CoinFabrik

Security Audit Report

XLink - Endpoints Update

March 2025

V202411

Executive Summary **3**
Scope **3**
Findings **3**
Critical Severity Issues 4
CR-01 Unvalidated Trait Called in finalize-peg-in-agg 4
High Severity Issues 5
Medium Severity Issues 5
ME-01 Improper Refund Handling in finalize-peg-in-agg 5
ME-02 Incorrect Recipient of aBTC Fee 6
ME-03 Reverted Peg-in due to Insufficient Balance in finalize-peg-in-agg 7
Low Severity Issues 7
Other Considerations **7**
Centralization 8
Upgrades 8
About CoinFabrik **8**
Methodology **8**
Severity Classification **9**
Issue Status **10**
Disclaimer **11**
Changelog **11**

Executive Summary

CoinFabrik was asked to audit the contracts for XLink's **Endpoints Update**.

During this audit we found one critical issue and three medium issues.

All the issues were resolved.

Scope

The audited files are from the git repository located at https://github.com/xlink-network/xlink, in the **`./packages/contracts/bridge-stacks/contracts/`** directory. The audit is based on the commit **`365709a9c5d7388d76f09cc63776b385fcc9af20`**. Fixes checked on **`5f6cd61dd126836195898c42f1345b7d78668f44`**.

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

- **`./btc-peg-in-v2-07e-agg.clar`**: Processes aggregated peg-in transactions from Bitcoin to Stacks, minting tokens and enabling swaps based on validated Bitcoin transaction data..
- **`./cross-peg-out-v2-01b-agg.clar`**: Facilitates aggregated peg-out operations from Stacks to other blockchains, handling token burns or transfers to release funds across chains.
- **`./meta-peg-in-v2-06e-agg.clar`**: Manages BRC-20/Runes peg-in operations from Bitcoin to Stacks.

Fixes were implemented in the following files:

- **`./btc-peg-in-v2-07e-agg.clar`**
- **`./meta-peg-in-v2-06h-agg.clar`**

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.

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.

Each severity label is detailed in the Severity Classification section. Additionally, the statuses are explained in the Issues Status section.

Id	Title	Severity	Status
CR-01	Unvalidated Trait Called in finalize-peg-in-agg	🟥 Critical	**Resolved**
ME-01	Improper Refund Handling in finalize-peg-in-agg	🟨 Medium	**Resolved**
ME-02	Incorrect Recipient of aBTC Fee	🟨 Medium	**Resolved**
ME-03	Reverted Peg-in due to Insufficient Balance in finalize-peg-in-agg	🟨 Medium	**Resolved**

Critical Severity Issues

CR-01 Unvalidated Trait Called in finalize-peg-in-agg

Location

- `./meta-peg-in-v2-06e-agg.clar:154-156`

Classification

- CWE-20: Improper Input Validation[1]

Description

In the `finalize-peg-in-agg` function, the contract calls `token-in-trait` to mint tokens before performing validation via `validate-tx-agg-extra`. Specifically, the `mint-fixed` method of `token-in-trait` is invoked to mint an amount (`amt-net`) to the `tx-sender`. This occurs prior to any checks that ensure `token-in-trait` corresponds to a legitimate, whitelisted token contract. Even though subsequent validation in `validate-tx-agg-extra` checks the `token-in` address from `order-details`, the error from this validation is handled by a `match` statement, allowing the transaction to proceed to an error branch (e.g., `refund`) rather than reverting. As a result, a malicious `token-in-trait` can execute arbitrary code during the minting step, regardless of whether validation fails.

In order to exploit these, there are some checks the trait should bypass:

[1] https://cwe.mitre.org/data/definitions/20.html

- Line 150 `check-trait` where `token-in-trait` must be equal to `order-details::token-in`
- Based on the previous checks, there are the checks `order-details::token-in` should bypass to exploit this issue:
  - Line 317 `check-token` where `order-details::swap-token-in` and `order-details::swap-token-in` must be equal or one must be an approved wrapper of the other. Since `cross-router-v2-03::get-approved-wrapped-or-fail` checks against a whitelist, the exploit can only work passing through the first check.

Recommendation

Explicitly validate `token-in-trait` against `validation-data::pair-details::token` before calling it.

Status

Resolved. In the new version of the contract (`meta-peg-in-v2-06h-agg`), this is validated in line 150.

High Severity Issues

No issues found.

Medium Severity Issues

ME-01 Improper Refund Handling in finalize-peg-in-agg

Location

- `./btc-peg-in-v2-07e-agg.clar:154-156`

Description

In the `finalize-peg-in-agg` function, the refund flow is initiated when an error is thrown by `cross-peg-out-v2-01b-agg::validate-transfer-to-swap`. The intended behavior is to refund the full amount, consisting of `amount-net + fee`, to the user. However, prior to the validation check, at line 136, the fee is transferred to the `fee-to-address`. In the event of a refund, this fee is not returned to the contract. As a result, the contract may lack sufficient funds to execute the refund of the total amount (`amount-net + fee`), leading to potential operational failures because of the transaction reverting.

The current flow can be summarized as follows:

1. The contract mints `amount-net + fee` to its own balance.
2. The fee is then transferred from the contract to `fee-to-address`.
3. If validation fails, the refund attempts to return `amount-net + fee`, but the fee is no longer available within the contract, having been sent to `fee-to-address`.

Recommendation

Instead of transferring the fee to `fee-to-address` before validation, move this transfer into the success branch of the validation check.

Status

Resolved. In the new version of the contract (`btc-peg-in-v2-07g-agg`), fee transfer was moved into the success branch.

ME-02 Incorrect Recipient of aBTC Fee

Location

- `./meta-peg-in-v2-06e-agg.clar:149`

Description

In the `finalize-peg-in-agg` function, the `aBTC` fee associated with a peg-in order is not transferred to `fee-to-address` as intended and as implemented in `btc-peg-in-v2-07e-agg.clar`. Instead, the fee is minted to the contract itself. This deviates from the expected behavior, where the fee should be directed to a designated `fee-to-address` for proper accounting and distribution. As a result, `aBTC` accumulates within the `meta-peg-in-v2-06e-agg.clar` contract, leading to an untracked buildup of funds that are not assigned to their intended recipient.

Recommendation

Modify the function to transfer the `aBTC` fee to the `fee-to-address` after minting.

Status

Resolved. In the new version of the contract (`meta-peg-in-v2-06h-agg`), fee is transferred to the fee address.

ME-03 Reverted Peg-in due to Insufficient Balance in finalize-peg-in-agg

Location

- `./meta-peg-in-v2-06e-agg.clar:154`

Description

In the `finalize-peg-in-agg` function, the `meta-peg-out-endpoint-v2-04` contract provides funds for executing non-burnable peg-in orders. However, at line 154, the subtraction operation assumes that the contract's balance is always sufficient to cover `amt-net` without performing a sanity check. If `amt-net` exceeds the contract's available balance, this operation results in an arithmetic underflow, triggering a Clarity runtime error and causing the transaction to fail without a proper code error.

Recommendation

Implement a sanity check to ensure that the `meta-peg-out-endpoint-v2-04` contract's balance is sufficient to fulfill the peg-in order before performing the subtraction.

Status

Resolved. In the new version of the contract (`meta-peg-in-v2-06h-agg`), assertion was added to check peg-out endpoint balance.

Low Severity Issues

No issues found.

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

There is a dependence on the DAO governance mechanism and the centralized registry for token whitelisting, concentrating authority over critical operations like pauses and approvals in a single entity or group.

Upgrades

The contracts do not implement upgradeability patterns.

About CoinFabrik

CoinFabrik is a research and development company specialized in Web3, with a strong background in cybersecurity. Founded in 2014, we have worked on over 500 decentralization projects, including EVM-based and other platforms like Solana, Algorand, and Polkadot. Beyond development, we offer security audits through a dedicated in-house team of senior cybersecurity professionals, working on code in languages such as Substrate, Solidity, Clarity, Rust, TEAL, and Stellar Soroban.

Our team has an academic background in computer science, software engineering, and mathematics, with accomplishments including academic publications, patents turned into products, and conference presentations. We actively research in collaboration with universities worldwide, such as Cornell, UCLA, and École Polytechnique in Paris, and maintain an ongoing collaboration on knowledge transfer and open-source projects with the University of Buenos Aires, Argentina. Our management and people experience team has extensive expertise in the field.

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 two weeks 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

- Race conditions
- Misuse of block timestamps
- Denial of service attacks
- Excessive runtime 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.

Severity Classification

Security risks are classified as follows[2]:

▮ Critical	• Manipulation of governance voting result deviating from voted outcome and resulting in a direct change from intended effect of original results • Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield • Direct theft of any user NFTs, whether at-rest or in-motion, other than unclaimed royalties • Permanent freezing of funds • Permanent freezing of NFTs • Unauthorized minting of NFTs • Predictable or manipulable RNG that results in abuse of the principal or NFT

[2] This classification is based on the smart contract Immunefi severity classification system version 2.3. https://immunefi.com/immunefi-vulnerability-severity-classification-system-v2-3/

	• Unintended alteration of what the NFT represents (e.g. token URI, payload, artistic content) • Protocol insolvency
High	• Theft of unclaimed yield • Theft of unclaimed royalties • Permanent freezing of unclaimed yield • Permanent freezing of unclaimed royalties • Temporary freezing of funds • Temporary freezing NFTs
Medium	• Smart contract unable to operate due to lack of token funds • Block stuffing • Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol) • Theft of gas • Unbounded gas consumption • Security best practices not followed
Low	• Contract fails to deliver promised returns, but doesn't lose value • Other security issues with minor impact

Issue Status

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

- **Unresolved:** The issue has not been resolved.
- **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.

- **Acknowledged:** The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Mitigated:** Implemented actions to minimize the impact or likelihood of the risk.

Disclaimer

Changelog

Date	Description
2025-03-31	Initial report based on commit `365709a9c5d7388d76f09cc63776b385fcc9af20`.
2025-04-01	Final report based on commit `5f6cd61dd126836195898c42f1345b7d78668f44`.