CoBridge Security Review

Aura Audits

June 19th 2025 - June 25th 2025

About

Aura Audits is a team of highly skilled smart contract security researchers dedicated to securing blockchain protocols. With a proven track record of uncovering numerous vulnerabilities, our experts deliver a thorough and detail-oriented audit process. While perfect security cannot be guaranteed, we commit to applying our extensive knowledge and meticulous approach to ensure your protocol's robustness.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

About Cobridge

Bridge Contract Summary: This is a cross-chain bridge contract that allows users to initiate transfers by depositing ETH with calculated fees and provides an authorized release mechanism on the destination chain. The contract implements signature-based validation for transfer requests with expiration timestamps, owner-controlled parameters (min/max amounts, fees), and maintains counters for initiated and released transactions. Key functionality includes fee calculation based on percentage scaling, signature verification using ECDSA recovery, and administrative controls for enabling/disabling the bridge operations.

Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - leads to a moderate material loss of assets in the protocol or moderately harms a group of users.
- Low - leads to a minor material loss of assets in the protocol or harms a small group of users.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

Findings Index

Severity	Name	Status
[H-01]	Account Abstraction Wallet Users Can Lose Funds During Bridging function	Close
[M-01]	Malicious Actor Can Execute Signature Replay Attacks	Close
[L-01]	Missing Sanity Checks on Scaled Fee Percent	Close
[L-02]	Improper Event Indexing	Close
[L-03]	Missing Reentrancy Guard in withdraw and release	Acknowledged
[L-04]	Outdated Pragma	Close
[L-05]	Centralization Risk	Acknowledged

Findings

[H-01] Account Abstraction Wallet Users Can Lose Funds During Bridging function

Severity

Impact:
High

Likelihood:
Medium

Vulnerability Details

The Bridge contract's bridging function assumes msg.sender will have the same address on L2, which fails for Account Abstraction (AA) wallets that use different addresses across chains. This causes bridged tokens to be sent to an unreachable address on L2, permanently locking user funds.

The Bridge contract's bridging functionality contains a critical flaw when interacting with Account Abstraction (AA) wallets. The vulnerability arises in the initiate() function, where the receiver (on different chain) is set to msg.sender as seen during event emission without accounting for potential address differences across chains for AA wallet users. Since AA wallets can have different addresses on different chains, this implementation may cause bridged tokens to be sent to an address the user does not control on the destination chain, resulting in permanent loss of funds.

```
function initiate(
        uint256 destinationChainId,
        uint256 amountIn,
        uint256 amountOut,
        uint256 signatureExpirationTimestamp,
        bytes calldata signature
    ) external payable {
        require(isEnabled, "The bridge is disabled");
        require(block.chainid != destinationChainId, "Invalid destinatior
        require(amountIn >= minAmountIn, "Value is less that minAmountIn"
        require(amountIn <= maxAmountIn, "Value is more that maxAmountIn"
        require(signatureExpirationTimestamp >= block.timestamp, "Signatu
        uint256 calculatedFee = calculateFee(amountIn);
        require(msg.value == amountIn + calculatedFee, "Incorrent value")
        require(
            checkSignature(
                msg.sender,
                block.chainid.toString(),
                destinationChainId.toString(),
                amountIn.toString(),
                amountOut.toString(),
                calculatedFee.toString(),
                signatureExpirationTimestamp.toString(),
                signature
            ),
            "Invalid signature"
        );
        emit Initiated(
            msg.sender,
            block.chainid,
            destinationChainId,
            amountIn,
            amountOut,
            calculatedFee,
            block.timestamp
        );
        initialsCounter++;
    }
```

Impact

AA wallet users risk losing their tokens permanently when bridging to different chains. The tokens will be locked on the destination chain under an address derived from a different key scheme than the user's AA wallet, making recovery impossible.

Recommendation

To prevent fund loss for AA wallet users, the contract should implement an explicit address resolution mechanism. Instead of hardcoding msg.sender as the recipient, users should provide their intended L2 address as a parameter, with appropriate validation to ensure they retain control.

[M-01] Malicious Actor Can Execute Signature Replay Attacks

Severity

Impact:
Medium
Likelihood:
Medium

Vulnerability Details

The Bridge contract contains a critical vulnerability in its signature verification mechanism that allows malicious actors to execute replay attacks by reusing previously valid signatures. The vulnerability stems from the inadequate signature validation logic in the `checkSignature()` function, which fails to implement proper nonce management and signature tracking mechanisms as outlined in EIP-712 standards.

The contract constructs signature messages by concatenating string representations of transaction parameters using a simple separator-based approach. This method lacks several essential security components that prevent signature reuse. The signature verification process in `checkSignature()` creates a message hash by combining the sender address, source and destination chain IDs, amounts, fees, and expiration timestamp into a single byte array using abi.encodePacked(). However, this implementation does not include any unique transaction identifiers, nonces, or mechanisms to track previously used signatures that would prevent the same signature from being valid across multiple transactions.

Impact

While the current implementation of the Bridge contract requires users to provide msg.value equal to amountIn + calculatedFee in the initiate() function, which limits the immediate exploitability of signature replay attacks, this vulnerability represents a significant security design flaw that could become critical in future contract iterations or upgrades.

The primary concern lies in the fundamental weakness of the signature verification system, which violates established security principles for cryptographic authorization. This design flaw creates

potential attack vectors that could be exploited if the contract undergoes modifications, upgrades, or if similar signature verification logic is reused in other contexts where payment requirements differ.

Recommendation

The vulnerability requires immediate remediation through implementation of EIP-712 structured data signing standards combined with robust nonce management and signature tracking mechanisms. The current string-based signature construction should be completely replaced with a standardized approach that provides cryptographic protection against replay attacks.

[L-01] Missing Sanity Checks on Scaled Fee Percent

Severity

Impact:
Low

Likelihood:
Low

Vulnerability Detail

The contract allows the owner to set scaledFeePercent to an arbitrary value without bounds. This could lead to fee values exceeding 100% or becoming abusive.

Impact

Users may unknowingly pay excessive fees, even exceeding the transferred amount.

Recommendation

Add a maximum cap on scaledFeePercent. Example (for max 100% fee):

require(newScaledFeePercent <= 100 * FEE_PERCENT_SCALING_FACTOR, "Fee exceeds 100%");

[L-02] Improper Event Indexing

Severity

Impact:
Low
Likelihood:
Low

Vulnerability Details

Key event parameters like accountAddress in Initiated and Released are not indexed, reducing their utility in off-chain indexing or event log searches.

Impact

Difficulty in tracking and filtering bridge activity on explorers or subgraphs

Recommendation

Index important parameters like so:

```
event Initiated(address indexed accountAddress, ...);
event Released(address indexed accountAddress, ...);
```

[L-03] Missing Reentrancy Guard in withdraw and release

Severity

Impact:
Low

Likelihood:
Low

Vulnerability Details

The Withdrawable.withdraw and Bridge.release functions transfer ETH using low-level .call without reentrancy protection.

Impact

affect the integrity of bridge operations and funds reserved for legitimate users.

Recommendation

Add ReentrancyGuard from OpenZeppelin and apply nonReentrant to withdraw and releas

[L-04] Outdated Pragma

Severity

Impact:
Low

Likelihood:
Low

Vulnerability Details

The smart contract is using an outdated version of the Solidity compiler specified by the pragma directive i.e. 0.8.26. Solidity is actively developed, and new versions frequently include important security patches, bug fixes, and performance improvements.

Impact

Using an outdated version exposes the contract to known vulnerabilities that have been addressed in later releases. Additionally, newer versions of Solidity often introduce new language features and optimizations that improve the overall security and efficiency of smart contracts.

Recommendation

It is suggested to use the 0.8.29 pragma version.

[L-05] Centralization Risk

Severity

Impact:
Low

Likelihood:
Low

Vulnerability Details

The current design of the contracts introduces significant centralization risk by concentrating critical powers in the hands of the owner and the authorized signer. The owner has unilateral authority over key operational and financial parameters, including:

Configuring fees through setMinFee, setScaledFeePercent, setMinAmountIn, and setMaxAmountIn.

Enabling or disabling the bridge entirely via setEnabled. Designating or changing the authorizedAddress, which controls fund releases. Withdrawing all funds from the contract at any time.

The authorized signer (as set by the owner) has full control over the release of funds to users through the release function.

Impact

If the private key of the authorized signer is compromised or the owner acts maliciously, funds can be drained or bridged arbitrarily.

Recommendation

Implement a multi-signature wallet (e.g., Gnosis Safe) as the owner.