certora

Security Assessment

aave

Aave v3.4 update

June-2025

Prepared for:
Aave DAO

Code developed by:
BORED
GHOSTS
DEVELOPING

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Aave v3.4	Github Repository	468e5dc	EVM

Project Overview

This document describes the verification of **Aave v3.4** code using manual code review. The work was undertaken from **February 27** to **June 11, 2025**.

The following contracts are considered in scope for this review:

- `src/contracts/*`

The team performed a manual audit of all the solidity contracts. During the audit, Certora didn't find any significant issues in the code.

Protocol Overview

The **Aave v3.4 update** is a new version of the existing protocol aiming to introduce new features like multicall support and position manager role, reduce gas cost, optimize operations, remove deprecated functionalities, improve rounding precision and simplify flash loan fees model. Additionally, this update intends to standardize `aGHO` and `vGHO` which had a unique implementation compared to other `aTokens` and `vTokens`. However, these specific changes will only be visible at the configuration level during the corresponding governance proposal.

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	1	1	1
Total	1	1	1

Severity Matrix

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

Detailed Findings

Audit Goals

1. Position Manager

A. Users have the ability to assign an address as a PositionManager. The manager should only be able to manage the user's collateral reserves and eMode.
B. Users should have the ability to revoke PositionManager privileges at any time.

2. Removal of unbacked

A. The portal feature has been removed and all references to it should be completely deleted and not be used anymore.

3. Multicall

A. The pool has been redesigned to support executing multiple operations in a single transaction through the introduction of multicall. This feature should not allow unrequested actions to be performed on behalf of another user.

4. Immutable variables

A. Multiple storage variables in different contracts have been removed and replaced by an immutable reference. All the interactions previously made with these storage variables must be re-tailored to use the immutable variables to maintain compatibility and expected logic.
B. The removal of storage variables must preserve storage layout to avoid unexpected storage changes.

5. Self-liquidation removal

A. A user should not be able to liquidate a position he owns by himself, using the same address.

6. UserState struct update

A. The `IncentivizedERC20::UserState` struct, inherited by `aToken` now supports a delegation feature. To do so, it reduces the existing `balance` field size from `uint128` to `uint120` in order to introduce a new `delegationMode` field that is 8 bits long. Since this structure is already populated in storage, this change should not collide with existing data and not pre-write the new field.

7. Flash loan fees model

A. Previously, users taking flash loans had to pay a premium that would be split between the liquidity providers and the protocol. The update changes this behavior to send the entire premium to the treasury. This should be done correctly while preserving pre-existing logic.

8. aToken with delegation

A. The existing aAAVE delegation code was migrated to the origin repo to unify codebases. This unification is expected to create no changes in logic.

9. Code style improvement and optimization

A. Multiple parts of the protocol have undergone significant improvements in terms of logic and storage access. These changes should preserve the integrity of the existing logic and not deviate from their intended functionalities.

10. Simplification of calculateCompoundedInterest() formula

A. The calculateCompoundedInterest() function has been refactored to improve accuracy and reduce gas costs. Both the first and second terms now apply multiplication by exp before dividing by SECONDS_PER_YEAR. This change is expected to preserve the original logic and should not introduce any precision loss or overflow issues, even in edge cases.

Coverage and Conclusions

1. Position Manager

A. The PositionManager can only manage a user's collateral reserves and eMode.
B. There is no way for any party to prevent a user from revoking a PositionManager privilege.

2. Removal of unbacked

A. There are no leftovers from the portal/unbacked feature except for the `CalculateInterestRatesParams::unbacked` structure field which is used to account for the reserve deficit during interest rate update.

3. Multicall

A. The pool inherits from Openzeppelin `Multicall` contract to chain operations in a single transaction. Since `_msgSender()` has not been overridden (and always resolves to `msg.sender`), impersonating another user is not possible.

4. Immutable variables

A. The interactions correctly rely on the newly introduced immutable variables across all relevant contracts.
The `AaveProtocolDataProvider` contract has introduced the `POOL` immutable variable to avoid reading the pool address from the `PoolAddressProvider` contract.
All `Pool` operations pass the `RESERVE_INTEREST_RATE_STRATEGY` immutable variable as parameter rather than being read in the reserves during logic execution.
`aToken` and `vToken` are now constructed with the `REWARDS_CONTROLLER` replacing the old `_incentivesController` storage variable. Additionally, `aToken` is now constructed with the `TREASURY` immutable variable, replacing the old `_treasury` storage variable.

B. Old storage variables replaced by an immutable variable have been renamed with the prefix `_deprecated` to preserve existing storage layout while preventing the slot from being accidentally written.

5. Self-liquidation removal

A. The `ValidationLogic::validateLiquidationCall()` function has a new check that enforces the liquidator and the borrower to be different addresses.

6. UserState struct update

A. In order for the `balance` and `delegationMode` fields to collide, one must have a `balance` with one of its 8 most significant bits set to 1. This means the balance should be an unrealistic number of tokens of magnitude `10^36`, bigger than `type(uint120).max.`
`{`

```
        2^120 = 1.329227995784916e+36
        2^128 = 3.402823669209385e+38
}
```

In case this occurs and `delegationMode` ends up with a value different than the ones allowed by the corresponding Enum, Solidity will fail to cast it and revert, leading to a Denial of Service. Moreover, the actual `balance` evaluated by Solidity may be reduced to be way less than the `balance` before the update.
Since the requirements for it to happen are unrealistic (given the fact that, from the Aave Dashboard data, the biggest aToken `totalSupply` is **USDe** on Ethereum and is currently at `400e24`), the structure update is safely implemented.

7. Flash loan fees model

A. In `FlashLoanLogic::_handleFlashLoanRepayment()`, the calculation responsible for splitting the premium between the treasury and liquidity providers has been removed. Now, the entire premium is directly transferred to the treasury. Additionally, the `Pool` has been updated to reflect this behavior, notably by returning `100_00 BPS` (100%) in the `FLASHLOAN_PREMIUM_TO_PROTOCOL` function, which determines the percentage of the premium owed to the protocol.

8. aToken with delegation

A. We verified that the introduced code matches the aAAVE implementation and generates no diff in logic. The delegation feature will still only be activated on aAAVE.

9. Code style improvement and optimization

A. Contracts have been displayed side to side to compare the changes from the live version to the v3.4 version. Changes related to style improvements and optimization preserve the core logic of each functionality.
For instance, `GenericLogic::calculateUserAccountData()` now uses `UserConfiguration::getNextFlags()` to get all the reserves a user is involved with rather than looping through every single reserve, regardless of whether it is relevant for the user. This significantly reduces the amount of iterations when getting a user's account data by skipping unused reserves while still maintaining the core functionality of the function.
Another example of an improvement is the compound interest calculation—the calculation now uses a tailor approximation instead of a binomial approximation. This gives a more accurate calculation of continuous interest and simplifies the calculation (saving gas). The

use of unchecked in the new calculation has been checked to ensure that an overflow is not possible.

10. Simplification of calculateCompoundedInterest() formula

A. Thorough checks confirm that the refactor maintains the correctness of the original formula and does not introduce rounding inconsistencies or risk of overflow, even in edge-case scenarios. This was ensured by verifying that both the original and refactored expressions simplify to the same formula, and by confirming that no overflows occur even when inputs are pushed to their extreme bounds. The change also contributes to overall gas savings by optimizing arithmetic execution.

Informational Issues

I-01. Inconsistent use of balanceOf and scaledBalanceOf

Description: Most uses of `balanceOf` have been switched to uses of `scaledBalanceOf`, but in a few places, `balanceOf` has been kept:

- In `BorrowLogic.sol`, in `executeRepay`, line 149.
- In `LiquidationLogic.sol`, in `executeLiquidationCall`, line 196.
- In `LiquidationLogic.sol`, in `_burnBadDebt`, line 664.

Customer's response: Acknowledged. This optimization will be added to these places in a future version.
Fix Review: Not required.

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.