OpenZeppelin | security

Pull Request #1436 Review

May 2, 2025

Table of Contents

Summary

Type	Cross-Chain
Timeline	From 2025-04-28 To 2025-04-29
Languages	Solidity

Scope

OpenZeppelin audited pull request #1436 of the matter-labs/era-contracts/ repository.

In scope were the following files:

```
era-contracts/l1-contracts/contracts/upgrades/
    - BaseZkSyncUpgrade.sol
    - BaseZkSyncUpgradeGenesis.sol
    - ZkSyncUpgradeErrors.sol
```

Overview

On April 28th, the Matter Labs team informed OpenZeppelin about a potential vulnerability in the code that handles ZKChain migration between settlement layers. Subsequently, a 2-day review of pull request #1436 was conducted to validate the correctness and security outlook of the fix that had been implemented to address this issue. The review team confirmed that pull request #1436 had correctly fixed the identified issue.

Reported Issue

The Matter Labs team described the issue in the following manner:

Withdrawing A Chain Back To L1 May Fail

> *We generally expect that once a chain migrates to Gateway, it will still have to maintain the two diamond proxies: on GW and on L1. If a chain is migrated back to L1 at some point, it would require to have the same protocol version in both places (on GW and on L1). But when our chain migrates away from L1, its upgrade data is written in the data to be moved to GW, while it is preserved on L1. So when we will try to upgrade the L1 diamond proxy of the migrated chain, a revert will happen since laying one upgrade*

> *transaction on top of another is prevented. Consequently, if a chain with upgrade data migrates on top of GW and after that another upgrade happens, there is no way to return to L1.*
>
> *We believe that it is secure to keep `AdminFacet` as is, and we should modify `DefaultUpgrade` so that if the chain is not on settlement layer then nothing related to upgrade transactions should be enforced.*

Issue and Fix Analysis

Upgrade Process

In order to understand the reported issue, it is worth describing some key points of the typical procedure that is followed when upgrading L2 system contracts. Hence, let us assume that L1 is the only Settlement Layer (SL).

When a protocol upgrade is performed on a chain, it is important to ensure that the protocol version on both L2 and L1 remains the same. So, when an upgrade takes place on L1, it must be ensured that the same upgrade also takes place on L2. For this reason, upon an upgrade on L1, the transaction hash of the protocol upgrade is stored in the chain's Diamond Proxy (DP) storage. Afterward, it is expected that the L2 upgrade TX will be processed in the upcoming submitted batch. During this processing, it is checked that the stored upgrade hash equals the cross-chain transaction's log.

Backward-Compatibility Issue

With the introduction of the Gateway (GW), it is now possible to migrate a chain's SL between L1 and GW. Regarding the upgrade process described above, this means that, at any given time, the currently "active" SL will be processing the L2 batches and will, thus, be enforcing the protocol version synchronization upon upgrades. However, note that in order to migrate a chain to an SL, it is required that its protocol version is up-to-date. This means that an "inactive" SL still needs to be upgraded to ensure that the chain is able to migrate back if needed.

Now, let us consider the following scenario which would make it impossible to upgrade the inactive SL and, therefore, result in an inability to migrate the chain back.

Consider a chain that settles on L1 and is upgraded at some point. Now, consider the specific point in time at which the L1 upgrade had been performed. The upgrade data has been stored in the chain's L1 DP while the L2 batch containing the L2 upgrade TX has been committed on L1 but has not yet been finalized. At this point, the chain is migrated to GW along with all of its DP data, including the upgrade data.

Note that the chain had been migrated before the upgrade TX was finalized on L1. This means that the stored data which denotes a pending state on the L2 upgrade will remain uncleared on L1 even if the corresponding batch has been finalized on GW. As a consequence, it will be impossible to perform another upgrade on L1. So, in case another upgrade is performed on the GW, the chain will not be able to migrate back to L1 because of the outdated protocol version.

The above scenario is the issue described by the Matter Labs. Note that this is only one of the possible scenarios that culminate in failing chain migration. Another possible scenario is upgrading a migrated chain more than once, but not being able to upgrade the "inactive" SL more than once because of the pending L2 batch processing. It should also be noted that the issue may apply to any SL from which a chain has migrated.

The review team estimates this issue to be of medium severity because it may disable an SL if specific upgrade actions take place without any funds being at risk (at least not immediately).

Conclusion - Fix Review

Pull request #1436 addresses the issue by updating the upgrade-related code so that no further actions are required when the upgrade is performed on an inactive SL. The standard process whereby the consistency of the upgrade transaction hash is verified through the chain's batch commitment still takes place on the active SL.

The current solution, as embodied by pull request #1436, assumes that for the upgrades in the inactive SL, the consistency of the upgrade transaction and data are guaranteed by the CTM's owner who is responsible for performing the upgrades. This trust assumption is reasonable considering that governance is the CTM's owner. However, extra caution is required to ensure the correctness of the data provided by governance. The fix also ensures that a genesis upgrade is only possible on the active SL.