Polygon

POL Token

Smart Contract Security Assessment

Version: 2.0

September, 2023

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. 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 Indicia is a feature in the revised Polygon protocol architecture that manages the minting and migration of a newly proposed token, `POL`. The system comprises three key contracts: `DefaultInflationManager`, `Polygon` and `PolygonMigration`.

The `DefaultInflationManager` contract oversees the inflation of `POL` tokens by permitting a 1% mint each year to specific contracts, calculated based on the total supply and the time elapsed since deployment. The `Polygon` contract represents the main `POL` token contract, allowing a 1-to-1 representation between `POL` and `MATIC`, and additional inflation based on specified requirements. It validates and mints new tokens based on the elapsed time since the last minting event, ensuring a cap on the minting rate. Lastly, the `PolygonMigration` contract handles the migration and unmigration of tokens between `MATIC` and `POL`, providing functionalities for 1-to-1 conversion, as well as locking and unlocking the reverse migration process.

Each contract in the system interacts to ensure the secure and functional management of `POL` tokens, leveraging OpenZeppelin libraries and incorporating upgradeability features for future improvements.

Security Assessment Summary

This review was conducted on the files hosted on the Polygon repository and were assessed at commit 1ea91a1.

Retesting activities were performed on commit 01b5f3e.

The list of assessed contracts is as follows:

1. `DefaultInflationManager.sol`
2. `Polygon.sol`
3. `PolygonMigration.sol`

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

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities 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 all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team used the following automated testing tools:

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

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 4 issues during this assessment. Categorised by their severity:

- Informational: 4 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
INDI-01	Sufficient `MATIC` Required For A Successful Minting	Informational	**Closed**
INDI-02	Disruption From `renounceOwnership()` In `PolygonMigration`	Informational	**Closed**
INDI-03	Reliance On Upgradeable Contract Could Cause Continuity Errors In Token State	Informational	**Resolved**
INDI-04	Miscellaneous General Comments	Informational	**Resolved**

INDI-01	Sufficient `MATIC` Required For A Successful Minting
Asset	`DefaultInflationManager.sol` & `PolygonMigration.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

A successful call to `DefaultInflationManager.mint()` requires a sufficient `MATIC` balance in the `PolygonMigration` contract.

The implementation in the `DefaultInflationManager.mint()` function indicates the need for a consistent migration from `MATIC` to `POL` tokens to ensure the `PolygonMigration` contract always has enough `MATIC` tokens to facilitate the minting process. There is no guarantee that migration would be successful and therefore minting could fail. While minting is delayed, the need for `MATIC` increases over time and will thus be more difficult to mint.

Recommendations

Confirm whether this behaviour is expected. Otherwise, consider allowing a delayed back-convert of `StakeManager` contract's `POL` to `MATIC` until `PolygonMigration` has adequate `MATIC` to facilitate the conversion.

Resolution

The development team have acknowledged this issue as expected behaviour, providing the following comments.

> This behavior is expected. As mentioned in the InflationManager (now emissionManager) tests, we anticipate Polygon ecosystem participants will be converting Matic to POL to provide sufficient one-way liquidity on PolygonMigration.sol. It is also worth mentioning that once Polygon Hub is released, all PoS operations including validator payout will occur with POL; and the current StakeManager will be deprecated. At this point, DefaultInflationManager (now DefaultEmissionManager) will be upgraded to remove the backconvert condition for StakeManager and minted POL will directly be sent to the hub in the place of StakeManager.

INDI-02	Disruption From `renounceOwnership()` In `PolygonMigration`
Asset	`Polygon.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The function `renounceOwnership()` is accessible in the `PolygonMigration` contract as it is inherited from OpenZeppelin's `Ownable2StepUpgradeable` contract.

Invoking `renounceOwnership()` will render the function `PolygonMigration.updateUnmigrationLock()` uncallable by the `Owner`, potentially disrupting the `PolygonMigration` contract's operation if `updateUnmigrationLock()` is intended to be invoked multiple times throughout the `PolygonMigration` contract's lifespan.

Recommendations

Confirm whether this behaviour is consistent with the intended functionality. If it is not, consider overriding or removing the `renounceOwnership()` function.

Resolution

The development team have acknowledged this issue, keeping the ability to renounce ownership.

INDI-03	Reliance On Upgradeable Contract Could Cause Continuity Errors In Token State
Asset	`DefaultInflationManager.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `POL` token, while non-upgradeable and adhering to best practices in ERC20 token creation, relies on a third-party upgradeable inflation manager contract. This dependence introduces a potential issue as the `DefaultInflationManager` contract, for various reasons such as governance decisions, may alter the `startTimestamp` of the staking period for `POL` tokens.

If the `startTimestamp` is changed and the `Polygon.mint()` function has already been invoked, the `POL` token will not reflect this alteration.

Recommendations

Confirm whether utilizing an upgradeable contract is advisable for `DefaultInflationManager`.

Resolution

The issue is resolved in commit 8986433, where the start time is no longer fetched from `DefaultInflationManager`. In the new implementation, `lastMint` is initially set during the constructor to `block.timestamp`. It is repeatedly updated to the current `block.timestamp` during consecutive calls to `mint()`.

INDI-04	Miscellaneous General Comments
Asset	`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 error handling in `DefaultInflationManager`**

   Related Asset(s): `DefaultInflationManager.sol`

   The function `initialize()` reverts when the caller is not `DEPLOYER` but does not revert with a custom error. Other cases of an invalid address calling a function are reverted with the custom error `InvalidAddress`.

   Consider reverting with the custom error to distinguish errors more easily.

2. **No Event on `setPolygonToken()`**

   Related Asset(s): `PolygonMigration.sol`

   Function `setPolygonToken()` is a state-changing function. However, it does not emit any event on successful transaction.

   Consider emitting an event for better tracking state changes.

3. **Potentially Misleading `immutable` Label on `inflationManager`**

   Related Asset(s): `Polygon.sol`

   The `inflationManager` address in `Polygon` is labeled as `immutable`. However, the `DefaultInflationManager` is an upgradeable contract. This causes a contradiction in terms of characteristics between immutability and upgradeability.

   Make sure appropriate documentation is provided to describe this behaviour to users.

4. **Versioning on Upgradeable Contracts**

   Related Asset(s): `DefaultInflationManager.sol`, `PolygonMigration.sol`

   Both `DefaultInflationManager` and `PolygonMigration` contracts are upgradeable. However, the current versions of these contracts do not incorporate versioning, making it challenging to determine the current implementation behind the proxy.

   Consider implementing versioning to distinguish between implementations more easily.

5. **Unused `Ownable2StepUpgradeable`**

   Related Asset(s): `DefaultInflationManager.sol`

   The current version of the `DefaultInflationManager` contract does not employ the `Owner` in any of its functions, rendering all features inherited from `Ownable2StepUpgradeable` unused. However, this could be subject to change in future iterations, given that the `DefaultInflationManager` is an upgradeable contract.

   Consider removing the parent contract if it is not currently utilized or if there are no plans to use it in the future.

6. **Upgradeable Proxy Risks in `PolygonMigration` Contract Related Asset(s): `DefaultInflationManager.sol`, `PolygonMigration.sol`**

   The `PolygonMigration` contract, while relatively straightforward in its requirements, is implemented as an upgradeable proxy contract. This introduces additional risks, such as administrator account takeover.

   It is recommended that the client accepts these risks or opts for a fixed contract with appropriate setter functions that can be managed by governance.

7. **Uncertain Token Prices Based on Market Response Related Asset(s): `PolygonMigration.sol`**

   The `PolygonMigration` contract currently assumes a 1:1 exchange rate for the migration and unmigration of `POL` to `MATIC` and vice versa. However, this assumption may not always align with the market conditions. Factors such as demand, liquidity, and overall market sentiment can lead to different token prices in the market, which may result in users preferring one token over the other, thereby questioning the validity of a 1:1 exchange rate.

   Additionally, if a significant price difference occurs between two trading pairs, for example, `MATIC/USDC` and `USDC/POL`, it could be exploited using arbitrage via flash loans. An attacker could purchase `MATIC` at a discount, trade it for `POL` using the `PolygonMigration` contract, and then trade the `POL` for `USDC` and back to `MATIC` at a discount to repay the flash loan obligations. This could potentially be used to drain all `POL` tokens in a short time.

   Ensure that this behaviour is expected and aligns with the economic model of the tokens. Possible resolutions could include the use of TWAP pricing through oracles, implementing a time delay for pulling future funds from the migration contract to prevent flash loan attacks, or reviewing and accepting this behaviour as intended if the price of POL is meant to be fixed to that of `MATIC`.

8. **Inconsistent Comments**

   Incorrect @dev Comment

   Related Asset(s): `DefaultInflationManager.sol`

   The comment on line [15] describes the following:

```
15 /// @dev The contract allows for a 1\% mint *each* per year (compounded every year) to the stakeManager and treasury
       ↪  contracts
```

   However, the actual code implementation suggests a different rate of approximately 2% annual minting. This inconsistency is reflected in the comment on line [20]:

```
20 // log2(2\%pa continuously compounded inflation per year) in 18 decimals, see _inflatedSupplyAfter
```

   And also in the comment on line [92]:

```
92 /// @dev interestRatePerYear = 1.02; 2\% per year
```

   Consider updating the comment to accurately reflect the implementation in the code.

   Inconsistent @dev Comments

   Related Asset(s): `PolygonMigration.sol`

   The migrate function includes a @dev documentation:

```
47 /// @dev The function does not do any validation since the migration is a one-way process
```

   However, this documentation is not present in the `unmigrate`, `unmigrateTo`, and `unmigrateWithPermit` functions.

   Consider updating the comments to accurately document the implementation in the code.

   Inconsistent @param Comments

   Related Asset(s): `PolygonMigration.sol`

   The `unmigrateWithPermit` function includes a @param documentation for `amount` but it is missing for `deadline`:

```
75 /// @notice This function allows for unmigrating from POL tokens to MATIC tokens using an EIP-2612 permit
   /// @param amount Amount of POL to migrate
77 function unmigrateWithPermit
```

Consider updating the comments to accurately document the implementation in the code.

Recommendations

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

Resolution

The development team have acknowledged these findings, addressing them where appropriate as follows:

1. Acknowledged, won't fix as this is only called once during `initialize()`.
2. Acknowledged, won't fix as this is only called once.
3. No longer relevant as `inflationManager` variable has been removed.
4. Fixed in commit a7bccac.
5. Acknowledged, owner functionality may be added to `DefaultInflationManager.sol` in the future.
6. Acknowledged, ownership will be given to a governance contract.
7. Acknowledged, this behaviour is expected.
8. Fixed in commits 2c6f1e1, c0c1fe8 and 4aeebb9.

Appendix A Test Suite

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

```
tests/test_DefaultInflationManager.py  ......      [24%]
tests/test_Matic.py                    .           [28%]
tests/test_Polygon.py                  ........    [60%]
tests/test_PolygonMigration.py         ..........  [100%]
```

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.

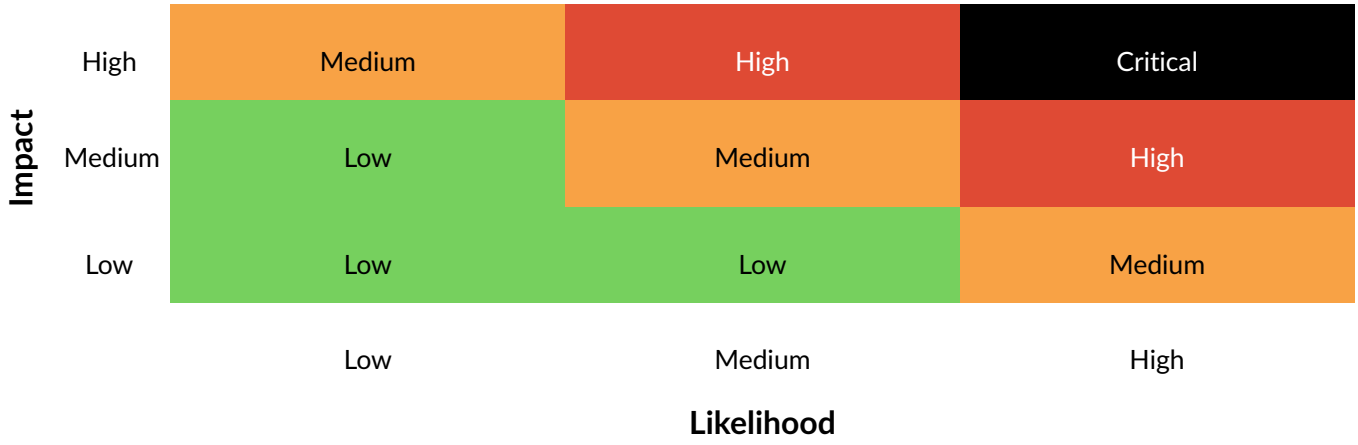

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

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].