PUBLIC

Code Assessment

of the xchain-helpers
Smart Contracts

July 24, 2024

Produced for

SPARK

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

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

SparkDAO implements a library for cross-chain message passing along with contracts able to receive cross-chain messages.

The most critical subjects covered in our audit are integration with the supported bridges, access control and functional correctness. The general subjects covered are unit testing, documentation and trustworthiness. 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	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 xchain-helpers 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	13 July 2024	e1f9443cfa7c2b13771c32cd4c89f21576210bb6	Initial Version

For the solidity smart contracts, any compiler version from `0.8.0` up to, but not including `0.9.0` is supported due to the contracts implementing a general crosschain-messaging helper library.

The contracts below are in scope:

```
forwarders:
    AMBForwarder.sol
    ArbitrumForwarder.sol
    CCTPForwarder.sol
    OptimismForwarder.sol
receivers:
    AMBReceiver.sol
    ArbitrumReceiver.sol
    CCTPReceiver.sol
    OptimismReceiver.sol
```

2.1.1 Excluded from scope

Generally, all contracts not mentioned above are out of scope. The correctness of the underlying bridging mechanisms is out of scope.

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.

SparkDAO offers a library for cross-chain messaging between Ethereum Mainnet and L2s. Note that these are helper contracts.

The contracts can be categorized into two types:

- **Forwarders**: Internal library contracts that act as a wrapper for underlying message-passing mechanism.

- **Receivers**: Contracts intended to be deployed on the respective chain to allow for receiving messages on L2 so that arbitrary calls can be made to a fixed target contract.

2.2.1 Forwarders

The `ArbitrumForwarder` implements:

- `sendMessageL1toL2`: Sends a message to the respective Arbitrum chain through its corresponding Inbox (invokes `createRetryableTicket`). Note that both Arbitrum One and Nova are supported (selected by passing inbox argument accordingly). Note that excess gas is burned and no L2 call value is supported.
- `sendMessageL2toL1`: Sends a message from an Arbitrum chain to Ethereum through the `ArbSys` system contract (invokes `sendTxToL1`).

The `OptimismForwarder` implements:

- `sendMessageL1toL2`: Sends a message to the respective Optimism chain through its corresponding cross-chain messenger (invokes `sendMessage`). Note that multiple Optimism chains are supported (selected by passing the cross-chain messenger argument accordingly).
- `sendMessageL2toL1`: Sends a message from an Optimism chain to Ethereum through the L2-side of the cross-chain messenger (invokes `sendMessage`).

The `AMBForwarder` implements:

- `sendMessage`: Generic function that sends a message to the chain on the other side of the bridge through the arbitrary-message passing bridge (invokes `requireToPassMessage`). Note that this function supports multiple AMB bridges (selected by passing destination argument accordingly).
- `sendMessageEthereumToGnosisChain`: Specialized function routing from Ethereum to Gnosis through the Ethereum-to-Gnosis AMB.
- `sendMessageGnosisChainToEthereum`: Specialized function routing from Gnosis to Ethereum through the Gnosis-to-Ethereum AMB.

The `CCTPForwarder` implements:

- `sendMessage` (two variants: address as `address` or as `bytes32`): Sends a message to the chain with the given domain ID through Circle's CCTP bridge (invokes `sendMessage`). Note that this function supports multiple CCTP bridges (selected by passing the destination ID accordingly).

2.2.2 Receivers

Receivers can receive cross-chain messages. Typically, they receive a call from the integrated with bridge, validate the origin (access control) and perform a call on a given target contract. The following receivers are implemented:

- `ArbitrumReceiver`: Implements a `fallback` function. The `msg.sender` with undone alias is validated to be the expected sender's address on L1. The call is then simply forwarded to the target contract. Supports Arbitrum chains (e.g. Arbitrum Nova and One).
- `OptimismReceiver`: Implements a `fallback` function. The `msg.sender` is expected to be the L2 side of the cross-domain messenger while the `xDomainMessageSender` is expected to be the expected sender's address on L1. Supports Optimism chains (e.g. Optimism, Base).
- `AMBReceiver`: Implements a `fallback` function. The `msg.sender` is expected to be the L2 side of the arbitrary-message passing bridge while the `messageSourceChainId` and `messageSender` are expected to match the expected origin chain and sender's address. Supports chains with AMB bridges (e.g. Gnosis).
- `CCTPReceiver`: Implements the CCTP recipients `handleReceiveMessage` function. The `msg.sender` is expected to be the deployed on chains side of the messenger while the source

domain and sender are expected to match the expected values. Supports chains where CCTP is deployed.

2.2.3 Roles and Trust Model

Bridges are fully trusted. The usage of the Forwarder libraries is expected to be correct. The receivers are expected to be deployed only on supported chains. The deployment of receivers is expected to be set up correctly (e.g. targets are correct). Further, connecting forwarders with receivers requires a careful setup of messages sent.

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