Security Assessment Report

AAVE-V3.1

03/24

Prepared for
Aave

Table of content

Project Summary

Project Scope

Repo Name	Repository	Commits	Compiler version	Platform
aave-v3-origin	Github Repository	cf6e2ac	0.8.19	multichain

Project Overview

This document describes the specification and verification of the changes that were introduced for **Aave-v3.1** using the Certora Prover and manual code review findings. The work was undertaken from **18 March 2024** to **24 April 2024**.

The following contract list is included in our scope:

```
- All contracts of Aave V3.1
```

The Certora team performed a manual audit for all the changes that were introduced for Aave V3.1.

Protocol Overview

The Aave-v3.1 is a new version of Aave v3, focused on different security and operational improvements. Aave-v3.1 introduces 18 changes based on the following Pull-Requests:

1. Virtual Accounting: Pull Request #81.
2. Stateful (params-wise) default interest rate strategy**:** Pull Request #79.
3. Freezing by EMERGENCY_GUARDIAN on PoolConfigurator: Pull Request #33.
4. Reserve data update on rate strategy and RF changes: Pull Request #59, Pull Request #89, Pull Request #108.
5. Minimum decimals for listed assets: Pull Request #82.
6. Liquidations grace sentinel: Pull Request #85.
7. LTV0 on freezing: Pull Request #55.
8. Permissionless movement of stable positions to variable: Pull Request #57.
9. try..catch on permit() used on Pool functions: Pull Request #47.

10. Different bug fixes added to production: Pull Request #96.
11. Properly manage flags when repaying with aToken: Pull Request #93.
12. Properly validating 0 amount on aToken transfer: Pull Request #94.
13. New getters on Pool and PoolConfigurator for library addresses: Pull Request #97.
14. Allow setting debt ceiling whenever LT is 0: Pull Request #106.
15. setPoolPause() and setReservePause() overloads: Github commit.
16. Changes on liquidation grace period mechanism: Pull Request #111.
17. Constructors cleanup: Pull Request #112.
18. Do Not allow flashing/borrowing more than aToken.totalSupply(): Pull Request #113.

The above changes will be applied by upgrading the implementation under proxy via on-chain governance proposals. In addition to the implementation upgrade, for some of the features (virtual accounting, stateful-params rate strategy), data setup for each asset will be required. The following payload (or logic on initialize()) is also a part of this review: the upgrade payload.

Audit Goals

Verify that the above-listed changes were implemented correctly and have not introduced a security bug.

Coverage

Below, we summarize our work regarding each change that was presented above.

1. Virtual accounting. See a detailed description in the next section.
2. We reviewed the DefaultReserveInterestRateStrategyV2 contract and checked that the added function _setInterestRateParams() makes sense. We also noted that in the updated version, the calculateInterestRates() function always returns currentStableBorrowRate as 0. This is fine since users cannot take any new stable debt position.
3. We verified that this change introduces a modifier which verifies that the msg.sender is either a risk admin, pool admin or emergency admin, which matches the description.
4. We verified that whenever an interest rate strategy is updated, the old rate is correctly applied for the time frame between the last update and the rate change. If the address of the strategy does not change, then only the interest rate parameters are updated.
5. We verified that this change adds a requirement that assets would have a minimum of 6 decimal digits.
6. We verified that this change allows an emergency admin or a pool admin to set a Liquidation Grace Period.

7. We verified that this change sets the LTV of a frozen asset to 0 whenever setReserveFreeze() is called. As mentioned in the description, the original value of the LTV (or the updated value if the asset was updated while it was frozen) is set back when the asset is unfrozen.
8. We verified that this change allows anyone to switch any open **stable** debt position to a **variable** debt position. We couldn't find any way for a user who already has a **stable** debt position to prevent other users from switching their position to a **variable** debt position (for example, by setting up something that would cause such a transaction to revert).
9. We approve that the permit() function is called inside a try-catch statement.
10. The flag update is done correctly, turning off the payer's collateral flag when aToken balance is zero.
11. We approve that usingAsCollateral is set to False whenever a repayment causes the balance of the aToken to be 0, which matches the description.
12. We verified that the scaled-down amount is compared to 0 instead of the scaled-up amount, which matches the description.
13. This change introduces a bunch of getters functions, as described.
14. We verified that this change allows the DAO to set a debt ceiling even if the supply of that asset is non-zero as long as the LT is 0, which matches the description of that change.
15. We verified that this change introduced setReservePause(address asset, bool paused) and setPoolPause(bool paused), which call setReservePause(asset, paused, 0) and setPoolPause(paused, 0), respectively, meaning with 0 as the grace period, as intended.
16. We verified that this change made the setting of the liquidation grace period to be a part of the unpausing mechanism (through the function setReservePause), and deleted the function setLiquidationGracePeriod, which matches the description of the change.
17. The constructor was cleaned in a safe manner.
18. We went over the proposed change. The aim of that change was to prevent a scenario in which the protocol receives a large amount of tokens (for example, from the flash-loan fees) compared to what is already in the protocol to begin with, which will cause the index to inflate, potentially causing a loss of precision in aToken calculations, that if large enough might cause users to earn/lose more than they should in a non-negligible way. We concluded that while the added line:
    require(IERC20(reserve.aTokenAddress).totalSupply() >= amount, Errors.INVALID_AMOUNT); achieves the intended purpose of preventing this potential issue from happening as a result of a single flash-loan, it does not solve the issue entirely, as it might still be possible in the case that many flash-loans are taken.
19. We verified the upgrade payload.

Change #1: virtual accounting

In this section, we elaborate on the verification we did for the virtual-accounting change. We first describe the coverage of our verification and then the conclusions we had.

Coverage

During the review of the virtual accounting change, we examined the following properties via manual review:

Virtual Accounting Updates

For every entry point in the pool (external functions), we applied the following steps to determine whether the updates of virtual accounting have been done correctly. We ensured that no necessary update was left out.

1. In the old version of the code (prior to the introduction of virtual accounting), for each entry point, we raised our expectation as to whether the internal accounting should be updated on interaction. If we expected the virtual accounting to be updated,
   - Should it grow or decrease?
   - In what magnitude should it change?

   Flashloan
   Special emphasis should be placed on flashloan behavior regarding the amount to be loaned, the amount to be repaid, and reentrancy attacks.

2. Verify the expectation against the implementation in each entry point and look for mismatches and unexpected behavior.

Virtual Accounting Reads

For every call to balanceOf(), we applied the following steps to determine whether the introduced changes were done correctly and whether other necessary modifications were omitted.

3. In the old version of the code (prior to the introduction of virtual accounting), for each call to balanceOf(), we raised our expectation as to whether the call should be modified to read from virtual accounting.

4. Verify the expectation against the implementation in each call and look for mismatches and unexpected behavior.

Conclusions

The virtual accounting's purpose is to represent in a more precise manner ***the available liquidity of an asset in the pool***, eliminating the ability to "donate" assets, which proved many times in the past to allow dangerous attacking vectors.

Virtual Accounting Updates

1. We expect **only** the following functions to update the virtual accounting upon being called:

    - backUnbacked(address asset, uint256 amount, uint256 fee) adds liquidity to the pool. Hence we expect an update of the virtual accounting - an increase of the backed asset by the amount + fee.

    - supply(address asset, uint256 amount, address onBehalfOf, uint16 referralCode), supplyWithPermit(...) adds liquidity to the pool on behalf of the supplier. Hence we expect an update of the virtual accounting - an increase of the asset by amount. [1]

    - withdraw(address asset, uint256 amount, address to) transfers out liquidity from the pool. Hence we expect an update of the virtual accounting - a decrease of the withdrawn asset by amount.

    - borrow(address asset, uint256 amount, uint256 interestRateMode, uint16 referralCode, address onBehalfOf) transfers out liquidity from the pool, hence we expect an update of the virtual accounting - a decrease of the withdrawn asset by amount.

    - repay(address asset, uint256 amount, uint256 interestRateMode, address onBehalfOf), repayWithPermit(...) adds liquidity to the pool by transferring the asset in debt from the user. Hence we expect an update of the virtual accounting – an increase of the repaid asset by the amount.

    - liquidationCall(address collateralAsset, address debtAsset, address user, uint256 debtToCover, bool receiveAToken) both add liquidity to the pool when repaying debt and transfer out liquidity from the pool to the liquidator in collateral when receiveAToken == false. Hence, we expect an update of the virtual accounting - an increase of the repaid asset by debtToCover and a decrease of the collateral asset by the corresponding value, including bonus and fees.

- repayWithATokens(), swapBorrowRateMode(...), setUserUseReserveAsCollateral(...), initReserve(...) and dropReserve(...) are expected to not update the virtual accounting.

Flashloan

1.1. Both flashloan functions should reduce the virtual accounting before calling the flashloan receiver to maintain a "clean state" that represents the true available liquidity in case of a callback/reentrancy.

1.2. The update should limit the de facto amount that can be loaned to a maximum of the available liquidity determined by virtual accounting.

1.3. In the case of immediate payment (as opposed to opening a loan position against the flashloan sum), the virtual accounting should be updated accordingly.

2. The implementation met all expectations.

Virtual Accounting Reads

3. Given our interpretation of what virtual accounting represents - the available liquidity of an asset in the pool - we conclude that out of 30 calls in the scope, only 2 occurrences require a modification to read from virtual accounting.
   Both:

   - vars.availableLiquidity = IERC20(params.asset).balanceOf(params.reserveCache.aTokenAddress); in validateBorrow(...),
   - vars.availableLiquidity = IERC20(params.reserve).balanceOf(params.aToken) + params.liquidityAdded - params.liquidityTaken; in calculateInterestRates.

   falls precisely to the interpretation we concluded within their context.
4. The implementation indeed updates these two occurrences only. Moreover, it was found to update the state of the virtual accounting correctly and pass the current value when calling calculateInterestRates(...).

Findings Summary

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

Severity	Discovered	Acknowledged	Code Fixed
Critical			
High			
Medium			
Low			
Informational	1	yes	1af1fc9
Total	1		

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Probability		

Detailed Findings

ID	Title	Severity
Info-1	Altered function visibility	Informational

Informational Severity Issues

INFO-1: Altered function visibility

Issue: The function setReservePause(...) is defined as public.

Description: This is relevant to change #15. The function setReservePause(address asset, bool paused) in PoolConfigurator.sol is defined with public visibility even though it appears that no other internal function calls this function.

Recommendation: Consider changing the visibility of the function to external.

Customer's response:

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.