PUBLIC

Code Assessment

of the Polygon Token (POL) Smart Contracts

October 04, 2023

Produced for

polygon

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Polygon team,

Thank you for trusting us to help Polygon with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Polygon Token (POL) according to Scope to support you in forming an opinion on their security risks.

Polygon implements the POL token, a fungible asset on Ethereum that supports the revised Polygon protocol architecture, and in particular its emission schedule and the migration from the previous MATIC token.

The most critical subjects covered in our audit are functional correctness and access control. Security regarding all the aforementioned subjects is high.

The general subjects covered are upgradeability, gas efficiency, and trustworthiness. We found that security regarding those subjects is high.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	3
• Code Corrected	2
• Acknowledged	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Polygon Token (POL) repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	18 Sept 2023	01b5f3e0a5f7b9eb7af742aa8dd9ada2b6ec00cd	Initial Version
2	02 Oct 2023	0c5d06179f17edfc34d81ba44ddd9bdb1b2f5420	Second Version
3	04 Oct 2023	40fff9047712e8e18e05ab7bcea92a100a452751	Third Version
4	04 Oct 2023	a780764684dd1ef1ca70707f8069da35cddbd074	Fourth Version

For the solidity smart contracts, the compiler version `0.8.21` was chosen.

The following contracts are in the scope of the review:

```
src/interfaces:
    IDefaultEmissionManager.sol
    IPolygonEcosystemToken.sol
    IPolygonMigration.sol
src:
    DefaultEmissionManager.sol
    PolygonEcosystemToken.sol
    PolygonMigration.sol
```

2.1.1 *Excluded from scope*

Any contracts not explicitly listed above are out of the scope of this review. Third-party libraries are out of the scope of this review. The library `PowUtil` is out of the scope of this review. We performed fuzz testing on the `exp2` function with inputs similar to the ones used by the `DefaultEmissionManager`, i.e., `0.028569152196770894e18 * secondsElapsed / 365 days`, with `secondsElapsed` up to 1000 years. This gave us confidence that the computation done in the `DefaultEmissionManager` is correct within a 10^{-5} absolute precision bounds.

2.2 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

The subsequent versions of the codebase fix issues found during this assessment but do not alter the behaviors of the system described in this overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Polygon offers the Polygon ecosystem token which will be the native token for Polygon 2.0 and the successor to the MATIC token. To achieve the transition to POL, a migration contract will be used to wrap/unwrap MATIC into/from POL, and a token emission contract will control the emission rate of POL.

The `PolygonMigration` and `DefaultEmissionManager` contracts will be deployed behind transparent upgradeable proxies.

2.2.1 PolygonEcosystemToken

This contract defines the POL ERC20 token which has built-in integrations for EIP-2612 and Permit2. The initial supply is 10 billion tokens to match the MATIC supply, but more tokens can be minted through emissions.

The contract uses role-based access control for permissioned operations. An address with the default administrator role can grant and remove roles to other addresses. Custom roles are:

- `EMISSION_ROLE`: can mint a limited amount of tokens per unit of time. It is expected to be the `DefaultEmissionManager`.
- `CAP_MANAGER_ROLE`: can set the emissions limit. It is expected to be the governance.
- `PERMIT2_REVOKER_ROLE`: can control the universal allowance granted to Permit2.

The `DEFAULT_ADMIN_ROLE` is expected to be the governance.

The emissions are bounded by the `mintPerSecondCap` parameter. It is set to 10 POL per second by default and can be updated arbitrarily by an address granted the `CAP_MANAGER_ROLE`. When `mint()` is called, the number of tokens that can be minted is capped by an amount relative to the time delta since the last mint event, i.e., `(block.timestamp - lastMint) * mintPerSecondCap`.

If `permit2Enabled` is set to true (the default), the Permit2 contract has unlimited allowance from any account. If it is disabled, then the default behavior applies, and individual accounts can choose to set an allowance for the Permit2 contract.

2.2.2 DefaultEmissionManager

This contract defines the fine-grained emission policy for the POL token. It allows the minting of 2% of the total supply of POL, compounding. The minted amount is distributed 50/50 to the out-of-scope stake manager and treasury contracts. The stake manager receives the emission in the form of MATIC tokens through POL unmigration.

The unmigration can be done only if there is enough MATIC token in the migration contract and if the unmigration is unlocked. The first case is assumed to be met by Polygon that specified:

> we anticipate Polygon ecosystem participants will be migrating MATIC to POL in order to provide sufficient one-way liquidity on PolygonMigration.sol

Unmigration is expected to be locked, the current Stake Manager to be deprecated, and `DefaultEmissionManager` to be upgraded when Polygon Hub goes live. Polygon specified:

> Once Polygon Hub is released, all PoS operations including validator payout will occur with POL; the current StakeManager will be deprecated. At this point, DefaultEmissionManager will be upgraded to remove the back convert condition for StakeManager and minted POL will directly be sent to the hub in the place of StakeManager. An expected timeline for this is a year.

2.2.3 PolygonMigration

This contract allows users to exchange MATIC for POL and vice versa at a 1-to-1 rate. The contract is ownable, and the owner can enable and disable the POL-to-MATIC conversion (unmigration) at will. The owner can also burn POL tokens by sending them to the dead address, so the total supply is not impacted.

The migration contract is assumed to be initialized during the proxy deployment.

2.2.4 Trust Model

Users of the system are generally untrusted and expected to behave unpredictably.

The following roles are fully trusted and expected to behave honestly and correctly.

- The administrator, the cap manager, and the Permit2 revoker of the POL token contract.
- The owner of the migration contract.
- The addresses granted the pauser role in the MATIC contract, as if MATIC is paused, migration/unmigration is locked.

Moreover, the `EMISSION_ROLE` of the `PolygonEcosystemToken` is assumed to be granted to the `DefaultEmissionManager`.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

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

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1

- ERC165 Partially Implemented Acknowledged

5.1 ERC165 Partially Implemented

Design Low Version 1 Acknowledged

CS-POLTOKEN-001

The contract `AccessControlEnumerable` implements ERC165 but the contract `PolygonEcosystemToken` that inherits `AccessControlEnumerable` in the codebase does not extend the implementation of ERC165. ERC165 should either return `true` for all the interfaces the contract implements or be completely disabled.

Acknowledged:

Polygon answered that:

> PolygonEcosystemToken is planned to only support AccessControlEnumerable interface and ERC20 Permit but it isn't industry practice for it to extend ERC165 so far.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	2

- Missing Input Sanitization Code Corrected
- Interfaces Are Missing Functions Code Corrected

6.1 Missing Input Sanitization

Design Low Version 2 Code Corrected

CS-POLTOKEN-008

The respective arguments of the constructors of `PolygonMigration` and `DefaultEmissionManager` are not ensured to be non-zero.

Code corrected:

The constructors have been updated to check that the addresses are non-zero.

6.2 Interfaces Are Missing Functions

Design Low Version 1 Code Corrected

CS-POLTOKEN-002

Some of the interfaces are missing functions from their implementations that could be useful for integrators. Here is a non-exhaustive list:

1. `IPolygonEcosystemToken` is missing `updatePermit2Allowance()` and the getter functions for the storage variables
2. `IPolygonMigration` is missing `getVersion()`, `burn()`, and the getter functions for the storage variables
3. `IDefaultEmissionManager` is missing `getVersion()`, `inflatedSupplyAfter()`, and the getter functions for the storage variables

---

Code corrected:

The interfaces have been updated to expose all relevant functions.

6.3 Storage Gap Inconsistency

Informational Version 3 Code Corrected

CS-POLTOKEN-009

In Version 3, the upgradable contracts __gap variables were updated with the intention that exactly 50 storage slots are used by each contract. In `PolygonMigration`, __gap was set to be 48 slots long as the contract contains two storage variables. However, as the storage variables are packed into one slot by the compiler, to have exactly 50 storage slots, __gap should have size 49.

Code corrected:

__gap was updated to have size 49.

6.4 Gas Optimizations

Informational Version 1 Code Corrected

CS-POLTOKEN-004

1. In the contract `DefaultEmissionManager`, the storage variables `migration`, `stakeManager`, and `treasury` can be set at deployment and thus be `immutable`. The initialization would only need to set `token`. This would allow to reduce the number of `SLOAD` performed upon minting.
2. In the function `DefaultEmissionManager.mint()`, the storage variable `token` can be loaded in memory to avoid multiple `SLOAD`.
3. In the contract `PolygonMigration`, the storage variable `matic` can be set at deployment and be `immutable`, thus saving storage reads.

Code corrected:

All optimizations have been applied.

6.5 Inconsistent Event Emission

Informational Version 1 Code Corrected

CS-POLTOKEN-005

In the codebase, events are mostly emitted before a state change. However, the functions `PolygonEcosystemToken._updatePermit2Allowance` and `PolygonMigration.updateUnmigrationLock` are not following that pattern.

Code corrected:

The two functions have been updated to emit events before performing state changes.

6.6 `Initializable` Is Inherited Twice

Informational Version 1 Code Corrected

CS-POLTOKEN-007

The contract `DefaultEmissionManager` inherits `Initializable` directly and also from `Ownable2StepUpgradeable`. Direct inheritance is not necessary.

Code corrected:

The `DefaultEmissionManager` now only inherits `Initializable` once.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Emission Manager Owner Has No Purpose

Informational Version 1 Acknowledged

CS-POLTOKEN-003

Although the contract `DefaultEmissionManager` is ownable and an owner must be given when calling `initialize`, the owner has no specific permissions and the role is never used in the contract.

Acknowledged:

Polygon is aware of this and explained that they want to proactively keep `Ownable` for now.

7.2 Planned Emissions Conflict With Supply Cap

Informational Version 1 Acknowledged

CS-POLTOKEN-006

In the `DefaultEmissionsManager`, the planned supply increase is 2% of the supply per year, compounding. The `PolygonEcosystemToken` contract enforces that no more than 10 tokens per second can be emitted. After about 22 years and 41 days, the manager will try to mint more tokens than what the cap allows, and the transaction will revert. To resume emissions, admin action will be needed to increase the cap.

Acknowledged:

Polygon acknowledged and stated:

> According to the current plan, it is expected that emissions will stop after 10 years (and hence all fuzz tests are done with a 10-year bound).

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Approval Events Do Not Reflect Permit2 Allowance

Note **Version 1**

The implicit infinite allowance granted to the Permit2 contract is invisible to applications that rely on `Approval()` events to track token allowances. Furthermore, when it is enabled or disabled, they will not be notified either since this action raises a different event.

Nothing can be done on-chain about this since the ERC-20 interface is not designed to support allowances on behalf of all users.