Blackthorn

Security Review For **Aave v3.6**

Collaborative Audit Prepared For:	**Aave DAO**
Code Developed By:	**BGD Labs**
Lead Security Expert(s):	**mstpr-brainbot**
	pkqs90
	TessKimy
Date Audited:	**October 24 - October 29, 2025**

Introduction

Aave v3.6 is an upgrade to the Aave v3 protocol, currently running on v3.5 in production across all networks.

It includes the following changes and improvements:

- Decoupling of LT, LB, and LTV0 dynamics between Liquid eModes and non-eMode.
- No automatic enabling as collateral anymore on transfer of aTokens, only keeping it on supply.
- Addition of a renounce allowance feature for aTokens and vTokens (credit delegation).
- Misc minor improvements and optimizations.

An exhaustive explanation of all changes included can be found on the Aave v3.6 features document on the Aave v3 origin repository.

Scope

Repository: bgd-labs/aave-v3-origin-blackthorn

Audited Commit: e34aaaa829078df2317ef53173101eb4db6a5757

Final Commit: ce5d9ef3b8ad30eb7dbbd3e16a998a1b4de8e954

Files:

- src/contracts/protocol/libraries/logic/GenericLogic.sol
- src/contracts/protocol/libraries/logic/LiquidationLogic.sol
- src/contracts/protocol/libraries/logic/SupplyLogic.sol
- src/contracts/protocol/libraries/logic/ValidationLogic.sol
- src/contracts/protocol/libraries/types/DataTypes.sol
- src/contracts/protocol/pool/PoolConfigurator.sol
- src/contracts/protocol/pool/Pool.sol

- src/contracts/protocol/tokenization/AToken.sol
- src/contracts/protocol/tokenization/base/DebtTokenBase.sol
- src/contracts/protocol/tokenization/base/IncentivizedERC20.sol
- src/contracts/protocol/tokenization/VariableDebtToken.sol

Final Commit Hash

ce5d9ef3b8ad30eb7dbbd3e16a998a1b4de8e954

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
1	0	2

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Security Experts Dedicated to This Review

@TessKimy

@mstpr-brainbot

@pkqs90

Issue H-1: eMode only collaterals cannot be liquidated

Source: https://github.com/sherlock-audit/2025-10-aave-v3-6-oct-24th/issues/12

Vulnerability Detail

In Aave v3.6, collateral eligibility is decoupled between normal mode (`eMode = 0`) and opt-in eModes. As a result, it's valid for a collateral to have `ReserveData.configuration.liquidationThreshold == 0` in normal mode while being enabled in a specific eMode.

However, during liquidation, the validation logic still checks that the collateral's `configuration.getLiquidationThreshold()` is non-zero. This causes a problem: collaterals that are enabled only within an eMode but not in normal mode will fail the check and thus cannot be liquidated.

https://github.com/sherlock-audit/2025-10-aave-v3-6-oct-24th/blob/main/aave-v3-origin-blackthorn/src/contracts/protocol/libraries/logic/ValidationLogic.sol#L342-L344

```
  function validateLiquidationCall(
    DataTypes.UserConfigurationMap storage borrowerConfig,
    DataTypes.ReserveData storage collateralReserve,
    DataTypes.ReserveData storage debtReserve,
    DataTypes.ValidateLiquidationCallParams memory params
  ) internal view {
    ...

@>  vars.isCollateralEnabled =
      collateralReserve.configuration.getLiquidationThreshold() != 0 &&
      borrowerConfig.isUsingAsCollateral(collateralReserve.id);

    //if collateral isn't enabled as collateral by user, it cannot be liquidated
@>  require(vars.isCollateralEnabled, Errors.CollateralCannotBeLiquidated());
  }
```

Impact

Positions using eMode-only collaterals become non-liquidatable, potentially leading to bad debt.

Recommendation

During liquidation, determine collateral validity using the liquidatee's active eMode configuration instead of relying solely on the base liquidation threshold.

Issue L-1: eMode-Only Collaterals Bypass Supplier Check When Setting Debt Ceiling

Source: https://github.com/sherlock-audit/2025-10-aave-v3-6-oct-24th/issues/13

Vulnerability Detail

In Aave v3.6, collateral eligibility is now separated between normal mode (`eMode = 0`) and opt-in eModes. However, the `setDebtCeiling()` function still relies on the normal mode liquidation threshold (`currentConfig.liquidationThreshold`) to determine whether an asset can have suppliers before assigning a new debt ceiling.

This logic assumes that a zero liquidation threshold implies no one has supplied the asset. While true in older versions, this assumption breaks under the new eMode design. An asset may have `currentConfig.getLiquidationThreshold() == 0` in normal mode but still be actively supplied within an eMode.

As a result, `_checkNoSuppliers()` may be skipped even though suppliers exist, violating the isolation mode requirement that no suppliers should exist when assigning a new debt ceiling.

https://github.com/sherlock-audit/2025-10-aave-v3-6-oct-24th/blob/main/aave-v3-origin-blackthorn/src/contracts/protocol/pool/PoolConfigurator.sol#L309-L311

```
  function setDebtCeiling(
    address asset,
    uint256 newDebtCeiling
  ) external override onlyRiskOrPoolAdmins {
    DataTypes.ReserveConfigurationMap memory currentConfig =
    ↪  _pool.getConfiguration(asset);

    uint256 oldDebtCeiling = currentConfig.getDebtCeiling();
@>  if (currentConfig.getLiquidationThreshold() != 0 && oldDebtCeiling == 0) {
      _checkNoSuppliers(asset);
    }
    currentConfig.setDebtCeiling(newDebtCeiling);
    _pool.setConfiguration(asset, currentConfig);
```

```
    if (newDebtCeiling == 0) {
      _pool.resetIsolationModeTotalDebt(asset);
    }

    emit DebtCeilingChanged(asset, oldDebtCeiling, newDebtCeiling);
  }
```

Impact

The protocol may set a non-zero debt ceiling for an asset that already has suppliers in eMode. This can lead to incorrect accounting of `isolationModeTotalDebt`.

Recommendation

Check all eMode configs the asset is not used as collateral.

Issue L-2: ltv0 assets not enforced to withdraw first when no borrowing exists

Source: https://github.com/sherlock-audit/2025-10-aave-v3-6-oct-24th/issues/14

Vulnerability Detail

According to https://github.com/bgd-labs/aave-v3-origin-blackthorn/blob/v3.6.0/docs/3.6/Aave-v3.6-properties.md#glossary, the ltv0 rule contains:

> ltv0 asset(s) must be withdrawn before withdrawing any other asset.

However, the current implementation only enforces this rule inside functions that are wrapped by an `isBorrowing()` check. Specifically, `executeWithdraw()` and `executeFinalizeTransfer()` functions. If the user has no active borrows, these functions skip the ltv0 rules enforcement.

Note since the user has no borrows, not forcing the ltv0 asset first withdrawal won't have much impact. The issue lies primarily in the deviation from the specified behavior in the protocol documentation.

https://github.com/sherlock-audit/2025-10-aave-v3-6-oct-24th/blob/main/aave-v3-origin-blackthorn/src/contracts/protocol/libraries/logic/SupplyLogic.sol#L164

```
function executeWithdraw(
  mapping(address => DataTypes.ReserveData) storage reservesData,
  mapping(uint256 => address) storage reservesList,
  mapping(uint8 => DataTypes.EModeCategory) storage eModeCategories,
  DataTypes.UserConfigurationMap storage userConfig,
  DataTypes.ExecuteWithdrawParams memory params
) external returns (uint256) {
  ...
  if (userConfig.isUsingAsCollateral(reserve.id)) {
    if (zeroBalanceAfterBurn) {
      userConfig.setUsingAsCollateral(reserve.id, params.asset, params.user,
      ↪  false);
    }
```

```
@>    if (userConfig.isBorrowingAny()) {
        ValidationLogic.validateHFAndLtvzero(
          reservesData,
          reservesList,
          eModeCategories,
          userConfig,
          params.asset,
          params.user,
          params.oracle,
          params.userEModeCategory
        );
      }
    }
    ...
  }

  function executeFinalizeTransfer(
    mapping(address => DataTypes.ReserveData) storage reservesData,
    mapping(uint256 => address) storage reservesList,
    mapping(uint8 => DataTypes.EModeCategory) storage eModeCategories,
    mapping(address => DataTypes.UserConfigurationMap) storage usersConfig,
    DataTypes.FinalizeTransferParams memory params
  ) external {
    ...
      if (fromConfig.isUsingAsCollateral(reserveId)) {
        if (params.scaledBalanceFromBefore == params.scaledAmount) {
          fromConfig.setUsingAsCollateral(reserveId, params.asset, params.from,
          ↪  false);
        }
@>      if (fromConfig.isBorrowingAny()) {
          ValidationLogic.validateHFAndLtvzero(
            reservesData,
            reservesList,
            eModeCategories,
            usersConfig[params.from],
            params.asset,
            params.from,
            params.oracle,
```

```
            params.fromEModeCategory
          );
        }
      }
  }
```

Impact

No real impact. Different implementation from specs.

Recommendation

Fix documentation or code accordingly.

Disclaimers

Blackthorn does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.