certora

Security Assessment



Kamino – KLend Withdraw Queue

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
Critical Severity Issues 9
C-01 Token-2022 Rent Mismatch Could DoS Queue 9
Medium Severity Issues 12
M-01 User can grief order executors to collect fees from closing TA accounts 12
Low Severity Issues 13
L-01 `min_withdraw_queued_liquidity_value` Not Enforced on Config Update 13
L-02 Token-2022 Extension Validation at Withdraw Instead of Enqueue 15
Informational Issues 17
I-01. Anchor Constraint for `collateral_token_program` 17
I-02. Batch Withdrawals and Min Check Over Batches 17
I-03. Large single-ticket withdrawals can temporarily overload the queue 18
I-04. `post_transfer_vault_balance_liquidity_reserve_checks` missing 19
I-05. `emergency_mode_disabled` missing for `seed_deposit_on_init_reserve` method 19
Disclaimer 20
About Certora 20

certora

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Klend	https://github.com/Kamino-Finance/klend-private	f26fb796	Solana

Project Overview

This document describes findings discovered during a manual code review. The work was undertaken from **25.08.2025** to **17.11.2025.** Fixes and "additional" are reviewed after the end date.

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

- PR#531, PR#533, PR#544, PR#583, PR#592, PR#593
- Fixes: PR#598, PR#609, PR#617, PR#634, PR#637
- Additional: PR#653, PR#670, PR#671, PR#672, PR#675, PR#680, PR#681, PR#683

Most PRs are included in this diff: fe1ad105...f26fb796

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.

Withdraw queue (ticketed withdrawals)

APIs: `enqueue_to_withdraw`, `withdraw_queued_liquidity`, `recover_invalid_ticket_collateral` in `lib.rs` lines 439–464.

- Introduces a FIFO withdraw queue that reserves liquidity for queued withdrawals ahead of other operations; the reserve tracks `queued_collateral_amount` and monotonically increasing sequence numbers.
- Market-level gates: issuance and redemption can be toggled independently; redemption requires a configured `min_withdraw_queued_liquidity_value` to avoid dust exploits.

How it works

- `enqueue_to_withdraw`: user locks cTokens into a per-owner queued collateral vault and receives a `WithdrawTicket` PDA recording: sequence number, owner, reserve, destination liquidity account, queued collateral, timestamp. Issuance is blocked if disabled. Destination TA must be non-delegated and unfrozen.
- `withdraw_queued_liquidity` (permissionless): when a ticket reaches the head of the queue and redemption is enabled, the handler:
- Verifies/creates the owner's destination ATA if missing (prepaid rent is taken at enqueue and refunded on creation), or marks the ticket invalid if the destination is incompatible.
- Refreshes the reserve, computes the withdrawable liquidity via the current exchange rate, enforces `min_withdraw_queued_liquidity_value`, burns queued collateral from the owner's queued collateral vault, and transfers liquidity from the reserve supply to the recorded destination.

- Fully consumed tickets are closed; partial redemptions keep the ticket in-place with reduced `queued_collateral_amount`. Returns `true` if processed, `false` when the ticket was invalidated and skipped.
- `recover_invalid_ticket_collateral`: allows the owner to recover remaining queued collateral for tickets that were marked invalid (e.g., destination TA repurposed), preserving queue liveness.
- `WithdrawQueue` locks liquidity for tickets (`queued_collateral_amount`) and advances `next_issued_ticket_sequence_number` and `next_withdrawable_ticket_sequence_number`. The reserve's "available" liquidity first serves ticketed withdrawals; surplus can serve other ops.

Findings Summary

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

Severity	Discovered	Fixed
Critical	1	1
High	-	-
Medium	1	1
Low	2	1
Informational	5	2
Total	9	5

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
C-01	Token-2022 Rent Mismatch Could DoS Queue	Critical	Fixed
M-01	User can grief order executors to collect fees from closing TA accounts	Medium	Fix introduced C-01
L-01	`min_withdraw_queued_liquidity_value` Not Enforced on Config Update	Low	Acknowledged
L-02	Token-2022 Extension Validation at Withdraw Instead of Enqueue	Low	Fixed
I-01	Anchor Constraint for `collateral_token_program`	Informational	Fixed
I-02	Batch Withdrawals and Min Check Over Batches	Informational	Acknowledged
I-03	Large single-ticket withdrawals can temporarily overload the queue	Informational	Acknowledged
I-04	`post_transfer_vault_balance_liquidity_reserve_checks` missing	Informational	Fixed

I-05	`emergency_mode_disabled` missing for `seed_deposit_on_init_reserve` method	Informational	Acknowledged

Critical Severity Issues

C-01 Token-2022 Rent Mismatch Could DoS Queue

Severity: **Critical**	Impact: **High**	Likelihood: **High**
Files: programs/kamino-lending/src/utils/token_transfer.rs:434-449 (prepayment) programs/kamino-lending/src/utils/token_transfer.rs:451-459 (refund) programs/kamino-lending/src/handlers/handler_withdraw_queued_liquidity.rs:43-75 (ATA creation branch)	Status: Fixed	Related to Withdraw Queue

Description:

- If extensions raised the required rent, the refund would fail to cover executor's cost or lead to repeated failures. The updated implementation computes exact rent using Token-2022 extensions.

```rust
pub fn destination_ata_rent_prepayment_transfer<'a>(
    owner: AccountInfo<'a>,
    withdraw_ticket: AccountInfo<'a>,
    system_program: AccountInfo<'a>,
) -> Result<()> {
    system_program::transfer(
        CpiContext::new(
            system_program,
            system_program::Transfer {
                from: owner,
                to: withdraw_ticket,
            },
        ),
        Rent::get()?.minimum_balance(TokenAccount::LEN),
    )
}
```

- Side effect of this bug ::: When the ticket owner uses an ATA as the destination, the protocol prepays rent into the ticket at enqueue and later refunds that rent to the executor who creates the ATA. On Token-2022 mints with extensions, required account size exceeds classic SPL token account size; naive prepay using classic size underfunds rent for the executor, enabling rent mismatches and griefing.

Basically on Tokens with extension, if the user close the ata the bot losses since he needs to pay more rent than the refunded amount. The user can then close the ATA and receive the extra rent.

```rust
let require_closing_ticket = match destination_ta_validity {
    DestinationTokenAccountValidity::AtaToBeCreated => {
        msg!("User's destination liquidity ATA does not exist; creating it");
        token_transfer::destination_ata_rent_refund_transfer(
            ctx.accounts.withdraw_ticket.to_account_info(),
            ctx.accounts.payer.to_account_info(),
            token_transfer::compute_token_account_rent_lamports_for_mint(
                &ctx.accounts.user_destination_liquidity.owner,
                &ctx.accounts.user_destination_liquidity,
            )?,
        )?;
        create_ata(...)?;
        true
    }
    ...
}
```

Recommendation

- Compute the rent based on the real account size.

Customer's response: Addressed in PR#634

Fix Review: Fix looks good

Medium Severity Issues

M-01 User can grief order executors to collect fees from closing TA accounts

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: handler_withdraw_queued_liquidity.rs#L28-L40	Status: Fixed	Related to PR#531

Description: Basically, to protect from a DoS where an exploiter closes his TA after creating a ticket, we make some checks to process the ticket closing if the account is not an ATA and is closed. However, if the account address matches the ATA address for the corresponding user and mint, we revert because anyone can initialize the ATA and close the ticket.

However, an exploiter can grief the bot/user that is closing the ticket to create the ATA and then close the account to claim the rent. The rent for an ATA is 0.00203928 SOL which is currently ~$0.5, but the exploit can repeated all over again for almost no cost for the exploiter

Recommendations: If the wallet is close ATA, treat is as `NonFixable`

Customer's response: Addressed in PR#598 by enforcing the creator to pre-pay for the ATA creation

Fix Review: Fix introduced C-01

Low Severity Issues

L-01 `min_withdraw_queued_liquidity_value` Not Enforced on Config Update

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: programs/kamino-lending/src/handlers/handler_update_lending_market.rs:217-222 programs/kamino-lending/src/utils/consts.rs:57	Status: Acknowledged	Related to Withdraw Queue

Description: The config update handler does not validate or constrain the minimum value; an unsafe value can be set and later enforced at withdraw time.

```rust
UpdateLendingMarketMode::UpdateMinWithdrawQueuedLiquidityValue => {
    config_items::for_named_field!(&mut market.min_withdraw_queued_liquidity_value)
        .validating(validations::check_not_zero)
        .set(&value)?;
}
```

```rust
/// The default minimum value that can be withdrawn in a single `withdraw_queued_liquidity()` call.
pub const MIN_WITHDRAW_QUEUED_LIQUIDITY_VALUE: u64 = 2;
```

Recommendation: Validate min threshold on update (bounds and units), and ensure consistency with withdraw path.

Customer's response: Acknowledged "the MIN_WITHDRAW_QUEUED_LIQUIDITY_VALUE const (as described in its rustdoc) is the default, to be used when a new market is initialized. We acknowledge that for the existing markets we will have to manually configure the runtime config value (very likely to $2)"

L-02 Token-2022 Extension Validation at Withdraw Instead of Enqueue

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: programs/kamino-lending/src/lending_market/lending_checks.rs:153-157 (validation at withdraw) programs/kamino-lending/src/lending_market/lending_checks.rs:128-131 (no validation at enqueue)	Status: Fixed	Related to Withdraw Queue

Description: Validation of Token-2022 liquidity token extensions is performed when withdrawing queued liquidity; misconfigured destinations are discovered late and can DOS the queue.

```rust
pub fn withdraw_queued_liquidity_checks(...) { /* validates token 2022 extensions */ }
```

Recommendation: Mirror extension checks (as applicable) at enqueue to fail early or mark the ticket Invalid at issuance if the destination cannot be used as configured; Generally, withdrawing a ticket should be non-blocking.

Customer's response: Fixed "We have to perform all the checks at withdraw, because the state can be changed after the enqueue. I agree, in the name of "fail fast", we can additionally check the same at enqueue. Also, you make a very good point about the token extensions validation: when user_destination_liquidity TA fails the validation, we should mark the ticket as invalid (because returning an error stalls the queue!). However, the other validations (reserve version and

status, Mint extensions) are more of “sanity checks” (i.e. they only fail if some state is fundamentally broken, and most other handlers also fail, and a stalled queue is the least of our problems).”

Informational Issues

I-01. Anchor Constraint for `collateral_token_program`

Location:
programs/kamino-lending/src/handlers/handler_enqueue_to_withdraw.rs:138-143

Description:
Add an explicit account owner check to ensure the provided `collateral_token_program` matches the owner of `user_source_collateral_ta` to fail fast at account validation.

Recommendation:

```rust
#[account(
    constraint = collateral_token_program.key() ==
user_source_collateral_ta.to_account_info().owner
)]
pub collateral_token_program: Program<'info, Token>,
```

Customer's response: Fixed

I-02. Batch Withdrawals and Min Check Over Batches

Location:
programs/kamino-lending/src/lib.rs:502-506

Description:
To mitigate small-ticket DoS and reduce on-chain overhead, consider enabling batch execution of multiple small withdrawals and applying the minimum over the batch amount.

Recommendation:
Introduce a batch redemption instruction that iterates over tickets up to a CU/size limit and enforces the minimum on the aggregate.

Customer's response: Acknowledged

I-03. Large single-ticket withdrawals can temporarily overload the queue

Location:
programs/kamino-lending/src/handlers/handler_withdraw_queued_liquidity.rs:65

Description:
If a user closes their ATA, the protocol must recreate it. Since the user only prepays the rent once, only a full withdrawal is permitted in such cases. This can become problematic if a whale attempts to withdraw a very large amount in a single ticket, potentially causing a DoS in the withdraw queue for an extended period, which is undesirable.

Recommendations:
- Invalidate the withdraw ticket if the user closes their ATA and full fulfillment isn't possible. (Simple solution, but maybe some MEV bots can take advantage of this);
- Virtual partial withdraw: add a one-time "refund_spent" flag on the ticket and allow partial redemptions after ATA creation; or preflight "only create ATA if full redemption possible";
- Add a rule that a single withdraw ticket cannot withdraw more than 1% of the total reserve. Large withdrawals should be split into smaller tickets.

Customer's response: Acknowledged

I-04. `post_transfer_vault_balance_liquidity_reserve_checks` missing

Location:
src/handlers/handler_seed_deposit_on_init_reserve.rs#L39

Description:
For consistency we recommend adding `post_transfer_vault_balance_liquidity_reserve_checks` in `handler_seed_deposit_on_init_reserve`.

Customer's response: Fixed

I-05. `emergency_mode_disabled` missing for `seed_deposit_on_init_reserve` method

Location:
programs/klend/src/lib.rs#L83

Customer's response: Acknowledged, "even if somebody would call seed_deposit_on_init_reserve, this reserve is unused, since somebody needs to call update reserve config to enable deposits in it. Most of the admin instructions are not access controlled".

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.