



FAIRYPROOF

# Accumulate Bridge

## AUDIT REPORT

Version 1.0.0

Serial No. 2022092900012029

Presented by Fairyproof

September 29, 2022

# 01. Introduction

---

This document includes the results of the audit performed by the Fairyproof team on the Accumulate Bridge project.

**Audit Start Time:**

September 8, 2022

**Audit End Time:**

September 28, 2022

## - Smart Contracts

---

**Audited Code's Github Repository:**

<https://github.com/AccumulateNetwork/bridge-contracts>

**Audited Code's Github Commit Number When Audit Started:**

26d898c9faf69fd744fb0f6c2dd047738feb64d4

**Audited Code's Github Commit Number When Audit Ended:**

4e074bc2271c195fea4077c36e70eb728a06b6bf

Note: the project uses the contracts from a third-party project `gnosis-safe v1.1.3`. These contracts were not covered by this audit.

## - Bridge Frontend

---

**Audited Code's Github Repository:**

<https://github.com/AccumulateNetwork/bridge-frontend>

**Audited Code's Github Commit Number When Audit Started:**

3f4bb8fe2275a2f051ad6f543d1dd7346c9d55a4

**Audited Code's Github Commit Number When Audit Ended:**

4a75d49de214554a750bec3a22ef296bfb713fd2

## - Bridge Node

---

**Audited Code's Github Repository:**

<https://github.com/AccumulateNetwork/bridge>

**Audited Code's Github Commit Number When Audit Started:**

af0c72853c16fa2d473c99ca5ca44870de695b57

**Audited Code's Github Commit Number When Audit Ended:**

e4d9e49daa65fab1c25f5fb30bea27ed8c306d4b

The goal of this audit is to review Accumulate Bridge's implementation for its cross-chain bridge function, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the Accumulate Bridge team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

## — Disclaimer

---

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from off-chain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

## — Methodology

---

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code Review, Including:

- Project Diagnosis

Understanding the size, scope and functionality of your project's source code based on the specifications, sources, and instructions provided to Fairyproof.

- Manual Code Review

Reading your source code line-by-line to identify potential vulnerabilities.

- Specification Comparison

Determining whether your project's code successfully and efficiently accomplishes or executes its functions according to the specifications, sources, and instructions provided to Fairyproof.

2. Testing and Automated Analysis, Including:

- Test Coverage Analysis

Determining whether the test cases cover your code and how much of your code is exercised or executed when test cases are run.

- Symbolic Execution

Analyzing a program to determine the specific input that causes different parts of a program to execute its functions.

3. Best Practices Review

Reviewing the source code to improve maintainability, security, and control based on the latest established industry and academic practices, recommendations, and research.

## — Structure of the document

---

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

## — Documentation

---

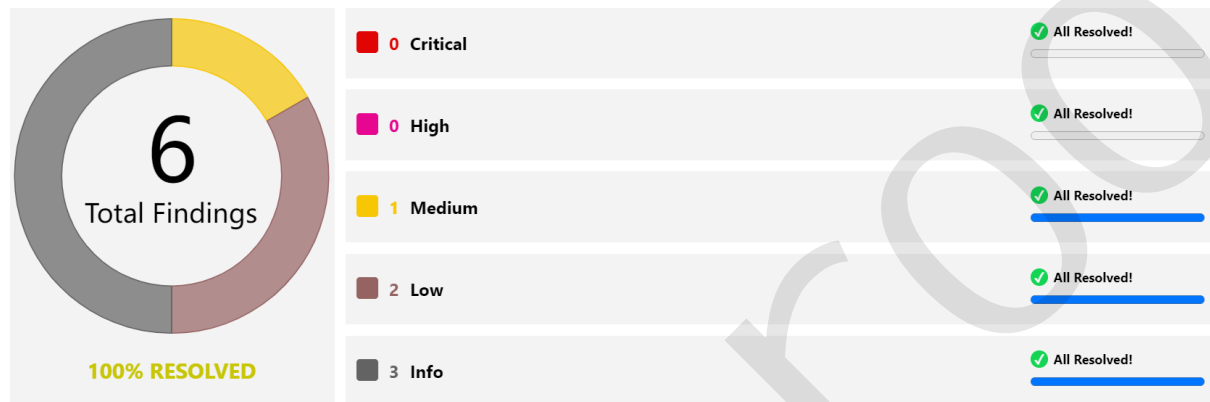
For this audit, we used the following sources of truth about how the Accumulate Bridge should work:

Source Files

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the Accumulate Bridge team or reported an issue.

## — Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2022092900012029	Fairyproof Security Team	Sep 8, 2022 - Sep 28, 2022	Passed



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, one issue of medium-severity, two issues of low-severity and three issues of informational-severity were uncovered. The Accumulate Bridge team fixed all the issues.

## 02. About Fairyproof

[Fairyproof](#) is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

## 03. Introduction to Accumulate Bridge

The Accumulate Bridge allows users to move native Accumulate tokens (ACME and any Accumulate-based tokens) from the Accumulate blockchain to Ethereum and vice versa. The bridge software consists of the bridge node (golang backend application), the bridge frontend (react web3 application), that allows users to interact with the bridge, and smart contracts (solidity smart contracts for Ethereum Virtual Machine).

The bridge nodes can be operated by different entities and use multi-signature on both Accumulate and Ethereum sides to sign txs (mint wrapped ACME on the Ethereum and release native ACME from the multi-sig token account on the Accumulate).

The bridge node, bridge smart contracts, and the bridge UI are open-source and will be released under the MIT license.

## 04. Audited functions

### - Bridge Node

Golang Application: <https://github.com/AccumulateNetwork/bridge>

#### 1. Integration with Accumulate

- a. Accumulate ADIs are used as bridge node operators identities on Accumulate side
- b. Bridge node operators setup multi-sig token accounts (ADI Token Accounts can only hold one type of token e.g. ACME)

acc://bridge.acme – ADI (Identity)

acc://bridge.acme//ACME – Token Account for token ACME

acc://bridge.acme//SMTH – Token Account for token SMTH

etc.

- c. The Token Account for the Ethereum Bridge may read as so:

acc://bridge.acme/Ethereum/ACME

- d. When a user mints wrapped tokens by sending Accumulate tokens to the bridge multi-sig token account, the memo field of tx is used to store the destination address for wrapped tokens on the Ethereum network. Bridge nodes collect a list of deposits and parse amounts and destinations.

- e. Multi-sig token redemption (burn wrapped token – redeem Accumulate token) – bridge nodes create a multisig tx from bridge token account and sign it.

#### 2. Integration with Ethereum

- a. Multi-sig setup for bridge operators (onboarding/offboarding/txs signing) – we use Gnosis Safe open-source smart contract for this (<https://gnosis-safe.io>)

- b. Gnosis safe owners are bridge node operators identities on Ethereum side

- c. Bridge operators parse burn txs, made by users, from Ethereum

- d. Bridge operators mint wrapped tokens by generating and signing multisig mint tx using Gnosis Safe API (<https://safe-transaction.gnosis.io>)

3. Fee module (flat fee without price ACME price oracles)
  - Fee1 = Bridge Node Reward (% of amount being sent) (in ACME)
  - Fee2 = Mint and Burn Operation that Bridge Validators use on Ethereum (est. in ACME)
  - Total Fee = Fee1 + Fee2
4. Minting/burning fees are accumulated into Accumulate multisig token accounts
5. JSON RPC-API (for bridge frontend connection)
6. Documentation

## - Bridge Frontend

---

React Application: <https://github.com/AccumulateNetwork/bridge-frontend>

1. Integration with Accumulate Bridge backend (via JSON-RPC API)
  - a. Fee calculation (GET /fees)
  - b. Bridge-related txs tracking
2. Integration with Ethereum (via web3)
  - a. MetaMask integration
  - b. WalletConnect integration
  - c. Interaction with smart contracts
    - i. Get token info (name, symbol, decimals)
    - ii. Get account balance
    - iii. Approve burn smart contract
    - d. Burn transactions generation (for users)
3. Documentation

## - Smart Contracts

---

Solidity Smart Contracts

<https://github.com/AccumulateNetwork/bridge-contracts>

1. Wrapped Token (ERC20) smart contract
  - a. Unified smart contract for all wrapped tokens (WACME, etc.)
  - b. Forked from audited Open Zeppelin (ERC20, ERC20Burnable, Pausable, Ownable)
 

openzeppelin-contracts/contracts/token/ERC20 at master · OpenZeppelin/openzeppelin-contracts · GitHub
  - c. The only difference from Open Zeppelin - `decimals` variable is used as construction param
  - d. <https://github.com/AccumulateNetwork/bridge-contracts/blob/develop/contracts/WrappedToken.sol>
2. Accumulate Bridge smart contract
3. Audited Gnosis Safe smart contract
 

<https://github.com/safe-global/safe-contracts/tree/main/contracts>

a.Example: <https://rinkeby.etherscan.io/address/0x5ca3ad054405cbe88b0907131cf021f8d24a6291>

- Bridge Operators are owners of Gnosis Safe smart contract (multi-sig M of N)
- Gnosis Safe smart contract is the owner of Accumulate Bridge smart contract (multi-sig required for admin bridge operations like mint)
- Accumulate Bridge smart contract is the owner of Wrapped Token(s) smart contract(s)
- Admin operations, e.g. token mint, require multi-sig tx, submitted to the Gnosis Safe smart contract, with the instruction to interact with controlled Accumulate Bridge smart contract, that calls `mint` of the controlled Wrapped Token smart contract

○ Example: <https://rinkeby.etherscan.io/tx/0x6a39ceed686e633b05aba7ccfea001ddb4e0d0840083eafc1eee382709e159e0>

## 05. Coverage of issues

---

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Re-entrancy Attack
- Replay Attack
- Reordering Attack
- Miner's Advantage
- Rollback Attack
- DDoS Attack
- Injection Attack
- Transaction Ordering Attack
- Race Condition
- Access Control
- Integer Overflow/Underflow
- Timestamp Attack
- Gas Consumption
- Inappropriate Callback Function
- Function Visibility
- Implementation Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Tx.origin
- Fake Deposit
- Shadow Variable
- Design Vulnerability
- Token Issuance
- Admin Rights
- Inappropriate Proxy Design
- Inappropriate Use of Slots
- Asset Security
- Contract Upgrade/Migration



- Code Improvement
- Misc

## 06. Severity level reference

---

Every issue in this report was assigned a severity level from the following:

**Critical** severity issues need to be fixed as soon as possible.

**High** severity issues will probably bring problems and should be fixed.







**Medium** severity issues could potentially bring problems and should eventually be fixed.

**Low** severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

**Informational** is not an issue or risk but a suggestion for code improvement.

## 07. List of issues by severity

---

Index	Title	Issue/Risk	Severity	Status
FP-1	Invalid Owner Provided	Implementation Vulnerability	Medium	 Fixed
FP-2	Missing Validation for EVM Address	Implementation Vulnerability	Low	 Fixed
FP-3	Missing Check for Account	Implementation Vulnerability	Low	 Fixed
FP-4	Missing Verification for BridgeAddress	Implementation Vulnerability	Info	 Fixed
FP-5	Redundant Replay Attack Protection	Code Improvement	Info	 Fixed
FP-6	Incorrect Button Status	Implementation Vulnerability	Info	 Fixed

## 08. Issue descriptions

### [FP-1] GS026: Invalid Owner Provided Medium ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

- Gnosis safe API kept returning mint tx as incomplete after it was submitted to the network
- Leader node submitted it again
- Tx had been reverted by the network

Error GS026 was generated.

Recommendation:

Consider storing the latest safetxhash and checking that safetxhash from API differs from the latest submitted.

Update/Status:

It has been fixed by the Accumulate Bridge team at <https://github.com/AccumulateNetwork/bridge/commit/6cd5f045fd135e7eda38082b2d65778126107f21>

### [FP-2] Missing Validation for EVM Address Low ✓

Fixed

Issue/Risk: Implementation Vulnerability

Description:

The `processNewDeposits` didn't validate the EVM address. If it was an invalid address, tokens transferred to that address might be lost permanently.

Recommendation:

Consider adding validation for the EVM address. If it is invalid, quit the transaction.

Update/Status:

It has been fixed by the Accumulate Bridge team at: <https://github.com/AccumulateNetwork/bridge/commit/abddcd75910a21e3f7d87c1f85b8da4315d25abf>

## [FP-3] Missing Check for Account Low ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

When `Release` is executed, the `acc` is not verified whether it is a `Token Account`. If a non `Token Account` is passed, the cross-chain transaction will fail.

Recommendation:

Consider adding the following logic:

- If a `Lite Account` is passed, the implementation should check if the account already exists. If it doesn't exist, an error message should be generated.
- If a non `Lite Account` is passed, it already exists and it is a `Token Account`, the transaction can proceed.

Update/Status:

It has been fixed by the Accumulate Bridge team at <https://github.com/AccumulateNetwork/bridge-frontend/commit/f6dba2c6364e7e7264abca967ab5bd9558457856>

## [FP-4] Missing Verification for BridgeAddress Informational ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

The `parseToken` doesn't verify `BridgeAddress`. If the address was incorrectly configured, cross-chain transactions with the address would fail.

Recommendation:

Consider adding verification for `BridgeAddress` in `parseToken`.

Update/Status:

It has been fixed by the Accumulate Bridge at <https://github.com/AccumulateNetwork/bridge/commit/e4d9e49daa65fab1c25f5fb30bea27ed8c306d4b>.

## [FP-5] Redundant Replay Attack Protection

Informational

✓ Fixed

Issue/Risk: Code Improvement

Description:

In `AccumulateBridge.sol` all external functions have already been protected from replay attacks.

Recommendation:

Consider removing the code to prevent replay attacks.

Update/Status:

It has been fixed by the Accumulate Bridge team at <https://github.com/AccumulateNetwork/bridge-contracts/commit/6042dd59abe4ebf30831c13535b00f0c4e593d29>.

## [FP-6] Incorrect Button Status

Informational

✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

After the Approve button is hit and the page reloads, button will still show `Approve`.

Recommendation:

Consider changing the code such that after the Approve button is hit and the page reloads the button will show `Approved`.

Update/Status:

It has been fixed by the Accumulate Bridge team at <https://github.com/AccumulateNetwork/bridge-frontend/commit/4b2e96e5b4ce533a412d89c10c676a9ef2bd1c70>

# 09. Recommendations to enhance the overall security

---

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- Consider limiting the maximum allowed tokens to be transferred daily. This is to mitigate the risk of the crypto assets being drained.

- Consider adding a function to combine multiple transactions into an atomic "combo" transaction. When such a "combo" transaction is executed, either all of the transactions included will succeed or none of the transactions will succeed.
- To complete a cross-chain transaction, the private keys of an EMV blockchain's wallet and an Accumulate's wallet are used to sign it. In the existing implementation the private keys are kept locally in configuration files which are accessible to operators. This introduces risks of the private keys being compromised. Consider using Clef for signing a transaction on an EVM chain and Walletd for signing a transaction on Accumulate. Clef and Walletd should be deployed independently from the bridge nodes. Access control to Clef and Walletd should be managed and maintained by different people. Although this may increase the maintenance costs, it will greatly enhance the overall security.



-  <https://medium.com/@FairproofT>
-  <https://twitter.com/FairproofT>
-  <https://www.linkedin.com/company/fairproof-tech>
-  [https://t.me/Fairproof\\_tech](https://t.me/Fairproof_tech)
-  [Reddit: https://www.reddit.com/user/FairproofTech](https://www.reddit.com/user/FairproofTech)

