Aave Aptos V3

Security Assessment

August 8th, 2025 — Prepared by OtterSec

Bartłomiej Wierzbiński dark@osec.io

Andreas Mantzoutas andreas@osec.io

Robert Chen r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Aave engaged OtterSec to assess the `aave-aptos-v3` program. This assessment was conducted between June 24th and July 24th, 2025. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 16 findings throughout this audit engagement.

In particular, we identified a vulnerability where the function responsible for retrieving the asset price fails to check if the Chainlink price data is stale, risking the utilization of outdated asset prices (OS-AAV-ADV-01). Additionally, it is possible for the asset listing admin to arbitrarily modify reserve configurations, effectively bypassing role separation and increasing governance risk (OS-AAV-ADV-02).

Furthermore, the rewards controller utilizes a vector-backed structure to track user data, which grows unbounded as users interact with the system, resulting in out-of-gas errors during critical operations such as minting or liquidation, creating a denial-of-service and potential protocol insolvency (OS-AAV-ADV-00).

We also recommended codebase modifications to enhance functionality and align with coding best practices (OS-AAV-SUG-09, OS-AAV-SUG-04), and suggested incorporating additional checks and assertions to mitigate potential security issues and improve error handling (OS-AAV-SUG-05, OS-AAV-SUG-07).

We further advised emitting events only on meaningful state changes, and ensuring that functions log events when necessary (OS-AAV-SUG-06). Lastly, we highlighted the necessity of removing multiple instances of repetitive and unutilized code for improved gas optimization, clarity, and maintainability (OS-AAV-SUG-08, OS-AAV-SUG-10).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/aave/aptos-v3. This audit was performed against commit a8f9c40.

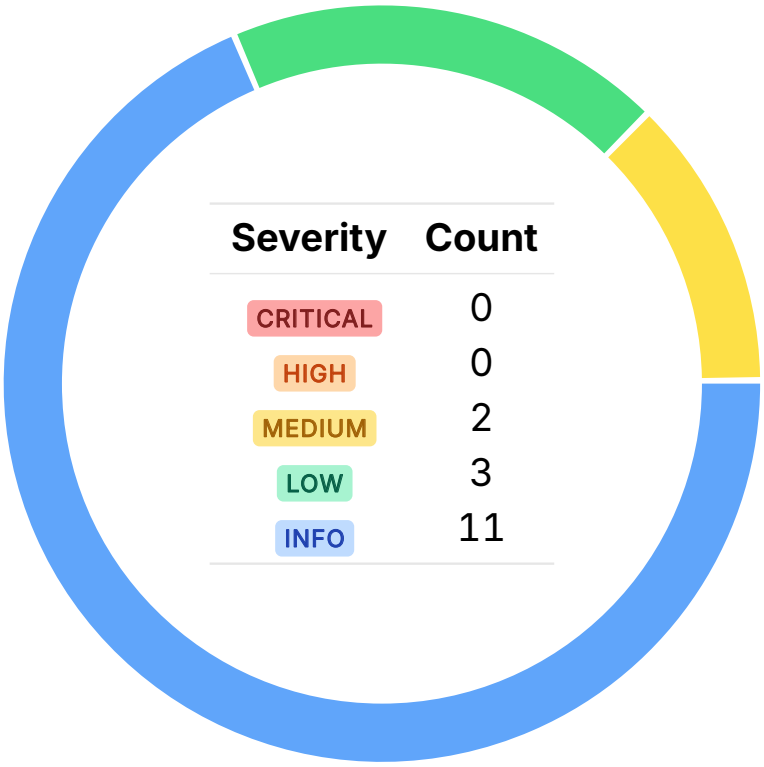
A brief description of the program is as follows:

Name	Description
aave-aptos-v3	A Move-based implementation of the Aave V3.3 lending protocol, enabling decentralized borrowing and lending with enhanced security, composability, and performance. It leverages Aptos's resource-oriented design, strict module boundaries, and formal verification to ensure safe and efficient asset management.

03 — Findings

Overall, we reported 16 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	0
HIGH	0
MEDIUM	2
LOW	3
INFO	11

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-AAV-ADV-00	MEDIUM	RESOLVED	The rewards controller utilizes a vector-backed structure to track user data, which grows unbounded as users interact with the system, resulting in out-of-gas errors during critical operations such as minting or liquidation, creating a denial-of-service and potential protocol insolvency.
OS-AAV-ADV-01	MEDIUM	RESOLVED	`get_asset_price_internal` does not check if the Chainlink price data is stale, risking the utilization of outdated asset prices.
OS-AAV-ADV-02	LOW	RESOLVED	The asset listing admin may arbitrarily modify reserve configurations via `set_reserve_configuration_with_guard`, effectively bypassing role separation and increasing governance risk.
OS-AAV-ADV-03	LOW	RESOLVED	The `SmartTable` is vulnerable to a hash-denial-of-service attack, where malicious users may overload a specific bucket, creating aborts and denying access to legitimate users with colliding keys.
OS-AAV-ADV-04	LOW	RESOLVED	The `pending_ltv_set` variable is shadowed inside the `if` block, resulting in the emitted event always reporting a zero value instead of the actual `LTV`.

Denial of Service Due to Unbounded Rewards Map MEDIUM OS-AAV-ADV-00

Description

The `rewards_controller` logic utilizes a `SimpleMap` to store `users_data` for each reward distribution. `SimpleMap` is implemented as a vector of key-value pairs, which implies that every insertion or lookup requires linear-time scanning of the vector. As this structure grows, its performance degrades, and operations that iterate over it may run out of gas. Specifically, in this case, every user is expected to maintain an entry in the map. As a result, the map is expected to grow indefinitely.

>_ *aave-core/sources/aave-periphery/rewards_controller.move* RUST

```
struct RewardData has key, store, drop, copy {
    index: u128,
    emission_per_second: u128,
    max_emission_rate: u128,
    last_update_timestamp: u32,
    distribution_end: u32,
    users_data: SimpleMap<address, UserData>
}
```

Since this is permissionless, a malicious actor may create many such entries by interacting with the pool repeatedly. Consequently, if the vector grows excessively, `handle_action` may start failing due to out-of-gas issues, as gas utilization for common operations such as `mint`, `burn`, `supply`, or `liquidate` may exceed the transaction limit, resulting in a denial-of-service scenario.

This behavior may have severe implications. For instance, an attacker may fill the vector with numerous positions, bloating the map until liquidation becomes impossible due to gas exhaustion, and then open a position where they borrow a significant amount of funds near the liquidation threshold. Subsequently, even if the position becomes eligible for liquidation, it may remain unliquidatable, potentially creating bad debt for the protocol.

Remediation

Avoid utilizing vector-backed structures for permissionless or unbounded data storage to prevent unbounded growth and potential denial-of-service risks.

Patch

Aave acknowledged this issue.

Failure to Check for Stale Oracle Price MEDIUM OS-AAV-ADV-01

Description

`oracle::get_asset_price_internal` does not validate the freshness of the data while retrieving prices from the feeds. While `chainlink-data-feed` ensures that older data does not overwrite newer entries by not updating the price if `feed.observation_timestamp ≥ observation_timestamp`, it does not prevent the oracle from returning stale prices if the feed stops updating. This creates a risk where outdated prices could be used in core protocol logic.

Remediation

Enforce a maximum timestamp age check to reject stale data explicitly while retrieving the asset price.

Patch

Resolved in PR#445.

Allocation of Excessive Privileges to Listing Admin LOW OS-AAV-ADV-02

Description

The current design of AAVE's governance framework delegates admin responsibilities across multiple roles, helping reduce centralization risks by distributing authority across different addresses. Specifically, the asset listing admin is responsible for adding new reserves by setting their configurations to default values and configuring oracles, while the pool admin manages critical parameters of active reserves.

>_ *aave-core/sources/aave-pool/pool.move* RUST

```rust
/// @notice Sets the reserve configuration with guard
/// @param account The account signer of the caller
/// @param asset The address of the underlying asset of the reserve
/// @param reserve_config_map The new configuration bitmap
public fun set_reserve_configuration_with_guard(
    account: &signer, asset: address, reserve_config_map: ReserveConfigurationMap
) acquires Reserves, ReserveData {
    assert!(
        acl_manage::is_asset_listing_admin(signer::address_of(account))
            || acl_manage::is_pool_admin(signer::address_of(account)),
        error_config::get_ecaller_not_asset_listing_or_pool_admin()
    );
    set_reserve_configuration(asset, reserve_config_map);
}
```

However, `pool::set_reserve_configuration_with_guard` permits the pool admin and the asset listing admin to update any reserve's configuration. While this is intended for the pool admin, the listing admins should not be allowed to arbitrarily overwrite the configuration of any reserve. This goes beyond their intended scope and grants them unrestricted control over reserve parameters. As a result, the listing admin effectively gains elevated super-admin privileges. This undermines the intended role hierarchy and significantly increases the protocol's governance and security risks.

Remediation

Ensure `set_reserve_configuration_with_guard` restricts the listing admin from modifying reserve configurations beyond their intended permissions.

Patch

Aave acknowledged this issue.

Denial of Service via Hash Collision LOW OS-AAV-ADV-03

Description

The `SmartTable` structure is susceptible to a hash-based denial-of-service attack, enabling an attacker who controls the keys to insert multiple entries that hash to the same bucket, temporarily blocking further insertions into that bucket. This vulnerability affects multiple instances across the codebase, with particularly serious implications in permissionless contexts such as Aave's `UsersConfig`, which maps user addresses to configuration data.

>_ *aave-core/sources/aave-pool/pool.move* RUST

```
/// @notice Map of users address and their configuration data (user_address =>
    ↪  UserConfigurationMap)
struct UsersConfig has key {
    value: SmartTable<address, UserConfigurationMap>
}
```

Since keys (addresses) are controlled by users, an attacker may generate multiple addresses that deliberately hash to the same bucket. Each of these attacker-controlled accounts may open small positions, triggering insertions into the `SmartTable`. As the targeted bucket fills up and hits the hardcoded limit of 10,000 entries, any new user whose address hashes to that same bucket will be unable to interact with the protocol, resulting in an unexpected abort. This effectively blocks legitimate users from participating, creating a denial-of-service for that bucket.

Remediation

Avoid utilizing `SmartTable` in contexts where anyone may permissionlessly add data. In this scenario, it is particularly more appropriate to use `Table` to store the users' configuration maps instead of `SmartTable`.

Patch

The Avve team acknowledged the issue but stated they do not plan to address it immediately, as the mainnet version has already been released. They noted that once new data collections such as `OrderedMap` stabilize, they intend to migrate to them due to their improved concurrency support and performance over both `SmartTables` and `Tables`.

Improper Event Emission Due to Variable Shadowing LOW OS-AAV-ADV-04

Description

In `pool_configurator::set_reserve_freeze`, the variable `pending_ltv_set` is declared twice, once outside the `if (freeze)` block and again inside it. This results in variable shadowing, where the inner declaration hides the outer one. As a result, when the `PendingLtvChanged` event is emitted, it always utilizes the outer `pending_ltv_set`, which remains zero. This results in incorrect event logging, even though the correct `LTV` is stored in the Smart Table.

>_ aave-core/sources/aave-pool/pool_configurator.move RUST

```
public entry fun set_reserve_freeze(
    account: &signer, asset: address, freeze: bool
) acquires InternalData {
    [...]
    let pending_ltv_set = 0;
    let ltv_set = 0;
    if (freeze) {
        let pending_ltv_set = reserve_config::get_ltv(&reserve_config_map);
        smart_table::upsert(&mut internal_data.pending_ltv, asset, pending_ltv_set);
        reserve_config::set_ltv(&mut reserve_config_map, 0);
    } else {[...]};

    event::emit(PendingLtvChanged { asset, pending_ltv_set });
    [...]
}
```

Remediation

Remove the re-declaration of `pending_ltv_set` within the `if` branch.

Patch

Resolved in PR#457.

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-AAV-SUG-00	Currently, the `liquidation_bonus` is incorrectly set to at most `PERCENTAGE_FACTOR`, implying that no real bonus is offered to liquidators.
OS-AAV-SUG-01	The script incorrectly checks `is_emergency_admin` instead of `is_emission_admin`, resulting in emission admins who are also emergency admins being skipped during setup.
OS-AAV-SUG-02	`set_reserve_pause` allows redundant or conflicting state changes, which may silently alter or preserve the liquidation grace period.
OS-AAV-SUG-03	`primary_fungible_store` may be utilized over `fungible_store` for cleaner code and proper handling of store initialization and dispatchable assets.
OS-AAV-SUG-04	Certain parts of the codebase can be updated to maintain consistency and ensure alignment with coding best practices.
OS-AAV-SUG-05	There are several instances where proper validation is not performed, resulting in potential security issues.
OS-AAV-SUG-06	Multiple functions emit events even when no state-change has occurred and others fail to log events when required.
OS-AAV-SUG-07	The codebase will benefit from additional assertions to improve general error handling and traceability of potential runtime errors.
OS-AAV-SUG-08	There are several redundant and repetitive code instances that may be removed to save gas.

OS-AAV-SUG-09	Recommendation for updating the codebase to improve overall clarity and functionality and mitigate potential security issues.
OS-AAV-SUG-10	The codebase contains multiple cases of redundant and unutilized code that should be removed for better maintainability and clarity.

Incorrect Liquidation Bonus Configuration OS-AAV-SUG-00

Description

The `liquidation_bonus` in `v1_values::build_reserve_config_mainnet` is incorrectly set within the range `[0, PERCENTAGE_FACTOR]`, which implies liquidators receive at most 100% of the repaid value. This misconfiguration fails to provide the intended bonus and removes the financial incentive for third-party liquidators, as the bonus should actually be greater than the `PERCENTAGE_FACTOR`. As a result, undercollateralized positions may remain unliquidated, increasing protocol insolvency risk. The issue is especially critical since it's already deployed on mainnet.

>_ aave-core/aave-data/sources/v1_values.move RUST

```
public fun build_reserve_config_mainnet(): SmartTable<string::String, ReserveConfig> {
    let reserve_config = smart_table::new<String, ReserveConfig>();
    smart_table::upsert(
        &mut reserve_config,
        utf8(APT_ASSET),
        ReserveConfig {
            base_ltv_as_collateral: (58 * math_utils::get_percentage_factor()) / 100, // ok
            liquidation_threshold: (63 * math_utils::get_percentage_factor()) / 100, // ok
            liquidation_bonus: (10 * math_utils::get_percentage_factor()) / 100, // ok
            [...]
        }
    );
    [...]
}
```

Remediation

Correct the liquidation bonus and adjust it to the correct value in the setters.

Inconsistency in Role Check Logic OS-AAV-SUG-01

Description

In `1_configure_acl`, there is a logic error in the emission admin setup block. It mistakenly checks `is_emergency_admin` instead of `is_emission_admin` before assigning the emission admin role. This results in the script skipping adding emission privileges for addresses that are already emergency admins.

Remediation

Ensure that the check correctly evaluates whether the emission role is already assigned.

Patch

Resolved in PR#453.

State Handling Discrepancies OS-AAV-SUG-02

Description

`set_reserve_pause` in `pool_configurator` lacks state validation, allowing operations such as pausing an already paused reserve or unpausing one that is already active. This will result in unnecessary emission of events and misleading logs. Moreover, unpausing a reserve with a new `grace_period` silently updates the grace period, even if it is already unpaused. Additionally, failing to set the grace period in case of 0 may introduce confusing edge cases. For example, if a reserve was paused, then unpaused with a grace period, then paused again, and then unpaused again without a grace period, the grace period from the first unpause may still be in effect.

Remediation

Add a state check in `set_reserve_pause` to prevent any attempts to pause an already paused reserve or unpause one that is not paused.

Patch

Aave acknowledged this issue by stating that `set_reserve_pause` may only be executed by the admins, who are trusted in nature.

Standardization of Asset Operations OS-AAV-SUG-03

Description

The `fungible_store` and `fungible_asset` APIs are widely utilized across the codebase for asset transfers, deposits, and balance operations. However, they require repetitive boilerplate code and they fail to gracefully handle cases where a store does not yet exist. It is more appropriate to utilize `primary_fungible_store` instead. This higher-level API handles store initialization automatically if it does not exist, supports dispatchable assets as expected by the framework, and reduces boilerplate code.

Remediation

Utilize `primary_fungible_store` over `fungible_store` API, standardizing asset operations.

Patch

Aave acknowledged the utilization of `fungible_store`, stating that many objects contain secondary stores, rendering `primary_fungible_store` unsuitable in those cases. They added that while `primary_fungible_store` has been utilized wherever possible, `fungible_store` remains necessary for their design and architecture.

Code Maturity OS-AAV-SUG-04

Description

1. `token_base::drop_token` currently removes only the `ManagedFungibleAsset` but leaves the `TokenBaseState` resource undeleted. This results in incomplete cleanup and unnecessary on-chain storage. The function should also remove `TokenBaseState` to fully drop the token.

   >_ aave-core/sources/aave-tokens/token_base.move RUST

   ```rust
   /// @notice Drops the token data from the token map
   /// @dev Only callable by the a_token_factory and variable_debt_token_factory module
   /// @param metadata_address The address of the token
   public(friend) fun drop_token(metadata_address: address) acquires ManagedFungibleAsset {
       assert_token_exists(metadata_address);
       assert_managed_fa_exists(metadata_address);
       // detach the managed fungibel asset from the metadata address
       move_from<ManagedFungibleAsset>(metadata_address);
   }
   ```

2. `ray_to_wad` and `wad_to_ray` do not explicitly handle the case when input `a == 0`, unlike other math functions in `aave_math::wad_ray_math`. Adding an early return if `a` is 0, aligns with existing patterns, improving consistency. Similarly, `wad_mul` should return early 0 if `a` or `b` is 0.
3. In `aave_math::math_utils`, functions such as `percent_mul` and `percent_div` lack clear documentation on rounding behavior, unlike `aave_math::wad_ray_math`, where the rounding direction is mentioned at the beginning and in every function's description. This inconsistency introduces ambiguity and may result in incorrect assumptions about numerical precision.
4. `token_base::transfer` performs unnecessary computations for zero-amount transfers. A quick return may optimize performance and reduce resource usage.

Remediation

Implement all the listed suggestions.

Patch

1. Resolved in PR#44.

Missing Validation Logic OS-AAV-SUG-05

Description

1. The oracle module currently permits either the asset listing admin or the pool admin to set a custom price for certain assets. However, `set_asset_custom_price` does not check whether the value of the custom price exceeds the asset's configured price cap. It fails to verify whether a price cap is defined for the asset. As a result, even if a high custom price (above the limit) is set, the oracle may still return the capped price, creating inconsistencies between the set custom price and the effective price utilized by the system.

   >_ aave-core/aave-oracle/sources/oracle.move RUST

   ```rust
   /// @notice Sets a custom price for an asset
   /// @param account Admin account that sets the price
   /// @param asset Address of the asset
   /// @param custom_price Custom price value
   public entry fun set_asset_custom_price(
       account: &signer, asset: address, custom_price: u256
   ) acquires PriceOracleData {
       only_asset_listing_or_pool_admin(account);
       assert!(custom_price > 0, error_config::get_ezero_asset_custom_price());
       update_asset_custom_price(asset, custom_price);
   }
   ```

2. `acl_manage` allows removal of all `DEFAULT_ADMIN_ROLE` holders without enforcing a minimum. This may result in loss of administrative control over the protocol. A safeguard should be added to ensure that at least one admin is always present.
3. `emission_manager::configure_assets` does not currently validate whether `distribution_ends` timestamps are in the future. This allows configuring rewards that are already expired, resulting in wasted state updates and unnecessary emissions.

Remediation

1. Ensure that `set_asset_custom_price` enforces validation to prevent the custom price from exceeding the defined price cap, if one exists.
2. Ensure at least one `DEFAULT_ADMIN_ROLE` holder remains at all times.
3. Add a check to ensure that each distribution period remains valid.

Patch

1. Resolved in PR#45.
2. Aave acknowledged this issue, stating that it reflects the original logic inherited from Aave v3 (Solidity) and is therefore by design, with no plans for change.
3. Aave acknowledged this issue, attributing it to an operational oversight, as not all participants may configure the `emission_manager`.

Event Emission Inconsistencies OS-AAV-SUG-06

Description

1. `fee_manager::set_apt_fee` emits a `FeeChanged` event even when the new fee is the same as the current one, which may mislead observers and clutter event logs. Avoid emitting the event unless the fee actually changes, consistent with how, e.g., `collect_apt_fee` suppresses no-op events.
2. Setters in `pool_configurator` should first check whether the desired value differs from the current state. For instance, `set_borrowable_in_isolation` emits a `BorrowableInIsolationChanged` event even if the asset is already configured as borrowable in isolation. To prevent confusion, avoid emitting events when no actual change occurs.

Remediation

Ensure functions emit events only when a meaningful state change occurs, as emitting events without changes may create confusion. For key state transitions, event emission is essential to enhance protocol transparency and traceability.

Patch

Aave stated that the event emission logic in `pool_configurator` matches that on Aave Solidity implementation, where the event is emitted regardless of whether the configuration is updated or not.

Error Handling OS-AAV-SUG-07

Description

1. `withdraw_apt_fee` currently lacks a check to ensure the fee collector holds enough `APT` before transferring. Without this, transfers may fail with unclear errors if the balance is insufficient. Adding an explicit balance check to ensure sufficient availability of `APT`.
2. Currently, the assertion in `6_configure_price_feeds::main` that checks if the caller is a risk or pool admin, returns an incorrect error code meant for an incorrect asset listing or pool admin (`get_ecaller_not_asset_listing_or_pool_admin`). This ambiguity will create confusion during failures. Replace it with the correct `get_ecaller_not_risk_or_pool_admin` error for clarity and accuracy.

>_ aave-core/aave-scripts/sources/6_configure_price_feeds.move RUST

```
fun main(account: &signer, network: String) {
    // Verify the caller has appropriate permissions
    assert!(
        acl_manage::is_risk_admin(signer::address_of(account))
            || acl_manage::is_pool_admin(signer::address_of(account)),
        error_config::get_ecaller_not_asset_listing_or_pool_admin()
    );
    [...]
}
```

3. `validate_flashloan_simple` in `validation_logic` only checks the `aToken` total supply, which does not reflect real-time liquidity. As a result, `execute_flash_loan_*` may pass validation but fail during execution when subtracting from the `virtual_underlying_balance` (`virtual_balance - (amount as u128)`) if there is insufficient available balance. This will result in unexpected runtime aborts. `validate_flashloan_simple` should also check the virtual balance to ensure sufficient liquidity. Similar issue is present in the borrowing mechanism.
4. `pool_token_logic::transfer` and `token_base::transfer` lack proper validation of transfer amounts. Attempting to send more than the sender's balance results in a subtraction underflow, aborting the transaction. Explicitly check that the sender has enough balance to satisfy the transfer amount.

Remediation

Incorporate the error handling checks mentioned above.

Patch

Resolved in PR#451.

Code Optimization OS-AAV-SUG-08

Description

1. `fee_manager::get_apt_fee` should utilize `borrow_with_default` to simplify logic by directly returning the configured fee or `DEFAULT_APT_FEE` if the asset is not found, eliminating the need for a separate `contains` check.
2. Frequent repeated borrows of the `FeeConfig` global object in the `fee_manager` module introduce unnecessary code duplication. Extract this logic into a helper function to improve readability and reduce redundancy.
3. Compute `signer::address_of(account)` just once and re-utilize the result instead of calling it twice (as highlighted below) in `fee_manager::init_module`. Also, in `get_user_asset_balances`, the `Object<Metadata>` should only be retrieved once.

>_ aave-core/sources/aave-pool/fee_manager.move RUST

```
fun init_module(account: &signer) {
    assert!(
        signer::address_of(account) == @aave_pool,
        error_config::get_enot_pool_owner()
    );
    let signer_addr = signer::address_of(account);
    [...]
}
```

4. `pool_configurator::set_asset_emode_category` continues execution even if the asset is already assigned to the target category. Add an early return when `new_category_id` matches the current one to save gas and prevent unnecessary state changes.
5. `acl_manage::renounce_role` unnecessarily takes a user parameter, even though it must always match the signer's address. This may be simplified by removing the user argument and directly utilizing `signer::address_of(admin)`, reducing redundancy and improving clarity.

>_ aave-core/aave-acl/sources/acl_manage.move RUST

```
public entry fun renounce_role(admin: &signer, role: String, user: address) acquires Roles
    ↪ {
    assert!(signer::address_of(admin) == user,
        ↪  error_config::get_erole_can_only_renounce_self());
    revoke_role_internal(admin, role, user);
}
```

6. In `emission_manager`, `set_pull_rewards_transfer_strategy` redundantly calls `get_rewards_controller_ensure_defined` twice, resulting in repeated state access and checks. This may be optimized by calling it once and storing the result in a local variable. Similarly, `configure_assets_internal` also calls `get_rewards_controller_ensure_defined` twice, instead of utilizing the already existing `rewards_controller` variable.

>_ aave-core/sources/aave-periphery/emission_manager.move RUST

```
public entry fun set_pull_rewards_transfer_strategy([...]) acquires EmissionManagerData {
    [...]
    assert!(
        transfer_strategy::pull_rewards_transfer_strategy_get_incentives_controller(
            pull_rewards_transfer_strategy
        ) == get_rewards_controller_ensure_defined(),
        error_config::get_eincentives_controller_mismatch()
    );

    rewards_controller::set_pull_rewards_transfer_strategy(
        reward,
        object::object_address(&pull_rewards_transfer_strategy),
        get_rewards_controller_ensure_defined()
    );
}
```

7. Within `validation_logic::validate_borrow`, both `siloed_borrowing_enabled` and `siloed_borrowing_address` are declared twice in the `if` block, which is unnecessary. Remove the repetitive declaration of these variables.

Remediation

Optimize the code as stated above.

Patch

Resolved in PR#451.

Code Refactoring OS-AAV-SUG-09

Description

1. In `ReserveConfigurationMap`, the comment describing the reserve configuration structure is inaccurate. There is no *virtual accounting* bit at position 252, nor any corresponding mask. Update the comment to accurately reflect the structure.
2. `v1_values::build_price_feeds_mainnet` includes an outdated hardcoded address for the `SUSDE_ASSET` price feed, which no longer points to the correct on-chain data source. Update the address to the current, correct oracle feed.

>_ aave-core/aave-data/sources/v1_values.move RUST

```
public fun build_price_feeds_mainnet(): SmartTable<String, vector<u8>> {
    let price_feeds_mainnet = smart_table::new<string::String, vector<u8>>();
    [...]
    smart_table::add(
        &mut price_feeds_mainnet,
        string::utf8(SUSDE_ASSET),
        x"01532c3a7e0003320000000000000000000000000000000000000000000000000000"
    );
    price_feeds_mainnet
}
```

3. `fee_manager::withdraw_apt_fee` transfers native APT coins directly, which may result in stuck funds if the `to` address (for example, a rewards contract) is unable to handle coins natively. While `coin` API handles both coins and paired fungible assets, the fungible asset API is capable of handling fungible assets only. Thus, for smart contracts that only support the fungible asset interface, such transfers may render the funds unusable. To prevent this, `APT` may be converted into a fungible asset before withdrawal.

Remediation

Modify the code to include the refactors.

Patch

1. Resolved in PR#451.
2. Resolved in 7163a7e.
3. Aave acknowledged this by stating that the fees are collected in `APT` via `aptos_account::transfer_coins`, as `APT` is a coin rather than a fungible asset, and thus, the pool admin must ensure the recipient has a coin store to receive the transfer. An account with the required store is already in place, and `APT` will not be converted to its mapped fungible asset, though an API exists for conversion if needed.

Unutilized Code

OS-AAV-SUG-10

Description

1. Several unutilized constants are defined across the codebase and may be removed, such as `EPRICE_ORACLE_SENTINEL_CHECK_FAILED` in `aave-config`, `EMISSING_OBJECT_REFERENCE` in `large_packages`, and `DEPLOYMENT_FAILURE` in `1_configure_acl` within `aave-scripts`.
2. There is no requirement to check for `only_pool_admin` in `variable_debt_token_factory` and `a_token_factory` before the `set_incentives_controller` call because this check is already performed inside `set_incentives_controller`.
3. The last `if` statement in `rewards_controller::claim_rewards_internal_update_data` is unnecessary as the function will return zero if `total_rewards` is zero.
4. In `ui_pool_data_provider_v3::get_reserves_data`, the `_underlying_asset_metadata` variable is unutilized and may be removed.
5. In simple flash loan, the `SimpleFlashLoansReceipt` structure does not need to store `on_behalf_of`, as this field is unutilized and it is not possible for flash loans to be executed on behalf of another user.
6. The `total_debt_accrued` field in `pool_logic::AccrueToTreasuryLocalVars` structure is unutilized as it is never populated.
7. `E_ACCOUNT_NOT_EXISTS` in `mock_underlying_token_factory` is never utilized and may be removed.

Remediation

Remove all instances of unutilized code.

Patch

Resolved in PR#42.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.