Security Audit Report

XLink - Peg-in Endpoints

November 2024

V202409

Executive Summary

CoinFabrik was asked to audit the contracts for the **XLink Peg-in Endpoints** project.

XLink serves as a bi-directional bridge, enabling the transfer of assets between Bitcoin and its Layer 2 networks. It facilitates interaction with L2 smart contracts using native Bitcoin (BTC) or assets issued on Bitcoin's main network

During this audit we found three critical issues, one medium issue and two minor issues. Also, two enhancements were proposed.

Critical and medium issues were resolved. One minor issue was acknowledged and the other one was partially resolved. The enhancements were not implemented.

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 `c97ffe567d1113475b63d6d6607215b99403a0a8`. Fixes were reviewed on commit `47094431f52927bdc7d220c95a610bf75d23ba20`.

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

- `./btc-peg-in-endpoint-v2-03.clar`: Endpoint for bridging Bitcoin native token (BTC) into the Stacks network.
- `./meta-peg-in-endpoint-v2-02.clar`: Endpoint for bridging BRC-20 tokens from Bitcoin into the Stacks network.
- `./cross-peg-in-endpoint-v2-03.clar`: Endpoint for bridging tokens from other blockchains into the Stacks network.

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	Bridge Hijacking due to Unverified Reveal Transactions	Critical	**Resolved**
CR-02	Incorrect Token Transfer Handling for Non-Burnable Tokens	Critical	**Resolved**
CR-03	Remote Code Execution Through Unvalidated Token Trait Parameter	Critical	**Resolved**
ME-01	Missing Validation of Token Approval Allows Bridging of Disapproved Tokens	Medium	**Resolved**
MI-01	Panicking on Possible Error	Minor	**Acknowledged**
MI-02	Valid Orders Might Be Rejected Due to Insufficient Buffer Size Limits	Minor	**Partially resolved**

Critical Severity Issues

CR-01 Bridge Hijacking due to Unverified Reveal Transactions

Location

- **`./btc-peg-in-endpoint-v2-03.clar: 177, 204`**
- **`./meta-peg-in-endpoint-v2-02.clar: 211, 239, 274, 312, 341`**

Classification

- CWE-345: Insufficient Verification of Data Authenticity[1]

Description

The following functions lack verification for the `reveal-tx` (reveal transaction) parameter to confirm its inclusion in the Bitcoin blockchain:

- **`./btc-peg-in-endpoint-v2-03.clar:`**
  - `finalize-peg-in-cross`
  - `finalize-peg-in-cross-swap`

[1] https://cwe.mitre.org/data/definitions/345.html

- `./meta-peg-in-endpoint-v2-02.clar:`
  - `finalize-drop-peg-in`
  - `finalize-peg-in-cross`
  - `finalize-peg-in-cross-swap`
  - `finalize-peg-in-add-liquidity`
  - `finalize-peg-in-remove-liquidity`

While `commit-tx` (the commit transaction) is verified using an SPV proof, `reveal-tx` is not verified. This omission enables attackers to perform front-running attacks by submitting a fraudulent `reveal-tx` before the legitimate transaction is mined. Consequently, attackers can mint tokens to their own address, undermining the peg-in process's integrity and leading to unauthorized asset creation.

Steps to Exploit

1. Monitor the mempool for `commit-tx` related to the peg-in process.
2. Craft a malicious `reveal-tx` with the attacker's Stacks principal.
3. Submit the malicious `reveal-tx` to the contract before the legitimate transaction is mined.
4. The contract erroneously accepts the unverified reveal transaction, allowing unauthorized minting of tokens to the attacker's address.

Recommendation

The listed functions should be modified to include SPV proof parameters for the reveal-tx, enabling the verification of its inclusion in the Bitcoin blockchain. Implementing a verification step similar to the one employed for `commit-tx` will ensure that only transactions of reveal confirmed in Bitcoin are accepted.

Status

Resolved. In the commit where the fixes were reviewed, `btc-peg-in-endpoint-v2-03.clar` was superseded by `btc-peg-in-endpoint-v2-04.clar`. This new contract removed `finalize-peg-in-cross-swap` and fixed this issue in `finalize-peg-in-cross`.

In `meta-peg-in-endpoint-v2-04.clar`, the new version of `meta-peg-in-endpoint-v2-02.clar`, this issue was fixed on `finalize-peg-in-cross`. `finalize-drop-peg-in`, `finalize-peg-in-cross-swap`, `finalize-peg-in-add-liquidity`, and `finalize-peg-in-remove-liquidity` functions were removed.

CR-02 Incorrect Token Transfer Handling for Non-Burnable Tokens

Location

- `./meta-peg-in-endpoint-v2-02.clar: 567`

Classification

- CWE-840: Business Logic Errors[2]

Description

In `finalize-peg-in-internal`, the contract checks if the token is burnable using the `no-burn` flag. If the token is burnable (`no-burn` is `false`), it correctly mints new tokens to the endpoint itself before routing them to the recipient.

However, when the token is non-burnable (`no-burn` is true), the contract skips the minting step but still attempts to route the tokens. Since the endpoint does not receive any tokens, the transfer fails because the endpoint lacks the necessary token balance.

This issue results in the peg-in process failing for non-burnable tokens, preventing users from receiving their tokens as intended.

Recommendation

Modify the finalize-peg-in-internal function to handle non-burnable tokens correctly by transferring the required amount of tokens from a designated source, such as the registry.

Status

Resolved. The tokens are now transferred from the peg-out endpoint to this contract in order to have the balance for the peg-in request.

CR-03 Remote Code Execution Through Unvalidated Token Trait Parameter

Location

- `./btc-peg-in-endpoint-v2-03.clar: 204-229`

[2] https://cwe.mitre.org/data/definitions/840.html

- `./meta-peg-in-endpoint-v2-02.clar: 211, 239, 274, 312, 341`

Classification

- CWE-20: Improper Input Validation[3]

Description

The functions `finalize-peg-in-cross-swap` in `btc-peg-in-endpoint-v2-03` and `finalize-peg-in-cross-swap` in `meta-peg-in-endpoint-v2-02` are vulnerable to remote code execution due to the lack of validation on the `routing-traits` parameter. By not validating this parameter, attackers can supply any contract that implements the trait.

Within these functions, there is a call to `cross-router-v2-02::route` using `as-contract`, which then calls methods of the first `token` trait in `routing-traits`. This grants the invoked methods the contract's authority, allowing the supplied contract to execute arbitrary code with elevated privileges. This vulnerability enables remote code execution.

This only works when `routing-traits` has only one token. Otherwise, the `amm-helper` will validate the token against its registry.

Possible calls enabled by this remote code execution:

- aBTC contract: mint, burn, and transfer tokens.
- Any wrapped BRC-20: mint, burn, and transfer tokens.
- `meta-bridge-registry-v2-03`: modify requests.
- Other contracts where DAO extensions are authorized.

Recommendation

Ensure that the first token trait in `routing-trait` corresponds to an authorized and trusted contract. Consider implementing a whitelist and checking parameter addresses against it.

Status

Resolved. The vulnerable functions were removed from the contracts.

High Severity Issues

No issues found.

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

Medium Severity Issues

ME-01 Missing Validation of Token Approval Allows Bridging of Disapproved Tokens

Location

- `./meta-peg-in-endpoint-v2-02.clar`

Classification

- CWE-20: Improper Input Validation[4]

Description

The bridge defines a token structure that includes an `approved` field intended to indicate whether a token is permitted to be bridged. However, throughout the contract, this approved field is never checked when bridging tokens. This allows tokens that are not approved to be bridged, undermining the intended input validation mechanisms.

Recommendation

Implement validation checks to enforce the `approved` field before performing any bridge operations.

Status

Resolved. Fixed according to the recommendation.

Minor Severity Issues

MI-01 Panicking on Possible Error

Location

- `./btc-peg-in-endpoint-v2-03.clar: 309`
- `./meta-peg-in-endpoint-v2-02.clar: 277, 436, 515`

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

Classification

- CWE-248: Uncaught Exception[5]

Description

Using `unwrap-panic` results in the transaction being finished because of a runtime error when the provided value is an error or a none. The runtime error does not allow the caller to handle that error and act in response. Also, this kind of error does not provide any information about the reason for the reverted transaction to the user.

While that form is a convenient method to unwrap values, it should not be used unless it is impossible to trigger the panic.

Recommendation

Replace `unwrap-panic` for `unwrap!` when there is a flow which might trigger an error or none value.

Status

Acknowledged. This was acknowledged by the development team.

MI-02 Valid Orders Might Be Rejected Due to Insufficient Buffer Size Limits

Location

- **`./btc-peg-in-endpoint-v2-03.clar: 115, 119, 129, 133, 143, 146`**
- **`./meta-peg-in-endpoint-v2-02.clar: 78, 82, 89, 93, 100, 103, 114, 117`**
- **`./cross-peg-in-endpoint-v2-03.clar: 93, 96, 105, 108, 117, 120`**

Description

The endpoints have encoding and decoding functions for order data. There is a potential issue where a valid order, when serialized using `to-consensus-buff?`, can exceed the buffer sizes expected by the decoding function due to the cumulative size of its fields. For instance, in `create-order-launchpad-or-fail` and `decode-order-launchpad-or-fail`, a 128-byte buffer parameter is defined in the signature of the decoding function, while the values of the order tuple could sum up to 384 bytes. If the serialized order exceeds the buffer size specified in the

[5] https://cwe.mitre.org/data/definitions/248.html

decoding function parameter, the contract will fail to process it, resulting in the rejection of legitimate orders.

Recommendation

Adjust buffer size in the decoding functions parameter to accept greater payloads.

Status

Partially resolved. In the commit where the fixes were reviewed, `btc-peg-in-endpoint-v2-03.clar` was superseded by `btc-peg-in-endpoint-v2-04.clar`. This new contract removed two pairs of encoding and decoding functions and increased the buffer size for the decoding function of the one kept (`decode-order-cross-or-fail`).

`meta-peg-in-endpoint-v2-02.clar` was superseded by `meta-peg-in-endpoint-v2-03.clar`, where the buffer size was increased in three of the four decoding functions (`decode-order-cross-swap-or-fail`, `decode-order-add-liquidity-or-fail`, and `decode-order-remove-liquidity-or-fail`).

In `cross-peg-in-endpoint-v2-03.clar`, the issue persists, but two of the three pairs of functions were removed. Only `create-cross-order` and `decode-cross-order` persist.

Enhancements

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

Id	Title	Status
EN-01	Remove Unused Functions	Not implemented
EN-02	Use secp256k1-verify for Signatures Verification	Not implemented

EN-01 Remove Unused Functions

Location

- **`./btc-peg-in-endpoint-v2-03.clar: 345, 351, 354`**
- **`./meta-peg-in-endpoint-v2-02.clar: 499, 502, 505, 508`**
- **`./cross-peg-in-endpoint-v2-03.clar: 136, 141, 146, 286, 289, 292`**

Description

The contracts implement utility and mathematical functions which are not used. Also, `cross-peg-in-endpoint-v2-03` has functions for a whitelist system which is not enforced.

Recommendation

Remove unused functions.

Status

Not implemented. This was acknowledged by the development team.

EN-02 Use secp256k1-verify for Signatures Verification

Location

- `./cross-peg-in-endpoint-v2-03.clar: 269`

Description

The endpoint verifies signatures by recovering the public key used with `secp256k1-recover?` and then comparing it to a public key. However, the Clarity language already provides a native function for doing it in a single instruction, `secp256k1-verify`.

Recommendation

Simplify the function by using `secp256k1-verify`.

Status

Not implemented. The development team expressed that previously the recommended function had a bug and they are planning to implement it once it is resolved.

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

Both bridges from Bitcoin (`btc-peg-in-endpoint-v2-03` and `meta-peg-in-endpoint-v2-02`) operate without dependence on a privileged role. The contract owner, which will be a DAO, can modify the address where the fees are transferred, set the fee percentages and pause the contracts.

In `cross-peg-in-endpoint-v2-03`, bridge operations depend on signatures from validators. The DAO can only pause the contract.

Upgrades

The contracts do not implement upgradeability mechanisms.

Privileged Roles

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

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 four 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 cost
- 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:

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**

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

This audit report has been conducted on a **best-effort basis within a tight deadline defined by time and budget constraints**. We reviewed only the specific smart contract code provided by the client at the time of the audit, detailed in the Scope section. We do not review other

components that are part of the solution: neither implementation, nor general design, nor business ideas that motivate them.

While we have employed the latest tools, techniques, and methodologies to identify potential vulnerabilities, **this report does not guarantee the absolute security of the contracts, as undiscovered vulnerabilities may still exist.** Our findings and recommendations are suggestions to enhance security and functionality and are not obligations for the client to implement.

The results of this audit are valid solely for the code and configurations reviewed, and any modifications made after the audit are outside the scope of our responsibility. CoinFabrik disclaims all liability for any damages, losses, or legal consequences resulting from the use or misuse of the smart contracts, including those arising from undiscovered vulnerabilities or changes made to the codebase after the audit.

This report is intended exclusively for the **XLink** team and should not be relied upon by any third party without the explicit consent of CoinFabrik. Blockchain technology and smart contracts are inherently experimental and involve significant risk; users and investors should fully understand these risks before deploying or interacting with the audited contracts.

Changelog

Date	Description
2024-11-01	Initial report based on commit `c97ffe567d1113475b63d6d6607215b99403a0a8`.
2024-11-22	Final report based on commit `47094431f52927bdc7d220c95a610bf75d23ba20`.