certora

Security Assessment

Final Report



Kamino – Klend – 1.17.0 release

March 2026

certora

Table of Contents

Project Summary **3**
Project Scope 3
Protocol Overview 5
Assessment Methodology **6**
Threat and Security Overview **7**
Findings Summary **13**
Severity Matrix 13
Detailed Findings **14**
High Severity Issues **16**
H-01: Rollover into Existing Target Borrow Overwrites Penalty State 16
Medium Severity Issues **19**
M-01: Cross-Reserve Accrual Transfer Defeats Per-Reserve Minimum Interest Guarantee 19
M-02: Borrow Orders Do Not Constrain Early Repay Penalty Parameters 21
M-03: Liquidation Does Not Update Early Repay Penalty Parameters 23
M-04: Disproportionate Early Repay Penalty for Late-Term Borrows 26
M-05: Early Repay Penalty Bypass via Staggered Multi-Obligation Grace Period Rotation 28
M-06: Rollover into Near-Maturity Reserve Traps Borrower 31
M-07: `clone_reserve_config` Bypasses the `is_immutable` Market Guard 32
Low Severity Issues **34**
L-01: Flash Loan Utilization Manipulation to Reduce Early Repay Penalty 34
L-02: Forced Term Extension via Permissionless Rollover Front-Running 36
L-03: A user can bypass part of early repay penalty if he rollover into open-term debt 37
L-04: Apply the min withdraw ticket check when cancelling a withdraw ticket 39
Informational Severity Issues 40
I-01: Validate the amount set in UpdateEarlyRepayRemainingInterestPct 40
I-02: “constraint = withdraw_ticket.load()?.owner == owner.key()” is a duplicate to the above “has_one = owner” 40
I-03: Consider checking whether the target maturity_timestamp has already ended and allowing the caller to specify a new or, or document the behavior 41
Disclaimer **42**
About Certora **42**

certora

Project Summary

Project Scope

This document describes the security review of **Kamino - Klend - Repay Penalty For Fixed Term Borrowing, Early repay penalty v2, implementation of cancel withdraw ticket handler and cloning a reserve config handler**. The work was undertaken from **February 16th, 2026** to **February 23rd, 2026 and from March 10 to March 17** .

The following list of PRs is included in our scope:

PR	Branch	Commit	Description
#629	liviu/early-repay-fee	a84169b1	Early repay penalty feature
#679	liviu/early-repay-rollover	6b14e0a	Preserve early repay fields on fixed-to-fixed different-reserve rollover (not merged into primary)
#717	liviu/early-repay-rollover-open-term	107fbdf	Settle outstanding penalty as additional debt on fixed-to-open rollover (builds on above)
#730	liviu/remove-grace-period	9e560e3f	Remove grace period mechanism entirely from ReserveConfig and get_early_repay_penalty
#742	feature/always_reset_debt_timestamp_on_borrow	ad377c9	Always set borrowed_at timestamp on borrows

#743	liviu/early-repay-revisited	eaa6061	Early repay penalty v2
#668	feature/clone_reserve_config_handler	e4e1caa	Implement clone_reserve_config() handler
#708	feature/cancel_withdraw_ticket	7aaac47	Cancel withdraw ticket handler impl (via tombstoning)

Protocol Overview

Klend Repay Penalty:

- Kamino Klend's early repay penalty is a mechanism that enforces a minimum interest obligation on fixed-term borrows. When a reserve is configured with a non-zero debt_term_seconds, a borrower takes on a commitment to hold the position for at least a minimum period before repaying without penalty.
  Configuration knobs (per reserve):
- debt_term_seconds — the fixed term duration; 0 means open-term (no penalty);

V2 of the implementation replaces the first fixed-term borrow penalty model with a simpler, more predictable one. Old model computed a "due interest" at borrow time (early_repay_due_interest) representing the minimum interest the protocol expected over the minimum interest period, and an "accrued interest" tracker updated continuously. The penalty was the gap between the proportional due amount and the already-accrued amount. This model had correctness issues: subsequent borrows on the same obligation didn't update the due-interest baseline, and liquidations skipped the penalty update entirely.

The new model computes the penalty at repay time as a function of the remaining term:

penalty = repay_amount × rate × (remaining_secs / year_secs) × early_repay_remaining_interest_pct

clone_reserve_config handler

Copies an entire ReserveConfig from a source reserve to a target reserve that shares the same liquidity mint. Target must be pre-deposit (fresh, usage-blocked). The caller — market owner or proposer authority — can pass optional overrides: a fixed borrow rate curve, a different debt_term_seconds, or a flag to zero out elevation groups. Aside from those overrides the config is copied verbatim; status is forced to Hidden, deposit/borrow limits are zeroed, and the proposer lock is cleared

cancel_withdraw_ticket handler (via tombstoning)

Lets a withdrawal ticket owner cancel part or all of their queued withdrawal and receive their ctokens back immediately. The queued_collateral_amount on the ticket is decremented (to zero on full cancel) and the reserve's queue total is reduced, but the ticket account is not closed — it stays in the queue as a zero-amount tombstone and is skipped when withdraw_queued_liquidity processes the queue in FIFO order. This prevents spam (closing would free the rent and encourage flooding), and keeps the queue ordering intact. Only the ticket owner can cancel; gated by a market-level is_withdraw_ticket_cancellation_enabled flag.

Assessment Methodology

Our assessment approach combines design level analysis with a deep review of the implementation to ensure that a protocol is secure, economically sound, and behaves as intended under realistic conditions.

At the design level, we evaluate the architecture, the economic assumptions behind the protocol, and the safety properties that should hold independently of a specific chain or environment. This process includes reviewing internal and cross protocol interactions, state transition flows, trust boundaries, and any mechanism that could be exploited to extract value, deny service, or alter core system behavior. At this stage, a focused threat modelling exercise helps identify key attack surfaces and adversarial capabilities relevant to the system. Design level issues often relate to incentive structures, governance implications, or systemic behavior that emerges under adversarial conditions.

Implementation analysis focuses on the concrete behavior of the code within the execution model of the target chain. This involves reviewing the correctness of logic, access control, state handling, arithmetic behavior, and the nuanced behaviors of the chain environment. Familiar classes of vulnerabilities such as reentrancy conditions, faulty permission checks, precision issues, or unsafe assumptions often surface at this layer. These findings require context aware reasoning that takes into account both the code and the architectural intent.

To support this analysis, the codebase is examined through repeated manual passes and supplemented by automated tools when appropriate. High-risk logic areas receive deeper scrutiny, invariants are validated against both design intent and actual implementation, and potential vulnerability leads are thoroughly investigated. Automated techniques such as static analysis, fuzzing, or symbolic execution may be used to complement manual review and provide additional insight.

Collaboration with the development team plays an important role throughout the audit. This helps confirm expected behaviors, clarify design assumptions, and ensure an accurate understanding of the protocol's intended operation. All findings are documented with clear reasoning, reproducible examples, and actionable recommendations. A follow up review is conducted to validate the applied fixes and verify that no regressions or secondary issues have been introduced.

certora

Threat and Security Overview

Actors and Trust Model

- **Lending Market Admin** — Trusted. Configures per-reserve penalty parameters (debt_term_seconds, early_repay_remaining_interest_pct).
- **Obligation Owner** — Trusted for own state. Takes borrows and repays; configures rollover settings that interact with penalty state.
- **Permissionless Executor** — Untrusted. Can trigger rollovers on any opted-in obligation, which resets timestamps.
- **Liquidator** — Untrusted. Partially or fully liquidates positions, which reduces principal without penalty.
- **Borrow Order Filler** — Untrusted. Permissionlessly fills borrow orders and chooses the target reserve, determining the penalty profile the order creator will be subject to.

Assets at Risk

- **Protocol fee revenue**: the penalty amount owed to the reserve's accumulated protocol fees, representing the minimum interest guarantee
- **Obligation owner's penalty expectations**: a borrower's assumption about when and how much penalty applies must not be invalidated by third-party actions
- **Reserve depositors' yield**: the minimum interest guarantee is effectively a yield floor for the reserve's lenders; violations deprive them of expected income

Trust Assumptions

- **Admin configuration is correct**
- **Oracle prices** are accurate and fresh at the time of LTV-triggered forced repays (liquidation correctness is a precondition for penalty calculation being meaningful)
- **debt_term_seconds = 0 (open-term)** reserves are fully out of scope for penalty mechanics — all penalty logic gates on debt_term_seconds > 0

Attack Vectors

- **Liquidation bypasses penalty accumulation** — liquidators repay fixed-term debt at zero penalty cost to the protocol.
- **Dust borrow resets penalty clock** — borrow() unconditionally sets last_borrowed_at_timestamp = current_timestamp on every call, including top-ups on an existing position. Adding a 1 lamport dust borrow near term end (when remaining_secs and thus penalty are small) resets debt_maturity_timestamp to now + debt_term_seconds, extending the term a full cycle. This is self-controlled (requires borrower's own signature) and can be used to time the timer favorably.
- **Penalty computed with current rate at repay time** — the penalty formula uses reserve.calculate_future_cumulative_borrow_rate() at the moment of repay, not a snapshotted rate from borrow time. A borrower can flash-inflate reserve liquidity immediately before repaying (dropping utilization and thus the borrow rate), execute repay at the artificially low rate, and withdraw the flash deposit in the same transaction — paying a fraction of the intended penalty.
- **Permissionless rollover front-running during grace period** — any Signer can call rollover_fixed_term_borrow if auto_rollover_enabled = 1. An observer can front-run a borrower's penalty-free grace period repay with a rollover, resetting last_borrowed_at_timestamp to now and forcing the repay to incur a fresh full-term penalty.
- **Same- and different-reserve rollover settle no penalty** — same-reserve rollover only updates last_borrowed_at_timestamp; different-reserve rollover calls the obligation-level repay() which does not trigger penalty accumulation. Any outstanding penalty at rollover time is silently discarded in both paths.
- **Rollover into near-maturity target reserve** — check_rollover_possible validates debt_term_seconds >= min_debt_term_seconds (relative config field) but does not check debt_maturity_timestamp - now. A borrower can roll into a target reserve whose maturity is only seconds away, effectively landing in an already-expired or imminently-expiring fixed-term position with a trivially small penalty on any subsequent repayment.
- **Borrow orders don't constrain penalty parameters** — BorrowOrderConfig validates max_borrow_rate_bps and min_debt_term_seconds but not early_repay_remaining_interest_pct. A permissionless filler selects a reserve with aggressive penalty config.

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	1	1	1
Medium	7	7	6
Low	4	4	2
Informational	3	3	3
Total	15	15	12

Severity Matrix

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

certora

Detailed Findings

ID	Title	Severity	Status
H-01	Rollover into Existing Target Borrow Overwrites Penalty State	High	Fixed
M-01	Cross-Reserve Accrual Transfer Defeats Per-Reserve Minimum Interest Guarantee	Medium	Fixed
M-02	Borrow Orders Do Not Constrain Early Repay Penalty Parameters	Medium	Acknowledged
M-03	Liquidation Does Not Update Early Repay Penalty Parameters	Medium	Fixed
M-04	Disproportionate Early Repay Penalty for Late-Term Borrows	Medium	Fixed
M-05	Early Repay Penalty Bypass via Staggered Multi-Obligation Grace Period Rotation	Medium	Fixed
M-06	Rollover into Near-Maturity Reserve Traps Borrower	Medium	Acknowledged
M-07	`clone_reserve_config` Bypasses the `is_immutable` Market Guard	Medium	Fixed
L-01	Flash Loan Utilization Manipulation to Reduce Early Repay Penalty	Low	Acknowledged

L-02	Forced Term Extension via Permissionless Rollover Front-Running	Low	Partially fixed
L-03	A user can bypass part of early repay penalty if he rollover into open-term debt	Low	Acknowledged
L-04	Apply the min withdraw ticket check when cancelling a withdraw ticket	Low	Fixed
I-01	Validate the amount set in UpdateEarlyRepayRemainingInterestPct	Informational	Fixed
I-02	“constraint = withdraw_ticket.load()?.owner == owner.key()” is a duplicate to the above “has_one = owner”	Informational	Fixed
I-03	Consider checking whether the target maturity_timestamp has already ended and allowing the caller to specify a new or, or document the behavior	Informational	Fixed

High Severity Issues

H-01: Rollover into Existing Target Borrow Overwrites Penalty State

Severity: High	Impact: High	Likelihood: Medium
Files: lending_operations.rs:1030-1045 obligation.rs:392-414	Status: Fixed	Related to v1 of repay early repay penalty

Description:

After PR#679, rollover_borrow_into_different_reserve does:

```rust
obligation.repay_fully(repaid_amount, source_borrow_index); // zeroes source slot

let (target_borrow, _) = obligation.find_or_add_liquidity_to_borrows(

    target_reserve_address, ...

)?;

target_borrow.borrow(borrowed_amount);

target_borrow.fixed_term_borrow_rollover_config = preserved_rollover_config;

target_borrow.early_repay_accrued_interest = preserved_early_repay_accrued; // unconditional

target_borrow.early_repay_due_interest = preserved_early_repay_due; // unconditional
```

find_or_add_liquidity_to_borrows searches for an existing borrow from target_reserve_address first and returns it if found (obligation.rs:398-399). There is no check in the rollover path to distinguish "found existing borrow" from "created new slot." No guard prevents rolling into a reserve where the obligation already has a borrow.

rollover_borrow_into_different_reserve (PR#679) reads early_repay_accrued_interest, early_repay_due_interest, and fixed_term_borrow_rollover_config from the source borrow, then unconditionally writes them to whatever find_or_add_liquidity_to_borrows returns for the target reserve. When the obligation **already has an active borrow in the target reserve**, find_or_add_liquidity_to_borrows returns that existing borrow rather than creating a fresh one — and the source values overwrite the target's penalty state in place.

Attack Scenario

Configuration: Two fixed-term USDC reserves (USDC-1, USDC-2), same mint, same debt term, rollover enabled.

Step	Action	USDC-2 borrow state
T-29d	Borrow dust from USDC-1 (same config)	USDC-1: due=95, accrued=0 (near term end)
T=0	Borrow large amount from USDC-2 (100% min_interest_period_pct)	due=10000, accrued=0
T+1d	USDC-1 near term end: accrued=94 ≈ due=95; enters rollover window	USDC-1: due=95, accrued=94 (near term end)
T+1d	USDC-1 enters rollover window; roll USDC-1 → USDC-2	due=95, accrued=94 ← OVERWRITTEN
T+1d	Repay full USDC-2 position	Penalty ≈ 1 instead of ≈ 10000

The attacker reduces the penalty on their large USDC-2 position from ~10000 lamports to ~1 lamport by using a cheap dust borrow in USDC-1 near its term end (where accrued ≈ due). The only cost is the dust penalty on the USDC-1 rollover itself (≈ 1 lamport).

The same mechanism can be used to overwrite fixed_term_borrow_rollover_config, replacing the existing USDC-2 borrow's user-configured rollover permissions with those of the source borrow.

Recommendations:

Instead of assigning source_borrow penalty related values to target_borrow, use += to just increase the corresponding early_repay_accrued_interest & early_repay_due_interest

Customer Response:

Fixed in PR722.

Fix Review:

Fix confirmed.

Medium Severity Issues

M-01: Cross-Reserve Accrual Transfer Defeats Per-Reserve Minimum Interest Guarantee

Severity: Medium	Impact: Medium	Likelihood: Medium
Files: programs/klend/src/lending_market/lending_operations.rs	Status: Fixed	Related to v1 of repay early repay penalty

Description:

early_repay_accrued_interest is semantically tied to a single lender pool: it represents interest earned **by and distributed to** a specific reserve's depositors. The additive merge treats it as interchangeable across pools.

- Reserve A: `accrued = 1000, due = 500` — Reserve A's lenders have received 1000 units; 500 units of excess exists beyond the min obligation.
- Reserve B (new borrow): `accrued = 0, due = 500` — Reserve B's lenders have received nothing.
- After additive merge: `combined_accrued = 1000, combined_due = 1000` → penalty = 0.

The 500 excess units from Reserve A are applied toward Reserve B's lender obligations — but those units were already paid out to Reserve A's depositors. Reserve B's depositors received nothing from this borrower and have no recourse.

The same-reserve case is fine by contrast. If a borrower holds the same reserve position until accrued >> due, the same depositor pool received all of the excess. There is no cross-pool free-riding, and the lender guarantee is intact.

Attack Scenario

1. Hold a borrow in Reserve A until `accrued = N × due` (excess = `(N-1) × due_A`). The longer it's held, the larger the banked "penalty credit."
2. Open a new borrow in Reserve B with `due_B ≤ (N-1) × due_A`.

3. Roll A → B (additive merge): `combined_accrued ≥ combined_due` → penalty = 0 immediately.

4. Repay Reserve B at once. Reserve B's lenders receive zero interest despite holding term risk.

Recommendations:

If merging must be supported, treat penalty fields as non-transferable: settle the source's outstanding penalty at rollover time, then initialise the target's fields fresh via update_early_repay_due_interest_on_borrow on the combined principal. Discard any source excess accrual rather than transferring it.

Customer Response:

Fixed in v2 logic of early repay penalty.

Fix Review:

Fix confirmed.

M-02: Borrow Orders Do Not Constrain Early Repay Penalty Parameters

Severity: Medium	Impact: Medium	Likelihood: Low
Files: programs/klend/src/state/borrow_order_operations.rs	Status: Acknowledged	Related to v1 of repay early repay penalty

Description:

BorrowOrderConfig lets the creator constrain max_borrow_rate_bps and min_debt_term_seconds, both validated in fill_borrow_order (borrow_order_operations.rs:80-127). However, there is no way to constrain early_repay_min_interest_period_pct or early_repay_grace_period_hours (the second one is removed in PR730) — the two reserve-level parameters that determine the early repay penalty.

Since filling is permissionless (handler_fill_borrow_order.rs:137 — payer is any Signer), the filler chooses the reserve. When the fill executes, update_early_repay_due_interest_on_borrow (obligation.rs:861-882) computes early_repay_due_interest using the filling reserve's early_repay_min_interest_period_pct — a value the order creator had no opportunity to constrain, which may result in unexpected behavior

Root Cause

fill_borrow_order validates two reserve properties against the order:

1. `reserve.max_borrow_rate_bps <= order.max_borrow_rate_bps`
2. `reserve.debt_term_seconds >= order.min_debt_term_seconds`

It does not validate reserve.early_repay_min_interest_period_pct or reserve.early_repay_grace_period_hours. Two reserves with identical mint, rate, and term can have vastly different penalty profiles (e.g., 10% vs 90% min interest period, 120h vs 0h grace period).

Impact Description

A borrow order creator may receive a loan with early repay penalty terms they did not anticipate. The filler-chosen reserve could have a high early_repay_min_interest_period_pct or low early_repay_grace_period_hours, locking the borrower into a substantial penalty they never agreed to.

NOTE that in combination with Disproportionate Early Repay Penalty for Late-Term Borrows it may result in borrowing from a reserve where the user has an active obligation, which may lead to having early_repay_due_interest which exceeds the possible interest for the entire debt term, so the victim is enforced to always pay penalty

Recommendations:

1. **Add `max_early_repay_min_interest_period_pct` to `BorrowOrderConfig`.** Validate in `fill_borrow_order` that the filling reserve does not exceed this threshold.

Customer Response:

Acknowledged.

Fix Review:

Acknowledged.

M-03: Liquidation Does Not Update Early Repay Penalty Parameters

Severity: Medium	Impact: Medium	Likelihood: Medium
Files: programs/klend/src/lending_market/lending_operations.rs	Status: Fixed	Related to v1 of repay early repay penalty

Description:

When a fixed-term debt position is liquidated, the liquidation code path does not update early_repay_due_interest or early_repay_accrued_interest, nor does it charge a penalty to the liquidator. This causes the min_interest_period penalty window to effectively extend beyond its intended duration, because the reduced principal accrues interest slower while early_repay_due_interest remains calibrated for the original (larger) principal.

Root Cause

The normal repay path (repay_obligation_liquidity at lending_operations.rs:760-761) calls both update_on_early_repay() (which proportionally reduces early_repay_due_interest and early_repay_accrued_interest) and accumulate_early_repay_penalty() (which credits the penalty to the reserve's protocol fees).

The liquidation path (repay_and_withdraw_from_obligation_post_liquidation at lending_operations.rs:2728-2731) only calls repay_reserve.liquidity.repay() and obligation.repay() — it skips both the penalty calculation and the parameter update entirely.

After liquidation, early_repay_due_interest remains at its original value (calibrated for the pre-liquidation principal), but early_repay_accrued_interest now grows slower because interest accrual in accrue_interest() (obligation.rs:748-753) is proportional to the current (reduced) borrowed_amount. This means the accrued interest takes longer to catch up to the due interest, extending the effective penalty window beyond the intended min_interest_period.

Example Scenario

Configuration: 1000 SOL borrowed at day 0, 30-day debt term, 10% APR, early_repay_min_interest_period_pct = 50% (15 days).

At borrow time:

- `early_repay_due_interest` = ~4.11 SOL (interest for 15 days on 1000 SOL)
- `early_repay_accrued_interest` = 0
- Interest accrual rate: ~0.274 SOL/day

At day 13 — 50% liquidation (500 SOL liquidated):

- `borrowed_amount` drops from 1000 to 500 **SOL**
- `early_repay_due_interest` stays at 4.11 SOL (unchanged)
- `early_repay_accrued_interest` ≈ 3.56 SOL (13 days × 0.274 SOL/day, unchanged)
- Interest accrual rate halves to ~0.137 SOL/day (proportional to new 500 SOL principal)
- No penalty is charged

Without liquidation — repay at day 15:

- `early_repay_accrued_interest` ≈ 4.11 SOL (15 × 0.274), matching `early_repay_due_interest`
- Penalty = 0. The `min_interest_period` ends exactly as designed.

With liquidation — remaining 500 SOL after day 13:

- At day 15: `early_repay_accrued_interest` ≈ 3.56 + (2 × 0.137) = 3.83 SOL. Still below 4.11. **Penalty ≈ 0.28 SOL.**
- At day 17: `early_repay_accrued_interest` ≈ 3.56 + (4 × 0.137) = 4.11 SOL. Penalty finally reaches 0.
- The effective `min_interest_period` extended from **day 15 to day 17** — 2 days beyond the intended window.

Worse with larger liquidation — 90% liquidated (100 SOL remaining):

- Accrual rate drops to ~0.027 SOL/day
- Gap to close: 4.11 - 3.56 = 0.55 SOL at 0.027/day ≈ 20 more days
- Effective penalty window extends to **day 33** — past the entire 30-day debt term

• The borrower can never escape the penalty before term end, regardless of when they repay

Recommendations:

1. **Proportionally reduce `early_repay_due_interest` on liquidation.** When liquidating, scale down `early_repay_due_interest` by the same ratio as the debt reduction: `due_interest *= (remaining_debt / original_debt)`. This keeps the penalty parameters consistent with the reduced principal, preserving the original `min_interest_period` boundary.

2. **Proportionally reduce `early_repay_accrued_interest` on liquidation.** Apply the same proportional reduction to `early_repay_accrued_interest` so that the ratio between accrued and due interest is preserved, maintaining the same relative position within the penalty window.

Customer Response:

Fixed in v2 logic of early repay penalty.

Fix Review:

Fix confirmed.

M-04: Disproportionate Early Repay Penalty for Late-Term Borrows

Severity: Medium	Impact: High	Likelihood: Medium
Files: programs/klend/src/state/obligation.rs	Status: Fixed	Related to v1 of repay early repay penalty

Description:

A user who borrows midway through (or late in) a debt term may face an early repay penalty that is disproportionately large relative to the remaining time in the term. This happens because early_repay_due_interest is calculated based on the early_repay_min_interest_period at borrow time, without accounting for how much of the debt term has already elapsed.

Root Cause

When a user borrows, update_early_repay_due_interest_on_borrow calculates the interest that *would* accrue over the early_repay_min_interest_period (e.g. 40% of the debt term) and adds it to early_repay_due_interest. This calculation is independent of when in the debt term the borrow occurs.

If a user borrows at e.g. 50% of the debt term, the early_repay_min_interest_period (say 40% of the term) would extend beyond the term end — but the penalty is still computed as if the user should owe the full min-interest-period worth of interest.

Example Scenario

Config - 30-day term, 10% APR, 50% early_repay_min_interest_period_pct (= 15 days).

User borrows 1000 SOL at day 18 (60% through the term, 12 days remaining).

- `update_early_repay_due_interest_on_borrow` computes 15-day interest ≈ 4.11 SOL and stores it in `early_repay_due_interest`. The 15-day window extends to day 33 — 3 days past term end.
- Even at term end (day 30, 12 days held): `accrued` ≈ 3.29 SOL → penalty still ≈ 0.82 SOL

The min_interest_period covers 125% of the user's remaining term (15 days vs 12 days available). Any repay before day 33 incurs a penalty — the due_interest obligation cannot be fully satisfied before the term expires regardless of when the user repays.

Additionally - Grace Period Eliminated by PR#730

PR#730 removes the grace period check from get_early_repay_penalty entirely. There is no longer any penalty-free window at any point in the term: a user in Scenario A who waits until term end still pays ~0.82 SOL penalty to avoid liquidations after debt term end.

Impact Description

The team decided to remove the grace period in PR730, which makes the impact bigger

Recommendations:

Cap early_repay_due_interest to natural accrual before term end:

When computing early_repay_due_interest on borrow, cap it to the interest that would naturally accrue between now and debt_term_end at the current rate: due_interest = min(full_min_period_interest, interest_until_term_end). This ensures the penalty never exceeds what the user would owe if they simply held the position to term end.

Customer Response:

Fixed in v2 logic of early repay penalty.

Fix Review:

Fixed confirmed.

M-05: Early Repay Penalty Bypass via Staggered Multi-Obligation Grace Period Rotation

Severity: Medium	Impact: High	Likelihood: Medium
Files: programs/klend/src/state/obligation.rs	Status: Fixed	Related to v1 of repay early repay penalty

Description:

A user can completely bypass the early repay penalty mechanism on fixed-term reserves by maintaining multiple obligations with staggered first_borrowed_at_timestamp values. By exploiting the grace period window and the fact that first_borrowed_at_timestamp is anchored by the initial (dust) borrow and not updated on subsequent borrows, the user can ensure that at any point in time, at least one of their obligations is within its penalty-free grace period. Combined with permissionless auto-rollover, the attack is self-sustaining indefinitely.

Root Cause

Two design properties combine to enable the bypass:

1. **`first_borrowed_at_timestamp` is anchored by the first borrow regardless of size.** A 1 lamport dust borrow sets the timestamp identically to a 1M SOL borrow. Subsequent borrows on the same obligation do not update it (`obligation.rs:675`).

1. **The grace period check is size-agnostic.** The penalty calculation at `obligation.rs:924` only checks `current_timestamp >= debt_term_grace_period_start` — it does not consider when or how much was borrowed relative to the grace window.

Attack Scenario

Configuration: 30-day debt term, 5-day grace period (120 hours), fixed-term rates better than open-term.

Phase 1 — Setup (one-time, minimal cost)

The attacker creates 6 obligations using different id/seed values and places a dust borrow on each, staggered 5 days apart:

Obligation	Dust borrow at	Grace period window
#1	Day 0	Day 25 – Day 30
#2	Day 5	Day 30 – Day 35
#3	Day 10	Day 35 – Day 40
#4	Day 15	Day 40 – Day 45
#5	Day 20	Day 45 – Day 50
#6	Day 25	Day 50 – Day 55

Each dust borrow requires negligible collateral and anchors first_borrowed_at_timestamp.

Auto-rollover (auto_rollover_enabled = 1) is enabled on all obligations. As each approaches its term end, anyone can permissionlessly call rollover_fixed_term_borrow, which resets first_borrowed_at_timestamp to the current time — keeping the staggered timeline alive indefinitely without manual intervention.

Phase 2 — Exploitation (repeated)

1. At day 25, obligation #1 enters its grace period.
2. Attacker deposits real collateral into #1 and borrows a large amount. `first_borrowed_at_timestamp` remains at day 0 (unchanged by design).
3. The attacker holds the position for up to 5 days, penalty-free.
4. Before day 30, the attacker repays obligation #1 — `get_early_repay_penalty()` returns zero because `current_timestamp >= debt_term_grace_period_start`.
5. If the attacker wants to continue borrowing, they move collateral to obligation #2 (now in its grace period at day 30) and borrow there.

6. The cycle repeats across the 6 obligations indefinitely.

Result: The attacker obtains fixed-term borrowing rates (which are presumably more favorable than open-term) while maintaining the flexibility to repay at any time without penalty — effectively getting the benefits of both fixed and open-term borrowing.

Recommendations:

1. **Track borrow timing relative to grace period.** Record the timestamp of each borrow amount increment and apply the penalty proportionally based on how long each tranche has been held, rather than using a single `first_borrowed_at_timestamp` for the entire position.

2. **Enforce minimum borrow duration for penalty exemption.** Require that the borrowed amount has been held for at least `early_repay_min_interest_period_pct` of the debt term before the grace period exemption applies — i.e., new borrows placed during the grace period should still be subject to the penalty.

3. **Validate grace period against debt term on config update.** Add a check in `UpdateEarlyRepayGracePeriodHours` ensuring `grace_period_seconds <= debt_term_seconds * max_acceptable_ratio` to limit the attack surface. While this doesn't fully prevent the attack, it increases the number of obligations needed.

Customer Response:

Fixed in PR730 by removing the grace period mechanism.

Fix Review:

Fix confirmed.

M-06: Rollover into Near-Maturity Reserve Traps Borrower

Severity: Medium	Impact: High	Likelihood: Medium
Files: programs/klend/src/lending_market/lending_operations.rs	Status: Acknowledged	Related to PR743

Description:

When rolling a fixed-term borrow into a different reserve, check_rollover_possible calls check_borrow_possible on the target — which only blocks reserves whose debt_maturity_timestamp has already passed. There is no check on how soon the target reserve's maturity is in the future.

A borrower with auto_rollover_enabled = 1 can be maliciously rolled (the instruction is permissionless) into a target reserve that satisfies all user-configured constraints (min_debt_term_seconds, max_borrow_rate_bps) yet has a debt_maturity_timestamp only hours away. Because the rollover resets last_borrowed_at_timestamp to now, the borrower immediately owes a full early-repay penalty if they try to exit, while simultaneously facing imminent debt-maturity liquidation if they don't.

Recommendations:

Don't allow to rollover into a destination reserve if it's debt_maturity_timestamp < last_borrowed_at_timestamp + debt_term.

Customer Response:

Acknowledged. In the near future we are not planning to be supporting reserves with a maturity timestamp. However, when/if we do, I think they will only have the maturity timestamp, not also a debt term, and so we won't be charging a penalty for these reserves.

M-07: `clone_reserve_config` Bypasses the `is_immutable` Market Guard

Severity: Medium	Impact: High	Likelihood: Medium
Files: programs/klend/src/handlers/handler_clone_reserve_config.rs	Status: Fixed	Related to PR668

Description:

handler_update_reserve_config explicitly blocks config writes on immutable markets, with a narrow carve-out only for the global_admin using global-admin-only modes. handler_clone_reserve_config has no equivalent check: it writes an entirely new ReserveConfig into the target reserve with no regard for the market's immutability flag.

handler_update_reserve_config guards every write:

```rust
// handler_update_reserve_config.rs:31-36
require!(
    !market.is_immutable()
        || (is_update_reserve_config_mode_global_admin_only(mode)
            && ctx.accounts.signer.key() == ctx.accounts.global_config.load()?.global_admin),
    LendingError::OperationNotPermittedMarketImmutable
);
```

handler_clone_reserve_config has no such check. The only signer constraint is:

```rust
// handler_clone_reserve_config.rs:28-32
constraint = lending_operations::utils::is_allowed_signer_to_clone_reserve_config(
    signer.key(),
    target_lending_market.load()?.deref(),
    target_reserve.load()?.deref(),
) @ LendingError::InvalidSigner
```

Which resolves to lending_market_owner || (proposer_authority && !proposer_authority_locked) — neither branch checks is_immutable.

Recommendations:

Add an is_immutable check to CloneReserveConfig, mirroring update_reserve_config

Customer Response:

Fixed in the following commit.

Fix Review:

Fix confirmed.

Low Severity Issues

L-01: Flash Loan Utilization Manipulation to Reduce Early Repay Penalty

Severity: Low	Impact: Low	Likelihood: Low
Files: programs/klend/src/lending_market/lending_operations.rs	Status: Acknowledged	

Description:

An attacker can manipulate the reserve's utilization rate within a single transaction to artificially lower the borrow rate at the moment of borrowing, causing early_repay_due_interest to be computed with a deflated rate. After the manipulation is unwound in the same transaction, the actual borrow rate returns to its normal (higher) level, and early_repay_accrued_interest accrues at the real rate — clearing the artificially low early_repay_due_interest faster than intended and effectively shortening the min_interest_period.

Root Cause

early_repay_due_interest is a snapshot: it's computed at borrow time by projecting current_borrow_rate() over the min_interest_period. The borrow rate is a pure function of the reserve's current utilization ratio (borrowed / (available + borrowed)). By temporarily inflating available via a large deposit, the utilization drops and so does the borrow rate.

Meanwhile, early_repay_accrued_interest is incremented on each refresh_obligation based on the **actual** interest accrued (via accrue_interest() at obligation.rs:737-753), which uses the real cumulative borrow rate that reflects the true (higher) utilization after the manipulation is unwound.

This creates a mismatch: due_interest is locked in at the manipulated low rate, while accrued_interest grows at the real (higher) rate, causing the penalty to clear faster than the protocol intended.

Recommendations:

Snapshot the borrow rate at reserve level. Track a smoothed or minimum borrow rate over a window and use that for penalty calculations, preventing flash manipulation from affecting the due interest computation.

Customer Response:

Acknowledged - for debt term reserves the rate will be fixed, so this doesn't apply in this case

L-02: Forced Term Extension via Permissionless Rollover Front-Running

Severity: Low	Impact: Medium	Likelihood: Low
Files: programs/klend/src/handlers/handler_rollover_fixed_term_borrow.rs	Status: Partially fixed	Related to v1 of repay early repay penalty

Description:

A user who has enabled auto-rollover on a fixed-term borrow can be griefed when attempting to repay during the grace period. Because rollover_fixed_term_borrow is permissionless (any Signer can call it as payer) and the rollover window overlaps with the grace period, an attacker can front-run the user's repay transaction with a rollover call. This resets first_borrowed_at_timestamp to the current time, placing the user at day 0 of a fresh term — causing their repay to incur an unexpected early repay penalty.

Root Cause

The rollover window and the repay grace period both extend backwards from the debt term end and typically overlap:

```
|------------- debt term -------------|
                    |-- rollover window --|
                         |-- grace period --|
                    Day 23    Day 25      Day 30
```

Because rollover execution is permissionless, the user cannot safely repay during the grace period if auto-rollover is enabled — any observer can call rollover first, resetting the term and eliminating the grace period.

Attack Scenario

Configuration: 30-day debt term, 5-day grace period, 7-day rollover window. The user has auto_rollover_enabled = 1.

1. User's borrow reaches day 25 (enters grace period). The user intends to repay penalty-free.

2. User submits transaction to repay the obligation.

3. Another user calls `rollover_fixed_term_borrow`.

4. Rollover succeeds — `first_borrowed_at_timestamp` is reset to now (day 25). The user's debt term is now day 25 to day 55.

5. User's subsequent repay tx now hits the early repay penalty: they are at day 0 of a fresh 30-day term, with `early_repay_due_interest` recalculated for the full position.

Result: The user pays an unexpected penalty on a repay they intended to be penalty-free. The griefer pays only the gas cost of the rollover transaction.

Recommendations:

• Don't allow new borrow for active rollover/grace period windows

Customer Response:

Partially fixed in PR730.

Fix Review:

Practically, it's still possible to rollover the obligation and reset the var with the new name last_borrowed_at_timestamp which is used to calculate the repay penalty in v2.

L-03: A user can bypass part of early repay penalty if he rollover into open-term debt

Severity: Low	Impact: Medium	Likelihood: Low
Files: programs/klend/src/lending_market/lending_operations.rs	Status: Acknowledged	Related to PR743

Description:

A user can avoid the interest penalty for the rollover window, if he uses an open-term reserve as destination and then repay.

Recommendations:

Accrue the penalty as a loan from the target reserve, if the user has enough collateral.

Customer Response:

Acknowledged.

L-04: Apply the min withdraw ticket check when cancelling a withdraw ticket

Severity: Low	Impact: Medium	Likelihood: Low
Files: programs/klend/src/handlers/handler_cancel_withdraw_ticket.rs	Status: Fixed	Related to PR708

Description:

Merging the changes introduced by PR73(introduce min withdraw ticket value) and insert the check in the cancel withdraw ticket handler.

Recommendations:

Fixed

Customer Response:

Fixed confirmed.

Informational Severity Issues

I-01: Validate the amount set in UpdateEarlyRepayRemainingInterestPct

Files:

programs/klend/src/lending_market/lending_operations.rs (PR743)

Description:

Consider validating the amount provided to the function with check_valid_pct to ensure proper config.
The var is checked in validate_reserve_config_integrity, but it's still possible to set it with invalid amount after the creation of the reserve.

Customer Response:

Fixed in eaa60615.

Fix Review:

Fix confirmed.

I-02: “constraint = withdraw_ticket.load()?.owner == owner.key()” is a duplicate to the above “has_one = owner”

Files:

programs/klend/src/handlers/handler_cancel_withdraw_ticket.rs (PR708)

Description:

Consider removing one of the duplicated checks

Customer Response:

Fixed. Deleted the one that was giving less-specific error.

Fix Review:

Fix confirmed.

I-03: Consider checking whether the target maturity_timestamp has already ended and allowing the caller to specify a new or, or document the behavior

Files:

programs/klend/src/state/reserve.rs (PR668)

Description:

Cloning from a Reserve with an elapsed debt_maturity_timestamp silently produces a non-borrowable reserve. Consider allowing the caller to specify a new or, or document the behavior

Customer Response:

Fixed by documenting it - “I could not see a scenario where this kind of mistake could hurt us or the users.”

Fix Review:

Fix confirmed.

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.