PUBLIC

Code Assessment

of the OP Token Bridge Smart Contracts

October 09, 2024

Produced for

MAKER

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

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

MakerDAO implements a custom token bridge between Ethereum and L2s based on the OP stack.

The most critical subjects covered in our audit are functional correctness, access control and the integration with the OP stack's messaging infrastructure. The general subjects covered are error handling, trustworthiness and specification. Security regarding all the aforementioned 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	0

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 OP Token Bridge 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	02 Sep 2024	bae891c15067738eae18baa4866f7972289b7841	Initial Version
2	18 Sep 2024	a01b8725f20896390e63a6f65b109c25f8fb823c	Fileable Escrow
3	08 Oct 2024	0f935505c0dc74ce3db2a9998320a56119321814	Upgradeable & L2 Withdrawal Limits

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

The following contracts and deployment scripts were in the scope of this review:

```
./src/Escrow.sol
./src/L1GovernanceRelay.sol
./src/L1TokenBridge.sol
./src/L2GovernanceRelay.sol
./src/L2TokenBridge.sol

./deploy/L1TokenBridgeInstance.sol
./deploy/L2TokenBridgeInstance.sol
./deploy/L2TokenBridgeSpell.sol
./deploy/TokenBridgeDeploy.sol
./deploy/TokenBridgeInit.sol
```

2.1.1 Excluded from scope

All other files and the correctness of the external systems are out of scope. Specifically, the OP stack is out of scope and assumed to work correctly as documented. The tokens that are bridged are expected to be standard ERC-20 tokens (e.g. no rebasing, no fees, no call-on-transfer) that conform to the required interfaces (e.g. support MakerDAO's `rely` / `deny` authentication).

2.2 System Overview

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

At the end of this report section we have added subsections for each of the changes accordingly to the versions.

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

MakerDAO offers a set of OP Token Bridge contracts, which enable users to deposit supported tokens to an OP Stack L2 and withdraw back to Ethereum (L1).

2.2.1 Escrow

When tokens are bridged from L1 to L2, the tokens will be locked in the escrow contract. When they are bridged back, the bridge will pull funds from the escrow to send them to the respective users.

Thus, the escrow features the standard `rely()` and `deny()` authorization functions to configure the `wards` which have the privilege to grant any token allowance to any spender with `approve()`.

2.2.2 Token Bridges

The `L1TokenBridge` and the `L2TokenBridge` contracts are the L1 and L2 entry points for bridging supported tokens to L2 and L1 respectively. They are bound to each other as `otherBridge`.

Both implement MakerDAO's common access control mechanism with `rely` and `deny` to (de-)authorize addresses. Authorized addresses can

- use `rely` and `deny`,
- close the bridge with `close` (closing one bridge only deactivates sending messages with it; hence, winding down the outflow of tokens from L1 to L2),
- and add support for tokens with `registerToken` (registers an L1-to-L2-token-mapping).

To bridge tokens, a user can call `bridgeERC20`, where the recipient will be the `msg.sender`, or `bridgeERC20To` with a designated recipient. The process works as follows:

1. `bridgeERC20` or `bridgeERC20To` is called.
2. Tokens are handled on the sending chain.
3. The bridge contract sends a message to the other bridge with `sendMessage` through the respective `CrossDomainMessenger`.
4. Eventually, the message arrives on the other layer.
5. Eventually, `finalizeBridgeERC20` is called on the other bridge (on the destination chain).
6. Tokens on the other side are handled.

Note that on L1, tokens are moved to the escrow when being sent to L2 and moved from the escrow when being received from L2. On L2, tokens are minted when being received from L1 and are burned when being sent to L1.

2.2.3 Governance Relay

The `L1GovernanceRelay` allows for sending messages with `relay` to the `L2GovernanceRelay`. Upon arrival of the message on L2, it will have its `relay` function called, which executes a governance spell via a `delegatecall`. Note that the `L1GovernanceRelay` features the standard `rely` and `deny` authorization functions and that only the wards can call `relay` on L1.

2.2.4 Contracts Deployment & Initialization

`deployL1()` will deploy `L1GovernanceRelay`, `Escrow`, and `L1TokenBridge` wired to the correct `L1CrossDomainMessenger`, `L2TokenBridge`, and `L2GovernanceRelay`. Then, the deployer sets a designated owner (expected to be `MCD_PAUSE_PROXY`) as the only authorized address for all three contracts.

`deployL2()` will deploy `L2GovernanceRelay`, `L2TokenBridge`, and `L2TokenBridgeSpell` wired to the correct `L2CrossDomainMessenger`, `L1TokenBridge`, and `L1GovernanceRelay`. Then, the deployer will `rely()` the `L2GovernanceRelay` on `L2TokenBridge` and `deny()` itself.

Note that some addresses need to be pre-computed by the deployer.

`TokenBridgeInit` will do some sanity checks on `L1TokenBridge` and `L1GovernanceRelay` to ensure the correct deployment. Then, it will register L1 and L2 token pairs on the `L1TokenBridge`. Eventually, it triggers a call to `init` on the `L2TokenBridgeSpell` (via the governance relay) which performs some sanity checks and registers tokens.

`L2TokenBridgeSpell` is a reusable L2 spell which provides the following functionalities:

- `rely()` and `deny()` to grant or revoke `wards` roles on the L2 bridge.
- `close()`: winds down the L2 token bridge.
- `registerTokens()`: registers L1 and L2 token pairs and grants the `wards` role of the L2 tokens to the L2 Bridge for authorized minting.
- `init`: ensures the correct L2 contract state with some sanity checks and then registers L1 and L2 token pairs.

2.2.5 Changes in Version 2

The `escrow` of `L1TokenBridge` is now modifiable with `file`. Hence, the `escrow` can be changed. Note that changing the escrow needs to be done carefully (e.g. governance moving funds from old escrow to the new one) as otherwise bridging back to L1 may revert.

2.2.6 Changes in Version 3

The L2 bridge has a maximum withdrawal amount per token that limits the amount bridgeable from L2 to L1 per call to `bridgeERC20()` or `bridgeERC20To` for a given token. The limit can be set by authorized addresses with `setMaxWithdraw()`.

While the bridging mechanism has been upgradeable in previous versions, the bridge contracts are now additionally made upgradeable by leveraging the UUPSUpgradeable library (EIP-1822 pattern with EIP-1967 proxy storage slots). Note that upgrades must consider messages sent with the previous version. As a consequence, an initializer function `initialize()` is provided along with a getter for the implementation `getImplementation()` and a `version` (currently `"1"` but should be changed with upgrades). Note that only authorized addresses can upgrade the contracts.

The deployment and initialization scripts have been adjusted accordingly to deploy the proxies with the respective implementation, to include additional sanity checks, to publish the L1 bridge implementation on Chainlog and to set the L2-to-L1 withdrawal limits. Additionally, the L2 spell now offers `setMaxWithdraws()` (function batching calls to the bridge's `setMaxWithdraw()`) and `upgradeToAndCall()` (upgrading the bridge).

2.2.7 Trust Model & Roles

The system defines the following key roles:

1. Users: Untrusted.

2. Governance (wards): Fully trusted and notably controls the escrow holding and the rights to upgrade. The Escrow, L1TokenBridge, and L1GovernanceRelay are expected to have `MCD_PAUSE_PROXY` as the only `wards`. The L2TokenBridge is expected to have the L2GovernanceRelay as the only `ward`.
3. OP Stack: Fully trusted. For example, if the OP stack misbehaves, the contracts could be DoSed or funds could be stolen.

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 our findings. The findings are split into these different categories:

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 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.

6.1 Redundant Parameter

Informational **Version 1** **Acknowledged**

CS-OPBRIDGE-001

The `L2TokenBridgeSpell.init` function takes `l2GovRelay_` as a parameter. However, given that the spell is executed by the `L2GovernanceRelay` with a `delegatecall`, the parameter should equal to `address(this)`. Ultimately, the parameter is redundant and gas could be saved on initialization.

Acknowledged:

MakerDAO prefers passing it as a parameter to mirror L1 spells that explicitly specify the pause proxy.

7 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.

7.1 Deployment Verification

Note Version 1

Since deployment of the contracts is not performed by the governance directly, special care has to be taken that all contracts have been deployed correctly. While some variables can be checked upon initialization through the `PauseProxy`, some things have to be checked beforehand.

We therefore assume that the initcode, bytecode, traces and storage (e.g. mappings) are checked for unintended entries, calls or similar. This is especially crucial for any value stored in a mapping array or similar (e.g. could break access control, could lead to stealing of funds).

7.2 Legacy Functionality Not Implemented

Note Version 1

Note that the token bridge does not implement OP legacy functionality. Hence, front-ends and integrators should not rely on that functionality. Further, logic for `gasPayingToken` is not implemented as it is not needed.

7.3 Optimism Deployment

Note Version 1

Governance should be aware that the deployment script does not consider the old escrow or governance relays (`OPTIMISM_ESCROW` as well as `OPTIMISM_GOV_RELAY` and its L2 counterpart).

Thus, in case governance wants to reuse these contracts for a potential deployment on Optimism, the scripts need to be adapted.