ALEX XLink Audit

MultisigWallet and BridgeToken

June 2024

By CoinFabrik

v202406

Executive Summary **3**
Scope **3**
Contracts Descriptions **3**
Methodology **4**
Findings **5**
Severity Classification 5
Issues Status 5
Critical Severity Issues 6
High Severity Issues 6
HI-01 Transaction Submitter can Confirm Twice 6
Medium Severity Issues 7
Minor Severity Issues 7
MI-01 Single-confirmation Multisig can be Initialized 7
Enhancements 8
EN-01 Unused Role and Function 8
EN-02 Unused Modifier 8
Other Considerations **9**
Centralization 9
Upgrades 9
Privileged Roles 9
MultisigWallet 9
BridgeToken 10
Changelog **10**

Executive Summary

CoinFabrik was asked to audit the contracts for the ALEX XLink project.

During this audit we found one high issue and one minor issue. Also, two enhancements were proposed.

Scope

The audited files are from the git repository located at https://github.com/xlink-network/xlink. The audit is based on the commit `dca10704a938e2820695e4a6d9ab15283f3f57e2`.

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

- `packages/contracts/bridge-solidity/contracts/MultisigWallet.sol`: Multisig wallet that requires a predefined number of approvals by designated signers to execute transactions.
- `packages/contracts/bridge-solidity/contracts/tokens/BridgeToken.sol`: Bridge token with various features such as minting, burning, snapshots, pausing, and transfer restrictions, managed by roles and governed by the multisig wallet.

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.

Contracts Descriptions

MultisigWallet

The `MultisigWallet` is designed to manage a wallet where transactions require multiple confirmations before execution. It inherits from `AccessControlEnumerable` to manage roles and permissions more effectively. The contract has specific roles: `APPROVED_ROLE` and `SIGNER_ROLE`.
Key functionalities include:

1. **Deposits and Balance Management**: It allows ether deposits and maintains the wallet balance.
2. **Transaction Submission and Confirmation**: Users with the `SIGNER_ROLE` can submit transactions, which are stored in an array until a sufficient number of confirmations are acquired.
3. **Transaction Execution**: Once the required confirmations are met, the transaction can be executed.

4. **Revoke Confirmation**: A signer can revoke their confirmation if the transaction is not yet executed.

Modifiers ensure that transactions exist, are not executed prematurely, and are confirmed by authorized users only.
This contract is well-suited for managing funds requiring a higher level of security as multiple signers must confirm transactions, reducing the risk of unauthorized movements.

BridgeToken

The `BridgeToken` contract defines a specialized ERC20 token with various additional features:

1. **Minting and Burning**: The token supports controlled minting and burning via the `MINTER_ROLE`.
2. **Snapshot Mechanism**: It includes snapshot capabilities to capture the token's state at specific blocks.
3. **Pausing**: The contract can be paused by the owner, leveraging `Pausable` to halt activities during maintenance or emergencies.
4. **Transfer Restrictions**: Transfers can be globally enabled or disabled by the owner.

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

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	Transaction Submitter can Confirm Twice	High	Unresolved
MI-01	Single-confirmation Multisig can be Initialized	Minor	Unresolved

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 Transaction Submitter can Confirm Twice

Location:

- `packages/contracts/bridge-solidity/contracts/MultisigWallet.sol: 121`

Classification:

- CWE-284: Improper Access Control[1].

In the `MultisigWallet` contract, the `submitTransaction()` function initializes a new transaction with `numConfirmations` set to `1`, implying that it already has one confirmation. However, it does not update the `isConfirmed` mapping to reflect that the submitter has confirmed the transaction. This creates a vulnerability because the submitter can call `confirmTransaction()` on the same transaction, effectively adding another confirmation. This bypasses the intended security measure by allowing the submitter to count their vote twice: once implicitly by submission and once explicitly by confirmation.

This vulnerability allows a single user to add multiple confirmations to a transaction, undermining the multisig security model which is designed to require confirmations from multiple distinct signers. This increases the risk of unauthorized or malicious transactions being executed.

Recommendation

To address this issue, update the `submitTransaction()` function to set the `isConfirmed` mapping for the submitter's address, ensuring they cannot confirm it again.

[1] https://cwe.mitre.org/data/definitions/284.html

Status

Unresolved.

Medium Severity Issues

No issues found.

Minor Severity Issues

MI-01 Single-confirmation Multisig can be Initialized

Location:

- `packages/contracts/bridge-solidity/contracts/MultisigWallet.sol: 68`

Classification:

- CWE-287: Improper Authentication[2].

In the `MultisigWallet` contract, during the initialization (constructor), the parameter `_numConfirmationsRequired` can be set to `1`. This means that only one confirmation is needed to approve and execute a transaction, effectively bypassing the fundamental "multi-signature" aspect of the wallet. This vulnerability circumvents the intended security mechanism, significantly increasing the risk of unauthorized transactions.

By allowing the required number of confirmations to be set to just one, the contract's security is compromised. A single signer has the power to approve and execute any transaction, which defeats the purpose of having a multisig wallet designed for higher security through multiple approvals. This means that if any signer's key is compromised, the entire wallet can be drained or used maliciously.

Recommendation

To mitigate this issue, enforce a minimum number of confirmations greater than one within the constructor.

Status

Unresolved.

[2] https://cwe.mitre.org/data/definitions/287.html

Enhancements

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

ID	Title	Status
EN-01	Unused Role and Function	Not implemented
EN-02	Unused Modifier	Not implemented

EN-01 Unused Role and Function

Location:

- `packages/contracts/bridge-solidity/contracts/MultisigWallet.sol: 9, 92`

In the `MultisigWallet` contract, the `APPROVED_ROLE` is defined and used within the `isApproved()` function. However, the `isApproved()` function itself is not utilized anywhere in the contract, making the `APPROVED_ROLE` redundant. Having unused code can lead to bloat and potential confusion for future maintainers. It also increases the contract's bytecode size unnecessarily, which could increase deployment.

Recommendation

To enhance the contract's quality and maintainability, remove the unused `APPROVED_ROLE` definition and the `isApproved()` function. This simplification makes the code cleaner and more efficient.

Status

Not implemented.

EN-02 Unused Modifier

Location:

- `packages/contracts/bridge-solidity/contracts/tokens/BridgeToken.sol: 9, 92`

In the `BridgeToken` contract, the `notContract` modifier is defined but not used anywhere in the contract. This modifier is intended to restrict access to specific functions, ensuring that only externally owned accounts (EOAs) and not contracts can call certain functions. However, since it is not applied to any functions, it does not serve any functional purpose.

Recommendation

To improve the code quality and maintainability, either remove the `notContract` modifier if it is not necessary or apply the modifier to the intended functions if you plan to restrict access to them.

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

Both the `MultisigWallet` and `BridgeToken` contracts exhibit elements of centralization, albeit in different ways. The `MultisigWallet` contract, while designed to increase security through multi-signature approvals, centralizes control among a specified group of signers, meaning that decisions and transactions still depend on a limited set of authorized participants. On the other hand, the `BridgeToken` contract includes several centralized control features, such as role-based access control for minting, pausing, and transferring capabilities, all managed by a multisig wallet owner. Although such centralization can enhance security and manageability, it also introduces trust dependencies.

Upgrades

Neither the `MultisigWallet` nor the `BridgeToken` contracts implement any standard upgradeability patterns.

Privileged Roles

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

MultisigWallet

APPROVED_ROLE

- Defined but not actively used in the contract's logic.

SIGNER_ROLE

- Assigned to the wallet's signers.
- Allows submitting, confirming, revoking, and executing transactions.

BridgeToken

DEFAULT_ADMIN_ROLE

- Initially granted to the multisig wallet owner address.
- Allows managing roles and performing administrative tasks such as pausing, unpausing, and taking snapshots.

MINTER_ROLE

- Allows the minting of new tokens.

Changelog

- 2024-06-10 – Initial report based on commit `dca10704a938e2820695e4a6d9ab15283f3f57e2`.

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.