CoinFabrik

# Security Audit Report

## XLink - EVM Endpoint

April 2025

V202411

# Executive Summary

**CoinFabrik** was asked to audit the contracts for XLink's **EVM Endpoint** .

**During this audit we found two medium issues and two low severity issues. Also, two enhancements were proposed.**

The two medium severity issues were mitigated and the low issues were acknowledged.

# Scope

The audited files are from the git repository located at https://github.com/xlink-network/xlink, in the **`./packages/contracts/bridge-solidity/contracts/`** directory. The audit is based on the commit **`db0661b544895a2a46f4bceb52cb4afab2a89a0a`**.

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

- **`./BridgeEndpointWithSwap.sol`**: Extends `BridgeEndpoint` to facilitate token swaps during cross-chain bridging operations.
- **`./SwapExecutor.sol`**: Utility contract for executing external token swap calls on behalf of its owner.

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 |
|---|---|---|---|
| ME-01 | Swap Payload Injection in finalizeSwap | Medium | **Mitigated** |
| ME-02 | Relayer Can Modify Unsigned Swap Parameters | Medium | **Mitigated** |
| LO-01 | Token Cleanup Sweeps Unrelated Balances | Low | **Acknowledged** |
| LO-02 | ERC20Fixed Fails for Tokens with more than 18 Decimals | Low | **Acknowledged** |

## Critical Severity Issues

No issues found.

## High Severity Issues

No issues found.

## Medium Severity Issues

### ME-01 Swap Payload Injection in finalizeSwap

#### Location

- `./BridgeEndpointWithSwap.sol: 244`

### Classification

- CWE-20: Improper Input Validation[1]

### Description

The `finalizeSwap` function lacks specific access control, allowing any account with the required `tokenIn` and approval to execute it for any pending order. This caller provides the `swapPayload` (dictating function to be called, swap route, etc.), overriding parameters potentially intended by the original validated transaction (though the target DEX address is fixed). The caller must provide the `tokenIn`, which is consumed in the swap, with the resulting `tokenOut` being directed to the original user via the bridge event.

While the attacker sacrifices their `tokenIn`, this vulnerability still allows griefing. An attacker can front-run the legitimate user's `finalizeSwap` call, consuming the order (`sent = true`). The attacker sacrifices `tokenIn + gas` purely to disrupt the user and get a lower value out of manipulating the swap route.

### Recommendation

Either implement an access control mechanism or add a field for the swap payload in `SwapOrderPackage`.

### Status

**Mitigated**. This is a design decision and it is mitigated by always setting `minAmountOut` to non-zero values.

## ME-02 Relayer Can Modify Unsigned Swap Parameters

### Location

- `./BridgeEndpointWithSwap.sol: 150-166`

### Classification

- CWE-20: Improper Input Validation[2]

### Description

The `transferToSwap` function validates bridge orders using validator signatures, however, the cryptographic hash that validators sign does not include the target swap contract address or the

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

`swapPayload` which dictates the swap's execution parameters. This means that the signature verification only confirms the validity of other details like tokens and amounts, but provides no guarantee that the target and swapPayload submitted by the relayer during the transaction are trusted.

A malicious relayer can exploit this by providing their own malicious contract address as the target when calling `transferToSwap`, while still using valid signatures for the rest of the order data. During execution (especially for burnable tokens where the swap is immediate), the `SwapExecutor` approves this malicious target for the full amountIn. The malicious contract can then execute code to transfer out all the approved tokens, while ensuring it returns just enough `tokenOut` (meeting the `amountOutMin` requirement) back to the `SwapExecutor` to pass the necessary checks. The bridge operation then completes, delivering the minimum expected output to the original user, but the rogue relayer profits from the value difference they extracted.

### Recommendation

The Order structure definition should be updated to include the target address and a hash of the swapPayload.

### Status

**Mitigated**. This is a design decision and it is mitigated by always setting `minAmountOut` to non-zero values.

## Low Severity Issues

### LO-01 Token Cleanup Sweeps Unrelated Balances

### Location

- `./BridgeEndpointWithSwap.sol: 308-315`

### Classification

- CWE-682: Incorrect Calculation[3]

### Description

The cleanup logic at the end of `_executeSwap` measures the contract's total balance of `tokenIn` via `balanceOfFixed` and burns/transfers this amount. This is flawed because it does not isolate

[3] https://cwe.mitre.org/data/definitions/682.html

funds related to the specific operation. If there are tokens in the contract balance non-related to the swap operation, this logic actively sweeps these funds, causing their loss.

### Recommendation

The contract should only manage funds explicitly involved in the current transaction.

### Status

**Acknowledged.**.

## LO-02 ERC20Fixed Fails for Tokens with more than 18 Decimals

### Location

- `./utils/ERC20Fixed.sol: 18, 31`

### Description

The library's normalization logic (`10 ** (18 - decimals)`) fails mathematically for tokens with more than 18 decimals due to attempting integer exponentiation with a negative exponent. Transactions involving such tokens via this library will revert without a descriptive error.

### Recommendation

Implement registry checks to prevent adding tokens with > 18 decimals.

### Status

**Acknowledged**. The development team added this to their book of work.

# Enhancements

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

| Id | Title | Status |
|---|---|---|
| EN-01 | Redundant approve(0) for SwapExecutor | Not implemented |
| EN-02 | Misleading NatSpec Comment | Not implemented |

## EN-01 Redundant approve(0) for SwapExecutor

### Location

- `./BridgeEndpointWithSwap.sol: 289`

### Description

`_executeSwap` revokes approval for `swapExecutor` after `swapExecutor` has already finished its execution and pulled the necessary tokens via `transferFrom`. This revocation serves no purpose as the allowance is no longer relevant.

### Recommendation

Remove the approval line from `_executeSwap`.

### Status

**Not implemented**.

## EN-02 Misleading NatSpec Comment

### Location

- `./SwapExecutor.sol: 14`

### Description

The NatSpec comment indicates the contract is "permissionless," but the core `executeSwap` function has an `onlyOwner` modifier.

### Recommendation

Update the comment for accuracy.

### 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

The contract Owner has the ability to pause/unpause operations and manage allowlists. Furthermore, the system relies on a centrally managed `BridgeRegistry` for critical functions like approving and managing the sets of Validators (who sign bridge messages) and Relayers (who submit bridge transactions), as well as controlling supported tokens, fees, and limits.

## Upgrades

The contracts in scope do not implement upgradeability mechanisms.

## Protocol Context and Off-Chain Handling

These audited contracts function as the swap execution component within a larger cross-chain bridge protocol. Orders are validated off-chain by Validators and submitted on-chain by permissioned Relayers. The handling of swap failures (e.g., reverts from `SwapExecutor`, insufficient output detected in `_executeSwap`) and any subsequent user refund or reconciliation processes are managed by off-chain components (Validators, Relayers, backend systems) triggered by events like `SwapExecutorError` and the use of `bridgePayloadFailure` in `SendMessageWithTokenEvent`. This off-chain logic is outside the scope of this EVM contract audit.

# 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. 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
- 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.

# Severity Classification

Security risks are classified as follows[4]:

| | |
|---|---|
| Critical | • Manipulation of governance voting result deviating from voted outcome and resulting in a direct change from intended effect of original results<br>• Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield<br>• Direct theft of any user NFTs, whether at-rest or in-motion, other than unclaimed royalties<br>• Permanent freezing of funds<br>• Permanent freezing of NFTs<br>• Unauthorized minting of NFTs<br>• Predictable or manipulable RNG that results in abuse of the principal or NFT<br>• Unintended alteration of what the NFT represents (e.g. token URI, payload, artistic content)<br>• Protocol insolvency |
| High | • Theft of unclaimed yield<br>• Theft of unclaimed royalties<br>• Permanent freezing of unclaimed yield<br>• Permanent freezing of unclaimed royalties<br>• Temporary freezing of funds<br>• Temporary freezing NFTs |

[4] 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/

| | |
|---|---|
| Medium | • Smart contract unable to operate due to lack of token funds<br>• Block stuffing<br>• Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)<br>• Theft of gas<br>• Unbounded gas consumption<br>• Security best practices not followed |
| Low | • Contract fails to deliver promised returns, but doesn't lose value<br>• 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

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 |
|---|---|
| 2025-04-14 | Initial report based on commit `db0661b544895a2a46f4bceb52cb4afab2a89a0a`. |
| 2025-04-16 | Final report based on development team feedback. |