ALEX Audit

Bridge Endpoints

December 2022

By CoinFabrik

v202209

Executive Summary

CoinFabrik was asked to audit the contracts for the ALEX bridge's project. The audited files are from the git repository located at https://github.com/alexgo-io/alex-bridge-backend. The audit is based on the commit `9c8b253647de2a2e80125855b0fc10776eca5b6b`. Fixes were checked on commit `75d6cac9b65123f25df670636df26ba97363fdf7`.

This project is a Stacks-Ethereum hybrid bridge which allows users to transfer their assets across those blockchains.

The whole bridge project consists of the audited endpoints, an interaction with the Wrapped vendor, and off-chain components that monitor on-chain transactions and approves the asset transfer.

During this audit, we found two minor issues. Also, two enhancements were proposed.

An issue was fixed, while the other was mitigated. The enhancements were implemented.

Scope

The scope for this audit includes and is limited to the following files:

- `clarity/contracts/bridge-endpoint.clar`: Stacks endpoint. Users send wrapped tokens in order to unwrap and withdraw them on Ethereum. Also, it holds the wrapped tokens for those who bridge from its Ethereum counterpart until a relayer executes the transfer.
- `solidity/contracts/BridgeEndpoint.sol`: Ethereum endpoint. Users send unwrapped tokens in order to wrap and bridge them to Stacks. This endpoint holds unwrapped tokens for transferring to those who bridge from Stacks, unwrapping their tokens.
- `solidity/contracts/utils/helpers/ERC20Fixed.sol`: Helper library for 18-decimal fixed point precision.

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Methodology

CoinFabrik was provided with the source code, including automated tests that define the expected behavior, and general documentation about the project. Our auditors spent one week auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies

beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses:

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

After delivering a report with our findings, the development team had the opportunity to comment on every finding and fix the issues they considered convenient. Once fixed and/or commented, our team ran a second review process to verify that the changes to the code effectively solve the issues found and do not unintentionally add new ones. This report includes the final status after the second review.

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

ID	Title	Severity	Status
MI-01	Owner Can Freeze Endpoint Withdrawals	Minor	Resolved
MI-02	Insecure Authentication Through tx-sender	Minor	Mitigated

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

No issues found.

Medium Severity Issues

Minor Severity Issues

MI-01 Owner Can Freeze Endpoint Withdrawals

Location:

- `clarity/contracts/bridge-endpoint.clar:136-141`
- `solidity/contracts/BridgeEndpoint.sol:34-46, 57-61`

Since relayers need to provide a minimum amount of signatures in order to transfer the tokens to the user, an endpoint could suffer a partial denial of service if that minimum is set to an unreachable value. Therefore, the owner can set it to 101 on the Stacks endpoint, which is also type-constrained to 100 signatures, and the condition required to execute `bridge-endpoint::transfer-to-wrap()` will never be met. This also applies for the Ethereum endpoint, but there is not a type constraint for the amount of signatures that can be provided in the function call.

Recommendation

Define a maximum value for the minimum amount of signatures required.

Status

Resolved. Maximum value defined and enforced.

MI-02 Insecure Authentication Through tx-sender

Location:

- `clarity/contracts/bridge-endpoint.clar: 213, 250, 282`

Using `tx-sender` for authentication is not secure. Actors in the system could be targeted for phishing.

This issue was found in:

- `clarity/contracts/bridge-endpoint::transfer-to-wrap()`
- `clarity/contracts/bridge-endpoint::check-is-owner()`
- `clarity/contracts/bridge-endpoint::transfer-to-unwrap()`

Functions that involve asset transfers cannot be called in the attack, thanks to proper use of post-conditions. Also, owner authentication will be less likely to be targeted once the DAO is set as owner.

Recommendation
Prefer `contract-caller` to `tx-sender` for authentication, unless it is specifically required and the risk is considered. Also, adding a mapping for trusted callers might be helpful if intermediary contracts are needed.

Status
Mitigated. The development team stated it is mitigated for owner authentication given that all these contracts will be owned by a DAO. For other cases, given that post-conditions can prevent asset transfers triggered by malicious actors through `tx-sender`, they are in favor of keeping `tx-sender` in place for composability.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

ID	Title	Status
EN-01	Unnecessary Zero-address Check	Implemented
EN-01	Prefer `_grantRole()` Over `_setupRole()`	Implemented

EN-01 Unnecessary Zero-address Check

Location:
- `solidity/contracts/BridgeEndpoint.sol:91`

The `transferToUnwrap()` function verifies if the `_to` address is equal to zero. However, `ERC20::transfer()` already verifies it.

Recommendation
Remove the unnecessary check in order to reduce runtime cost.

Status
Implemented.

EN-02 Prefer _grantRole() Over _setupRole()

Location:
- `solidity/contracts/BridgeEndpoint.sol:42`

According to the library documentation[1], `_setupRole()` is deprecated in favor of `_grantRole()`.

Recommendation

Replace `_setupRole()` call with `_grantRole()`.

Status

Implemented.

[1] https://github.com/OpenZeppelin/openzeppelin-contracts/blob/3d7a93876a2e5e1d7fe29b5a0e96e222afdc4cfa/contracts/access/AccessControl.sol#L203

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Centralization

Since this bridge project took a hybrid approach, there are unavoidable centralization points:

- Endpoints depend on an intermediary process in order to work.
- Tokens are in custody of a third party (Wrapped).
- A relayer sends a transaction in order to execute token transfers.
- The relayer has to provide validator signatures.
- A contract owner defines the required amount of signatures, the approved recipients and the relayers.

Upgrades

The audited contracts do not provide mechanisms for eventual upgrades.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

BridgeEndpoint.sol and bridge-endpoint.clar

Owner

Defines system parameters like designated validators, relayers, signatures requirement, and approved tokens.

Validator

Verifies bridge transactions and approves them providing a signature with the order hash to a relayer.

Relayer

Completes bridges by providing validator signatures in a call to the endpoint where the tokens are transferred to the user.

Changelog

- 2022-12-15 – Initial report based on commit `9c8b253647de2a2e80125855b0fc10776eca5b6b`.
- 2023-01-03 – Fixes were checked on commit `75d6cac9b65123f25df670636df26ba97363fdf7`.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the ALEX project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.