PUBLIC

Code Assessment

of the Grove xchain-helpers Smart Contracts

September 01, 2025

Produced for

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

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

GroveLabs offers a library (a fork of Spark xchain-helpers) for cross-chain messaging between Ethereum Mainnet and L2s.

This review focused on the first version of Grove xchain-helpers where ArbitrumERC20Forwarder is newly introduced. Other parts of the system are covered by the Spark xchain-helpers Review.

The most critical subjects covered in our audit are integration with the supported bridges, access control and functional correctness.

The general subjects covered are documentation and integrity.

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 Grove 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	20 Aug 2025	670964a31e11ef3994f79fff4701baee25b70c9c	Initial Version
2	01 Sep 2025	adb270a78138a13ac46cd5a1c1737b7afc8e5fc9	After Intermediate Report

For the Solidity library contracts under review, the code is compatible with compiler version `0.8.x`.

Grove xchain-helpers is a fork of Spark xchain-helpers. The review scope covered the differences between Grove xchain-helpers's initial commit and Spark xchain-helpers's latest commit. The following file is in scope:

```
ArbitrumERC20Forwarder.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.

GroveLabs offers a library (a fork of Spark xchain-helpers) 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 mechanisms.
- 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.

For more details regarding these contracts, please consult the Spark xchain-helpers Review.

2.2.1 ArbitrumERC20Forwarder

Contract ArbitrumERC20Forwarder was added in Grove xchain-helpers to enable cross-chain messaging between Ethereum and Arbitrum stack chains with non-ETH as the native token. It provides the following functions:

- `sendMessageL1toL2`: Sends a message to L2 via its ERC20Inbox by invoking `createRetryableTicket()`. Gas token (an ERC-20) is paid on Ethereum. Excess gas is burned, and L2 call value is not supported. For sending messages to Plume Network, PLUME token is used.
- `sendMessageL2toL1`: Sends a message from L2 to Ethereum through the ArbSys system contract (invokes sendTxToL1).

An ArbitrumReceiver is expected to be deployed on L2. It 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.

2.3 Trust Model

The following privileged roles exist for an Abitrum stack chain:

Bridge admin: fully trusted; can upgrade the bridge contract to DoS cross-chain messaging or drain the gas tokens approved.

Rollup contract and its owner: semi-trusted; can pause the bridge or turn on the allowlist to block certain cross-chain messaging in the worst case.

The L2 bridge is trusted to work honestly and correctly as documented.

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.

For other bridges please refer to the Trust Model of Spark xchain-helpers Review.

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 Open 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 Open 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

- Dangling Approval Code Corrected

6.1 Dangling Approval

Informational Version 1 Code Corrected

CS-GRVXCH-001

When creating a retryable ticket with the ERC20Inbox contract, it will only pull extra tokens required from the `msg.sender`. In case the gas token balance of ERC20Inbox is positive, the allowance approved during `sendMessageL1toL2()` will not be used up, leading to dangling approval from the `msg.sender` to ERC20Inbox.

Code corrected:

At the end of `sendMessageL1toL2()`, the gas token allowance is now explicitly reset to 0, ensuring no dangling approval remains.