Security Assessment Report

Aave Gho CCIP Update

September-2025

Prepared for:
Aave DAO

Code developed by:
Token Logic

Table of contents

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Aave Gho CCIP Update	https://github.com/bgd-labs/aave-helpers	9ae7f2b	EVM

Project Overview

This document describes the security review of **Aave Gho CCIP Update** code using manual codereview. The work was undertaken from **September 22** to **September 24.**

The following contracts are considered in scope for this review:

- `src/bridges/ccip/AaveGhoCcipBridge.sol`
- `src/dependencies/chainlink/CCIPReceiver.sol`

The team performed a manual audit of all the solidity contracts. Issues discovered during the review are listed in the following pages.

Protocol Overview

The Aave GHO CCIP Bridge enables cross-chain transfers of the GHO stablecoin using Chainlink's Cross-Chain Interoperability Protocol (CCIP). On the source chain, the contract escrows GHO, constructs a CCIP message, and sends it via the CCIP router. On the destination chain, the bridge receives the message through the off-ramp and releases GHO to the Aave Collector. Failed transfers are recorded and can be recovered by the owner.

This review focuses on the **AaveGhoCcipBridge contract**, which is intended to be deployed across all supported chains as part of the bridging architecture.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	-	-	-
Low	-	-	-
Informational	4	4	3
Total	4	4	3

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

Detailed Findings

ID	Title	Severity	Status
I-01	Missing sanity check in processMessage for destination token	Informational	Fixed
I-02	Failed message recovery does not clear storage, leaves partial state and increases gas costs	Informational	Fixed
I-03	Missing Balance Check for ERC20 Fee Tokens	Informational	Fixed
I-04	Replace Ownable with Ownable2Step for safer ownership transfers	Informational	Acknowledged

Informational Issues

I-01. Missing sanity check in processMessage for destination token

File: `AaveGhoCcipBridge.sol`

Description:
`processMessage` currently reads `message.destTokenAmounts` and transfers the contract's local `GHO_TOKEN` without validating that the delivered destination token matches `GHO_TOKEN`.

Although the contract validates the message source and CCIP delivery is atomic, a token pool misconfiguration or an unexpected message could cause a semantic mismatch where the contract attempts to pay out GHO even though CCIP did not deliver destination-side GHO.

Impact:
This issue is unlikely if CCIP mappings and source validation are correct, but a mismatch could still cause failed processing, unintended transfers, or unnecessary on-chain reverts.

Recommendation:
Add a minimal sanity check in `processMessage` that only pays out when `message.destTokenAmounts[0].token == GHO_TOKEN`, and revert or mark the message failed if no matching entry exists.
This documents the GHO-only intent in code and provides defense in depth against unlikely token-pool misconfiguration.

Customer's response: Fixed here.

Fix Review: Fix looks good.

I-02. Failed message recovery does not clear storage, leaves partial state and increases gas costs

File: AaveGhoCcipBridge.sol

Description:
In `recoverFailedMessageTokens`, the contract transfers failed tokens to the collector but does not delete the array `_failedTokenTransfers[messageId]`. As a result, the data remains permanently in storage even though it is no longer needed.
According to the Solidity documentation on the delete keyword, deleting storage variables refunds gas by clearing the slot.

Impact:
Accumulated stale entries left in storage gradually increase the contract's overall state size, leading to storage bloat.
At the same time, the contract forfeits potential gas refunds that would normally be issued by the EVM when unused storage is cleared, making operations more costly than necessary.

Recommendation:

After successfully recovering tokens, explicitly delete the storage entry for the failed message:

```javascript
delete _failedTokenTransfers[messageId];
```

This reclaims storage, reduces state size, and ensures the contract receives the gas refund.

Customer's response: Fixed here.

Fix Review: Fix looks good.

I-03. Missing Balance Check for ERC20 Fee Tokens

File: AaveGhoCcipBridge.sol

Description:
In the `send` function, when the fee is paid in ERC20 tokens `(i.e., feeToken != address(0))`, the contract only increases the allowance of `ROUTER` using safeIncreaseAllowance but does not verify that the contract holds a sufficient balance of the fee token. This differs from the native fee case `(feeToken == address(0))`, where an explicit balance check is performed.

Impact:
If the contract lacks sufficient fee tokens, the subsequent ccipSend call will revert, resulting in wasted gas and unclear error reporting.

Recommendation:

Add a balance check similar to the native fee case to ensure the contract holds enough fee tokens before increasing allowance.

Customer's response: Fixed here.

Fix Review: Fix looks good.

I-04. Replace Ownable with Ownable2Step for safer ownership transfers

File: AaveGhoCcipBridge.sol

Description:
The contract currently inherits `Ownable`, which performs single-step ownership transfers using `transferOwnership(newOwner)`. This allows accidental or malicious immediate transfers if a wrong address is supplied. Using `Ownable2Step` makes ownership transfer a two-step process, requiring the new owner to explicitly accept ownership, which prevents accidental lockout and reduces risk during governance changes or automated deployments.

Impact:
A mistaken `transferOwnership` call to the zero address or to an incorrect address can permanently lock critical admin functionality

Recommendation:

Replace `Ownable` with `Ownable2Step` and update ownership-change flows to the two-step pattern: `transferOwnership(newOwner)`, followed by `newOwner.acceptOwnership()`.

Customer's response: Acknowledged. We have decided on using the simple Ownable as it's controlled by the Executor with thorough reviews, and managed by governance.

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.