Echo Protocol LSD

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Mon Dec 23 2024

Echo Protocol LSD Audit Report

1 Executive Summary

1.1 Project Information

Description	An LSD protocol on Aptos that allows users to stake assets, earn staking rewards, and access liquidity through tokenized staking derivatives
Type	Liquid Staking Derivatives
Auditors	MoveBit
Timeline	Thu Nov 21 2024 - Mon Dec 23 2024
Languages	Move
Platform	Aptos
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/echo-proto/echo-lsd
Commits	4223109991c30d9abddd1c7b56e580bd3e1749b5 37f4c29f39970f5110d66746c3d5f72657b004f4 75b76a108d26143bd6c623e7fd2220c1f8eabf3d 15f37d0ff7b07deb5f923bfbbe3e6f5dc9eccb78 15f37d0ff7b07deb5f923bfbbe3e6f5dc9eccb78 2769ee4dc03221d1f58b7296dc870c19ebee0b34 262a2caa30ea561a829a68ab08f5296e3598aedf

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MOV	Move.toml	3d45e31d0be0ea5555da8113a8a2c3dc9389572d
EAP	sources/eapt.move	95e07bbe641a713a61f6b9980f9bc7583150faab
LSD	sources/lsd.move	8b57403ec690b51c4285cdf54c9c563f2e2c79a6

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	13	13	0
Informational	3	3	0
Minor	3	3	0
Medium	6	6	0
Major	1	1	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(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 Echo Protocol to identify any potential issues and vulnerabilities in the source code of the Echo Protocol LSD 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 13 issues of varying severity, listed below.

ID	Title	Severity	Status
LSD-1	Repeated Reward Claiming	Major	Fixed
LSD-2	The `update_amounts()` Function Should be a Private Function	Medium	Fixed
LSD-3	Unstake Within the Same Epoch Can Be Exploited for Profit	Medium	Fixed
LSD-4	Missing Reward Update	Medium	Fixed
LSD-5	Missing `update_amounts` Function Call	Medium	Fixed
LSD-6	Incorrect `add_stake` Amount	Medium	Fixed
LSD-7	Incorrect Implementation of `get_most_underutilized_pool` Function	Medium	Fixed
LSD-8	Add a Function to Remove Validator Pools	Minor	Fixed
LSD-9	Repeated Calls to `user_claim_withdraw_apt` Can Disrupt The Protocol	Minor	Fixed

MOV-1	No Upgrade Policies	Minor	Fixed
LSD-10	Unused Constants	Informational	Fixed
LSD-11	Condition Optimization	Informational	Fixed
LSD-12	Unused Private Functions	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Echo Protocol LSD Smart Contract :

User

- `stake` : Users can stake AptosCoin to receive liquid eAPT tokens and earn staking rewards.
- `unstake` : Users can request to redeem their staked AptosCoin by burning eAPT tokens.
- `user_claim_withdraw_apt` : Users can claim their unlocked AptosCoin after the withdrawal lock period ends.

Admin

- `attach_pool` : Adds a new validator pool to the staking system.
- `set_target_utilization` : Updates target utilization rates for validator pools.
- `enable_stake` : Enables or disables staking functionality.
- `enable_unstake` : Enables or disables unstaking functionality.
- `update_lock_time_for_withdraw` : Updates the unstaking lock time period.
- `update_min_amount` : Sets the minimum staking and unstaking amount.
- `update_batch_amount` : Updates the staking batch amount.
- `set_stake_APT_fee_rate` : Configures the fee rate for staking.
- `set_unstake_APT_fee_rate` : Configures the fee rate for unstaking.
- `set_commission_fee_rate` : Updates the commission fee rate for validators.
- `set_transaction_fee_addr` : Sets the address to receive transaction fees.
- `set_commission_fee_addr` : Sets the address to receive commission fees.
- `set_operator_addr` : Assigns a new operator address.
- `unlock_commission_fee` : Unlocks the commission fees for withdrawal.
- `claim_withdraw_commission_fee` : Withdraws the unlocked commission fees.

4 Findings

LSD-1 Repeated Reward Claiming

Severity: Major

Status: Fixed

Code Location:

sources/lsd.move#291-353

Descriptions:

The `user_claim_withdraw_apt` function is designed to allow users to claim `AptosCoin` after calling the `unstake` function and once the lock-up period has expired. It extracts a specified amount of `AptosCoin` based on the `amount` recorded in the `UnstakeRequest`. However, the function lacks a mechanism to check for duplicate usage of the `UnstakeRequest`. This means that users can repeatedly claim `AptosCoin` using the same `UnstakeRequest`.

```
struct UnstakeRequest has store, copy, drop {
    sender: address, // Address of the user making the request
    request_id: u64,
    amount: u64, // Amount of APT to be sent out
    unstake_fee: u64,
    end_timestamp: u64
}

struct UserUnstake has key {
    requests: smart_table::SmartTable<u64, UnstakeRequest>
}
```

Suggestion:

Remove the `UnstakeRequest` after the `AptosCoin` have been successfully claimed.

Resolution:

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

LSD-2 The `update_amounts()` Function Should be a Private Function

Severity: Medium

Status: Fixed

Code Location:

sources/lsd.move#355-375

Descriptions:

In the `sync_rewards()` function, the protocol calls `get_amounts()` before and after invoking `update_amounts()` to calculate `balance_with_rewards` and `inactive_amount`, and then computes the difference in `total_APT_amount` to update `global.total_rewards_amount`.

However, the issue is that `update_amounts()` is currently an entry function, which allows anyone to call it.

A malicious attacker could exploit this by repeatedly calling `update_amounts()`, manipulating updates to `balance_with_rewards` and `inactive_amount`. This could lead to incorrect calculations of `global.total_rewards_amount`.

```
public entry fun update_amounts() acquires GlobalManager {
    let global = borrow_global_mut<GlobalManager>(@echo_lsd);
    let config_object_signer =
        account::create_signer_with_capability(&global.resource_signer_cap);

    // Update the current balance of the vault including rewards
    global.current_balance_with_rewards =
        current_balance_with_rewards(
            signer::address_of(&config_object_signer),
            global.config.delegator_pools
        );

    // Update the total supply of LP tokens in the vault
    global.current_total_eAPT_amount = (option::destroy_some(eapt::supply()));

    global.current_inactive_amount =
        current_unlocked_balance(
            signer::address_of(&config_object_signer),
            &global.config.delegator_pools
```

```
        );
    }
```

Suggestion:

To prevent this vulnerability, `update_amounts()` should be changed to a private function to ensure that it can only be invoked by trusted functions within the protocol.

Resolution:

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

LSD-3 Unstake Within the Same Epoch Can Be Exploited for Profit

Severity: Medium

Status: Fixed

Code Location:

sources/lsd.move#218

Descriptions:

There is a vulnerability in the protocol's `Unstake` logic that allows users to profit from withdrawing within the same epoch.

For example, suppose initially:

`global.current_total_eAPT_amount = 1000`

`global.current_balance_with_rewards = 1000`

- **User1 stakes 100:**
  After staking:
  `global.current_total_eAPT_amount = 1000 + 100 = 1100`
  The `eapt_amount_to_mint` is calculated as 100.

- **User2 unstakes 500:**
  After unstaking:
  `global.current_total_eAPT_amount = 1100 - 500 = 600`.

Now, when User1 unstakes, the calculation for the amount of APT they will receive is:

```
APT = 100 * (1100 (or even greater) / 600) = 183
```

This means User1 gains 183 APT tokens, resulting in a profit of `183 - 100 = 83`.

Suggestion:

To prevent such exploitation, the protocol should record the unstaked amount during the `unstake` process to ensure accurate calculations and avoid unfair profits.

Resolution:

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

LSD-4 Missing Reward Update

Severity: Medium

Status: Fixed

Code Location:

sources/lsd.move#308

Descriptions:

if the `user_claim_withdraw_apt` function is called directly in a new epoch, rewards will be lost due to the `update_amounts` function. To resolve this, the sync_rewards function should be called at the beginning of the user_claim_withdraw_apt function to calculate the rewards.

Suggestion:

The `sync_rewards function` should be called at the beginning of the `user_claim_withdraw_apt` function to calculate the rewards.

Resolution:

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

LSD-5 Missing `update_amounts` Function Call

Severity: Medium

Status: Fixed

Code Location:

sources/lsd.move#308

Descriptions:

In the `user_claim_withdraw_apt()` function, the protocol invokes `withdraw_inactive_stake()` to withdraw inactive funds, causing a reduction in the amount of inactive funds. However, the protocol fails to call the `update_amounts()` function to update `global.current_inactive_amount`.

As a result, when the protocol calls `sync_rewards()` during unstaking, the updated `global.total_rewards_amount` becomes larger than expected. This discrepancy increases the value of `total_staked_apt()` and consequently inflates the withdrawal amount calculated by the protocol based on the user's LP. This allows users to withdraw more tokens than they are entitled to.

Suggestion:

Add a call to the `update_amounts` function in this section.

Resolution:

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

LSD-6 Incorrect `add_stake` Amount

Severity: Medium

Status: Fixed

Code Location:

sources/lsd.move#805

Descriptions:

In the `transfer_stake` function, if `pending_stake` exceeds `batch_amount`, the function adds the stake to the validator's staking pool and sets `pending_stake` to 0. However, the staked amount used is `stake_apt_amount`, whereas the correct staked amount should be `pending_stake`.

```
fun transfer_stake(stake_apt_amount: u64) acquires GlobalManager {
    let global = borrow_global_mut<GlobalManager>(@echo_lsd);
    let config_object_signer =
        account::create_signer_with_capability(&global.resource_signer_cap);
    let vault_addr = signer::address_of(&config_object_signer);

    if (global.pending_stake >= global.config.batch_amount) {
        let validator_address =
            get_most_underutilized_pool(vault_addr, global.target_utilization);

        // Add stake to the validator's staking pool
        dp::add_stake(&config_object_signer, validator_address, stake_apt_amount);

        global.pending_stake = 0;
    }
}
```

Suggestion:

Modify the code to use `pending_stake` as the staked amount instead of `stake_apt_amount`. This ensures that the correct amount is staked.

Resolution:

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

LSD-7 Incorrect Implementation of `get_most_underutilized_pool` Function

Severity: Medium

Status: Fixed

Code Location:

sources/lsd.move#887

Descriptions:

The `get_most_underutilized_pool` function is designed to retrieve the most underutilized pool. Consider the following code:

```
if (actual_utilization < expect_utilization
    && most_underutilized > expect_utilization - actual_utilization) {
    most_underutilized = expect_utilization - actual_utilization;
    result = *vector::borrow(&key_vec, count);
};
```

The current condition retrieves the pool if `most_underutilized > expect_utilization - actual_utilization`, which is incorrect. To correctly identify the most underutilized pool, the condition should be:

```
most_underutilized < expect_utilization - actual_utilization
```

This ensures that the pool with the largest difference between `expect_utilization` and `actual_utilization` (i.e., the least utilized pool) is selected.

Suggestion:

Update the condition in the function implementation as described above.

Resolution:

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

LSD-8 Add a Function to Remove Validator Pools

Severity: Minor

Status: Fixed

Code Location:

sources/lsd.move#436-461

Descriptions:

The `attach_pool` function is designed to add a validator pool, but it lacks a mechanism to remove one. If a validator pool exits staking or becomes inactive (e.g., goes down), it should be considered for removal. Before removing a pool, any remaining staked `AptosCoin` should be withdrawn to avoid loss of funds.

```
public entry fun attach_pool(
    sender: &signer, whitelist_address: address
) acquires GlobalManager {
    ensure_authorized(signer::address_of(sender));
    let global = borrow_global_mut<GlobalManager>(@echo_lsd);
    assert!(
        !simple_map::contains_key<address, u64>(
            &global.target_utilization, &whitelist_address
        ),
        ERR_POOL_ALREADY_EXIST
    );
    assert!(
        dp::delegation_pool_exists(whitelist_address), ERR_POOL_NOT_EXIST_IN_DP
    );
    if (simple_map::length<address, u64>(&global.target_utilization) == 0) {
        simple_map::add<address, u64>(
            &mut global.target_utilization, whitelist_address, FEE_DENOMINATOR
        );
    } else {
        simple_map::add<address, u64>(
            &mut global.target_utilization, whitelist_address, 0
        );
    };
```

```
    vector::push_back(&mut global.config.delegator_pools, whitelist_address);
}
```

Suggestion:

- Introduce a `remove_pool` function to handle pool removals.
- Ensure that prior to removal, the remaining staked `AptosCoin` is securely withdrawn and transferred back to the appropriate entity.

Resolution:

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

LSD-9 Repeated Calls to `user_claim_withdraw_apt` Can Disrupt The Protocol

Severity: Minor

Status: Fixed

Code Location:

sources/lsd.move#307

Descriptions:

In the `user_claim_withdraw_apt` function, if the condition `if (current_balance < withdraw_amount) { break; }` is triggered, users can repeatedly invoke `user_claim_withdraw_apt`. Each invocation results in the `unstake_fee_apt_coin` being transferred from the `config_object_signer` to the `transaction_fee_addr`, which is unreasonable and could lead to unintended consequences.

Suggestion:

It is recommended to use `assert` instead of `break`.

Resolution:

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

MOV-1 No Upgrade Policies

Severity: Minor

Status: Fixed

Code Location:

Move.toml#1-5

Descriptions:

The protocol has no escalation measures, which may result in the contract not being able to be used under a single, stable, well-known account address that doesn't change.

https://aptos.dev/en/build/smart-contracts/book/package-upgrades

Suggestion:

It is recommended to add upgrade_policy = "compatible" to the Move.toml file.

Resolution:

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

LSD-10 Unused Constants

Severity: Informational

Status: Fixed

Code Location:

sources/lsd.move#29

Descriptions:

There are unused constants in the contract.

```
const ERR_INVALID_ADDRESS: u64 = 3;
const ERR_DEPOSIT_CAP: u64 = 6;
const ERR_NONE_POOL_WITHDRAWN: u64 = 8;
const ERR_UNAUTHORIZED_POOL: u64 = 12;
```

Suggestion:

It is recommended to remove unused constants if there's no further design.

Resolution:

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

LSD-11 Condition Optimization

Severity: Informational

Status: Fixed

Code Location:

sources/lsd.move#169-173

Descriptions:

In the `stake` function, there is a balance check for the user's account as follows:

```
// Assert if the sender's account balance of APT meets the minimum staking amount
assert!(
    coin::balance<AptosCoin>(signer::address_of(sender))
        >= global.config.min_amount,
    ERR_MIN_THRESHOLD
);
```

A better condition would be to check if the user's balance is greater than `input_amount`.

Suggestion:

Instead of checking if the user's balance meets `min_amount`, it is recommended to verify whether the user's balance exceeds `input_amount`.

Resolution:

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

LSD-12 Unused Private Functions

Severity: Informational

Status: Fixed

Code Location:

sources/lsd.move#896-995

Descriptions:

The functions `get_total_stakes` and `get_pools_reward_rate` are private and not used anywhere in the codebase. Additionally, the private function `get_stakes` is only called within `get_total_stakes`.

Suggestion:

- Evaluate whether these functions are necessary. If they are not required, consider removing them to reduce code clutter.
- If these functions are intended to provide reusable logic or information to external callers, convert them into `public view` functions for better accessibility.

Resolution:

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

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.