Lotus Finance Smart Contract

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Tue May 27 2025

Lotus Finance Smart Contract Audit Report

1 Executive Summary

1.1 Project Information

Description	Lotus Finance is a trading strategy protocol built on SUI
Type	DeFi
Auditors	MoveBit
Timeline	Fri Jan 17 2025 - Tue May 27 2025
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/0xhipo/lotus-finance-contract
Commits	1a775ba78347ed9e4bbfc0510860ac54a94a6d49 8e68c1e5228cfadd17a8530474be540428559643 79a0b7bff357ad8ac07ff310b4c53f4f0ee6ca76 018e439d8f0409fd5d5b278239d2761bf0902710 7c80d0751531954edca35d89111b105731a79fde 6fe035304c835079fe2da3f8b190530ac6d3dbba

1.2 Files in Scope

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

ID	File	SHA-1 Hash
LTO	packages/lotus_finance/sources/lp_token/lp_token.move	f4d2810f3d07036c1f4b96f8c201cd4cf8fa3681
LCO	packages/lotus_finance/sources/lotus_config.move	60b18005b8fdba5c3a0de4fd971a089643f38505
CON	packages/lotus_finance/sources/consts.move	3e8c34f31b2d56bd4e567781cb098f1786996ef2
MOV2	packages/lotus_finance/Move.toml	375c3ab6d2b3216d2b833eb9abfe296f7a00eb95
LLF	packages/lotus_finance/sources/farm/lotus_lp_farm.move	618b27b3106e68acd00757e2946b1c65e7f58afe
OAG	packages/lotus_finance/sources/oracle/oracle_ag.move	e5012988b90a6a4ef3b3f28f09777553a4c8560d
LDV	packages/lotus_finance/sources/vault/lotus_db_vault.move	8b2f26a83ded6a87f850fb4b32eccf3d6e0b0bec
MAT	packages/lotus_finance/sources/helper/math.move	4221e3dfd1a0faac393226d7d154423fed89938e

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	12	8	4
Informational	3	1	2
Minor	5	3	2
Medium	0	0	0
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(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 Lotus to identify any potential issues and vulnerabilities in the source code of the Lotus Finance 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 12 issues of varying severity, listed below.

ID	Title	Severity	Status
LDV-1	Lack of Pairing Checks Between `LotusDBVault` and `LotusDBVaultCap` Causes Security Risks	Major	Fixed
LDV-2	Security Flaw in Deposit-to-Mint LP Token Mechanism	Minor	Acknowledged
LDV-3	Permission Logic of Vault Deposit Function Mismatches Comment Description	Informational	Fixed
LDV-4	Replace the Old Syntax with the New Syntax	Informational	Acknowledged
LLF-1	Missing Pairing Check Between `LotusLPFarm` and `LotusLPFarmCap` Leads to Permission Abuse	Major	Fixed
LLF-2	Security Vulnerabilities in `LotusLPFarmUpdatePoolWeightTic`	Major	Fixed

	`ket` Management		
LLF-3	OracleAggregator Forgery Vulnerability Enables Price Manipulation	Major	Fixed
LTO-1	The Metadata in Currency should be Frozen	Minor	Fixed
OAG-1	Different Assets should Have Different `max_age`	Minor	Fixed
OAG-2	The case where the Expo is Positive is not Taken into Account	Minor	Fixed
OAG-3	Redundant Code	Informational	Acknowledged
TEL-1	Unbound `LotusLPFarmCreatePoolTicket` Types Cause LP Token Minting Vulnerability	Minor	Acknowledged

3 Participant Process

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

Admin

- Admin can add a `TD_Farm` in the `LotusLPFarm` through `add_td_farm<LP, Incentive>()`.
- Admin can set the `TD_Farm` unlock rate through `set_farm_unlock_rate<LP, Incentive>()`.
- Admin can add the allowed deposit asset through `add_allowed_deposit_asset<LP, T>()`.
- Admin can new an `Lp` teller through `new<LP>()`.
- Admin can update the pyth oracle price id in the `OracleAggregator` through `update_pyth_price_id<T>()`.

User

- User can create an incentivized vault and deposit coins in through `create_incentivized_db_vault<LP, T>()`.
- User can top up the td pool through `top_up_to_td_pool<LP, Incentive>()`.
- User can update the td pool weight through `update_vault_weight_permissionless<LP, Incentive, Base, Quote>()`.
- User can collect the incentive rewards to the vault through `collect_incentive_rewards_to_vault<LP, Incentive>()`.
- User can create an `LotusDBVault<LP>` through `new<LP>()`.
- User can mint the lotus trade cap through `mint_lotus_trade_cap<LP>()`.
- User can deposit coins in the `LotusDBVault<LP>` through `deposit<T, LP>()`.
- User can withdraw coins from the `LotusDBVault<LP>` through `redeem_all<LP, T>()`.
- User can redeem the incentive coins through `redeem_incentive<LP, Incentive>()`.

- User can withdraw the settled coins in the `DBPool` through `withdraw_settled_amounts<LP, Base, Quote>()`.
- User can place a limit order in the `DBPool` through `place_limit_order<LP, Base, Quote>()`.
- User can place a market order in the `DBPool` through `place_market_order<LP, Base, Quote>()`.
- User can cancel an order in the `DBPool` through `cancel_order<LP, Base, Quote>()`.
- User can cancel orders in the `DBPool` through `cancel_orders<LP, Base, Quote>()`.
- User can cancel all orders in the `DBPool` through `cancel_all_orders<LP, Base, Quote>()`.
- User can stake `LP` coins and get a `Stake` object through `deposit_shares_new<LP>()`.
- User can restake `LP` coins in the `Stake` object through `deposit_shares<LP>()`.

4 Findings

LDV-1 Lack of Pairing Checks Between `LotusDBVault` and `LotusDBVaultCap` Causes Security Risks

Severity: Major

Status: Fixed

Code Location:

packages/lotus_finance/sources/vault/lotus_db_vault.move#163,191,226,255,269,284

Descriptions:

In the `lotus_db_vault` module, there is a lack of pairing checks between `LotusDBVault<LP>` and `LotusDBVaultCap` objects. This allows any generated `LotusDBVaultCap` to control `LotusDBVault<LP>` objects. For example, in the `redeem_incentive` function, attackers can call the `new<LP>` function to generate a new pair of `LotusDBVault<LP>` and `LotusDBVaultCap` objects. Since the `redeem_incentive` function lacks pairing checks, attackers can use their self-generated `LotusDBVaultCap` to manipulate the project's `LotusDBVault<LP>` object and extract all reward tokens from the vault.

Suggestion:

It is recommended to implement pairing checks between `LotusDBVault<LP>` and `LotusDBVaultCap` in the module to ensure that only the `LotusDBVaultCap` object paired with the target `LotusDBVault<LP>` can invoke the function.

Resolution:

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

LDV-2 Security Flaw in Deposit-to-Mint LP Token Mechanism

Severity: Minor

Status: Acknowledged

Code Location:

packages/lotus_finance/sources/vault/lotus_db_vault.move#153

Descriptions:

We identified a flaw in the "Deposit-to-Mint LP Tokens" mechanism in the protocol, which could lead to an unlimited minting arbitrage risk. In the protocol, users can call the `create_incentivized_db_vault` function to deposit Coins into a `Vault` for strategy investments. This function generates a deposit certificate that allows users to mint `LP` tokens equivalent to the deposit amount. However, the withdrawal functionality seems to lack proper restrictions, enabling users to deposit funds, mint `LP` tokens, and then withdraw their deposits repeatedly. By looping this process, users can achieve cost-free arbitrage, potentially causing significant losses to the protocol's assets.

Suggestion:

It is recommended to introduce a lock-in period for deposited funds to prevent frequent deposit and withdrawal operations.

Resolution:

The client replied that the number of all LP tokens minted from Farm is based on the value of the user's deposit, and the LP is only used for data storage and has no impact on the protocol.

LDV-3 Permission Logic of Vault Deposit Function Mismatches Comment Description

Severity: Informational

Status: Fixed

Code Location:

packages/lotus_finance/sources/vault/lotus_db_vault.move#122

Descriptions:

The `deposit` function is intended to allow tokens to be deposited into the vault, and its comment states `Anyone can deposit`. However, in reality, only the creator of the vault can call this function. This is because the `deposit` function relies on the `deposit` method from the `deepbook::balance_manager` module, which has built-in permission checks that restrict its use to the `BalanceManager` object's creator. As a result, regular users may not be able to use this function to deposit tokens into the vault after the contract is deployed, contradicting the comment and user expectations.

Suggestion:

It is recommended to confirm it aligns with your design.

Resolution:

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

LDV-4 Replace the Old Syntax with the New Syntax

Severity: Informational

Status: Acknowledged

Code Location:

packages/lotus_finance/sources/vault/lotus_db_vault.move

Descriptions:

Currently, the Sui contract has adopted the new syntax, such as `module lotus_finance::lotus_db_vault;` . However, the protocol is still using the old syntax in some parts of the codebase.

```
module lotus_finance::lotus_db_vault {
    use 0x1::u64;
    use std::debug;
    use std::type_name::{
```

Suggestion:

It is recommended to update the protocol to use the new syntax

LLF-1 Missing Pairing Check Between `LotusLPFarm` and `LotusLPFarmCap` Leads to Permission Abuse

Severity: Major

Status: Fixed

Code Location:

packages/lotus_finance/sources/farm/lotus_lp_farm.move#110,117

Descriptions:

In the `set_farm_unlock_rate` and `add_allowed_deposit_asset` functions, there is a lack of pairing checks between `LotusLPFarm<LP>` and `LotusLPFarmCap`. Since `LotusLPFarmCap` lacks generic controls, any type of `LotusLPFarmCap` has control over `LotusLPFarm`. This allows anyone to modify the farm's unlock rate and add asset types, posing a security risk and potential abuse.

Suggestion:

It is recommended to add pairing checks between `LotusLPFarm<LP>` and `LotusLPFarmCap` or generic control for the `LotusLPFarmCap`.

Resolution:

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

LLF-2 Security Vulnerabilities in `LotusLPFarmUpdatePoolWeightTicket` Management

Severity: Major

Status: Fixed

Code Location:

packages/lotus_finance/sources/farm/lotus_lp_farm.move#124,145,139

Descriptions:

In the `lotus_lp_farm` module, some functions rely on the "Hot Potato" characteristic of `LotusLPFarmUpdatePoolWeightTicket` certificates to ensure atomic operations. However, since the creation and destruction of these certificates are managed by public functions, the following security risks arise:

1. Attackers can create tickets using the `new_create_pool_ticket` function, mint arbitrary amounts of `LP` tokens by calling `mint_lp_token_with_ticket` in the `teller` module, and then destroy the ticket using `add_incentivized_db_vault_to_td_farm_with_ticket` and `destroy_create_pool_ticket`. This results in an arbitrary mint attack.
2. Attackers can bypass normal deposit logic to directly manipulate the pool's weight values, leading to inaccuracies in weight calculations and other risks.

Suggestion:

It is recommended to modify the visibility of functions such as `new_create_pool_ticket` and `destroy_create_pool_ticket` to internal or package to prevent external calls and review other public functions related to tickets to ensure strict access control.

Resolution:

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

LLF-3 OracleAggregator Forgery Vulnerability Enables Price Manipulation

Severity: Major

Status: Fixed

Code Location:

packages/lotus_finance/sources/farm/lotus_lp_farm.move#169,244;

packages/lotus_finance/sources/oracle/oracle_ag.move#58

Descriptions:

In the `oracle_ag` module, users can create `OracleAggregator` and `OracleAggregatorCap` objects by calling the new function. This allows attackers to use forged `OracleAggregator` instances within the protocol. For instance, in the `lotus_lp_farm` contract, the `update_vault_weight_permissionless` function relies on a provided `OracleAggregator` to validate the `PriceInfoObject` of three token types, fetch their prices, calculate the total account value, and update the vault weight. However, since attackers can pass in a forged `OracleAggregator`, price manipulation becomes possible. For example, an attacker could set the `USDT` token's `PriceInfoObject` to represent a `BTC` token type, leading to inaccurate price data and incorrect vault weight updates.

Suggestion:

It is recommended to add access control to the `new` function to restrict the creation of `OracleAggregator` and `OracleAggregatorCap` objects to authorized roles only or change the visibility of the `new` function.

Resolution:

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

LTO-1 The Metadata in Currency should be Frozen

Severity: Minor

Status: Fixed

Code Location:

packages/lotus_finance/sources/lp_token/lp_token.move#10-24

Descriptions:

In the `init()` function, the protocol invokes `transfer::public_share_object()` to share the `meta_data` object.

```
fun init(otw: LP_TOKEN, ctx: &mut TxContext) {
    let (treasury_cap, deny_cap, meta_data) = coin::create_regulated_currency(
        otw,
        6,
        b"Lotus LP Token",
        b"LFLP",
        b"LP token for Lotus Finance",
        option::none(),
        ctx,
    );

    let sender = ctx.sender();
    transfer::public_transfer(deny_cap, sender);
    transfer::public_transfer(treasury_cap, sender);
    transfer::public_share_object(meta_data);
}
```

This action grants the individual holding the treasury cap unrestricted access to modify critical information within the meta_data object, such as the name, symbol, and other associated details.

Suggestion:

It is recommended to freeze the `meta_data` instead of sharing.

```
transfer::public_freeze_object(meta_data);
```

Resolution:

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

OAG-1 Different Assets should Have Different `max_age`

Severity: Minor

Status: Fixed

Code Location:

packages/lotus_finance/sources/oracle/oracle_ag.move#73

Descriptions:

In the new `function()`, the protocol configures the `max_age` parameter to define the expiration time for oracle prices.

```
fun new(ctx: &mut TxContext): (OracleAggregator, OracleAggregatorCap) {
    let ag = OracleAggregator {
        id: object::new(ctx),
        pyth_price_id_table: table::new(ctx),
        coin_decimal_registry: table::new(ctx),
        max_age: GET_PYTH_MAX_AGE(),
    };
    let ag_cap = OracleAggregatorCap {
        id: object::new(ctx),
        oracle_aggregator: object::id(&ag),
    };
    (ag, ag_cap)
}
```

While the protocol supports multiple assets, it currently assigns the same `max_age` value to all assets, which is not ideal. This approach is problematic because different assets may have varying levels of price stability. For instance, stable assets typically experience minimal price fluctuations, whereas more volatile assets can have significant price changes over short periods. Therefore, using a uniform `max_age` for all assets is incorrect, as it does not account for the differing price stability characteristics of each asset.

Suggestion:

It is recommended to set asset-specific `max_age` values to better reflect their individual price behaviors.

Resolution:

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

OAG-2 The case where the Expo is Positive is not Taken into Account

Severity: Minor

Status: Fixed

Code Location:

packages/lotus_finance/sources/oracle/oracle_ag.move#119-142

Descriptions:

In the `get_price()` function, the protocol retrieves the price decimal and timestamp from the Pyth oracle.

```
let price_i64 = price::get_price(&price_struct);
        let decimal_i64 = price::get_expo(&price_struct);
        let timestamp_sec = price::get_timestamp(&price_struct);

        let price = price_i64.get_magnitude_if_positive();
        let decimal = decimal_i64.get_magnitude_if_negative();
```

However, there is an issue with how the function handles the decimal values. Specifically, some assets have negative decimals, while others have positive decimals. When the decimal is positive, the function will revert in the `get_magnitude_if_negative subroutine()` .

Suggestion:

It is recommended to get the price based on the sign of `expo` . For example:

```
let i64_expo = price::get_expo(&pyth_price);
        let expo = (if(i64::get_is_negative(&i64_expo))
            i64::get_magnitude_if_negative(&i64_expo) else
i64::get_magnitude_if_positive(&i64_expo));
```

Resolution:

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

OAG-3 Redundant Code

Severity: Informational

Status: Acknowledged

Code Location:

packages/lotus_finance/sources/oracle/oracle_ag.move#37

Descriptions:

The `assert_price_object_type` function code is commented out, and the same check already exists in the `get_price` function, which is redundant.

Suggestion:

It is recommended to remove the redundant code.

TEL-1 Unbound `LotusLPFarmCreatePoolTicket` Types Cause LP Token Minting Vulnerability

Severity: Minor

Status: Acknowledged

Code Location:

packages/lotus_finance/sources/lp_token/teller.move

Descriptions:

In the `teller` module, the `mint_lp_token_with_ticket` function enables users to mint equivalent `LP` tokens based on the deposit certificates created after vault deposits. However, the function has the following issues: Users can forge a `Farm` of a specific type, make a deposit to obtain a fake deposit certificate, and then call the `mint_lp_token_with_ticket` function in the legitimate protocol to mint valid `LP` tokens, resulting in token inflation and asset loss.

Suggestion:

It is recommended to add binding logic in the `Teller<LP>` module to link deposit certificates to specific `LP` token types, ensuring only certificates from the corresponding `Farm` type can mint that type of `LP` token.

Resolution:

The client replied that the number of all LP tokens minted from Farm is based on the value of the user's deposit, and the LP is only used for data storage and has no impact on the protocol.

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.