Security Assessment Report

Kamino Lending Vault

February 6, 2025

Summary

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

The artifact of the audit was the source code of the following programs, excluding tests, in a private repository.

The initial audit focused on the following versions and revealed 3 issues or questions.

program	type	commit
Kamino Lending Vault	Solana	8273031ea75e3972cce5314cefe3e88f962927c5

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

Table of Contents

Result Overview

Issue	Impact	Status
KAMINO LENDING VAULT		
[I-01] Inaccuracy in management fee calculation	Info	Acknowledged
[Q-01] Discrepancy in token allocation and usable token calculation	Question	Acknowledged
[Q-02] Performance fee calculation in loss and recovery scenarios	Question	Acknowledged

Findings in Detail

KAMINO LENDING VAULT

[I-01] Inaccuracy in management fee calculation

During the calculation of management fees, the current implementation computes the management fee by multiplying the annual management fee rate by the number of seconds elapsed since the last fee collection, dividing this product by the number of seconds in a year, and then multiplying the result by the Assets Under Management (AUM).

To ensure greater precision, the constant "SECONDS_PER_YEAR", representing the number of seconds in a year, accounts for leap years and is stored as a floating-point number. However, the current implementation rounds this value up to the nearest integer when using it. Although the rounded value is very close to the original, its use in the denominator causes the calculated management fee to be slightly lower than expected.

```
/* programs/kamino-vault/src/operations/vault_operations.rs */
502 | let mgmt_fee_yearly = Fraction::from_bps(vault.management_fee_bps);
503 | let mgmt_fee = mgmt_fee_yearly * u128::from(seconds_passed)
504 |     / SECONDS_PER_YEAR.ceil().to_u128().unwrap();
505 | let mgmt_charge = Fraction::from(prev_aum).mul(mgmt_fee);

/* programs/kamino-lending/src/utils/consts.rs */
021 | pub const SECONDS_PER_YEAR: f64 = 365.242_199 * 24.0 * 60.0 * 60.0;
```

Resolution

The team clarified that this issue has minimal impact on the management fee amounts and the rounding favors users over the vault admin, which is deemed acceptable.

KAMINO LENDING VAULT

[Q-01] Discrepancy in token allocation and usable token calculation

In the invest instruction, the token allocation for each reserve is calculated based on predefined weights. The total token amount used for this calculation is derived from the current AUM, which is defined as "`AUM = token_available + invested.total - pending_fees`"

However, during the subsequent operations, the actual usable token amount is based solely on "`token_available`" and does not account for the deduction of "`pending_fees`".

```
/* programs/kamino-vault/src/state.rs */
330 | pub fn refresh_target_allocations(&mut self, invested: &Invested) -> Result<()> {
331 |     let total_tokens = self.compute_aum(invested)?;

/* programs/kamino-vault/src/state.rs */
171 | pub fn compute_aum(&self, invested: &Invested) -> Result<Fraction> {
172 |     // if the vault only has pending fees, it should not be possible to withdraw
173 |     let pending_fees = self.get_pending_fees();
174 |
175 |     if Fraction::from(self.token_available) + invested.total < pending_fees
176 |         && pending_fees > Fraction::ZERO
177 |     {
178 |         return err!(KaminoVaultError::AUMBelowPendingFees);
179 |     }
180 |
181 |     Ok(Fraction::from(self.token_available) + invested.total - pending_fees)
182 | }

/* programs/kamino-vault/src/operations/vault_operations.rs */
664 | pub fn available_to_invest(vault: &VaultState) -> u64 {
665 |     vault.token_available
666 | }
```

This discrepancy does not introduce any immediate issues, as the "`withdraw_pending_fees`" instruction allows for the redemption of already invested tokens to facilitate the withdrawal process. In fact, this behavior could potentially lead to increased returns in some scenarios. While this behavior does not appear to cause functional problems, we'd like to confirm if this is the intended behavior.

Resolution

The team confirmed this is a known behavior. It is considered beneficial for all parties, including the vault manager and kLend, as it allows the fees to be invested.

KAMINO LENDING VAULT

[Q-02] Performance fee calculation in loss and recovery scenarios

As per the design, no performance fee should be charged to users in the event of a loss. In the current implementation, when a loss occurs (i.e., "`new_aum`" < "`prev_aum`"), the performance fee is indeed set to zero. However, the "`prev_aum`" is still updated to "`new_aum - new_fees`".

```
/* programs/kamino-vault/src/operations/vault_operations.rs */
515 | let earned_interest = new_aum.saturating_sub(prev_aum);
516 | let perf_charge = Fraction::from_bps(vault.performance_fee_bps) * earned_interest;
517 | msg!(
518 |     "perf_charge {} earned_interest {}",
519 |     perf_charge.to_display(),
520 |     earned_interest.to_display()
521 | );
522 |
523 | vault.set_cumulative_mgmt_fees(vault.get_cumulative_mgmt_fees().saturating_add(mgmt_charge));
524 | vault.set_cumulative_perf_fees(vault.get_cumulative_perf_fees().saturating_add(perf_charge));
525 | vault.set_cumulative_earned_interest(
526 |     vault
527 |         .get_cumulative_earned_interest()
528 |         .saturating_add(earned_interest),
529 | );
530 |
531 | let new_fees = (mgmt_charge + perf_charge).min(new_aum);
532 | let pending_fees = vault.get_pending_fees() + new_fees;
533 | vault.set_pending_fees(pending_fees);
534 | update_prev_aum(vault, new_aum - new_fees);
```

For example:

- Before the loss, the AUM is x.
- After the loss, the AUM decreases to y.
- At a later point, the AUM recovers to z ($z \geq x$).

During the recovery from y to z, this increase is treated as "interest," and a performance fee is charged on this amount. Furthermore, this portion of the recovery is recorded in the cumulative earned interest.

Although the update of AUM during this process is necessary for accurate tracking, the double calculation of the $x - y$ segment raises concerns about accuracy in fee assessment, we'd like to confirm if this is the intended behavior.

Resolution

The team explained that this scenario is only relevant in cases of socialized loss, which is considered highly unlikely. As a result, the decision was made not to address this issue at this time.

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.