Aave Aptos V3.0.2 Core Security Review

Auditors

Christoph Michel, Lead Security Researcher

Emanuele Ricci, Lead 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 V3.0.2 Core 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 38 days in total, Aave Labs engaged with Spearbit to review the aptos-v3-sb-audit protocol. In this period of time a total of **69** issues were found.

Summary

Project Name	Aave Aptos V3.0.2 Core
Repository	aptos-v3-sb-audit
Commit	3668ac01
Type of Project	DeFi, Lending
Audit Timeline	Mar 12th to Apr 19th

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	1	1	0
Low Risk	27	25	2
Gas Optimizations	3	3	0
Informational	37	27	10
Total	**69**	**57**	**12**

5 Findings

5.1 High Risk

5.1.1 Dust amounts of high-value tokens can be stolen for profit due to rounding

Severity: High Risk

Context: token_base.move#L231, token_base.move#L321

Description: The `mint` and `burn` actions for the `aToken` and `vToken` use the same rounding direction to convert between (rebased) amounts and the scaled amounts, rounding "half-up":

```
let amount_scaled = wad_ray_math::ray_div(amount, index);
```

This can be abused to extract tokens from the protocol.

Example: `liquidityIndex = 1.5e27`.

- `supply(1)`: pay 1 `amount` to receive `round(1 / 1.5) = round(0.66) = 1` `scaled_amount`.
- `withdraw(2)`: receive 2 amount for burning `round(2 / 1.5) = round(1.33) = 1` `scaled_amount`.

In total, the user supplied 1 amount and received 2 amounts, making a profit of 1.

Impact Explanation: The token can't be listed on Aave Aptos or if it is listed the entire pool can be stolen for profit.

Likelihood Explanation: Note that this issue also exists on Aave Solidity, however, on Aptos it is more likely because:

1. The token decimals are much lower on Aptos because the balances are restricted to `u64`s. Token decimals are usually between `6` and `8` but all tokens need to be checked individually. Therefore, the smallest unit of a token is more valuable compared to high-decimal tokens.
2. The transaction cost is much lower on Aptos.

Imagine the following conservative but still realistic assumptions to estimate the cost and profitability of the attack:

1. Average gas price of 100 octas per gas unit (`1e-6` APT = 0.000001 APT).
2. Amortized cost per `supply & withdraw` iteration to steal a unit of the asset is 10 gas units. This is likely higher in practice and should be properly estimated by gas profiling the proof of concept when run as a script against a deployed network via `aptos move run-script <..> --profile-gas` (we were unable to deploy the local test network as chainlink deployment modules failed with dependency error `Move abort in 0x1::code: EPACKAGE_DEP_MISSING(0x60005): Dependency could not be resolved to any published package.`).
3. APT price of `1$` / APT. As of April 10 2025, the APT price is `4.56$` and price drops of 80% are not uncommon in crypto.

With these assumptions, the cost to steal 1 asset is `10 gas units * 100 octas / gas unit` = `1e-4` APT. To be profitable, 1 unit of a 6-decimal token must be worth more than `0.0001$`, or "1.0" of the 6-decimal token must be worth more than 10$.

There are several lists of coins on Aptos, for example, the following ones would be exploitable:

- Bridged WETH: LayerZero's 6-decimals coin, price of `1582$`. Same with the 8-decimal Wormhole and Celer bridged WETH.
- Bridged BTC: LayerZero's 6-decimals, price of `81785$`. Same with the 8-decimal Wormhole and Celer bridged BTC.

Recommendation: Consider rounding in favor of the protocol:

- For `aTokens`: rounding up burned scaled amounts (`withdraw`), rounding down minted scaled amounts (`supply`). `transfer`s can round up scaled amounts to ensure the receiver gets at least the specified rebased `amount`.

- For `vTokens`: rounding down burned scaled amounts (`repay`), rounding up minted scaled amounts (`borrow`).

> Note: This is a very intrusive change and requires careful handling all over the code base whenever `totalSupply` or a user's `balance` is computed (whenever `wad_ray_math` is called). This also leads to an asymmetry between `supply <> withdraw` and `borrow <> repay`: a larger amount must be `supplied` (`repaid`) compared to what can be `withdrawn` (`borrowed`). For example, this breaks fee-free borrow-flashloans as more needs to be repaid than what was borrowed. A similar approach is planned for Aave 3.4 and it might be worth coordinating with BGD Labs on these changes.

Alternatively, ensure that for all tokens listed on Aave, attacks similar to the one mentioned above are not profitable. On average, the attacker can be expected to steal `floor(liquidityIndex / RAY)` tokens. Onboarding requires pricing this amount (for an expected liquidity index growth after X years) and comparing it to the expected cost of the attack.

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

```
**[test]:** fun test_steal_dust() {
    let d = deployment_helper::create_default_deployment();
    let reserves = d.get_reserves();
    let user = &d.get_user_signer(0);
    let lp = &d.get_user_signer(1);

    let c0 = reserves.borrow(0);

    // change pool liquidity index
    pool::set_reserve_liquidity_index_for_testing(
        c0.get_underlying(),
        // 1.5e27
        1500000000000000000000000000u128
    );

    // provide some initial supply so user has something to steal
    supply_logic::supply(
        lp,
        c0.get_underlying(),
        100_000_000,
        signer::address_of(lp),
        0
    );
    supply_logic::supply(
        user,
        c0.get_underlying(),
        1,
        signer::address_of(user),
        0
    );

    let initial_user_uc0_balance = mock_underlying_token_factory::balance_of(
        signer::address_of(user), c0.get_underlying()
    );
    // do the attack
    let iterations = 1_000;
    for (i in 0..iterations) {
        supply_logic::supply(
                user,
                c0.get_underlying(),
                1,
                signer::address_of(user),
                0
```

```
        );
        supply_logic::withdraw(
                user,
                c0.get_underlying(),
                2,
                signer::address_of(user)
        );
    };

    let final_user_uc0_balance = mock_underlying_token_factory::balance_of(
        signer::address_of(user), c0.get_underlying()
    );
    assert_eq_s(final_user_uc0_balance - initial_user_uc0_balance, iterations, b"user balance");
}
```

Aave Labs: Fixed in PR 403.

Spearbit: Fix verified.

5.2 Medium Risk

5.2.1 `MIN_BASE_MAX_CLOSE_FACTOR_THRESHOLD` has outdated value

Severity: Medium Risk

Context: liquidation_logic.move#L106

Description: `MIN_BASE_MAX_CLOSE_FACTOR_THRESHOLD` is gas price dependent variable, for Mainnet it's configured as 2000 USD. And uses same value `2000e8` on Aptos:

```
/// @dev This constant represents a base value threshold.
/// If the total collateral or debt on a position is below this threshold, the close factor is raised to
↪  100%.
/// @notice The default value assumes that the basePrice is usd denominated by 8 decimals and needs to
↪  be adjusted in a non USD-denominated pool.
/// 2000 * 10 ** 8
const MIN_BASE_MAX_CLOSE_FACTOR_THRESHOLD: u256 = 200000000000;
```

However it's incorrect for 2 reasons:

1) It must have `1e18` precision, because Chainlink on Aptos uses 1e18 precision for price and therefore `base_values` are denominated in `1e18`.
2) 2000 USD is reasonable for Mainnet because of high gas price, however Aptos transactions cost significantly less.

As a result, incorrect value of `MIN_BASE_MAX_CLOSE_FACTOR_THRESHOLD`:

1. Allows to leave dust positions after liquidation:

```
    // to prevent accumulation of dust on the protocol, it is enforced that you either
    // 1. liquidate all debt
    // 2. liquidate all collateral
    // 3. leave more than MIN_LEFTOVER_BASE of collateral & debt
    if (vars.actual_debt_to_liquidate < vars.user_reserve_debt
        ...

        assert!(
            is_debt_more_than_leftover_threshold
                && is_collateral_more_than_leftover_threshold,
            error_config::get_emust_not_leave_dust()
        )
    };
```

2. Prevents full liquidation of any size instead of intended value:

```
// by default whole debt in the reserve could be liquidated
let max_liquidatable_debt = vars.user_reserve_debt;
// 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.user_reserve_collateral_in_base_currency
    >= MIN_BASE_MAX_CLOSE_FACTOR_THRESHOLD
    && vars.user_reserve_debt_in_base_currency
        >= MIN_BASE_MAX_CLOSE_FACTOR_THRESHOLD
    && vars.health_factor > CLOSE_FACTOR_HF_THRESHOLD) {
```

Recommendation: Refactor 2000 USD value and use e18 precision.

Aave Labs: Fixed in PR 410.

Spearbit: Fix verified.

5.3 Low Risk

5.3.1 Emode LTV & LT invariants can be broken

Severity: Low Risk

Context: pool_configurator.move#L496-L499

Description: Entering an emode category as a user should never be worse for the user's health. Therefore, functions like `pool_configurator::set_emode_category` check that the following `reserve <> emode[reserve.emodeCategroy]` invariant holds:

```
reserve.lt  < emode.lt
reserve.ltv < emode.ltv
```

However, this invariant does not hold because not all functions that can change the reserve's / emode's LTV/LT enforce it:

1. `pool_configurator::set_emode_category` enforces this.
2. `pool_configurator::set_asset_emode_category` does not enforce this for LTV.
3. `pool_configurator::configure_reserve_as_collateral` does not enforce this (if the collateral is already part of an eMode category).
4. `pool_configurator::set_reserve_freeze` does not enforce this.

The invariant can for example be broken by calling `configure_reserve_as_collateral` with a reserve that is already part of an eMode category and setting the reserve's LT & LTV higher than its emode's LT & LTV.

Recommendation: Explicitly check the invariant for these functions:

1. `pool_configurator::set_emode_category`: Needs to check the invariant for all reserves that use this emode category.
2. `pool_configurator::set_asset_emode_category`: Needs to check the invariant for the reserve whose emode category is being changed. Note that dropping the emode (setting reserve's emode back to category 0) should be allowed (to prevent deadlocks). However, this will significantly impact the health of the users who use this emode category.
3. `pool_configurator::configure_reserve_as_collateral`: Needs to check the invariant only for the reserve being edited.

As `set_reserve_freeze` sets the reserve's LTV to 0, changing the emode's LTV while its frozen should not be possible if the emode functions perform the invariant checks.

Aave Labs: Fixed in PR 374.

Spearbit: Fix verified.

5.3.2 `set_user_emode` health check always needs to be performed

Severity: Low Risk

Context: emode_logic.move#L105

Description: The `emode_logic::set_user_emode` function skips the health check when the user goes from no-emode to new emode:

```
if (prev_category_id != 0) {
    validation_logic::validate_health_factor(
        &user_config_map,
        account_address,
        category_id,
        reserves_count,
        emode_ltv,
        emode_liq_threshold,
        emode_asset_price
    );
};
```

The idea is likely that if the following invariants hold, the health can only increase, and therefore `set_user_emode` cannot be used to turn a healthy user into an unhealthy user:

```
reserve.lt  < emode.lt
reserve.ltv < emode.ltv
```

However, this invariant does not hold because of the finding "Emode LTV & LT invariants can be broken".

The impact is that a healthy user can call `set_user_emode` and end up unhealthy after the call. This is bad for the user as they can be liquidated afterwards. In addition, having a programmatic way for turning any position liquidatable (under the right circumstances) in a single transaction poses security risks (f.i., see the Euler Finance hack analysis by Cyfrin) and protocol bad debt risk.

Recommendation: Always perform the health check in `set_user_emode`, even if the `prev_category_id` is 0.

```
- if (prev_category_id != 0) {
      validation_logic::validate_health_factor(
          &user_config_map,
          account_address,
          category_id,
          reserves_count,
          emode_ltv,
          emode_liq_threshold,
          emode_asset_price
      );
- };
```

Aave Labs: Fixed in PR 275.

Spearbit: Fix verified.

5.3.3 Chainlink returns raw benchmark price, misinterpreting negative prices

Severity: Low Risk

Context: oracle.move#L22

Description: The `oracle.get_asset_price` fetches the price from chainlink router's benchmark data. This benchmark is extracted from the raw report data and its price, which is a signed `i192`. However, the `i192` is treated and returned as a raw `u256` by chainlink:

```
// registry.move
let feed = simple_map::borrow_mut(&mut registry.feeds, &feed_id);

if (schema == SCHEMA_V3 || schema == SCHEMA_V4) {
    // offsets are the same for timestamp and benchmark in v3 and v4.
    observation_timestamp = (
        to_u32be(vector::slice(&report_data, 3 * 32 - 4, 3 * 32)) as u256
    );
    // NOTE: aptos has no signed integer types, so can't parse as i196, this is a raw representation
    benchmark_price = to_u256be(vector::slice(&report_data, 6 * 32, 7 * 32));
}

feed.benchmark = benchmark_price;
```

Currently, Aave does not perform further checks on this returned `u256` value. Misinterpreting a negative value as a large value would lead to having near-unlimited collateral value to borrow against.

Recommendation:

1. If the price is negative, consider reverting in `get_asset_price(s)` as cryptocurrency prices should never be negative.

   ```
   // 192-th bit is sign bit.
   // all positive values are in [0, 2^191 - 1]
   const I192_MAX = 3138550867693340381917894711603833208051177722232017256447; // 2^191 - 1

   // ...
   let price = chainlink::get_benchmark_value(benchmark);
   assert!(
       price <= I192_MAX,
       error_config::??()
   );
   ```

   > Note: This behavior matches `AaveOracle.sol`'s if the price is negative (and no fallback oracle is provided).

   Chainlink should clarify and consider fixing the comment if the `price` is an `i192` or `i196`.

2. To be in sync with `AaveOracle.sol`'s, consider reverting also if the price is 0 as we could encounter division by 0 errors down the line during health computations.

Aave Labs: Fixed in PR 279.

Spearbit: Fix verified.

5.3.4 `oracle_base::only_risk_or_pool_admin` auth logic is wrong

Severity: Low Risk

Context: oracle_base.move#L56

Description: By looking at the same function _onlyAssetListingOrPoolAdmins in the Solidity codebase, we can see that the users that should be able to call the function protected by this auth function are the ones with the role "asset listing admin" and "pool admin".

On the Aptos implementation instead, we have that:

- "Risk admin" are allowed.
- "Asset listing admin" are not allowed.

Recommendation: Align the auth logic to the one existing in the Solidity codebase and update the `@dev` natspec documentation to reflect the change.

Aave Labs: Fixed in PR 280.

Spearbit: Fix verified.

5.3.5 `get_oracle_base_currency` view function is acquiring a mutable reference to the Global Storage

Severity: Low Risk

Context: oracle_base.move#L103

Description: The `get_oracle_base_currency` function is declared as a `view` function and should not acquire a mutable reference to the global storage.

Recommendation: Use `borrow_global` instead of `borrow_global_mut` inside any function declared as `#[view]`.

Aave Labs: Fixed in PR 281.

Spearbit: Fix verified.

5.3.6 Price returned by the Chainlink feed could be stale or replaced by an older price (compared to the previous one)

Severity: Low Risk

Context: oracle.move#L32

Description: The current implementation of the `oracle::get_asset_price` and `oracle::get_assets_prices` functions do not perform any sanity check on the staleness of the price. This absent of check is aligned with the corresponding implementation on the `AaveOracle` on the Solidity codebase.

One key difference in the Aptos implementation for the Chainlink protocol is that they seem to allow the replacement of the latest "benchmark" (the price) with an updated benchmark that could be older (stale) compared to the current one returned by Chainlink.

If you look at the logic of the `fun perform_update` function in the Chainlink `registry` module codebase, called by the `on_report` function when the price needs to be updated, you can see that the logic **does not** revert when the `observation_timestamp` of the report is equal or **older** compared to the current one used by the actual `feed`.

```
if (feed.observation_timestamp >= observation_timestamp) {
 event::emit(
     StaleReport {
         feed_id,
         latest_timestamp: feed.observation_timestamp,
         report_timestamp: observation_timestamp
     }
 );
};

feed.observation_timestamp = observation_timestamp;
feed.benchmark = benchmark_price;
feed.report = report_data;
```

The logic will just emit the `StaleReport` event and replace the feed with the stale price.

Recommendation: Aave Protocol should directly contact Chainlink to get more details on this odd behavior and when this edge case scenario could happen. At the same time, Aave Protocol should consider adding some kind of sanity check on the staleness of the price returned by Chainlink given that, at least on Aptos, the price could indeed be stale or overridden (same `observation_timestamp`).

Aave Labs: Fixed in PR 356.

Spearbit: Fix verified.

5.3.7 `batch_set_asset_feed_ids` should revert when `assets` and `feed_ids` lengths are not the same

Severity: Low Risk

Context: oracle.move#L69

Description: The `batch_set_asset_feed_ids` is not checking if the length of the two input vectors `assets` and `feed_ids` matches. If `feed_ids` is bigger compared to `assets`, the function will not revert but will not configure all the `feed_ids` to an asset.

Recommendation: Aave Protocol should revert if the length of the two input parameters does not match.

Aave Labs: Fixed in PR 282.

Spearbit: Fix verified.

5.3.8 Collateral flag for liquidator can be overridden by cached user config when self liquidating

Severity: Low Risk

Context: liquidation_logic.move#L556-L563

Description:

1. The `liquidation_logic::liquidation_call` keeps a cached `user_config_map` for the violator throughout the function.
2. In the middle of the function, the `liquidate_a_token` call can set the collateral flag to true of the liquidator (which is also the user when self-liquidating).
3. Afterwards, `burn_bad_debt` → `burn_debt_token` is called which users the **cached** `user` **config** `user_config_map` that does not have the collateral flag set. If `outstanding_debt == 0`, it turns off the borrowing for the reserve and sets the entire user config again - overwriting the collateral flag from `liquidate_a_tokens`.

Recommendation: Re-fetch the user config in the `user_config_map` var after `liquidate_a_token` is called.

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

Aave Labs: Fixed in PR 398.

Spearbit: Fix verified.

5.3.9 `price_cap_stable_adapter` implementation is incompatible with the requirements

Severity: Low Risk

Context: price_cap_stable_adapter.move#L1

Description: The `price_cap_stable_adapter` should mimic the implementation of the `PriceCapAdapterStable` contract.

The scope of such a contract is to cap the price of the USD pegged assets, and it's currently used in the Aave ecosystem when Aave needs to fetch the prices of tokens like `USDC` (and other USD pegged assets).

This is, for example, the asset price source used for `USDC` in the `AaveOracle` contract deployed on Ethereum Mainnet: 0x736bf902680e68989886e9807cd7db4b3e015d3c.

The current implementation and usage of `price_cap_stable_adapter` is incompatible with the requirements and needs and requires a full refactor of the logic and data structure.

1. The `InternalData` struct only supports one `asset` while it should support multiple stable coin assets. Unlike on EVM, where you can deploy multiple instances of the same contract, one for each stable coin, on Move you need to structure the module as a "factory" that can manage and handle multiple assets (like the `a_token_factory`). For each stablecoin `asset`, the module should have a specific `InternalData`.
2. If the `price_cap` should be the same across all the assets managed by `price_cap_stable_adapter` it could be exported from the `InternalData` and added to the "common" `struct` applied across all the assets.

3. `decimals` can be exported too, given that on Aptos, Chainlink always uses `18` decimals for their prices.
4. when `init_price_cap_stable_adaptor` is executed to initialize the Global Storage, the `decimals` are fetched by calling `fungible_asset::decimals(address_to_object<Metadata>(asset))`. The decimals should represent the decimal of the `price` and not of the asset. Given that Chainlink is the only price source used, you could instead call `oracle::get_asset_price_decimals` which returns the correct value.
5. The `PriceCapUpdated` event should include the `asset` for which the `price_cap` has been updated for. This suggestion should be adapted depending on if `price_cap` is common across all the assets or can be specified for each one.
6. Consider implementing a way to remove an `asset` from the `price_cap_stable_adapter` global storage (it will directly fetch the price from Chainlink without any upper bound cap).
7. Add the implementation of all the needed getter functions to retrieve the asset's information stored in the Global Storage.

After the refactoring of the `price_cap_stable_adapter` module, Aave must also refactor the `oracle` and `oracle_base` module to support the usage of `price_cap_stable_adapter`.

The `oracle` module must fetch the price from the `price_cap_stable_adapter` if the `asset` has been configured inside the `price_cap_stable_adapter`.

> Note 1: the severity has been reduced to Low because the client has stated that CAPO will not be deployed to Aave Aptos Implementation at the beginning.

Recommendation: Aave Protocol should fully refactor the `price_cap_stable_adapter` module to support multiple stable coin assets at the same time.

After the refactoring of the module, the `price_cap_stable_adapter` should be integrated inside the `oracle` module's logic. If an asset has been configured inside the `price_cap_stable_adapter`, it should fetch the price (capped) from the `price_cap_stable_adapter` and not directly from Chainlink.

Aave Protocol should be aware that the Aave Oracle on EVM is not only using the `PriceCapAdapterStable` (for stable coins) but multiple other adapters depending on the token type they need to upper cap. The full list of all the CAPO adapters currently deployed and used by Aave on the Ethereum Mainnet has been detailed in the appendix "Aave Oracle and which assets are protected by the CAPO system".

Aave Labs: Fixed in PR 394.

Spearbit: Fix verified.

5.3.10 `ISOLATED_COLLATERAL_SUPPLIER_ROLE` `validate_automatic_use_as_collateral` logic does not work

Severity: Low Risk

Context: validation_logic.move#L502-L504

Description: In Aave's Solidity implementation, the `validation_logic::validate_automatic_use_as_collateral` function checks whether the `msg.sender` has the `ISOLATED_COLLATERAL_SUPPLIER_ROLE`. (As the logic contracts are delegatecalled, the `msg.sender` is for example the `Pool` contract). The purpose of the role is for migration contracts to be able to set collateral flags on behalf of users when dealing with isolated assets.

Recommendation: Consider removing the check and always return `false` for isolated assets in `validate_automatic_use_as_collateral`. The role constant `ISOLATED_COLLATERAL_SUPPLIER_ROLE` can also be removed.

Aave Labs: Fixed in PR 325.

Spearbit: Fix verified.

5.3.11 `update_state` is not updating the `cache`

Severity: Low Risk

Context: pool_logic.move#L43-L55

Description: The `pool_logic::update_state` function, which computes the new indexes, takes a `&mut ReserveCache` but does not update its timestamp. Functions that take a `&mut ReserveCache` should perform the code based on the values in the reserve cache and return an updated reserve cache.

Recommendation:

1. Consider basing the decision of whether to update on the cache's timestamp instead of the latest state's timestamp.
2. Update the reserve cache's latest timestamp field.

It should match Aave's Solidity code `updateState` function:

```
function updateState(
  DataTypes.ReserveData storage reserve,
  DataTypes.ReserveCache memory reserveCache
) internal {
  // If time didn't pass since last stored timestamp, skip state update
  //solium-disable-next-line
  if (reserveCache.reserveLastUpdateTimestamp == uint40(block.timestamp)) {
    return;
  }

  _updateIndexes(reserve, reserveCache);
  _accrueToTreasury(reserve, reserveCache);

  //solium-disable-next-line
  reserve.lastUpdateTimestamp = uint40(block.timestamp);
  reserveCache.reserveLastUpdateTimestamp = uint40(block.timestamp);
}
```

Aave Labs: Fixed in PR 411.

Spearbit: Fix verified.

5.3.12 `default_reserve_interest_rate_strategy::set_reserve_interest_rate_strategy` should be declared as `public(friend)`

Severity: Low Risk

Context: default_reserve_interest_rate_strategy.move#L81, pool_token_logic.move#L105

Description: The current implementation of `set_reserve_interest_rate_strategy` in the `default_reserve_interest_rate_strategy` module declares the function as `public entry` and allows any authed user to initialize or update the interest rate configuration of a reserve (existing or not) without any insurance if the reserve index has been already updated.

The function should be declared as `public(friend)` and only the `pool_configurator` module should be allowed to call it. By only allowing the `update_interest_rate_strategy` and the `init_reserves` to call the `set_reserve_interest_rate_strategy` function, we can ensure that:

1. `update_interest_rate_strategy` can only update the IRS of an existing reserve and the index of such reserve will be updated before the IRS config is changed.
2. `init_reserves` can be called only for non-existing reserves that will be deployed and configured during the process.

Recommendation: Aave Protocol should perform the following changes.

1. Declare the `default_reserve_interest_rate_strategy::set_reserve_interest_rate_strategy` as `public(friend)`.
2. Declare the `pool_configurator` as `default_reserve_interest_rate_strategy` friend.
3. Remove the `only_risk_or_pool_admins` check from the `set_reserve_interest_rate_strategy` function. The users are already authed by the root caller. This is a hard requirement given that the auth roles needed by the root caller are different.
4. Add the IRS config as an input parameter of `init_reserves` for each reserve.
5. Execute `default_reserve_interest_rate_strategy::set_reserve_interest_rate_strategy` before `pool_token_logic::init_reserve`. Theoretically, you could remove `interest_rate_strategy::asset_interest_rate_exists` from the internal `pool_token_logic::init_reserve` logic given that we have just initialized the IRS for such reserve.

Aave Labs: Fixed in PR 355.

Spearbit: Fix verified.

5.3.13 `set_reserve_interest_rate_strategy` should perform additional sanity checks on the input parameters

Severity: Low Risk

Context: default_reserve_interest_rate_strategy.move#L94-L97

Description: The `set_reserve_interest_rate_strategy` logic is currently only performing a sanity check on the `optimal_usage_ratio` input parameter.

Following the logic applied by `DefaultReserveInterestRateStrategyV2`, the `set_reserve_interest_rate_strategy` should also implement the following required sanity checks:

- `optimal_usage_ratio <= MAX_OPTIMAL_POINT`.
- `optimal_usage_ratio >= MIN_OPTIMAL_POINT`.
- `variable_rate_slope1 <= variable_rate_slope2`.
- `base_variable_borrow_rate + variable_rate_slope1 + variable_rate_slope2 <= MAX_BORROW_RATE`.

On solidity, the above `constant` values (used as lower/upper bounds) are defined as follows:

- `uint256 public constant MAX_BORROW_RATE = 1000_00;`.
- `uint256 public constant MIN_OPTIMAL_POINT = 1_00;`.
- `uint256 public constant MAX_OPTIMAL_POINT = 99_00;`.

Note that in Solidity, the above constant values are expressed in **bps** and must be converted into the format used by the Aptos implementation (RAY).

> Also note that there's a difference between Aptos/Solidity in the current allowed value for `optimal_usage_ratio`. On Aptos, it's allowed to have `optimal_usage_ratio <= 1e27` (which is 100%), on Solidity the max value for `optimal_usage_ratio` is `0.99e27` (which is 99%).

Recommendation: Aave Protocol should consider adding the suggested sanity checks when the interest rate strategy is initialized or configured for a reserve.

Aave Labs: Fixed in PR 355.

Spearbit: Fix verified.

5.3.14 `token_base` events do not identify the token used

Severity: Low Risk

Context: token_base.move#L25-L65

Description: The `token_base` module is used by all aTokens and vTokens to perform the base token actions like transfers. The emitted `Transfer`, `Mint`, `Burn` events do not identify the actual tokens these events were emitted for. The events are currently not useful for data processing because one can't identify if the `Transfer` was for aUSDC, aAPT, etc.

Recommendation: Consider adding identifying information for the token. For example, add a token field for the `Object<Metadata>` address of the aToken/vToken.

Aave Labs: Fixed in PR 329.

Spearbit: Fix verified.

5.3.15 AToken/vToken factories functions work with tokens of the opposite type

Severity: Low Risk

Context: variable_debt_token_factory.move#L235

Description: The `token_base` function is used as the shared module between aTokens and vTokens. To get a token's balance one can call the public functions on either the `a_token_factory` or `variable_debt_token_factory` modules. However, not all functions check if the token address parameter is indeed a token of the factory's type, for example:

- A user can call `variable_debt_token_factory::scaled_balance_of(owner, metadata_address=a_token)` with an `a_token` address and receive the aToken balance, and vice versa for aToken factory and vToken parameters.

This should not be valid and in the worst case, this can lead to exploits in integrators that don't perform further checks on the `metadata_address` (like interpreting a vToken address as an aToken collateral balance in a `a_token_factory::scaled_balance(owner, metadata_address=vToken)`).

Recommendation: All functions of `variable_debt_token_factory` and `a_token_factory` that accept a token as a parameter should perform checks that this token is of the factory's type before forwarding the call to `token_base`. This can be done by using the existing `assert_token_exists(metadata_address)` function. Some functions already call `get_token_data` which internally performs this check.

Aave Labs: Fixed in PR 327.

Spearbit: Fix verified.

5.3.16 Wrong overflow check in `percent_div`

Severity: Low Risk

Context: math_utils.move#L127

Description: The canonical overflow prevention check for the `(value * PERCENTAGE_FACTOR + percentage / 2) / percentage` computation should be:

```
assert!(
    value <= (U256_MAX - percentage / 2) / PERCENTAGE_FACTOR,
    error_config::get_eoverflow()
);
```

Recommendation: Consider changing the check to the above-mentioned code.

Aave Labs: Fixed in PR 284.

Spearbit: Fix verified.

5.3.17 Unnecessary checks in `wad_ray_math`

Severity: Low Risk

Context: wad_ray_math.move#L75, wad_ray_math.move#L149

Description: Some checks in wad_ray_math.move‘ are unnecessary or simply always true.

Recommendation: Consider changing the following two functions:

1. The `a == 0` check can be removed as the main computation naturally returns `0` if `a == 0`. (The `b == 0` check is required for the assertion if the custom error wants to be kept.).

```
   public fun wad_mul(a: u256, b: u256): u256 {
-      if (a == 0 || b == 0) {
+      if (b == 0) {
           return 0
       };
       assert!(
           a <= (U256_MAX - HALF_WAD) / b,
           error_config::get_eoverflow()
       );
       (a * b + HALF_WAD) / WAD
   }
```

2. The only way the assertion could be false is if the previous computation had already overflowed, but then the `assert` would never be reached. It is currently always true. The `assert` can be removed. Alternatively, one can perform a different `assert` *before* the computation if one desires custom overflow errors.

```
   public fun wad_to_ray(a: u256): u256 {
+      assert!(a <= U256_MAX / WAD_RAY_RATIO, error_config::get_eoverflow());
       let b = a * WAD_RAY_RATIO;
+      // this is always true at this point, can be removed
-      assert!(
-          b / WAD_RAY_RATIO == a,
-          error_config::get_eoverflow()
-      );
       b
   }
```

Aave Labs: Fixed in PR 286.

Spearbit: Fix verified.

5.3.18 Repaying with the `AToken` does not turn off the use-as-collateral flag when the user use the whole `AToken` balance

Severity: Low Risk

Context: borrow_logic.move#L382-L388

Description: Users can repay their debt by using the corresponding `AToken` instead of the debt's `underlying`. If the user consumes the whole `AToken` balance and the `AToken` was used as collateral, the system should update the user's `using_as_collateral` flag to `false` for such asset.

By not turning it to false, the system creates an inconsistency between the user's balance state and the config's state which could lead to unexpected behaviors given that such a flag is widely used both directly and indirectly across the protocol's logic.

Recommendation: Inside the `use_a_tokens` logic branch, Aave should update the `using_as_collateral` to false if such a flag was set to `true` and the user has no more `AToken` scaled-balance. After updating the user's config, the `ReserveUsedAsCollateralDisabled` event must be emitted.

Aave Labs: Fixed in commit 43d7f32b.

Spearbit: Fix verified.

5.3.19 Liquidation does not turn off the `use-as-collateral` flag when the liquidator seize the whole borrower's collateral

Severity: Low Risk

Context: liquidation_logic.move#L524-L525

Description: During the liquidation process, the liquidator could have seized the whole borrower's collateral. In such case, the protocol must update the borrower's user config and turn off the `use-as-collateral` flag for the `collateral` token. The Aptos codebase is not performing such operation that could lead to unexpected behavior for the borrower.

Recommendation: Aave should check if the liquidator has seized the whole borrower's collateral and update the user's config of the borrower, setting the `use-as-collateal` flag to `false` for the `collateral` token.

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

Aave Labs: Fixed in commit 374b6ce1.

Spearbit: Fix verified.

5.3.20 `validate_flashloan_complex` is not reverting when the list of assets have duplicates

Severity: Low Risk

Context: validation_logic.move#L32-L36

Description: The `validate_flashloan_complex` is missing a sanity check introduced Aave v3.1.0 (see ValidationLogic.sol#L338-L340) that requires the uniqueness of the asset requested to be flashloaned.

This check should be added to the Aptos implementation.

```
for (uint256 j = i + 1; j < assets.length; j++) {
    require(assets[i] != assets[j], Errors.INCONSISTENT_FLASHLOAN_PARAMS);
}
```

The Aptos implementation should introduce the same sanity check and prevent the flashloaner from requesting the flashloan/flash-borrow of the same asset within the same request.

Recommendation: Aave Protocol should revert the complex flashloan operation when the assets in the `assets` input vector are not unique.

Aave Labs: Fixed in PR 301.

Spearbit: Fix verified.

5.3.21 `pool_configurator` module should expose a getter function for the `pending_ltv` value of a reserve

Severity: Low Risk

Context: pool_configurator.move#L260

Description: The `pool_configurator` module defines the `pending_ltv` in the Global Storage as.

```
/// Internal module data
struct InternalData has key {
    /// map between an asset address and its pending ltv
    pending_ltv: SmartTable<address, u256>
}
```

In the Solidity implementation, it's possible to fetch such information for a specific `asset` via the `getPendingLtv(address asset)` function. This feature is not available in the Aptos codebase and should be implemented.

Recommendation: Aave Protocol should implement a getter function to allow the caller to fetch the current value of `pending_ltv` for a specific reserve `asset`.

Aave Labs: Fixed in PR 297.

Spearbit: Fix verified.

5.3.22 `init_reserves` does not support yet the deployment of a reserve with preconfigured `incentives_controllers`

Severity: Low Risk

Context: pool_configurator.move#L325

Description: The current implementation of `pool_configurator::init_reserves` does not include the input parameter `incentives_controllers` which is passed "empty" (`option::none()`) to the `pool_token_logic::init_reserve`. This means that every reserve will always be initialized with an empty incentive controller that needs to be configured later on.

Recommendation: Aave Protocol should add the `incentives_controllers` input parameter to the `pool_configurator::init_reserves` to allow the deployer to deploy and configure a reserve with existing `incentives_controllers`. When the input parameter will be added, it will need to be documented in the natspec documentation function and properly sanity checked to ensure that the value passed conforms to the spec.

Aave Labs: Fixed in PR 295.

Spearbit: Fix verified.

5.3.23 `finalize_transfer` should use the `scaled_amount` and not `amount`

Severity: Low Risk

Context: pool_token_logic.move#L339

Description: The current implementation of `pool_token_logic::finalize_transfer` performs the transfer validation and user config updates if the `amount` is greater than zero.

This logic has been correctly changed in the Aave Solidity Implementation (see SupplyLogic.sol#L191-L193) to not rely on the non-scaled amount but on the scaled one. While `amount` could be greater than zero (the one specified by the user), the transfer operation transfers from an account to another the scaled version of that amount, that could be equal to zero because of rounding down.

To avoid any possible errors and unexpected behaviors, the Aptos implementation should be aligned to the Solidity one.

Recommendation: Aave Protocol should calculate the `scaled_amount` version of `amount` and use that value in the `if (from != to && amount != 0)` logic instead of ‘amount.

Aave Labs: Fixed in PR 390.

Spearbit: Fix verified.

5.3.24 "Same" events are defined multiple times across modules

Severity: Low Risk

Context: borrow_logic.move#L63, isolation_mode_logic.move#L19, liquidation_logic.move#L36, liquidation_logic.move#L45, supply_logic.move#L49, supply_logic.move#L58, pool.move#L39, pool_token_logic.move#L51, pool_token_logic.move#L60, pool_token_logic.move#L71, a_token_factory.move#L65

Description: In Aptos, events with the same names and `struct` fields that are defined in separate modules will be treated as separate events. This makes it hard to query for the event as the event needs to be queried across all modules it is defined in and the results must be merged. The proper approach is to have the event defined in a single module.

Some events that represent a single event in Aave Solidity are defined across several modules in Aave Move:

1. `ReserveUsedAsCollateralEnabled` is defined across `liquidation_logic.move`, `supply_logic.move` and `pool_token_logic.move`.
2. `ReserveUsedAsCollateralDisabled` is defined across `liquidation_logic.move`, `supply_logic.move` and `pool_token_logic.move`.
3. `IsolationModeTotalDebtUpdated` is defined across `borrow_logic.move`, `isolation_mode_logic.move` and `pool.move`.
4. `BalanceTransfer` is defined across `pool_token_logic.move` and `a_token_factory.move`.

Recommendation: Define the event in only a single module (the equivalent to the Solidity one) and trigger the event emission from this single module instead, for example, by calling into a function of this module.

For the `BalanceTransfer` event, consider refactoring the code in `pool_token_logic.move` so it does not directly call into `token_base.move`. The `transfer` function entrypoint could be defined in `a_token_factory` instead and call into `pool_token_logic` just for the `finalize_transfer` checks.

Aave Labs: Fixed in PR 316.

Spearbit: Fix verified.

5.3.25 `token_base` allows A/V token to de-sync the incentive controllers

Severity: Low Risk

Context: token_base.move#L149

Description: When the reserve is deployed, the caller provides only one common (empty or not) incentive controller to be used for both the `AToken` and `VariableDebtToken`. The `set_incentives_controller` allows the caller to set a new (empty or not) `incentives_controller` for the single `AToken` or `VariableDebtToken`. This behavior allows the system to de-sync the tokens incentive controller that should be the same given the assumption used during the reserve's initialization.

With the current logic, we could end up with:

- A/V tokens are configured with different incentive controllers.
- One of the two token have the incentive controller configured and the other not.

Recommendation: Aave Protocol should consider refactoring this logic, enforcing the `pool_admin` to configure both the tokens with the same (empty or not) incentive controller.

Aave Labs: Fixed in PR 328.

Spearbit: Fix verified.

5.3.26 Side effects of deploying the `AToken` and `VariableDebtToken` as `FungibleAsset`

Severity: Low Risk

Context: pool_token_logic.move#L268

Description: In the current implementation of the Aptos codebase both the `AToken` and `VariableDebtToken` are deployed as `FungibleAsset`. When the user mint `AToken` or `VariableDebtToken`, the receiving wallet is immediately frozen (see token_base.move?lines=260,265).

```
// freeze account
if (!fungible_asset::is_frozen(to_wallet)) {
    fungible_asset::set_frozen_flag(transfer_ref, to_wallet, true);
};
```

The same happens when the `AToken` is transferred to another user (see token_base.move?lines=454,451). The amount of `AToken` and `VariableDebtToken` that are minted/burned to the user's wallet and added/removed from the `FungibleAsset` supply are the **scaled** version of the original amount (divided by the current index).

These two facts imply two different and critical side effects that compromise the integration of the Aave Protocol with third-party Wallet, integrators and third-party protocols:

1. The `AToken` and `VariableDebtToken` token supply and balance of the user returned by the `FungibleAsset` is the scaled version that does not include the index. This mean that the value returned will not include the accrued interest. The scaled version of such an amount is meaningless and does not represent the current supply or user's balance.
2. Wallets (and so users), integrators and protocols won't be able to transfer the `AToken` by interacting directly with the `AToken` because the wallet is frozen. Unless Wallets or third-party protocols implement a custom behavior just for Aave to execute the `aave_pool::pool_token_logic::transfer` function, the integration with Aave will be impossible.

Recommendation: The only possible fix for the two above problems is to determine the viability of deploying and configure both the `AToken` and `VariableDebtToken` as `DispatchableFungibleAsset`, implementing all the needed hooks and performing the needed sanity checks upon `AToken` transfer.

Aave Labs: Acknowledged.

Spearbit: Acknowledged.

5.3.27 `validate_set_use_reserve_as_collateral` edge case handling

Severity: Low Risk

Context: validation_logic.move#L153

Description: The current implementation of the `validate_set_use_reserve_as_collateral` will always revert if the `user_balance` is equal to zero. There could be edge case scenarios where the protocol has failed to turn-off automatically (rounding down errors, logic errors and so on) the `use-as-collateral` flag for the `asset` in the user's config and the user would like to manually turn that flag to `false`.

Recommendation: One possible solution, that should have no side effects, would be to allow the user to turn off the `use-as-collateral` flag as an exception when all these conditions are true:

- The flag is currently `true`.
- The user has indeed no balance.
- The `use_as_collateral` input parameter passed to `set_user_use_reserve_as_collateral` is equal to `false`.

Aave Labs: Fixed in PR 404.

Spearbit: Fix verified.

5.4 Gas Optimization

5.4.1 `validate_supply` 's `a_token_address` can be reused

Severity: Gas Optimization

Context: validation_logic.move#L87

Description: The `validation_logic::validate_supply` function computes the AToken address several times.

Recommendation: In Line 87, the `a_token_address` variable from Line 76 can be reused.

```
  let a_token_address = pool_logic::get_a_token_address(reserve_cache);
  // ...

  let a_token_scaled_total_supply =
      a_token_factory::scaled_total_supply(
-         pool_logic::get_a_token_address(reserve_cache)
+         a_token_address
      );
```

Aave Labs: Fixed in PR 320.

Spearbit: Fix verified.

5.4.2 `pow` can be improved

Severity: Gas Optimization

Context: math_utils.move#L134

Description: The `math_utils::pow` function can be improved using exponentiation by squaring. It runs in $\mathcal{O}(\log(n))$ instead of $\mathcal{O}(n)$.

Recommendation: The Move version using bitshifts could look like this (not tested!).

```
public fun pow(base: u256, exponent: u256): u256 {
    let result: u256 = 1;

    while (exponent > 0) {
        if (exponent & 1 == 1) {
            result = result * base;
        }

        base = base * base;
        exponent = exponent >> 1;
    }

    result
}
```

Aave Labs: Fixed in PR 285.

Spearbit: Fix verified.

5.4.3 `generic_logic::calculate_available_borrows` can optimize gas usage

Severity: Gas Optimization

Context: generic_logic.move#L230

Description: When the `available_borrows_in_base_currency` is equal to `total_debt_in_base_currency` the code should early return to avoid computing the subtraction and wasting gas.

Recommendation: Aave should perform the following code change to avoid wasting gas.

```
- if (available_borrows_in_base_currency < total_debt_in_base_currency) {
+ if (available_borrows_in_base_currency <= total_debt_in_base_currency) {
      return 0
  };
```

Aave Labs: Fixed in PR 291.

Spearbit: Fix verified.

5.5 Informational

5.5.1 `emode.ltv` is ignored if `reserve.ltv = 0`

Severity: Informational

Context: generic_logic.move#L148

Description: In `generic_logic::calculate_user_account_data` the reserve's emode's LTV is only taken into account if the reserve's own LTV is non-zero:

```
if (vars.ltv != 0) {
    let ltv = if (vars.is_in_emode_category) {
        vars.emode_ltv
    } else {
        vars.ltv
    };
    vars.avg_ltv = vars.avg_ltv
        + vars.user_balance_in_base_currency * ltv;
} else {
    vars.has_zero_ltv_collateral = true
};
```

However, it can happen that the reserve's emode category's LTV is non-zero while the reserve's is zero, see issue "emode LTV & LT invariants can be broken" (for example, via `pool_configurator::configure_reserve_as_collateral`). The impact is that the user cannot borrow as much as their emode configuration technically allows.

Recommendation: Clarify the intended behavior if the reserve's LTV is zero but its emode category's LTV is non-zero.

Response: The reserve's LTV supersedes. It's intended behaviour.

Aave Labs: Fixed in PR 276.

Spearbit: Fix verified.

5.5.2 `ReserveConfigurationMap.liquidation_grace_period_until` should be removed

Severity: Informational

Context: reserve_config.move#L108, reserve_config.move#L496-L503

Description: The `liquidation_grace_period_until` data is already stored in the `ReserveData.liquidation_grace_period_until` inside the `aave_pool::pool` module.

Recommendation: The one stored in `ReserveConfigurationMap.liquidation_grace_period_until` is never used and should be fully removed alongside the removal of the related getter `public fun get_liquidation_grace_period_until`.

Aave Labs: Fixed in PR 254.

Spearbit: Fix verified.

5.5.3 Documentation and natspec related issues

Severity: Informational

Context: acl_manage.move#L49-L51, error_config.move#L294, reserve_config.move#L20-L21, reserve_config.move#L288, user_config.move#L155, math_utils.move#L63, generic_logic.move#L78, liquidation_logic.move#L785-L788, validation_logic.move#L55-L58, validation_logic.move#L407-L414, validation_logic.move#L489-L495, emission_manager.move#L114, rewards_controller.move#L321, pool_configurator.move#L660, pool_configurator.move#L674-L677, pool_logic.move#L38-L43, pool_logic.move#L166,

pool.move#L213, pool.move#L485, pool.move#L787-L795, pool_token_logic.move#L78-L89, a_token_factory.move#L223, a_token_factory.move#L228, token_base.move#L22, variable_debt_token_factory.move#L52, variable_debt_token_factory.move#L54

Description:

1. math_utils.move?lines=63,63: `calculate_compounded_interest` miss the documentation for the `current_timestamp: u64` input parameter.
2. generic_logic.move?lines=78,78:

   > /// @return True if the ltv is zero, false otherwise.

   This is about `has_zero_ltv_collateral` not about their LTV being zero. Consider updating docs to "True if the user has a zero-LTV asset enabled as collateral". This is also present in the Solidity version.
3. validation_logic.move?lines=407,407: Remove the comment as it was decidedthat price oracle sentinels are not required on Aptos.
4. pool_configurator.move?lines=660,660: Typo: `util` → `until`.
5. pool.move?lines=213,213: Typo: `data_obejct` → `data_object`.
6. a_token_factory.move?lines=223,223: Instead of "tokens locked in this contract", it could say "It transfers out any token that is not the underlying that has been sent to the specified aToken's resource account.".
7. token_base.move?lines=22,22: The `friend aave_pool::user_logic;` can be removed as `user_logic` is not using any of `token_base`'s functions.
8. variable_debt_token_factory.move?lines=52,52: Typo: `a_token_factor` → `a_token_factory`.
9. variable_debt_token_factory.move?lines=54,54: This is a mapping of `underlying token address` ⇔ `VToken address`, not AToken address.
10. acl_manage.move?lines=49,49: Role values should not end with `_ROLE` to keep the style consistent. It should be `"EMISSION_ADMIN"` and `"REWARDS_CONTROLLER_ADMIN"`.
11. reserve_config.move: Inconsistent style: The functions `set_active`, `set_siloed_borrowing`, `set_borrowing_enabled` don't put parenthesis around the bit shifts whereas other functions like `set_borrowable_in_isolation` do.
12. user_config.move?lines=155,155:

    > /// @dev this uses a simple trick - if a number is a power of two (only one bit set) then n & (n - 1) == 0.

    While this is true, the code makes use of the other direction: "$\forall n > 0, n\&(n-1) == 0 \Rightarrow n = 2^k$ (only one bit set = only one collateral set)" because only then do we get the result that exactly 1 bit is set. It's still true in this direction (it's an equivalence), so this only needs a comment update to: "this uses a simple trick - a number is a power of two (only one bit set) if and only if n & (n - 1) == 0".
13. validation_logic.move?lines=55,58: The `validate_supply` function is missing the natspec documentation for the `on_behalf_of` parameter.
14. liquidation_logic.move?lines=785,788: The `burn_debt_tokens` natspec documentation should be fully rewritten because it's based on the old logic/signature.
15. pool_logic.move?lines=38,43: The `reserve_data_mut` natspec parameter name is wrong and should be updated to `reserve_data`. Note: this change must be made across all the functions where the old parameter name is used.
16. pool.move?lines=485,485: The input parameter `util` in the `set_liquidation_grace_period` should be renamed `until`.

17. pool.move?lines=787,795: `cumulate_to_liquidity_index` natspec needs to be updated. `asset` is not passed anymore as a parameter + `reserve_data_mut` is now called `reserve_data` and the natspec description of the parameter is wrong in general.
18. pool_logic.move?lines=166,166: Add that it's also callable by `pool_configurator.move`.
19. a_token_factory.move?lines=228,228: Typo `meadata_address` → `metadata_address`.
20. reserve_config.move?lines=20,21: All the deprecated fields should be documented like in the Solidity version. It improve the readibility and knowledge understanding. See how they have done in the ReserveConfiguration.sol contract.
21. emission_manager.move?lines=114,114 Typo `implemennтation` → `implementation`.
22. rewards_controller.move?lines=321,332: The `TODO` seems outdated. All `handle_action` calls directly happen in `token_base`, there should be no need to do them in `supply_logic.move` and other modules.
23. pool.move?lines=534,534: Typo `reserve_dara` → `reserve_data`.
24. coin_migrator.move?lines=14,14: Typo `CointToFaConvertion` → `CoinToFaConversion`.
25. coin_migrator.move?lines=25,25: Typo `FaToCointConvertion` → `FaToCoinConversion`.
26. validation_logic.move?lines=418,418: Typo `debt_resreve_liquidation_grace_priod_until` → `debt_reserve_liquidation_grace_period_until`.

Multiple modules and functions are completely missing the natspec documentation.

Recommendation: Aave Protocol should fix all the suggestions listed above. Every function should be covered by a complete natspec documentation.

5.5.4 Consider migrating the `AaveProtocolDataProvider` contract to the Aptos codebase

Severity: Informational

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

Description: The Aptos codebase is missing the implementation of the `AaveProtocolDataProvider` contract that offers utility getters that are used by both the Aave Core Solidity implementation and by probably the Aave dApps.

Recommendation: Aave Protocol should consider if the getters offered by the `AaveProtocolDataProvider` are useful to be implemented also in the Aptos context.

Aave: Acknowledged.

Spearbit: Acknowledged.

5.5.5 Consider merging `oracle` and `oracle_base` modules

Severity: Informational

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

Description: The `oracle_base` module offers multiple `public(friend)` functions that are declared `friend` only to be called directly by the `oracle` module. Merging those modules and declaring those functions as `private` would make the code easier to read without the friction to have to just between one module to another.

Recommendation: Consider merging the `oracle_base` and `oracle` modules.

Aave Labs: Fixed in PR 278.

Spearbit: Fix verified.

5.5.6 Consolidate all the error declarations in the existing `aave_config::error_config` module

Severity: Informational

Context: oracle_base.move#L22-L26, price_cap_stable_adapter.move#L12-L14

Description: The `aave_config::error_config` module is the module that consolidates all the errors used within the Aave. Any error declared in other modules should be moved into the `aave_config::error_config` module and referenced via a `public fun` getter function.

Recommendation: Migrate the errors from existing modules into the common `aave_config::error_config` module and reference them via an ad-hoc getter function.

Aave Labs: Fixed in PR 365.

Spearbit: Fix verified.

5.5.7 `oracle` and `oracle_base` refactoring suggestions

Severity: Informational

Context: oracle_base.move#L113, oracle_base.move#L114, oracle_base.move#L160, oracle_base.move#L162, oracle.move#L62, oracle.move#L73, oracle.move#L81, oracle.move#L92, oracle.move#L129

Description:

oracle_base.move?lines=113,113: `get_oracle_resource_account` and `oracle_address` functions return the same information. Remove one of them and replace all the other references with the other.

oracle_base.move?lines=114,114: Rename `collector_data` to `oracle_data`.

oracle_base.move?lines=160,160: The `public(friend) fun get_feed_id` function is declared as `friend` but it's only used by the `test_get_feed_id` test function. Remove it and use the code directly into the test function, or simply declare it as a `public` view function that could be accessed externally if there's such need.

oracle_base.move?lines=162,162: If `asset` is not defined in `asset_price_list.asset_feed_ids` the `get_feed_id` function will revert with a "native" error. Consider using `assert_asset_feed_id_exists` or removing the whole function at this point.

- ☐ oracle.move?lines=62,62: `oracle_base::emit_asset_price_feed_updated` should automatically be called inside `oracle_base::set_asset_feed_id` and not manually by `oracle::set_asset_feed_id`.
- ☐ oracle.move?lines=73,73: `oracle_base::emit_asset_price_feed_updated` should automatically be called inside `oracle_base::set_asset_feed_id` and not manually by `oracle::batch_set_asset_feed_ids`.
- ☐ oracle.move?lines=81,81: `oracle_base::emit_asset_price_feed_removed` should automatically be called inside `oracle_base::remove_feed_id` and not manually by `oracle::remove_asset_feed_id`.
- ☐ oracle.move?lines=92,92: `oracle_base::emit_asset_price_feed_removed` should automatically be called inside `oracle_base::remove_feed_id` and not manually by `oracle::batch_remove_asset_feed_ids`.

oracle.move?lines=129,129: The value returned by `get_asset_price_decimals` should be declared as a specific `const` variable and not as a "magic number". Consider also adding a natspec documentation, given the crucial information represented by such constant.

Recommendation: Aave should implement all the suggestions listed above.

Aave Labs: Fixed in PR 363.

Spearbit: Fix verified. Suggestions have been implemented on the marked elements.

5.5.8 `ComplexFlashLoansReceipt.interest_rate_mode` does not need to be an `Option`

Severity: Informational

Context: flashloan_logic.move#L81

Description: The `flashloan_logic::ComplexFlashLoansReceipt` struct's `interest_rate_mode` field is currently defined as a `Option<u8>` but notice that it's never created with `None`. The available interest rate modes are always either `0` for normal flashloan or `2` for variable borrow flashloan.

Recommendation: Consider defining `ComplexFlashLoansReceipt.interest_rate_mode` as a `u8`.

Aave Labs: Fixed in PR 283.

Spearbit: Fix verified.

5.5.9 Proper `drop_reserve` clean up

Severity: Informational

Context: pool_configurator.move#L339

Description: The `pool_configurator::drop_reserve` function completely removes a reserve from the protocol. Ideally, all state associated with the removed reserve would be removed. The following calls are done as part of `drop_reserve`:

```
pool_configurator::drop_reserve
  pool_token_logic::drop_reserve(asset) // checks aToken/vToken.totalSupply is 0
    a_token_factory::drop_token(a_token_address)
      token_base::drop_token(metadata_address)
    variable_debt_token_factory::drop_token(variable_debt_token_address)
      token_base::drop_token(metadata_address)
```

Recommendation: Consider the following additions:

1. `pool_configurator::drop_reserve`: Remove any pending LTV for the reserve in `InternalData.pending_ltv` if it exists.
2. Optional: `a_token_factory::drop_token(..)` and `variable_debt_token_factory::drop_token(..)`: Remove the `TokenData` struct from the objects via `move_from` as they act as an aToken/vToken marker.
3. Optional: `token_base::drop_token(..)`: Remove the `TokenBaseState` and `ManagedFungibleAsset` struct from the token `metadata_address` (total supply and therefore balances are guaranteed to be zero).

Before dropping a reserve give users enough time to claim rewards to be on the safe side that dropping it does not interfere with rewards claiming (which at the moment reads total supply and user balances directly from `fungible_asset` and `primary_fungible_stores`).

Aave Labs: Fixed in PR 366.

Spearbit: Fix verified.

5.5.10 The `ReserveInterestRateStrategy` event should be aligned to the one used by Solidity

Severity: Informational

Context: default_reserve_interest_rate_strategy.move#L19

Description: The current `ReserveInterestRateStrategy` event emitted when the IRS is initialized or updated has a different name compared to the one used by the Solidity counterpart.

```
/**
 * @notice emitted when new interest rate data is set in a reserve
 *
 * @param reserve address of the reserve that has new interest rate data set
 * @param optimalUsageRatio The optimal usage ratio, in bps
 * @param baseVariableBorrowRate The base variable borrow rate, in bps
 * @param variableRateSlope1 The slope of the variable interest curve, before hitting the optimal ratio,
 ↪  in bps
 * @param variableRateSlope2 The slope of the variable interest curve, after hitting the optimal ratio,
 ↪  in bps
 */
event RateDataUpdate(
    address indexed reserve,
    uint256 optimalUsageRatio,
    uint256 baseVariableBorrowRate,
    uint256 variableRateSlope1,
    uint256 variableRateSlope2
);
```

Recommendation: Aave Protocol should align the event to the one used by Solidity and defined in IDefaultInterestRateStrategyV2.sol#L47-L62.

Aave Labs: Fixed in PR 288.

Spearbit: Fix verified.

5.5.11 `token_address(owner, symbol)` function requires knowing the owner

Severity: Informational

Context: a_token_factory.move#L583

Description: The `a_token_factory::token_address(owner: address, symbol: String)` function requires knowing the `owner` that created the token. Note that the owner can be any address with the pool admin or asset listing role. Users and integrators might only know the token symbol but not its creator. Furthermore, no two listed assets should have the exact same symbol name, this can be assured by the owners.

Recommendation: Consider removing the `owner` parameter from the function and simply search for the `symbol`, reverting if the symbol exists twice (which is already currently done). The same issue applies to the `variable_debt_token_factory`.

Aave Labs: Fixed in PR 340.

Spearbit: Fix verified.

5.5.12 Replace `abort` with the already adopted `assert!` standard

Severity: Informational

Context: emission_manager.move#L104

Description: In the Aave Periphery codebase, some required statements have been implemented using the `abort` pattern instead of using the widely adopted `assert!` pattern. To be consistent with the vast majority of the codebase, all the `abort` instances should be replaced with the `assert!` one.

Recommendation: Aave Protocol should replace all the `abort` reverts with the `assert!` one.

Aave: Fixed in PR 333.

Spearbit: Fix verified.

5.5.13 Redundant `caller == on_behalf_of` could be added to vToken's `mint`

Severity: Informational

Context: variable_debt_token_factory.move#L161-L163

Description: The vToken's `mint` function in Aave Solidity allows for borrows on behalf of another user if the user has given its allowance to the spender. This functionality has been removed in Aave Move but the APIs stayed the same and `mint` is still called with `caller` and `on_behalf_of`.

However, the function does not perform any allowance check right now. This is currently not a security issue as all callers check that `caller == on_behalf_of`.

Recommendation: Consider adding a redundant check by asserting `caller == on_behalf_of` in this non-public `variable_debt_token_factory::mint` function as long as `borrow_on_behalf` is not supported. This makes it easier to reason about the correctness of this function in the absence of any allowance checks.

Aave Labs: Fixed in PR 290.

Spearbit: Fix verified.

5.5.14 `DEFAULT_ADMIN_ROLE` could have a proper value

Severity: Informational

Context: acl_manage.move#L41

Description: The `DEFAULT_ADMIN_ROLE` is set to the empty string which might be confusing if the role is queried and displayed off-chain. This is likely an artifact of mirroring the OpenZeppelin Solidity `AccessControl` version which Aave Solidity uses.

Note that this is done in Solidity because uninitialized `structs` (like `RoleData` from the `_roles` mapping) are default-initialized to the empty bytes, and the library wants to default to the `DEFAULT_ADMIN_ROLE`. However, in Move, `structs` (like `RoleData`) are always explicitly created anyway:

```
if (!smart_table::contains(&role_res.acl_instance, role)) {
    let members = acl::empty();
    acl::add(&mut members, user);
    let role_data = RoleData { members, admin_role: default_admin_role() };
    smart_table::add(&mut role_res.acl_instance, role, role_data);
}
```

Recommendation: Consider setting a proper value for `DEFAULT_ADMIN_ROLE`:

```
- const DEFAULT_ADMIN_ROLE: vector<u8> = b"";
+ const DEFAULT_ADMIN_ROLE: vector<u8> = b"DEFAULT_ADMIN";
```

> Note: This diverges from the Solidity version and any scripts that use a hardcoded `DEFAULT_ADMIN_ROLE` of `""` need to be adjusted - if they exist and are ported to Move. However, they need to be adjusted for all roles anyway as the other Solidity roles are the hashes of the Move role values.

Aave Labs: Fixed in PR 287.

Spearbit: Fix verified.

5.5.15 Integrators can't read any information from the `receipt` returned by the flashloan functions

Severity: Informational

Context: flashloan_logic.move#L57-L83

Description: When an integrator executes `flash_loan` or `flash_loan_simple`, the function will return a receipt object that contains all the information needed by the `receiver` of the flashloan to be able to repay the flashloan

itself. With the current implementation, the receiver won't be able to access this information because the module does not provide any `public` function marked as `#[view]` that returns the struct values.

Recommendation: Aave should implement all the view functions needed by the integrators to access the fields of the `receipt` struct returned by the `flash_loan` or `flash_loan_simple` functions.

Aave Labs: Fixed in PR 314.

Spearbit: Fix verified.

5.5.16 Function visibility and `view` modifiers

Severity: Informational

Context: oracle_base.move#L51, oracle.move#L96, token_base.move#L142

Description: Some functions are declared as `public` but the public is never expected to call them, their visibility should be restricted more. The reason is usually because some Move or TypeScript tests call them. In addition, there is no consistency regarding when getter functions are to be defined as `view`.

Recommendation: Consider the following rules and apply them across the code base:

1. If a function only needs to be called from Move tests, consider adding the test module as a friend and define the visibility as `public(friend)` and the function as `#[test_only]`.

```
#[test_only]
friend aave_oracle::oracle_base_tests;

#[test_only]
public(friend) fun only_oracle_admin(account: &signer) {
    assert!(signer::address_of(account) == @aave_oracle, E_ORACLE_NOT_ADMIN);
}
```

2. If a function only needs to be called from TypeScript tests, consider routing them through a test wrapper module (for example, `test_wrappers.move`) that implements a wrapper function for the function that should be called. The wrapper function just forwards it to the actual module::function that should be called. Now, the function can be declared as `public(friend)` and the function as `#[test_only]`. The `test_wrappers.move` can be friended.

```
#[test_only]
friend testing::test_wrappers;

#[test_only]
public(friend) entry fun set_chainlink_mock_price(
    account: &signer, price: u256, feed_id: vector<u8>
) {
    oracle_base::only_risk_or_pool_admin(account);

    // set the price on chainlink
    let feed_timestamp = (timestamp::now_seconds() * 1000) as u256;
    chainlink::perform_update_for_test(
        feed_id,
        feed_timestamp,
        price,
        vector::empty<u8>()
    );
}
```

> Compile the contracts in "test" mode via `aptos move test` as they need to be deployed to a local testnet with the `test_only` functions for TypeScript tests to access them.

3. Come up with a strategy for when to define a function as `view` and apply it consistently across the code base. For example:
    1. All `public` functions that don't modify state *and are expected to be called from off-chain* should have the `#[view]` attribute.
    2. Functions declared as `public(friend)` or `private` should not have the `#[view]` attribute.
    3. If a function is defined as `view` in Aave's Solidity codebase, the corresponding Move function (if it exists) should also have the `view` attribute.
    4. Functions with the `#[view]` attribute should never acquire a mutable reference, always use `borrow_global` instead of `borrow_global_mut` to avoid state side effects.

Aave Labs: Fixed in PR 302.

Spearbit: Fix verified.

5.5.17 Build and apply a recommended style guide

Severity: Informational

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

Description: Unlike for the Solidity language, the Aptos/Move seems to not have a well-defined "style guide". Aave Protocol should build one and apply it across the whole codebase. By doing so it will make the development, read and review of the code much more easy and consistent.

Some practical example that can also be extrapolated from the Solidity Style Guide:

- Move all the `#[test_only]` functions at the bottom of the file.
- Order all the function declaration in a specific order: `public` → `public(friend)` → `private` → `#[test_only]`.
- Public functions can be sorted again in `#[view]` → non-view (modify the state) functions.
- And so on...

Recommendation: Aave Protocol should build and apply across the whole codebase a programming style guide.

Aave: Acknowledged.

Spearbit: Acknowledged.

5.5.18 Unnecessary struct abilities

Severity: Informational

Context: default_reserve_interest_rate_strategy.move#L31, gho_interest_rate_strategy.move#L33

Description: Some abilities of `struct`s are not necessary and could be removed.

1. default_reserve_interest_rate_strategy.move?lines=31,31: `key` can be removed.
2. gho_interest_rate_strategy.move?lines=33,33: `key` can be removed.

Recommendation: Consider removing unnecessary attributes.

Aave Labs: Fixed in PR 302.

Spearbit: Fix verified.

5.5.19 `get_overall_borrow_rate` can be simplified

Severity: Informational

Context: default_reserve_interest_rate_strategy.move#L286

Description: The `default_reserve_interest_rate_strategy::get_overall_borrow_rate` function was historically used to compute the weighted average borrow rate of variable loans and stable loans. However, as stable loans have been deprecated, this just returns the variable borrow rate.

Recommendation: Consider simplifying this function to:

```
  fun get_overall_borrow_rate(
      total_variable_debt: u256, current_variable_borrow_rate: u256
  ): u256 {
      let totalDebt = total_variable_debt;
      if (totalDebt == 0) return 0;

-      let weighted_variable_rate =
-          wad_ray_math::ray_mul(
-              wad_to_ray(total_variable_debt),
-              current_variable_borrow_rate
-          );
-
-      let overall_borrow_rate =
-          wad_ray_math::ray_div((weighted_variable_rate), wad_to_ray(totalDebt));
-
-      overall_borrow_rate
+      current_variable_borrow_rate
  }
```

Aave Labs: Fixed in PR 355.

Spearbit: Fix verified.

5.5.20 The `pool_addresses_provider` module should be fully removed

Severity: Informational

Context: pool_addresses_provider.move#L1

Description: The `aave_pool::pool_addresses_provider` module is currently only used by the liquidation module to fetch the `Umbrella` address to auth the `eliminate_reserve_deficit` operation. The Umbrella has not been yet ported to the Aptos codebase and will be deployed as a module executed by a single known address (likely `@umbrella`) following the already adopted approach already used across other modules.

Recommendation: Aave Protocol should remove the `pool_addresses_provider` module and evaluate the need only after the full implementation of the Umbrella system in Aptos.

Aave Labs: Fixed in PR 364.

Spearbit: Fix verified.

5.5.21 Category ids parameters for the `EModeAssetCategoryChanged` event should be defined as `u8`

Severity: Informational

Context: pool_configurator.move#L178-L179

Description: The `EModeAssetCategoryChanged` event is defined as follows:

```
#[event]
/// @dev Emitted when the category of an asset in eMode is changed.
/// @param asset The address of the underlying asset of the reserve
/// @param old_category_id The old eMode asset category
/// @param new_category_id The new eMode asset category
struct EModeAssetCategoryChanged has store, drop {
    asset: address,
    old_category_id: u256,
    new_category_id: u256
}
```

On Solidity, the `category_id` of an eMode category is a `uint8`.

Recommendation: Aave Protocol should refactor the `EModeAssetCategoryChanged` event defining both `old_category_id` and `new_category_id` as `u8` type.

Aave Labs: Fixed in PR 292.

Spearbit: Fix verified.

5.5.22 `sync_indexes_state` and `sync_rates_state` should be declared as private

Severity: Informational

Context: pool_configurator.move#L375, pool_configurator.move#L389

Description: The `sync_indexes_state` and `sync_rates_state` functions of the `pool_configurator` module can and should only be called by the `set_reserve_factor` and `update_interest_rate_strategy` functions and should not be declared as `public`.

Recommendation: Aave Protocol should change the visibility of the `sync_indexes_state` and `sync_rates_state` functions from `public` to private.

Aave Labs: Fixed in PR 293.

Spearbit: Fix verified.

5.5.23 Rename the `reserve` parameter to `reserve_data` in all the `pool` functions to be consistent

Severity: Informational

Context: pool.move#L357

Description: Other functions across the codebase name the `Object<ReserveData>` type parameter as `reserve_data`. Plenty of functions inside the `pool` modules are still referring to it as `reserve`.

Recommendation: Aave Protocol should consider renaming the input parameter `reserve` of type `Object<ReserveData>` to `reserve_data` to be consistent with the already adopted best practice. The relative natspec documentation should also be updated accordingly.

Aave Labs: Fixed in PR 315.

Spearbit: Fix verified.

5.5.24 The `Initialized` event is missing the `incentives_controller` input

Severity: Informational

Context: a_token_factory.move#L51, variable_debt_token_factory.move#L41

Description: The `Initialized` event in both `a_token_factory` and `variable_debt_token_factory` is missing the `incentives_controller` address input. .

Recommendation: Add the `incentives_controller` to the `Initialized` events.

Aave Labs: Fixed in PR 317.

Spearbit: Fix verified.

5.5.25 Tokens cannot be rescued anymore after a reserve has been dropped

Severity: Informational

Context: a_token_factory.move#L240-L243

Description: After a reserve has been dropped by executing the `a_token_factory::drop_token` (part of drop reserve flow), the `pool_admin` won't be able to call the `rescue_tokens` function anymore and all those tokens owned by the `a_token_resource_account` won't be able to be rescued anymore.

Recommendation: We don't see an easy way to solve this side effect of dropping a reserve. Aave Protocol should be aware of this problem and ensure that all the tokens owned by the `a_token_resource_account` of the reserve have been rescued before proceeding with the drop of the reserve.

Aave Labs: Acknowledged.

Spearbit: Acknowledged.

5.5.26 `AToken` tokens can't be rescued via `rescue_tokens`

Severity: Informational

Context: a_token_factory.move#L246-L251

Description: The current implementation of the Aave Protocol allows users to transfer the `AToken` tokens only via the `aave_pool::pool_token_logic::transfer` function. Calling `rescue_tokens` for an `AToken` will revert because the tokens are frozen in the wallet.

Recommendation: The only current solution (unless the `AToken` is implemented as a Dispatchable Fungible Asset) is to make a custom logic that calls `aave_pool::pool_token_logic::transfer` if the token to be rescued is an `AToken`.

Aave Labs: Acknowledged.

Spearbit: Acknowledged.

5.5.27 Consider removing the `scaled_total_supply` and `user_state[user].balance` from `token_base`

Severity: Informational

Context: token_base.move#L82

Description: The current implementation of the `token_base` stores the scaled total supply of the A/V token in the `scaled_total_supply` attribute of the `TokenBaseState` struct, and the `user_state[user].balance` user balance in the `UserState` struct. These values are already accounted and managed by the `fungible_asset` itself, created for every deployed A/V token.

Recommendation: To avoid any redundant data and make the code and logic more clear and clean, Aave Protocol should consider removing those two attributes from the `token_base` data structure and only rely on the values returned by the `fungible_asset` module.

Aave Labs: Acknowledged.

Spearbit: Acknowledged.

5.5.28 Redundant Data Structure & iterations

Severity: Informational

Context: rewards_controller.move#L662-L685

Description: The current implementation introduces unnecessary computational and storage overhead by iterating over both the `rewards_list` and the assets vectors. Since the assets table already contains the required data, maintaining a separate `rewards_list` is redundant. This duplication leads to increased gas costs, additional storage usage, and added complexity in the codebase.

Remediation: The code should be refactored to remove the redundant data structures `rewards_list: vector<address>` and `assets_list: vector<address>`. Instead, the implementation should use the `assets` table directly. This refactoring will simplify the codebase, reduce storage requirements, improve gas efficiency, and eliminate redundant data maintenance. The direct use of the assets table will streamline the process and make the code more maintainable.

Aave: Fixed in PR 402.

Spearbit: Fix verified.

5.5.29 The `transfer` function should correctly handle the case where `sender == recipient`

Severity: Informational

Context: token_base.move#L419-L424

Description: The Aave Protocol allows the user to transfer the `AToken` to himself, but the current logic of `token_base::transfer` is not correctly handling this scenario. The `sender` data is loaded, updated and saved **before** the one relative to the `recipient` is loaded. For that reason, the value calculated for `recipient_balance_increase` won't be correct.

This is not a security issue because such value is only used to emit the `Transfer` and `Mint` events that are emitted only when `sender != recipient`.

Recommendation: Even if the current logic does not introduce a security issue and the `recipient_balance_increase` is not used when `sender == receiver`, for the sake of correctness, Aave Protocol should consider refactoring the `transfer` logic to fix the issue.

Aave Labs: Acknowledged.

Spearbit: Acknowledged.

5.5.30 `SimpleFlashLoansReceipt` struct contains unnecessary fields

Severity: Informational

Context: flashloan_logic.move#L57

Description: The `flashloan_logic::SimpleFlashLoansReceipt` struct's `interest_rate_mode` field is currently always set to `option::none()` as simple flashloans cannot be repaid by opening a borrow position, they must always be repaid with transfers. In addition, the index `i` is not needed as simple flashloans only return a single flashloan per call.

Recommendation: Consider removing the `interest_rate_mode` and `i: u256` fields from `SimpleFlashLoansReceipt`.

Aave Labs: Fixed in PR 303.

Spearbit: Fix verified.

5.5.31 Remove all the outdated, deprecated or commented code

Severity: Informational

Context: liquidation_logic.move#L736-L783, liquidation_logic.move#L885-L977

Description: In the `liquidation_logic` contract there are some functions (old codebase) that has been fully commented out because they have been deprecated by the new liquidation logic.

Recommendation: Aave Protocol should take care to remove all these functions from the codebase.

Aave Labs: Fixed in PR 306.

Spearbit: Fix verified.

5.5.32 High liquidation bonus leads to profitable self-liquidation oracle update sandwiches

Severity: Informational

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

Description: An attacker can perform the following attack by sandwiching a price oracle update:

1. Taking on a *leveraged* position (by flashloaning collateral and max-borrowing the debt token).
2. Letting the price update be performed.
3. Liquidating themself.

> Note that the attack does not need to be done atomically in a single transaction or block. A leveraged position can be built up without a flashloan by supplying collateral, max borrowing against it, swapping the debt token to collateral, repeat. This increases the risk of the attack but does not fundamentally change it.

Profitability: The attack is profitable when the entire collateral balance is seized (to repay the flashloan) while repaying fewer debt assets than assets borrowed. This difference of `maxBorrowAssets - maxRepayAssets` of debt assets is the profit:

```
**collateralPrice_0 = price before the oracle update:**

# collateralPrice_1 = price after the oracle update
collateralPrice_1 = collateralPrice_0 * (1 - priceDrop)

**the maximum debt asset we can borrow is:** maxBorrowAssets = LTV * collateralBalance *
↪  collateralPrice_0 / debtPrice

**from the liquidation code we see that to seize all collateral we require:**

# vars.debt_amount_needed = math_utils::percent_div(
**((vars.collateral_asset_price * vars.collateral_amount:**

#          * debt_asset_unit)
**/ (debt_asset_price * collateral_asset_unit)),:**

#     liquidation_bonus
**);:** maxRepayAssets
= (collateralPrice_1 * collateralBalance / debtPrice) / liquidationBonus

**profitable if this inequality holds:** maxBorrowAssets > maxRepayAssets
LTV * collateralBalance * collateralPrice_0 / debtPrice
> (collateralPrice_1 * collateralBalance / debtPrice) / liquidationBonus
<=> LTV > (1 - priceDrop) / liquidationBonus

**expressed as the max liquidation bonus::** liquidationBonus < (1 - priceDrop) / LTV
```

Example: LTV = 90%, LT = 95%. Oracle quotes 1 collateral at $1 (and debt is fixed at $1). Sandwich collateral oracle price update to $0.95 (5% drop).

Imagine the liquidation bonus is set too high at `110%` instead of the max `105.55%`. (And imagine we can always full liquidate because the individual account positions are less than the `MIN_LEFTOVER_BASE` threshold.).

1. Flashloan 1000 collateral and build a position of (1000 collateral, 900 debt) at LTV.
2. Oracle sets the collateral price to $0.95.

3. Liquidate self by repaying `maxRepayAssets = (1000 * 0.95$ / 1$) / 110% = 863.63`.
4. Profit `900 - 863.63 = 36.37` debt tokens.
5. The attacker's profit is the protocol's bad debt (`deficit`).

Recommendation: Proper risk management needs to be done and choosing the liquidation bonus should depend on the LTVs and expected volatility of the assets.

> Note: A similar calculation for bounds on the liquidation bonus & LTV can be done by looking at *price increases of the debt token* instead of price decreases of the collateral token.

Aave Labs: Acknowledged.

Spearbit: Acknowledged.

5.5.33 Remove everything related to Stable Debt and the Rebalancing feature

Severity: Informational

Context: error_config.move#L504-L506

Description: The stable debt and the feature to rebalance it has been deprecated in Aave Solidity Implementation and has never been implemented in Aave Aptos Implementation. Everything related to `EINTEREST_RATE_REBALANCE_CONDITIONS_NOT_MET` should be removed by the codebase.

Recommendation: Remove everything related to the stable debt and the rebalance feature.

Aave Labs: Fixed in PR 319.

Spearbit: Fix verified.

5.5.34 Build and apply a recommended style guide

Severity: Informational

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

Description: Unlike for the Solidity language, the Aptos/Move seems to not have a well-defined "style guide". Aave Protocol should build one and apply it across the whole codebase. By doing so it will make the development, read and review of the code much more easy and consistent.

Some practical example that can also be extrapolated from the Solidity Style Guide:

- Move all the `#[test_only]` functions at the bottom of the file.
- Order all the function declaration in a specific order: `public` → `public(friend)` → `private` → `#[test_only]`.
- Public functions can be sorted again in `#[view]` → non-view (modify the state) functions.
- And so on...

Recommendation: Aave Protocol should build and apply across the whole codebase a programming style guide.

Aave Labs: Acknowledged.

Spearbit: Acknowledged.

5.5.35 Aave Oracle is assuming that all the all the prices will be directly returned and supported by Chainlink

Severity: Informational

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

Description: The appendix "Aave Oracle and which assets are protected by the CAPO system" and "Aave Reward System on Ethereum Mainnet" have shown two important information:

1. The source of some asset's prices won't directly use Chainlink but will pass through a middle layer capped Aave CAPO to cap the max value of a price.
2. The reward of an asset on the Aave Reward system could be an `AToken` itself and not a "normal" token.

Aave CAPO: If Aave Protocol plans to support the Aave CAPO system also on Aptos, the `oracle` module needs to implement some kind of "routing" system that will query Chainlink if the price can be directly fetched from Chainlink or would call the CAPO module middleware otherwise.

`AToken` **as rewards:** We assume that somewhere in the Aave UI, the dApp will need to show the monetary value of the user's accrued rewards. This means that the `oracle` module must be called and must return the price of the reward asset. The current implementation of the `oracle` module assume that all the assets will use Chainlink as the price source and that the asset itself will be supported by Chainlink.

If Aave on Aptos follows the same behavior as Aave EVM, users could receive `AToken` as part of their rewards, meaning that Chainlink will need to support `AToken` as an asset. As far as we are aware, at least for the EVM chains, Chainlink is not supporting `AToken` and it's highly probably that they won't support them also on Aptos.

There are two possible solutions at this point:

1. Chainlink add support to all the `AToken` deployed by Aave (quite improbable).
2. Aave needs to re-architecture the `oracle` module to return the price of the underlying of the `AToken` when the price for the `AToken` is requested.

Recommendation: Aave should be aware of the limitations and incompatibilities of the current design of the `oracle` module in case of the need to adopt the Aave CAPO system and use the `AToken` as a user's reward.

If they plan to use Aave CAPO or `AToken` as rewards, the `oracle` module must be refactored.

Aave Labs: Acknowledged. We will definitely take this into account, but as mentioned, there won't be any AToken rewards as of now (not planned).

Spearbit: Acknowledged.

5.5.36 `base_currency` refactoring and simplification

Severity: Informational

Context: oracle_base.move#L87, oracle_base.move#L92-L110, ui_pool_data_provider_v3.move#L45-L46, ui_pool_data_provider_v3.move#L275-L284

Description: From the information we were able to gather, we can make this assumption: the `BASE_CURRENCY` information on Aave Oracle is used to represent the `USD` currency with `8` decimals.

On Solidity the `BASE_CURRENCY` on Aave Oracle is represented as an `immutable` state variable that can be configured during the Aave Oracle deployment but has been consolidated, as mentioned above, as the 1 unit of USD with 8 decimals.

Given this assumption, we can simplify and refactor the Aptos codebase everywhere the `base_currency` is used.

- The `base_currency: Option<BaseCurrency>` attribute from the `PriceOracleData` struct can inside the `oracle_base` module can be removed with all the functions and usage inside the module itself.
- `ui_pool_data_provider_v3` can be refactored with these changes:

```
- let opt_base_currency = oracle_base::get_oracle_base_currency();
- let (market_reference_currency_unit, market_reference_currency_price_in_usd) =
-     if (option::is_some(&opt_base_currency)) {
-         let unit =
-             oracle_base::get_base_currency_unit(
-                 &*option::borrow(&opt_base_currency)
-             );
-         ((unit as u256), (unit as u256))
-     } else {
-         (APT_CURRENCY_UNIT, oracle::get_asset_price(apt_mapped_fa_asset))
-     };
+ let market_reference_currency_unit = USD_CURRENCY_UNIT;
+ let market_reference_currency_price_in_usd = USD_CURRENCY_UNIT;
```

With `USD_CURRENCY_UNIT` defined as `const USD_CURRENCY_UNIT: u256 = 1_000_000_000_000_000_000;` given that Chainlink by default uses `18` decimals for their prices. Both the `EMPTY_ADDRESS` and `APT_CURRENCY_UNIT` constant variables can be **removed**.

> **Note:** it's still unclear if other modules inside the Aave Protocol (that we are not aware of), integrators modules or dApps need the "direct" access to the `base_currency` information from the Aave Oracle. If that's the case, Aave Protocol will need to provide direct access to it or simply ask them to hardcode it as a constant given the above assumptions.

Recommendation: If the assumption made are valid, Aave Protocol should refactor and simplify all the code and logic that is based on the `base_currency` concept as explained above.

Aave Labs: Fixed in PR 338.

Spearbit: Fix verified.

5.5.37 Chainlink feeds are publicly accessible through Aave oracle wrapper

Severity: Informational

Context: oracle.move#L24-L29

Description: Note that the `oracle::get_asset_price` function calls Chainlink router's `router::get_benchmarks` function. This function requires a `signer authority` and `billing_data` parameters.

While these parameters are currently not used by Chainlink their existence indicates that this might change in the future and either access checks on the `authority` are implemented or payments are required:

```
public fun get_benchmarks(
    _authority: &signer, feed_ids: vector<vector<u8>>, _billing_data: vector<u8>
): vector<Benchmark> acquires Router {
    let _router = borrow_global<Router>(get_state_addr());

    registry::get_benchmarks_unchecked(feed_ids)
}
```

Aave's `oracle::get_asset_price` function is a public wrapper around the Chainlink data feeds that anyone can call. If billing or access checks are required, other users could use Chainlink on behalf of Aave, leading to either additional cost to Aave or circumventing Chainlink's access restrictions.

Recommendation: Clarify how Chainlink will use the `authority` and `billing_data` parameters on Aptos in the future and if access to Aave oracles needs to be restricted to the Aave protocol or can remain publicly available.

Aave Labs: Acknowledged.

Spearbit: Acknowledged.