

# Kamino Borrow Orders

Security Assessment

February 11th, 2026 — Prepared by OtterSec


Tamta Topuria tamta@osec.io

Ajay Shankar Kunapareddy d1r3wolf@osec.io

Artem Astakhov aikasimirov@osec.io

# Table of Contents


**Executive Summary** **2**

Overview 2

Key Findings 2

**Scope** **3**

**Findings** **4**

**Vulnerabilities** **5**

OS-KBO-ADV-00 | Front Running Borrow Order 6

OS-KBO-ADV-01 | Premature Liquidation on Uninitialized Timestamp 7

OS-KBO-ADV-02 | UpdateDebtMaturityTimestamp can Allow Immediate Liquidations 8

**General Findings** **9**

OS-KBO-SUG-00 | Debt Maturity Timestamp Validation 10

OS-KBO-SUG-01 | Code Refactoring 11

## Appendices

**Vulnerability Rating Scale** **13**

**Procedure** **14**


© 2026 Otter Audits LLC. All Rights Reserved.

# 01 — Executive Summary

## Overview

Exponent Finance engaged OtterSec to assess the `borrow-order` feature. This assessment was conducted between September 1st and October 18th, 2025. For more information on our auditing methodology, refer to Appendix B.

## Key Findings

We produced 5 finding throughout this audit engagement.

In particular, we identified a vulnerability concerning the possibility of front-running a borrow order update to overwrite the post-fill remaining debt amount, allowing an attacker to reopen borrowing capacity and force the borrower to take on more debt than they ever approved (OS-KBO-ADV-00). Additionally, an uninitialized first borrow timestamp will result in obligations appearing overdue, allowing premature liquidation of healthy positions (OS-KBO-ADV-02).

We also advised restricting the debt maturity timestamp to prevent malicious actors from backdating it and inflating liquidation bonuses (OS-KBO-SUG-00). Furthermore, we suggested refactoring the code to remove unnecessary code and prevent additional debt when an obligation has surpassed its debt term (OS-KBO-SUG-01).

© 2026 Otter Audits LLC. All Rights Reserved.

# 02 — Scope

The source code was delivered to us in a Git repository at https://github.com/Kamino-Finance/klend. This audit was performed on the following PR's:

1. PR#522
2. PR#524
3. PR#526
4. PR#537
5. PR#547
6. PR#538
7. PR#540
8. PR#541
9. PR#560
10. PR#572
11. PR#574
12. PR#580
13. PR#581
14. PR#608
15. PR#610
16. PR#619
17. PR#631
18. PR#641

**A brief description of the program is as follows:**

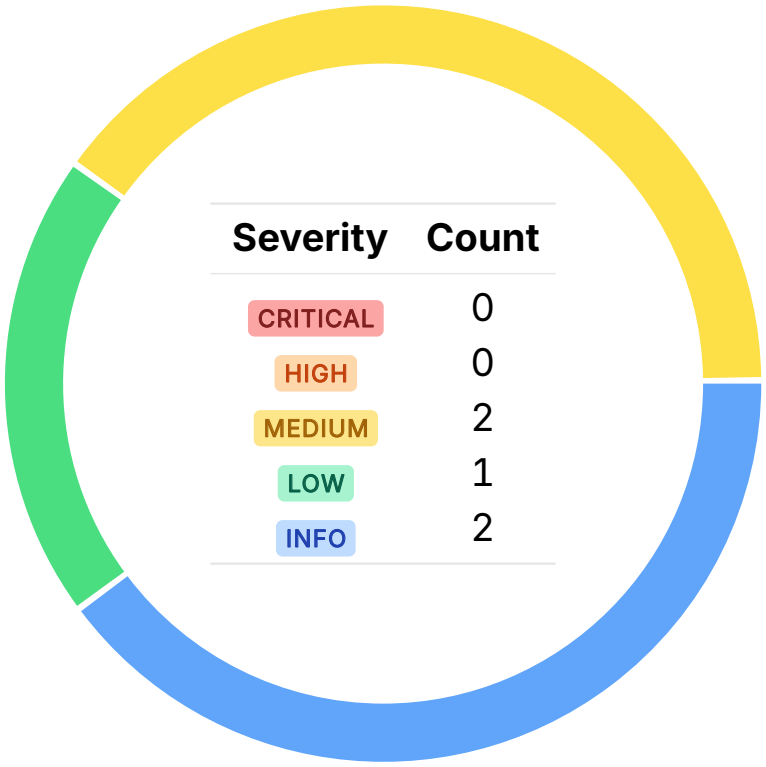
| Name | Description |
| --- | --- |
| borrow-order | Introduces the Borrow Order feature, a limit-order-style mechanism for borrowing on Kamino Lend, that allows borrowers to express their intent, desired amount, maximum acceptable rate, and minimum loan term as a persistent on-chain order. Any third-party executor can permissionlessly fill the order against a matching reserve, enabling passive, intent-based borrowing with rate and term guarantees. |

© 2026 Otter Audits LLC. All Rights Reserved.

# 03 — Findings

Overall, we reported 5 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



| Severity | Count |
| --- | --- |
| CRITICAL | 0 |
| HIGH | 0 |
| MEDIUM | 2 |
| LOW | 1 |
| INFO | 2 |

© 2026 Otter Audits LLC. All Rights Reserved.

# 04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

| ID | Severity | Status | Description |
|---|---|---|---|
| OS-KBO-ADV-00 | MEDIUM | RESOLVED | It is possible to front-run a borrow order update to overwrite the post-fill `remaining_debt_amount`, allowing an attacker to reopen borrowing capacity and force the borrower to take on more debt than they ever approved. |
| OS-KBO-ADV-01 | MEDIUM | RESOLVED | Uninitialized `first_borrowed_at_timestamp` will result in obligations appearing overdue, allowing premature liquidation of healthy positions. |
| OS-KBO-ADV-02 | LOW | RESOLVED | A reserve owner can shorten or introduce a `debt_maturity_timestamp` on a reserve with active borrows, making existing borrows liquidatable earlier than expected. |

© 2026 Otter Audits LLC. All Rights Reserved.

## Front Running Borrow Order MEDIUM OS-KBO-ADV-00

### Description

The protocol allows a borrower to update a `borrow_order` to reduce `remaining_debt_amount`, but the update logic overwrites on-chain state without accounting for intervening fills. Because `fill_borrow_order` is permissionless, an attacker may front-run the update and fill the order before the update executes. When the update transaction is later processed, it resets `remaining_debt_amount` to a higher value than the post-fill remainder, resulting in the borrower incurring more debt than originally agreed.

### Proof of Concept

The scenario below illustrates the front-running issue:

1. The user submits a transaction (`tx1`) that creates a borrow order with `remaining_debt_amount` = 2100.
2. The user submits another transaction (`tx2`) that reduces `remaining_debt_amount` to 1000.
3. An attacker front-runs `tx2` with `tx3`, which partially fills the order for 2000 USDC via `fill_borrow_order`(2000). As a result, `tx3` executes and 2000 USDC is borrowed, leaving `remaining_debt_amount` of 100 USDC.
4. Subsequently, `tx2` executes and overwrites `remaining_debt_amount` back to 1000, creating a regression where the remaining debt increases instead of decreasing as intended.
5. The attacker then submits `tx4` to fill the new remainder.
6. Consequently, 3000 USDC is borrowed in total, exceeding the intended maximum of 2100 or 1000, which is incorrect even if the user was merely delayed in updating `remaining_debt_amount`.

### Remediation

Introduce a `expected_remaining_debt_amount` parameter to the update instruction and abort if it does not match the on-chain `remaining_debt_amount` at execution.

### Patch

Resolved in 42ea028.

© 2026 Otter Audits LLC. All Rights Reserved.

## Premature Liquidation on Uninitialized Timestamp MEDIUM OS-KBO-ADV-01

### Description

`check_borrow_reserve_debt_term_reached` assumes that `liquidity.first_borrowed_at_timestamp` is always initialized for active borrows. If this value is 0 (for existing obligations with an uninitialized `liquidity.first_borrowed_at_timestamp`, the computed debt term end falls near the Unix epoch. Because the current timestamp is far greater than this value, the subtraction operation produces a large `secs_since_debt_term_end`, incorrectly signaling that the debt term has been exceeded. As a result, healthy positions become eligible for liquidation prematurely.

>_ *programs/kamino-lending/src/state/liquidation_operations.rs* RUST

```rust
fn check_borrow_reserve_debt_term_reached(
    &LiquidationCheckInputs {
        lending_market,
        collateral_reserve,
        liquidity,
        debt_reserve,
        obligation,
        timestamp,
        ..
    }: &LiquidationCheckInputs,
) -> Option<LiquidationParams> {
    let debt_term_seconds = debt_reserve.config.debt_term_seconds;
    if debt_term_seconds == 0 {
        return None; // default case: indefinite debt term
    }
    let debt_term_end_timestamp = liquidity.first_borrowed_at_timestamp + debt_term_seconds;
    let Some(secs_since_debt_term_end) = timestamp.checked_sub(debt_term_end_timestamp) else {
        return None; // debt maturity not reached yet
    };
    [...]
}
```

### Remediation

Add a check to verify whether `liquidity.first_borrowed_at_timestamp` is zero in `check_borrow_reserve_debt_term_reached` to ensure backward compatibility.

### Patch

Resolved in PR#572.

© 2026 Otter Audits LLC. All Rights Reserved.

## UpdateDebtMaturityTimestamp can Allow Immediate Liquidations LOW OS-KBO-ADV-02

### Description

A reserve owner can update `debt_maturity_timestamp` to the current timestamp (or barely into the future) on a reserve with active borrows, instantly making all existing borrowers liquidatable via the mature-reserve-debt liquidation path. The validation, `check_not_set_in_the_past`, prevents setting it to a past value but accepts "now", so the owner can add a maturity even if there wasn't one before, or shorten a far-future maturity to the present, causing surprise liquidations for healthy positions.

### Remediation

`UpdateDebtMaturityTimestamp` should check that the maturity timestamp is not shortened (or newly introduced) on a reserve with active borrows, mirroring the existing `check_debt_term_not_shortened_on_reserve_having_borrows` validation used by `UpdateDebtTermSeconds`.

### Patch

Team acknowledged the issue.

© 2026 Otter Audits LLC. All Rights Reserved.

# 05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

| ID | Description |
| --- | --- |
| OS-KBO-SUG-00 | `debt_maturity_timestamp` may be set arbitrarily, allowing malicious actors to backdate it and inflate liquidation bonuses. |
| OS-KBO-SUG-01 | Recommendations for removing unnecessary code, and to prevent additional debt when an obligation has surpassed its debt term. |

© 2026 Otter Audits LLC. All Rights Reserved.

## Debt Maturity Timestamp Validation OS-KBO-SUG-00

### Description

The `debt_maturity_timestamp` field in `lending_operations::update_reserve_config` currently lacks validation, allowing it to be set to any arbitrary value. This timestamp is utilized to calculate the `liquidation_bonus_rate`, which increases based on the number of days elapsed since the debt matured. Without validation, a malicious actor may backdate the timestamp, artificially inflating the bonus rate and extracting more collateral during liquidations.

>_ *programs/kamino-lending/src/lending_market/lending_operations.rs* RUST

```rust
pub fn update_reserve_config(
    reserve: &mut Reserve,
    mode: UpdateConfigMode,
    value: &[u8],
) -> Result<()> {
    // TODO: Enforce value type on mode
    match mode {
        [...]
        UpdateConfigMode::UpdateDebtMaturityTimestamp => {
            config_items::for_named_field!(&mut reserve.config.debt_maturity_timestamp)
                .set(value)?;
        }
        [...]
    }
    reserve.last_update.mark_stale();
    Ok(())
}
```

### Remediation

Restrict `debt_maturity_timestamp` to be either zero or no earlier than the current timestamp when set.

### Patch

Resolved in PR#572.

© 2026 Otter Audits LLC. All Rights Reserved.

## Code Refactoring OS-KBO-SUG-01

### Description

1. The `fill_borrow_order_checks` in `handler_fill_borrow_order::process_impl` is redundant because `borrow_obligation_liquidity_process_impl` already performs the same borrow-level validations. Remove the call to `fill_borrow_order_checks`.

*>_ programs/kamino-lending/src/handlers/handler_fill_borrow_order.rs* RUST

```
fn process_impl<'info>(ctx: &Context<'_, '_, '_, 'info, FillBorrowOrder<'info>>) ->
    ↪  Result<()> {
    lending_checks::fill_borrow_order_checks(&ctx.accounts.borrow_accounts)?;

    let accounts = &ctx.accounts.borrow_accounts;
    let remaining_accounts = ctx.remaining_accounts;

    // Borrow Obligation immutably, for a moment, just to find out the amount we want to
        ↪  borrow:
    let order_remaining_amount = accounts
        .obligation
        .load()?
        .borrow_order
        .remaining_debt_amount;

    // Fully re-use the logic of `handler_borrow_obligation_liquidity`:
    let fill_amount = borrow_obligation_liquidity_process_impl(
        &BorrowObligationLiquidity::from(accounts.clone()),
        remaining_accounts,
        BorrowSize::AtMost(order_remaining_amount),
    )?;
    [...]
}
```

2. The borrow logic in `lending_operations::borrow_obligation_liquidity` only checks whether the reserve has reached its debt maturity, but ignores whether the obligation itself has surpassed its debt term, enabling users to add new debt to an obligation whose existing borrow is already matured and liquidatable. Validate that an obligation has not exceeded its debt term before permitting additional liquidity to be borrowed.

*>_ programs/kamino-lending/src/lending_market/lending_operations.rs* RUST

```
#[allow(clippy::too_many_arguments)]
pub fn borrow_obligation_liquidity<'info, T>([...]) -> Result<CalculateBorrowResult>
where
    T: AnyAccountLoader<'info, Reserve>,
{
```

© 2026 Otter Audits LLC. All Rights Reserved.

```
    [...]
    // Technically, borrowing from a reserve that reached its debt maturity would be
        ↪ harmless to the
    // protocol (i.e. the debt would immediately be liquidatable), but it makes no sense
        ↪ from the
    // business PoV:
    if let Some(secs_since_debt_maturity_reached) = borrow_reserve
        .config
        .get_secs_since_debt_maturity_reached(current_timestamp)
    {
        msg!(
            "Cannot borrow from a reserve that reached its debt maturity {} seconds ago",
            secs_since_debt_maturity_reached
        );
        return err!(LendingError::ReserveDebtMaturityReached);
    }
    [...]
}
```

### Remediation

Incorporate the above refactors.

### Patch

1. Issue #1 resolved in PR#631.
2. Issue #2 resolved in PR#572.

© 2026 Otter Audits LLC. All Rights Reserved.

# A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

**CRITICAL**

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

**HIGH**

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

**MEDIUM**

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

**LOW**

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

**INFO**

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

© 2026 Otter Audits LLC. All Rights Reserved.

# B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.

© 2026 Otter Audits LLC. All Rights Reserved.