zkSync Bootloader Audit Report

OPENZEPPELIN SECURITY | JANUARY 27, 2023

Security Audits

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

Table of Contents

Summary

Type

Rollup

Languages

Solidity

Total Issues

29 (25 resolved, 1 partially resolved)

Critical Severity Issues

1 (1 resolved)

High Severity Issues

2 (2 resolved)

Medium Severity Issues

3 (2 resolved)

Low Severity Issues

11 (11 resolved)

Notes & Additional Information

12 (9 resolved, 1 partially resolved)

Scope

We audited the matter-labs/system-contracts repository at the `4ad1f26ae205d5a973216d141833e0ac37d72ec8` commit.

In scope were the following contracts:

```
└── contracts
    ├── BootloaderUtilities.sol
    ├── L2EthToken.sol
    ├── MsgValueSimulator.sol
    ├── interfaces
    |   ├── IBootloaderUtilities.sol
    |   ├── IL2StandardToken.sol
    |   └── IEthToken.sol
    └── libraries
        └── RLPEncoder.sol
```

System Overview

zkSync is a layer 2 scaling solution for Ethereum based on zero-knowledge rollup technology. The protocol aims to provide low transaction fees and high throughput while maintaining a large degree of EVM compatibility.

The scope of this audit included the `bootloader` which bundles transactions and executes them by delegating core functionality to system-contracts deployed on layer 2. The execution environment is the zkEVM, which uses distinct opcodes from the Ethereum-VM and different precompiles, but maintains a large degree of compatibility on the source code level.

To read the audit report of the layer 1 contracts: See our blog.

Bootloader

The bootloader is a key component of the system that manages the execution of layer 2 transactions. It is a specialized software that is not deployed like a regular contract, but rather runs within a node as part of the execution environment. The validator modifies transactions and stores them in an array, which is then written to the bootloader memory and executed. In order to avoid the possibility of the entire process rolling back due to a failure in a single transaction, the bootloader uses a softer fail condition known as a `near call panic`. The bootloader's functionality is divided between a large Yul file and the `BootLoaderUtilities` contract.

MsgValueSimulator

L2EthToken

The `L2EthToken` contract is a token that wraps the native ETH asset. Unlike ERC-20 tokens, it does not support direct user interaction, but is instead meant to be used by `MsgValueSimulator` to support the usage of `msg.value` within zkEVM.

Privileged Roles and Security Assumptions

In its current form, all validators for zkSync are operated by zkSync. However, the plan is to eventually decentralize this process.

The integrity of the bootloader is protected through a hash commitment on layer 1, which can be upgraded by Matter Labs.

Further, Matter Labs currently has the ability to force-redeploy system contracts on layer 2 in order to ensure their updateability.

Critical Severity

Useless Assertion Check

The `assertEq` function in the bootloader compares two values. If they are not equal, it throws an error. There are two issues with this implementation:

- The function compares the value of the `value1` parameter to itself, which means the comparison will always be true and the function serves no purpose.
- The function header includes the `value1` parameter twice, which causes a compile-time error when using a Solidity compiler such as `solc`.

Consider correcting the comparison by checking two distinct variables. Additionally, ensure that the custom zkEVM compiler throws an error when declaring two variables with the same name in the same scope.

Update: *Resolved in pull request #133 at commit 6e3c054.*

The `BootloaderUtilities` contract generates transaction hashes for different types based on the transaction data, including the signature. The signature includes a `v` value that encodes a parity bit `y` for address recovery purposes. The way that the parity bit is encoded into `v` depends on the type of transaction:

- For legacy transaction hashes, `v` can be either `27 + y`, or `35 + chainid * 2 + y` when the `chainid` is included (Ethereum Yellowpaper, page 5).
- For EIP2930 and EIP1559 transactions, `v` is simply encoded as `y` (either 0 or 1).

However, the `DefaultAccount` contract enforces that `v` must be either 27 or 28. This causes problems in the legacy transaction hash function because, when the `chainid` is included in the encoding, the value of `35 + block.chainid * 2` is added to `v`. But, since `v` is already set to either 27 or 28, as enforced by the `DefaultAccount` contract, the resulting `v` value does not comply with the standard described above, thereby giving a non-standard transaction hash.

Additionally, in the EIP-2930 and EIP-1559 transaction hash functions, the `v` value is again fetched as either 27 or 28 and encoded as such, even though it should be `0` or `1` as defined by the standard. Hence, the transaction hashes generated by these functions are also non-standard.

Consider fixing this by deriving the correct value from the `v` value that is given or changing the way the `v` value is enforced in the first place.

Update: *Resolved in pull request #157 at commits 3472d28 and 471a450.*

Unbound RLP Length Encoding

The `RLPEncoder` library allows the encoding of bytes and list-type values. These dynamic types need to be prefixed to indicate the type and length of the data. The type is indicated through an offset:

- `0x80` for bytes
- `0xc0` for a list

- Bytes 1st byte range: `[0x80, 0xb7]`
- List 1st byte range: `[0xc0, 0xf7]`

2. Length ≥ 56: The length of the data is encoded between offset and data. The length of the length in bytes is added onto the offset.
   - Bytes 1st byte range: `[0xb8, 0xbf]`
   - List 1st byte range: `[0xf8, 0xff]`

Visualization of 2.:

```
offset + length_of_data_length || data_length || data
```

In the (2.) case, we can see that the encoding of the length can at most be 8 bytes long. However, in the RLP library a length of up to 32 bytes is taken as input to encode the length. Hence, when encoding a length equal or greater than `2**64`, the length encoding requires 9 or more bytes. This bound violation ends up in a corrupted encoding. For instance, a byte encoding could thereby end up as a list encoding.

This issue was not identified as a problem for the codebase in scope, as the length to encode is based on transaction data, which is unlikely to be of size `2**64` or greater. However, as this library finds adoptions across other projects, the current implementation could lead to severe issues or introduction of vulnerabilities if not used properly.

Consider checking that the length to encode is bound to `2**64 - 1`.

Update: *Resolved in pull requests #134 and #160 at commits 61b8138 and 6e45486.*

Medium Severity

Overshadowing and Uncaptured Returns

In the `bootloader` contract, the function `setTxOrigin` has a return value `success`, which is overshadowed by a local variable of the same name within the function scope. In the `solc` Solidity compiler this triggers a compiler error.

when using other compilers.

The same issue applies to the `precompileCall` function, which has a return value that is not captured when it is called inside the `nearCallPanic` function.

To address these issues, consider removing all unused return values.

Update: *Resolved in pull request #135 at commit 45c04f9.*

String Literal Exceeds 32 Bytes Limit

In the `bootloader` contract, the string literal "Tx data offset is not in correct place" is used as an input to the `assertionError` function. It exceeds the 32-byte limit for string literals in Yul, which leads to a compile error with the `solc` Solidity compiler. The usage of another compiler might lead to a similar error or silently discard parts of the string.

To avoid any possible compiler and runtime issues, consider making the string shorter by removing or abbreviating some of the words.

Update: *Resolved in pull request #136 at commit d506790.*

Unprotected Initialization Function

In the `L2EthToken` contract, an unresolved comment acknowledges the fact that the `initialization` function is unprotected and anyone could set the `l2Bridge` address if they call the function before the legitimate operator. The comment describes the problem without presenting a solution.

Consider using the TypeScript-based templating system that is already present in the codebase to inject a constant address that limits the `initialization` call to one specific `msg.sender`.

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

> *We plan to rethink the approach of bridging ether in the new upgrade, the issue will be resolved there.*

Low Severity

documentation were identified:

- The `MAX_POSTOP_SLOT` constant is documented to account for 4 slots which are required for the encoding of the `callPostOp` call. However, the implementation accounts for 7 slots.
- The `MAX_MEM_SIZE` constant is of size `2**24`, while it is described as `2**16`.
- In the `callPostOp` function, the `txResult` parameter is described as `0` if successful and `1` otherwise. However, the `txResult` value is coming from low-level `call` that returns `1` if being successful and `0` otherwise. Proceeding forward, this result is correctly handled through the `ExecutionResult` enum, thereby affecting the documentation only.
- The `setErgsPrice` function is incorrectly documented as "Set the new value for the **tx origin** context value".
- The `lengthToWords` function is implemented differently from what the name and documentation are indicating. From the description it suggest to return the number of words needed for a specified length of bytes. However, the implementation returns the next bigger bytes length of words that are needed. The implementation is also inefficient and can be simplified.

In the `RLPEncoder` library, the comment on line 7 describes the size as equal to "14" bytes (dec), while "0x14" bytes (hex) is the correct size.

In the `BootloaderUtilities` contract, the comment on line 21 refers to "signedHash" while "signedTxHash" is the correct identifier name.

In the `L2EthToken` contract, the `transferFromTo` function claims to "rely on SameMath" which is probably referring to a `SafeMath` library which is also not used in that function.

Consider correcting the above mismatches to be precise about the implementation and its documentation to ease code review.

Update: *Resolved in pull requests #137 and #161 at commits 68f99cb, 7b66b8a, and e09b067.*

Incompatible Function Syntax

However, the Solidity compiler `solc` does not support the declaration of Yul functions with a parameter list containing a trailing comma.

Consider removing the trailing comma and ensure that your codebase maintains as much compatibility with the Solidity compiler whenever possible.

Update: *Resolved in pull request #138 at commit 5e1cf33.*

Inexplicit Fail in Bridge Burn

The `bridgeBurn` function in the `L2EthToken` contract adjusts a user's balance without checking their current balance first. This could result in an underflow error, which is providing insufficient information to the user.

To prevent this issue, consider adding a `require` statement to explicitly fail with a descriptive error string if the user's balance is exceeded.

Update: *Resolved in pull request #139 at commit 2053757.*

Inexplicit Imports

In the `BootloaderUtilities` and `MsgValueSimulator` contract, the `Constants.sol` file inexplicitly imports all constants. This hinders the visibility of what other components are actually used within the contract.

Consider changing the imports to explicitly import specific constants for better code clarity.

Update: *Resolved in pull request #156 at commit 94c79a5.*

Lack of Revert Messages

In the `L2EthToken` contract, the `require` statements in line 36-37, 42, and 54-58 lack an error message.

Consider adding the error message to fail more explicitly and ease debugging.

Update: *Resolved in pull request #140 at commit be287c9.*

the contract does not inherit from this interface. It appears that the intent may have been to inherit from the interface, but due to this oversight, there is a mismatch between the function names and return types in the interface and the contract.

In the interface, the function is defined as:

```
function getTransactionHash(Transaction calldata _transaction) ext
```

But in the contract, it is defined as:

```
function getTransactionHashes(Transaction calldata _transaction) e
```

This inconsistency may result in errors or unexpected behavior.

Consider inheriting from the `IBootloaderUtilities` interface in the `BootloaderUtilities` contract to ensure that the function names and return types are consistent and match the intended functionality. This will improve the reliability and maintainability of the codebase.

Update: *Resolved in pull request #141 at commit 827aad6.*

Missing Interface for `L2EthToken`

The `L2EthToken` contract implements two interfaces to enforce compliance with their respective functions. However, the contract also implements the standard ERC-20 functions `name`, `symbol`, `decimals`, and `totalSupply`.

Currently, nothing prevents misspellings and there is no convenient way to call these functions through address to interface conversion.

Consider either integrating them into one of the existing interfaces or defining an additional interface `PartialERC20`.

Update: *Resolved in pull requests #142 and #162 at commits f6fbaa9 and 5082111.*

stated that the first and second reserved slots of the transaction struct are used to store the `nonce` and `value`, respectively. These values are common to all types of transactions and could also be stored as explicit elements of the transaction struct.

To improve the code clarity, consider adding dedicated entries to the transaction struct for these common elements.

Update: *Resolved in pull request #143 at commits af4c5b9 and cd55ab2.*

TypeScript Constant Names Are Inconsistent

The `bootloader` contract includes constants that are defined in TypeScript code with the format , and are replaced with their actual values during compilation. These constants, which are often used for selectors, may or may not be padded with zeros. However, the names of these constants do not consistently indicate whether they have been padded or not, which is important for determining the memory layout in the bootloader.

To improve the clarity and consistency of the codebase, consider using consistent naming conventions for constants that indicate whether they have been padded or not.

Update: *Resolved in pull request #144 at commit 8ca44a7.*

Unused Functions

The following internal functions are defined in the `bootloader` contract, but appear to be unused:

- `ETH_L2_TOKEN_ADDR`
- `min`
- `ETH_CALL_ERR_CODE`
- `UNACCEPTABLE_ERGS_PRICE_ERR_CODE`
- `TX_VALIDATION_FAILED_ERR_CODE`

To improve the codebase, consider either using these functions or removing them. Removing unused functions can help to reduce clutter and make the code easier to understand.

> *want to avoid any confusion when starting to use such error codes in bootloader.*

Wrong EIP-712 Transaction Type Check

In the `validateTypedTxStructure` function of the bootloader, the EIP-712 `txType` is checked against 112, while it is supposed to be 113.

Following the execution flow, it can be seen in the `getTransactionHashes` function that a call reverts if the transaction type does not match one of 0, 1, 2, or 113. So for a transaction of type 113, the foreseen checks in the bootloader are skipped. This means the reserved slots can be arbitrary. However, no negative consequences were identified.

Consider correcting the transaction type check from 112 to 113. Further, consider implementing a default case and using the unused `valid` return variable to fail early if the transaction type does not match.

Update: *Resolved in pull request #146 at commit c343256.*

Notes & Additional Information

Code Redundancy

In the `BootloaderUtilities` contract, different types of transactions are encoded and hashed. These transaction types share similarities in their encoding which leads to redundancy of code. For instance, both the legacy and EIP-2930 type transaction have the consecutively encoded fields `nonce`, `gas price`, `gas limit`, `to`, and `value`.

Consider moving some parts of the encoding into a function to reuse the code for both transaction types.

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

> *Specifically in the specified places, we choose readability over performance.*

Commented-out Code

Consider removing the commented-out code as well as the comments describing them.

Update: *Resolved in pull request #147 at commit 7c344be.*

Extra Code

In the `bootloader` contract, the computation of a `switch` statement condition in the transaction processing includes unnecessary code that makes the code harder to read. The variables `txType` and `isL1Tx` are not used elsewhere in the function. Additionally, the `FROM_L1_TX_TYPE` variable appears to be a constant, but it is not declared as such.

To improve the clarity and readability of the code, consider simplifying the switch statement computation and properly declaring variables as constants if they are intended to be constant.

Update: *Resolved in pull requests #148 and #158 at commits 613636f and cdc301f.*

Inconsistent Declaration of Constants

There is an inconsistency in the way constants are declared in the code, with some being declared as functions and others being declared as variables. This deviation from a consistent pattern can be confusing and make it difficult to understand the purpose and usage of these constants. The following constants are declared as variables:

- `TX_DESCRIPTION_SIZE`
- `TXS_IN_BLOCK_LAST_PTR`

Consider using a consistent method for declaring constants throughout the code as functions. This will improve the readability and understandability of the code and make it easier for others to work with it.

Update: *Resolved in pull request #149 at commits c7042cd and fee4b5b.*

Inconsistent Declaration of Integers

In the `bootloader` contract, there is an inconsistency in the way memory offsets are declared in the codebase, with some being expressed in decimal and others being expressed in

- `add(0x20, txDataOffset)` in line 291
- `add(0x20, txDataOffset)` in line 300
- `mstore(txDataOffset, 0x20)` in line 431
- `add(txDataOffset, 0x20)` in line 797
- `add(txDataOffset, 0x20)` in line 1170
- `add(dataPtr, 0x20)` in line 1400

Consider using a consistent notation for expressing memory offsets throughout the codebase.

Update: *Resolved in pull request #150 at commit ecdbebd.*

Performance Optimization

In the `bootloader` contract, we have identified several opportunities for performance optimization:

- In the `validateTypedTxStructure` function, when checking if the `reservedDynamicLength` value is not zero, consider loading the length from the `getReservedDynamicPtr` pointer directly to skip the `lengthToWords` computation.
- In the `callAccountMethod` function, consider using the existing `txDataOffset` pointer instead of advancing the `txDataWithHashesOffset` a second time to save an `add`.
- In the function call to `executeL1Tx`, consider propagating the given `innerTxDataOffset` instead of recalculating it in the function itself.
- In the `getFarCallAbi` function, `dataOffset` and `memoryPage` are zero and could be skipped. Consider starting with the shifted `dataStart` value instead.
- The `executeL2Tx` function is only called within the `ZKSYNC_NEAR_CALL_executeL2Tx()` function where the `from` transaction value is present. Consider propagating that `from` value to the `executeL2Tx` call instead of recomputing it.

Update: *Partially resolved in pull request #151 at commit fe92113. The Matter Labs team stated:*

could be such an optimization, because we have an EVM backend.

TODOs Are Present in the Codebase

In the `bootloader` contract, there are incomplete implementations and missing functionality in the codebase, as indicated by the presence of "TODO" comments. This can make it difficult to understand the intended functionality and behavior of the code, and may also make it difficult to properly test and maintain the codebase. For instance:

- "make user pay for sending back the L1 message"
- "(SMA-1220): refunds are not supported as of now"
- "(SMA-1220): support refunds […]"

Consider completing the implementations and addressing the missing functionality. This will improve the overall quality and reliability of the codebase and make it easier for others to work with it.

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

> *The issue will be resolved in upcoming upgrades.*

Typographical Errors

In the codebase the following typographical errors were identified:

- "shoud" → "should"
- "Firsly" → "Firstly"
- "succedes" → "succeeds"
- "Reseting" → "Resetting"
- "mainntet" → "mainnet"
- "only one higher" → "only the first one above"
- "again" → "against"

Consider correcting these and any further issues.

Update: *Resolved in pull request #152 at commits 9dd29e3 and 630625f.*

value `proved_block` or `playground_block`. On line 1523, the inclusion of a block is determined by negating the value of `BOOTLOADER_TYPE`. While this may be semantically correct, it can be confusing to readers and might introduce errors or misunderstandings among developers.

To improve readability, consider using the non-negated form exclusively to specify which code should be included. For example, using

```
if BOOTLOADER_TYPE == 'proved_block'
```

to include the block when `BOOTLOADER_TYPE` is set to `proved_block`, and

```
if BOOTLOADER_TYPE == 'playground_block'
```

to include the block when `BOOTLOADER_TYPE` is set to `playground_block`.

Update: *Resolved in pull request #153 at commit 9b9d534.*

Unintuitive Naming

In the `bootloader` contract, two occurrences of unintuitive variable naming were identified:

- The naming of `RESERVED_FREE_SLOTS` is somewhat confusing, as it suggests that the slots are both reserved and free at the same time. In reality, the reserved slots may be free, but most of the slots will be pre-filled.
- The terms `txInnerDataOffset` and `innerTxDataOffset` are both used for semantically similar parameters. This inconsistency in naming can be confusing and limit the ability to easily search for all occurrences.

Consider using more consistent and intuitive naming conventions for variables and terms in the codebase.

Update: *Resolved in pull request #163 at commit 83931d4.*

- `IBootloaderUtilities` interface,
- `SystemContractHelper` library, and
- `Constants` file

are imported but never used.

Consider using or removing them.

Update: *Resolved in pull request #154 at commit 26523a2.*
The `IBootloaderUtilities` *interface is now used as part of the L-06 fix.*

Unused Variable

The `encodedChainId` variable in the `BootloaderUtilities` contract is unused.

Similarly, in the `bootloader` contract the variables `from` and `ergsLimit` remain unused within the `ZKSYNC_NEAR_CALL_validateTx` function.

Consider removing these unused variables.

Update: *Resolved in pull request #155 at commit bca9d68.*

Conclusions

This is the third report of the engagement. The audit lasted for 4 weeks, during which we had the opportunity to work closely with the Matter Labs team. They were very responsive in answering our questions and provided us with extensive and dedicated documentation that was extremely useful.

The scope of the audit included the `bootloader` as well as three more layer 2 system contracts with corresponding interfaces and one library. The `bootloader` is a complex component that orchestrates various system contracts in order to execute layer 2 transactions and log them to layer 1. It is a unique piece of software and therefore very interesting to audit.

During the audit, we identified several issues, including one critical and two high severity issue, as well as a number of medium and lower severity issues. Overall, the audit was successful in

Appendix

Monitoring Recommendations

While audits help in identifying code-level issues in the current implementation and potentially the code deployed in production, we encourage the Matter Labs team to consider incorporating monitoring activities in the production environment. To ensure the security of the project, we use both on-chain and node monitoring to identify potential threats and issues. With the goal of providing a comprehensive security assessment, we recommend the following measures:

On-Chain Monitoring

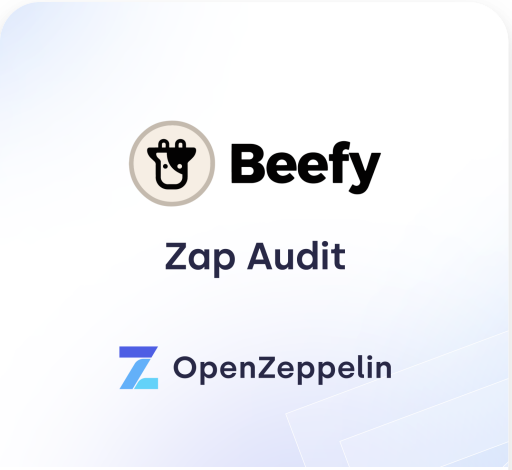
Critical: Monitor which addresses act as an EOA and trigger a warning when code is deployed on that address. In case this happens unwillingly, the incident could mean loss of power for that EOA by having a malicious contract acting on behalf of that user.

High: The implemented account abstraction allows the creation of custom accounts. Custom accounts may implement calls with the `isSystem` call flag set. This is a sensitive functionality which enables impersonation of the message sender through mimic calls. Unintentional usage of this flag could mean loss of funds through impersonation by a malicious party.

Node Monitoring

Low: The `vmHook` memory data of the `bootloader` can be leveraged by the node operator to check whether execution flow of the bootloader is as intended. Any violation to the control flow integrity could mean a compromise of the system.

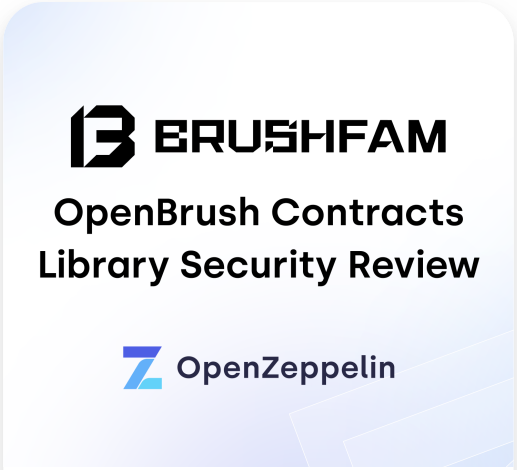
Low: The bootloader is responsible for collecting transaction fees for the operator address. The operator can verify that the fees received on-chain match the fees defined in the raw transaction data received by the node. If there is a discrepancy, it could indicate a problem with the implementation of fee management.



Beefy Zap Audit

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

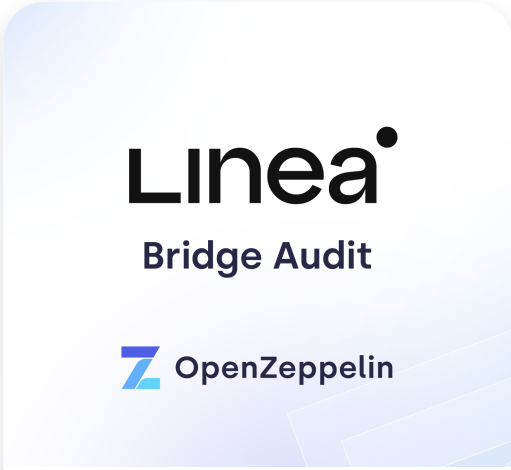
Security Audits



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