ALEX XLink Audit

BridgeEndpoint, BridgeEndpointWithAxelar & BridgeRegistry

June 2024

By CoinFabrik

v202406

Executive Summary

Earlier in June 2024, CoinFabrik was asked to audit contracts for the ALEX XLink project implementing a bridge between Stacks and Ethereum (or another EVM-compatible blockchain). The project underwent several redesigns and modifications to its contracts on the Solidity side, in particular segregating data and logic functionality by adding a registry contract. This audit is concerned with some of the changes as described in the scope upcoming section.

During this audit we found no critical issues, two high-severity issues, two medium-severity issues and four minor-severity issues. Also, several enhancements were proposed.

All high and medium severity issues were resolved. Of the four minor issues, two were resolved, one was partially resolved, and one was acknowledged. Additionally, two of the five proposed enhancements were implemented.

Scope

The audited files are from the git repository located at https://github.com/alexgo-io/xlink. The audit is based on the following commit `948fd511c3ed7c15bb965515e63b8df3180e7446`. Fixes were checked on commit `0dbfc65362ebf414ba3674d5e2c44b88f4e15be9`.

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

- `contracts/bridge-solidity/contracts/BridgeEndpoint.sol`: Provides an operational interface for peg-in users to initiate wrapping processes to Stacks chain through the `transferTo*()` function. After deducting a fee for the registry, tokens are transferred from the user to the XLink cold wallet (or burned, if applicable). The contract also serves as an interface for peg-out user operations, where relayers are expected to call the `transferToUnwrap()` function with validated orders to be then fulfilled by the so-called “fillers”. On burnable cases, minted tokens are directly sent to the user or to the `TimeLock` contract, depending on the amount.
- `contracts/bridge-solidity/contracts/BridgeEndpointWithAxelar.sol`: This contract inherits from `BridgeEndpoint` contract and adds a specific `transferToAxelar()` function, which is expected to be called by relayers on cross-chain operations.
- `contracts/bridge-solidity/contracts/BridgeRegistry.sol`: This contract provides the data and access control layer for the bridge operations. With the `MultisigWallet` as the owner, it manages roles, approved tokens, fees, and other related functionalities.

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.

Project Description

This project is a Stacks - EVM Chains hybrid bridge which allows users to transfer their assets across those blockchains. A wrapping process takes tokens from an EVM chain to Stacks. It starts with a user transferring its tokens to an EVM chain endpoint smart contract (typically `BridgeEndpoint`), and is followed by a set of validators listening to these on-chain events, and a backend process generating a proof of the peg-in order and storing it in the backend. The proofs are caught by a relayer which then must send proofs to the Stacks blockchain. Finally, a Stacks smart contract validates these proofs and mints/transfers the tokens to the settle address established by the user.

The unwrap process goes in the opposite direction: a user sends tokens to a smart contract in the Stacks blockchain and expects to receive corresponding tokens on the EMV chain to a pre-established settlement address. Again, validators read this order, and generate a proof if the order is validated. The proofs are stored in the backend and after a relayer notices there are enough proofs, it relays the proof to the bridge smart contract on an EVM blockchain. This is done by calling the functions `transferToWrap()`, `transferToLaunchpad()`, `transferToAxelar()`, etc.

When unwrapping, the steps are different between burnable and non-burnable tokens in this new `BridgeEndpoint` version. For burnable, minted tokens are either sent to the recipient based or to `TimeLock` contract if an amount threshold is exceeded. `TimeLock` contract features essentially delay the unwrapping conclusion on the EVM blockchain side by locking the tokens and generating an agreement.

Assumptions

If an assumption fails, the system could be liable to unreported threats.

- The `BridgeEndpoint` is expected to be used independently from the `BridgeEndpointWithAxelar`, which is intended to be deployed on a specific EVM blockchain as a hand-off point to/from Axelar.
- All burnable tokens have 18-digit precision.
- The owner of the `BridgeRegistry` and `BridgeEndpoint` contracts is an instance of the `MultisigWallet` contract.

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
HI-01	Tokens Are Not Minted to Recipient	High	Resolved

ID	Title	Severity	Status
HI-02	Incorrect Token Transfer on Unwrapping Operation	High	Resolved
ME-01	Required Validators Can Be Zero	Medium	Resolved
ME-02	Excessive Permission for Changing Order Status	Medium	Resolved
MI-01	Unvalidated Owner Address at Constructor	Minor	Resolved
MI-02	Set Maximum Fee for Transfers	Minor	Acknowledged
MI-03	Non Approved Tokens Funds Are Locked	Minor	Resolved
MI-04	Floating Pragma	Minor	Resolved

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.
- **High:** These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and **demand immediate attention**.
- **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. The reported risk is accepted by the development team.
- **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.

High Severity Issues

HI-01 Tokens Are Not Minted to Recipient

Location:

- `contracts/bridge-stacks/contracts/BridgeEndpoint.sol: 322`

Classification:

- CWE-670: Always-Incorrect Control Flow Implementation[1]

In the case of burnable unwrapping operations, tokens are minted to the `BridgeEndpoint` contract using `address(this)` instead of being minted to the recipient. This means that users do not receive their unwrapped funds; on the contrary, they are held by the endpoint.

Recommendation

Implement the correct flow on `transferToUnwrap()` for the burnable token scenario.

Status

Resolved. The time lock functionality was implemented on `BridgeEndpoint` to handle burnable tokens on unwrapping operations. Tokens are minted to the user if the amount is less than `timeLockThreshold`; if not, there is a delay on the release managed by the `TimeLock` contract.

[1] https://cwe.mitre.org/data/definitions/670.html

HI-02 Incorrect Token Transfer on Unwrapping Operation

Location:

- `contracts/bridge-stacks/contracts/BridgeEndpoint.sol: 431`

Classification:

- CWE-669: Incorrect Resource Transfer Between Spheres[2]

Once a relayer has called `transferToUnwrap()` for a non-burnable token, the order is queued in the `unwrapSent` map. These orders, which act as "requests", are then filled by others who send their own tokens to the recipient and finalize the request. This is done through the `finalizeUnwrap()` public function, which receives an array of order hashes. For each order, the internal function `_finalizeUnwrap()` is called. If checks are passed (order not sent, etc.) the `BridgeEndpoint` transfers the amount to the recipient. However, this is not the expected behavior, as tokens must be sent from the filler (`msg.sender`). This can lead to incorrect behaviors, potentially exhausting bridge funds and preventing the finalization of user's peg-outs.

Recommendation

Replace `transferFixed()` with `transferFromFixed()` and add `msg.sender` as the `address _from` argument. This way, the code correctly implements the intended hot wallet logic.

Status

Resolved. Fixed according to the recommendation.

Medium Severity Issues

ME-01 Required Validators Can Be Zero

Location:

- `packages/contracts/bridge-solidity/contracts/BridgeRegistry.sol: 73, 101`

Classification:

- CWE-754: Improper Check for Unusual or Exceptional Conditions[3]

The public variable `requiredValidators` can be set to zero. The function `setRequiredValidators(uint256 _requiredValidators)` only checks for an upper bound. This issue also appears at deployment time, where `requiredValidators` is given its initial value on the constructor function. Although this is an only-owner action and an

[2] https://cwe.mitre.org/data/definitions/669.html
[3] https://cwe.mitre.org/data/definitions/754.html

unlikely scenario, the risk associated is very high since orders can be executed without proofs.

Recommendation
Add checks to ensure `requiredValidators` is always greater than zero. Additionally, it is advisable to add a `MIN_REQUIRED_VALIDATORS` constant, similar to the upper bound. In constructor, it might even be prudent to set `requiredValidators` as `MAX_REQUIRED_VALIDATORS`.

Status
Resolved. Fixed according to the recommendation.

ME-02 Excessive Permission for Changing Order Status

Location:

- `packages/contracts/bridge-solidity/contracts/BridgeRegistry.sol: 178-180`

Classification:

- CWE-285: Improper Authorization[4]

Approved addresses from the `MultisigWallet` contract can change the status of a sent order from true to false by calling `setOrderSent(bytes32 orderHash, bool sent)` function. While switching the status from false to true is essential within the protocol, allowing the inverse change is highly permissive and can have significant unexpected consequences.

A similar issue exists with the `setOrderValidatedBy(bytes32 orderHash, address signer, bool validated)` function. Permitting changes in both directions for these statuses can lead to unauthorized modifications and potential security risks.

Recommendation
Restrict these functions to only allow status changes from false to true. If necessary, add an additional feature specifically to reset an order status under strict permissions and conditions. See related enhancement EN-04 Least Privilege Principle In Registry Functions.

Status
Resolved. Fixed according to the recommendation; if the boolean status is to be changed from `true` to `false`, only the owner can perform it.

[4] https://cwe.mitre.org/data/definitions/285.html

Minor Severity Issues

MI-01 Unvalidated Owner Address at Constructor

Location:

- `packages/contracts/bridge-solidity/contracts/BridgeEndpoint.sol: 134-146`
- `packages/contracts/bridge-solidity/contracts/BridgeRegistry.sol: 66-74`

The constructor for the `BridgeEndpoint` contract accepts three addresses as input. While `_registry` and `_pegInAddress` are checked to be nonzero, the `owner` is not checked. Same issue occurs in the `BridgeRegistry` contract. An owner set to the zero address would result in an unusable contract that needs to be redeployed.

Recommendation

Require the owner to be non-zero in the constructor function. Also, refer to the related EN-01 Constructor Improvements which provides an overall enhancement for constructors.

Status

Resolved. Fixed according to the recommendation. A zero address check was added to the `BridgeEndpoint` and `BridgeRegistry` constructors.

MI-02 Set Maximum Fee for Transfers

Location:

- `packages/contracts/bridge-solidity/contracts/BridgeEndpoint.sol`

When calling the internal `_transfer()` during wrapping operations, it is advisable to ensure that fees are not excessive. Since the `BridgeRegistry::setMinFeePerToken()` function is not capped, users could be made to pay fees in excess to their expectations. In fact, even if users check fees before initiating a transaction, they could still be frontrun and face higher fees.

Note: This issue corresponds to MI-03 from the previous ALEX XLink Bridge Stacks & EVM Chain audit report.

Recommendation

Introduce a `maximumFeeToPay` parameter to the transfer function. This parameter will enable users to set the maximum fee they are willing to pay. If the fee exceeds this value, the function should revert.

Status
Acknowledged.

MI-03 Non Approved Tokens Funds Are Locked

Location:

- `packages/contracts/bridge-solidity/contracts/BridgeRegistry.sol`

There are two functions within the contract to withdraw the registry's token balance: `collectAccruedFee()` and `transferFixed()`. However, both functions use the `onlyApprovedToken` modifier. This restriction means that funds of a non-approved token cannot be withdrawn from the registry. The only way to withdraw such funds is by approving the token again, which might not be desirable if the token disapproval was triggered due to a security issue.

Recommendation
Add a mechanism to withdraw non-approved token funds. This can be archived by either:

- Removing `onlyApprovedToken` modifier from the `transferFixed()` function.
- Adding a specific only-owner function to facilitate this purpose.

Note: There is no `setAccruedFee()` function, so any token transfer movement should consider this limitation, such as setting the accrued fee to zero when withdrawing all funds. See related EN-05 Collect Accrued Fee Improvement.

Status
Resolved. Fixed according to the recommendation. The `transferFixed()` function was restricted to only-owner access and now allows transfers of any token held by the registry.

MI-04 Floating Pragma

Contracts should be deployed with the same compiler version that they have been thoroughly tested with. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that negatively affect the contract system.

Recommendation
Lock the pragma version, replacing `pragma solidity ^0.8.17;` with a specific patch, preferring the most updated version. For example, `pragma solidity 0.8.26`.

Status
Resolved. The pragma version was locked to version `0.8.17`.

Enhancements

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

ID	Title	Status
EN-01	Constructor Improvements	Not implemented
EN-02	Group Modifiers To Avoid Code Repetition	Not implemented
EN-03	Set Approved Token Improvements	Implemented
EN-04	Least Privilege Principle In Registry Functions	Implemented
EN-05	Collect Accrued Fee Improvement	Not implemented

EN-01 Constructor Improvements

Location:

- `packages/contracts/bridge-solidity/contracts/BridgeEndpoint.sol: 134-146`
- `packages/contracts/bridge-solidity/contracts/BridgeRegistry.sol: 66-74`

The constructor parameter `owner` is unnecessarily typed as `MultisigWallet`. It is only used to be cast to an address type and passed as a function argument. In `BridgeEndpoint`, it is only used to define the owner at line 145. In `BridgeRegistry`, it is used to define the owner and to set the default admin role of the `AccessControl` contract at lines 71 and 72.

Furthermore, executing `_transferOwnership(address(owner))` can actually be replaced by invoking the `Ownable` contract constructor[5] directly. This can be achieved by using `Ownable(address(owner))` between the function parameters and body. If the former observation is implemented, it may also be simplified to `Ownable(owner)`.

Status

Not implemented.

[5] https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol#L38

EN-02 Group Modifiers To Avoid Code Repetition

Location:

- `packages/contracts/bridge-solidity/contracts/BridgeEndpoint.sol`
- `packages/contracts/bridge-solidity/contracts/BridgeEndpointWithAxelar.sol`

In all public transfer functions, the following four modifiers are consistently used in the same order: `nonReentrant`, `whenNotPaused`, `onlyAllowlisted`, `onlyApprovedToken(token)`, `notContract`.

Similarly, for privileged transfer functions `transferToUwrap()` and `transferToAxelar()`, six modifiers are uniformly applied in the same order: `onlyApprovedRelayer`, `nonReentrant`, `whenNotPaused`, `onlyApprovedToken(token)`, `notContract`, `notWatchlist(recipient)`.

Recommendation

Consider grouping these modifiers into composite modifiers to reduce code repetition and minimize potential mistakes. Say, `publicTransfers` is the modifier for the public transfers, and `priviledgedTransfers` for unwrap privileged operations.

Status

Not implemented.

EN-03 Set Approved Token Improvements

Location:

- `packages/contracts/bridge-solidity/contracts/BridgeRegistry.sol: 66-74`

The `BridgeRegistry::setApprovedToken()` setter function is very powerful since it has the ability to change every field of a bridge's token and simultaneously approve or disapprove it for operation on the bridge. Adding more granularity to these features is advisable to reduce potential errors. For instance, since `AccessControl::grantRole()` and `AccessControl::revokeRole()` can be used for approving purposes, an hypothetical `BridgeRegistry::setTokenDetails()` can replace the current implementation just for setting the remaining fields.

Additionally, there are no boundaries for the `uint256` parameters of the `setApprovedToken()` function. The suggested checks are as follows.

1. `feePctPerToken` must be less than `1e18` since it is a percentage. Setting this higher leads to `SUB_OVERFLOW` error in `BridgeEndpoint::_transfer()`, as the fee will be greater than the amount.
2. `minAmount` should be less than or equal to `maxAmount` for correctness.

3. `minFeePerToken` should be less than or equal to `minAmount` for consistency.

Status
Implemented.

EN-04 Least Privilege Principle In Registry Functions

Location:
- `packages/contracts/bridge-solidity/contracts/BridgeRegistry.sol`

Some registry functions that have the `onlyApproved` modifier are solely called by the `BridgeEndpoint` contract within the audited scope. These functions are `addAccruedFee()`, `setOrderSent()` and `setOrderValidatedBy()`. Then remains `transferFixed()` which is not called by contracts within the scope and allows the transfer of arbitrary amounts of an approved token from the registry to the caller.

Evaluate whether it makes sense to make the approved role more granular by adding, for example, an endpoint role. This enhancement is related to ME-02 Excessive Permission for Changing Order Status.

Status
Implemented. The enhancement has been implemented as a result of fixes for issues ME-02 Excessive Permission for Changing Order Status and MI-03 Non Approved Tokens Funds Are Locked.

EN-05 Collect Accrued Fee Improvement

Location:
- `packages/contracts/bridge-solidity/contracts/BridgeRegistry.sol`

Currently there is an `addAccuredFee()` function which is called by the `BridgeEndpoint` when receiving peg-in orders. These fees can be collected with `collectAccruedFee()` by the owner. However, bridge funds can vary due to the possibility of approved addresses calling `transferFixed()`. As a result, the registry's token balance may be less than `collectAmount` (line 149), causing `ERC20Fixed::transferFixed()` to fail due to insufficient funds.

Recommendation
It might be useful to make `collectAccruedFee()` flexible for the case where the total balance is less than `collectAmount` by transferring all balance in such cases. Note `CollectAccruedFeeEvent` should be updated accordingly.

Status
Not implemented.

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

BridgeEndpoint

The whole `BridgeEndpoint` functionality can be paused by the owner. The owner is also responsible for managing `Allowlistable` functionalities and adding or removing users for the list. It is notable that `onlyAllowlisted` modifier is present in all the public transfer functions to initiate a wrapping process.

Furthermore, the owner is capable of managing `TimeLock` configurations, such as setting the `TimeLock` contract instance address to be used as the `timeLock` variable. In particular, `TimeLock`'s `releaseDelay` storage variable is not bounded.

BridgeRegistry

The owner of this contract is initially set to the `MultisigWallet` contract. Additionally, the `DEFAULT_ADMIN_ROLE` is also initially assigned to it. Therefore, the `MultisigWallet` can perform all only-owner operations and manage access control as well.

Upgrades

Contracts are not upgradable.

Privileged Roles

All the contracts are children of OpenZeppelin's `Ownable` contract, so the owner can perform all inherited functions[6].

BridgeEndpoint

Owner

The owner of the contract can call the following functions.

[6] https://docs.openzeppelin.com/contracts/4.x/access-control#ownership-and-ownable

- `pause()`, `unpause()`. Allows pausing and unpausing the entire endpoint functionality. Note the `whenNotPaused` modifier is present in all `transferTo*()` functions and in the `finalizeUnwrap()` function.
- `onAllowList()`, `offAllowList()`. Turns on and off the `Allowlistable` functionality. When on, only listed address can call the functions guarded by the `onlyAllowlisted` modifier.
- `addAllowlist()`, `removeAllowlist()`. Adds and removes addresses from the `allowlisted` map.
- `setTimeLock()`. Sets the `TimeLock` contract instance address.
- `setTimeLockThreshold()`, `setTimeLockThresholdByToken()`. Sets the threshold amount for using the `TimeLock` contract. There is a general threshold and a specific threshold for each token. The maximum of both is used.

Relayer
A relayer may call the following functions.

- `transferToUnwrap()`. Initiates a peg-out order on the EVM chain by sending order validators' proofs.

BridgeEndpointWithAxelar

As this contract inherits from `BridgeEndpoint`, only changes or additional roles are documented here.

Owner
The owner of the contract can call the following functions.

- `setTimeLock()`. Sets the `TimeLockWithAxelar` contract instance address.

Relayer
A relayer can call the following functions.

- `transferToAxelar()`. Initiates a cross-chain peg-out order on the EVM chain by sending order validators' proofs.

BridgeRegistry

Owner
The owner of the contract can call the following functions.

- `transferFixed()`. Transfers an arbitrary amount of a token from the registry's balance to the owner.
- `setMinFeePerToken()`. Sets the minimum fee for a token.
- `setRequiredValidators()`. Sets the minimum required validators' proofs to consider an order valid.

- `setWatchlist()`. Sets an address as whitelisted or not by modifying the `watchlist` map. This functionality is currently unused but might be used by other contracts outside of this audit scope.
- `setApprovedToken()`. Adds or modifies a bridge token, including the ability to approve or disapprove it.
- `collectAccruedFee()`. Collects accrued fee of a certain token and sends it to the owner.
- `grantValidators()`, `revokeValidators()`. Grants or revokes addresses their `VALIDATOR_ROLE`.
- Since the owner is an instance of the `MultisigWallet` contract, the owner can also call the functions in the section below (Approved), with the caveats for two functions.
    - `setOrderSent()`. Allows setting an order, identified by its `orderHash`, as sent or unsent by modifying the `orderSent` map.
    - `setOrderValidatedBy()`. Allows setting an order as validated (or unvalidated) by a certain signer (validator) by modifying the `orderValidatedBy` map.

Approved

The approved role is managed by an instance of the `MultisigWallet` contract. The privileged calls related to this role are guarded by the `onlyApproved` modifier, which ensures that the caller has the `APPROVED_ROLE` on the `MultisigWallet` or that the caller is the `MultisigWallet` itself (essentially the owner of the contract, see Assumptions section). Approved addresses have the following privileges.

- `addAccruedFee()`. Adds accrued fee per token when performing a peg-in operation.
- `setOrderSent()`. Allows setting an order, identified by its `orderHash`, as sent by modifying the `orderSent` map. Changing the status from `true` to `false` is not permitted (it is an only-owner action).
- `setOrderValidatedBy()`. Allows setting an order as validated by a certain signer (validator) by modifying the `orderValidatedBy` map. Changing the status from `true` to `false` is not permitted (it is an only-owner action).

DEFAULT_ADMIN_ROLE

This role is initially the same as the contract owner, which is the `MultisigWallet` contract. `AccessControl` includes a special role called `DEFAULT_ADMIN_ROLE`, which acts as the default admin role for all roles. This role is also its own admin: it has permission to grant and revoke its own role[7].

[7] https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/AccessControl.sol#L44

`VALIDATOR_ROLE`
Addresses with the `VALIDATOR_ROLE` are responsible for signing orders. This role is used in the `BridgeEndpoint` to validate orders.

`RELAYER_ROLE`
TAddresses with the `RELAYER_ROLE` are responsible for submitting peg-out orders. The role is used in the `BridgeEndpoint` and `BridgeEndpointWithAxelar` to perform privilege `transferToUnwrap()` and `transferToAxelar()` calls.

Audit Process

A previous audit was conducted by CoinFabrik in early June, titled *ALEX XLink Bridge Stacks & EVM Chain Audit 2024-06* which included the files in scope. Afterward, design changes to the protocol required a re-audit to the bridge endpoint and registry. Some of the issues from the previous audit still remain and are reported here with a corresponding note.

Changelog

- 2024-06-21 – Initial report based on commit `948fd511c3ed7c15bb965515e63b8df3180e7446`.
- 2024-07-08 – Final report based on commit `0dbfc65362ebf414ba3674d5e2c44b88f4e15be9`.

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