OpenZeppelin | security

AAVE GHO Audit

November 10th, 2022

This security assessment was prepared by OpenZeppelin.

Table of Contents

Summary

The Aave team asked us to review some changes to the code of their GHO stablecoin in order to prepare for launch on Mainnet V3 markets. This includes the GHO aToken, GHO variable debt token, GHO oracle, discount rate strategy, and the interest rate mechanism.

The main modifications between this commit and the commit audited in the previous audit round are:

- The code was adapted to use Aave V3 markets instead of Aave V2 markets
- The oracle price feed for the GHO token is now fixed and denominated in USD (using 8 decimals) instead of via Chainlink oracles and denominated in ETH (using 18 decimals).
- The Discount Lock Period feature was introduced to reward early adopters with a higher discount for a given period of time.

Type	Stablecoins
Timeline	From 2022-10-18 To 2022-10-24
Languages	Solidity

Total Issues	14 (2 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (0 resolved)
Low Severity Issues	5 (1 resolved, 1 partially resolved)
Notes & Additional Information	8 (1 resolved)

Scope

We audited the `aave/gho` repository at the `6f0888af6824e7eb7cb2cb5da8c6afc70322ffaa` commit.

In scope were the following contracts:

```
contracts
├── faciliatators
    ├── aave
        ├── interestStrategy
        │   ├── GhoDiscountRateStrategy.sol
        │   └── GhoInterestRateStrategy.sol
        ├── tokens
        │   ├── base
        │   │   └── ScaledBalanceTokenBase.sol
        │   ├── interfaces
        │   │   ├── IGhoVariableDebtToken.sol
        │   │   └── IGhoAToken.sol
        │   ├── GhoVariableDebtToken.sol
        │   └── GhoAToken.sol
        └── oracle
            └── GhoOracle.sol
```

System overview, privileged roles and trust assumptions

High-level overview of the system

GHO is a decentralized ERC20-compliant stablecoin that will be available on Ethereum mainnet. The token will be minted and burned under certain strategies, defined by different entities called Facilitators.

Facilitators are chosen by Aave governance and can be very different in nature as they all provide different stabilization mechanisms to keep GHO pegged.

Each Facilitator can mint up to a certain amount of GHO, defined by the maximum capacity of their buckets (a value set for each Facilitator by the Aave governance). In particular, the sum of the bucket levels for each Facilitator at any moment in time is the total supply of the token and the sum of all the buckets' maximum capacities is the maximum total supply.

Aave governance has proposed the first Facilitator be the Aave protocol, with specific GHO aToken and GHO Variable Debt Tokens to be deployed. The main difference between a regular Aave market and the GHO market, is that GHO tokens will not be supplied by users, but by the Facilitators.

Borrowing, repaying and liquidating GHO

Borrowing

As in any other Aave market, users are able to borrow GHO at a specific borrow rate. However, since there is no supply side, the borrow interest rate will be static and modified as needed by the governance. In regular Aave markets, the borrow amount limit is the amount supplied by other users, but for the GHO Aave market, the borrow amount limit is given by its bucket's max capacity. The protocol allows users to always borrow GHO tokens at 1 USD/GHO (price hardcoded in the GHO specific Oracle) regardless of the price of the token in parallel markets — as long as the bucket's max capacity has not been reached. This mechanism allows users to arbitrage: when market prices are above 1 USD, users can borrow GHO in the Aave market at the price of 1 USD and sell it to make a profit in parallel markets.

Users wishing to borrow GHO are incentivized to stake AAVE tokens in order to benefit from a discounted interest rate charged on borrowing GHO. Specifically, the discount rate in the audited commit is 20% on 100 GHO borrowed per stkAAVE held (e.g. a user holds 5 stkAAVE then they get 20% discount on the interest rate on up to 500 GHO.) Additionally, user's discount rate is updated anytime they send **or** receive the discount token (namely stkAAVE for now). Finally, there is a discount lock period where users are entitled to a discount for a given amount of time without needing to perform any additional actions.

Repaying and liquidating

Borrowers can also repay their debts or get liquidated as they would in typical Aave markets. For the former, in order to close their position and get their collateral back, borrowers have to return the GHO they borrowed plus any accrued interest. As with borrowing GHO, the protocol allows users that have borrowed to always repay GHO at 1 USD/GHO, which means that if borrowers can buy GHO in parallel markets at a discount, they can then repay in the Aave market at 1 USD/GHO and arbitrage a profit, helping maintain the peg.

Note: Unlike other over-collateralized stablecoins, the protocol does not provide a mechanism to influence demand of GHO through changes in monetary policy from the demand perspective. At the time of this audit, there is no mechanism to earn yield with GHO, so the peg relies entirely on borrowers until new Facilitators are introduced.

Finally, as in any other Aave market, if the position becomes too risky because the collateral value has dropped below specified thresholds, the GHO borrower can get liquidated.

Privileged roles and trust assumptions

Users of GHO place utmost trust in the Aave DAO as well as GHO Facilitators.

The Aave DAO, through the `pool admin` role in the `GhoVariableDebtToken` and `GhoAToken` contracts, can:

- Update the interest rate for GHO borrows to maintain a good ratio between supply and demand.
- Update the discount rate for discount token (`stkAAVE`) holders who also make up a portion of the DAO votes.
- Update the discount token.
- Update the discount lock period.

Additionally, the Aave DAO, through the `owner` role in the `GhoToken` contract, can:

- Set new Facilitator addresses.
- Set the max capacity of each Facilitator's bucket.

GHO Facilitators are capable of minting and burning GHO without additional permissions, therefore, they should be vetted by the Aave DAO **AND** their implementation should be audited to ensure it maintains a peg of 1 USD/GHO. The Aave DAO should carefully analyze how Facilitators might impact one another when setting and updating bucket capacities in order to best maintain the peg.

Findings

Here we present our findings.

Medium Severity

M-01 Discount rebalancing lock period can be bypassed

The `rebalanceUserDiscountPercent` function cannot be called until a certain period of time has passed, in order to allow a user to enjoy the applicable discount rate for a minimum amount of time.

However, anyone can bypass this timelock and force a user's discount rate to be updated by transferring any amount of `stkAAVE` into the their address, since that will trigger an automatic call to the `updateDiscountDistribution` function, which will execute the same actions as `rebalanceUserDiscountPercent` on both the `sender` and the `recipient`.

As an example, a user with a large active GHO principal debt who is entitled to a full discount rate for a given lock period, could effectively have it revoked in a scenario where governance decides to lower the discount rate by issuing a new discount rate strategy and anyone transferring `stkAAVE` to that user's address. Anyone with a vested interest over the GHO treasury would be incentivized to do so, since it would result in more interest paid to the treasury. The user themselves could inadvertently hurt their discount rate by simply staking additional AAVE or purchasing more `stkAAVE`.

In order to allow users entitled to a discount rate for a locked period to enjoy the full period, consider adding additional protections in the situations mentioned above to prevent their discount rate from being rebalanced.

Update: *Acknowledged, not resolved. The Aave team stated:*

> *At this time, there is not a practical solution to avoid this.*

Low Severity

L-01 Discounted borrow rate implementation does not match the technical paper

The Aave GHO technical paper defines the final borrow rate for a given user *u* that qualifies for a full discount as $R_{gho_u} = R_{gho} - R_{gho} * R_d$, where R_d is the applied discount rate and R_{gho} is the borrow rate set by governance.

However, the audited implementation applies the R_d discount rate to accrued interest rather than on the borrow rate itself. This creates a mismatch between the expected discounted borrow rate and the actual rate applied.

Assuming $R_d = 20\%$ and $R_{gho} = 10\%$, then the effective full discounted borrow rate would be $R_{gho_u} = 10\% * (1 - 20\%) = 8\%$. If a given user holding enough `stkAAVE` to enjoy the full 20% discount was to borrow 100 million GHO and hold that debt for a year, the total accrued interest at the end of the term (using the continuous compounding formula $A = P * e^{r*t}$ where `P` is the borrowed principal, `A` is the final outstanding debt, `r` is the interest rate, `t` is the time elapsed in years and `e ~ 2.71828` is Euler's number) would be as follows:

- According to the technical paper formula, after applying an effective $R_{gho_u} = 8\%$ interest rate over a year, the final outstanding debt would be 108,328,706.77 GHO.
- According to the audited implementation formula, after a year at a 10% borrow rate, the user would owe 10,517,091.81 GHO in accrued interest of which 20% would be forgiven, resulting in an outstanding debt of 108,413,673.45 GHO.

The implementation results in 84,966.68 GHO more outstanding debt than the technical paper formula. This effect is magnified as both the interest rate and/or holding period become larger.

For clarity, consider changing the technical paper or the code so that the implemented interest is aligned with and clear from the technical paper.

L-02 Missing docstrings

Within GhoVariableDebtToken.sol there are several functions that do not have docstrings. For instance:

- `allowance` on line 189
- `approve` on line 193
- `transferFrom` on line 201
- `increaseAllowance` on line 205
- `decreaseAllowance` on line 209

To improve documentation generation and readability, consider adding docstrings for each function individually.

Update: *Acknowledged, not resolved. The Aave team stated:*

> *As these functions are not implemented, and not intended to be used, no doc strings are required. This aligns with the current versions in aave-v3-core.*

L-03 Incorrect docstring comment format

Within GhoVariableDebtToken.sol there are several functions that try to inherit docstrings but are not in the correct comment format. For instance:

- `updateDiscountDistribution` on line 258
- `getDiscountPercent` on line 317
- `getBalanceFromInterest` on line 322
- `decreaseBalanceFromInterest` on line 327
- `rebalanceUserDiscountPercent` on line 333
- `updateDiscountLockPeriod` on line 365
- `getDiscountLockPeriod` on line 372
- `getUserRebalanceTimestamp` on line 377

To ensure documentation is generated correctly, consider updating the comment format to adhere to the Solidity Style Guide on documentation style.

Update: *Partially resolved in commit 14ed7ab. The last instance on line 377 was not fixed.*

L-04 Use of outdated or unpinned Solidity versions

Throughout the codebase there are Solidity pragma statements that use an outdated or unpinned version of Solidity. For instance:

- The `pragma` statement on line 2 of `GhoDiscountRateStrategy.sol`.
- The `pragma` statement on line 2 of `GhoOracle.sol`.
- The `pragma` statement on line 2 of `GhoAToken.sol`.
- The `pragma` statement on line 2 of `GhoVariableDebtToken.sol`.
- The `pragma` statement on line 2 of `ScaledBalanceTokenBase.sol`.
- The `pragma` statement on line 2 of `IGhoAToken.sol`.
- The `pragma` statement on line 2 of `IGhoVariableDebtToken.sol`.

Consider taking advantage of the latest Solidity version to improve the overall readability and security of the codebase. Regardless of which version of Solidity is used, consider pinning the version consistently throughout the codebase to prevent bugs due to incompatible future releases.

Update: *Acknowledged, not resolved. The Aave team expressed they would prefer to use* `^0.8.0` *for flexibility in GHO related contracts.*

L-05 Require statement with multiple conditions

Within GhoVariableDebtToken.sol there is a require statement on line 335 that requires multiple conditions to be satisfied.

To simplify the codebase and to raise the most helpful error messages for failing require statements, consider having a single require statement per condition.

Update: *Resolved in commit a86b493.*

Notes & Additional Information

N-01 Incorrect technical paper formula

The technical paper formula that describes the effective GHO interest rate calculation for a certain user *u*, taking discounts into consideration, is:

$$R_{GHO_u} = \begin{cases} R_{GHO} & \text{if } B_{stkAAVE}(u) = 0 \\ R_{GHO} - R_{GHO} * R_d & \text{if } B_{stkAAVE} > 0 \wedge P(u) \leq B_{stkAAVE} * T_d \\ \frac{R_{GHO} * P(u)_d + (R_{GHO} - R_{GHO} * R_d) * P(u)_n d}{P(u)} & \text{if } B_{stkAAVE} > 0 \wedge P(u) > B_{stkAAVE} * T_d \end{cases} \quad (1)$$

Where:

- *u* is a specific user.
- `BstkAAVE(u)` is the current `stkAAVE` balance owned by *u*.
- `P(u)` is the principal borrowed by *u*, in GHO tokens.

However, the audited discount rate strategy states that both the `stkAAVE` balance and the borrowed principal must be at least `1e18` in order to qualify for a discount.

Consider updating the technical paper formula or the code accordingly.

N-02 Stray GHO in GhoAToken contract cannot be rescued

The `rescueTokens` function allows the pool admin to rescue any mistakenly sent tokens from the `GhoAToken` contract. For regular Aave markets, adding a validation to prevent rescuing the underlying token address is necessary to avoid sweeping all users' deposits from the contract, but for the GHO token, it is not necessary since the supply side of the pool does not exist.

Consider allowing the pool admin to rescue GHO tokens that were mistakenly sent to the `GhoAToken` contract by removing this require statement.

Update: *Acknowledged, not resolved. The Aave team stated:*

> *A future change will required the AToken contract to hold GHO tokens, making this not feasible.*

N-03 Unnecessary virtual label on functions

In the contract GhoAToken.sol, there are functions that are labeled as virtual but the contract is not meant to be inherited:

- On line 47
- On line 102
- On line 112
- On line 134
- On line 142
- On line 163
- On line 169

For clarity and code cleanliness, consider removing the virtual label from functions that are not intended to be overridden.

Update: *Acknowledged, not resolved. The Aave team stated:*

> *Leave for consistency with v3 ATokens and VariableDebtTokens.*

N-04 Use of hardcoded error messages

Since Solidity version 0.8.4, custom errors provide a more consistent, cleaner, and more cost-efficient way to explain to users why an operation failed.

Throughout the codebase, instances of revert and/or require messages were found:

- In the GhoAToken.sol contract:
- the revert statement with the message `OPERATION_NOT_PERMITTED`.
- the revert statement with the message `OPERATION_NOT_PERMITTED`.
- the revert statement with the message `OPERATION_NOT_PERMITTED`.
- the revert statement with the message `OPERATION_NOT_PERMITTED`.
- the revert statement with the message `OPERATION_NOT_PERMITTED`.
- the revert statement with the message `OPERATION_NOT_PERMITTED`.
- the revert statement with the message `OPERATION_NOT_PERMITTED`.

- the require statement with the message `VARIABLE_DEBT_TOKEN_ALREADY_SET`.
- the require statement with the message `Errors.POOL_ADDRESSES_DO_NOT_MATCH`.
- the require statement with the message `Errors.UNDERLYING_CANNOT_BE_RESCUED`.
- In the GhoVariableDebtToken.sol contract:
- the require statement with the message `CALLER_NOT_DISCOUNT_TOKEN`.
- the require statement with the message `CALLER_NOT_A_TOKEN`.
- the require statement with the message `ATOKEN_ALREADY_SET`.
- the require statement with the message `DISCOUNT_PERCENT_REBALANCE_CONDITION_NOT_MET`.
- the require statement with the message `Errors.POOL_ADDRESSES_DO_NOT_MATCH`.
- the revert statement with the message `Errors.OPERATION_NOT_SUPPORTED`.
- the revert statement with the message `Errors.OPERATION_NOT_SUPPORTED`.
- the revert statement with the message `Errors.OPERATION_NOT_SUPPORTED`.
- the revert statement with the message `Errors.OPERATION_NOT_SUPPORTED`.
- the revert statement with the message `Errors.OPERATION_NOT_SUPPORTED`.
- the revert statement with the message `Errors.OPERATION_NOT_SUPPORTED`.
- the require statement with the message `Errors.INVALID_MINT_AMOUNT`.
- the require statement with the message `Errors.INVALID_BURN_AMOUNT`.
- In the ScaledBalanceTokenBase.sol contract:
- the require statement with the message `Errors.INVALID_MINT_AMOUNT`.
- the require statement with the message `Errors.INVALID_BURN_AMOUNT`.

For consistency, consider replacing hard coded require and revert messages with the use of the `Errors` library. Additionally, for conciseness and gas savings, consider replacing the `Errors` library with custom errors.

Update: *Acknowledged, not resolved. The Aave team stated:*

> *The Errors library belongs to the Aave protocol repo. In some instances, where errors align with the protocol it makes sense to take in advantage of this library. It likely does not make sense to include GHO specific errors in the general Aave Errors library. Perhaps we can have a GhoErrors library in the future.*
>
> *We are deciding not to use custom errors.*

N-05 Function mutability can be stricter

To provide more clarity on the price oracle answer being fixed to 1 USD with 8 decimals, consider changing the mutability of the `latestAnswer` function in `GhoOracle.sol` from `view` to `pure`.

Update: *Resolved in commit 3387624.*

N-06 Unused function parameters

Throughout the codebase there are functions that have unused parameters. For instance:

- The `calculateInterestRates` function's `params` parameter.
- The `mint` function's `caller`, `onBehalfOf`, `amount`, and `index` parameters.
- The `burn` function's `from`, `receiverOfUnderlying`, `amount`, and `index` parameters.
- The `mintToTreasury` function's `amount` and `index` parameters.
- The `transferOnLiquidation` function's `from`, `to`, and `value` parameters.
- The `permit` function's `owner`, `spender`, `value`, `deadline`, `v`, `r`, and `s` parameters.
- The `_transfer` function's `from`, `to`, `amount`, and `validate` parameters.
- The `_transfer` function's `from`, `to`, and `amount` parameters.

To provide clarity that these parameters are not used, consider removing the names of the parameters and keeping the types to retain the function signatures.

Update: *Acknowledged, not resolved. The Aave team stated:*

> *Leave for consistency with v3 AToken and Variable Debt Tokens.*

N-07 Inconsistent event emission parameters

Different from other events in the system where both the old and new values set are emitted as parameters, the `VariableDebtTokenSet` event in the `GhoAToken` contract and the `ATokenSet` event in the `GhoVariableDebtToken` contract only emit the new value.

To favor consistency, consider emitting both the old and new values in these and any other events.

Update: *Acknowledged, not resolved. The Aave team stated*:

> *There is a subtle difference between the set and update functions. Set only occurs once and cannot be updated. For that reason there is no old and new in the event.*
>
> *Generally this type of variable would be set in the initializer. In this case, while it is possible to do this, it is complicated and introduces some risk because we need to set the `_variableDebtToken` address, which is a proxy that will be created in the same transaction that creates and initializes the aToken contract proxy.*
>
> *For this reason, we believe it is less risky and error prone to just set this variable after both proxies have been created and before the reserve is active.*

N-08 Gas optimizations

When a given user performs their first borrow, the execution flow does not differ from subsequent borrows. Around 1% gas savings can be achieved if the first borrow is handled separately, since there is no need to accrue interest and most user state values are already set to zero.

To reduce the initial GHO borrowing gas consumption for a user, consider updating the relevant code to be more performant.

Update: *Acknowledged, not resolved. The Aave team expressed they will not make changes to resolve this at this time.*

Conclusions

One (1) Medium severity issue was found. Some changes were proposed to follow best practices and reduce the potential attack surface.

Appendix

Monitoring Recommendations

GHO is a decentralized stable token that, in the near future, will allow other Facilitators to mint GHO up to a certain bucket capacity. Since these parties might be completely autonomous and are able to follow different rules for minting/burning GHO, and since the Aave DAO cannot ensure these parties follow Aave's security standards, we recommend the following sensitive actions be monitored:

- Monitor when the bucket capacity of a certain Facilitator (especially the Aave Facilitator) is reached, or when it is close to being reached (e.g. 80% or 90% of the capacity).
- Monitor the `mint` and `burn` functions from the `GhoToken` contract, or the `transferUnderlyingTo` and `handleRepayment` functions from the `GhoAToken` contract on big mints/burns.
- Monitor the correlation between the amount of GHO minted/burned and the underlying asset deposited/withdrawn to back it, and trigger alerts in case the correlation is not within certain defined boundaries.
- Monitor all functions that are only callable by the governance or by the pool admin to ensure that all actions are authorized by the Aave DAO and that the values they set or outcomes are in line with their expectations. The following functions are a few examples:
  - setAToken
  - updateDiscountRateStrategy
  - updateDiscountToken
  - updateDiscountLockPeriod
  - rescueTokens
  - setVariableDebtToken
  - updateGhoTreasury
  - addFacilitators
  - removeFacilitators
  - setFacilitatorBucketCapacity