certora

Security Assessment Report

Aave Aptos Core V3.0.2

April 2025

Prepared for Aave Labs

certora

Table of content

certora

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Aave Aptos	https://github.com/aave/aptos-v3	80d6e6f	Aptos

Project Overview

This document describes the manual code review and findings of **Aave V3 on Aptos**. The work was undertaken from **Feb 2, 2025** to **Apr 7, 2025**

The following contract list is included in our scope:

```
aptos-v3/aave-core/aave-acl/*
aptos-v3/aave-core/aave-config/*
aptos-v3/aave-core/aave-math/*
aptos-v3/aave-core/aave-oracle/*
aptos-v3/aave-core/aave-rates/*
aptos-v3/aave-core/sources/*
```

The team performed a manual audit of all the Aptos smart contracts. During the manual audit, the Certora team discovered bugs in the Aptos smart contracts code, as listed on the following page.

Protocol Overview

Aave V3 on Aptos is a decentralized, non-custodial liquidity protocol built on the Aptos blockchain using the Move language, adapted from its original Ethereum implementation. At its core, Aave enables users to participate as liquidity providers by depositing assets into lending pools, or as borrowers who can take out loans using their deposited assets as collateral.
Its architecture revolves around a unified pool containing different assets, where each asset has an associated aToken that represents a user's deposit and automatically accrues interest, while borrowers receive debt tokens that track their loan obligations and accrued interest. Key actors include lenders who earn passive income, borrowers who can make overcollateralized loans or flash loans, liquidators who maintain the system's solvency by liquidating undercollateralized positions, and governance participants who decide on protocol parameters. This Aptos implementation leverages Move's resource-oriented programming model and native security features while maintaining the same economic mechanics and risk parameters as the Ethereum version.

Security Considerations

Aave Aptos is an Aave V3 core and periphery code adapted from EVM (Solidity) to Aptos (Move language). The goal of this project is to deploy Aave V3 on Aptos as the first non-EVM instance of the protocol. Therefore, this code was designed to behave exactly like the EVM version, containing the same logic and flows. Since the framework and programming languages are fundamentally different, there are expected differences in the implementations as well as potential issues that may emerge.
Bearing that in mind, we conducted a thorough audit of the entire codebase, constantly comparing it with the EVM version to ensure the logic flow remained identical. We also inspected the flow and underlying assumptions from a fresh perspective of the new framework and environment on Aptos.
Note that any issues or design choices that were reported and discussed in the past on the EVM version aren't mentioned here. However, these were fully considered in the audit itself to ensure that no assumptions and limitations were further broken when moving to the new framework.

Audit Goals

1. Verify that the initialization of modules is done correctly and adheres to the equivalent initializations and configurations in EVM.

2. User actions should not exceed any permissions and should maintain EVM parity.

3. Check if removal of EVM permit functionality behaves as expected using Aptos native equivalent.

4. Debt tokens are implemented as non-donatable, as opposed to the default ability to withdraw fungible assets from an owned store and transfer them.

5. Function visibility and access control is implemented correctly using the Aptos framework.

6. Verify that the internal accounting has been introduced properly and donations cannot be used to manipulate the system.

7. Verify that the adaptation from Ethereum's 18-decimal precision to Aptos's 8-decimal maximum precision is properly implemented across all protocol functions, particularly in interest accrual, liquidation thresholds, and health factor calculations.

8. All numerical casting operations should maintain mathematical integrity and do not introduce vulnerabilities through truncation, overflow, or precision loss, especially in high-value scenarios.

9. Validate that the Move resource-oriented programming model correctly implements the same asset handling and transfer logic as Solidity's account-based model, with no funds being unaccounted for.
    a. Validate flashloan functionality is implemented correctly as it requires different mechanisms and assumptions from the original EVM version.

Coverage and Conclusions

1. Module initializations are done correctly and correspond to its equivalent EVM functionality.

2. User actions do not exceed any permission and maintain EVM parity.

3. Removal of EVM permit functionality and introduction of Aptos native equivalent behaves as expected.

4. The non-transferability was implemented via freezing all stores. This was properly implemented in token_base, and the debt tokens (and a-tokens) use this functionality exclusively.

5. Function visibility and access control has been implemented correctly using the Aptos framework.

6. The internal accounting has been introduced properly and donations cannot be used to manipulate the system.

7. The 8-decimal precision limitation in Aptos (compared to Ethereum's 18) has been properly handled throughout all financial calculations with appropriate scaling factors. However, it is still important to keep in mind that:
    a. Rounding effects are more significant for high-value and high-volatility assets like ETH, where small percentage changes can translate to meaningful value differences when working with fewer decimal places.
    b. Interest accrual calculations maintain mathematical correctness despite the precision constraints.
    c. The protocol has implemented safeguards to prevent precision loss during critical operations like liquidations and health factor calculations.

8. All numerical casting operations maintain mathematical integrity and do not introduce vulnerabilities.

9. The Move resource-oriented model successfully translates Solidity's account-based asset handling:
    a. All asset transfers maintain precise accounting with no untracked funds.
    b. The implementation properly handles Move's capability-based security model while achieving equivalent functionality to the EVM version.
    c. Resource ownership and transfer semantics correctly match the authorization flows from the original Solidity implementation, including the flashloan mechanism.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	0	0	0
High	0	0	0
Medium	2	2	2
Low	1	1	1
Informational	3	3	3
Total	6	6	6

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

certora

Detailed Findings

ID	Title	Severity	Status
M-01	Health factor check is too low	Medium	Fixed
M-02	Multiple indexes can map to the same reserve	Medium	Fixed
L-01	LTV could be mistakenly reset due to missing set_reserve_freeze() check	Low	Fixed
I-01	Gas Optimization in get_overall_borrow_rate	Informational	Fixed
I-02	Non-view getter functions	Informational	Fixed
I-03	Typos	Informational	Fixed

Medium Severity Issues

M-01 Health factor check is too low		
Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: validation_logic.move	Status: Fixed	

Description: The function *validate_liquidation_call* first checks if health_factor is less than 0.95e18 and then checks if health_factor is less than 1e18.

The second check is "redundant" at face value. However, asserting that health_factor must be less than 0.95e18 to be liquidated means that a health factor being below the *HEALTH_FACTOR_LIQUIDATION_THRESHOLD* is not enough for a debtor to be liquidated. A user needs to go considerably under the health factor in order to be liquidatable.

This leads to a higher possibility of bad debt and less profitable outcomes for liquidators, placing the whole system at risk.

In particular, if the liquidation threshold times the liquidation bonus if above 0.95, this will result in a loss to the protocol on liquidation. A quick calculation to show this:

LT * LB > 0.95

0.95 > HF = collateral * LT / debt > (collateral / debt) * 0.95 / LB

So: 1 > (collateral / debt) / LB

And so: debt * liquidation_bonus > collateral

Meaning that if the debt is liquidated, we will need to pay more than the collateral - resulting in bad debt.

This obviously depends on the configuration of each reserve and emode. The limit for liquidation threshold times the liquidation bonus in the configuration is 1. This can realistically occur in particularly high-efficiency emodes (for example, the stablecoin emode during development had a LT of 0.97 and a liquidation bonus of 1.01, which when multiplied give 0.9797 > 0.95).

Recommendations: We recommend removing the first check comparing the health factor against *get_minimum_health_factor_liquidation_threshold()*.

Customer's response: Fixed

Fix Review: Fix confirmed.

certora

M-02 Multiple indexes can map to the same reserve		
Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: pool_token_logic.move	Status: Fixed	

Description: The function *init_reserve()* does not break when finding a valid index to use.

When adding a new reserve, the function first looks for an unused index in the current range (and increases the range of indexes if none were found). If an unused index was found, we map this index to the new reserve. However, we do not break from the loop, but continue looking for more unused indexes. If we find multiple such indexes, we will have multiple indexes mapping to the same reserve. (For multiple indices to be unused within the current range, *drop_reserve()* will have to be called twice between uses of *init_reserve()*.)

This will then lead to counting the reserve multiple times when calling *calculate_user_account_data()*, which can lead to counting the same coin as collateral twice and taking debt against it, resulting in a loan worth more than its collateral.

For this to work, *drop_reserve()* will have to be called twice between uses of *init_reserve()*.

Recommendations: We recommend adding a break to the loop after finding an unused index.

Customer's response: Fixed

Fix Review: Fix confirmed.

Low Severity Issues

L-01 LTV could be mistakenly reset due to missing set_reserve_freeze() check

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: pool_configurator.move	Status: Fixed	

Description: The function *set_reserve_freeze*() is not checking if *freeze* is different from the current *frozen* state, which could lead to resetting the current or pending ltv accidentally.

set_reserve_freeze() is supposed to work as a toggle for freezing/unfreezing reserves. However, if we perform the same action twice, for example, unfreezing and then unfreezing again, the *set_reserve_freeze()* functionality goes beyond toggling and undermines the configuration set by *the configure_reserve_as_collateral() function*.

Recommendations: We recommend introducing the following check to ensure *set_reserve_freeze()* behaves in line with its purpose.

```
Unset
    public entry fun set_reserve_freeze(
        account: &signer, asset: address, freeze: bool
    ) acquires InternalData {
        assert!(
            only_risk_or_pool_or_emergency_admins(signer::address_of(account)),
            error_config::get_ecaller_not_risk_or_pool_or_emergency_admin()
        );
        let reserve_config_map = pool::get_reserve_configuration(asset);
+     assert!(
+       reserve_config::get_frozen(reserve_config_map) != freeze,
+         error_config::get_einvalid_freeze_flag()
+       );
```

Customer's response: Fixed

Fix Review: Fix confirmed.

Informational Issues

I-01. Gas Optimization in get_overall_borrow_rate()

Description: Since there is no *stable_borrow_rate* the *overall_borrow_rate* should just be equal to the *current_variable_borrow_rate*. The *calculate_interest_rates()* function currently multiplies with the *total_variable_debt* and then divides again by the same value. It also returns 0 if total_debt is 0 (to avoid division by zero).

Recommendation: Change the get_overall_borrow_rate() function to simply return the *current_variable_borrow_rate*.

```
Unset
    fun get_overall_borrow_rate(
        total_variable_debt: u256, current_variable_borrow_rate: u256
    ): u256 {
        return current_variable_borrow_rate;
    }
```

https://github.com/aave/aptos-v3/blob/main/aave-core/aave-rate/sources/default_reserve_interest_rate_strategy.move#L286C1-L286C33

Customer's response: Fixed

Fix Review: After further discussions, we believe this line does not need to call a function and can use *vars.current_variable_borrow_rate* directly.

I-02. Non-view functions

Description: There are a number of getter functions which are not view as they should be.
In the *pool* module (*pool.move*), there are a number of getter functions which can be defined as view functions but currently are not.

List of functions which are getters but are not declared as view in *pool.move*:
get_reserve_list_length
get_reserve_configuration_by_reserve_data
get_reserve_last_update_timestamp
get_reserve_id
get_reserve_a_token_address
get_reserve_accrued_to_treasury
get_reserve_variable_borrow_index
get_reserve_liquidity_index
get_reserve_current_liquidity_rate
get_reserve_current_variable_borrow_rate
get_reserve_variable_debt_token_address
get_reserve_unbacked
get_reserve_isolation_mode_total_debt
get_normalized_income_by_reserve_data
get_normalized_debt_by_reserve_data
get_reserve_addresses_list_length
get_siloed_borrowing_state

List of functions which are getters (or should otherwise be view functions) but are not declared as view in *emode_logic.move*:
get_emode_e_mode_price_source
get_emode_configuration
get_emode_e_mode_label
get_emode_e_mode_liquidation_bonus
is_in_emode_category
get_emode_category_price_source

List of functions which should be view but are not declared as view in *token_base.move*:
get_previous_index

get_incentives_controller
scaled_balance_of
scaled_total_supply
get_scaled_user_balance_and_supply
name
symbol
decimals

Note that functions which receive references cannot be view functions due to language limitations. The following functions seem like they should be view but cannot be changed to view due to this limitation. They are listed for completeness:

List of functions in emode_logic.move:
get_emode_category_ltv
get_emode_category_liquidation_threshold
get_emode_category_liquidation_bonus
get_emode_category_label

List of functions in pool_logic.move:
get_reserve_cache_configuration
get_reserve_factor
get_curr_liquidity_index
get_next_liquidity_index
get_curr_variable_borrow_index
get_next_variable_borrow_index
get_curr_liquidity_rate
get_curr_variable_borrow_rate
get_a_token_address
get_variable_debt_token_address
get_curr_scaled_variable_debt
get_next_scaled_variable_debt

In pool.move, the function "*get_isolation_mode_state*".

In generic_logic.move, the function "*calculate_user_account_data*" (which is view in the EVM equivalent).

Recommendation: Add the *#[view]* decorator to said functions.

Customer's response: Fixed (parts 1, 2)

Fix Review: Fixed.

I-03. Typos

Description: *reserve_dara* instead of *reserve_data*

Location:
https://github.com/aave/aptos-v3/blob/ee9752dddf034d5fee4485d7d7a30bd14b03818b/aave-core/sources/aave-pool/pool.move#L569

Recommendation:
Rename the variable from *reserve_dara* to *reserve_data* for consistency and clarity.

Customer's response: Fixed

Fix Review: Fixed.

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard

deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.