OpenZeppelin

zkSync Layer 1 Diff Audit

OPENZEPPELIN SECURITY | DECEMBER 13, 2022

Security Audits

December 13th, 2022

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

Table of Contents

Summary

Type	Rollup
Timeline	From 2022-11-21 To 2022-11-25
Languages	Solidity
Total Issues	16 (15 resolved)
Critical Severity Issues	1 (1 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	2 (2 resolved)
Low Severity Issues	4 (4 resolved)
Notes & Additional Information	9 (8 resolved)

Scope

We performed a diff-audit of the matter-labs/zksync-2-dev repository at the `534dc37c7b86753803a31b836f78896d18372e7e` commit against the post-audit

In scope were the following contracts:

```
contracts/ethereum/contracts
├── zksync
│   ├── Storage.sol
│   ├── DiamondInit.sol
│   ├── DiamondProxy.sol
│   ├── Config.sol
│   ├── facets
│   │   ├── Executor.sol
│   │   ├── Mailbox.sol
│   │   ├── DiamondCut.sol
│   │   ├── Getters.sol
│   │   ├── Governance.sol
│   │   └── Base.sol
│   ├── libraries
│   │   ├── Diamond.sol
│   │   ├── PriorityQueue.sol
│   │   ├── SafeCast.sol (new)
│   │   ├── UncheckedMath.sol (new)
│   │   └── Merkle.sol
│   └── interfaces
│       ├── IMailbox.sol
│       ├── IExecutor.sol
│       ├── IGetters.sol
│       ├── IDiamondCut.sol
│       ├── IGovernance.sol
│       └── IZkSync.sol
└── common
    ├── AllowList.sol (new)
    ├── AllowListed.sol (new)
    └── interfaces
        └── IAllowList.sol (new)
```

protocol aims to provide low transaction fees and high throughput while maintaining full EVM compatibility.

An overview of the layer 1 system architecture has been provided in a previous audit report: zkSync Layer1 Audit.

Privileged Roles and Security Assumptions

In addition to the privileged roles described in the last audit report, the system contains the following additions and changes:

An additional **allowlist** enables granular access management to the `Mailbox` facet and the L1 ⇔ L2 bridges built upon it.

The **security council** changed from a set of individual addresses to a single address that is assumed to be a Gnosis multisig. It continues to be limited to reducing the time a governor must wait between proposing and executing a system upgrade.

The **governor** is no longer forced to publicly reveal any information about an upgrade. It can choose between a transparent upgrade of the system, which reveals the diamond cut to be applied, and a shadow upgrade procedure that only reveals its hash. The execution of an upgrade is bound to a hash commitment during the proposal, either based on the transparent diamond cut data or given through the governor.

Deployment Status

A previous version of the codebase has already been deployed on Ethereum mainnet. Its usage is limited to participants of the alpha launch.

Critical Severity

Storage collision leads to failure of the system

In the `Storage.sol` file, the `AppStorage` struct keeps track of all stored values for the layer 1 system as part of the Diamond proxy pattern. The first entry of this struct is the `DiamondCutStorage` struct for the deployed code version and

As seen by comparing their entries, the structs have changed in size. Originally, `DiamondCutStorage` occupied 7 storage slots, while `UpgradeStorage` only occupies 2. Respectively, the slots of all following variables in the `AppStorage` struct shift, thereby resulting in a storage collision. Due to this desynchronization of storage, the entire system functionality would break and come to a halt.

For instance, the new `governor` address slot would match the former `lastDiamondFreezeTimestamp` value. Hence, by applying the upgrade, the operator loses the governor privilege. As such, the ability to perform further upgrades is lost and no other mechanism of recovery is present in the system. Moreover, the new `verifier` mapping would match the old `securityCouncilMembers` mapping. Thus, former council members would become validators. Since all stored block info would be lost, the validators could not commit any new blocks anyways.

In order to apply the upgrade in spite of the storage collision, one could re-initialize the `AppStorage` during the initialization stage of the *same exact* Diamond upgrade. However, properly overwriting the mappings does not seem feasible from a gas consumption perspective. Additionally, it requires a diligent off-chain data collection of all mapping data in need of overwrites. In conclusion, missing this opportunity is very probably and likely to fail.

Consider leaving the former struct as is and only appending new data to the end of the `AppStorage` struct. Thoroughly document that while `DiamondCutStorage` is unused, it must remain a member of `AppStorage`. Additionally, create a machine-readable artefact describing the deployed `AppStorage` layout, and integrate a check against it in your CI system.

Update: *Resolved in commit cd417be*.

Medium Severity

Arbitrary `l2Logs` length can lead to unprovable blocks and log inclusion

In the `ExecutorFacet` contract, the layer 2 logs are processed as part of committing blocks to layer 1. These layer 2 logs have a structured format and are used to extract data of what

- All logs affect the commitment that is later used for proof verification. Any mismatched log data would result in an unprovable commitment and would need to be reverted.
- In the `Mailbox` facet, it can be checked whether specific logs were emitted as part of a block by performing a Merkle inclusion check with a proof length of 9. In the event of more than 512 logs per block, their inclusion cannot be proven.
- The constant `L2_TO_L1_LOGS_COMMITMENT_BYTES` is unused.

Consider making use of the constant to check the size of the layer 2 logs such that the aforementioned concerns are eliminated.

Update*: Resolved in commits 39e83ee and 7dceb6b.*

Governor can immediately execute diamond upgrades

In the `executeUpgrade` function of the `DiamondCut` facet, the Boolean value `upgradeNoticePeriodPassed` indicates whether the upgrade waiting period has passed. However, the value is never used, thereby allowing the governor role to propose and immediately execute a transparent proposal without the need of council approval. This could have a severe impact for a rogue governor.

Consider checking that the Boolean value holds true such that the implementation reflects the original intention.

Update*: Resolved in commits ba31745 and 65ad664.*

Low Severity

Missing error messages in require statements

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

- The `require` statement on line 43
- The `require` statement on line 45
- The `require` statement on line 291

Update: *Resolved in commit 6f446e6.*

Require statement with multiple conditions

There is a `require` statement in the `AllowList` contract that requires multiple conditions to be satisfied.

In order to simplify the codebase and raise the most helpful error messages for failing `require` statements, consider having a single require statement per condition.

Update: *Resolved in commit 33cb3f5.*

Lack of explicit proposal id

In the `DiamondCut` contract, the `proposeShadowUpgrade` function checks the equality of its parameter `_proposalId` to the current proposal id incremented by one. In contrast, the `proposeTransparentUpgrade` function auto increments the current proposal id obtained from storage and does not require passing an additional parameter.

Consider adding the `_proposalId` as an additional parameter to the `proposeTransparentUpgrade` function to provide a consistent proposal flow and avoid any confusion related to the proposal id.

Update: *Resolved in commit c70dda0.*

Reimplementation of library functionality

Throughout the codebase we found two occurrences of re-implemented functionality that is available in community-vetted Solidity libraries:

- The two-step ownable control flow implemented in the `AllowList` contract is equivalent to the functionality implemented in the OpenZeppelin `Ownable2Step` contract. Consider extending the `Ownable2Step` contract with the `AllowList` functionality.
- The `SafeCast` library is a copied code snippet from the OpenZeppelin library. Consider replacing the imports and removing the custom `SafeCast` library.

Update: *Resolved in commit 60b66b0.*

Notes & Additional Information

`NewOwner` event should emit old and new state

The `NewOwner` event emits the new owner whenever the pending owner accepts ownership of the `AllowList` contract.

Consider emitting both the old and new owner to allow for better traceability of ownership changes via off-chain clients.

Update: *Resolved. The Matter Labs team stated:*

> *We implemented L04 and inherited the Ownable2Step contract so no change is needed.*

Gas optimizations

A few places in the codebase could benefit from gas optimization, for example:

- In the `senderCanCallFunction` modifier, instead of casting the function signature from `msg.data`, consider using `msg.sig`.
- The loop in the `Executor` facet could have incremented `currentTotalBlocksVerified` once, instead of doing it twice in line 237 and line 250.
- In the `Executor` facet, the `proveBlocks()` function performs a safety check to ensure that the number of verified blocks is smaller or equal to the total blocks committed. Consider to fail early by placing the require statement before the proof verification process, i.e., after the loop of committed block checks in line 252.

Consider optimizing these code sections to make them more gas efficient.

Update: *Resolved in commit 894c0c8.*

Invalid docstring

Consider removing this docstring.

***Update**: Resolved in commit 9bdec5a.*

Unused constants

Within the `Config.sol` file, there are a few constants that are defined but never used throughout the codebase. For instance:

- `L2_TO_L1_LOGS_COMMITMENT_BYTES`
- `INITIAL_STORAGE_CHANGES_COMMITMENT_BYTES`
- `REPEATED_STORAGE_CHANGES_COMMITMENT_BYTES`

Consider either using these constants, documenting why they are left in the code, or removing them.

***Update**: Resolved in commit 526a3fe.*

Files import non-existent contract

The files `Storage.sol` and `IGovernance.sol` import the file `Verifier.sol`, but the file does not exist.

Consider removing these imports.

***Update**: Resolved, not an issue. The Matter Labs team stated:*

> *The `Verifier.sol` contract is generated by the server according to the public keys. So, the import file exists, but in a non-explicit way.*

Indecisive licensing

Throughout the codebase there are several files that state an SPDX license identifier of "MIT OR Apache-2.0".

Consider agreeing on one license per file to prevent confusion on how these files can be used.

In the `Getters` facet, the function `isFacetFreezable` is implemented even though it is not defined in the `IGetters` interface.

In the `Governance` facet, the function `setPorterAvailability` has the Boolean parameter `_zkPorterIsAvailable` while it is called `_isPorterAvailable` in the `IGovernance` interface.

Consider correcting the above differences.

***Update**: Resolved in commit 74eb251.*

Non-explicit imports are used

Throughout the codebase, non-explicit imports are used, which reduces code readability and could lead to conflicts between the names defined locally and the ones imported. This is especially important if many contracts are defined within the same Solidity files or the inheritance chains are long.

Following the principle that clearer code is better code, consider using a named import syntax (`import {A, B, C} from "X"`) to explicitly declare which contracts are being imported.

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

> *We acknowledge that this issue raises a valid concern. It does not pose a security risk, so we have added it to our development backlog.*

Typographical errors

In the `AllowList` contract a typographical error was identified where "The address that the owner proposed as one that will replace its" should be "The address proposed by an owner to replace their address".

Consider fixing this error and any other typographical errors for readability and explicitness.

***Update**: Resolved in commit e337941.*

latest changes to the codebase since the fix review process was completed by the Matter Labs team on October 19th, 2022. For the most part, the changes were high-level, but also new features were implemented, which have improved the codebase overall.

However, as these changes perform an upgrade to already deployed contracts, a critical issue was identified. Further, two medium severity issues were found, as well as five low severity issues.

Working with the Matter Labs team continues to be a great experience as they provide extensive explanations and documentation in a timely manner.

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. Ongoing monitoring of deployed contracts helps in identifying potential threats and issues affecting the production environment. Hence, with the goal of providing a complete security assessment, we want to extend the recommendations of our previous audit based on the new features.

Upgrades: The Diamond pattern is still the central piece of upgradeability. However, the proposal of upgrades is split into two functions: a transparent mode and a shadow mode. These respectively emit the events `ProposeTransparentUpgrade` and `ProposeShadowUpgrade`. While the transparent mode works as the previous proposal, the shadow mode allows upgrades with unexposed diamond cuts. Nonetheless, this upgrade must be approved by the security council and does not underlie the time delay.

As the upgrade can be ambiguous, these proposals and executions should be thoroughly monitored.

Governance: The governance aspect of the layer 1 contracts got extended by an `AllowList` contract. The owner of the contract can thereby manage different access modes for a target address:

- Closed – No one can call anything.

Contracts enable the allowlist by applying a modifier to the respective functions. The modifier then checks against the `AllowList` contract to determine whether the caller is eligible to make the call.

As part of maintaining this allowlist, the following events are emitted:

- `UpdateAccessMode` – changing the high-level permission
- `UpdateCallPermission` – changing the permission per caller, target, and function
- `NewPendingOwner` – initiating a new owner for the allowlist
- `NewOwner` – acceping the ownership of the allowlist

It is important to monitor whether suspicious addresses are given the permission to call sensitive functions of the system. Unplanned changes to the access mode could also indicate a DoS (Denial-of-Service) attack. Lastly, any changes to the function-critical owner role should carefully be tracked.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users'

OpenBrush Contracts Library Security Review

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-

Security Audits

Security Audits

Security Audits

OpenZeppelin

Defender Platform

Secure Code & Audit
Secure Deploy
Threat Monitoring
Incident Response
Operation and Automation

Services

Smart Contract Security Audit
Incident Response
Zero Knowledge Proof Practice

Learn

Docs
Ethernaut CTF
Blog

Company

About us
Jobs
Blog

Contracts Library

Docs