Polygon

PR 478 Changes Review

Security Assessment Report

Version: 2.0

July, 2025

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Polygon components in scope. The review focused solely on the security aspects of the Solidity implementation of the contracts, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the components in scope. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Polygon components contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Polygon components in scope.

Overview

This review covers changes enabling the migration from state transition rollups to pessimistic proof rollups.

A new function has been introduced to initiate the migration, with the system designed to automatically finalise the transition upon successful verification of the first post-upgrade proof. These changes aim to ensure a secure and consistent migration aligned with the existing rollup lifecycle.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the Agglayer Contracts repository.

The scope of this time-boxed review was strictly limited to changes made to the Solidity files at PR-478.

The fixes of the identified issues were assessed at tag v11.0.0-rc.2 (commit fcb5121).

Note: third party libraries and dependencies were excluded from the scope of this assessment.

Approach

The security assessment covered components written in Solidity.

The manual review focused on identifying issues associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout).

Additionally, the manual review process focused on identifying vulnerabilities related to known Solidity anti-patterns and attack vectors, such as re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers.

For a more detailed, but non-exhaustive list of examined vectors, see [1, 2].

To support the Solidity components of the review, the testing team may have used the following automated testing tools:

- Aderyn: `https://github.com/Cyfrin/aderyn`
- Slither: `https://github.com/trailofbits/slither`
- Mythril: `https://github.com/ConsenSys/mythril`

Output for these automated tools is available upon request.

Coverage Limitations

Due to the time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 3 issues during this assessment. Categorised by their severity:

- Informational: 3 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Polygon components in scope. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the components, including optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected components(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
478-01	Unclear Comment On Migration Process	Informational	Resolved
478-02	Shadowed Variable Names	Informational	Resolved
478-03	Miscellaneous General Comments	Informational	Resolved

478-01	Unclear Comment On Migration Process
Asset	`v2/PolygonRollupManager.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The bootstrap certificate process appears to always use a `lastLocalExitRoot` value of `bytes32(0)` during migration. This is due to the logic in `verifyPessimisticTrustedAggregator()` where the field is explicitly set to zero:

```
// In this special case, we consider lastLocalExitRoot is zero.
rollup.lastLocalExitRoot = bytes32(0);
```

Although this value is later overwritten on line [**1363**]:

```
rollup.lastLocalExitRoot = newLocalExitRoot;
```

The zero value is still used when preparing the input for the prover:

```
bytes memory inputPessimisticBytes = _getInputPessimisticBytes(
    rollupID,
    rollup,      // @audit contains rollup.lastLocalExitRoot of zero
    l1InfoRoot,
    newLocalExitRoot,
    newPessimisticRoot,
    aggchainData
);
```

The comment above this assignment lacks detail about why `bytes32(0)` is used and what the "special case" entails. Expanding the comment to explain that this zero value is intentionally used only as prover input during the bootstrap phase would significantly improve code clarity and help future readers understand the nuances of the migration process.

Recommendations

Consider expanding the comment and clarifying the use of a `lastLocalExitRoot` of zero.

Resolution

The comment was expanded in commit b0e9505.

478-02	Shadowed Variable Names
Asset	`v2/PolygonRollupManager.sol` `v2/lib/LegacyZKEVMStateVariables.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The names of two variables in the inherited contract `LegacyZKEVMStateVariables`:

```
// Last pending state
/// @custom:oz-renamed-from lastPendingState
uint64 internal _legacyLastPendingState;

// Last pending state consolidated
/// @custom:oz-renamed-from lastPendingStateConsolidated
uint64 internal _legacyLastPendingStateConsolidated;
```

are used as variable names for two of the return variables in the function `rollupIDToRollupDataDeserialized()`:

```
function rollupIDToRollupDataDeserialized(
    uint32 rollupID
)
    public
    view
    returns (
        address rollupContract,
        uint64 chainID,
        address verifier,
        uint64 forkID,
        bytes32 lastLocalExitRoot,
        uint64 lastBatchSequenced,
        uint64 lastVerifiedBatch,
        uint64 _legacyLastPendingState,
        uint64 _legacyLastPendingStateConsolidated,
        uint64 lastVerifiedBatchBeforeUpgrade,
        uint64 rollupTypeID,
        VerifierType rollupVerifierType
    )
```

Although this has no immediate impact, it introduces a risk of confusion or misuse if the state variables are inadvertently accessed in place of the intended return variables.

Note, given that the variables are deprecated, the likelihood is low, but addressing this aligns with best practices to avoid future errors.

Recommendations

Consider changing the names of one of the occurrences of the variables.

Resolution

The shadowed variables were renamed in commit 1e04283.

478-03	Miscellaneous General Comments
Asset	All contracts
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Inconsistent Styles For Identical Tests**

   Related Asset(s): v2/PolygonRollupManager.sol

   This check from `initMigrationToPP()`:

   ```
   // No pending batches to verify allowed before migration
   require(
       rollup.lastBatchSequenced == rollup.lastVerifiedBatch,
       AllSequencedMustBeVerified()
   );
   ```

   is also performed in `updateRollupByRollupAdmin()`:

   ```
   // Check all sequenced batches are verified
   if (rollup.lastBatchSequenced != rollup.lastVerifiedBatch) {
       revert AllSequencedMustBeVerified();
   }
   ```

   One check is written as a positive condition, the other as a negative, but both evaluate the same status and trigger the same error.

   Consider unifying the logic to improve consistency and readability.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team's responses to the raised issues above are as follows:

1. **Inconsistent Styles For Identical Tests**

   The logic was unified in commit 9d8f9ad.

Appendix A Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: `https://blog.sigmaprime.io/solidity-security.html`. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: `http://www.dasp.co/`. [Accessed 2018].