SPEARBIT

Aave Aptos V3.1-3.3 Core Upgrades Security Review

Auditors

Christoph Michel, Lead Security Researcher

Mengxu, Lead Security Researcher

Emanuele Ricci, Security Researcher

T1moh, Associate Security Researcher

Jay, Security Researcher

Report prepared by: Lucas Goiriz

June 18, 2025

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Aave Labs creates smart contract-enabled products and public goods (open source protocols) that incorporate decentralized blockchain technologies and token-based economies.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Aave Aptos V3.1-3.3 Core Upgrades
according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 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

4 Executive Summary

Over the course of 3 days in total, Aave Labs engaged with Spearbit to review review Aave Aptos V3.1-3.3 upgrades. In this period of time a total of **17** issues were found.

Summary

Project Name	Aave Aptos V3.1-3.3 Core Upgrades
Repository	aptos-v3-sb-audit
Commit	3668ac01
Type of Project	DeFi, Lending
Audit Timeline	May 19th to May 21st

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	2	2	0
Medium Risk	1	1	0
Low Risk	8	6	2
Gas Optimizations	0	0	0
Informational	6	6	0
Total	**17**	**15**	**2**

5 Findings

5.1 High Risk

5.1.1 Incorrect next scaled variable debt update in liquidations leads to wrong interest rates

Severity: High Risk

Context: liquidation_logic.move#L821-L823

Description: When liquidating and `liquidation_logic::burn_debt_tokens` is called, the `debt_reserve_cache`'s `scaled_variable_debt` (debt token total supply) needs to be updated after the burn by setting it to the `next_scaled_variable_debt` variable.

However, the code currently sets the next borrow index (not scaled total supply) to the `next_scaled_variable_debt` value, which are different units, borrow index is in RAY (`1e27`), while `next_scaled_variable_debt` is in debt token units (usually 6-8 token decimals).

```
pool_logic::set_next_variable_borrow_index( // <-- borrow index
    debt_reserve_cache, next_scaled_variable_debt // <-- scaled var debt
);
```

The impact is that the `pool_logic::update_interest_rates_and_virtual_balance` call that follows, updates the interest rates incorrectly as its usage ratio computation is invalid. After each liquidation, the interest rates of the debt token are wrong (usually small interest rates as `borrow_index > scaled_variable_debt` on Aptos and `total_variable_debt` is close to 0):

```
let total_variable_debt =
    wad_ray_math::ray_mul(
        reserve_cache.next_scaled_variable_debt, // <-- this is current_scaled_variable_debt
        reserve_cache.next_variable_borrow_index // <-- this is next_scaled_variable_debt
    );
```

Borrows pay less interest and suppliers earn less afterward until a new interest rate update is triggered.

Recommendation: Consider changing the code:

```
- pool_logic::set_next_variable_borrow_index(
+ pool_logic::set_next_scaled_variable_debt(
      debt_reserve_cache, next_scaled_variable_debt
  );
```

Consider adding more `liquidation_call` tests that cover all the edge cases and assert that all fields of the involved `ReserveData`s, `UserConfigurationMap`s, user balances, and interest rates are updated as expected.

Proof of Concept: See "Testing setup: Default deployment & Liquidation proofs of concept" in the Appendix.

Aave Labs: Fixed in PR 254, in commit b26a428e more specifically.

Spearbit: Fix verified.

5.1.2 Liquidation logic allows the liquidator to liquidate more than it should

Severity: High Risk

Context: liquidation_logic.move#L460-L466

Description: The Aave V3.3 liquidation logic has introduced a mechanism that re-calculates the max amount of debt that the liquidator can liquidate for a specific debt token depending on the borrower's `HF` and debt positions.

```
// by default whole debt in the reserve could be liquidated
uint256 maxLiquidatableDebt = vars.userReserveDebt;
// but if debt and collateral is above or equal MIN_BASE_MAX_CLOSE_FACTOR_THRESHOLD
// and health factor is above CLOSE_FACTOR_HF_THRESHOLD this amount may be adjusted
if (
  vars.userReserveCollateralInBaseCurrency >= MIN_BASE_MAX_CLOSE_FACTOR_THRESHOLD &&
  vars.userReserveDebtInBaseCurrency >= MIN_BASE_MAX_CLOSE_FACTOR_THRESHOLD &&
  vars.healthFactor > CLOSE_FACTOR_HF_THRESHOLD
) {
  uint256 totalDefaultLiquidatableDebtInBaseCurrency = vars.totalDebtInBaseCurrency.percentMul(
    DEFAULT_LIQUIDATION_CLOSE_FACTOR
  );

  // if the debt is more then DEFAULT_LIQUIDATION_CLOSE_FACTOR % of the whole,
  // then we CAN liquidate only up to DEFAULT_LIQUIDATION_CLOSE_FACTOR %
  if (vars.userReserveDebtInBaseCurrency > totalDefaultLiquidatableDebtInBaseCurrency) {
    maxLiquidatableDebt =
      (totalDefaultLiquidatableDebtInBaseCurrency * vars.debtAssetUnit) /
      vars.debtAssetPrice;
  }
}

vars.actualDebtToLiquidate = params.debtToCover > maxLiquidatableDebt
  ? maxLiquidatableDebt
  : params.debtToCover;
```

By default, the liquidator can liquidate the whole debt token position, but if the debt token value in USD is `>= 2000 USD`, the collateral token value is `>= 2000 USD` and the `HF` of the borrower is `> 0.95e18`, the `maxLiquidatableDebt` could be re-calculated.

The `maxLiquidatableDebt` is recalculated when the total liquidable debt in USD for the debt token is greater than the 50% of the total (across all the debt positions) debt in USD.

The Move codebase had incorrectly implemented this feature and will recalculate the debt when the `userReserveDebtInBaseCurrency` is lower than the `totalDefaultLiquidatableDebtInBaseCurrency`.

Let's assume this scenario and see the outcome in both the Solidity and Aptos implementation.

- Borrower has `5000 USDC` of debt.
- Borrower has `2000 USDT` of debt.
- Borrower has `8000 DAI` of collateral.
- Borrower has `0.95 < HF < 1e18`, can be liquidated but not in full.

The liquidator tries to liquidate the whole USDC debt of `5000 USD`. In both Solidity and Aptos we enter the first part of the `if` branch and `totalDefaultLiquidatableDebtInBaseCurrency` is calculated as `5000 USD + 2000 USD / 2 = 3500 USD`. On Solidity, we enter the `if (vars.userReserveDebtInBaseCurrency > totalDefaultLiquidatableDebtInBaseCurrency)` given that the debt of the `USDC` token is greater than the 50% of the total debt and the `maxLiquidatableDebt` will be equal to `3500 USD`.

On Aptos codebase, instead, we have the opposite check and the `maxLiquidatableDebt` is **not** recalculated, allowing the liquidator to **fully liquidate** the whole `USDC` position and seize the corresponding collateral.

Recommendation: Aave Protocol must update the logic, aligning it to the Solidity implementation to prevent the liquidator to being able to over-liquidate the borrower's position.

```
- if (vars.user_reserve_debt_in_base_currency < total_default_liquidatable_debt_in_base_currency) {
+ if (vars.user_reserve_debt_in_base_currency > total_default_liquidatable_debt_in_base_currency) {
      max_liquidatable_debt = (
          total_default_liquidatable_debt_in_base_currency
              * vars.debt_asset_unit
      ) / vars.debt_asset_price;
  }
```

Proof of Concept: See "Testing setup: Default deployment & Liquidation proofs of concept" in the Appendix.

Aave Labs: Fixed in commit 9e95433e.

Spearbit: Fix verified.

5.2 Medium Risk

5.2.1 `GHO` is misconfigured in Aptos deployment

Severity: Medium Risk

Context: pool_configurator.move#L421

Description: The GHO, aGHO and vGHO tokens on the Ethereum Mainnet are ERC20 tokens with some specific custom behavior and properties. For this reason, the Solidity codebase had to introduce custom logic in the code to distinguish the case where the user was dealing with a "normal" ERC20 reserve or the custom one: non-`GHO` tokens are identified by the "use-virtual-accounting" flag.

These are just some of the "custom logics" used for the `GHO` token (and `vGHO`).

- DefaultReserveInterestRateStrategyV2.sol#L132-L134: The interest rate strategy only uses the `baseVariableBorrowRate` to calculate the interest rate for `GHO` borrowers, resulting in a "fixed" borrow rate.
- GenericLogic.sol#L147-L154: Fetch the debt balance directly, using the `vGHO.balanceOf` to include the discount on debt.
- LiquidationLogic.sol#L556-L584: Custom liquidation logic to handle the debt burning of `vGHO` to update the internal state of the variable debt token.
- LiquidationLogic.sol#L119-L158: Custom logic to handle the elimination of the deficit for `GHO` bad debt.
- Plenty of other cases where the `reserveConfiguration.getIsVirtualAccActive()` is checked.

On other chains that are not Ethereum Mainnet (like Base and Arbitrum) the `GHO` configuration is quite different:

- the `GHO` token is bridged as a "normal" ERC20 token.
- the `aGHO` and `vGHO` tokens have been deployed as a "normal" supply and variable debt token without any "custom" logic.
- the `GHO` reserve is configured with the `IsVirtualAccActive` flag set to `TRUE`.
- `DefaultReserveInterestRateStrategyV2` is configured to use the normal strategy logic:
- `optimalUsageRatio = 9000 (bps).`
- `baseVariableBorrowRate = 0.`
- `variableRateSlope1 = 750 (bps).`
- `variableRateSlope2 = 5000 (bps).`

The same configuration used for Base and Arbitrum will probably also be applied to the deployment and configuration of `GHO` for the Aptos blockchain. xThe issue in this case is that the Aptos codebase is implemented to handle `GHO` as it's custom-handled in Ethereum Mainnet.

We know for sure that at least the interest rate will **only** use the fixed borrow interest rate (see how the interest_-rate_strategy module works and how the gho_interest_rate_strategy has been implemented) and the codebase also hints (for example) that `GHO` will be configured with the `virtual_acc_active` flag turned to `false`.

Given the behavior of the `GHO` interest strategy, we will have the following problems:

- Suppliers won't earn any interest on the risk taken.
- If the `reserve_factor` is non-empty, only the Aave treasury will be able to earn on the borrower's interest.
- All the borrower's interest (excluding the one minted as fees from the `reserve_factor`) will be "stuck" in the `aGHO` resource account and won't be possible to be rescued. The `a_token_factory::rescue_tokens` reverts when the `pool_admin` tries to rescue the `AToken`'s underlying.

Recommendation: Relative to the interest rate strategy, Aave Protocol should:

- Delete the `aave_rate::gho_interest_rate_strategy` module.
- Delete the `aave_rate::interest_rate_strategy`.
- Update the `aave_rate::default_reserve_interest_rate_strategy` to implement the `fun init_module(account: &signer)` initialization procedure.
- Change the current codebase to directly call the `default_reserve_interest_rate_strategy`.

Given that the Aptos codebase will exclusively be used for Aptos (unlike the Solidity one that is common across different chains) and given that the `virtual_acc_active == false` flag is used on Solidity exclusively to identify `GHO`, we can assume that all the reserves in Aptos will always be configured as "virtual accounting" reserves.

With such assumptions, Aave Protocol could simplify and cleanup the codebase by:

- Removing all the checks on the `virtual_acc_active` flag.
- Removing all the logics executed when `virtual_acc_active == false`.
- Considering removing the `virtual_acc_active` flag from the reserve configuration.

In addition to the above changes, we suggest Aave Protocol to pay particular attention to the configuration and deployment of the `GHO` reserve which, at least on Base and Arbitrum, can be supplied but not used as collateral. In general, the `GHO` reserve on mainnet should be considered a special and unique case that should not be used as a reference for other chains.

Aave Labs: Fixed in PR 355.

Spearbit: Fix verified.

5.3 Low Risk

5.3.1 `default_reserve_interest_rate_strategy` should be based on the logic from Solidity `DefaultReserveInterestRateStrategyV2`

Severity: Low Risk

Context: default_reserve_interest_rate_strategy.move#L6

Description: The current implementation of `aave_rate::default_reserve_interest_rate_strategy` is based on the deprecated DefaultReserveInterestRateStrategy from Aave V3.0.2.

Given that most of the Aptos code has been moved to Aave V3.3, the same upgrade should also apply to upgrade the logic of the interest rate strategy to the new DefaultReserveInterestRateStrategyV2.

Upgrading to the new codebase would provide the following benefits that would be anyway requested in other issues:

- Use the new event `RateDataUpdate` instead of `ReserveInterestRateStrategy`.
- Perform the required sanity checks when the reserve interest rate config is updated.
- Simplify the logic of calculation for the supply and borrow rate.

- Store the configuration in BPS format instead of RAY.

Recommendation: Aave Protocol should consider migrating the current `default_reserve_interest_rate_-strategy` module to use the same logic used by the DefaultReserveInterestRateStrategyV2 from Aave V3.3.

Aave Labs: Fixed in PR 355.

Spearbit: Fix verified.

5.3.2 `validate_flashloan_simple` misses a sanity check introduced by Aave V3.3

Severity: Low Risk

Context: validation_logic.move#L41-L53

Description: The `validateFlashloanSimple` function on Aave V3.3 reverts if the requested amount is greater than the `aToken.totalSupply()` (see ValidationLogic.sol#L357-L361).

```
require(
  !configuration.getIsVirtualAccActive() ||
    IERC20(reserve.aTokenAddress).totalSupply() >= amount,
  Errors.INVALID_AMOUNT
);
```

This sanity check is **not** performed in the Aptos `public fun validate_flashloan_simple`.

Recommendation: Aave Protocol should revert the flashloan operation (simple or complex) if the amount requested is greater than the `AToken` total supply, aligning the validation logic to the one used by the Aave V3.3 codebase.

Aave Labs: Fixed in PR 299.

Spearbit: Fix verified.

5.3.3 `validate_borrow` misses a sanity check introduced by Aave V3.1

Severity: Low Risk

Context: validation_logic.move#L195

Description: The Aave V3.1 release has introduced an additional sanity check on the borrow `amount` requested by the caller (see ValidationLogic.sol#L154-L158).

```
require(
    !params.reserveCache.reserveConfiguration.getIsVirtualAccActive() ||
    IERC20(params.reserveCache.aTokenAddress).totalSupply() >= params.amount,
    Errors.INVALID_AMOUNT
);
```

This sanity check is not performed in the Aptos `validate_borrow` function.

Recommendation: Aave Protocol should revert the `borrow` operation if the requested `amount` is greater than the `AToken` total supply.

Aave Labs: Fixed in PR 376.

Spearbit: Fix verified.

5.3.4 `set_reserve_freeze` does not revert when the reserve is already in the desired state

Severity: Low Risk

Context: pool_configurator.move#L570-L571

Description: The current implementation of `pool_configurator::set_reserve_freeze` is not reverting when the state of the reserve is equal to the `freeze` input parameter.

This issue has been already reported and addressed on the Aave Solidity Implementation codebase (see Cantina-contest-AaveV3.1 and should also be replicated on the Aptos codebase.

By not performing such check we will encounter unexpected behaviors.

- Scenario 1) reserve is already frozen.

  In this case, `_pendingLtv[asset]` is set to `0` and we lose the previous value of the LTV stored in the `_pendingLtv` that would be re-used once the reserve will be unfrozen.

- Scenario 2) reserve is already unfrozen.

  The LTV of the reserve will be set to `_pendingLtv[asset]` which is equal to `0`.

Recommendation: Aave Protocol must revert when the reserve is already in the same frozen state expressed by the `freeze` input parameter.

Aave Labs: Fixed in PR 269.

Spearbit: Fix verified.

5.3.5 Users near `MIN_BASE_MAX_CLOSE_FACTOR_THRESHOLD` can be fully liquidated by using multi liquidations

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: Aave 3.3 introduced new minimum position size thresholds that must exist after liquidations if the entire balance cannot be cleared. This is to prevent position sizes of small amounts that are not unprofitable to liquidate, for example, because of gas costs.

- The `MIN_BASE_MAX_CLOSE_FACTOR_THRESHOLD` is currently set to 2000$.
- The "dust value" is half of that, set to `MIN_LEFTOVER_BASE = 1000$`.

The rules are as follows:

- **Full-liquidation:** The max close factor is 50% of the user's total debt value (*no* full liquidation) if-and-only-if `HF > 95%` AND `debt$ >= 2000$` AND `collateral$ >= 2000$`. Otherwise, the close factor is 100%.
- **Liquidation failures due to dust position sizes:** Liquidation fails if both collateral and debt balances are non-zero after the liquidation and *any* of the balances is less than 1000$: `collateral_left$ > 0` AND `debt_left$ > 0` AND (`collateral_left$ < 1000$` OR `debt_left$ < 1000$`).

Impact: These rules allow a liquidator first to perform a small liquidation bringing the user below the position thresholds, so the next liquidation can be a full liquidation. In the end, the user is fully liquidated, which shouldn't have been possible given the rules for their initial position.

Example:

- Collateral `C`: 4200$.
- Debt `D` = 2210$.
- LT = 50%.
- Choose `liqBonus = 1.10`.

User's health factor (HF) = `4200 * 0.5 / 2210 = 95.02%` > 95%.

Note that we can't full liquidate the user right now because the rules for `max_close_factor = 50%` apply. Liquidating 2210$ to seize all 2431$ collateral does not work. However, the liquidator can make it happen in two liquidation steps:

1. Liquidate 211$ debt => seize 232$.

- C = 4200 - 232 = 3968.
- D = 2210 - 211 = 1999.
- HF = 3968 * 0.5 / 1999 = 99.24% < 100%.

The user is still unhealthy, so they can be liquidated again. However, their debt value now slipped below the 2000$ threshold, so they can be fully liquidated with `max_close_factor = 100%`:

2. Liquidate 1,999$ debt for 2,199$ collateral.

- C = 3,968 - 2,199 = 1769$.
- D = 1,999 - 1,999 = 0$.

The liquidation does not revert because `collateral_balance = 0`. In total, the liquidator full-liquidated user's `1999 + 211 = 2,210$` debt for `2,431$` collateral, which shouldn't and wouldn't have been possible with a single liquidation.

Recommendation: By introducing new rules when a user can be fully liquidated, it can now become profitable to perform multiple smaller liquidations that end up liquidating more than what was possible with a single liquidation. Consider revisiting the minimum position size thresholds feature and rules about when a user can be fully liquidated.

Aave Labs: Acknowledged.

Spearbit: Acknowledged.

5.3.6 Minimum position size post-liquidation checks can end up reverting legitimate liquidations

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: Aave 3.3 introduced new minimum position size thresholds that must exist after liquidations if the entire balance cannot be cleared. This is to prevent position sizes of small amounts that are not unprofitable to liquidate, for example, because of gas costs.

- The `MIN_BASE_MAX_CLOSE_FACTOR_THRESHOLD` is currently set to 2000$.
- The "dust value" is half of that, set to `MIN_LEFTOVER_BASE = 1000$`.

The rules are as follows:

- **Full-liquidation:** The max close factor is 50% of the user's total debt value (*no* full liquidation) if-and-only-if `HF > 95%` AND `debt$ >= 2000$` AND `collateral$ >= 2000$`. Otherwise, the close factor is 100%.
- **Liquidation failures due to dust position sizes:** Liquidation fails if both collateral and debt balances are non-zero after the liquidation and *any* of the balances is less than 1000$: `collateral_left$ > 0` AND `debt_left$ > 0` AND (`collateral_left$ < 1000$` OR `debt_left$ < 1000$`).

Impact: Some liquidations that seize a legitimate user collateral balance and repay a legitimate debt balance end up reverting.

Example:

- Two collaterals `C_0 = 2000$` and `C_1 = 2000$`.
- Debt `D = 2001$`.
- LT = 50%.
- Choose `liqBonus = 1.20`.

User's health factor (HF) `= 4000 * 0.5 / 2001 = 99.9%` > 95%. All values are at or above `MIN_BASE_MAX_-CLOSE_FACTOR_THRESHOLD = 2000`, we can only liquidate 50%. If we liquidate we can choose only one collateral, it doesn't matter which one, it's symmetric, let's say `C_0`.

The max close factor is 50%, repaying `1000$` of debt allows seizing `1200$` of collateral. After liquidation:

- C_0 = 800$, C_1 = 2000$.
- D = 1000$.

Note that the collateral balance C_0 is below MIN_LEFTOVER_BASE and therefore reverts.

Recommendation: Liquidators can't simply trust that liquidations that seize up to max close_factor collateral succeed, even if the user has that amount of collateral. They need to explicitly compute how much debt to repay such that the post-liquidation collateral balance after seizing liquidation_bonus * repaid_value does not fall below the MIN_LEFTOVER_BASE threshold.

Note that even then, there can be a DoS if the violator frontruns the liquidation and reduces their collateral balance by a small amount by using repay_with_a_tokens, such that, the collateral balance after liquidation is again below the threshold.

Aave Labs: Acknowledged.

Spearbit: Acknowledged.

5.3.7 Full removal of the admin_controlled_ecosystem_reserve

Severity: Low Risk

Context: admin_controlled_ecosystem_reserve.move#L1

Description: By looking at the Aave V3.3 codebase in GitHub relative to the Treasury management, we can see that the "old" AdminControlledEcosystemReserve contract has been fully removed. From our own research, such a contract is not used anymore within the Aave ecosystem, and the removal from the codebase is validating our hypothesis.

On every EVM chain, the AToken treasury is deployed as a Collector contract, and the REWARDS_-VAULT of the PullRewardsTransferStrategy contract is configured as the "ACI multisig address" 0xac140648435d03f784879cd789130F22Ef588Fcd.

Given all these facts, we think that the aave_pool::admin_controlled_ecosystem_reserve should be fully removed from the Aave Aptos Implementation codebase.

Recommendation: Aave Protocol should fully remove the aave_pool::admin_controlled_ecosystem_reserve module from the codebase unless there's a specific purpose that, currently, has not been explained and justified to Spearbit. Please note that this finding is not detailing all the existing issues that have been already raised relative to the current implementation of admin_controlled_ecosystem_reserve_tests. They have not been mentioned because we are suggesting the full removal of the module from the database.

Aave Labs: Fixed in PR 383.

Spearbit: Fix verified.

5.3.8 AggregatedReserveData misses the virtual_underlying_balance and is_virtual_acc_active data attributes

Severity: Low Risk

Context: ui_pool_data_provider_v3.move#L48

Description: The AggregatedReserveData struct in ui_pool_data_provider_v3 is missing two attributes that should be returned by the get_reserves_data function.

These two values are:

- virtual_underlying_balance which represents the underlying balance held in the AToken resource account.
- is_virtual_acc_active reserve flag.

Recommendation: Aave Protocol should add this information to the `AggregatedReserveData` struct and fetch for each reserve's asset in the `get_reserves_data` function execution.

Aave Labs: Fixed in PR 343.

Spearbit: Fix verified.

5.4 Informational

5.4.1 `mint_to_treasury` Move ⇔ Solidity discpreancy

Severity: Informational

Context: pool_token_logic.move#L233

Description: The solidity implementation of `mint_to_treasury` (PoolLogic.executeMintToTreasury) skips invalid reserve addresses while the Move version reverts upon encountering an invalid reserve in `pool::get_reserve_configuration_by_reserve_data(reserve_data)`.

Recommendation: Clarify the intended behavior. Consider removing the following comment as it does not reflect the current behavior of the Move version:

> // this cover both inactive reserves and invalid reserves since the flag will be 0 for both.

```
let reserve_data = pool::get_reserve_data(asset_address); // <-- would revert here already for "invalid
↪   reserves"
let reserve_config_map =
    pool::get_reserve_configuration_by_reserve_data(reserve_data);

// this cover both inactive reserves and invalid reserves since the flag will be 0 for both
if (!reserve_config::get_active(&reserve_config_map)) {
    continue
};
```

Aave Labs: Fixed in PR 326.

Spearbit: Fix verified.

5.4.2 The `VIRTUAL_ACC_ACTIVE` reserve flag should be deprecated

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The "Virtual Account Active" flag has been introduced in Aave Protocol EVM to distinguish the logic to be applied when the supply of the reserve's underlying was updated or queried. Now, the flag will mainly identify if the `reserve` is the `GHO` token or a "normal" `ERC20` token.

Given that on Aptos the `GHO` token will be a normal `ERC20` token bridged from other chains, there's no more the need for such a flag: all the tokens will use the virtual underlying balance to account for the underlying supply.

By deprecating such a flag, Aave can remove part of the code and simplify all the logic, eliminating all the possible confusion and unexpected behavior in the case where the reserve is wrongly configured.

Recommendation: Aave should deprecate the `VIRTUAL_ACC_ACTIVE` flag from the `aave_config::reserve_config` and perform the following actions:

- Comment `VIRTUAL_ACC_ACTIVE_MASK` in `aave_config::reserve_config`.
- Comment `VIRTUAL_ACC_START_BIT_POSITION` in `aave_config::reserve_config`.
- Remove the `set_virtual_acc_active` function.
- Remove the `get_virtual_acc_active` function.

- Everywhere the `reserve_config::get_virtual_acc_active` was checked, only implement the logic path where the flag was equal to `true` and remove the other part of the logic branch.

Aave Labs: Fixed in PR 313.

Spearbit: Fix verified.

5.4.3 `update_interest_rate_strategy` should emit the `ReserveInterestRateDataChanged` event

Severity: Informational

Context: pool_configurator.move#L430

Description: The `update_interest_rate_strategy` should follow the same behavior as the `_updateInterestRateStrategy` function on the Solidity codebase and emit the `ReserveInterestRateDataChanged` event once the interest rate strategy configuration for the `asset` has been updated.

Recommendation: Aave Protocol should define the `ReserveInterestRateDataChanged` event like in the Solidity codebase and emit it when the IRS configuration for the reserve has been updated.

Aave Labs: Fixed in PR 296.

Spearbit: Fix verified.

5.4.4 `pool_configurator` does not offer the `set_reserve_pause_no_grace_period` function implemented in Solidity

Severity: Informational

Context: pool_configurator.move#L642

Description: In the Solidity codebase, the `PoolConfigurator` contract exposes the `function setReservePause(address asset, bool paused) external;` function which internally calls `setReservePause(asset, paused, 0);`.

Basically, it's a utility version that won't give any grace period. Move does not allow you to overload a function's signature, so if Aave wants to provide the other flavor of the function they need to provide a new meaningful name like `set_reserve_pause_withoutgrace_period`.

```
public entry fun set_reserve_pause_no_grace_period(
    account: &signer,
    asset: address,
    paused: bool
) {
    set_reserve_pause(account, asset, paused, 0);
}
```

Recommendation: Aave Protocol should consider implementing the `set_reserve_pause_no_grace_period` to pause a reserve without a grace period, like the Solidity implementation already offer.

Aave Labs: Fixed in PR 310.

Spearbit: Fix verified.

5.4.5 `pool_configurator` does not offer the `set_pool_pause_no_grace_period` function implemented in Solidity

Severity: Informational

Context: pool_configurator.move#L1028

Description: Similar to Finding #61, the `pool_configurator` should offer a function that allows the caller to set the pause state of all the reserves without specifying a `grace_period` like the Solidity codebase is already offering.

Recommendation: Aave Protocol should consider implementing the `set_pool_pause_no_grace_period` utility function already offered in the Solidity codebase.

Aave Labs: Fixed in PR 311.

Spearbit: Fix verified.

5.4.6 Full removal of `eMode` `price_source`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Aave has never used the custom `price_source` stored in the eMode category and has officially removed the support for such feature with the deployment of Aave V3.2 with the introduction of Liquid eMode.

Even if the current implementation of Aave on Aptos does have not migrated to Liquid eMode, and it's still using the "old" eMode feature, we strongly suggest the team to remove completely the support to `price_source`.

In the previous review we have suggested to forcefully setting the `price_source` to `@0x0` to ensure that the eMode price source won't ever be used, but we have come to the conclusion that the full removal is the best solution to keep the code more safe, secure and clear.

Recommendation: Aave Protocol should:

- Remove `price_source` from the `EModeCategory` struct in the `emode_logic` module.
- Remove any direct getter or setter relative to the `price_source` attribute.
- Remove `price_source` from any function's signatures.
- Remove `price_source` everywhere from the codebase and use directly the `asset` to fetch the price from the oracle.

Aave Labs: Fixed in PR 305.

Spearbit: Fix verified.