Aave Security Review

Pashov Audit Group

Conducted by: defsec, Said, santipu, btk

September 5th - September 15th

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **bgd-labs/aave-v3-origin-pashov** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Aave V3.2

Aave Protocol is a decentralized system where users can supply liquidity to earn interest, borrow assets with more collateral than they borrow, or participate in liquidations. Aave v3 introduced eMode, which allows users to group related assets, such as ETH and WETH, for higher-risk setups, but each asset could only belong to one eMode, limiting flexibility. Aave v3.2 update introduces liquid eModes, letting assets be part of multiple eModes with more detailed settings, such as whether an asset can be borrowed or used as collateral within an eMode, and new configuration options offer more granular control over asset usage in different eModes.

5. Risk Classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - a4849111a0ce57e3af1ca5cd9a9b8c6a8cdad1e0

fixes review commit hash - dd0bbecb90a53628fe15c076217eac3a7275182f

Scope

The following smart contracts were in the scope of the audit:

- `AaveProtocolDataProvider`
- `IPoolConfigurator`
- `IPoolDataProvider`
- `EModeConfiguration`
- `ReserveConfiguration`
- `Errors`
- `ConfiguratorLogic`
- `EModeLogic`
- `GenericLogic`
- `LiquidationLogic`
- `ValidationLogic`
- `DataTypes`
- `PoolConfigurator`

Attack vectors covered

1. Suboptimal LTV and Liquidation Threshold

Description

An attacker tries to manipulate asset selection to take advantage of suboptimal rates in E-Mode, potentially gaining more favorable terms than intended. Could the new EMode unexpectedly result in a lower LTV or liquidation threshold than the default mode? Or, when users enable EMode, do they get suboptimal LTV and liquidation thresholds due to new assumptions?

Protection

The system always uses the most conservative (safest) parameters between E-Mode and regular mode for each asset, ensuring that users cannot exploit discrepancies to gain unfair advantages. EMode LT always being higher than non-eMode LT for all assets is not a constraint, so it is not an issue. For suboptimal LTV and liquidation thresholds, it is not stated on docs but it is intended behavior.

2. Persistence of E-Mode Parameters

Description

An attacker attempts to front-run asset removal from an E-Mode to lock in favorable LTV and liquidation threshold parameters.

Protection

E-Mode parameters are applied dynamically during health factor calculations. Even if an asset is removed from E-Mode, the system will use the regular mode parameters for that asset, preventing exploitation of outdated E-Mode settings.

3. Instant Liquidations on Collateral Removal

Description

An attacker tries to exploit the transition period when collateral is removed from E-Mode, potentially triggering unfair liquidations.

Protection

The system requires that the health factor remains above 1 when switching E-Modes or when E-Mode parameters change. This ensures that positions remain healthy during transitions, preventing instant liquidations.

4. Asymmetric Handling of Borrowing and Collateral

Description

An attacker attempts to leverage the difference in treatment between borrowed assets and collateral in E-Mode to create an exploitable position.

Protection

The system applies consistent rules for both borrowing and collateral within E-Mode. Assets must be explicitly enabled for both functions in E-Mode, and the most conservative parameters are always used, preventing the exploitation of asymmetries.

5. Storage Corruption Risk Due to New Variables

Description

There is a risk of storage corruption due to the changes in the EModeCategory struct, particularly with the introduction of new variables like `isCollateralBitmap` and `isBorrowableBitmap`.

Protection

The new variables are fully compatible with the previous storage layout, and no data corruption occurs. The system ensures this remains true, assuming Aave has not set a non-zero address in the `priceSource` field within `EModeCategory`.

6. Administrator-Induced Data Corruption

Description

An administrator could unknowingly corrupt the data structure when changing Loan-to-Value (LTV), Liquidation Threshold (LT), or Liquidation Bonus (LB) for an eMode.

Protection

Admins can safely adjust LTV, LT, or LB without causing any corruption to the `isCollateralBitmap` and `isBorrowableBitmap` data fields. The system architecture ensures the integrity of these fields during administrative changes.

7. E-Mode Category Disabling

Description

An attacker attempts to exploit an active but undesired E-Mode category that cannot be easily disabled, potentially manipulating the system to their advantage.

Protection

The system is designed to operate safely even with active E-Mode categories. Disabling an E-Mode category is a sensitive operation that requires careful consideration of existing positions, making it intentionally difficult to prevent unintended consequences.

8. Limited Granularity in Risk Management

Description

An attacker tries to exploit the broad application of E-Mode parameters across all assets within a category, potentially finding edge cases where risk is underestimated.

Protection

While E-Mode parameters apply broadly to a category, the system still considers individual asset parameters. It uses the more conservative option between E-Mode and individual asset parameters, ensuring that risk is not underestimated for any specific asset.

9. E-Mode Parameter Precedence

Description

An attacker attempts to manipulate the interaction between global asset parameters and E-Mode parameters to create a more favorable position than intended.

Protection

The system has clear precedence rules, always using the more conservative option between global and E-Mode parameters. This ensures that the stricter risk management approach is always applied, preventing the exploitation of parameter interactions.

10. The sequence of EMode Configuration Changes

Description

Find all possible sequences of EMode configuration changes, user actions, and user states that do not account for this current update. (e.g., already activated EMode, wanting to deactivate EMode, wanting to borrow, already borrowed, wanting to provide collateral, already provided collateral, wanting to liquidate)

Protection

Checks and safeguards are already in place to ensure all operations are correct. All operations depend on the same validation within `ValidationLogic`, so updating the safeguards in one place ensures all operations remain correct.

11. Unintended Consequences from EMode Changes during Borrowing

Description

Are there any unintended consequences if users change EMode while borrowing an asset (for instance, instantly liquidatable, not liquidatable, collateral not counted, etc.)

Protection

`ValidationLogic` and `calculateUserAccountData` within `GenericLogic` are already updated properly so there are no unintended consequences.

12. Impact of Borrowed Asset Removal from EMode

Description

Can users still benefit from EMode once a borrowed asset is removed from EMode, since `calculateUserAccountData` doesn't check if the borrowed asset is registered in EMode?

Protection

It is possible, but devs state it is by design "Disabling borrowing is a very non-intrusive action - in and outside eMode. The existing positions stay intact people can just no longer increase their exposure via increased borrows."

13. Bypassing EMode Restrictions with Flash Loans

Description

Can users bypass any EMode restrictions by leveraging flash loans, for instance, by borrowing assets that are not registered within their configured EMode?

Protection

When users use a flash loan and choose not to repay the borrowed assets, `BorrowLogic.executeBorrow` is already in place to ensure that the borrowed assets are valid, including EMode validation.

14. Manipulation of EMode Liquidation Bonus

Description

EMode has a separate liquidation threshold and liquidation bonus. Not using both EMode values can cause serious problems for the protocol. Can borrowers manipulate or change the EMode liquidation bonus?

Protection

`LiquidationLogic` has the exact same state check when deciding whether users are using the EMode liquidation bonus as `GenericLogic.calculateUserAccountData`. So, both (the liquidation bonus and the liquidation threshold) will always either use EMode or non-EMode values.

15. In-Scope Logic Impact from EMode Update

Description

Are there any existing operations that should consider this new behavior but currently do not? Checking all in-scope logic that doesn't have any diffs within the commit.

Protection

All of them (in-scope logic that doesn't have any diffs within the commit), if they do not interact with or depend on EMode configuration, are already using `ValidationLogic`. Therefore, they are safe and do not require any changes.

16. Impact of EMode on Other Features

Description

Does the new EMode unexpectedly impact other features (e.g., Siloed Borrowing, Isolation Mode, etc.)? For instance, does activating EMode accidentally disable other features or modes?

Protection

The safeguards and validations for each feature are independent within their respective libraries and `ValidationLogic`, so there is no impact.

17. Handling of eMode = 0 Cases

Description

Are all cases where eMode = 0 handled correctly? eMode = 0 is the default case, so ensure there are no functions or logic that accidentally access eMode = 0 and use the user's EMode LTV and liquidation threshold. Also, ensure it is not possible for an admin to configure eMode = 0.

Protection

Functions within `ValidationLogic` and `GenericLogic` already exclude eMode = 0 (by ensuring `params.userEModeCategory != 0`), and `configureEModeCategory` already ensures that configuration changes to eMode = 0 are restricted (by ensuring `id != 0`).

18. Simultaneous Admin Function Calls

Description

Two admins may attempt to call the functions `setCollateral` and `setBorrowable` simultaneously with the same arguments, potentially corrupting the final state.

Protection

Both `setCollateral` and `setBorrowable` are idempotent functions, meaning multiple calls with the same parameters will not change the final state. This prevents any unintended state changes from repeated transactions by different administrators.

19. Health Factor Decrease from EMode Switch

Description

A user might attempt to switch their eMode, unknowingly decreasing the HF, resulting in an instant liquidation.

Protection

The system checks the Health Factor (HF) after allowing any user to change their eMode. If the Health Factor drops below 1 as a result of the change, the action is blocked, ensuring users cannot do it.

20. eMode Mask Removal and Data Corruption Risk

Description

The removal of the eMode mask within the asset configuration bitmap could potentially lead to data corruption.

Protection

The eMode mask has been cleanly removed without impacting the rest of the data stored in the bitmap, ensuring no data corruption or unintended side effects.

21. Impact of EMode Update on Existing Users

Description

Users currently in a specific eMode may experience issues or exploit changes introduced by the update.

Protection

Users in eMode remain unaffected by the update. The mapping that tracks user eModes (`_usersEModeCategory`) remains unchanged. Additionally, as long as the values of LTs and LTVs hold, the `calculateUserAccountData` and `executeLiquidationCall` functions will behave as before the update.

22. Overflow Risk in `setCollateral()`

Description

Overflow in `setCollateral()` function when setting the `bit` variable.

Protection

This issue is mitigated as the reserveIndex is validated to ensure it is always less than 128.

23. Storage Slot Collision Risk in `DataTypes` Library

Description

Storage slot collision in DataTypes library.

Protection

Storage slots are properly managed to prevent collisions.

7. Executive Summary

Over the course of the security review, defsec, Said, santipu, btk engaged with BGD Labs to review Aave V3.2. In this period of time a total of **1** issues were uncovered.

Protocol Summary

Protocol Name	Aave V3.2
Repository	https://github.com/bgd-labs/aave-v3-origin-pashov
Date	September 5th - September 15th
Protocol Type	Lending Protocol

Findings Count

Severity	**Amount**
Low	1
Total Findings	**1**

Summary of Findings

ID	**Title**	**Severity**	**Status**
[L-01]	setUserEMode should check if categoryId is the same to avoid unnecessary validation	Low	Resolved

8. Findings

8.1. Low Findings

[L-01] `setUserEMode` should check if `categoryId` is the same to avoid unnecessary validation

When the user calls `setUserEmode`, the execution doesn't check if the provided `categoryId` is the same as the user's previous configuration. This will require users to perform unnecessary health factor validation and spend extra gas. Compared to when users call `setUserUseReserveAsCollateral` to configure a reserve as collateral, it will return early if the provided flag is the same as the existing flag.

```
function executeUseReserveAsCollateral(
    mapping(address => DataTypes.ReserveData) storage reservesData,
    mapping(uint256 => address) storage reservesList,
    mapping(uint8 => DataTypes.EModeCategory) storage eModeCategories,
    DataTypes.UserConfigurationMap storage userConfig,
    address asset,
    bool useAsCollateral,
    uint256 reservesCount,
    address priceOracle,
    uint8 userEModeCategory
  ) external {
    DataTypes.ReserveData storage reserve = reservesData[asset];
    DataTypes.ReserveCache memory reserveCache = reserve.cache();

    uint256 userBalance = IERC20(reserveCache.aTokenAddress).balanceOf
      (msg.sender);

    ValidationLogic.validateSetUseReserveAsCollateral
      (reserveCache, userBalance);

>>> if (useAsCollateral == userConfig.isUsingAsCollateral(reserve.id)) return;
    // ...
}
```

Consider using the same pattern inside `setUserEmode` / `executeSetUserEMode`. If the provided category is the same as the existing one, return early to avoid unnecessary validation.

```
function executeSetUserEMode(
    mapping(address => DataTypes.ReserveData) storage reservesData,
    mapping(uint256 => address) storage reservesList,
    mapping(uint8 => DataTypes.EModeCategory) storage eModeCategories,
    mapping(address => uint8) storage usersEModeCategory,
    DataTypes.UserConfigurationMap storage userConfig,
    DataTypes.ExecuteSetUserEModeParams memory params
  ) external {
    ValidationLogic.validateSetUserEMode(
      eModeCategories,
      userConfig,
      params.reservesCount,
      params.categoryId
    );

+   if (usersEModeCategory[msg.sender] == params.categoryId) return;
    usersEModeCategory[msg.sender] = params.categoryId;
    ValidationLogic.validateHealthFactor(
      reservesData,
      reservesList,
      eModeCategories,
      userConfig,
      msg.sender,
      params.categoryId,
      params.reservesCount,
      params.oracle
    );
    emit UserEModeSet(msg.sender, params.categoryId);
  }
```