Kamino Lend Integration

Security Assessment

October 16th, 2024 — Prepared by OtterSec

Ajay Shankar Kunapareddy d1r3wolf@osec.io

Akash Gurugunti sud0u53r.ak@osec.io

Robert Chen r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Exponent Finance engaged OtterSec to assess the `kamino-lend-standard` program. This assessment was conducted between August 24th and October 9th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 8 findings throughout this audit engagement.

In particular, we identified a high-risk vulnerability, where the reward index update is based only on staked `SY` tokens, while unstaked `SY` tokens should also accrue rewards for the treasury, resulting in inaccurate reward allocation (OS-KLI-ADV-00). Additionally, we highlighted the possibility of passing an incorrect reward index, resulting in execution failure (OS-KLI-ADV-02), and an inconsistency in the exchange rate calculations (OS-KLI-ADV-04).

Furthermore, while withdrawing the obligation collateral, the same token program is utilized for liquidity and collateral transactions, which may result in failures if the token program types differ (OS-KLI-ADV-03).

We also made recommendations to ensure adherence to coding best practices (OS-KLI-SUG-01) and suggested the removal of unutilized and redundant code within the system for increased readability (OS-KLI-SUG-02). We further advised incorporating additional checks within the codebase for improved robustness and security (OS-KLI-SUG-00).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/exponent-finance/exponent-core. This audit was performed against commit e9f5248.

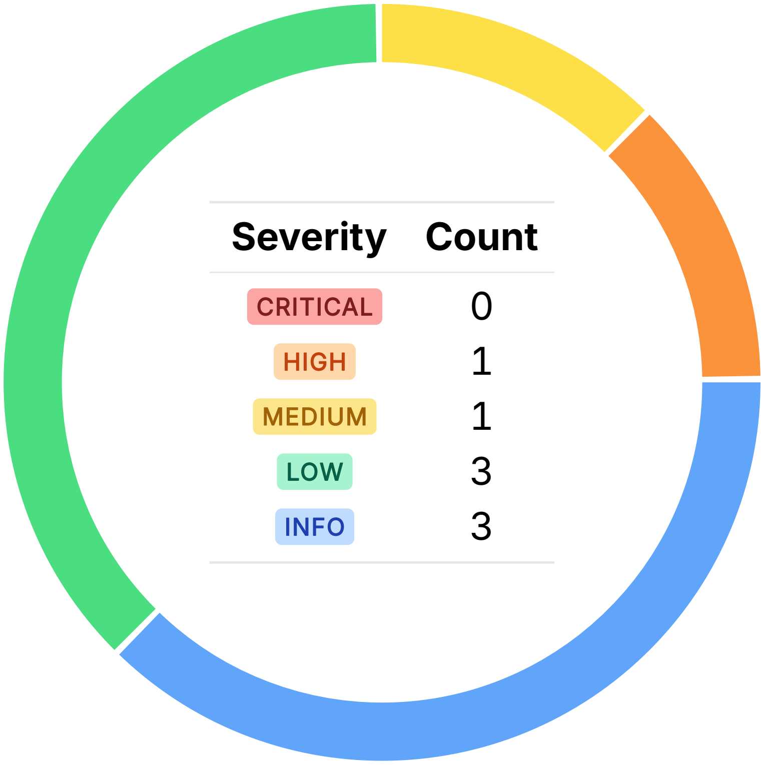
A brief description of the program is as follows:

Name	Description
kamino-lend-standard	It allows users to mint a receipt of a position in a DeFi protocol, which can then be utilized as SY in the Exponent core program.

03 — Findings

Overall, we reported 8 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	1
MEDIUM	1
LOW	3
INFO	3

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-KLI-ADV-00	HIGH	RESOLVED ✓	The reward index update is based only on staked `SY` tokens, while unstaked `SY` tokens should also accrue rewards for the treasury, resulting in inaccurate reward allocation.
OS-KLI-ADV-01	MEDIUM	RESOLVED ✓	The `obligation_farm` is not stored in `SyMeta`, enabling the transfer of ownership of a different `obligation_farm` to the Kamino lending authority, allowing an attacker to abuse the system during deposit and withdrawal operations.
OS-KLI-ADV-02	LOW	RESOLVED ✓	`process_reward_emissions` may pass an incorrect reward index to `harvest_reward` if some emissions have not been added to `meta.emissions`, potentially resulting in execution failure.
OS-KLI-ADV-03	LOW	RESOLVED ✓	`withdraw_obligation_collateral_ctx` utilizes the same token program for both liquidity and collateral transactions, which may result in failures if the token program types differ.

OS-KLI-ADV-04	LOW	RESOLVED ✓	There is an inconsistency in the exchange rate calculations in `SyEmissions::to_sy_state`, which utilizes market price data, and `collateral_exchange_rate` in Kamino Lending, which relies on total liquidity.

Faulty Distribution of Emissions HIGH OS-KLI-ADV-00

Description

The issue arises from a misalignment in how rewards (emissions) are allocated to staked `SY` tokens and the treasury's share for unstaked `SY` tokens. In the `Emission` structure, the `index` field is utilized to track how much reward (emissions) each staked `SY` token holder is entitled to. It is updated based on new emissions that are credited to the system. Currently, the index is calculated only based on the staked `SY` tokens, but it is also utilized to calculate the rewards for unstaked `SY` tokens (which belong to the treasury). This is problematic because the total emissions should be distributed based on the total supply of `SY` tokens, not just the staked portion.

>_ *kamino_lend_standard/src/state/meta.rs* RUST

```rust
pub fn increase_from_token_credit(&mut self, sy_supply: u64, token_amount: u64) {
    let index_delta = Number::from_ratio(token_amount.into(), sy_supply.into());

    self.index += index_delta;
    self.last_seen_total_accrued_emissions += token_amount as u128;
}
```

This may result in the incorrect allocation of rewards to both staked `SY` token holders and the treasury.

Remediation

Ensure the index is calculated as the reward amount for the entire `sy_mint` supply, and not just for the staked `SY` tokens.

Patch

Resolved in PR#570.

Failure to Store User State MEDIUM OS-KLI-ADV-01

Description

In the `init_sy` instruction in `kamino_lend_standard`, the `obligation_farm`, which represents the user's state in Kamino Farms for earning rewards and is utilized to verify the `user_state` in `deposit` and `withdraw` instructions, is not stored in the `SyMeta` structure. As a result, it allows for a potential exploit where an attacker may transfer ownership of an `obligation_farm` to the Kamino lending (`klend`) authority, thus enabling the attacker to utilize a different `user_state` to interact with the farm.

Without storing the `obligation_farm` in `SyMeta`, there is no direct reference to verify that the correct user state is used while interacting with the `deposit` and `withdraw` instructions, as they rely on correctly verifying the user's state to ensure that the correct user is interacting with the system and that they may claim or modify rewards or assets tied to their state.

Remediation

Store the `obligation_farm` in `SyMeta` to ensure that every interaction with the system verifies the correct farm state, such that only the legitimate owner may modify or interact with the farm state.

Patch

Resolved in PR#610.

Potential Reward Index Mismatch LOW

OS-KLI-ADV-02

Description

In `utils::process_reward_emissions`, there is a potential mismatch between the enumerated index utilized during reward harvesting and the actual index of the reward in the `meta.emissions` data structure. `process_reward_emissions` iterates over escrow accounts and performs reward harvesting utilizing the Kamino Farms program via cross-program invocation (CPI) call to `harvest_reward`. During each iteration, the function passes an index parameter (calculated from the loop's position) to `harvest_reward`, representing the specific reward being collected for the user.

>_ *kamino_lend_standard/src/utils.rs* RUST

```rust
pub fn process_reward_emissions<'info>(
    escrow_accounts: &Vec<Pubkey>,
    remaining_accounts: &[AccountInfo<'info>],
    farm_state: AccountInfo<'info>,
    farm_vaults_authority: AccountInfo<'info>,
    global_config: AccountInfo<'info>,
    owner: AccountInfo<'info>,
    meta: &Meta,
    scope_prices: AccountInfo<'info>,
    user_state: AccountInfo<'info>,
    kamino_farms_program: AccountInfo<'info>,
) -> Result<()> {
    for (index, escrow_account) in escrow_accounts.into_iter().enumerate() {
        let i = 5 * index;
        let target_token_account = remaining_accounts[i].key;
        [...]

        // using index should not be an issue as filtered out optional values can not be between
            ↪ 2 Some values
        harvest_reward(
            harvest_reward_cpi_context.with_signer(&[&meta.kamino_account_authority_seeds()]),
            index as u64,
        )?;
    }
    Ok(())
}
```

However, if certain rewards have not been added to `meta.emissions` for a user, the index passed to `harvest_reward` may not correspond to the actual reward position in the Kamino Farms program. This desynchronization between the index passed and the internal reward data on the Kamino side may result in an error during execution, failing the instruction as the function is trying to harvest a reward at an incorrect index.

Remediation

Ensure that the system is in a fully synchronized state before allowing any deposit or withdrawal operations. Halt all deposit or withdrawal activities until all the expected rewards are fully populated in `meta.emissions` , or pass the reward index specific to the collected rewards instead of using the enumerated index.

Patch

The development team acknowledged the issue and assured that they would take the necessary precautions while adding emissions.

Token Program Mismatch LOW OS-KLI-ADV-03

Description

In `kamino_lend_standarad::RedeemSy`, `withdraw_obligation_collateral_ctx` utilizes the `token_program_base` account for both the `liquidity_token_program` and the `collateral_token_program`. This may result in problems when the two token programs (liquidity and collateral) are different. Specifically, suppose one token program is the standard SPL Token program, and the other is the token-2022 program. In that case, it may result in failures during execution as utilizing the same `token_program_base` for both operations will result in a mismatch.

>_ *kamino_lend_standard/src/instructions/redeem_sy.rs* RUST

```rust
fn withdraw_obligation_collateral_ctx(
    &self,
) -> CpiContext<'_, '_, '_, 'info,
    ↪  WithdrawObligationCollateralAndRedeemReserveCollateral<'info>>
{
    let accs = WithdrawObligationCollateralAndRedeemReserveCollateral {
        lending_market: self.lending_market.to_account_info(),
        collateral_token_program: self.token_program_base.to_account_info(),
        obligation: self.kamino_obligation.to_account_info(),
        lending_market_authority: self.lending_market_authority.to_account_info(),
        owner: self.authority_klend_account.to_account_info(),
        liquidity_token_program: self.token_program_base.to_account_info(),
        [...]
    };
    CpiContext::new(self.kamino_lend_program.to_account_info(), accs)
}
```

Remediation

Distinguish between the token programs utilized for liquidity and collateral. Instead of utilizing a single `token_program_base`, the function should explicitly specify the appropriate token programs for each operation.

Patch

Resolved in PR#611.

Exchange Rate Calculation Discrepancy LOW OS-KLI-ADV-04

Description

There is an inconsistency in the calculation of `exchange_rate` in `SyEmissions::to_sy_state` and in `collateral_exchange_rate` in Kamino Lending. `SyEmissions::to_sy_state` calculates the exchange rate from the `market_price_sf` field of the liquidity structure within the reserve, whereas `collateral_exchange_rate` bases the exchange rate on the total supply of liquidity (the `total_supply` of liquidity tokens). As a result, the exchange rate in `SyEmissions` overlooks the role of total liquidity and collateral.

Remediation

Ensure consistency in calculating the `exchange_rate` by aligning both calculations.

Patch

Resolved in PR#607.

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-KLI-SUG-00	Additional safety checks may be incorporated within the codebase to make it more robust and secure.
OS-KLI-SUG-01	Suggestions regarding codebase inconsistencies and ensuring adherence to coding best practices.
OS-KLI-SUG-02	The codebase contains multiple cases of unutilized and redundant code that should be removed for better maintainability and clarity.

Missing Validation Logic

OS-KLI-SUG-00

Description

1. In the existing `AddEmission` instruction in `kamino_lending`, implement a check to ensure that the `emission_token_account` is associated with the correct mint. Additionally, validate the `emission_mint` against `reserve_farm_state.reward_infos[index].token.mint`.
2. In the `InitSy` instruction within `kamino_lending`, ensure the freeze and metadata authorities are explicitly configured in a manner similar to that used in `marginfi_standard`.
3. In `kamino_lend_standard::add_emission`, there is no check to verify if the owner of `emission_token_account` is equal to `meta.authority_klend_account`, resulting in the possibility of funds getting locked due to an incorrect token account input. Verify that the owner of `emission_token_account` is equal to `meta.authority_klend_account`.

   >_ *kamino_lend_standard/src/instructions/add_emission.rs* RUST

   ```
   #[derive(Accounts)]
   pub struct AddEmission<'info> {
       pub signer: Signer<'info>,
       [...]
       #[account(token::token_program = emission_token_program, token::mint = emission_mint)]
       pub emission_token_account: InterfaceAccount<'info, TokenAccount>,
       [...]
   }
   ```

4. Add seed checks for `token_base_account_authority` in the `mint_st` and `redeem_sy` instructions.

Remediation

Add the missing validations mentioned above.

Patch

1. Issue #1 was resolved in 4c66c56
2. Issue #2 was resolved in PR#614.
3. Issue #2 was resolved in PR#516.
4. Issue #4 was acknowledged.

Code Maturity

OS-KLI-SUG-01

Description

1. The comment on the `authority_klend_account` field in the `SyMeta` structure incorrectly states *"Authority over the Marginfi account"* instead of *"Authority over the Kamino account"*. Update the comment to reflect the correct account.
2. `init_obligation_farms_for_reserve` in `kamino_lend_cpi` utilizes `invoke_signed` even though the owner account does not need to be signer.

```rust
>_ kamino_lend_cpi/src/lib.rs                                                    RUST
pub fn init_obligation_farms_for_reserve<'info>(
    ctx: CpiContext<'_, '_, '_, 'info, InitObligationFarmsForReserve<'info>>,
    mode: u8,
) -> Result<()> {
    [...]
    invoke_signed(
        &instruction,
        &ctx.accounts.to_account_infos().as_slice(),
        ctx.signer_seeds,
    )?;

    Ok(())
}
```

3. In the `collect_treasury_emission` instruction, it would be more efficient to utilize `Emission::collect` to handle the increment of `total_claimed_emissions`.

Remediation

1. Implement the above-mentioned suggestions.
2. Utilize `invoke` instead of `invoke_signed` in `init_obligation_farms_for_reserve`.
3. Utilize `Emission::collect` to handle the increment of `total_claimed_emissions`.

Patch

1. Issue #1 was resolved in PR#517.
2. Issue #2 was resolved in PR#515.
3. Issue #2 was resolved in PR#615.

Code Redundancy OS-KLI-SUG-02

Description

1. `BASE_SEED_USER_METADATA` and `USER_METADATA_SIZE` constants in `kamino_lend_cpi` are not required.
2. The `kamino_reserve` and `emission_token_program` accounts in the `kamino_lend_standard::add_emission` instruction seem unnecessary and may be removed.
3. `deposit_obligation_collateral`, `refresh_reserve`, `refresh_obligation`, `refresh_obligation_farms_for_reserve`, and `Obligation::find_collateral_in_deposits` are unutilized and may be removed.

Remediation

Remove the redundant and unutilized code instances highlighted above.

Patch

1. Issue #1 was resolved in PR#518.
2. Issue #2 and #3 were acknowledged.

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.

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.