PUBLIC

Code Assessment

of the Starknet Teleport Smart Contracts

June 21, 2022

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 Starknet Teleport according to Scope to support you in forming an opinion on their security risks.

The smart contracts implement an extension to integrate the Starknet DAI Bridge into Teleport which facilitates fast transfers of DAI between different L2/L1 called "domains".

The most critical subjects covered in our audit are functional correctness, security and the users control over their own funds without having to trust third parties more than necessary.

While the contracts overall implement the same functionality as their counterpart for Optimism/Arbitrum, the implementation and interfaces exposed differ.

Security regarding all the aforementioned subjects is high as the issues reported have been resolved.

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	2
• Code Corrected	2
Low-Severity Findings	2
• Code Corrected	2

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 Starknet Teleport 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	09 June 2022	b6233cfdbd3068fa58c1951d38901f883e39a8be	Initial Version
2	15 June 2022	3d3385cd16f516a8e70709fbc7910ba517d0b0a7	Second Version
3	20 June 2022	33b2989e55ec68c59b3c4e8f3df2638ea07c1fa4	Third Version

For the solidity smart contracts, the compiler version `0.8.13` was chosen. In the second version, the compiler version `0.8.14` was chosen. For the cairo smart contracts, the compiler version `0.8.1` was chosen. For the second version, the compiler version `0.9.0` was chosen.

The only files in scope were:

- L1DAITeleportGateway.sol
- l2_dai_teleport_gateway.cairo

2.1.1 *Excluded from scope*

All files not listed above.

The Starknet DAI Bridge has been reviewed as part of another review.

DSS-Teleport (formerly known as DSS-Wormhole) has been reviewed as part of another review.

For the purpose of this audit, the StarkNet bridge and the corresponding contracts are assumed to work correctly. They are not part of this review.

For the cairo contracts, the focus was on the source code files inside the repository. The imported functionality from `starkware.starknet.common.*`, `starkware.cairo.common.*` as well as the built-ins have not been reviewed in depth, generally these are assumed to work as described.

StarkNet Alpha has been released recently and has not been audited yet. StarkWare states that changes, fixes and improvements are to be expected. The main part of this review took place at the beginning of June 2022.

2.2 System Overview

The smart contracts implement an extension to integrate the Starknet DAI Bridge into Teleport which facilitates fast transfers of DAI between different L2/L1 called "domains".

For details about the Starkware DAI Bridge or Teleport (formerly known as DSS-Wormhole) please refer to their respective reviews.

While the contracts implement the same functionality as their counterpart for Optimism/Arbitrum, their behavior is not identical. Most notably, `initiate_teleport()` does not immediately trigger the crosschain message but stores the `teleport`. If needed, the cross chain message must be triggered manually using `finalize_register_teleport()`. Furthermore the exposed interface differs slightly: There is only one entrypoint to `initiate_teleport()` and `flush()` returns the amount of DAI flushed.

On Starknet, this extension exposes three functions:

- `initiate_teleport()`: Allows users to initiate transfers of their DAI to another domain.
- `finalize_register_teleport()`: Allows to relay the teleport information via the starknet L2 - L1 messaging. To be used if for any reason no attestation can be obtained from the oracles. Note that this is different from Teleport for Optimism and Arbitrum where the message is always sent.
- `flush()`: Initiates the settlement of the DAI "teleported" via the starknet L2 - L1 messaging, bridges the DAI.

On L1 Ethereum:

- `finalizeRegisterTeleport():` Allows users to finalize the relay of teleport information via the starknet L2-L1 messaging initiated using `finalize_register_teleport()` on L2.
- `finalizeFlush()`: Finalizes the settlement of the bridged DAI.

We assume that domains, represented as *felt* on Starknet, do not end with `0x00` such that L1 domain strings, short right-padded strings represented as `bytes32`, can be correctly converted to left-padded `bytes32` strings to match the domain string sent from L2 to L1.

2.3 Trust Model & Roles:

The trust model defined in the reports for DSS-Teleport / Starknet-DAI-Bridge apply.

More specificly, for the contracts in scope:

Wards: For every contract, each address set to `1` in any of the `wards` mapping is fully trusted and expected to behave correctly.

Users are generally untrusted.

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:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- 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	2

- file() Has No Access Control Code Corrected
- finalize_register_teleport() Only Possible When Open Code Corrected

Low-Severity Findings	2

- No Event for finalize_teleport() Code Corrected
- Unused Code Code Corrected

6.1 `file()` Has No Access Control

Security Medium Version 1 Code Corrected

Function `file()` can add and remove valid domains and should be called carefully by governance. However, the function lacks access control.

Code corrected:

Access control was added to `file()`.

6.2 `finalize_register_teleport()` Only Possible When Open

Correctness Medium Version 1 Code Corrected

Calling `finalize_register_teleport()` on Starknet allows users to take the slower route through Starknet message passing to L1 to prevent censorship by the oracles and ensure availability if oracles are down. However, given that the function is not callable if the bridge is closed, the system, in contrast to the Teleport for Optimism and Arbitrum, is not fully trustless, as users could be censored by closing the teleport instance. Further, censored users that had their L2 DAI burned, will not be able to recover it.

Code corrected:

The precondition of `finalize_register_teleport()` has been removed.

6.3 No Event for `finalize_teleport()`

Design Low Version 2 Code Corrected

`finalize_teleport()` emits no event in contrast to other functions (e.g. `initiate_teleport()`). Emitting more events could lead to a better user-experience and easier integration with front-ends.

Code corrected:

An event has been added.

6.4 Unused Code

Design Low Version 1 Code Corrected

l2_dai_teleport_gateway prepares a payload in `initiate_teleport()` even though it remains unused.

l2_dai_teleport_gateway has several unused imports:

- BitwiseBuiltin
- hash2
- assert_le
- is_not_zero
- get_contract_address
- uint256_lt
- uint256_check

Code corrected:

Unused imports and unused code were removed.