Polygon

LX/LY Bridge - Sovereign Chains

Version: 2.0

January, 2025

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Polygon smart contracts. 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 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 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 in scope.

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. It is also supported by the Agglayer. The AggLayer operates on two fundamental principles: aggregating ZK proofs from interconnected chains and ensuring the safety of near-instant atomic cross-chain transactions.

This review focuses on changes to add support for sovereign chains. Sovereign chains are those with miscellaneous state transition functions and are secured on the Agglayer by pessimistic proofs.

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 changes in GitHub pull request 330. The fixes of the identified issues were assessed at commit f448f90.

Additionally, the update script at `deployment/v2/utils/updateVanillaGenesis.ts` was reviewed at commit a4b0c93.

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`
- Surya: `https://github.com/ConsenSys/surya`
- Aderyn: `https://github.com/Cyfrin/aderyn`

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 6 issues during this assessment. Categorised by their severity:

- Medium: 2 issues.
- Low: 1 issue.
- Informational: 3 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Polygon smart contracts 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 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
ZKEVM05-01	Sovereign Token And Origin Token May Have Different Decimals	Medium	Resolved
ZKEVM05-02	`removeLastGlobalExitRoots()` Does Not Work For Multiple Roots	Medium	Resolved
ZKEVM05-03	BridgeManager is `address(0)` When Upgrading A Bridge To `BridgeL2SovereignChain`	Low	Closed
ZKEVM05-04	`globalExitRootUpdater` Is Fixed	Informational	Resolved
ZKEVM05-05	Sovereign Chains May Not Support `PUSH0`	Informational	Resolved
ZKEVM05-06	Miscellaneous General Comments	Informational	Resolved

ZKEVM05-01	Sovereign Token And Origin Token May Have Different Decimals		
Asset	`contracts/v2/sovereignChains/BridgeL2SovereignChain.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

When mapping a sovereign token in `setSovereignTokenAddress()` there is no check that the sovereign token has the same number of decimals as the origin token. As a result, a token may be mapped to a sovereign token with different decimals.

This would result in issues during bridging operations when a token is exchanged for an equal amount of the other token. Similarly during a migration in `migrateLegacyToken()`, a wrapped token is exchanged for an equal amount of sovereign token. As a result, an attacker may drain a large amount of value from the bridge.

USDC and USDT are examples of tokens which have different decimals in different chains. These tokens have 6 decimal places on Ethereum mainnet and 18 decimal places on most other chains and L2s.

Recommendations

Consider adding a check that ensures both tokens have equal number of decimals.

Alternatively, implement logic to convert token amounts, accounting for the difference in decimals. This would allow for tokens with different decimals to still be mapped.

Resolution

The development team have acknowledged the issue and added further documentation to the smart contracts. The comments may be seen in PR #384.

Bridges are expected to use equivalent token decimals on each side of the bridge. Furthermore, the amount of tokens transferred in terms of the `uint256 amount` will not be modified. The impact will therefore be restricted to UI and third party protocols if the decimals are set differently on each chain.

ZKEVM05-02	`removeLastGlobalExitRoots()` Does Not Work For Multiple Roots		
Asset	`contracts/v2/sovereignChains/GlobalExitRootManagerL2SovereignChain.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

The `removeLastGlobalExitRoots()` function can be used to remove global exit roots from the `globalExitRootMap`. Global exit roots may only be removed in reverse order, meaning the most recent root must be removed first. To enforce this, `insertedGERCount` is cached in the memory variable `insertedGERCountCache` which is then checked and decremented for every root.

However, the use of this cache variable is incorrect. `insertedGERCount` is decremented instead of `insertedGERCountCache` on every iteration. Meaning that `insertedGERCountCache` will not change and the check on line 99 will fail on the second iteration.

As a result calling `removeLastGlobalExitRoots()` for multiple roots will always fail and roots have to be removed individually instead.

GlobalExitRootManagerL2SovereignChain.sol

```
85  function removeLastGlobalExitRoots(
        bytes32[] calldata gersToRemove
87  ) external onlyGlobalExitRootUpdater {
        uint256 insertedGERCountCache = insertedGERCount;
89      // Can't remove if not enough roots have been inserted
        if (gersToRemove.length > insertedGERCountCache) {
91          revert NotEnoughGlobalExitRootsInserted();
        }
93      // Iterate through the array of roots to remove them one by one
        for (uint256 i = 0; i < gersToRemove.length; i++) {
95          bytes32 rootToRemove = gersToRemove[i];

97          // Check that the root to remove is the last inserted
            uint256 lastInsertedIndex = globalExitRootMap[rootToRemove];
99          if (lastInsertedIndex != insertedGERCountCache) {
                revert NotLastInsertedGlobalExitRoot();
101         }

103         // Remove from the mapping
            delete globalExitRootMap[rootToRemove];
105         // Decrement the counter
            insertedGERCount--; //@audit `insertedGERCountCache` should be decremented here instead
107
            // Emit the removal event
109         emit RemoveLastGlobalExitRoot(rootToRemove);
        }
111 }
```

Recommendations

Decrement `insertedGERCountCache` instead of `insertedGERCount` on line [106], and set `insertedGERCount` to `insertedGERCountCache` at the end of the function.

Resolution

The recommendation has been implemented in PR #359.

ZKEVM05-03	BridgeManager is `address(0)` When Upgrading A Bridge To `BridgeL2SovereignChain`		
Asset	`contracts/v2/sovereignChains/BridgeL2SovereignChain.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

If an existing, already initialised, bridge is upgraded to `BridgeL2SovereignChain` the `bridgeManager` can not be set.

The issue occurs because the `initialize()` function is used to set `bridgeManager`, which will not be an option for an already initialised bridge. For this case, `bridgeManager` will always be `address(0)` and thereby prevents significant functionality of `BridgeL2SovereignChain` bridge.

For newly deployed bridges this is not an issue, since they can makes use of the `initialize()` function.

Recommendations

Consider modifying the permissions on `setBridgeManager()` or adding a new function to set `bridgeManager` to an initial value after upgrading.

If required the `reinitializer` modifier could be added to a function to handle this case.

Resolution

The development team have acknowledged the issue and provided the following response.

> Sovereign bridges should never be upgraded from non-sovereign bridges, but in case this may happen we would add the reinitialize feature in the future. We are currently very tight in terms of bytecode and adding this code would make the bytecode surpass the supported limit for deploying.

ZKEVM05-04	`globalExitRootUpdater` Is Fixed
Asset	`contracts/v2/sovereignChains/GlobalExitRootManagerL2SovereignChain.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `globalExitRootUpdater` address can not be changed after initialisation. This may cause issues in case of private key compromise.

As the `globalExitRootUpdater` can insert arbitrary global exit roots which can be used to withdraw all funds from the bridge, it is a high value role.

Recommendations

Consider implementing logic to allow changing `globalExitRootUpdater`.

Resolution

A function `setGlobalExitRootUpdater()` has been implemented to allow modifying the `globalExitRootUpdater`. Changes can be seen in PR #359.

ZKEVM05-05	Sovereign Chains May Not Support `PUSH0`
Asset	`contracts/v2/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Many of the contracts use a solidity pragma of 0.8.20. This switches the default target EVM version to Shanghai, which means that the generated bytecode will include `PUSH0` opcodes.

Sovereign chains may not support `PUSH0`, meaning that these contracts would not be deployable on these chains.

Recommendations

When compiling contracts for sovereign chains, check the appropriate EVM version and recompile if necessary.

Resolution

The development team have acknowledged the issue and will monitor EVM versions for each chain.

ZKEVM05-06	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. **Comparison With Boolean**
   Related Asset(s): contracts/v2/sovereignChains/BridgeL2SovereignChains.sol
   On line [**110**], there is a direct comparison with a boolean value. This is logically unnecessary.
   Consider removing the comparison with `true`. However, this expression might be considered clearer and more readable than the variable name on its own. As this test is part of a more complex expression, the development team might consider the current version preferable to removing it.
2. **Incorrect Comment**
   Related Asset(s): contracts/v2/sovereignChains/BridgeL2SovereignChains.sol
   The comment on line [**11**] in `BridgeL2SovereignChains` mentions that this contract, "will be deployed on Ethereum and all Sovereign chains". However, as noted by the development team, the sovereign bridge will not be deployed on Ethereum mainnet.
3. **Unclear Naming**
   Related Asset(s): `contracts/v2/sovereignChains/BridgeL2SovereignChain.sol`
   - The function `removeLegacySovereignTokenAddress()` takes a parameter `sovereignTokenAddress`. It would be clearer to rename this parameter `legacySovereignTokenAddress` to avoid any confusion with the current sovereign token address of the token.
   - The functions `activateEmergencyState()` and `deactivateEmergencyState()` revert with error `NotValidBridgeManager`, consider renaming this error to something like `EmergencyStateNotAllowed()`.
4. **Typing Errors**
   Related Asset(s): contracts/v2/PolygonZkEVMGlobalExitRootV2.sol
   On line [**81**] in `PolygonZkEVMGlobalExitRootV2.sol` the word "temporal" should be replaced with "temporary".

Recommendations

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

Resolution

All issues have been addresses in PR #359.

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 8 tests for test/tests-local/GlobalExitRootManagerL2SovereignChain.t.sol:GlobalExitRootManagerL2SovereignChainTest
[PASS] test_initialize() (gas: 19883)
[PASS] test_insertGlobalExitRoot_alreadySet() (gas: 67798)
[PASS] test_insertGlobalExitRoot_notUpdaterUpdaterSet() (gas: 20401)
[PASS] test_insertGlobalExitRoot_removalOutOfOrder() (gas: 147017)
[PASS] test_insertGlobalExitRoot_singleRemoval() (gas: 81022)
[PASS] test_insertGlobalExitRoot_success() (gas: 100248)
[PASS] test_insertGlobalExitRoot_tooManyRemovals() (gas: 95967)
Suite result: FAILED. 7 passed; 1 failed; 0 skipped; finished in 15.32ms (6.93ms CPU time)

Ran 31 tests for test/tests-local/BridgeL2SovereignChain.t.sol:BridgeL2SovereignChainTest
[PASS] test_activateEmergencyState() (gas: 26443)
[PASS] test_deactivateEmergencyState() (gas: 26371)
[PASS] test_initialize() (gas: 54745)
[PASS] test_initialize_cantReinitialize() (gas: 28565)
[PASS] test_initialize_gasTokenNetworkMustBeZero() (gas: 5662697)
[PASS] test_initialize_invalidSovereignWETHAddressParams() (gas: 5738011)
[PASS] test_initialize_wrongInitializer() (gas: 5615652)
[PASS] test_migrateLegacyToken() (gas: 1520245)
[PASS] test_migrateLegacyToken_afterRemoval() (gas: 1481227)
[PASS] test_migrateLegacyToken_alreadyUpdated() (gas: 722928)
[PASS] test_migrateLegacyToken_mintable() (gas: 1449657)
[PASS] test_migrateLegacyToken_notMapped() (gas: 1268390)
[PASS] test_removeLegacySovereignTokenAddress() (gas: 122434)
[PASS] test_removeLegacySovereignTokenAddress_onlyBridgeManager() (gas: 17906)
[PASS] test_removeLegacySovereignTokenAddress_tokenNotRemapped() (gas: 114757)
[PASS] test_setBridgeManager() (gas: 29870)
[PASS] test_setBridgeManager_invalidBridgeManager() (gas: 20403)
[PASS] test_setBridgeManager_onlyBridgeManager() (gas: 19998)
[PASS] test_setMultipleSovereignTokenAddress() (gas: 174712)
[PASS] test_setMultipleSovereignTokenAddress_invalidLength() (gas: 63799)
[PASS] test_setMultipleSovereignTokenAddress_onlyBridgeManager() (gas: 20377)
[PASS] test_setMultipleSovereignTokenAddress_zero() (gas: 22580)
[PASS] test_setSovereignTokenAddress() (gas: 85178)
[PASS] test_setSovereignTokenAddress_alreadyMapped() (gas: 78768)
[PASS] test_setSovereignTokenAddress_invalidOriginNetwork() (gas: 26677)
[PASS] test_setSovereignTokenAddress_onlyBridgeManager() (gas: 21706)
[PASS] test_setSovereignTokenAddress_repeated() (gas: 164410)
[PASS] test_setSovereignTokenAddress_zeroAddress() (gas: 35946)
[PASS] test_setSovereignWETHAddress() (gas: 7027101)
[PASS] test_setSovereignWETHAddress_noGasToken() (gas: 5690419)
[PASS] test_setSovereignWETHAddress_onlyBridgeManager() (gas: 17984)
Suite result: ok. 31 passed; 0 failed; 0 skipped; finished in 15.48ms (11.62ms CPU time)
```

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.

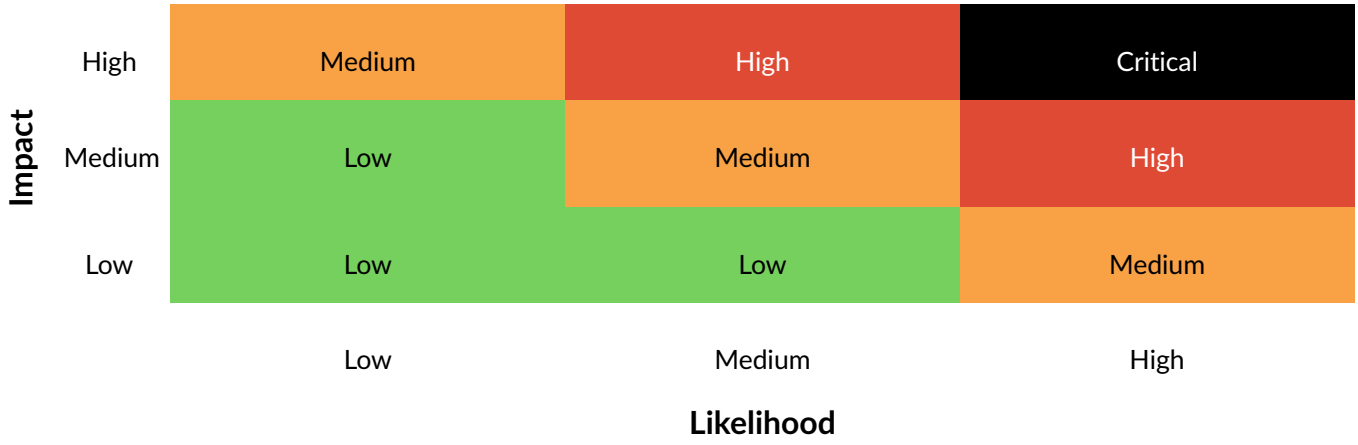

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

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].