Merkle Trade

Smart Contract

Audit Report

MOVEBIT

contact@movebit.xyz

https://twitter.com/movebit_

07/21/2023

Merkle Trade Smart Contract Audit Report

1 Executive Summary

1.1 Project Information

Description	Merkle Trade is a decentralized trading platform
Type	Derivatives
Auditors	MoveBit
Timeline	June 26, 2023 – July 19, 2023
Languages	Move
Platform	Aptos
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/tauruslabs/merkle-contract
Commits	e0e4cb0f0c606f319f52f729b26fc7a38e6e0d48 57cdac1a7e923b6b91b43301fc8f83690cb5864f

1.2 Files in Scope

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

ID	Files	SHA-1 Hash
MCT	merkle-contract/Move.toml	ea762e2a02fa2145435226ed7d7997538a1849cf
TDC	merkle-contract/sources/trading_calc.move	3f862b1092482a8c398a182142fdf9ba3ef8e609

MGT	merkle-contract/sources/managed_trading.move	5339ae12a8e09ae20212a88380f6ff6f48de567c
FDB	merkle-contract/sources/fee_distributor.move	9b6f5785eff39c3df013dc964c31b87a2dcb1178
MFD	merkle-contract/sources/managed_fee_distributor.move	9c5b91b2d0290e2fca27a06859748b087008093b
MHL	merkle-contract/sources/managed_house_lp.move	9fc05ae034136f996c7dd9e39a528313a5f1e2e1
PRO	merkle-contract/sources/price_oracle.move	9107d415439ec49d1399933c9029981a70df2574
HSL	merkle-contract/sources/house_lp.move	eda18122ef47ca3f2049f844b9f28a8748d6cd0b
SBS	merkle-contract/sources/common/switchboard_scripts.move	0439668c4c1c49b403c10f3ec4da531a2055a9e0
PYS	merkle-contract/sources/common/pyth_scripts.move	8a9a914e8546786ff5990fb4e343ccb859c5eeb4
SMU	merkle-contract/sources/common/safe_math_u64.move	1f3ffc9a667fc922d8fa24a51b94f4edb9814975
TRD	merkle-contract/sources/trading.move	a4931253e8c40bc3ae477344b6d8d95554410698
VAT	merkle-contract/sources/vault.move	f4e425c19383895154aad25f409ff7e8c2951dc9

MNV	merkle-contract/sources/managed_vault.move	249039306a879c01a74a9af47b25815f8890c777
VLT	merkle-contract/sources/vault_type.move	93bb25fd8882abfd0d805e592775ecf3d8a4c9f6
MPO	merkle-contract/sources/managed_price_oracle.move	584de8e225b9cd967c25cd9b52f50eb184b42424

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	8	7	1
Informational	2	1	1
Minor	4	4	
Medium	2	2	
Major			
Critical			

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 **Merkle Trade** to identify any potential issues and vulnerabilities in the source code of the **Merkle Trade** 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 **8** issues of varying severity, listed below.

ID	Title	Severity	Status
TRD-1	Limit Orders Cannot Be Executed	Medium	Fixed
TRD-2	`tp_percent` Should Be Less Than or Equal to `pair_info.maximum_profit`	Minor	Fixed
SBS-1	Missing Check for Negative in `get_switchboard_price()` Function	Medium	Fixed
HSL-1	Zero Fee Deposit for Small Amounts	Minor	Fixed
HSL-2	Error Code `E_COIN_NOW_INITIALIZED` is A Bad Naming	Informational	Fixed
HSL-3	Check Sufficient lp Collectral	Minor	Fixed
MGT-1	The Admin Lacks the Permission for the Burn ExecuteCapability Capability	Minor	Fixed
MGT-2	Centralization Risk	Informational	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the **Merkle Trade** Smart Contract :

Admin:

- The admin has the authority to withdraw accumulated fees from the stake vault and the dev vault, as well as the ability to set the weights for stake, LP, and dev in the fee distribution through `managed_fee_distributor.move` .
- The admin has the authority to register users, deposit and withdraw funds for a specific collateral type in the house_lp module, as well as set deposit fees, withdrawal fees, withdrawal divisions, and minimum deposit amounts for house_lp through `managed_house_lp.move` .
- The admin has the authority to register oracles, manage allowed update addresses, perform updates, and configure various parameters in the price_oracle module for specific pair types through `managed_price_oracle.move` .
- The admin has the authority to initialize the module, set address executor candidates, claim and burn execute capabilities, place orders, cancel orders, execute orders and exit positions, pause and restart trading, and configure various parameters in the managed_trading module for specific pair and collateral types through `managed_trading.move` .
- The admin has the authority to register a vault for a specific VaultT and AssetT in the `managed_vault.move` .

User:

- Users can pledge assets to obtain MKLP tokens.
- Users can destroy MKLP tokens, retrieve pledged assets, and claim rewards.
- Users can create and cancel market/limit orders.
- Users can execute market decrease orders.

Executor:

- Executor can execute orders.
- Executor can burn ExecuteCapability.
- Executor can execute the take-profit, stop-loss, or liquidate function.

4 Findings

TRD-1 Limit Orders Cannot Be Executed

Severity: Medium

Status: Fixed

Code Location: merkle-contract/sources/trading.move#L861

Descriptions:

The function `execute_order()` is used to execute an order. Inside the function, It checks if more than 30 seconds have passed since the order was created. If the condition is met, it cancels the order by calling `cancel_order_internal()`. However, if the order is a limit order, it means that the order has a specific price set by the trader at which they are willing to buy or sell the asset, if this timeout has elapsed, the order is considered expired.

```
let now = timestamp::now_seconds();
      if (now - order.created_timestamp > 30) {
          cancel_order_internal<PairType, CollateralType>(
              _order_id,
              order,
              T_CANCEL_ORDER_EXPIRED
          );
          return
      };
```

Suggestion: Suggest checking whether the order is a market order or a limit order.

Resolution: Added a code check to ensure that only market orders can be canceled.

SBS-1 Missing Check for `Negative` in `get_switchboard_price()` Function

Severity: Medium

Status: Fixed

Code Location: merkle-contract/sources/common/switchboard_scripts.move#L6-L17

Descriptions:

The function `get_switchboard_price()` is used to retrieve the price and round confirmed timestamp from Switchboard.

However, it does not follow the best practices as recommended by the official guidelines.

https://docs.switchboard.xyz/guides/feeds/best-practices#integration-checklist

According to the recommendations, it is advised to include a check to ensure the `negative` is not true. If the negative value is true, it implies that there might be some problem with the price received from Oracle, potentially because the price is expired or has some error.

However, this check is missing in the provided code, which can lead to potential issues.

```
    public fun get_switchboard_price(
      aggregator: &Aggregator,
    ): (u128, u8, u64) {
      let (price, updated_time) = aggregator::latest_value(aggregator);
      let (price_value, price_decimals, negative) = switchboard_math::unpack
  (price);
      assert!(negative == false, ERR_NEGATIVE_SWITCHBOARD_PRICE);

      (price_value, price_decimals, updated_time)
    }
  }
```

Suggestion: It is recommended to add a check `assert!(negative == false, ERR_NEGATIVE_SWITCHBOARD_PRICE);` to ensure that the obtained price from the Switchboard is not negative.

```
  let (value, _, negative) = math::unpack(result);
  assert!(negative == false, ERR_NEGATIVE_SWITCHBOARD_PRICE);
```

Resolution: Checked switchboard neg flag.

HSL-1 Zero Fee Deposit for Small Amounts

Severity: Minor

Status: Fixed

Code Location: merkle-contract/sources/house_lp.move#L179

Descriptions:

In the `deposit()` function, there is a possibility for users to deposit a very small amount that results in a fee of zero. This allows users to bypass paying any deposit fees.

The function calculates the deposit fee based on the house_lp.deposit_fee percentage and the original amount deposited.

If the amount is extremely small, the calculated fee may round down to zero.

Consequently, the _amount variable will remain unchanged, and the user can deposit the entire amount without incurring any fee.

This issue allows users to make deposits without paying the intended deposit fee, potentially leading to a loss of revenue for the system. The same issue for `withdraw()` function.

```
 let house_lp_coin_balance = vault::vault_balance<vault_type::HouseLPVault, AssetT>();
        let supply = (option::extract<u128>(&mut coin::supply<MKLP>()) as u64);
        let fee = safe_mul_div(_amount, house_lp.deposit_fee, FEE_POINTS_DIVISOR);
        _amount = _amount - fee;
        let mintAmount: u64;
        if (supply == 0) {
            mintAmount = _amount;
        } else {
            mintAmount = safe_mul_div(supply, _amount, (house_lp_coin_balance - (_amount + fee)));
        };
```

Suggestion: It is recommended to implement a minimum fee threshold or establish a lower limit for the fee calculation. By setting a minimum fee, even for small deposits, users will be required to pay a nominal fee, ensuring fairness and maintaining the intended revenue model for the system.

Resolution: Added `minimum_deposit` to limit the minimum deposit amount.

TRD-2 `tp_percent` Should Be Less Than or Equal to `pair_info.maximum_profit`

Severity: Minor

Status: Fixed

Code Location: merkle-contract/sources/trading.move#L1637

Descriptions:

The function `execute_increase_order_internal()` is responsible for executing an increase order in a trading pair.

Inside the function, the stop-loss and take-profit prices of the position are updated based on the order's type and the specified conditions.

If the order's take-profit trigger price and the maximum take-profit price are the same, the code will assign that value to the take-profit trigger price of the position. In other words, there is no preference for either value in this scenario, and they are considered equal.

```
 if (_order.is_long) {
                position_ref_mut.stop_loss_trigger_price =
                    if (_order.stop_loss_trigger_price < position_ref_mut.avg_price )
                        {_order.stop_loss_trigger_price} else { 0 };

                let maximum_take_profit_price = safe_mul_div(
                    position_ref_mut.avg_price,
                    (position_ref_mut.size + maximum_profit),
                    position_ref_mut.size
                );
                position_ref_mut.take_profit_trigger_price = min(_order.take_profit_trigger_price, maximum_take_profit_price)
            } else {
                position_ref_mut.stop_loss_trigger_price =
                    if (_order.stop_loss_trigger_price > position_ref_mut.avg_price )
                        {_order.stop_loss_trigger_price} else { U64_MAX };
                // If maximum profit is less than or equal to 0 in a short position, it will be set to 1.
                // This is because a price of 0 cannot occur.
                let maximum_take_profit_price = safe_mul_div(
                    position_ref_mut.avg_price,
                    if (position_ref_mut.size > maximum_profit) { position_ref_mut.size - maximum_profit } else 1,
                    position_ref_mut.size
                );
                position_ref_mut.take_profit_trigger_price = max(_order.take_profit_trigger_price, maximum_take_profit_price)
            };
```

However, inside the `update_position_tp_sl()`, an assertion is made to validate that the calculated take-profit percentage is less than the maximum allowed profit percentage specified in the pair's information. The business logic of this function, `update_position_tp_sl()`, is different from `execute_increase_order_internal()`.

```
1       // validate max profit
2          let price_change = safe_mul_div(
3              abs(_take_profit_trigger_price, position_ref_mut.avg_price),
4              BASIS_POINT,
5              position_ref_mut.avg_price
6          );
7          let tp_percent = safe_mul_div(position_ref_mut.size, price_change,
    position_ref_mut.collateral);
8          assert!(tp_percent < pair_info.maximum_profit, E_UPDATE_TAKE_PROFIT
    _INVALID);
```

Suggestion: Ensure that `tp_percent` is less than or equal to `pair_info.maximum_profit` to avoid an invalid update to the take-profit trigger price.

`assert!(tp_percent < = pair_info.maximum_profit, E_UPDATE_TAKE_PROFIT_INVALID);`

Resolution: Updated the code to ensure that `tp_percent` is less than or equal to pair_info.maximum_profit.

MGT-1 The Admin Lacks the Permission for the Burn `ExecuteCapability` Capability.

Severity: Minor

Status: Fixed

Code Location: merkle-contract/sources/managed_trading.move#L62-106

Descriptions:

The admin can only register the `ExecuteCapability` to other addresses, but the admin lacks permission for the burn `ExecuteCapability` capability. The following code only allows the executor candidate to burn the `ExecuteCapability`.

```
    /// Burn ExecuteCapability
    /// Only allowed for executor candidate.
    public entry fun burn_execute_cap<PairType, CollateralType>(
        _host: &signer,
    ) acquires ExecuteCapabilityStore {
        // If @merkle calls this function, the modules may no longer be available
        assert!(address_of(_host) != @merkle, E_NOT_AUTHORIZED);
        move_from<ExecuteCapabilityStore<PairType, CollateralType>>(address_of(_host));
    }
```

Suggestion: It is recommended to add admin permission to burn the `ExecuteCapability`.

Resolution: The admin has been granted this permission.

HSL-2 Error Code `E_COIN_NOW_INITIALIZED` is A Bad Naming

Severity: Informational

Status: Fixed

Code Location: merkle-contract/sources/house_lp.move#L33, L120

Descriptions:

The naming of `E_COIN_NOW_INITIALIZED` is not correct according to the comment: `When the asset register with house_lp is not a coin`, it should be `E_COIN_NOT_INITIALIZED` instead. Change it to the correct name to avoid confusion.

```
/// When the asset register with house_lp is not a coin
const E_COIN_NOW_INITIALIZED: u64 = 4;
```

```
assert!(coin::is_coin_initialized<AssetT>(), E_COIN_NOW_INITIALIZED);
```

Suggestion:

```
/// When the asset register with house_lp is not a coin
const E_COIN_NOT_INITIALIZED: u64 = 4;

assert!(coin::is_coin_initialized<AssetT>(), E_COIN_NOT_INITIALIZED);
```

Resolution: The name has been changed.

HSL-3 Check Sufficient lp Collectral

Severity: Minor

Status: Fixed

Code Location: merkle-contract/sources/house_lp.move#L248

Descriptions:

It's a good practice to check that `HouseLPVault` has enough collateral to `withdraw`, otherwise, it will go deep down to the aptos_std::coin::extract to check the balance.

```
let coin_balance = vault::vault_balance<vault_type::HouseLPVault, AssetT>();
let supply = (option::extract<u128>(&mut coin::supply<MKLP>()) as u64);
let return_amount = safe_mul_div(coin_balance, _amount, supply);
let fee = safe_mul_div(return_amount, house_lp.withdraw_fee, FEE_POINTS_DIVISOR);
return_amount = return_amount - fee;
let withdraw_coin = vault::withdraw_vault<vault_type::HouseLPVault, AssetT>(return_amount);
coin::deposit(user_addr, withdraw_coin);
```

Suggestion: Add an assertion to make sure that LP collateral is more than withdraw amount.

```
let coin_balance = vault::vault_balance<vault_type::HouseLPVault, AssetT>();
let supply = (option::extract<u128>(&mut coin::supply<MKLP>()) as u64);
let return_amount = safe_mul_div(coin_balance, _amount, supply);
let fee = safe_mul_div(return_amount, house_lp.withdraw_fee, FEE_POINTS_DIVISOR);
return_amount = return_amount - fee;
assert!(coin_balance >= return_amount, E_HOUSE_LP_AMOUNT_NOT_ENOUGH);
let withdraw_coin = vault::withdraw_vault<vault_type::HouseLPVault, AssetT>(return_amount);
```

Resolution: This check has been added.

MGT-2 Centralization Risk

Severity: Informational

Status: Acknowledged

Code Location: merkle-contract/sources/managed_fee_distributor.move;

merkle-contract/sources/managed_house_lp.move;

merkle-contract/sources/managed_price_oracle.move;

merkle-contract/sources/managed_trading.move;

merkle-contract/sources/managed_vault.move

Descriptions:

These are some centralized risks in the contract.

- The admin has the authority to withdraw accumulated fees from the stake vault and the dev vault, as well as the ability to set the weights for stake, LP, and dev in the fee distribution through `managed_fee_distributor.move` .
- The admin has the authority to register users, deposit and withdraw funds for a specific collateral type in the house_lp module, as well as set deposit fees, withdrawal fees, withdrawal divisions, and minimum deposit amounts for house_lp through `managed_house_lp.move` .
- The admin has the authority to register oracles, manage allowed update addresses, perform updates, and configure various parameters in the price_oracle module for specific pair types

through `managed_price_oracle.move.`

- The admin has the authority to initialize the module, set address executor candidates, claim and burn execute capabilities, place orders, cancel orders, execute orders and exit positions, pause and restart trading, and configure various parameters in the managed_trading module for specific pair and collateral types through `managed_trading.move.`
- The admin has the authority to register a vault for a specific VaultT and AssetT in the `managed_vault.move.`
- The executor has the authority to execute orders.
- The executor has the authority to burn `ExecuteCapability` .
- The executor has the authority to execute the take-profit, stop-loss, or liquidate function.

Suggestion: To mitigate the centralization risk in the smart contract, we recommend the following. **Resolution:**

- Foster community governance and participation to ensure decision-making power is distributed among the system's participants.
- Implement robust security measures to protect against potential attacks or malicious actions, such as multi-signature.

Removing all centralized methods can be considered to permanently address centralized risks.

Appendix 1

Issue Level

- **Informational** Informational items 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.
- **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.