certora

Security Assessment
Final Report



Kamino – Borrow Orders

02.2026

Prepared for Kamino finance

Table of contents

Project Summary **3**
Project Scope 3
Project Overview 3
Protocol Overview 4
Findings Summary 5
Severity Matrix 6
Detailed Findings **7**
Medium Severity Issues 8
M-01 Setting order does not take into account fees which may result in exploiting the creator 8
M-02 Race condition in borrow order update and filling 10
Low Severity Issues 11
L-01 Consider cleaning expired orders before update 11
L-02 Instant Liquidations from debt_term_seconds updates 12
L-03 No Protection against setting debt terms on reserves with active borrows 13
Informational Issues 14
I-01. Divergence between documented behavior and implementation (PR#538) 14
I-02. Consider introducing an incentive for the borrow order executor (PR#540) 14
Disclaimer **15**
About Certora **15**

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Klend	https://github.com/Kamino-Finance/klend-private	17b313	Solana

Project Overview

This document describes findings discovered during a manual code review. The work was undertaken from **08.2025** to **09.2025.**

The scope includes changes made in the following PRs under **programs/kamino-lending/src** directory:

- Klend-private - BorrowOrders:
  - PR#551, PR#574, PR#560, PR#541, PR#540, PR#552, PR#538, PR#537, PR#631, PR#608, PR#580, PR#703, #641, #572, #581, #619, #631
  - Klend-private - Reserve.debt_maturity_timestamp
    - PR#524, PR#526, PR#547
  - Klend-private - PR#537 - Reserve.debt_term_seconds
  - Klend-private - PR#514 - Allow many reserves of same mint on the same market

Protocol Overview

Kamino Lending (KLend) is a decentralized finance (DeFi) platform designed to optimize the borrowing and lending of crypto assets on the Solana blockchain. The protocol team actively develops new features and updates the already deployed program.

- Overcollateralized lending: users deposit assets into per-asset reserves to mint cTokens (collateral), then borrow against obligations that track deposits, borrows, LTV and risk.
- Risk is enforced via reserve configs, utilization-based borrow rates, liquidation thresholds, price oracles, and emergency-mode gates.
- The program exposes granular instructions for deposit/redeem, borrow/repay, liquidation, flash loans, referrers, elevation groups, and market/global configuration.

Borrow orders and permissionless execution

- A user posts a borrow request onto their obligation with constraints: debt token mint, amount, max APR, min debt term, and fillable-until timestamp; then any executor can fill all/part of it using a reserve that satisfies the order. Creation and execution are market-gated.
- `set_borrow_order(order_config, min_expected_current_remaining_debt_amount)`: creates/updates/cancels the order. Validations: non-zero amount, non-zero max APR, fillable-until in the future. Non-updatable: debt mint, destination account; to change them, cancel and place a new order.
- `fill_borrow_order` (permissionless): reuses the standard borrow logic but caps borrow size to the order's `remaining_debt_amount`, enforces exact destination account and mint, and then decrements `remaining_debt_amount`.
- Reserve selection constraints:
    - Mint must match the order's debt mint.
    - Reserve's max borrow rate bps must be <= order's `max_borrow_rate_bps`.
- Reserve's offered debt term (fixed term or maturity timestamp) must be >= order's `min_debt_term_seconds`.
- Must be before `fillable_until_timestamp`.
- On full fill, the order is cleared; on partial fill, it remains with updated remainder. Funds are delivered to the pre-declared destination account; fees go to the reserve's fee vault. Execution is blocked if globally disabled.

TL;DR

- Borrow orders: user-specified constraints; permissionless fills reuse core borrow checks; strict gating and mint/term/rate/time validations ensure orders are only filled under acceptable conditions.

Findings Summary

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

Severity	Discovered	Fixed
Critical	0	0
High	0	0
Medium	2	1
Low	3	2
Informational	2	1
Total	7	4

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

Detailed Findings

ID	Title	Severity	Status
M-01	Setting order does not take into account fees which may result in exploiting the creator	Medium	Acknowledged
M-02	Race condition in borrow order update and filling	Medium	Fixed
L-01	Consider cleaning expired orders before update	Low	Fixed
L-02	Instant Liquidations from debt_term_seconds Updates	Low	Acknowledged
L-03	No Protection against setting debt terms on reserves with active borrows	Low	Acknowledged

Medium Severity Issues

M-01 Setting order does not take into account fees which may result in exploiting the creator

Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: obligation.rs#L991-L992	Status: Acknowledged	Related to PR#538

Description: When a user creates a borrow order, he sets a `remaining_debt_amount` var, which is the raw amount that he wants to receive from a loan. However, there is no validation of fees that the user agrees to pay when opening a loan. This can result in unexpected losses for the user if the order is filled with a reserve with a large fee.

Note: We do **not** assume any malicious behavior from the lending market creator.

The only **precondition** is that there are **two reserves for the same mint with different origination fees**.

The issue arises because the **order executor is a different entity from the obligation owner**, meaning the executor can intentionally choose the reserve with the higher fee.Moreover, at the time the order was placed, there may have been **only one reserve** for that mint, leading the user to assume it would be used for their loan — but that assumption can be broken if a second reserve is later added.

Recommendations: Introduce a user provided var `max_fee` when the order is created, which would later be compared to the fee *f*rom the borrowed reserve.

Customer's response: We confirmed that we do not want to treat this particular "malicious / irresponsible market owner" case individually. The concern is much broader and we will need more comprehensive plan when we decide to address it in future.

M-02 Race condition in borrow order update and filling

Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: borrow_order_operations.rs	Status: Fixed	Related to PR#538

Description: If an obligation owner attempts to **reduce the** `remaining_debt_amount` **from X to Y**, but **at the same time**, a transaction is executed that **fills the original order for the full amount X**, the system will end up registering an **order for a total of X + Y**.

This occurs because the reduction and the fill happen concurrently, resulting in an **overlapping update** to the same order state.

Recommendations: Introduce a user provided arg and compare it against `remaining_debt_amount` to determine if the user if there isn't race condition.

Customer's response: Fixed in PR#610

Fix Review: Fix confirmed.

Low Severity Issues

L-01 Consider cleaning expired orders before update		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: handler_set_borrow_order.rs#L8-L9	Status: Fixed	Related PR#560

Description:

To be consistent in cases when lending market has disabled creation of borrow order, but there are existing expired orders, consider cleaning them up to mitigate creator expanding the order with an “update”

Recommendations: Enforce a check about `check_obligation_fully_refreshed_and_not_null` in `set_borrow_order`

Customer’s response: The user - depending on whether a refresh happened or not - would be able to "revive" an order or not. Indeed, it is better to be consistent. However, requiring obligation refresh on `set_borrow_order()` could be a bit of an overkill, so here is my suggested fix instead: #614

Fix Review: Fix confirmed.

L-02 Instant Liquidations from debt_term_seconds updates

Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: lending_operations.rs #L2375-L2378	Status: Acknowledged	Related to PR#537

Description: When `debt_term_seconds` is updated to a new value (smaller than the previous one) on a reserve, it can immediately trigger liquidations for existing obligations that have already borrowed from that reserve.
Recommendations: Consider introducing grace period when updating `debt_tern_seconds`

Customer's response: Fixed in PR 641

Fix Review: Fix confirmed

L-03 No Protection against setting debt terms on reserves with active borrows

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: [lending_operations.rs #L2375-L2378]	Status: Acknowledged	Related to PR#537

Description: The system lacks validation to prevent setting `debt_term_seconds` on reserves that currently have active borrows.

Without this safeguard, an attacker (or simply a fast user) could front-run the config update with a borrow instruction, creating positions that bypass the new term semantics and lead to inconsistent liquidation eligibility.

Recommendations: Disallow setting or shortening `debt_term_seconds` on any reserve that has active borrows. This protects users from unexpected behavior where existing debts would not become liquidatable under the intended "obligation debt term" policy:

```rust
fn validate_debt_term_update( reserve: &Reserve, new_debt_term_seconds: u64, ) -> Result<()>
{
// Prevent setting debt term on reserves with active borrows
 if reserve.liquidity.borrowed_amount_sf > 0
  { return err!(LendingError::CannotSetDebtTermOnReserveWithActiveBorrows);
 } ...
}
```

Customer's response: Acknowledged

Informational Issues

I-01. Divergence between documented behavior and implementation (PR#538)

Description:
In the `set_borrow_order` handler a comment on the `filled_debt_destination` states that it can be freely changed throughout the order's lifetime.

which is incorrect as the update will revert if one tries to modify this field with a call to `borrow_order_operations::set_borrow_order`. Moreover, in `borrow_order_operations::set_borrow_order` the doc-comment states that

Another location related to PR#541:

Comment above `borrow_order_execution_enabled` - `Whether the borrow orders should be evaluated during liquidations`, is irrelevant for the purpose that the value

Recommendation: Fix the documentation

Customer's response: Fixed in 9aa17 and 9cc1923
Fix Review: Fix confirmed

I-02. Consider introducing an incentive for the borrow order executor (PR#540)

Customer's response: Planned for a future update

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.