LayerBank

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Wed Jan 22 2025

LayerBank Audit Report

1 Executive Summary

1.1 Project Information

Description	A universal permissionless on-chain bank
Type	DeFi
Auditors	MoveBit
Timeline	Mon Nov 11 2024 - Wed Jan 22 2025
Languages	Move
Platform	Movement
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/layerbank/aptos-contracts
Commits	fe7748f197b9c9d6bd77540bbd409980b0889dd4 193f55cb6327d7215462c725a91ba38b2d287e0c 46c8ee737ba8e96707d043fe9edb792000d91538

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
MOV1	aave-core/aave-config/Move.toml	eb5ec6b482f1a36d99ebed5b1de021f655243d08
HEL	aave-core/aave-config/sources/helper.move	e577bd18d945c8fdd0d75cde25dbe6043b28d032
STO	aave-core/sources/aave-tokens/standard_token.move	eb0ae06e1d5a8847ca98b0dab732ad212dc4a1e7
CWR	aave-core/sources/aave-periphery/coin_wrapper.move	3090c43e7523c8cb0345640ea757340a9d35bfe6
STR	aave-core/sources/aave-periphery/stream.move	2d91c37fdbc7b4e9ae620e165c855879133dfc86
MOV1	aave-core/aave-oracle/Move.toml	0ef4122f9fe86e4d3afef05741e446a9bd2f7e4e
ORA	aave-core/aave-oracle/sources/oracle.move	a637ef6aaac33c088a8721560c9069779c18c216
RCO	aave-core/aave-config/sources/reserve_config.move	d39ead6ae5249c8f32efd5cd266e5d8cd0dec0bf
UCO	aave-core/aave-config/sources/user_config.move	67a6e7eebcb401a1d7a3aee57175dd6510178679
ECO	aave-core/aave-config/sources/error_config.move	ebd324765a4982d59880bce2dea78d70d7a7caa2
MOV5	aave-core/aave-math/Move.toml	2466400930ebcf00df0e950c4068f46730f6c5ec

MUT	aave-core/aave-math/sources/math_utils.move	f402f4243f3aeb0f8fc302b0196b2ec9a976e277
WRM	aave-core/aave-math/sources/wad_ray_math.move	f36dd5a86d48dc9273f9ec21b91b9f2f116bdf14
MOV6	aave-core/aave-large-packages/Move.toml	153cc0ecf47c73c36cba4c8242344e47fd0f17d9
LPA	aave-core/aave-large-packages/sources/large_packages.move	0d181b21707f2ee7384fccf939232337985e820c
MOV8	aave-core/aave-acl/Move.toml	ed2bdbd23d44ac5e995e6fb73bc68fbfb9f2ab68
AMA	aave-core/aave-acl/sources/acl_manage.move	5ad5173408dba8336bad91395e3e7cdbecda9abc
ATF	aave-core/sources/aave-tokens/a_token_factory.move	32083cbe870ab7142b8ce0210930851f07ca81ff
TBA	aave-core/sources/aave-tokens/token_base.move	366b747b91178c973931e39269560c4ddd760182
FAM	aave-core/sources/aave-tokens/fungible_asset_manager.move	3167b679c247b02a8a33fbc271db5769e963be6b
VDTF	aave-core/sources/aave-tokens/variable_debt_token_factory.move	87bec05f324ecc35324c4e9acf86a4d13a2b1380
SLO	aave-core/sources/aave-logic/supply_logic.move	3a775a4b1f9289048cec61e925e4cd5f71e00e0e
FLO	aave-core/sources/aave-logic/flashloan_logic.move	bb3432b8bb78bfd56a318993d960bf3dbb8466d3
GLO	aave-core/sources/aave-logic/generic_logic.move	5a43b6e97dc7c7871d2a738246f1154f7227a765

BLO	aave-core/sources/aave-logic/bridge_logic.move	3fcaa0ffcbdf0898ddf36e2ed6cada80436da0d4
LLO	aave-core/sources/aave-logic/liquidation_logic.move	6fb3cd5ab41cc64867cc4265e34d3a6129d90f6f
VLO	aave-core/sources/aave-logic/validation_logic.move	60f7c4eae61ec8fe46a035c8133947fbedfd8780
ULO	aave-core/sources/aave-logic/user_logic.move	0dde64c2628b5ff2070774a4c5ed142df0ca45f1
BLO1	aave-core/sources/aave-logic/borrow_logic.move	3c560945a4de89b4a1cc01f68b3126c65c0228ca
ELO	aave-core/sources/aave-logic/emode_logic.move	ca015f4061cdc7871e167f1f6eafb8f1d1a9fad8
IML	aave-core/sources/aave-logic/isolation_mode_logic.move	75eb13f99c7246588da86706e729ff8b142b4361
PDP	aave-core/sources/aave-pool/pool_data_provider.move	f88134b68c030e4666fec088982932bcff6c66b6
PCO	aave-core/sources/aave-pool/pool_configurator.move	5d36e3b8c134c6a5c4fd57a106dba2dd6a2b87ac
POO	aave-core/sources/aave-pool/pool.move	f74ef4405aa71b04dc7686ee6d13bf7e6057bbc8
STO2	aave-core/sources/aave-periphery/staked_token.move	b061a80e9c30819b22b5cc1e1c44b21d7cf9d3da
EMA	aave-core/sources/aave-periphery/emission_manager.move	3a5f8be65d99fd341a0fe08480bb8bc19a42c6fa
UPDPV3	aave-core/sources/aave-periphery/ui_pool_data_provider_v3.move	2ec929f8d75f6fc277476947b1c5086cb64d7745

EAP	aave-core/sources/aave-periphery/eac_aggregator_proxy.move	670d11097c35b8244be7db1763d9f250c517240e
COL	aave-core/sources/aave-periphery/collector.move	75aea06f548615f9ac44aa098028a117a24d7f02
UIDPV3	aave-core/sources/aave-periphery/ui_incentive_data_provider_v3.move	50ff59afd7fc5492dbe4603a1f915bfa68b13eab
PMA	aave-core/sources/aave-periphery/package-manager.move	4aef563a07e5d4e6750ff238935157cfa5de1e02
ACER	aave-core/sources/aave-periphery/admin_controlled_ecosystem_reserve.move	e126bdb3c546c091f660547e2f2e75b052ac44bc
RCO1	aave-core/sources/aave-periphery/rewards_controller.move	3c409e9bd4714bc156ebd3f99105eafb03364c31
ERV2	aave-core/sources/aave-periphery/ecosystem_reserve_v2.move	7f318a80f285a3930cebf49e03aee70cd7f0164e
CMI	aave-core/sources/aave-periphery/coin_migrator.move	6ce3f30f34415c6bc2fe10a5002f27bf5e0b33f7
TST	aave-core/sources/aave-periphery/transfer_strategy.move	c2cac0eefbd27e4b6e97300cc29246bfddba9c07

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	19	19	0
Informational	1	1	0
Minor	6	6	0
Medium	3	3	0
Major	5	5	0
Critical	4	4	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification(Optional)

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by LayerBank to identify any potential issues and vulnerabilities in the source code of the LayerBank smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 19 issues of varying severity, listed below.

ID	Title	Severity	Status
COL-1	Lack of Explicit Error Handling in `withdraw` Function	Minor	Fixed
ERV-1	Incorrect Implementation of `withdraw_from_stream`	Critical	Fixed
ERV-2	Incorrect Permission Verification Logic in `is_recipient` Function	Minor	Fixed
FLO-1	Mock Contracts Should Not Be Used	Minor	Fixed
GLO-1	The Token Price Used during Liquidation is not Up-to-date	Medium	Fixed
MUT-1	Optimize `pow` Function with Fast Exponent	Minor	Fixed
ORA-1	The Price Validity Period is Too Long	Critical	Fixed
ORA-2	Different Tokens Should Have Different Validity Periods	Major	Fixed
ORA-3	The Price Calculation is Incorrect	Major	Fixed

ORA-4	`aave_oracle` Lacks Support for Asset Removal	Minor	Fixed
ORA-5	Unnecessary use of `borrow_global_mut`	Informational	Fixed
POO-1	The Calculation of `total_debt_accrued` is Incorrect	Major	Fixed
POO-2	There is an extra Comma in the Parameters when Calculating `curr_total_variable_debt`	Minor	Fixed
RCO-1	Logical Error in Branch Conditions of `claim_rewards_internal` Function	Critical	Fixed
SLO-1	Incorrect Borrowing Status Update After Full Repayment	Major	Fixed
TBA-1	`transfer` Transfers an Incorrect Amount	Critical	Fixed
TBA-2	The implementation of `token_base` is incorrect	Major	Fixed
TBA-3	The `set_frozen_flag()` Function Call is Missing in the `transfer()` Function	Medium	Fixed
TBA-4	Evading Debt	Medium	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the LayerBank Smart Contract :

Owner

- The owner can initialize interest rate strategy through `init_interest_rate_strategy()`
- The owner can set the reserve interest rate strategy through `set_reserve_interest_rate_strategy()`
- The owner can initialize reserves through `init_reserves()`
- The owner can drop a reserve through `drop_reserve()`
- The owner can enable or disable borrowing on a reserve through `set_reserve_borrowing()`
- The owner can configure a reserve as collateral through `configure_reserve_as_collateral()`
- The owner can enable or disable flash loan on a reserve through `set_reserve_flash_loaning()`
- The owner can activate or deactivate a reserve through `set_reserve_active()`
- The owner can freeze or unfreeze a reserve through `set_reserve_freeze()`
- The owner can set a reserve as borrowable in isolation through `set_borrowable_in_isolation()`
- The owner can pause or unpause a reserve through `set_reserve_pause()`
- The owner can change the reserve factor of a reserve through `set_reserve_factor()`
- The owner can set a debt ceiling on a reserve through `set_debt_ceiling()`
- The owner can set siloed borrowing state on a reserve through `set_siloed_borrowing()`
- The owner can set a borrow cap on a reserve through `set_borrow_cap()`
- The owner can set a supply cap on a reserve through `set_supply_cap()`
- The owner can set a liquidation protocol fee on a reserve through `set_liquidation_protocol_fee()`

- The owner can set an e-mode category configuration through `set_emode_category()`
- The owner can set an asset's e-mode category through `set_asset_emode_category()`
- The owner can set an unbacked mint cap on a reserve through `set_unbacked_mint_cap()`
- The owner can pause the entire pool through `set_pool_pause()`
- The owner can update the bridge protocol fee through `update_bridge_protocol_fee()`
- The owner can update the total premium for a flashloan through `update_flashloan_premium_total()`
- The owner can update the protocol's share of the flashloan premium through `update_flashloan_premium_to_protocol()`
- The owner can configure multiple reserves in one function call through `configure_reserves()`
- The owner can initialize the pool through `init_pool()`
- The owner can initialize a reserve through `init_reserve()`
- The owner can drop a reserve through `drop_reserve()`
- The owner can set the accrued treasury amount for a reserve through `set_reserve_accrued_to_treasury()`
- The owner can update reserve interest rates through `update_interest_rates()`
- The owner can set the unbacked value for a particular reserve through `set_reserve_unbacked()`
- The owner can set the isolation mode total debt for a particular reserve through `set_reserve_isolation_mode_total_debt()`
- The owner can set the reserve's configuration through `set_reserve_configuration()`
- The owner can set the bridge protocol fee through `set_bridge_protocol_fee()`
- The owner can set the flash loan premiums through `set_flashloan_premiums()`
- The owner can mint to the treasury for specified asset addresses through `mint_to_treasury()`

- The owner can cumulate additional amounts to the liquidity index of a given reserve through `cumulate_to_liquidity_index()`
- The owner can reset the total isolation mode debt to zero for a given reserve through `reset_isolation_mode_total_debt()`
- The owner can transfer tokens for rescue or redistribution purposes through `rescue_tokens()`
- The owner can set user configurations through `set_user_configuration()`
- The owner can issue a token cap through `issue_cap()`

User

- The user can set user-specific Enhanced Mode (EMode) settings through `set_user_emode()`
- The user can check if two EModes are the same through `is_in_emode_category()`
- The user can get the price source address for a specific EMode category through `get_emode_e_mode_price_source()`
- The user can retrieve EMode configuration details such as Loan-To-Value (LTV), liquidation threshold, and asset price through `get_emode_configuration()`
- The user can get the label of a specific EMode category through `get_emode_e_mode_label()`
- The user can get the liquidation bonus for a specific EMode category through `get_emode_e_mode_liquidation_bonus()`
- The user can calculate their account data related to collateral, debt, LTV, and health factor through `calculate_user_account_data()`
- The user can calculate how much they can still borrow based on their total collateral, existing debt, and LTV through `calculate_available_borrows()`
- The user can validate health factor and loan-to-value through `validate_hf_and_ltv()`
- The user can check automatic collateral usability validation through `validate_automatic_use_as_collateral()`
- The user can validate use as collateral through `validate_use_as_collateral()`
- The user can validate health factor through `validate_health_factor()`

- The user can validate setting of user economic mode through `validate_set_user_emode()`
- The user can borrow assets through `borrow()`
- The user can repay borrowed assets through `repay()`
- The user can repay with A-tokens through `repay_with_a_tokens()`
- The user can liquidate a debt through `liquidation_call()`
- The user can supply assets to the pool through `supply()`
- The user can withdraw assets from the pool through `withdraw()`
- The user can finalize asset transfers between users within the pool through `finalize_transfer()`
- The user can set whether a reserve should be used as collateral by a user through `set_user_use_reserve_as_collateral()`
- The user can deposit assets to the pool on behalf of another user through `deposit()`
- The user can initiate a complex flash loan through `flashloan()`
- The user can initiate a simple flash loan through `flash_loan_simple()`
- The user can repay a complex flash loan through `pay_flash_loan_complex()`
- The user can repay a simple flash loan through `pay_flash_loan_simple()`

4 Findings

COL-1 Lack of Explicit Error Handling in `withdraw` Function

Severity: Minor

Status: Fixed

Code Location:

aave-core/sources/aave-periphery/collector.move#132

Descriptions:

```
public fun withdraw(
    sender: &signer,
    asset_metadata: Object<Metadata>,
    receiver: address,
    amount: u64
) acquires CollectorData {
    // check sender is the fund admin
    is_funds_admin(signer::address_of(sender));

    // borrow the global collector data
    let collector_data = borrow_global_mut<CollectorData>(collector_address());

    // check if we have a secondary fungible store for the asset, if now, throw an error
    if (smart_table::contains(&collector_data.fungible_assets, asset_metadata)) {
        let collector_fungible_store =
            smart_table::borrow(&collector_data.fungible_assets, asset_metadata);
        let collector_fungible_store_signer =
            object::generate_signer_for_extending(&collector_data.extend_ref);
        let receiver_fungible_store =
            primary_fungible_store::ensure_primary_store_exists(
                receiver, asset_metadata
            );

        // transfer the amount from the collector's sec store to the receiver's store using the collectors signer which is also the owner of the sec.store
        fungible_asset::transfer(
            &collector_fungible_store_signer,
            *collector_fungible_store,
```

```
            receiver_fungible_store,
            amount
        );
    }
}
```

In the `withdraw` function of `aave_pool::collector`, the condition `smart_table::contains(&collector_data.fungible_assets, asset_metadata)` is checked to validate whether the asset metadata exists. However, if this condition is not satisfied, there is no explicit error handling or meaningful response to address the failed check.

Suggestion:

Implement explicit error handling to address cases where the condition `smart_table::contains(&collector_data.fungible_assets, asset_metadata)` is not satisfied.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ERV-1 Incorrect Implementation of `withdraw_from_stream`

Severity: Critical

Status: Fixed

Code Location:

aave-core/sources/aave-periphery/ecosystem_reserve_v2.move#270

Descriptions:

The `withdraw_from_stream` function contains several critical issues:

1. **Visibility Scope**:
   The function should be declared as `public`. If it remains private, it cannot be called externally, defeating its intended purpose. The `create_stream` function should also be public.

2. **Missing Funds Transfer**:
   The function does not include the essential operation of transferring funds to the `recipient`. This omission makes the implementation incomplete and non-functional for its intended use.

Suggestion:

1. **Change Visibility to Public**:
   Update the function's visibility to ensure it is accessible when needed.

2. **Implement Funds Transfer**:
   Add logic to transfer the specified `amount` from the contract to the `recipient`. This step is critical for meeting the function’s requirements.

3. **Reference for Implementation**:
   Review the implementation in Aave's AaveEcosystemReserveV2.sol for guidance.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ERV-2 Incorrect Permission Verification Logic in `is_recipient` Function

Severity: Minor

Status: Fixed

Code Location:

aave-core/sources/aave-periphery/ecosystem_reserve_v2.move#84

Descriptions:

```
    fun is_recipient(account: address, stream_id: u256): bool acquires
EcosystemReserveV2Data {
        let ecosystem_reserve_v2_data =
            borrow_global<EcosystemReserveV2Data>
(ecosystem_reserve_v2_data_address());

        if (!smart_table::contains(&ecosystem_reserve_v2_data.streams, stream_id)) {
            return false
        };
        let stream_item =
            smart_table::borrow(&ecosystem_reserve_v2_data.streams, stream_id);
        let recipient = stream::recipient(stream_item);

        recipient != account
    }
```

the function contains a logical error in its return condition:

```
recipient != account
```

This condition incorrectly returns false when the account matches the recipient, and true otherwise.

Suggestion:

Correct the return condition in the is_recipient function to ensure accurate permission verification:

```
recipient == account
```

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Since the current function is not utilized in the present repository, the severity is categorized as minor.

FLO-1 Mock Contracts Should Not Be Used

Severity: Minor

Status: Fixed

Code Location:

aave-core/sources/aave-flash-loan/flash_loan.move#525

Descriptions:

Frequent use of mock contracts in smart contract development can introduce security risks, inconsistencies, maintainability issues, performance problems, and insufficient test coverage.

```
mock_underlying_token_factory::transfer_from(
    repayment_params.receiver_address,
    a_token_account_address,
    (amount_plus_premium as u64),
    repayment_params.asset,
);
```

Suggestion:

It is recommended that mock contracts are not used.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

GLO-1 The Token Price Used during Liquidation is not Up-to-date

Severity: Medium

Status: Fixed

Code Location:

aave-core/sources/aave-pool/generic_logic.move#122-127

Descriptions:

In the `calculate_user_account_data()` function, the protocol calls `oracle::get_asset_price()` to obtain the `asset_price`, which is then used to calculate `user_balance_in_base_currency` and `user_debt_in_base_currency`.

```
    vars.asset_price = if (vars.emode_asset_price != 0 && user_emode_category ==
(vars.emode_asset_category as u8)) {
            vars.emode_asset_price
        } else {
            oracle::get_asset_price(vars.current_reserve_address)
        };
```

However, in the `get_asset_price()` function, the protocol directly calls `get_pyth_price()` to retrieve the price.

```
public fun get_asset_price(asset: address): u256 acquires PythAssetPriceList {
    let asset_price_list = borrow_global<PythAssetPriceList>(@echo_oracle);
    if (!simple_map::contains_key(&asset_price_list.value, &asset)) {
        return 0
    };
    get_pyth_price(simple_map::borrow(&asset_price_list.value, &asset))
}
```

The issue here is that the protocol does not update the Pyth oracle's price before retrieving it, so there is a possibility of returning an outdated price.

While the protocol has a function, `set_asset_price()`, to update the price, this function can only be called by the admin.

```
public entry fun set_asset_price(
    account: &signer, asset: address, pyth_price_update: vector<vector<u8>>
) acquires PythAssetPriceList {
    // ensure only admins can call this method
    check_is_asset_listing_or_pool_admin(signer::address_of(account));
    let asset_price_list = borrow_global_mut<PythAssetPriceList>(@echo_oracle);
    assert!(simple_map::contains_key(&asset_price_list.value, &asset),
E_ASSET_NOT_EXISTS);
```

In the EVM ecosystem, such as in Aave V3, we observed that the protocol calls Chainlink's `latestAnswer()` to get the most recent price.

```
function getAssetPrice(address asset) public view override returns (uint256) {
  AggregatorInterface source = assetsSources[asset];

  if (asset == BASE_CURRENCY) {
    return BASE_CURRENCY_UNIT;
  } else if (address(source) == address(0)) {
    return _fallbackOracle.getAssetPrice(asset);
  } else {
    int256 price = source.latestAnswer();
    if (price > 0) {
      return uint256(price);
    } else {
      return _fallbackOracle.getAssetPrice(asset);
    }
  }
}
```

To resolve this, we recommend calling `pyth.get_price_no_older_than()` to ensure the latest price is used, or updating the price before using it.

Suggestion:

It is recommended to call `pyth.get_price_no_older_than()` to ensure the latest price is used, or updating the price before using it.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

MUT-1 Optimize `pow` Function with Fast Exponent

Severity: Minor

Status: Fixed

Code Location:

aave-core/aave-math/sources/math_utils.move#134-140

Descriptions:

```
public fun pow(base: u256, exponent: u256): u256 {
    let result = 1;
    for (_i in 0..exponent) {
        result = result * base;
    };
    result
}
```

The current pow function uses a loop with O(n) complexity to compute the power by iterating exponent times. This linear approach results in high gas costs for large exponents due to the repeated multiplications.

Suggestion:

Refactor the pow function to use fast exponentiation (exponentiation by squaring), which reduces the time complexity from O(n) to O(logn).

This optimization will decrease the number of operations required, effectively reducing gas consumption and making the function more efficient for larger exponents.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ORA-1 The Price Validity Period is Too Long

Severity: Critical

Status: Fixed

Code Location:

aave-core/aave-oracle/sources/oracle.move#34

Descriptions:

The contract uses the Pyth Oracle, but the price validity period is set too long. For example, with highly volatile token prices, this provides attackers with significant arbitrage opportunities.

Current setting:

```
const PYTH_MAX_SECONDS_OLD: u64 = 7200;
```

Suggestion:

Reduce the price validity period.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ORA-2 Different Tokens Should Have Different Validity Periods

Severity: Major

Status: Fixed

Code Location:

aave-core/aave-oracle/sources/oracle.move#34

Descriptions:

Currently, the contract applies the same validity period to all tokens. However, different tokens have varying levels of price volatility. For tokens with higher volatility, the validity period should be shorter, while for tokens with lower volatility, a longer validity period is acceptable. This approach helps ensure more accurate and timely price data.

Current Setting:

```
const PYTH_MAX_SECONDS_OLD: u64 = 7200;
```

Suggestion:

Use Different Validity Periods for Different Tokens Based on Their Volatility.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ORA-3 The Price Calculation is Incorrect

Severity: Major

Status: Fixed

Code Location:

aave-core/aave-oracle/sources/oracle.move#217-234

Descriptions:

In the `oracle.get_price()` function, the protocol first retrieves `price_positive`, then `expo_magnitude`, and finally returns `price_positive * pow(10, expo_magnitude)`. This is incorrect.

```
// construct the price
    let price_positive =
        if (i64::get_is_negative(&price::get_price(&price))) {
            i64::get_magnitude_if_negative(&price::get_price(&price))
        } else {
            i64::get_magnitude_if_positive(&price::get_price(&price))
        };
    let expo_magnitude =
        if (i64::get_is_negative(&price::get_expo(&price))) {
            i64::get_magnitude_if_negative(&price::get_expo(&price))
        } else {
            i64::get_magnitude_if_positive(&price::get_expo(&price))
        };
    (price_positive * pow(10, expo_magnitude),
        price::get_conf(&price),
        price::get_timestamp(&price))
```

On the Aptos chain, `price::get_expo(&price)` is generally negative.

https://pyth.network/price-feeds/crypto-apt-usd

If it's negative, the final price should be `price = price_positive / pow(10, expo_magnitude)`. If it's positive, multiplication should be used.

Suggestion:

It is recommended to account for scenarios where the `expo` value is either positive or negative.

Resolution:

This issue has been fixed, and the client has correctly calculated the price.

ORA-4 `aave_oracle` Lacks Support for Asset Removal

Severity: Minor

Status: Fixed

Code Location:

aave-core/aave-oracle/sources/oracle.move#147

Descriptions:

The `aave_oracle::add_asset` module currently provides the capability to add new assets to the oracle but does not support the removal of assets. This absence of functionality could result in difficulties when managing scenarios where an asset needs to be deprecated or removed from the supported list.

Suggestion:

Introduce a `remove_asset` function . This would enhance the flexibility and robustness of the oracle by allowing dynamic updates to the list of supported assets.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ORA-5 Unnecessary use of `borrow_global_mut`

Severity: Informational

Status: Fixed

Code Location:

aave-core/aave-oracle/sources/oracle.move#137,208;

aave-core/sources/aave-pool/pool.move#875

Descriptions:

```
fun get_oracle_base_currency(): Option<BaseCurrency> acquires OracleData {
    // get the oracle data
    let oracle_data = borrow_global_mut<OracleData>(@aave_oracle);
}
```

```
fun get_asset_identifier(asset: String): Option<vector<u8>> acquires OracleData {
    // check the asset is not the base currency
    let oracle_data = borrow_global_mut<OracleData>(@aave_oracle);
}
```

```
!smart_table::contains(&mut reserve_address_list.value, index)
```

In cases where `borrow_global_mut` is used without any actual data modification, it can create confusion by implying intent to alter data, leading to reduced code clarity and unnecessary mutable borrowing.

Suggestion:

Replace `borrow_global_mut` with `borrow_global` in such scenarios to enhance readability and avoid unnecessary mutable borrowing.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

POO-1 The Calculation of `total_debt_accrued` is Incorrect

Severity: Major

Status: Fixed

Code Location:

aave-core/sources/aave-pool/pool.move#804-816

Descriptions:

In the `accrue_to_treasury()` function, the protocol first calculates `prev_total_variable_debt`, then `curr_total_variable_debt`, and afterwards calculates `total_debt_accrued = curr_total_variable_debt - prev_total_variable_debt`.

```
        let prev_total_variable_debt =
            wad_ray_math::ray_mul(curr_scaled_variable_debt,
(reserve_data.variable_borrow_index as u256));

        let curr_total_variable_debt =
            wad_ray_math::ray_mul(curr_scaled_variable_debt,
(reserve_data.variable_borrow_index as u256), );

        let total_debt_accrued = curr_total_variable_debt - prev_total_variable_debt;
```

We found that when calculating both `prev_total_variable_debt` and `curr_total_variable_debt`, the protocol uses `curr_scaled_variable_debt * reserve_data.variable_borrow_index` for both. This causes `total_debt_accrued` to always be zero. https://github.com/aave/aave-v3-core/blob/master/contracts/protocol/libraries/logic/ReserveLogic.sol#L243-L250

Suggestion:

It is recommended to use the next `variableBorrowIndex` to calculate the current total debt.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

POO-2 There is an extra Comma in the Parameters when Calculating `curr_total_variable_debt`

Severity: Minor

Status: Fixed

Code Location:

aave-core/sources/aave-pool/pool.move#810-814

Descriptions:

In the `accrue_to_treasury()` function, the protocol calculates `curr_total_variable_debt` in this way.

```
        let curr_total_variable_debt =
            wad_ray_math::ray_mul(curr_scaled_variable_debt,
(reserve_data.variable_borrow_index as u256), );
```

We found that there is an extra comma in the parameters of `wad_ray_math::ray_mul()` .

Suggestion:

It is recommended to remove this comma.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

RCO-1 Logical Error in Branch Conditions of `claim_rewards_internal` Function

Severity: Critical

Status: Fixed

Code Location:

aave-core/sources/aave-periphery/rewards_controller.move#537-539

Descriptions:

In the `claim_rewards_internal` function of `aave_pool::rewards_controller`, the branch conditions for two mutually exclusive actions are identical. Specifically:

```
if (smart_table::contains(
    &rewards_controller_data.pull_rewards_transfer_strategy_table, reward
)) {
    transfer_pull_rewards_transfer_strategy_rewards(
        caller,
        to,
        reward,
        total_rewards,
        rewards_controller_data
    )
} else if (smart_table::contains(
    &rewards_controller_data.pull_rewards_transfer_strategy_table, reward
)) {
    transfer_staked_token_transfer_strategy_rewards(
        caller,
        to,
        reward,
        total_rewards,
        rewards_controller_data
    )
};
```

This causes the second branch to never execute, as the condition for the first branch will always match when the second would.

The same issue also exists in the `claim_all_rewards_internal` function.

```
if (smart_table::contains(
            &rewards_controller_data.pull_rewards_transfer_strategy_table,
            *vector::borrow(&rewards_list, i)
        )) {
            transfer_pull_rewards_transfer_strategy_rewards(
                caller,
                to,
                *vector::borrow(&rewards_list, i),
                *vector::borrow(&claimed_amounts, i),
                rewards_controller_data
            )
        } else if (smart_table::contains(
            &rewards_controller_data.pull_rewards_transfer_strategy_table,
            *vector::borrow(&rewards_list, i)
        )) {
            transfer_staked_token_transfer_strategy_rewards(
                caller,
                to,
                *vector::borrow(&rewards_list, i),
                *vector::borrow(&claimed_amounts, i),
                rewards_controller_data
            )
        };
```

Suggestion:

Adjust the conditional logic to ensure each branch corresponds to a distinct scenario.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SLO-1 Incorrect Borrowing Status Update After Full Repayment

Severity: Major

Status: Fixed

Code Location:

aave-core/sources/aave-supply-borrow/supply_logic.move#325

Descriptions:

In the `internal_repay` function, after a user fully repays their debt, the code incorrectly sets the user's borrowing status to `true` using the `set_borrowing` function. This indicates that the user is still considered to have an outstanding loan, even though the `variable_debt` has been fully repaid. The correct logic should set the borrowing status to `false`, reflecting that the user has no remaining debt. This error could lead to issues such as incorrect interest calculations, inaccurate debt tracking, and other logic errors that depend on the user's borrowing status.

```
        if (variable_debt - payback_amount == 0) {
            user_config::set_borrowing(&mut user_config_map,
(pool::get_reserve_id(&reserve_data) as u256), true);
            pool::set_user_configuration(on_behalf_of, user_config_map);
        };
```

Suggestion:

It is recommended to modify the code to set the user’s borrowing status to false when the variable_debt is fully repaid.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

TBA-1 `transfer` Transfers an Incorrect Amount

Severity: Critical

Status: Fixed

Code Location:

aave-core/sources/aave-tokens/token_base.move#478

Descriptions:

In the `transfer` function of the `token_base` contract, the wrong `amount` is being transferred:

```
fungible_asset::transfer_with_ref(
    transfer_ref,
    from_wallet,
    to_wallet,
    (amount as u64)
);
```

The correct amount to transfer should be `amount_ray_div`. Using the incorrect `amount` causes transfer failures, leading to issues such as failed liquidations when accepting AToken, among others.

Suggestion:

Update the code to transfer `amount_ray_div` instead of `amount`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

TBA-2 The implementation of `token_base` is incorrect

Severity: Major

Status: Fixed

Code Location:

aave-core/sources/aave-tokens/token_base.move#311;

aave-core/sources/aave-tokens/token_base.move#368

Descriptions:

In the `token_base` contract, the `mint_scaled` and `burn_scaled` functions calculate the mint and burn amounts based on `amount` instead of `amount_scaled`. This can often cause repayment failures. During repayment, the interest charged can result in more FA being burned than minted, leading to transaction failures.

```
let fa = fungible_asset::mint(&managed_fungible_asset.mint_ref, (amount as u64));
```

```
fungible_asset::burn_from(burn_ref, from_wallet, (amount as u64));
```

Suggestion:

Update the `mint_scaled` and `burn_scaled` functions to calculate the mint and burn amounts based on `amount_scaled`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

TBA-3 The `set_frozen_flag()` Function Call is Missing in the `transfer()` Function

Severity: Medium

Status: Fixed

Code Location:

aave-core/sources/aave-tokens/token_base.move#397-473

Descriptions:

In the `mint_scaled()` function, the protocol calls `fungible_asset::set_frozen_flag()` to disable the store's ability to perform direct transfers of the fungible asset. After that, the asset is minted to the `to_wallet` .

```
// freeze account
    fungible_asset::set_frozen_flag(
        &managed_fungible_asset.transfer_ref, to_wallet, true
    );

    let fa = fungible_asset::mint(&managed_fungible_asset.mint_ref, (amount as u64));
    fungible_asset::deposit_with_ref(
        &managed_fungible_asset.transfer_ref, to_wallet, fa
    );
```

However, in the `transfer()` function, the protocol does not call `set_frozen_flag()` to disable the recipient wallet's store's ability to perform direct transfers of the fungible asset. This creates a potential inconsistency, where the asset's transfer to the recipient is not accompanied by the same restriction as applied to the sender's wallet, allowing the recipient to transfer the asset without the intended limitations.

```
    // transfer fungible asset
    let asset = get_metadata(metadata_address);
    let transfer_ref = &obtain_managed_asset_refs(asset).transfer_ref;
    let from_wallet = primary_fungible_store::primary_store(sender, asset);
    let to_wallet =
        primary_fungible_store::ensure_primary_store_exists(recipient, asset);
    fungible_asset::transfer_with_ref(
```

```
        transfer_ref, from_wallet, to_wallet, (amount as u64)
    );
```

Suggestion:

It is recommended to call the `set_frozen_flag()` function in the `transfer()` function before transferring the asset to the `to_wallet`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

TBA-4 Evading Debt

Severity: Medium

Status: Fixed

Code Location:

aave-core/sources/aave-tokens/token_base.move#307-309

Descriptions:

In the protocol, the function `mint_scaled` utilizes `fungible_asset::set_frozen_flag` to restrict accounts from transferring funds within the FungibleStore. The function `set_frozen_flag` intended to enable or disable a store's ability to perform direct transfers of the fungible asset. However, despite this restriction being enforced on the primary FungibleStore by default, it doesn't adequately address the transfer capabilities of secondary stores. This loophole allows the potential circumvention of the controls by transferring a token and variable token through secondary stores, thus creating a possible avenue for users to evade liabilities.

A more robust approach to addressing this vulnerability is to utilize the `set_untransferable` function when creating a token. By doing so, all stores of the fungible asset are set to be untransferable, effectively preventing any transfers from one account to another, and thus precluding the possibility of bypassing the frozen flag. Here's how the corrected code would implement this solution:

```
// Set ALL stores for the fungible asset to untransferable.
// This preemptively blocks the ability of any store to be transferred between accounts,
// ensuring the effective utilization of the frozen flag to restrict unauthorized fund transfers.
fungible_asset::set_untransferable(constructor_ref);
```

This approach ensures that the susceptibility concerning fund transfers within secondary stores is properly mitigated, thereby enhancing the security of the protocol against potential evasion of debt obligations.

Suggestion:

This issue has been fixed. The client has adopted our suggestions.

Resolution:

The issue has not been fixed in the `create_variable_token` function.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.