSENGER` and `IL2Messenger`). Consider removing it altogether.
- The file contains additional constants and interfaces that are not used within it, but are used in other importing files. For example, `IContractDeployer` or `FORCE_DEPLOYER` are not used in this file, among others. Consider removing the unused instances.

Update: *Resolved in pull request #55 at commits dfb4e4b and e2b2cf0.*

Local variable shadows state variable

In `Allowlist.sol` the `_owner` local variable shadows the `_owner` state variable from `Ownable`.

Consider renaming the local variable to avoid potential errors.

Update: *Resolved in pull request #56 at commit 40cd9d0.*

Facet contract names and filenames do not match

Facet contracts are suffixed with `Facet` (e.g., `MailboxFacet`) but the filenames lack the suffix (e.g., `Mailbox.sol`).

Consider renaming the files to match the contract names.

Update: *Acknowledged, will resolve. The Matter Labs team stated that they will resolve the issue:*

> *We are aware of this, and will address it when higher-priority tasks are solved.*

`require` statement lacking revert message

The `require` statement in `_requestL2Transaction` lacks an error message.

Consider adding one to improve the readability and clarity of the codebase.

Update: *Resolved in pull request #57 at commit 38b8426.*

Naming issues hinder code understanding and readability

- On line 29, `Withdrawal` should be `WithdrawalLimit`.
- On line 30, `withdrawalLimitation` should be `enabled`.
- On line 37, `Deposit` should be `DepositLimit`.
- On line 60, `getTokenWithdrawalLimitData` should be `getTokenWithdrawalLimit`.
- On line 62, `getTokenDepositLimitData` should be `getTokenDepositLimit`.
- On line 28, `tokenWithdrawal` should be `withdrawalLimits`.
- On line 31, `tokenDeposit` should be `depositLimits`.

Update: *Acknowledged, will resolve. The Matter Labs team stated that they will resolve the issue:*

> *Good suggestion! We are going to remove the deposit limitation (next milestone), and are already reimplementing the withdrawal limitation in H-02, so we will not apply these changes now.*

Refund can be used to split bridged ETH and force transfer

The bridged ETH amount is split between the `_l2Value` and the gas payment on L2, which is unknown at the time of the L1 submission. The refund mechanism therefore exists to refund the excess gas payment to a user-selected recipient. However, currently the L2 gas cost for an L1 transaction is set to 0 during transaction serialization. Since no gas payment is taken on L2, there is no current need for splitting the ETH amount. Allowing the splitting with no gas costs creates a problem by allowing to split ETH between two destinations.

The ETH refund in case of a successful transaction is always the difference between the `msg.value` and the user provided `_l2Value`. In this case, the bridging mechanism has these qualities:

- ETH is bridged simultaneously to two addresses (the recipient, and the refund recipient), in quantities determined by the user.
- ETH is forced upon the refund recipient even if it is a contract that has no payable fallback and is not able to transfer it later. This is because refunded ETH is transferred without a call (with only a forced balance update).

checking that `_l2Value` is always equal to `msg.value`.

Update: *Acknowledged, not resolved. The Matter Labs team stated:*

> *Acknowledged. L1-to-L2 transactions will become payable in the next milestone.*

Unused method

`_deriveL2GasPrice` is an unused method and can be removed to improve readability and reduce deployment gas costs.

The method could not be audited without usage context.

Update: *Acknowledged, not resolved. The Matter Labs team stated:*

> *It is planned to enable paid L1-to-L2 transactions for the next milestone, so we will not remove the method, since we will need to restore it soon.*

Grammar and spelling issues

In `AddressAliasHelper .sol`, consider improving the following comments:

- `Utility function that converts the address in the L1 that submitted a tx to the inbox to the msg.sender viewed in the L2` should be `Utility function converts the address that submitted a tx to the inbox on L1 to the msg.sender viewed on L2`.
- `Utility function that converts the msg.sender viewed in the L2 to the address in the L1 that submitted a tx to the inbox` should be `Utility function that converts the msg.sender viewed on L2 to the address that submitted a tx to the inbox on L1`.
- In line 134 of `Allowlist.sol`, `withdrwal` should be `withdrawal`.

Update: *Resolved in pull request #59 at commit ce11429.*

Inefficient code

```
blockOverheadForTransaction = Math.max(blockOverheadForTransaction
```

It is not necessary to use `Math.max` here, as `blockOverheadForTransaction` is 0 and `txSlotOverhead >=0`. Consider changing it to `blockOverheadForTransaction = txSlotOverhead` instead.

Update: *Acknowledged, will resolve. The Matter Labs team stated that they will resolve the issue:*

> *Acknowledged. The block overhead is removed for now, but we will fix the issue soon.*

Conclusions

During a two-week period, we conducted a differential audit that focused on the codebase changes related to the implementation of new functions such as ETH bridging and gas-related modifications. Our audit identified 2 high-severity issues, 5 medium-severity issues, as well as some low-severity issues and notes. Most of these issues are associated with the newly introduced functions and design decisions. Working with the Matter Labs team continues to be a great experience.

Appendix

Extra Monitoring Recommendations

While we have recommended monitoring solutions for the system in the past, it is important to also consider new monitoring solutions for the recently added functions.

Technical

Medium: Since the current system design relies on the operator to cover the gas cost of the rollup process associated with L1, consider monitoring the ETH balance of the operator’s address to ensure the system operates smoothly.

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