Security Assessment & Formal Verification Report

Modular Gho Stewards

September 2024

certora

aave

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
gho-core	https://github.com/aave/gho-core	c82c63b	EVM/Solidity 0.8.10

Project Overview

This document describes the specification and verification of Modular Gho Stewards using the Certora Prover and manual code review findings. The work was undertaken in September 2024.

The following contract list is included in our scope:

- **GhoAaveSteward.sol**
- **GhoBucketSteward.sol**
- **GhoCcipSteward.sol**
- **GhoGsmSteward.sol**
- **RiskCouncilControlled.sol**
- **FixedFeeStrategyFactory.sol**

The Certora Prover demonstrated that the implementation of the Solidity contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all the Solidity contracts. During the verification process and the manual audit, the Certora team discovered bugs in the Solidity contracts code, as listed below.

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	-	-	-
Low	2	2	2
Informational	2	2	2
Total	4	4	4

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
L-01	Unintentional temporary lock of update functionality	Low	Fixed
L-02	Council is granted power to setRiskConfig()	Low	Fixed
I-01	Dead storage variables	Informational	Fixed
I-02	Redundant code	Informational	Fixed

Low Severity Issues

L-01 Unintentional temporary lock of update functionality

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: Multiple files	Status: Fixed	

Description: The functions `UpdateWithinRange`, `isDiffLowerThanMax` across the contracts allow a change of 0% (`to == from`). In this case, if the update is to the current value (either by a mistake or on purpose) the delay will be enforced without any change and event will be emitted on the corresponding update function of the component (gho token, ccip, gsm, etc.)

Exploit Scenario: There's no concern of exploitation, but this is a clear undesired behaviour in an edge case call. If an update function is ever called with a 0 change, the timelock will temporarily disable the ability to update for the entire update delay time, although the call is equivalent to not executing the call. A redundant error will be emitted.

Recommendations: Revert on calls that enforce 0 change.

Customer's response:

The functions' logic has been modified to revert if no change occurs.

L-02 Council is granted power to setRiskConfig()

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: GhoAaveSteward.sol	Status: Fixed	

Description: `setRiskConfig()` should be an `ownerOnly` function rather than a `riskCouncilOnly` function. In the current state, the risk council can decide its own limit; however, the point is that the steward will be limited by a DAO decision.

Exploit Scenario: There's no realistic concern of exploitation, however the steward is a limited mean for making changes to gho related configurations. These elevated permissions were granted to the risk council by the DAO with intentional restrictions. The ability of the risk council to modify the restrictions as they desire, defiles the purpose of the imposed restrictions.

Recommendations: Set the `setRiskConfig()` function with an `ownerOnly` permission rather than `riskCouncilOnly`.

Customer's response:

The function modifier has been fixed to ensure that only the DAO can modify risk configuration parameters.

Informational Issues

I-01 Dead storage variables		
Severity: **Informational**	Impact: **informational**	Likelihood: **informational**
Files: GhoAaveSteward.sol	Status: Fixed	

Description:

1. The `minDelay` var of each IR config isn't used at all in the contract. Instead, the `notTimelock` modifier checks against the global `MIN_DELAY`.

2. `GHO_BORROW_RATE_CHANGE_MAX` isn't used anywhere. The current mechanism specifies a max percentage for each variable within the struct rather than a global one.

Recommendations:

1. Either remove the IR config-specific delay or check against this value in the time lock modifier.
2. Remove the global max borrow rate change.

Customer's response:

The `minDelay` has been removed from IR config struct. The global `GHO_BORROW_RATE_CHANGE_MAX` has also been removed.

I-02 Redundant code

Severity: **Informational**	Impact: **informational**	Likelihood: **informational**
Files: GhoAaveSteward.sol	Status: Fixed	

Description: In the function `_validateRatesUpdate` it's redundant to check that the IR is not 0 in `getReserveData()`, and therefore retrieve the reserve data. This is because the final update call is using the `poolConfigurator`'s `setReserveInterestRateData()`, which tries to call `setInterestRateParams()` through a cast of the IR stored in the data. If the address is 0, the call will fail and revert.

Recommendations: The code can be removed.

Customer's response:

This code has been removed as it is unnecessary.

Formal Verification

Assumptions and Simplifications

Project General Assumptions

- Loop unrolling: We assume any loop can have at most 1 iteration, except for specific multireward properties.
- View functions filtering: Rules checking state changes of all available functions do not check view functions.

Verification Notations

✅Indicates the rule is formally verified.
❌Indicates the rule is violated.

Formal Verification Properties

GhoAaveSteward

✅ 1. `ghoBorrowRateLastUpdate__updated_only_by_updateGhoBorrowRate`
Update of `ghoBorrowRateLastUpdate` is only occurring by calling to `updateGhoBorrowRate()`.

✅ 2. `updateGhoBorrowRate_update_correctly__ghoBorrowRateLastUpdate`
A call to `updateGhoBorrowRate()` updates `ghoBorrowRateLastUpdate` to the timestamp at time of the call.

✅ 3. `updateGhoBorrowRate_timelock`
A call to `updateGhoBorrowRate()` succeed (non-reverting) only if the timestamp at time of call surpassed `ghoBorrowRateLastUpdate` by at least `MIN_DELAY`.

✅ 4. `ghoBorrowCapLastUpdate__updated_only_by_updateGhoBorrowCap`
Update of `ghoBorrowCapLastUpdate` is only occurring by calling to `updateGhoBorrowCap()`.

✅ 5. `updateGhoBorrowCap_update_correctly__ghoBorrowCapLastUpdate`
A call to `updateGhoBorrowCap()` updates `ghoBorrowCapLastUpdate` to the timestamp at time of the call.

✅ 6. `updateGhoBorrowCap_timelock`
A call to `updateGhoBorrowCap()` succeed (non-reverting) only if the timestamp at time of call surpassed `ghoBorrowCapLastUpdate` by at least `MIN_DELAY`.

✅ 7. `ghoSupplyCapLastUpdate__updated_only_by_updateGhoSupplyCap`
Update of ghoSupplyCapLastUpdate is only occurring by calling to updateGhoSupplyCap().

✅ 8. updateGhoSupplyCap_update_correctly__ghoSupplyCapLastUpdate
A call to `updateGhoSupplyCap()` updates `ghoSupplyCapLastUpdate` to the timestamp at time of the call.

✅ 9. `updateGhoSupplyCap_timelock`
A call to `updateGhoSupplyCap()` succeed (non-reverting) only if the timestamp at time of call surpassed `ghoSupplyCapLastUpdate` by at least `MIN_DELAY`.

✅ 10. `only_RISK_COUNCIL_can_call__updateGhoBorrowCap`
`updateGhoBorrowCap()` will be successful (non-reverting) only if the caller was the risk council.

✅ 11. `only_RISK_COUNCIL_can_call__updateGhoBorrowRate`
`updateGhoBorrowRate()` will be successful (non-reverting) only if the caller was the risk council.

✅ 12. `only_RISK_COUNCIL_can_call__updateGhoSupplyCap`
`updateGhoSupplyCap()` will be successful (non-reverting) only if the caller was the risk council.

✅ 13. `only_owner_can_call__setBorrowRateConfig`
setBorrowRateConfig() will be successful (non-reverting) only if the caller was the owner.

✅ 14. `updateGhoBorrowCap__correctness`
1. `updateGhoBorrowCap()` updates the `borrowCap` on storage to the passed value.
2. The new value of `borrowCap` is no more than double the current value (as per current specification).

✅ 15. `updateGhoSupplyCap__correctness`
1. `updateGhoSupplyCap()` updates the `supplyCap` on storage to the passed value.
2. The new value of `supplyCap` is no more than double the current value (as per current specification).

✅ 16. `updateGhoBorrowRate__correctness`
The new borrow rate values (base, slope1, slope2) enforced by calling `updateGhoBorrowRate()` cannot surpass `GHO_BORROW_RATE_MAX`.

GhoBucketSteward

✅ 17. `timestamp__updated_only_by_updateFacilitatorBucketCapacity`
Update of `_facilitatorsBucketCapacityTimelocks` is only occurring by calling to `updateFacilitatorBucketCapacity()`.

✅ 18. `updateFacilitatorBucketCapacity_update_correctly__timestamp`
A call to `updateFacilitatorBucketCapacity()` updates `_facilitatorsBucketCapacityTimelocks` to the timestamp at time of the call.

✅ 19. `updateFacilitatorBucketCapacity_timelock`
A call to `updateFacilitatorBucketCapacity()` succeed (non-reverting) only if the timestamp at time of call surpassed `_facilitatorsBucketCapacityTimelocks` by at least `MIN_DELAY`.

✅ 20. `only_RISK_COUNCIL_can_call__updateFacilitatorBucketCapacity`
`updateFacilitatorBucketCapacity()` will be successful (non-reverting) only if the caller was the risk council.

✅ 21. `only_owner_can_call__setControlledFacilitator`
setControlledFacilitator() will be successful (non-reverting) only if the caller was the owner.

✅ 22. `updateFacilitatorBucketCapacity__correctness`

1. `updateFacilitatorBucketCapacity()` updates the `bucketCapacity` on storage to the passed value.
2. The new value of `bucketCapacity` is no more than double the current value (as per current specification).

GhoCcipSteward

✅ 23. `bridgeLimitLastUpdate__updated_only_by_updateBridgeLimit`
Update of `bridgeLimitLastUpdate` is only occurring by calling to `updateBridgeLimit()`.

✅ 24. `updateBridgeLimit_update_correctly__bridgeLimitLastUpdate`
A call to `updateBridgeLimit()` updates `bridgeLimitLastUpdate` to the timestamp at time of the call.

✅ 25. `updateBridgeLimit_timelock`
A call to `updateBridgeLimit()` succeed (non-reverting) only if the timestamp at time of call surpassed `bridgeLimitLastUpdate` by at least `MIN_DELAY`.

✅ 26. `rateLimitLastUpdate__updated_only_by_updateRateLimit`
Update of `rateLimitLastUpdate` is only occurring by calling to `updateRateLimit()`.

✅ 27. `updateRateLimit_update_correctly__rateLimitLastUpdate`
A call to `updateRateLimit()` updates `rateLimitLastUpdate` to the timestamp at time of the call.

✅ 28. `updateRateLimit_timelock`
A call to `updateRateLimit()` succeed (non-reverting) only if the timestamp at time of call surpassed `rateLimitLastUpdate` by at least `MIN_DELAY`.

✅ 29. `only_RISK_COUNCIL_can_call__updateBridgeLimit`
`updateBridgeLimit()` will be successful (non-reverting) only if the caller was the risk council.

✅ 30. `only_RISK_COUNCIL_can_call__updateRateLimit`
`updateRateLimit()` will be successful (non-reverting) only if the caller was the risk council.

✅ 31. `updateBridgeLimit__correctness`
The new values of each of the following `outboundCapacity`, `outboundRate`, `inboundCapacity`, `inboundRate` is updated to no more than double their current value after a successful call to `updateRateLimit()` (as per current specification).

GhoGsmSteward

✅ 32. `gsmExposureCapLastUpdated__updated_only_by_updateGsmExposureCap`

Update of gsmExposureCapLastUpdated is only occurring by calling to updateGsmExposureCap().

✅ 33. updateGsmExposureCap_update_correctly__gsmExposureCapLastUpdated
A call to updateGsmExposureCap() updates gsmExposureCapLastUpdated to the timestamp at time of the call.

✅ 34. updateGsmExposureCap_timelock
A call to updateGsmExposureCap() succeed (non-reverting) only if the timestamp at time of call surpassed gsmExposureCapLastUpdated by at least MIN_DELAY.

✅ 35. gsmFeeStrategyLastUpdated__updated_only_by_updateGsmBuySellFees
Update of gsmFeeStrategyLastUpdated is only occurring by calling to updateGsmBuySellFees().

✅ 36. updateGsmBuySellFees_update_correctly__gsmFeeStrategyLastUpdated
A call to updateGsmBuySellFees() updates gsmFeeStrategyLastUpdated to the timestamp at time of the call.

✅ 37. updateGsmBuySellFees_timelock
A call to updateGsmBuySellFees() succeed (non-reverting) only if the timestamp at time of call surpassed gsmFeeStrategyLastUpdated by at least MIN_DELAY.

✅ 38. only_RISK_COUNCIL_can_call__updateGsmExposureCap
updateGsmExposureCap() will be successful (non-reverting) only if the caller was the risk council.

✅ 39. only_RISK_COUNCIL_can_call__updateGsmBuySellFees
updateGsmBuySellFees() will be successful (non-reverting) only if the caller was the risk council.

✅ 40. updateGsmExposureCap__correctness
1. updateGsmExposureCap() updates the exposurCap on storage to the passed value.

2. The new value of `exposureCap` is no more than double the current value (as per current specification).

✅ 41. `updateGsmBuySellFees__correctness`

1. The new values of each of the following `buyFee`, `sellFee` is updated to no more than `GSM_FEE_RATE_CHANGE_MAX` more than their current value after a successful call to `updateGsmBuySellFees()` (as per current specification).

Disclaimer

The Certora Prover takes a contract and a specification as input and formally proves that the contract satisfies the specification in all scenarios. Notably, the guarantees of the Certora Prover are scoped to the provided specification and the Certora Prover does not check any cases not covered by the specification.

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.