Polygon

LX/LY Bridge - Banana Upgrade

Version: 2.0

June, 2024

Contents

Introduction **2**
- Disclaimer . . . 2
- Document Structure . . . 2
- Overview . . . 2

Security Assessment Summary **3**
- Scope . . . 3
- Approach . . . 3
- Coverage Limitations . . . 3
- Findings Summary . . . 3

Detailed Findings **5**

Summary of Findings **6**
- `rollbackBatches` Does Not Account For Pending Batches . . . 7
- Sequencing Fees Are Not Refunded On Rollback . . . 8
- Forced Batches Can Be Censored Indefinitely With `rollbackBatches()` . . . 9
- `batchToRollback` Is Not The Batch To Roll Back . . . 10
- Miscellaneous General Comments . . . 11

A Test Suite **12**

B Vulnerability Severity Classification **15**

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Polygon "Banana Upgrade" smart contracts (working title). The review focused solely on the security aspects of the Solidity implementation of the contract, 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 smart contract. 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 smart contracts 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*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Polygon smart contracts.

Overview

Polygon run multiple zero-knowledge (ZK) rollup scaling solutions designed to work with the Ethereum Virtual Machine (EVM). These zkEVMs support the deployment of smart contracts written for the EVM while providing scaling in the ZK prover. Due to the ZK prover, there is a faster security consensus achieved than with other optimistic rollup designs.

Faster consensus enables faster bridging between Polygon zkEVM and other Layer 2s or Ethereum Mainnet, this is natively supported via the Polygon LX/LY bridge. The LX/LY bridge enables cross-chain communication between various Polygon chains and/or the Ethereum Mainnet.

This review focuses on the "Banana" upgrades. These updates aim to add more robustness to the rollup system by introducing rollbacks and more flexible level 1 info root usage.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the Polygon zkEVM repository.

The scope of this time-boxed review was strictly limited to files added in the "Banana" feature branch up to commit 0ab3f10.

Retesting activities were performed on commit 5184e5b

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

Approach

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 this review, the testing team also utilised the following automated testing tools:

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

Output for these automated tools is available upon request.

Coverage Limitations

Due to a 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 5 issues during this assessment. Categorised by their severity:

- High: 1 issue.
- Medium: 1 issue.

- Low: 1 issue.
- Informational: 2 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Polygon smart contracts. 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 contracts, including gas 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 contract(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
ZKEVM04-01	`rollbackBatches` Does Not Account For Pending Batches	High	Resolved
ZKEVM04-02	Sequencing Fees Are Not Refunded On Rollback	Medium	Closed
ZKEVM04-03	Forced Batches Can Be Censored Indefinitely With `rollbackBatches()`	Low	Closed
ZKEVM04-04	`batchToRollback` Is Not The Batch To Roll Back	Informational	Resolved
ZKEVM04-05	Miscellaneous General Comments	Informational	Resolved

ZKEVM04-01	`rollbackBatches` Does Not Account For Pending Batches		
Asset	`PolygonRollupManager.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

When rolling back to a previous batch it is not checked if that batch is pending. Additionally, none of the state for pending batches is cleared. This means that when rolling back to a batch x, variables such as `rollup.pendingStateTransitions` and `rollup.lastPendingState` may still contain values for batches larger than x.

As a result these pending states can still be used elsewhere in the contract. For example, they can be consolidated by calling `consolidatePendingState()`. This results in `lastVerifiedBatch` and `lastLocalExitRoot` being set to the rolled back pending state. Additionally a main invariant is broken as `lastVerifiedBatch` would have a larger value than `lastBatchSequenced`. Depending on how this is handled offchain it may allow for double spends.

Another example of how the stale pending states may be misused is by trivially making `_proveDistinctPendingState()` succeed.

Imagine the following scenario:

1. Assume there are x verified batches and y sequenced batches.
2. A third party verifies from x to x+10 batches, such that they are pending.
3. A rollback occurs from y to x.
4. More batches are sequenced, such that there are more than x+10 sequenced batches
5. `_proveDistinctPendingState()` can now be called and will trivially succeed. This is because the range of old pending state had different batches (with different transactions) and as a result will produce a different state root compared to the range of new batches from step 4.

Recommendations

Ensure the above comments are understood and consider adding checks to ensure that pending states can not be rolled back.

Resolution

The development team has fixed the above issue by clearing any pending state transitions when calling `rollbackBatches()`. These changes can be seen in commits 57f3184 and 5184e5b.

ZKEVM04-02	Sequencing Fees Are Not Refunded On Rollback		
Asset	`PolygonRollupManager.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

In order to sequence batches the sequencer has to pay a fee. This is the case for normal batches, sequenced by the trusted sequencer, but also for forced batches, sequenced by an untrusted third party.

However, when batches are rolled back with `rollbackBatches()` the fees are not returned to the sequencers, instead they remain in `PolygonRollupManager` and will eventually be paid out to the aggregators. As a result, if a sequencer wants to sequence their batches again, they will have paid double the sequencing fees.

Recommendations

Ensure the above comments are understood. A possible way to fix this is to store the sequencer's address and the amount of POL fees paid for every batch, this would allow for refunds during `rollbackBatches()`.

Resolution

The development team has opted to close the issue with the following comment.

> *This is an emergency mechanism, so it does not make sense to store so many variables and waste so much gas for this case.*

ZKEVM04-03	Forced Batches Can Be Censored Indefinitely With `rollbackBatches()`		
Asset	`PolygonRollupManager.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

`rollbackBatches()` can be used to censor forced batches indefinitely. For example, if a forced batch is sequenced, the admin can simply rollback to remove the forced batch. This means the user would have to call `forceBatch()` and wait for the `forceBatchTimeout` a second time, after which the admin could potentially rollback again.

Recommendations

Ensure the above comments are understood and acknowledged.

Resolution

The development team has opted to close the above issue with the following comment.

> *We are aware that the forced batches could be rolled back, and we are aware that this mechanism makes the system more centralized.*

ZKEVM04-04	`batchToRollback` Is Not The Batch To Roll Back
Asset	`PolygonRollupManager.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The parameter `batchToRollback` in `rollbackBatches()` is described in comments as the "batch to rollback". However, the batch is not rolled back at all. It is in fact the latest batch that is not rolled back.

This significantly lowers code readability and clarity, given the essential role of this distinction.

Recommendations

Consider renaming the variable to `batchToRollbackTo` or perhaps `targetBatch` and change the comment on line [**735**] to, *"rollback up to but not including this batch."*

Resolution

The development team has resolved this issue by renaming the appropriate variables and parameters, as seen in commit 57f3184.

ZKEVM04-05	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. **Incorrect Comment**

   Related Asset(s): PolygonRollupManager.sol

   The comment on line [**640**] relates to `rollupID` but the code is testing whether there are any sequenced batches that have not been verified.

   Update the comment to reflect the function of the code.

Recommendations

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

Resolution

The development team has resolved the above issue in commit 57f3184.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `forge` framework was used to perform these tests and the output is given below.

```
Ran 1 test for test/Basic.t.sol:BasicTest
[PASS] test_VariablesWork() (gas: 5466)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 218.90ms (75.65µs CPU time)

Ran 3 tests for test/PoCs.t.sol:PoCs
[PASS] test_VariablesWork() (gas: 5466)
[PASS] test_consolidateRolledBackState_PoC() (gas: 275075)
[PASS] test_proveDistinctRoot_PoC() (gas: 394193)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 224.15ms (4.26ms CPU time)

Ran 6 tests for test/PolygonZkEVMGlobalExitRootV2.t.sol:PolygonZkEVMGlobalExitRootV2Test
[PASS] test_VariablesWork() (gas: 5466)
[PASS] test_checkUpgrade() (gas: 84928)
[PASS] test_updateExitRoot_NotFromBridgeOrManager() (gas: 16143)
[PASS] test_updateExitRoot_rootAlreadyExists() (gas: 102880)
[PASS] test_updateExitRoot_updateMainnetRoot() (gas: 195598)
[PASS] test_updateExitRoot_updateRollupRoot() (gas: 195641)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 230.48ms (6.16ms CPU time)

Ran 52 tests for test/PolygonValidiumEtrog.t.sol:PolygonValidiumEtrogTest
[PASS] test_acceptAdminRole_happy() (gas: 48358)
[PASS] test_acceptAdminRole_onlyPendingAdmin() (gas: 41147)
[PASS] test_checkUpgrade() (gas: 59000)
[PASS] test_forceBatch_AboveMaxLength() (gas: 63568)
[PASS] test_forceBatch_NotDuringEmergency() (gas: 63814)
[PASS] test_forceBatch_NotEnoughPOL() (gas: 39116)
[PASS] test_forceBatch_OnlyAllowedSender() (gas: 33616)
[PASS] test_forceBatch_happy() (gas: 142010)
[PASS] test_sequenceBatchesValidium_exceedMaxBatches() (gas: 1572553)
[PASS] test_sequenceBatchesValidium_finalAccInputHashDoesntMatch() (gas: 158903)
[PASS] test_sequenceBatchesValidium_forcedBatchesMustMatch() (gas: 157141)
[PASS] test_sequenceBatchesValidium_happy() (gas: 243658)
[PASS] test_sequenceBatchesValidium_invalidL1InfoLeafIndex() (gas: 50320)
[PASS] test_sequenceBatchesValidium_invalidTimestamp() (gas: 25965)
[PASS] test_sequenceBatchesValidium_onlyTrustedSequencer() (gas: 22200)
[PASS] test_sequenceBatchesValidium_zeroBatches() (gas: 19649)
[PASS] test_sequenceBatches_exceedMaxBatches() (gas: 2517224)
[PASS] test_sequenceBatches_finalAccInputHashDoesntMatch() (gas: 153275)
[PASS] test_sequenceBatches_forcedDataMustMatch() (gas: 158835)
[PASS] test_sequenceBatches_happy() (gas: 241368)
[PASS] test_sequenceBatches_invalidL1InfoLeafIndex() (gas: 52626)
[PASS] test_sequenceBatches_invalidTimestamp() (gas: 28279)
[PASS] test_sequenceBatches_onlyTrustedSequencer() (gas: 19588)
[PASS] test_sequenceBatches_sequenceWithDataAvailabilityNotAllowed() (gas: 27188)
[PASS] test_sequenceBatches_transactionsTooLong() (gas: 1102290)
[PASS] test_sequenceBatches_zeroBatches() (gas: 19163)
[PASS] test_sequenceForceBatches_AfterEmergencyTimeout() (gas: 211487)
[PASS] test_sequenceForceBatches_BatchOverflow() (gas: 32304)
[PASS] test_sequenceForceBatches_ExceedMaxBatches() (gas: 2526895)
[PASS] test_sequenceForceBatches_ForceBatchTimeout() (gas: 134892)
[PASS] test_sequenceForceBatches_ForcedDataMismatch() (gas: 136205)
[PASS] test_sequenceForceBatches_OnlyAllowedSender() (gas: 179599)
[PASS] test_sequenceForceBatches_ZeroBatches() (gas: 29384)
[PASS] test_sequenceForceBatches_happy() (gas: 232646)
[PASS] test_setDataAvailabilityProtocol_happy() (gas: 25184)
[PASS] test_setDataAvailabilityProtocol_onlyAdmin() (gas: 15746)
[PASS] test_setForceBatchAddress_forceBatchesDecentralized() (gas: 19746)
[PASS] test_setForceBatchAddress_happy() (gas: 25437)
[PASS] test_setForceBatchAddress_onlyAdmin() (gas: 15786)
[PASS] test_setForceBatchTimeout_happy() (gas: 38028)
[PASS] test_setForceBatchTimeout_higherThanAggregationTimeout() (gas: 16351)
[PASS] test_setForceBatchTimeout_higherThanPrevious() (gas: 81856)
```

```
[PASS] test_setForceBatchTimeout_onlyAdmin() (gas: 15780)
[PASS] test_setTrustedSequencerURL_happy() (gas: 29651)
[PASS] test_setTrustedSequencerURL_onlyAdmin() (gas: 16446)
[PASS] test_setTrustedSequencer_happy() (gas: 25206)
[PASS] test_setTrustedSequencer_onlyAdmin() (gas: 15656)
[PASS] test_switchSequenceWithDataAvailability_happy() (gas: 25932)
[PASS] test_switchSequenceWithDataAvailability_onlyAdmin() (gas: 19070)
[PASS] test_switchSequenceWithDataAvailability_sameValue() (gas: 19652)
[PASS] test_transferAdminRole_happy() (gas: 42356)
[PASS] test_transferAdminRole_onlyAdmin() (gas: 15806)
Suite result: ok. 52 passed; 0 failed; 0 skipped; finished in 242.04ms (83.58ms CPU time)

Ran 21 tests for test/PolygonRollupManager.t.sol:PolygonRollupManagerTest
[PASS] test_PRMaddExistingRollup_chainAlreadyExists() (gas: 2962424)
[PASS] test_PRMaddExistingRollup_chainIdOutOfRange() (gas: 2962578)
[PASS] test_PRMaddExistingRollup_success() (gas: 3108346)
[PASS] test_PRMcreateNewRollup_chainIdAlreadyExists() (gas: 2963884)
[PASS] test_PRMcreateNewRollup_chainOutOfRange() (gas: 2961843)
[PASS] test_PRMcreateNewRollup_doesNotExist() (gas: 2961285)
[PASS] test_PRMcreateNewRollup_typeObselete() (gas: 2972140)
[PASS] test_PRMcreateNewRollup_vanilla() (gas: 3708091)
[PASS] test_PRMrollbackBatches_invalidRollbackBatch() (gas: 2969239)
[PASS] test_PRMrollbackBatches_notAdmin() (gas: 2973574)
[PASS] test_PRMrollbackBatches_notEndOfSequence() (gas: 3002956)
[PASS] test_PRMrollbackBatches_rollupDoesntExist() (gas: 3053652)
[PASS] test_PRMrollbackBatches_success() (gas: 3006166)
[PASS] test_PRMupdateRollupByRollupAdmin_NotAdmin() (gas: 2960716)
[PASS] test_PRMupdateRollupByRollupAdmin_NotCompatible() (gas: 3100625)
[PASS] test_PRMupdateRollupByRollupAdmin_Obselete() (gas: 3049861)
[PASS] test_PRMupdateRollupByRollupAdmin_RollupDoesNotExist() (gas: 3118980)
[PASS] test_PRMupdateRollupByRollupAdmin_RollupTypeDoesNotExist() (gas: 3032494)
[PASS] test_PRMupdateRollupByRollupAdmin_Success() (gas: 3072704)
[PASS] test_PRMupdateRollupByRollupAdmin_TypeTooOld() (gas: 3032230)
[PASS] test_PRMupdateRollupByRollupAdmin_unverifiedSequences() (gas: 2961068)
Suite result: ok. 21 passed; 0 failed; 0 skipped; finished in 242.52ms (62.93ms CPU time)

Ran 38 tests for test/PolygonRollupBaseEtrog.t.sol:PolygonRollupBaseEtrogTest
[PASS] test_acceptAdminRole_happy() (gas: 48336)
[PASS] test_acceptAdminRole_onlyPendingAdmin() (gas: 41103)
[PASS] test_checkUpgrade() (gas: 58791)
[PASS] test_forceBatch_AboveMaxLength() (gas: 63457)
[PASS] test_forceBatch_NotDuringEmergency() (gas: 63792)
[PASS] test_forceBatch_NotEnoughPOL() (gas: 39094)
[PASS] test_forceBatch_OnlyAllowedSender() (gas: 33623)
[PASS] test_forceBatch_happy() (gas: 141905)
[PASS] test_sequenceBatches_exceedMaxBatches() (gas: 2515151)
[PASS] test_sequenceBatches_finalAccInputHashDoesntMatch() (gas: 151147)
[PASS] test_sequenceBatches_forcedDataMustMatch() (gas: 156542)
[PASS] test_sequenceBatches_happy() (gas: 238949)
[PASS] test_sequenceBatches_invalidL1InfoLeafIndex() (gas: 50486)
[PASS] test_sequenceBatches_invalidTimestamp() (gas: 26184)
[PASS] test_sequenceBatches_onlyTrustedSequencer() (gas: 17524)
[PASS] test_sequenceBatches_transactionsTooLong() (gas: 1100192)
[PASS] test_sequenceBatches_zeroBatches() (gas: 17156)
[PASS] test_sequenceForceBatches_AfterEmergencyTimeout() (gas: 211194)
[PASS] test_sequenceForceBatches_BatchOverflow() (gas: 32259)
[PASS] test_sequenceForceBatches_ExceedMaxBatches() (gas: 2526895)
[PASS] test_sequenceForceBatches_ForceBatchTimeout() (gas: 134694)
[PASS] test_sequenceForceBatches_ForcedDataMismatch() (gas: 135985)
[PASS] test_sequenceForceBatches_OnlyAllowedSender() (gas: 179285)
[PASS] test_sequenceForceBatches_ZeroBatches() (gas: 29384)
[PASS] test_sequenceForceBatches_happy() (gas: 232135)
[PASS] test_setForceBatchAddress_forceBatchesDecentralized() (gas: 19658)
[PASS] test_setForceBatchAddress_happy() (gas: 25326)
[PASS] test_setForceBatchAddress_onlyAdmin() (gas: 15808)
[PASS] test_setForceBatchTimeout_happy() (gas: 37889)
[PASS] test_setForceBatchTimeout_higherThanAggregationTimeout() (gas: 16307)
[PASS] test_setForceBatchTimeout_higherThanPrevious() (gas: 81643)
[PASS] test_setForceBatchTimeout_onlyAdmin() (gas: 15758)
[PASS] test_setTrustedSequencerURL_happy() (gas: 29651)
```

```
[PASS] test_setTrustedSequencerURL_onlyAdmin() (gas: 16402)
[PASS] test_setTrustedSequencer_happy() (gas: 25293)
[PASS] test_setTrustedSequencer_onlyAdmin() (gas: 15764)
[PASS] test_transferAdminRole_happy() (gas: 42334)
[PASS] test_transferAdminRole_onlyAdmin() (gas: 15740)
Suite result: ok. 38 passed; 0 failed; 0 skipped; finished in 244.08ms (75.86ms CPU time)

Ran 6 test suites in 278.00ms (1.40s CPU time): 121 tests passed, 0 failed, 0 skipped (121 total tests)
```

Appendix B 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.

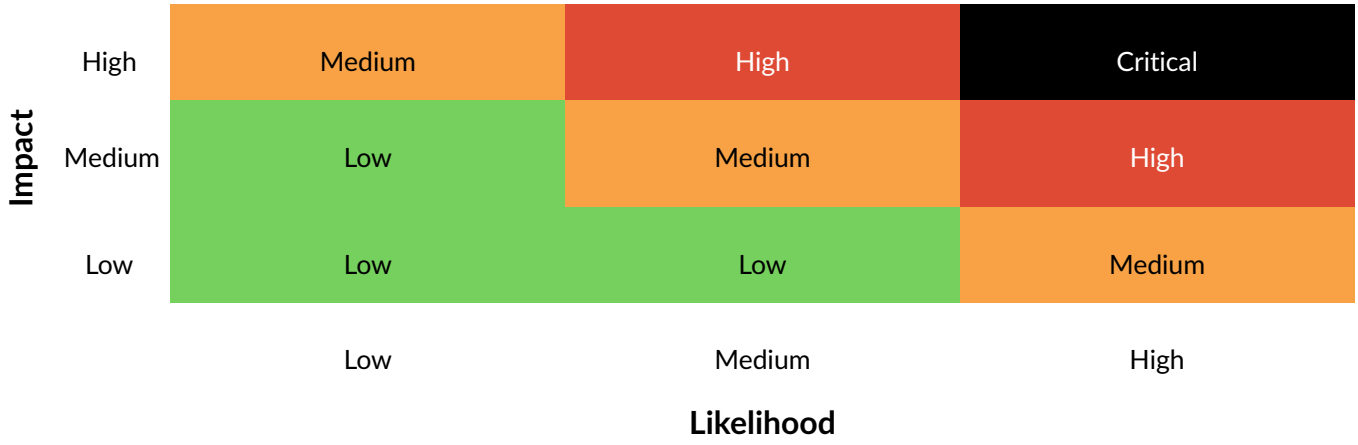

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

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].