PUBLIC

Code Assessment

of the Optimism DAI Bridge Smart Contracts

July 7, 2021

Produced for

MAKER

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Derek,

First and foremost we would like to thank MakerDAO for giving us the opportunity to assess the current state of their Optimism DAI Bridge system. This document outlines the findings, limitations, and methodology of our assessment.

Our assessment did not uncover severe issues. The code is clean and short. The communication with MakerDAO was always professional.

We are happy to receive questions and feedback to improve our service and are highly committed to further support your project.

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	3

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 Optimism DAI Bridge repository at https://github.com/makerdao/optimism-dai-bridge and based on the documentation files provided by MakerDAO. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	2 June 2021	e5ba6e18f589e978a7d74b8e0d0ca4499b49c262	Initial Version
2	29 June 2021	58798012527603f5af2b4cbbd2825c64cb546781	Version 2
3	7 July 2021	aece6f7697972e1950dad3cfab79d49027fca332	Version 3

For the solidity smart contracts, the compiler version `0.7.6` was chosen. For the contracts that need to be deployed on the Optimism chain, the corresponding version of Optimism solidity compiler is assumed.

2.1.1 *In Scope*

The following contracts were reviewed as part of the Initial Version:

- `dai.sol`
- `L1GovernanceRelay.sol` & `L2GovernanceRelay.sol`
- `L1Escrow.sol`

The following contracts were reviewed in Version 2:

- `L1DAITokenBridge.sol` & `L2DAITokenBridge.sol`

All fixes are based on the contracts in Version 3.

2.1.2 *Excluded from scope*

All other solidity contracts in the repository are out of scope, including the inherited contracts based on the OVM libraries provided by Optimism. A non-complete list of these exclusions is:

- The OVM compiler
- The contracts inherited from OVM
- The contracts that are part of the Optimism chain core infrastructure.

3 System Overview

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

The smart contracts reviewed are part of the DAI to Optimism Token Bridge.

3.1 Contracts

3.1.1 L1Escrow

The DAI tokens are locked in a separate Escrow contract in order to facilitate updates of the `L1Gateway` should the need arise. The Optimism L2 solution is quite recent and breaking changes requiring the update of contracts are expected.

The `L1Gateway` contract ensures DAI tokens are transferred to the Escrow while depositing.

Apart from access control, this contract implements an approve function where the `L1Gateway` can be approved to spend DAI on behalf of the escrow. This is required for withdrawals of DAIs from L2 back to L1 DAI.

The call to approve on the DAI token of L1 assumes the call always succeeds or reverts which holds given the original DAI Token contract.

3.1.2 L1GovernanceRelay & L2Governance

Maker's governance works with so called spells, which are calls to be executed.

Through the `L1GovernanceRelay.relay()` function whitelisted wards can forward a call to be executed on L2 through the `L2GovernanceRelay` contract. Such a call is executed as `DELEGATECALL` on L2. Using `DELEGATECALL` means that the call is executed in the storage context of the `L2GovernanceRelay` contract. There are safeguards in place in order to ensure that the important state variables of the contract cannot be changed. Optimism prevents through static code analysis that no contract containing the `SELFDESTRUCT` instruction can be deployed, preventing that the `L2GovernanceRelay` can be deleted with a misbehaving `DELEGATECALL`.

3.1.3 DAI

This contract represents DAI on L2. This DAI token offers additional functionality and is not a copy of the DAI contract as known from L1.

3.2 Roles

`Wards` of the individual contracts: Trusted roles expected to behave honestly and correctly at all times.
`Messenger`: part of Optimism on L1 and L2. Fully trusted to relay messages eventually.

4 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 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 associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. 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 the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviours other than initially determined in the business specification.

5 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 severities. 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.

6 Findings

In this section, we describe any open findings. Findings that have been resolved, have been moved to the Resolved Findings section. All of 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	0

7 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	3

- Burn Function Redundant Checks Code Corrected
- Contract L2GovernanceRelay Unnecessary Statefulness Code Corrected
- Init Function of L2GovernanceRelay Code Corrected

7.1 Burn Function Redundant Checks

Design Low Version 1 Code Corrected

In Dai contract, some gas savings are possible.

The safeMath operator _sub can be removed in `totalSupply = _sub(totalSupply, value);`, because of the check:

```
uint256 balance = balanceOf[from];
require(balance >= value, "Dai/insufficient-balance");
```

Code corrected:

The redundant check was removed. Similar check in `mint` function was found and removed by MakerDAO team themselves.

7.2 Contract L2GovernanceRelay Unnecessary Statefulness

Design Low Version 1 Code Corrected

The L2GovernanceRelay contract defines the `l1GovernanceRelay` field and inherits from the `messenger` field from OVM_CrossDomainEnabled.

Those 2 fields could be declared as immutable as they are never changed after the initial assignment. The L1GovernanceRelay address can be precomputed and passed to L2GovernanceRelay as a constructor variable.

Code corrected:

The `l1GovernanceRelay` is an immutable field now. The `messenger` is still a storage field, because changing it will require a change in the Optimism contracts library, that are out of scope for this assessment.

7.3 Init Function of L2GovernanceRelay

Design Low Version 1 Code Corrected

The function `init` for L2GovernanceRelay contract is needed to set the `l1GovernanceRelay` field. This function is not protected by any access modifier and can be called by anyone. The attacker can potentially call this function himself and ruin the deployment. Such attack will require the redeployment of L2GovernanceRelay contract and potentially of the L1GovernanceRelay. In addition the attacker can find a potential transaction that will revert the optimism history to such extend, where the L2GovernanceRelay deployment has happened, but init hasn't. This way attacker can init contract again himself, effectively getting a full control over the L2GovernanceRelay.

Code corrected:

The L2GovernanceRelay now has only a constructor, where the immutable `l1GovernanceRelay` is set.

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 Constructors Marked Public

Note **Version 1**

L2DAITokenBridge.sol and dai.sol contracts have constructor with public modifier. This visability modifier will be ingnored by the solidity compiler.