PUBLIC

Code Assessment

of the Arbitrum Token Bridge Smart Contracts

July 10, 2024

Produced for

by

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 Arbitrum 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 Arbitrum that supports the bridging of multiple tokens.

The most critical subjects covered in our audit are functional correctness, access control and the integration with Arbitrum'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 Arbitrum 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	01 Jul 2024	6248966bd8dac6261fb296f9743a0b9432cfec71	Initial Version
2	09 Jul 2024	c3c60c2fcd870d48d6886c187c7cabb38c22ab76	After Intermediate Report

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

The files in scope were:

```
deploy/
    L2TokenGatewayInstance.sol
    L2TokenGatewaySpell.sol
    TokenGatewayDeploy.sol
    TokenGatewayInit.sol

src/
    L1TokenGateway.sol
    L2TokenGateway.sol
```

2.1.1 Excluded from scope

All other files are not in scope. Arbitrum and its messaging are not in scope and are expected to work correctly. 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). Additionally, other interacted with contracts are out of scope (e.g. escrow, governance relays).

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.

Since Endgame introduces a set of new tokens to the system, MakerDAO introduces a more generalized Arbitrum token bridge to support multiple tokens (e.g. DAI, NST, sNST, ...).

2.2.1 Token Bridge

Following Arbitrum's token bridging design with token gateways, gateways on L1 (Ethereum Mainnet) and on L2 (Arbitrum One) gateways are deployed. `L1TokenGateway` and `L2TokenGateway` are deployed on L1 and L2 respectively.

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

1. use `rely` and `deny`,
2. close the bridge with `close` (closing one gateway only deactivates sending messages to its counterpart),
3. and add support for tokens with `registerToken` (registers an L1-to-L2-token-mapping).

Sending funds from L1 to L2 does the following:

1. A user calls `outboundTransfer` or `outboundTransferCustomRefund` on `L1TokenGateway`.
2. Funds from the user are pulled an moved to an `escrow` contract.
3. A retryable ticket to call `finalizeInboundTransfer` is generated through Arbitrum's inbox (sends a message).
4. Eventually, the message arrives on Arbitrum.
5. `finalizeInboundTransfer` is called on the `L2TokenGateway`.
6. The L2 token is minted with `mint`.

Sending funds from L2 to L1 is similar and is described below:

1. A user calls `outboundTransfer` (one of the two available functions) on `L2TokenGateway`.
2. Funds from the user are burned with `burn`.
3. A message to call `finalizeInboundTransfer` on the L1 gateway is sent through the ArbSys contract.
4. Eventually, the message arrives on L1.
5. `finalizeInboundTransfer` is called on the `L1TokenGateway`.

6. The bridged token is moved from the escrow to the user. Furthermore the code is designed to support calls through the Arbitrum Router. When a call originates from the router contract set, the data is processed accordingly to extract the address to receive the tokens. Note that Arbitrum-specific details are omitted and, hence, some differences are not evident from the descriptions (e.g. sending messages from L1 to L2 requires paying fees in ETH on L1).

2.2.2 Deployment and Initialization

The deployment is intended to be performed with the `TokenGatewayDeploy` script (see Notes for considerations). The following actions are performed:

- L1: `deployL1Gateway` script is intended for usage on `L1` and deploys `L1TokenGateway` and gives an `owner` (expected to be MCD_PAUSE_PROXY).
- L2: `deployL2Gateway` script is intended for usage on `L2` and deploys the `L2TokenGateway` and gives an `owner` (expected to be MakerDAO's governance relay on L2 ("ARBITRUM_GOV_RELAY")). Further, it deploys a reusable spell contract `L2TokenGatewaySpell` described below.

`L2TokenGatewaySpell` is a contract implementing a set of functions intended to be delegatecalled by the governance relay as part of bridged governance spells. Namely, it implements the following functionality

- `rely` and `deny` to call `rely` and `deny` on the `L2TokenGateway`, respectively.
- `close` to call `close` on the `L2TokenGateway`.
- `registerTokens` to register a batch of tokens on the `L2TokenGateway` with calls to `registerToken`. Further, note that the gateway is authorized for each L2 token (so that tokens can be minted).
- `init` to intialize the L2 (similar to `registerTokens` but with sanity checks).

The initialization is implemented in function `initGateways` defined in `TokenGatewayInit` and performs the following:

1. A set of sanity checks on the `L1TokenGateway` and the parameters
2. Registration of the L1 tokens and escrow allowance management
3. Relaying a governance message through the L1 side of the governance relay so that the L2 side delegatecalls into `init` of the `L2TokenGatewaySpell`

Eventually, the message is relayed and `init` is used (recall that `init` performs the sanity checks on L2).

2.2.3 Trust Model and Roles

The system defines the following key roles:

1. Users: Untrusted.
2. Governance: Fully trusted and notably controls the escrow holding the funds. Within the contracts in scope, governance has the capability to, for example, temporarily map another token to legitimate L2 tokens (or L1 tokens) in order to steal funds.
3. Arbitrum: Fully trusted. If Arbitrum or its contracts misbehave, tokens could be stolen or arbitrarily minted.

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:

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	0
Low-Severity Findings	0
Informational Findings	1

- Redundant Initialization Parameter Code Corrected

6.1 Redundant Initialization Parameter

Informational Version 1 Code Corrected

CS-MKRArbBr-001

The configuration of `TokenGatewayInit.initGateways` contains the L2 gateway's address as `cfg.counterpartGateway`. However, the `l2GatewayInstance` passed to the function is referring to the same address. Ultimately, the address is passed twice to the function.

Code corrected:

`cfg.counterpartGateway` has been removed and `l2GatewayInstance.gateway` is used.

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 Expiry of Retryable Tickets

Note Version 1

Users should be aware that the automatic execution of L1-to-L2 messages may fail. In such cases, the retryable ticket must be executed manually. However, the ticket may expire (~1 week).