PUBLIC

Code Assessment

of the Plasma Bridge Update Smart Contracts

December 11, 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 Plasma Bridge Update according to Scope to support you in forming an opinion on their security risks.

Polygon implements an update to the Polygon Plasma Bridge, which enables the migration of bridged Matic to POL.

The most critical subjects covered in our audit are functional correctness, asset solvency, access control, and event handling. There was one issue regarding functional correctness which has been addressed, see DepositBulk reverts for POL. Security regarding the other subjects is high.

The general subjects covered are code complexity and gas efficiency. Security regarding code complexity is good, as only minimal changes were introduced to implement the new functionality, while the majority of the code is unchanged. Slight improvements to gas efficiency were pointed out and implemented, see Gas Optimizations.

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	1
• Specification Changed	1
Low-Severity Findings	1
• Code Corrected	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 Plasma Bridge Update repository based on the documentation files.

This review was targeted at Pull Request #483. And only the following file was considered in scope:

- contracts/root/depositManager/DepositManager.sol

The review was done with the understand that this is part of a legacy codebase, where it is prioritized to make only minimal changes.

The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	09 November 2023	a7d91442c6c22a0463fcf58203fb9efbd22c2cef	Initial Version
2	11 December 2023	e6a5e745bc50a6c82bbb5f0db64505d69e01a263	After Fixes

For the solidity smart contracts, the compiler version `0.5.17` was chosen.

2.1.1 Excluded from scope

Any other file not explicitly mentioned in the scope section. In particular tests, scripts, external dependencies, and configuration files are not part of the audit scope.

This review only covers the extended functionality of the DepositManager, hence other components of the Polygon Plasma Bridge, such as proof verification, are not part of the review.

2.2 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment 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 an update to the DepositManager of the Polygon PoS Plasma Bridge, which enables:

- Migration of Matic tokens already held by the DepositManager to POL (Polygon Ecosystem Token).
- When users withdraw native Matic from Polygon PoS, they will now receive POL on Ethereum instead of Matic.
- When users deposit Matic to Polygon PoS, the tokens locked in the DepositManager will be converted to POL. However, the user will still receive native Matic on Polygon PoS.
- When users deposit POL to Polygon PoS, they will receive native Matic instead of POL.

2.2.1 Plasma Bridge

Polygon Plasma Bridge enables bridging ERC20 and ERC721 tokens between Ethereum and Polygon PoS network.

To bridge tokens to Polygon PoS:

- The user deposits the tokens to the DepositManager.
- The bridged amount should be within an upper limit `maxErc20Deposit`.
- Only tokens that are mapped by the `governance` in the `registry` can be bridged.
- The deposited tokens will be locked in the DepositManager. A `Deposit` event will be emitted in `stateSender.syncState()`.
- Polygon validators listen for events from the stateSender contract.
- `onStateReceive()` will be invoked on the child chain through a special system address, which mints the mirrored token to the receiver.

To withdraw tokens back to Ethereum:

- The user burns their tokens via `withdraw()`, which emits a `Withdraw` event.
- After the checkpoints of the Polygon State have been updated on the root chain, the user can initialize a withdraw from the `predicate` contract (`startExitWithBurntTokens()`), which verifies the inclusion of the receipt for the `Withdraw` event in the checkpoint and adds a new withdrawal to the queue (`addExitToQueue()`) in the WithdrawManager, if the verification succeeds.
- In case the withdrawal is not successfully challenged during the challenge period, the withdrawal can be completed via `WithdrawManager.processExits()`, which invokes `predicate.onFinalizeExit()`. Eventually, `DepositManager.transferAssets()` is invoked by the authorized predicate, which sends the tokens to the user.

DepositManager also has a public `updateChildChainAndStateSender()` function, which should be called immediately to update the cached values whenever they change in the registry.

2.2.2 Extended Functionality

Polygon has deployed a migration contract, which functions as a 1:1 converter between Matic and POL tokens. Users can convert Matic to POL in a permissionless way. Converting POL to Matic (`unmigrate()`) could be paused/unpaused by its owner.

The implementation of DepositManager has been updated to facilitate the migration from Matic to POL tokens on Ethereum when using the Plasma Bridge. The ERC20 token Matic will be migrated to POL on Ethereum, while the native token stays Matic on Polygon PoS.

An external function, `migrateMatic()` with its internal version `_migrateMatic()` has been added. It migrates an amount of Matic that is locked in the DepositManager to POL. This function is restricted to `governance`. It approves the amount to the migration contract and calls `migrate()`, where the Matic will be transferred to the migration contract and POL will be transferred to the DepositManager.

`transferAssets()` has been adjusted to always transfer POL to the user upon a Matic withdrawal.

The logic of `_createDepositBlock()` has been adjusted to handle Matic and POL deposits:

- In case the deposited token is Matic, the Matic received will be swapped to POL via the migration contract. The user will still receive native Matic on Polygon PoS.
- In case the deposited token is POL, the emitted token address (`_token`) will be changed to Matic, which will result in the user receiving native Matic tokens on Polygon PoS.

2.2.3 Roles and Trust Model

The DepositManager is deployed as a proxy, whose implementation can be upgraded by its owner. In addition, the owner can update the root chain contract address (`updateRootChain()`).

The DepositManager has another privileged role called governance, with privileges to:

- Pause (`lock()`) or unpause (`unlock()`) the DepositManager.
- Migrate an amount of Matic held by the DepositManager to POL (`migrateMatic()`).
- Update the max ERC20 deposit amount (`updateMaxErc20Deposit()`).

The DepositManager relies on the Registry as a source of truth to function correctly. The Registry is fully controlled by its governance, which has the privileges to:

- `updateContractMap()`: update the address given a key.
- `mapToken()`: register a pair of root and child tokens and define if the token is an ERC721.
- Add (`addErc20Predicate()`/`addErc721Predicate()`) or remove (`removePredicate()`) a predicate given a token.

The migration contract is also deployed as an upgradable proxy, whose owner has the privileges to:

- Upgrade the implementation.
- Set POL token address.
- Disable unmigration (converting POL to Matic).
- Burn POL tokens by sending them to a specific address.

The StateSender has an owner role, which has the privileges to:

- register a new contract, allowing them to emit `StateSynced` events, which will be received by the ChildChain contract.

This review was based on the following assumptions about these roles:

1. The DepositManager (proxy) owner is fully trusted not to upgrade to a bad implementation contract. In the worst case, they can drain all funds held by the DepositManager.
2. The governance of the DepositManager is assumed to be honest. In the worst case, they can indefinitely pause bridge deposits. However, this could be overwritten by the proxy owner changing the implementation contract.
3. The governance of the Registry is assumed to be honest. In the worst case, they can drain all funds from DepositManager.
4. The owner of the migration contract is fully trusted. In the worst case, they can change the implementation of `migrate()`, which could leave the DepositManager with fewer POL than needed to honor all bridge withdrawals. They can also drain all Matic/POL in the migration contract, which would likely make those tokens held by the DepositManager worthless.
5. The owner of the StateSender is trusted not to register bad contracts. In the worst case, this could lead to minting an unlimited number of native Matic on the Polygon PoS chain. These could be bridged back, leading to the DepositManager becoming insolvent.

Tokens with non-standard behaviors, such as rebasing tokens, tokens with transfer fees, and tokens that do not return `true` on transfers are not expected to be used. The system is also subject to the potential risks of upgradability, blacklisting, pausing, and freezing of deposited tokens.

It is further assumed that the updates will be executed atomically.

2.2.4 Changes in Version 2

In Version 2 it was decided that the POL Token will be mapped in the registry using `mapToken()`. This removes the need for exceptions to the `isTokenMapped()` modifier. The side-effect of this change is that there will no longer be a 1-to-1 mapping between child and root tokens. `rootToChildToken` of both MATIC and POL on Ethereum will return the native MATIC token's address. However, `childToRootToken` of native MATIC will now only return the POL address, whereas it returned MATIC's address before.

An extra action item has been added to the upgrade process, making it 4 in total:

1. call `updateContractMap()` on the Registry to add “matic”, “pol” and “polygonMigration”
2. call `mapToken()` on the Registry to map POL to native MATIC
3. deploy new DepositManager version and update proxy
4. call `migrateMatic` on the new DepositManager, migrating all MATIC

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
- Correctness: Mismatches between specification and implementation

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	0

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	1

- depositBulk Reverts for POL Specification Changed

Low-Severity Findings	1

- Gas Optimizations Code Corrected

6.1 depositBulk Reverts for POL

Correctness Medium Version 1 Specification Changed

CS-POLPLSM-001

In `depositBulk()`, it is checked that the provided tokens are mapped, using `isTokenMappedAndIsErc721()` from the registry contract. This function will revert if a token is not mapped.

```
// will revert if token is not mapped
if (_registry.isTokenMappedAndIsErc721(_tokens[i]))
```

As the POL token is handled as a special case, it will not be mapped in the registry. As a result, the check will revert and the entire `depositBulk()` call will revert.

Specification changed:

It has been decided that POL will be mapped in the registry using `mapToken()`. As a result, `depositBulk()` will no longer revert.

This also removes the need for exceptions to the `isTokenMapped()` modifier.

The side-effect of this change is that there will no longer be a 1-to-1 mapping between child and root tokens. `rootToChildToken` of both MATIC and POL on Ethereum will return the native MATIC token's address. However, `childToRootToken` of native MATIC will now only return the POL address, whereas it returned MATIC's address before.

`childToRootToken` is only used in the deprecated ERC20Predicate contract, which is no longer in use. It has been replaced with ERC20PredicateBurnOnly, which does not rely on `childToRootToken`. As a result, this change should have no impact on the Plasma Bridge.

6.2 Gas Optimizations

Design Low Version 1 Code Corrected

The function `_migrateMatic` computes the keccak256 of `polygonMigration` and calls `registry.contractMap()` twice to load the same address. It could be optimized by cacheing the first call's return value. This would save one keccak256 and one external call.

The same optimization can be applied in `_createDepositBlock()`, where keccak256 and `registry.contractMap()` will be invoked twice if `_token` is POL.

Code corrected:

Return values from getting addresses from the registry are now cached in `_migrateMatic()` and `_createDepositBlock()`.

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 Atomic Deployment

Informational Version 1 Acknowledged

CS-POLPLSM-002

The deployment of the updates contains the following steps:

1. Set the new entries (address of POL, Matic, and polygonMigration) correctly in the registry.
2. Update the implementation of the DepositManager Proxy.
3. Migrate all its existing Matic to POL by the new governance restricted function `migrateMatic`.

The deployment is expected to be executed atomically. Otherwise, an exit of Matic tokens between Step.2 and Step.3 may fail due to insufficient POL balance. Consequently, the funds will be locked since the exit message has been deleted from the exit queue.

Acknowledged:

Polygon acknowledged and responded:

```
Extra attention will be paid to the fact that the DepositManager update and
“migrateMatic” should happen in the same transaction.
This will avoid any potential risk of the DepositManager not being adequately funded.
```

In Version 2 an extra action item has been added to the upgrade process, making it 4 in total:

1. call `updateContractMap()` on the Registry to add “matic”, “pol” and “polygonMigration”
2. call `mapToken()` on the Registry to map POL to native MATIC
3. deploy new DepositManager version and update proxy
4. call `migrateMatic` on the new DepositManager, migrating all MATIC

7.2 depositBulk Does Not Enforce maxERC20Deposit

Informational Version 1 Acknowledged

CS-POLPLSM-004

The DepositManager has a variable named `maxERC20Deposit`, which can be set by the governance. In `depositERC20ForUser()`, this max is enforced. However, when depositing ERC20 tokens using `depositBulk()`, the limit is not enforced.

It is unclear why the limit is necessary in the first place, but the behavior is inconsistent.

Acknowledged:

Polygon is aware, but has decided not to make a change, in order to keep the number of legacy code changes minimal.

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 Cross-Chain Messages Must Not Revert

Note Version 1

It is important to note that messages are only executed once. If the execution on the target chain reverts, the message is lost and cannot be retried.

For example, if `transferAssets()` was called for Matic while the DepositManager has insufficient POL tokens to cover the full withdrawal, the call will revert. The message would be lost and could not be retried, even if the DepositManager had enough tokens at a later time. Also, if any of the registry functions in `transferAssets()` were to revert, messages would be lost. Another revert reason could be insufficient gas.

The full amount of Matic tokens that are in circulation on the Polygon PoS chain should always be held as POL on the DepositManager, so it should be impossible for the situation to occur where `transferAssets()` reverts due to insufficient balance.

Also, note that messages from Ethereum to Polygon behave the same way. If the execution of `onStateReceive()` reverts (for example due to insufficient gas), the message is lost.

8.2 Matic Withdrawals Require Controlling the Same Address on Ethereum

Note Version 1

The Matic token on Polygon PoS can be bridged back to Ethereum using the `withdraw` function. It always uses `msg.sender` as the receiver on Ethereum. There is no `withdrawTo` function.

Users of smart contract wallets, such as Multisig Safes, must ensure that they are able to make transactions from the address they are withdrawing from on both chains. This is guaranteed for EOAs, but usually not the case for smart contract wallets.