PUBLIC

Code Assessment

of the zkSync DAI Bridge Smart Contracts

March 20, 2023

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 zkSync DAI Bridge according to Scope to support you in forming an opinion on their security risks.

MakerDAO implements a layer 2 DAI contract for zkSync 2.0, a ZK-rollup for Ethereum, along with DAI bridging contracts. That also includes contracts for sending governance spells from layer 1 to layer 2.

The most critical subjects covered in our audit are the functional correctness of the DAI bridging mechanism, the L2-DAI ERC-20 contract and the relay of governance spells, protection against censorship, and upgradeability.

Security regarding all other aforementioned subjects is high. However, users should be aware of the trust model, see Trust Model & Roles.

The general subjects covered are upgradeability, error handling, trustworthiness, documentation, and testing. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a good 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	1
• Code Corrected	1
High-Severity Findings	0
Medium-Severity Findings	2
• Code Corrected	2
Low-Severity Findings	4
• Code Corrected	3
• Specification Changed	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 zkSync DAI Bridge repository based on the documentation files:

- `l1/L1DAITokenBridge.sol`
- `l1/L1Escrow.sol`
- `l1/L1GovernanceRelay.sol`
- `l2/dai.sol`
- `l2/L2DAITokenBridge.sol`
- `l2/L2GovernanceRelay.sol`

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

V	Date	Commit Hash	Note
1	06 Oct 2022	aaf9e329554d36b8cb544b12ecafa80741c25bd4	Initial Version
2	07 Nov 2022	ab1e06a669f744493cb742b24954a0452f6d14b6	Second Version
3	21 Nov 2022	5fa63ab1025d20105ad8f86164852aed0263a027	Third Version
4	19 Jan 2023	b5b113446c015e666e6aac39fb012d548aa0bd61	Fourth Version
5	08 Mar 2023	875a1381df5d1d3f080e315c73116a8812817494	Fifth Version
6	10 Mar 2023	0a2409f5a94a01467d4ee782f1e0c0bee3b8cb2d	Sixth Version
7	20 Mar 2023	1ac9448ab3f7adc645f5aec58aad4c004252d280	Seventh Version

For the solidity smart contracts, the compiler version `0.8.15` was chosen.

2.1.1 Excluded from scope

Any file not listed above and third-party libraries were outside the scope of this code assessment. zkSync 2.0 and its internal logic are excluded from scope.

zkSync 2.0 is still in alpha and deployed solely on testnet. Note that changes, fixes and improvements are to be expected. The main part of this review took place in the middle of October 2022. This review cannot account for future changes and possible bugs in zkSync 2.0.

2.2 System Overview

The set of Solidity contracts implement a bridge for DAI from Ethereum on layer 1 to the zkSync 2.0 layer 2 solution and vice versa. It follows the concept of other similar DAI bridges from L1 to other L2 solutions.

Following contracts are deployed on L1:

- L1DAITokenBridge

- L1Escrow
- L1GovernanceRelay

Following contracts are deployed on L2:

- dai
- L2DAITokenBridge
- L2GovernanceRelay

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

2.2.1 L2 DAI

On the L2 side, a DAI contract with functionality similar to an ERC20 token is deployed. All default ERC20 functionality such as `transfer`, `transferFrom`, `approve`, `allowance`, `balanceOf` and `totalSupply` are implemented. Additionally, the DAI contract implements `mint` which is used by the token bridge to mint tokens after a deposit from L1. The `burn` function which can be used by anyone to burn his tokens is used by the bridge contract during withdrawal of DAI from L2 back to L1. In order to mitigate the approval problem of the default ERC20 approve function, the contract implements `increase_allowance()` and `decrease_allowance()`. The L2 DAI token contract also implements the `permit` functionality along with ERC1271, so users can sign and contracts can approve an allowance. Finally, a magic number used as amount during approval works as truly unlimited approval, for callers with this magic number as approved amount the approved amount is not reduced.

2.2.2 Bridge

The `L1DAITokenBridge` contract implements all logic to deposit and withdraw DAI to and from L2. The deposited DAI are held by the `L1Escrow` contract. This separation of the funds from the logic makes the system upgradable: the `L1DAITokenBridge` logic contract can be replaced independently of the Escrow holding the DAI.

Similarly, the `L2DAITokenBridge` contract implements the logic for handling deposits and withdrawals. Note that on L2 the bridge can mint and burn DAI and no escrow is needed on L2. Also, the bridge on L2 can be upgraded through a governance spell through the `L2GovernanceRelay` contract that could remove the bridge's minting right and assign it to a new bridge.

The bridge may be paused independently on L1 and L2. Once a bridge contract is paused, it cannot be unpaused. Pausing the L1 bridge contract means deposits from L1 are no longer possible. Pausing the L2 bridge contract prevents new withdrawals from being initiated. Finalizing withdrawals on L1 that have been initiated before the L2 bridge contract was paused remains possible even when the L1 bridge contract is paused. Similarly, even when the L2 bridge contract is paused, deposit requests from L1 can still be finalized.

Depositing DAI from L1 works as follows:

1. First the user approves the `L1DAITokenBridge` contract to transfer the amount of DAI.
2. Next the user calls `L1DAITokenBridge.deposit()` passing the amount, the source of the funds and the destination address for the DAI on L2.
3. After some time, the zkSync 2.0 bridge automatically process the transaction on L2 and mints the DAI for the specified address

Withdrawing DAI from L2 works as follows:

1. First the user approves the `L2DAITokenBridge` contract to transfer the amount of L2 DAI.
2. The user executes `withdraw()` on the `L2DAITokenBridge` contract. The user specifies the amount and the L1 address that will receive the funds.

3. After a period of time, the funds can be withdrawn through the L1 bridge by presenting the message with a proof, whose validity is checked by the zkSync 2.0 bridge. By calling `L1DAITokenBridge.finalizeWithdrawal()` with the correct parameters, the funds are withdrawn. Note that anyone can perform this action for any user.

A user may have multiple outstanding withdrawals pending at the same time.

If a deposit fails on L2, the user can use `claimFailedDeposit()` with a proof of failure on L2 to get its DAI back.

2.2.3 Governance Relay

The governance relay contract on L2 is a ward, a privileged account, in all L2 contracts of the system. The MakerDAO governance on L1 can decide to execute any action on L2 using this contract. Technically this works as follows: A new contract is deployed on L2 featuring code to execute. On L1 the governance decides to execute this action and the call is relayed via the `L1GovernanceRelay` contract. On L2 this triggers the execution of the specified contract's code as delegatecall in the contract of the `L2GovernanceRelay` contract.

2.3 Trust Model & Roles

We assume the deployment and initialization (e.g., of the ward roles in order for the `L1DAITokenBridge` to be able to transfer DAI from the Escrow during withdrawal, for the `L2DAITokenBridge` to be able to mint DAI on L2 or the GovernanceRelay) to be done correctly before the bridge becomes operational.

- **User**: Users are fully untrusted.
- **Wards**: Accounts holding the ward role in a contract of the maker system have access to all privileged functionalities of this contract. As required the other contracts of the system and the Maker governance have the ward role in the contracts. These parties are assumed to act honestly and correctly at all times.
- **zkSync 2.0**: Trusted given the reasons below.

More specifically, we assume the following about zkSync 2.0:

1. We assume that any message sent from layer 1 to layer 2 will result in a message from layer 2 to layer 1 to confirm the execution as documented here.
2. We assume that layer 1 to layer 2 messages will be placed in the priority queue.
3. We assume that the priority queue dequeuing follows the rules specified.

```
- All transactions are processed sequentially.
- The operator must do at least X amount of work (see below) on the priority queue or the priority queue should be empty.
```

4. We assume that eventually, all users will always be able to call `requestL2Transaction` on layer 1 to queue transactions in the priority queue. Currently, this function has whitelisting and hence zkSync is trusted as long as the whitelisting exists.
5. We assume that in case the operator fails to follow the rules in 3., priority mode is activated and that anyone in that scenario can activate it.
6. We assume that during priority mode anyone can stake to become an operator and that there will be at least one party capable of doing so.
7. We assume that interfaces will not change when interacting with the Mailbox.
8. We assume that the zkSync 2.0 system will not be upgraded maliciously.
9. We assume that no contract is force-deployed to an address so that access control can be bypassed on layer 2.
10. We assume that zkSync 2.0 will only accept correct proofs.

11. We assume that (only) failed L2 transactions can be proven with `IMailbox.proveL2LogInclusion()` when `value=bytes32(0)`.

Having these assumptions broken may result in:

- Potential draining of the escrowed DAI on layer 1 in case of zkSync 2.0 becoming malicious or working incorrectly. However, that could be prevented in some cases when reaction time is high and the escrow's approvals are removed.
- Blocked funds due to upgrades of zkSync 2.0 and censorship of DAI users. MakerDAO governance has the ability to close the bridge (2 days delay), preventing any further deposits to zkSync.

2.3.1 Changes in V2

The ergs limit is not a constant anymore. An authorized `file()` function in the L1 bridge contract is introduced so that governance can update the value. For the L1 governance relayer, the ergs limit is passed as an additional argument to `relay()`. Further, `factoryDeps` was added as an additional parameter was added to `relay()`.

2.3.2 Changes in V3

More comments and documentation have been added.

2.3.3 Changes in V4

The constants `SYSTEM_CONTRACTS_OFFSET` and `L1_MESSENGER_CONTRACT` have been copied from zkSync 2.0 `IL1Messenger` into `L2DAITokenBridge` to avoid importing the whole file and its dependencies. More comments and documentation have been added.

2.3.4 Changes in V5

The contracts have been updated to match the new ZkSync API.

2.3.5 Changes in V6

The `msg.sender` is aliased in `_refundRecipient` if the message sender is not `tx.origin` .

2.3.6 Changes in V7

The contracts have been updated to match the new ZkSync API.

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	1

- Outdated Interfaces Code Corrected

High-Severity Findings	0

Medium-Severity Findings	2

- Refund Recipient Is Not Aliased Code Corrected
- Critical Tests Missing Code Corrected

Low-Severity Findings	4

- Inconsistent Use of Interfaces Code Corrected
- Lack of Documentation Specification Changed
- Remaining TODO in the Source Code Code Corrected
- Unused enum Code Corrected

6.1 Outdated Interfaces

Correctness Critical Version 1 Code Corrected

The zkSync interfaces for `L2Log` and `L2Message` have been updated and the ones used in the DAI bridge current codebase are deprecated. The malformed L2 logs or messages would block any attempt of withdrawal or claim of a failed deposit.

Code corrected:

The structs used correspond now to the most recent version of the zkSync 2.0 structs.

6.2 Refund Recipient Is Not Aliased

Design Medium Version 5 Code Corrected

On ZkSync the contract addresses are aliased using the `AddressAliasHelper.applyL1ToL2Alias` function, to distinguish between `L1` and `L2` initiated transactions. However, the refund recipient in the `Mailbox.requestL2Transaction` call in the `L1DAITokenBridge` contract doesn't alias the `msg.sender` address, even if it is a contract address.

Code corrected:

Refund recipient address is aliased if the `msg.sender` is a contract.

6.3 Critical Tests Missing

Design Medium Version 1 Code Corrected

Some critical tests are missing in the test suite, for example, the e2e test for claiming a failed deposit is incomplete.

Code corrected:

After Matter Labs provided the necessary sdk functions to generate the proof required by claimFailedDeposit a test case was added.

6.4 Inconsistent Use of Interfaces

Design Low Version 1 Code Corrected

When `L1DAITokenBridge` calls `finalizeDeposit`, it uses the `L2DAITokenBridgeLike` interface. The `L2DAITokenBridge` implements `IL2Bridge`, but `IL2Bridge` and `L2DAITokenBridgeLike` are not connected.

Code corrected:

MakerDAO uses the `IL2Bridge` interface now and has removed the `L2DAITokenBridgeLike` interface.

6.5 Lack of Documentation

Design Low Version 1 Specification Changed

The main functionality is sufficiently documented. However, the interaction with zkSync 2.0 remains undocumented. This is of high importance as zkSync 2.0's documentation is incomplete. Furthermore, the emergency shutdown process remains undocumented.

Specification changed:

Natspec documentation was added.

6.6 Remaining `TODO` in the Source Code

Design Low Version 1 Code Corrected

There is a leftover `TODO` comment in the code of `L1DAITokenBridge`.

Code corrected:

The TODO was removed.

6.7 Unused `enum`

Design | Low | Version 1 | Code Corrected

There is an unused `QueueType` enum in the file `L1GovernanceRelay.sol`.

Code corrected:

MakerDAO has removed the unused `enum`.

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 `zksolc` Not Up-To-Date

Informational **Version 1**

The version of the compiler that currently used is `1.3.3`, at the time of writing the latest compiler version is `1.3.5`.

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 Contract Address Aliasing Maps to Non-Operational Addresses

Note Version 6

The contract deployment processes are different on zkSync and Ethereum mainnet, this discrepancy has as an effect that two similar contracts deployed by the account will not have the same address on L1 and L2, even considering address aliasing.

The `L1DAITokenBridge` specifies the `msg.sender` (or it's alias for smart contracts) as the refund recipient.

The contracts that plan to use the `L1DAITokenBridge` need to be able to access the refunded funds. While the refunded funds will be credited on L2, the L1 contracts should be able to call zkSync bridges to spend those funds.