Security Assessment Report

Kamino Lending Program

February 6, 2025

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the Kamino Lending Program smart contracts.

The artifact of the audit was the source code of the following programs, excluding tests, in https://github.com/Kamino-Finance/klend/tree/c02bcf7.

The initial audit focused on the following versions and revealed 5 issues or questions.

program	type	commit
kamino_lending	Solana	`c02bcf7dfe932af27d429df4111b3a3ca05a0dd3`

This report provides a detailed description of the findings and their respective resolutions.

Table of Contents

Result Overview

Issue	Impact	Status
KAMINO_LENDING		
[L-01] Inaccurate margin call period calculation	Low	Resolved
[L-02] Potential TWAP expiry oversight	Low	Acknowledged
[I-01] Update pyth oracle to V2	Info	Resolved
[I-02] Inconsistent implementation of "collateral_exchange_rate_ceil"	Info	Resolved
[I-03] Potential improvements to "update_reserve_config" validation	Info	Resolved

Findings in Detail

KAMINO_LENDING

[L-01] Inaccurate margin call period calculation

When specific market conditions trigger a deleveraging event, the current design allows users to respond within a defined margin call period. Auto-deleverage can only occur after this period ends.

However, the current implementation records the time of the deleveraging event as the Solana slot number instead of a Unix timestamp. The margin call period is then determined by calculating the slot number difference and estimating the time elapsed based on an empirical value of 2 slots per second.

Given that the margin call period is typically long (e.g., 72 hours), relying on an empirical estimate introduces non-negligible inaccuracies, potentially causing the actual timing of auto-deleverage to deviate from user expectations.

```
/* programs/klend/src/state/liquidation_operations.rs */
488 | fn has_margin_call_period_expired(
489 |     reserve: &Reserve,
490 |     slots_since_deleveraging_started: u64,
491 | ) -> bool {
492 |     let secs_since_deleveraging_started = slots::to_secs(slots_since_deleveraging_started);
493 |     let deleveraging_margin_call_period_secs = reserve.config.deleveraging_margin_call_period_secs;
494 |     if secs_since_deleveraging_started < deleveraging_margin_call_period_secs {
495 |         xmsg!("Reserve is eligible for auto-deleveraging, but margin call period not expired
↪  ({secs_since_deleveraging_started}/{deleveraging_margin_call_period_secs} seconds)");
496 |         false
497 |     } else {
498 |         true
499 |     }
500 | }
```

Resolution

Fixed by commit "f3b2ce1".

KAMINO_LENDING

[L-02] Potential TWAP expiry oversight

The "TokenInfo" structure contains two fields, "max_age_price_seconds" and "max_age_twap_seconds", which define the validity periods for the spot price and the time-weighted average price, respectively.

However, in the instructions like "refresh_reserve", only "max_age_price_seconds" is considered when determining the need to refresh the price or invalidate the current recorded price. This is done via the "is_price_refresh_needed" and "is_saved_price_age_valid" functions, without accounting for "max_age_twap_seconds".

If "max_age_twap_seconds" is configured to be shorter than "max_age_price_seconds", this oversight may result in scenarios where the TWAP has expired but is still being used.

```
/* programs/klend/src/lending_market/lending_operations.rs */
077 | pub fn is_saved_price_age_valid(reserve: &Reserve, current_ts: clock::UnixTimestamp) -> bool {
078 |     let current_ts: u64 = current_ts.try_into().expect("Negative timestamp");
079 |     let price_last_updated_ts = reserve.liquidity.market_price_last_updated_ts;
080 |     let price_max_age = reserve.config.token_info.max_age_price_seconds;
081 |
082 |     current_ts.saturating_sub(price_last_updated_ts) < price_max_age
083 | }
084 |
085 | pub fn is_price_refresh_needed(
086 |     reserve: &Reserve,
087 |     market: &LendingMarket,
088 |     current_ts: clock::UnixTimestamp,
089 | ) -> bool {
090 |     let current_ts = current_ts as u64;
091 |     let price_last_updated_ts = reserve.liquidity.market_price_last_updated_ts;
092 |     let price_max_age = reserve.config.token_info.max_age_price_seconds;
093 |     let price_refresh_trigger_to_max_age_pct: u64 =
094 |         market.price_refresh_trigger_to_max_age_pct.into();
095 |     let price_refresh_trigger_to_max_age_secs =
096 |         price_max_age * price_refresh_trigger_to_max_age_pct / 100;
097 |
098 |     current_ts.saturating_sub(price_last_updated_ts) >= price_refresh_trigger_to_max_age_secs
099 | }
```

Resolution

The team clarified that this is the expected behavior. TWAP is used as a reference to ensure price validity. Once the price is validated, it is considered valid up to its maximum age. It is intentional that the maximum age of the TWAP itself is neither directly checked nor utilized.

KAMINO_LENDING

[I-01] Update pyth oracle to V2

The Pyth Pull Oracle has been released, with the previous model scheduled for deprecation.

The Pyth SDK has transitioned from "`pyth_sdk_solana`" to "`pyth_solana_receiver_sdk`".

Consider upgrading Pyth oracle to V2. For detailed migration and update instructions, please refer to the Pyth V2 Migration Guide on Solana.

Resolution

Fixed by commits "`c183a70`".

KAMINO_LENDING

[I-02] Inconsistent implementation of "collateral_exchange_rate_ceil"

Based on its name, "`collateral_exchange_rate_ceil`" is supposed to return the ceiling value of "`collateral_exchange_rate`".

However, its current implementation is identical to "`collateral_exchange_rate`":

```
/* programs/klend/src/state/reserve.rs */
196 | pub fn collateral_exchange_rate(&self) -> LendingResult<CollateralExchangeRate> {
197 |     let total_liquidity = self.liquidity.total_supply()?;
198 |     self.collateral.exchange_rate(total_liquidity)
199 | }

201 | pub fn collateral_exchange_rate_ceil(&self) -> LendingResult<CollateralExchangeRate> {
202 |     let total_liquidity = self.liquidity.total_supply()?;
203 |     self.collateral.exchange_rate(total_liquidity)
204 | }
```

Since "`collateral_exchange_rate_ceil`" is not utilized anywhere in the current codebase, it does not introduce inconsistent behaviors.

Resolution

Fixed by commit "`f3b2ce1`".

KAMINO_LENDING

[I-03] Potential improvements to "update_reserve_config" validation

The "`update_reserve_config`" instruction allows the lending market owner to modify fields within the "`Reserve`". While this instruction can only be called by the lending market owner, who is responsible for ensuring the correctness of the modifications, the program also implements checks to ensure the validity of the updated fields. However, these checks can be further improved in the following ways:

1. **Bypassing ReserveStatus validation in UpdateEntireReserveConfig.**

While "`UpdateReserveStatus`" validates the provided value as a legitimate "`ReserveStatus`" using "`ReserveStatus::try_from`", the "`UpdateEntireReserveConfig`" instruction directly deserializes and assigns user-provided data without ensuring the validity of the "`ReserveStatus`".

It is recommended to add the corresponding validation to "`validate_reserve_config`".

2. **Inconsistent validation of borrow fee range.**

Currently, a borrow fee of 100% is disallowed in "`validate_reserve_config`". However, the error message states that the valid range for borrow fees is the closed interval "`[0, 100%]`".

This discrepancy between the edge case handling and the documented range should be resolved to ensure consistency.

3. **Adding more validations.**

Some fields like "`host_fixed_interest_rate_bps`", "`utilization_limit_block_borrowing_above`", and "`elevation_group.ltv_pct`" currently lack validation checks.

Consider adding simple validation rules for these fields would enhance the completeness and robustness of the configuration checks.

Resolution

Fixed by commits "c183a70".

Appendix: Methodology and Scope of Work

Assisted by the Sec3 Scanner developed in-house, the manual audit particularly focused on the following work items:

- Check common security issues.
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of scope of this work

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a Sec3 (the "Company") and the Client. This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

This Report is considered null and void if the Report (or any portion thereof) is altered in any manner.

ABOUT

The Sec3 audit team comprises a group of computer science professors, researchers, and industry veterans with extensive experience in smart contract security, program analysis, testing, and formal verification. We are also building automated security tools that incorporate static analysis, penetration testing, and formal verification.

At Sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.