Kana Smart Contract

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Thu Feb 20 2025

Kana Smart Contract Audit Report

1 Executive Summary

1.1 Project Information

Description	An orderbook-based perpetual futures platform on Aptos
Type	DeFi
Auditors	MoveBit
Timeline	Wed Jan 15 2025 - Mon Feb 10 2025
Languages	Move
Platform	Aptos
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/kanalabs/perpetual-core https://github.com/0xAnto/move-integers
Commits	https://github.com/kanalabs/perpetual-core: db32c0b1655fcdda43e4afca2fa4a0d3837d9af8 2af3abf8b5d7a836757ceaaa29fb570fa9de27e2 https://github.com/0xAnto/move-integers: 85db3cc1fe520359c95fb22c85d899705c7907f9 ca7a1ebb997c71fa2789e2b1cca3e2e5489be2be

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MOV	Move.toml	68d61ac4e8b7eeaf6867212d6a618ecc4064760b
I8	sources/i8.move	b8d56c94a5e966c73c06fd7fd6f693eea36d7c0f
I32	sources/i32.move	41f72b56f504424196678205be7c8af6c5015e33
I12	sources/i128.move	a5383808321ce85437c4f4595a9ce5c5dd5ad9e3
I64	sources/i64.move	47e26286588fea76182b7e46071fde0c27b8f6a2
I25	sources/i256.move	b382ad1b37565cc1f81dedd2b6fa25dd56b8c387
I16	sources/i16.move	1fbc5c08907320e8ed9513d5aac8837888975326
MOV	Move.toml	ef4e51b7b67b861d47862afd69c2b677693ff46f
RES	sources/resource.move	e28bf04bf0d085e3956dae9ff92f27d05bb5bfdf
DSC	sources/delegated_scripts.move	be3c6bb6c2e2ec53b82943500084384367b9552c
POR	sources/price_oracle.move	fe71c204fd778be618b6ef107bcf7ec5920f1493

DPR	sources/delegate_proxy.move	aa050fdf9703bbe1fd3d21bf7bf2bc67222e275b
PCO	sources/perpetual_core.move	55ba95c2eedf9a335bd89907608f67f8d7d25ee1
UTI	sources/utils.move	41f1eef89b01daddc3c2ac54399182606ce6f080
TRE	sources/treasury.move	29df06cc3e3be8809c49acacd781a1cca09aee70
PSC	sources/perpetual_scripts.move	a42b747c196c620264d55b71fe036805b7022fa3
USP	sources/utils.spec.move	b98e18a1490fb5ed177635cc6c02fcd2544cc007

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	5	5	0
Informational	1	1	0
Minor	1	1	0
Medium	0	0	0
Major	3	3	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 Kanalabs to identify any potential issues and vulnerabilities in the source code of the Kana 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 5 issues of varying severity, listed below.

ID	Title	Severity	Status
I32-1	Division Function Lacks Division-by-Zero Check	Minor	Fixed
PCO-1	Full-Matching Requirement in Liquidation Logic May Lead to Delayed Liquidation	Major	Fixed
PCO-2	`entry_price` Calculation Error	Major	Fixed
PCO-3	Unrestricted Market Registration May Cause Protocol Freeze	Major	Fixed
PCO-4	Code Optimisation	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Kana Smart Contract :
Market Creater

- `register_market<UtilityType>` : Register a new perpetual market.
- `contribute_to_insurance` : Contribute to the insurance fund.
- `update_maintenance_margin` : Update maintenance margin.
- `update_max_leverage` : Update max leverage.
- `update_max_lots` : Update max lots for a market.
- `withdraw_fees<UtilityType>` : Withdraw fees from a market.

User

- `deposit_asset` : Deposit collateral asset into trading account.
- `withdraw_asset` : Withdraw collateral asset from trading account.
- `withdraw_asset_all_markets` : Withdraw collateral asset from trading account after checking positions in multiple markets.
- `enable_all_markets` : Enable trading for all available markets for the caller.
- `execute_tpsl_market` : Execute take profit or stop loss market order for a position.
- `liquidate_position` : Liquidate a position that has fallen below maintenance margin requirements.
- `place_market_order_external` : Place a market order.
- `place_limit_order_external` : Place a limit order.
- `place_multiple_orders_external` : Place multiple orders.
- `cancel_and_place_multiple_orders_external` : Cancel and place multiple orders.
- `cancel_multiple_orders_external` : Cancel multiple orders.
- `update_position_external` : Update position.
- `update_take_profit_external` : Update take profit.
- `update_stop_loss_external` : Update stop loss.

- `collapse_position_external` : Collapse position.
- `add_delegation` : Manage delegation.
- `remove_delegation` : Remove delegation.
- `change_delegated_proxy` : Change delegated proxy.
- `toggle_delegation` : Toggle delegation.
- `extend_delegation` : Extend delegation.
- `place_market_order_delegated` : Place a market order.
- `place_limit_order_delegated` : Place a limit order.
- `cancel_multiple_orders_delegated` : Cancel multiple orders.
- `update_position_delegated` : Update position.
- `update_take_profit_delegated` : Update take profit.
- `update_stop_loss_delegated` : Update stop loss.
- `collapse_position_delegated` : Collapse position.

4 Findings

I32-1 Division Function Lacks Division-by-Zero Check

Severity: Minor

Status: Fixed

Code Location:

sources/i32.move#90;

sources/i128.move#102;

sources/i256.move#99;

sources/i16.move#90

Descriptions:

In the `i16`, `i32`, `i128`, and `i256` modules, the `div` function lacks a division-by-zero check. When the divisor is zero, the function does not detect and handle the condition, which may lead to runtime errors, incorrect computation results, or abnormal program termination, thus posing potential security risks.

Suggestion:

It is recommended to integrate a check within the `div` function to verify whether the divisor is zero. If zero is detected, the function should immediately throw an exception or return an error, preventing erroneous computations.

Resolution:

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

PCO-1 Full-Matching Requirement in Liquidation Logic May Lead to Delayed Liquidation

Severity: Major

Status: Fixed

Code Location:

sources/perpetual_core.move

Descriptions:

In the protocol's liquidation logic, market orders are required to match the entire liquidation quantity of a position exactly. However, in scenarios of insufficient liquidity, this requirement can lead to unsuccessful liquidations. If liquidation is delayed due to inadequate liquidity, the position may eventually be liquidated at a price significantly lower than expected. Consequently, the liquidation cost would be much lower, and the insurance fund would have to compensate for the difference, exposing it to substantial loss risks.

```
    assert!(
        filled_lots == (liquidation_lots as u128),
        E_INSUFFICIENT_LIQUIDITY_IN_OB_TO_LIQUIDATE
    );
```

Suggestion:

It is recommended to allow liquidation orders to be partially filled when liquidity is low instead of requiring full matching of the liquidation quantity in one go, or that other effective measures be taken to prevent liquidation failures or delays due to illiquidity.

Resolution:

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

PCO-2 `entry_price` Calculation Error

Severity: Major

Status: Fixed

Code Location:

sources/perpetual_core.move#4133-4140,4039-3043

Descriptions:

```
short.filled_open_size += new_filled_size;
            short.entry_price = ((short.entry_price
               * (short.filled_open_size as u128))
               + (((price as u128) * SCALING_FACTOR) * (new_filled_size as u128))) / (
               (short.filled_open_size as u128) + (new_filled_size as u128)
            );
```

In `handle_short_cancel_order` and `handle_long_cancel_order` functions,to calculate the price here, you should first calculate the value of `entry_price`, and then add `new_filled_size` to `filled_open_size`.

Suggestion:

It is recommanded that price is calculated first, then `new_filled_size` is added.

Resolution:

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

PCO-3 Unrestricted Market Registration May Cause Protocol Freeze

Severity: Major

Status: Fixed

Code Location:

sources/perpetual_core.move#1200,1480

Descriptions:

When calling the `register_market` function, the protocol automatically adds the newly created market to the `available_markets` list. Later, when a user deposits collateral assets into their trading account, the protocol iterates through all markets in `available_markets` and creates an account for the user in each market.

However, since `register_market` is a public function without access control, any user can call it multiple times, leading to an uncontrolled increase in the number of markets. As the number of markets grows, the gas cost for deposit operations will significantly rise, potentially causing the protocol to freeze (out of gas or computation limit exceeded) in extreme cases.

```
 // Add market_id to PlatformConfig
        let platform = borrow_global_mut<PlatformConfig>(resource_address);
        platform.available_markets.push_back(market_id);

// register market accounts
        let markets = available_markets(resource_address);
        markets.for_each_ref(|market_id| {
            register_market_account(caller, *market_id)
        });
```

Suggestion:

It is recommended to confirm it aligns with your design.

Resolution:

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

PCO-4 Code Optimisation

Severity: Informational

Status: Fixed

Code Location:

sources/perpetual_core.move#2206

Descriptions:

In the `execute_short_market_open` function, there is an assertion `assert_fill(filled_lots, (order_lots as u128));` . Currently, this assertion is executed later in the function, which may result in significant gas consumption before the condition fails.

Suggestion:

It is recommended to move this assertion earlier in the function.

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.