certora

Security Assessment Report

Aave Aptos Core V3.1–V3.3

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	aaa6570	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	1	1	1
Medium	1	1	1
Low	0	0	0
Informational	2	2	2
Total	4	4	4

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
H-01	actual_collateral_to_liquidate is burned instead of actual_debt_to_liquidate	High	Fixed
M-01	Multiple indexes can map to the same reserve	Medium	Fixed
M-02	set_next_variable_borrow_index() used instead of set_next_scaled_variable_debt()	Medium	Fixed
I-01	Fully liquidated asset is still counted as collateral	Low	Fixed
I-02	Typos	Informational	Fixed

High Severity Issues

H-01 actual_collateral_to_liquidate is burned instead of actual_debt_to_liquidate

Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: liquidation_logic.move	Status: Fixed	

Description: The *liquidation_call()* function contains an issue in the *burn_debt_tokens()* function call. It incorrectly passes *actual_collateral_to_liquidate* as the debt amount to burn, instead of *actual_debt_to_liquidate*.

This mismatch would lead to incorrect debt burning during liquidations, causing debt tokens to be either overly or insufficiently burned or a denial of service on the liquidation . The amount of debt being burned should correspond to the actual debt being liquidated, not the collateral amount.

Recommendations: Modify the *burn_debt_tokens()* function call to pass the correct parameter

```
JavaScript
burn_debt_tokens(
&mut debt_reserve_cache,
debt_reserve,
&mut user_config_map, params.user,
params.debt_asset,
vars.user_reserve_debt,
- vars.actual_collateral_to_liquidate,
+ vars.actual_debt_to_liquidate,
has_no_collateral_left );
```

Customer's response: Fixed

Fix Review: Fix confirmed.

Medium Severity Issues

M-01 set_next_variable_borrow_index() used instead of set_next_scaled_variable_debt()

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: liquidation_logic.move pool_logic.move	Status: Fixed	

Description: In the function liquidation_logic::*burn_debt_tokens*() a call to *set_next_variable_borrow_index*() has been wrongly introduced in the place of *set_next_scaled_variable_debt*(). This approach fails to update the relevant variable n*ext_scaled_variable_debt* and falsely updates *next_variable_borrow_index*, leading to the *total_variable_debt* and consequently, the *current_liquidity_rate* and *current_variable_borrow_rate* being updated to much lower values than they should.

Recommendation: We recommend replacing *set_next_variable_borrow_index()* with *set_next_scaled_variable_debt*().

Customer's response: Fixed

Fix Review: Fix confirmed.

Informational Issues

I-01 Fully liquidated asset is still counted as collateral

Description: The function *liquidation_call*() does not set the asset as no longer being used as collateral if the total liquidation value is equal to the user's balance.

For example, if a user borrows ETH against USDC collateral, and their ETH debt gets repaid, they will no longer be borrowing ETH, so their borrowing flag for ETH is set to false.

Similarly, If a user gets liquidated and their USDC is taken, it is no longer being used as collateral. However, there is no corresponding setter operation to indicate that USDC is no longer being used as collateral. Specifically by calling *set_using_as_collateral()*.

This would lead to a waste of gas in many functions that would count this asset as collateral, even though it contains zero value. It would also falsely maintain the user in isolation mode if the user was previously in isolation mode before the liquidation.

Recommendations: We recommend calling *set_using_as_collateral()* if the total liquidation value is equal to the user's balance.

Customer's response: Fixed

Fix Review: Fix confirmed.

I-02. Typos

Description: *debt_resreve* instead of *debt_reserve*

Location:
https://github.com/aave/aptos-v3/pull/248/files#diff-27bd0d294b958d61acd076735d2dc049fcf4004cdf9c147144bb5cc73d710416R383

Recommendation:
Rename the variable from *debt_resreve* to *debt_reserve* for consistency and clarity.

Description: *util* instead of *until*

Locations:
https://github.com/aave/aptos-v3/blob/main/aave-core/sources/aave-pool/pool.move#L475

https://github.com/aave/aptos-v3/pull/248/files#diff-ae51cd2585f7660c2b4d6d949a9820ae44e0e62551650b44a675f47158feecb2R596

Recommendation:
Correct the typos from *util* to *until* to improve code documentation clarity.

Description: *data_obejct* instead of *data_object*

Location:
https://github.com/aave/aptos-v3/blob/main/aave-core/sources/aave-pool/pool.move#L203

Recommendation:
Rename the variable from *data_obejct* to *data_object* for consistency and clarity.

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

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.