OFFSIDE LABS

Exponent Kamino Standard

Smart Contract Security Assessment

October 2024

Prepared for:

Exponent Finance

Prepared by:

Offside Labs

Ripples Wen

Siji Feng

Contents

1 About Offside Labs

Offside Labs is a leading security research team, composed of top talented hackers from both academia and industry.

We possess a wide range of expertise in modern software systems, including, but not limited to, *browsers*, *operating systems*, *IoT devices*, and *hypervisors*. We are also at the forefront of innovative areas like *cryptocurrencies* and *blockchain technologies*. Among our notable accomplishments are remote jailbreaks of devices such as the **iPhone** and **PlayStation 4**, and addressing critical vulnerabilities in the **Tron Network**.

Our team actively engages with and contributes to the security community. Having won and also co-organized *DEFCON CTF*, the most famous CTF competition in the Web2 era, we also triumphed in the **Paradigm CTF 2023** within the Web3 space. In addition, our efforts in responsibly disclosing numerous vulnerabilities to leading tech companies, such as *Apple*, *Google*, and *Microsoft*, have protected digital assets valued at over **$300 million**.

In the transition towards Web3, Offside Labs has achieved remarkable success. We have earned over **$9 million** in bug bounties, and **three** of our innovative techniques were recognized among the **top 10 blockchain hacking techniques of 2022** by the Web3 security community.

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs

2 Executive Summary

Introduction

Offside Labs completed a security audit of *Exponent* smart contracts, starting on *October 1st, 2024*, and concluding on *October 4th, 2024*.

Project Overview

The Kamino Lend Standard program enables users to mint receipt tokens representing positions in Kamino Lend, which can be used as SY in the Exponent core program. Key functionalities include:

1. mint_sy and redeem_sy: Manage deposits into escrow and handle minting or redeeming of receipt tokens for each Kamino reserve.
2. Emissions Management: Accrues emissions for staked receipt tokens, facilitating easier tracking for the core program. Non-staked emissions are directed to the protocol's treasury.
3. Farms Integration: Unlike the Marginfi protocol, Kamino requires the Kamino Farms program to collect emissions from incentivized reserves. Kamino Farms can handle up to 10 rewards, and the program iterates through shared rewards with its emission tracker.

Audit Scope

The assessment scope contains mainly the smart contracts of the *Kamino Lend Standard* program for the *Exponent* project.

The audit is based on the following specific branches and commit hashes of the codebase repositories:

- *Kamino Lend Standard*
  - Codebase: https://github.com/exponent-finance/exponent-core
  - Commit Hash: 69d5571dc2d4def34e24033021bbcbf5234ab631

We listed the files we have audited below:

- *Kamino Lend Standard*
  - solana/programs/kamino_lend_standard/src
  - solana/libraries/kamino_farms_cpi
  - solana/libraries/kamino_fraction
  - solana/libraries/kamino_lend_cpi

Findings

The security audit revealed:

- 0 critical issue

- 1 high issues
- 0 medium issue
- 1 low issues
- 0 informational issue

Further details, including the nature of these issues and recommendations for their remediation, are detailed in the subsequent sections of this report.

3 Summary of Findings

ID	Title	Severity	Status
01	Risk of Emissions Harvesting with Fake User State and Flash Loan	High	Fixed
02	Risk of Premature Obligation Closure	Low	Acknowledged

4 Key Findings and Recommendations

4.1 Risk of Emissions Harvesting with Fake User State and Flash Loan

Severity: High	Status: Fixed
Target: Smart Contract	Category: Data Validation

Description

In the `deposit` , `withdraw` , `get_position` , and `get_sy_state` IXs, the `refresh_emission` function is invoked to synchronize emissions from Kamino. Within the `refresh_emission` function, the `HarvestReward` method of Kamino Farms is utilized to withdraw emissions.

```rust
#[derive(Accounts)]
#[instruction(reward_index: u64)]
pub struct HarvestReward<'info> {
    #[account(mut)]
    pub owner: Signer<'info>,

    #[account(mut,
        has_one = owner,
        has_one = farm_state,
    )]
    pub user_state: AccountLoader<'info, UserState>,
```

programs/kfarms/src/handlers/handler_harvest_reward.rs#L92-L102

In `HarvestReward` , the `user_state` account is only checked for the owner and `farm_state` , meaning any `user_state` belonging to the owner can be passed. Furthermore, in Kamino Farms, the owner of a `user_state` can either be set to any account through `InitializeUser` or transferred to another account using `TransferOwnership` .

```rust
#[derive(Accounts)]
pub struct InitializeUser<'info> {
...
    pub owner: AccountInfo<'info>,
```

programs/kfarms/src/handlers/handler_initialize_user.rs#L63-L71

```rust
#[derive(Accounts)]
pub struct TransferOwnership<'info> {
    pub owner: Signer<'info>,

    #[account(mut,
        has_one = owner,
    )]
    pub user_state: AccountLoader<'info, UserState>,
}
```

programs/kfarms/src/handlers/handler_transfer_ownership.rs#L9-L33

As a result, an attacker could create a fake `user_state` and use it to refresh emissions, which would complete successfully but generate no emissions. The attacker could then use the fake `user_state` to deposit SY and the original `user_state` to withdraw SY, thereby collecting emissions without holding SY. By employing a flash loan, the attacker could execute the entire exploit within a single IX and capture the majority of the emissions.

Recommendation

- Store the `user_state` within the `SyMeta` structure and ensure consistency when accessing or modifying it across all relevant functions. This would prevent attackers from using inconsistent or fake `user_state` addresses.
- Alternatively, implement a PDA check for the `user_state` to ensure that only the correct and authorized address is used. This would guarantee that the `user_state` cannot be arbitrarily replaced with a fake address.

Mitigation Review Log

Exponent Team: This has been addressed and a user_state check has been added to all the relevant instructions.

4.2 Risk of Premature Obligation Closure

Severity: Low	**Status: Acknowledged**
Target: Smart Contract	**Category: Edge Case**

Description

The `obligation` is initialized within the `init_sy` without locking a small amount of base tokens immediately. This introduces a risk whereby an actor could execute both `mint_sy` and `redeem_sy` in quick succession, resulting in the closure of the `obligation` following the `redeem_sy` call.

Since `init_sy` not only initializes the `obligation` but also performs other critical operations, it cannot be invoked multiple times, thus preventing reinitialization. Additionally, The address of `obligation` is bound to its owner, which is a PDA, `kamino_account_authority` , in our case. Thus this `obligation` can only be initialized by the program itself, meaning manual calls to `InitObligation` are not permitted.

```rust
#[derive(Accounts)]
#[instruction(args: InitObligationArgs)]
pub struct InitObligation<'info> {
    pub obligation_owner: Signer<'info>,

    #[account(mut)]
    pub fee_payer: Signer<'info>,

    #[account(init,
        seeds = [&[args.tag], &[args.id],
    ↪   obligation_owner.key().as_ref(), lending_market.key().as_ref(),
    ↪   seed1_account.key().as_ref(), seed2_account.key().as_ref()],
        bump,
        payer = fee_payer,
        space = OBLIGATION_SIZE + 8,
    )]
    pub obligation: AccountLoader<'info, Obligation>,
```

programs/klend/src/handlers/handler_init_obligation.rs#L41-L55

Recommendation

- Add a validation check within the `redeem_sy` to ensure that a minimum number of tokens remain after redemption.
- Alternatively, implement a dedicated instruction for initializing the obligation separately to allow for safe reinitialization, if necessary.

Mitigation Review Log

Exponent Team: **Acknowledged**. There is always going to be a minimum amount deposited into the program. This can also be found in a comment in the Kamino standard program.

5 Disclaimer

This audit report is provided for informational purposes only and is not intended to be used as investment advice. While we strive to thoroughly review and analyze the smart contracts in question, we must clarify that our services do not encompass an exhaustive security examination. Our audit aims to identify potential security vulnerabilities to the best of our ability, but it does not serve as a guarantee that the smart contracts are completely free from security risks.

We expressly disclaim any liability for any losses or damages arising from the use of this report or from any security breaches that may occur in the future. We also recommend that our clients engage in multiple independent audits and establish a public bug bounty program as additional measures to bolster the security of their smart contracts.

It is important to note that the scope of our audit is limited to the areas outlined within our engagement and does not include every possible risk or vulnerability. Continuous security practices, including regular audits and monitoring, are essential for maintaining the security of smart contracts over time.

Please note: we are not liable for any security issues stemming from developer errors or misconfigurations at the time of contract deployment; we do not assume responsibility for any centralized governance risks within the project; we are not accountable for any impact on the project's security or availability due to significant damage to the underlying blockchain infrastructure.

By using this report, the client acknowledges the inherent limitations of the audit process and agrees that our firm shall not be held liable for any incidents that may occur subsequent to our engagement.

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

OFFSIDE LABS

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs