Security Assessment

Draft Report

Liquidity Integration and Matching Orders (LIMO)

April - May 2025

Prepared for Kamino Finance

certora

Table of content

Draft

certora

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
LIMO	github.com/Kamino-Finance/limo-private	4c3d2e5	Solana

Project Overview

This document describes the security assessment of the liquidity integration and matching orders (LIMO) codebase.
The work was undertaken from April 23, 2025 to May 13 2025.

All contracts in the following files are included in our scope:

limo-private/programs/limo/src/token_operations.rs
limo-private/programs/limo/src/lib.rs
limo-private/programs/limo/src/utils/consts.rs
limo-private/programs/limo/src/utils/fraction.rs
limo-private/programs/limo/src/utils/flash_ixs.rs
limo-private/programs/limo/src/utils/mod.rs
limo-private/programs/limo/src/utils/constraints.rs
limo-private/programs/limo/src/utils/macros.rs
limo-private/programs/limo/src/utils/log_user_swap_balance_introspection.rs
limo-private/programs/limo/src/state.rs
limo-private/programs/limo/src/seeds.rs
limo-private/programs/limo/src/operations.rs
limo-private/programs/limo/src/handlers/close_order_and_claim_tip.rs
limo-private/programs/limo/src/handlers/create_order.rs
limo-private/programs/limo/src/handlers/update_global_config_admin.rs
limo-private/programs/limo/src/handlers/initialize_vault.rs
limo-private/programs/limo/src/handlers/update_global_config.rs
limo-private/programs/limo/src/handlers/mod.rs
limo-private/programs/limo/src/handlers/flash_take_order.rs
limo-private/programs/limo/src/handlers/log_user_swap_balances.rs
limo-private/programs/limo/src/handlers/take_order.rs

draft

limo-private/programs/limo/src/handlers/initialize_global_config.rs
limo-private/programs/limo/src/handlers/withdraw_host_tip.rs

where focus is on the added functionality of creation and processing of limit orders and the changes made to the update configuration instructions.

The team performed a manual audit and formal verification of all the Rust contracts. During the manual audit and the formal verification project, the Certora team discovered issues in the code, as listed in the following pages. Moreover, the formal verification ensures that already validated invariants are not broken by recent code updates, thereby ensuring safe code changes.

Protocol Overview

Kamino's LIMO (Liquidity Integration and Matching Order) Solana program implements a limit order protocol. This protocol allows makers to create orders for exchanging tokens and allows takers to (partially) fill such orders....

The LIMO system makes use of Vault accounts. Each Vault holds one specific token, and Vaults are shared amongst all orders.

Besides regular take orders the LIMO program also allows flash take orders, whereby the taker first receives the order input, allowing them to perform other operations, until later on in the same transaction the order output amount is paid.

The LIMO program allows for both permissioned and permissionless order taking (configured in the global config). In case of permissioned order taking, the Express Relay program is used with the Pythx Express router to validate that a take order is permitted.

Draft

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	0	0	0
Informational	8	0	0
Total	0	0	0

Severity Matrix

Impact					
	Critical	Medium	Medium/High	High	Critical
	High	Low/Medium	Medium	Medium/High	High
	Medium	Low	Low/Medium	Medium	Medium/High
	Low	Informational	Low	Low/Medium	Medium
		Rare	Unlikely	Likely	Very Likely
			Likelihood		

Draft

Detailed Findings

ID	Title	Severity	Status
I-01	Inconsistent naming of access control functions	Informational	
I-02	Usage of b"authority" instead of seeds::GLOBAL_AUTH in InitializeGlobalConfig	Informational	
I-03	Log message for global config initialization does not include global config	Informational	
I-04	Log message for vault initialization does not include vault	Informational	
I-05	Log message for update global config does not include global config	Informational	
I-06	UpdateHostFeeBps check uses 10000 instead of FULL_BPS	Informational	
I-07	Typo in comment of Order::flash_start_taker_output_balance	Informational	
I-08	Function create_order should set order.tip_amount to zero.	Informational	

Informational Severity Issues

I-01. Inconsistent naming of access control functions.

Description: constraints.rs has several functions that are used for instruction access control: `emergency_mode_disabled`, `flash_taking_orders_disabled`, `create_new_orders_disabled`, and `taking_orders_disabled`.

The `emergency_mode_disabled` function returns `Ok()` when emergency mode is disabled and returns an error when it is enabled.
The other three functions behave in the opposite direction, returning `Ok()` if it is enabled and returning an error if it is disabled/blocked.

Recommendations: Rename `flash_taking_orders_disabled`, `create_new_orders_disabled`, and `taking_orders_disabled` to `..._enabled` to make naming consistent with the result.

Customer's response:

I-02. Usage of b"authority" instead of seeds::GLOBAL_AUTH in InitializeGlobalConfig

Description: InitializeGlobalConfig uses inline `b"authority".as_ref()` in the seeds for `pda_authority`, while there also exists a constant for it.

Recommendations: Change
`seeds = [b"authority".as_ref(), global_config.key().as_ref()],`
to
`seeds = [seeds::GLOBAL_AUTH, global_config.key().as_ref()],`

Customer's response:

Draft

I-03. Log message for global config initialization does not include global config.

Description: When initializing a global config, a message is logged with `"Initializing global config with global authority {} and bump {}"`. This message does not include the global_config account. This makes it harder to know which account was created, based on the logs.

Recommendations: Add the global_config account to the log message

Customer's response:

I-04. Log message for vault initialization does not include vault.

Description: When initializing a global config, a message is logged with `"Initializing vault for global config {} with mint {}"`. This message does not include the vault account. This makes it harder to know which account was created, based on the logs.

Recommendations: Add the `global_config` account to the log message

Customer's response:

I-05. Log message for update global config does not include global config.

Description: When updating global config, it logs the config mode and new value, but it does not include the global config. This makes it harder to know which config's settings were updated, based on the logs.

Recommendations: Add the `global_config` account to the log message

Customer's response:

Draft

I-06. UpdateHostFeeBps check uses 10000 instead of FULL_BPS.

Description: In update_global_config for `UpdateGlobalConfigMode::UpdateHostFeeBps` it uses an inline number 10000 while a constant `FULL_BPS` already exists

Recommendations: To improve readability, use `FULL_BPS` instead of 10000

Customer's response:

I-07. Typo in comment of Order::flash_start_taker_output_balance

Description: The comment for `Order::flash_start_taker_output_balance` in state.rs contains a typo in the first sentence:
`/// This is only used for flash operations, and is set to the blanance on the start`

Recommendations: Change to "balance"

Customer's response:

I-08. Function create_order should set order.tip_amount to zero.

Description: Function `operation::create_order` does not set `tip_amount` to zero. This is okay as the order is assumed to be zeroed out by Anchor. However, for safety, it is better to set `tip_amount` to zero explicitly.

Recommendations: Set `order.tip_amount` to zero.

Customer's response:

Draft

Formal Verification

Methodology

We performed verification of the Kamino Limo Protocol using the Certora verification tool which is based on Satisfiability Modulo Theories (SMT) and symbolic execution. In short, the Certora verification tool works by compiling formal specifications written in the Certora Verification Language (CVLR) and Kamino's implementation source code written in Rust. More information about Certora's tooling can be found in the Certora Technology Whitepaper.

If a property is verified with this methodology it means the specification in CVLR holds for all possible inputs. However specifications must introduce assumptions to rule out situations which are impossible in realistic scenarios (e.g. to specify the valid range for an input parameter). Additionally, SMT-based verification is notoriously computationally difficult. As a result, we occasionally introduce overapproximations (replacing real computations with broader ranges of values) or underapproximations (replacing real computations with fewer values) to make verification feasible.

Rules: A rule is a verification task possibly containing assumptions, calls to the relevant functionality that is symbolically executed and assertions that are verified on any resulting states from the computation.

Inductive Invariants: Inductive invariants are proved by induction on the structure of a smart contract. We use constructors/initialization functionality as a base case, and consider all other (relevant) externally callable functions as step cases.
Specifically, to prove the base case, we show that a property holds in any resulting state after a symbolic call to the respective initialization function. For proving step cases, we generally assume a state where the invariant holds (induction hypothesis), symbolically execute the functionality under investigation, and prove that after this computation any resulting state satisfies the invariant. Each such case results in one rule.

Note that to make verification more tractable, we sometimes prove on lower level functions that contain the relevant logic. In the case of Kamino, we prove invariants correct by proving properties on the relevant functionality provided in `operations.rs`.

Draft

General Assumptions and Simplifications

Configuration and Munging

1) Prover Configuration: The Solana contracts were compiled to SBFv1 using the Rust compiler version 1.79. The Solana version was solana-cli 1.18.16.
2) Loops are inherently difficult for formal verification. We handle loops by unrolling them a specific number of times. We thus use an underapproximation on the number of validators in the deposit function. Consequently, we use a **loop_iter of 1,** unrolling each loop once.

Verification Notations

Formally Verified	The rule is verified for every state of the contract(s), under the assumptions of the scope/requirements in the rule.
Formally Verified After Fix	The rule was violated due to an issue in the code and was successfully verified after fixing the issue.
Violated	A counterexample exists that violates one of the assertions of the property.

Formal Verification Properties

Order type invariants

The `Order` datatype must follow certain invariants in order to ensure the correctness of the business logic. These invariants ensure for instance that the price of the order never changes by maintaining `initial_input_amount` and `expected_output_amount` as constant.
This module covers the following invariants over the `Order` datatype:

- `initial_input_amount` and `expected_output_amount` never changes.
- `remaining_input_amount` cannot increase
- `filled_output_amount`, `tip_amount` and `number_of_fills` cannot decrease
- `remaining_input_amount` must be less than or equal to `initial_input_amount`

P-01. LIMO operations preserve the Order datatype invariants	
Status: Verified	Specification: If the Order datatype invariants hold before performing an arbitrary user operation, then they must hold after the user operation.

Rule Name	Status	Description	Link to rule report
ordertype_create_order	Verified	*Base Case: create_order()* *The rule checks that above described Order datatype invariants hold after create_order. An additional assumption that order.tip_amount is zero before the call to create_order is required due to I-08.*	*Report*

ordertype_take_order	Verified	*Step Case: take_order()*	
ordertype_close_order_and_claim_tip	*Verified*	*Step Case: close_order_and_claim_tip()*	
ordertype_flash_withdraw_order_input	Verified	*Step Case: flash_withdraw_order_input()*	
ordertype_flash_pay_order_output	Verified	*Step Case: flash_pay_order_output()*	

Matching Order rounds in favor of the maker

When an order is matched (partially or fully), it is critical that the taker never receives a better price than the maker. Otherwise, the taker can break their order into small fragments, damaging the maker's price. This would lead to a shortage of makers providing liquidity to the market.

P-02. Matching price is rounded towards the maker	
Status: Verified	Specification: take_order provides better price (input/output) to the maker compared to the taker.

Rule Name	Status	Description	Link to rule report
take_order_round_towards_maker	Verified	*On the input of maker_input and maker_output, take_order computes taker_input and taker_output such that maker_input * taker_output >= maker_output * taker_input.*	*Report*

Flash Take Order Properties

P-03. Flash take order must not allow intermediate operations and must provide the same price as take_order

Status: Verified	

Rule Name	Status	Description	Link to rule report
flash_take_order_no_intermediate_op	Verified	*After a call to flash_withdraw_order_input, if take_order or flash_withdraw_order_input is called again, then it will produce an error.*	*Report*
flash_take_order_equivalent_to_take_order	Verified	*Flash_pay_order_output computes the same price as take_order when evaluated against identical orders.*	

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.

Draft