Echo Lending

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Mon Sep 09 2024

Echo Lending Audit Report

1 Executive Summary

1.1 Project Information

Description	A decentralized finance protocol for lending and collateralization
Type	Lending
Auditors	MoveBit
Timeline	Mon Aug 12 2024 - Mon Sep 09 2024
Languages	Move
Platform	Aptos
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/echo-proto/echo-lending.git
Commits	0d29b9b5c5586cd278ac545b2ddee3d7a5365878 264dca3672effdf7875c84e74942992b02c83c3f ced8f4e1155d76b3f1ff066acf9abef2053adb54 a0b7d12d8793643bbe097c7b37f9349ac61cf4c5 cef2c6376e0f0c21b92589248f039bacc0a8c3b9 33a829faae3397c7edd46efe3103aca09688a4c6 2157f7f028f118b93fcfff37950addc2d690272c

1.2 Files in Scope

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

ID	File	SHA-1 Hash
AMA	echo-core/echo-acl/sources/acl_manage.move	d11c8710a44cc1a3bd54fdee6ed8ad3c25dda5d6
MUT1	echo-core/echo-math/sources/math_utils.move	d14bcb4a1c61f397b285bcc4106e254a110df1fe
WRM	echo-core/echo-math/sources/wad_ray_math.move	cfb33a69eb48260903459f0b77f21c884c473314
ATF	echo-core/echo-tokens/sources/a_token_factory.move	f764f79609b9f6f47fc2a73fe1681f201b8413c8
VTF	echo-core/echo-tokens/sources/variable_token_factory.move	d42586572d0f27b5d6783e7bbee69638af718b63
TCA	echo-core/echo-tokens/sources/token_cap.move	5e8c628da41477cdcfccea5f77b2b8ba02dc575a
RCO	echo-core/echo-config/sources/reserve_config.move	50510d5567267f0c241d43999766b32f6c60f2ec
UCO	echo-core/echo-config/sources/user_config.move	be44ef1cc81b0a974862e722adb791092ce38750
PDP	echo-core/echo-router/sources/pool_data_provider.move	bf5e16832954b73a153a0d535a3c6fa13cdc33b9
PAP	echo-core/echo-router/sources/pool_addresses_provider.move	1df70a6fbda23bf7973a1d9d230303dc3eb6c7df
FLO	echo-core/sources/echo-flash-loan/flash_loan.move	e34366fa82dca899a14e1890eb5e902155087d00

TAC	echo-core/sources/token_access.move	a5b7034a0dcd85c52dc2c4eb7ff6d6e2207795d9
PCO	echo-core/sources/echo-pool/pool_configurator.move	43473c6d345feb4072a06c813d1d97c3aba8ae26
VLO	echo-core/sources/echo-pool/validation_logic.move	9f9d548a1a1bf0be25b4a20766a3653c1fa0f88d
IML	echo-core/sources/echo-pool/isolation_mode_logic.move	e2c87bf9180a1ac35cb228732567c99556ada0f2
BLO	echo-core/sources/echo-bridge/bridge_logic.move	2254ddee84385d080efd6334d0654591bdd59ea5
ULO	echo-core/sources/echo-supply-borrow/user_logic.move	03635f85adff8de16563ad7fd92bba4abcd8fa4a
ICA	echo-core/echo-scripts/sources/issue_cap.move	14edaff16c3d0f15470119b5bb2f158a4cec178e
PPU	echo-core/echo-scripts/sources/post-publish.move	3cf1ceffe94b945b28386277dacb0b77379a0b9c
STO	echo-core/echo-tokens/sources/standard_token.move	46137378235e11549f364937c008da6b1823a86c
GLO	echo-core/sources/echo-pool/generic_logic.move	40dec00c2c422dd24dd4b9d63dadd84cde413810
ORA	echo-core/echo-oracle/sources/oracle.move	6a99d2175ac49b9ef6c3c3f161c5e211d321bbac
TBA	echo-core/echo-tokens/sources/token_base.move	46d833c704bf2f695891f60a52c80024c4a8e1ed
TTR	echo-core/echo-tokens/sources/token_transfer.move	1add8161386b65c0ba3cd7d3969b4a94e242fe86

UTF	echo-core/echo-tokens/sources/underlying_token_factory.move	5bcc2660fa977f9c48e969a84b6c35baa8f45e4b
HEL	echo-core/echo-config/sources/helper.move	1925e1a4238eaf230058186f5d871efe364c3870
WHI	echo-core/echo-limiter/sources/whitelist.move	e32de68f9a5a8bb77b7e93820832aa59a93294f0
ROR	echo-core/echo-router/sources/router_oracle.move	99fe88f608393b6228d371c5f8c16e1ee5b544a5
ROU	echo-core/echo-router/sources/router.move	2bda9aa3c8b9a3e64ae894bbdeb41b71b447e35c
DRIRS	echo-core/sources/echo-pool/default_reserve_interest_rate_strategy.move	20196a2888d37a1083f369d0ca76f81774b88b77
POO	echo-core/sources/echo-pool/pool.move	606ac3f4ee154fc5d2cbbc802074f5f4826309ca
ELO	echo-core/sources/echo-pool/emode_logic.move	11dfdc65ed6b01f8b4316dc506e7c24333c62158
BLO1	echo-core/sources/echo-supply-borrow/borrow_logic.move	249002b5580bdb5bd781066f5cf2e306291c5814
ECO	echo-core/echo-config/sources/error_config.move	7e069716e486884d11d30edc68e7f7d90c313e6c
SLO	echo-core/sources/echo-supply-borrow/supply_logic.move	41bedeb8934a40826e630b75f1b409e7a6d445b8
ROU1	echo-core/sources/echo-supply-borrow/router.move	2eae9abc7d43e71a7effcb9e4a96557ee58c9c57
VLO1	echo-core/sources/echo-supply-borrow/validation_logic.move	72ee89e35709ceebc75e8512d70f75e86e9a412d

LLO	echo-core/sources/echo-supply-borrow/liquidation_logic.move	e5416a9ba6e2c3d861989d2b51aa2ef2664ea50b
RCO1	echo-core/sources/echo-supply-borrow/reward_controller.move	de226661ec5b98b24746108f94f085cfae4774f4

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	17	16	1
Informational	0	0	0
Minor	10	10	0
Medium	3	2	1
Major	4	4	0
Critical	0	0	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

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 Echo Protocol to identify any potential issues and vulnerabilities in the source code of the Echo Lending 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 17 issues of varying severity, listed below.

ID	Title	Severity	Status
BLO-1	Incorrect Borrowing Status Update After Full Repayment	Major	Fixed
GLO-1	The Token Price Used during Liquidation is not Up-to-date	Medium	Fixed
LLO-1	User Rewards are not Updated during Liquidation	Minor	Fixed
ORA-1	The Price Calculation is Incorrect	Major	Fixed
ORA-2	Use Dual Oracles to Avoid a Single Point of Failure	Medium	Acknowledged
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
POO-3	Lack of Parameter Validation	Minor	Fixed
RCO-1	The Calculation of `user_amount` is Incorrect	Minor	Fixed

RCO-2	The Computation of Accumulated Prize Is Incorrect	Minor	Fixed
RCO-3	After the Period Ends, Users can still Accumulate Rewards	Minor	Fixed
RCO-4	Users cannot Claim Rewards	Minor	Fixed
RCO-5	There is a Precision Loss in Calculating the Prize	Minor	Fixed
RCO-6	Whitelisted Users cannot be Removed	Minor	Fixed
TBA-1	Evading Debt	Medium	Fixed
VLO-1	Health Factor Conditional Error	Major	Fixed
VLO-2	Unused Flashloan Validation Functions	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Echo Lending 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 owner can finalize asset transfers between users within the pool through `finalize_transfer()`
- The owner 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()`

4 Findings

BLO-1 Incorrect Borrowing Status Update After Full Repayment

Severity: Major

Status: Fixed

Code Location:

echo-core/sources/echo-supply-borrow/borrow_logic.move#301

Descriptions:

In the `internal_repay_2` 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 modified the code to set the user’s borrowing status to false when the variable_debt is fully repaid.

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

Severity: Medium

Status: Fixed

Code Location:

echo-core/sources/echo-pool/generic_logic.move#106

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.

LLO-1 User Rewards are not Updated during Liquidation

Severity: Minor

Status: Fixed

Code Location:

echo-core/sources/echo-supply-borrow/liquidation_logic.move#323-344

Descriptions:

The `liquidation_coin()` function is used to liquidate a user. In this function, the liquidated user's assets are reduced, and the liquidator receives rewards and the liquidated assets.

```
public(friend) fun liquidation_coin<CoinTypeCollateral,CoinTypeDebt>(
        account: &signer,
        collateral_asset: address,
        debt_asset: address,
        user: address,
        debt_to_cover: u256,
        receive_a_token: bool,
    ) {
        let (a_token_address, actual_debt_to_liquidate, actual_collateral_to_liquidate) =
            liquidation_call_step_1(account, collateral_asset, debt_asset, user, debt_to_cover,
receive_a_token);

        if (!receive_a_token) {
            // unwrap CoinTypeCollateral asset to coin
            underlying_token_factory::unwrap<CoinTypeCollateral>(account,
(actual_collateral_to_liquidate as u64), collateral_asset);
        };

        // wrap CoinTypeDebt coin to asset
        underlying_token_factory::wrap<CoinTypeDebt>(account, (actual_debt_to_liquidate
as u64), debt_asset);

        liquidation_call_step_2(account, collateral_asset, debt_asset, user, receive_a_token,
            a_token_address, actual_debt_to_liquidate, actual_collateral_to_liquidate);
    }
```

However, when assets are decreased or increased, the user's rewards are not updated.This could result in the liquidated user having no assets left but still accumulating rewards.

Suggestion:

It is recommedned to update the user's reward during liquidation.

Resolution:

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

ORA-1 The Price Calculation is Incorrect

Severity: Major

Status: Fixed

Code Location:

echo-core/echo-oracle/sources/oracle.move#202

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. The client has adopted our suggestions.

ORA-2 Use Dual Oracles to Avoid a Single Point of Failure

Severity: Medium

Status: Acknowledged

Code Location:

echo-core/echo-oracle/sources/oracle.move#95-101

Descriptions:

In `oracle.get_asset_price()`, the protocol directly retrieves the price from Pyth.

```
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))
}
```

If Pyth fails or returns an inaccurate price, the protocol lacks a fallback oracle. In Aave V3's EVM code, if Chainlink returns an inaccurate price, the protocol uses a fallback oracle.

```
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);
    }
  }
}
```

Suggestion:

It is recommended to implement a dual-oracle system to cross-verify prices.

POO-1 The Calculation of `total_debt_accrued` is Incorrect

Severity: Major

Status: Fixed

Code Location:

echo-core/sources/echo-pool/pool.move#637

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:

echo-core/sources/echo-pool/pool.move#643

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 removed the comma.

POO-3 Lack of Parameter Validation

Severity: Minor

Status: Fixed

Code Location:

echo-core/sources/echo-pool/pool.move#144-253

Descriptions:

When executing the 'init reserve' operation, it's necessary to add checks to verify whether the asset interest rate parameters are configured. You can refer to the latest implementation in Aptos Aave's code

Suggestion:

It is recommended to add parameter validation for the asset interest rate.

Resolution:

This issue has been fixed. The client has added parameter validation for the asset interest rate.

RCO-1 The Calculation of `user_amount` is Incorrect

Severity: Minor

Status: Fixed

Code Location:

echo-core/sources/echo-supply-borrow/reward_controller.move#134-195

Descriptions:

In the `pull_user_state_for_farming_period` function, when `us.last_balance` is greater than or equal to `current_a_token_balance`, the following logic is executed:

```
period.user_amount = period.user_amount + (us.last_balance - current_a_token_balance); // It should subtract
us.last_balance = current_a_token_balance;
```

However, this calculation is incorrect. The correct calculation should be:

```
period.user_amount = period.user_amount - (us.last_balance - current_a_token_balance);
```

Suggestion:

It is recommended to change the calculation of `user_amount` to the correct logic.

Resolution:

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

RCO-2 The Computation of Accumulated Prize Is Incorrect

Severity: Minor

Status: Fixed

Code Location:

echo-core/sources/echo-supply-borrow/reward_controller.move#462-464

Descriptions:

The current code calculates the accumulated prize as follows: `us.accumulated_prize = us.accumulated_prize + time_passed * calculate_emission_rate(period);`

The function `calculate_emission_rate` is defined as:

```
inline fun calculate_emission_rate(period: &mut Period): u64 {
    period.total_prize / ((period.cap as u64) * (period.end - period.start))
}
```

In this calculation, the `accumulated prize` does not consider the user's `last_balance`, which should be a factor in determining the proportion of the prize the user is entitled to based on their stake in the period. The correct way to compute the `accumulated prize` should involve multiplying by the user's `last_balance` to get a fair and proportionate distribution of the prize based on the user's contribution or stake.

Suggestion:

It is recommended to modify the calculation of `accumulated prize` to include the user's `last_balance` in the formula.

Resolution:

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

RCO-3 After the Period Ends, Users can still Accumulate Rewards

Severity: Minor

Status: Fixed

Code Location:

echo-core/sources/echo-supply-borrow/reward_controller.move#281

Descriptions:

In the `reward_controller.update()` contract, if the period is not started or has ended, the protocol will not update the user's rewards.

```
        let period = vector::borrow_mut(&mut asset_config.periods, i);
         if (period.start > now_sec && period.end < now_sec) { // skip not-started or
already ended periods
            continue
         };
```

However, the current condition is incorrect; it should be `period.start > now_sec || period.end < now_sec`. If it uses `&&`, this condition will never be met. When `period.end < now_sec`, rewards can still be updated, allowing users to claim more rewards.

Suggestion:

It is recommended to change the code to: `period.start > now_sec || period.end < now_sec`.

Resolution:

This issue has been fixed,the client has modified this part of the logic.

RCO-4 Users cannot Claim Rewards

Severity: Minor

Status: Fixed

Code Location:

echo-core/sources/echo-supply-borrow/reward_controller.move#353

Descriptions:

The `claim()` function allows users to claim rewards. The condition for claiming rewards is that the period must have already started. However, the protocol's condition is incorrect,

```
// not started yet
    assert!(period.start > now_sec, ERR_ASSET_NO_REWARD);
    if (period.end < now_sec) {
        now_sec = period.end;
    };
```

it should be `period.start < now_sec`, otherwise, users will not be able to claim rewards.

Suggestion:

It is recommended to change `period.start > now_sec` to `period.start < now_sec`.

Resolution:

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

RCO-5 There is a Precision Loss in Calculating the Prize

Severity: Minor

Status: Fixed

Code Location:

echo-core/sources/echo-supply-borrow/reward_controller.move#625-627

Descriptions:

In the `calculate_prize()` function, the protocol calculates the prize as follows.

```
    inline fun calculate_prize(last_balance: u64, time_passed: u64, total_prize: u64, cap: u64, start: u64, end: u64): u64 {
        (((total_prize as u128) * (last_balance as u128) / (cap as u128) * (time_passed as u128) / ((end - start) as u128)) as u64)
    }
```

We know that division causes precision loss,performing division before multiplication amplifies this loss.

Suggestion:

It is recommended to calculate the prize as follows:

`prize=total_prize*last_balance*time_passed/(cap*(end - start))`

Resolution:

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

RCO-6 Whitelisted Users cannot be Removed

Severity: Minor

Status: Fixed

Code Location:

echo-core/sources/echo-supply-borrow/reward_controller.move#261

Descriptions:

The admin can call `pull_user_state_for_farming_period()` to add whitelisted users to `period.users`,

```
if (!smart_table::contains(&period.users, user)) {
            add_user(user, period, current_a_token_balance, now_sec);
        }
```

but there doesn't seem to be a way to remove a user from the whitelist.

Suggestion:

It is recommended to implement a method to remove whitelisted users.

Resolution:

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

TBA-1 Evading Debt

Severity: Medium

Status: Fixed

Code Location:

echo-core/echo-tokens/sources/token_base.move#228-272

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:

It is recommended to adopt the mitigation strategy described to ensure enhanced security of the protocol.

Resolution:

This issue has been fixed. The client has adopted the mitigation strategy described to ensure enhanced security of the protocol.

VLO-1 Health Factor Conditional Error

Severity: Major

Status: Fixed

Code Location:

echo-core/sources/echo-supply-borrow/validation_logic.move#320-321

Descriptions:

When executing the liquidation logic, the `validate_liquidation_call` function is invoked to check whether liquidation can proceed. Within this function, the following condition is evaluated:

```
assert!(health_factor < user_config::get_minimum_health_factor_liquidation_threshold(),
    error_config::get_eprice_oracle_sentinel_check_failed());
```

This means that liquidation can only occur if the `health_factor` is less than the `MINIMUM_HEALTH_FACTOR_LIQUIDATION_THRESHOLD`, which is set as a constant at:

```
const MINIMUM_HEALTH_FACTOR_LIQUIDATION_THRESHOLD: u256 =
950000000000000000;
```

However, liquidation should be allowed when the `health_factor` is less than 1. The current logic mistakenly restricts liquidation to cases where the `health_factor` is below the `MINIMUM_HEALTH_FACTOR_LIQUIDATION_THRESHOLD`.

Suggestion:

It is recommended to refer to the code implementation here: Aave V3 Validation Logic.

Resolution:

This issue has been fixed. The client has removed the conditional judgment here.

VLO-2 Unused Flashloan Validation Functions

Severity: Minor

Status: Fixed

Code Location:

echo-core/sources/echo-supply-borrow/validation_logic.move#19-40

Descriptions:

The functions `validate_flashloan_complex` and `validate_flashloan_simple` are implemented to perform flashloan validation checks within the protocol. However, since the protocol does not support flashloans, these functions are redundant and should not be present in the codebase.

```
    // Flashloan Validate
    public fun validate_flashloan_complex(
        reserve_data: &vector<ReserveData>, assets: &vector<address>, amounts:
&vector<u256>,
    ) {
        assert!(vector::length(assets) == vector::length(amounts),
            error_config::get_einconsistent_flashloan_params());
        for (i in 0..vector::length(assets)) {
            validate_flashloan_simple(vector::borrow(reserve_data, i));
        }
    }

    // Simple Flashloan Validation
    public fun validate_flashloan_simple(reserve_data: &ReserveData) {
        let reserve_configuration = pool::get_reserve_configuration_by_reserve_data(
            reserve_data);
        let (is_active, _, _, is_paused) = reserve_config::get_flags(&reserve_configuration);
        assert!(!is_paused, error_config::get_ereserve_paused());
        assert!(is_active, error_config::get_ereserve_inactive());
        let is_flashloan_enabled =
reserve_config::get_flash_loan_enabled(&reserve_configuration);
        assert!(is_flashloan_enabled, error_config::get_eflashloan_disabled());
    }
```

Suggestion:

It is recommended to remove the `validate_flashloan_complex` and `validate_flashloan_simple` functions from the codebase.

Resolution:

This issue has been fixed. The client has removed the `validate_flashloan_complex` and `validate_flashloan_simple` functions from the codebase.

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.