OFFSIDE LABS

Kamino Vault

Smart Contract Security Assessment

April 2025

Prepared for:

Kamino

Prepared by:

Offside Labs

Ronny Xing

Yao Li

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 four security audits of *Kamino Vault* smart contracts, with periods as follows:

1. August 5 - August 9, 2024;
2. February 19 - February 21, 2025;
3. March 20 - March 20, 2025;
4. April 12 - April 12, 2025;

Project Overview

Kamino Vault is a cross-market liquidity vault built on *Kamino Lending*. For any given collateral token, *Kamino Vault* can invest and auto-rebalance across multiple *Kamino Lending* markets to maximize capital efficiency and yield. Interest earned from collateral in each market accrues to the vault in real-time, allowing users to withdraw their earnings at any time.

Audit Scope

The assessment scope contains mainly the smart contracts of the kamino_vault program for the *Kamino Vault* project.

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

- *Kamino Vault*
  - Branch: *master*
  - Commit Hash:
    1. 0791b3398f7c4e8e96222ab710bda3fe9530e00f
    2. 586ec4ebc991f949b8be26183459c4511817ae38
    3. 2c9c01a6935d7911de10ed394d0daa507d0acf43
    4. ee552e93608a76bcc096633fad6710114b41d07b
  - Codebase Link
- *Kamino Lending*
  - Branch: *master*
  - Commit Hash: d23536ce7da780603a8881af0f6d31e487d85435
  - Codebase Link

We listed the files we have audited below:

- *Kamino Vault*
  - programs/kamino-vault/src/**/*.rs
- *Kamino Lending*
  - programs/kamino-lending/src/handlers/handler_refresh_reserves_batch.rs

Findings

The security audit revealed:

- 0 critical issue
- 0 high issue
- 4 medium issues
- 3 low issues
- 6 informational issues

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	Reserve is Mutually Exclusive Among Multiple Vaults	Medium	Fixed
02	Accounting Error in withdraw_pending_fees	Medium	Fixed
03	withdraw_pending_fees Returns Incorrect invested_liquidity_to_disinvest	Medium	Fixed
04	Incorrect AUM Update in give_up_pending_fees Instruction	Medium	Fixed
05	Incorrect Management Fee Calculation	Low	Fixed
06	withdraw_pending_fees Loses Truncation Errors	Low	Fixed
07	Macro kmsg May Lead to DoS	Low	Fixed
08	Incorrect Variable Name perf_fee_yearly	Informational	Fixed
09	Lack of Validation for the allocation_cap in the update_reserve_allocation Instruction	Informational	Fixed
10	Users Should Not Pay crank_fund_fee for Reserves Which are Disabled For Investment	Informational	Fixed
11	Rounding Error Loss in withdraw Can Be Optimized	Informational	Fixed
12	Socialized losses Might Cause Vaults to Malfunction	Informational	Fixed
13	Withdrawals May Cause Users Unexpected Precision Loss	Informational	Acknowledged

4 Key Findings and Recommendations

4.1 Reserve is Mutually Exclusive Among Multiple Vaults

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

The `update_reserve_allocation` instruction initializes `ctoken_vault` using the following constraints:

```
    #[account(init_if_needed,
        seeds = [CTOKEN_VAULT_SEED, reserve.key().as_ref()],
        bump,
        payer = admin_authority,
        token::mint = reserve_collateral_mint,
        token::authority = base_vault_authority,
        token::token_program = token_program
    )]
    pub ctoken_vault: Box<InterfaceAccount<'info, TokenAccount>>,
```

programs/kamino-vault/src/handlers/handler_update_reserve_allocation.rs#L60-L68

Therefore, for a reserve, the program can only initialize a single `ctoken_vault` .

The issue is that `base_vault_authority` corresponds one-to-one with `vault_state` . Therefore, for a reserve, there can only be one corresponding vault, which is the first to add this reserve.

Impact

Since the `init_vault` instruction is permissionless, this might lead to a race and front-run to register the reserve first.

Recommendation

Append the pubkey of `vault_state` in the seeds, or initialize the `ctoken_vault` authority as a unique PDA independent of `vault_state` .

Mitigation Review Log

Offside Labs: Fixed. The `vault_state.key()` is added as a part of the seeds for the `ctoken_vault` .

4.2 Accounting Error in withdraw_pending_fees

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

The `withdraw_pending_fees` method returns a result indicating that `WithdrawPendingFeesEffects.invested_liquidity_to_send_to_user` is the actual amount of tokens withdrawn and transferred directly from the vault. However, during the vault's accounting process, `withdraw_pending_fees` fails to properly manage `invested_liquidity_to_send_to_user` and `liquidity_rounding_error`.

```
    let actual_invested_liquidity_to_send_to_user =
        invested_liquidity_to_send_to_user - liquidity_rounding_error;
    let disinvested_amount_left_in_vault =
        invested_liquidity_to_disinvest -
 ↪  actual_invested_liquidity_to_send_to_user;

    // Accounting
    ...
    common::deposit_into_strategy(vault,
     ↪  disinvested_amount_left_in_vault)?;
```

programs/kamino-vault/src/operations/vault_operations.rs#L205-L205

The final `deposit_into_strategy` results in `disinvested_amount_left_in_vault = invested_liquidity_to_disinvest - invested_liquidity_to_send_to_user + liquidity_rounding_error` being retained in the vault as a precision loss within the accounting process.

If `liquidity_rounding_error` is greater than 0, it will be kept in `vault.token_available` due to the above accounting process, but it is already included in `invested_liquidity_to_send_to_user` and will be withdrawn by the fee admin.

Impact

This discrepancy causes the vault's internal accounting to show more tokens than are actually held in the token accounts, potentially leading to a temporary DOS due to underflow in the worst case.

Recommendation

Shoud return `actual_invested_liquidity_to_send_to_user` as the `WithdrawPendingFeesEffects.invested_liquidity_to_send_to_user`.

Mitigation Review Log

Offside Labs: Fixed

4.3 withdraw_pending_fees Returns Incorrect invested_liquidity_to_disinvest

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

The `withdraw_pending_fees` method returns a result indicating that `WithdrawPendingFeesEffects.invested_liquidity_to_disinvest` the actual amount of tokens will be redeemed from the reserve.

The issue is that, it uses `invested_liquidity_to_send_to_user` as the value of `invested_liquidity_to_disinvest`.

```
    Ok(WithdrawPendingFeesEffects {
        available_to_send_to_user,
        invested_to_disinvest_c_tokens,
        invested_liquidity_to_send_to_user,
        invested_liquidity_to_disinvest:
    ↪   invested_liquidity_to_send_to_user,
    })
```

programs/kamino-vault/src/operations/vault_operations.rs#L295-L300

This is incorrect. When there is truncation error, `invested_liquidity_to_disinvest` will be greater than `invested_liquidity_to_send_to_user`.

Impact

The post check `reserve_supply_liquidity_diff == i128::from(invested_liquidity_to_disinvest)` in `handler_withdraw_pending_fees` will fail, programs/kamino-vault/src/handlers/handler_withdraw_pending_fees.rs#L136-L143, preventing the admin from withdrawing fees unless truncation errors are absent.

Recommendation

Should use `invested_liquidity_to_disinvest` as the return instead of `invested_liquidity_to_send_to_user`.

Mitigation Review Log

Offside Labs: Fixed

4.4 Incorrect AUM Update in give_up_pending_fees Instruction

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `charge_fees` function, when the condition `available + invested < pending_fees` is met, the AUM becomes theoretically negative. Then it is set to zero instead of accurately reflecting the deficit:

```rust
    let new_aum = vault.compute_aum(invested).unwrap_or(Fraction::ZERO);
```

programs/kamino-vault/src/operations/vault_operations.rs#L493-L493

This issue is compounded in the `give_up_pending_fees` instruction, where the AUM is directly increased by `amount_to_give_up` without considering the theoretical state of negative AUM:

```rust
    // give up pending fee is aimed at unlocking a vault that suffered a
    ↪ loss.
    // To ensure the next charge fee don't prevent this, update
    ↪ last_fee_charge_timestamp to the current timestamp and increase
    ↪ the prev aum by the amount given up

    vault.last_fee_charge_timestamp = current_timestamp;
    let prev_aum = vault.get_prev_aum();
    common::update_prev_aum(vault, prev_aum + amount_to_give_up);
```

programs/kamino-vault/src/operations/vault_operations.rs#L369-L374

When the AUM is already theoretically negative, directly increasing it with `amount_to_give_up` results in an incorrect AUM update.

Proof of Concept

- If `available + invested = 0` , and `pending_fees = amount_to_give_up > 0` :
- The `prev_aum` will be calculated as `0` .
- Adding `amount_to_give_up` directly to `prev_aum` will result in an incorrect positive value for AUM.
- The expected outcome is that the AUM should remain `available + invested` , which should be zero in the above case instead of `amount_to_give_up` .

Impact

This incorrect AUM update can result in the vault failing to unlock successfully. This failure happens because the unexpected fees generated due to the incorrect AUM calculation could cause the vault to remain locked.

Recommendation

The new AUM should be `vault_state.token_available + invested.total - new_pending_fees`. If an overflow error is thrown, the default value will be 0.

Mitigation Review Log

Fixed in PR11.

4.5 Incorrect Management Fee Calculation

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Math Error

Description

```
let mgmt_charge = Fraction::from(aum).mul(mgmt_fee);
```

programs/kamino-vault/src/operations/vault_operations.rs#L447-L447

In the `charge_fees` function, the `mgmt_charge` is calculated using `aum`, `slot_passed` and `mgmt_fee_yearly`. However, during the time interval, the value of AUM is dynamic and usaually increases. The current formula uses the latest AUM value, which is not a precise approximation.

Impact

The `mgmt_charge` is always larger than expected.

Recommendation

If we disregard the fact that management fees should be deducted from AUM in real-time during calculations (the impact is extremely small and can be ignored), we can use the average AUM `aum = (prev_aum + current_aum) / 2` to caculate the `mgmt_charge` more accurately.

Mitigation Review Log

Offside Labs: Fixed. Computing the mgmt fee based on the lower edge of the AUM is good.

The calculation formulas we described in our recommendations hold true when AUM grows linearly and fee accumulation is not considered. Generally, this would still slightly amplify the fees. Although the error is already very small. So computing the mgmt fee based on the lower edge of the AUM is quite a good approach.

$$AUM(x) = \text{prev\_aum} + (\text{current\_aum} - \text{prev\_aum}) \cdot \frac{x}{t}$$

$$\begin{aligned} mgmt\_fee &= \int_0^t r \cdot AUM(x)\,,dx \\ &= r \cdot \left[\text{prev\_aum} \cdot x + (\text{current\_aum} - \text{prev\_aum}) \cdot \frac{x^2}{2t}\right]\Bigg|_0^t \\ &= r \cdot \left[\text{prev\_aum} \cdot t + (\text{current\_aum} - \text{prev\_aum}) \cdot \frac{t}{2}\right] \\ &= r \cdot t \cdot \frac{\text{prev\_aum} + \text{current\_aum}}{2} \end{aligned}$$

4.6 withdraw_pending_fees Loses Truncation Errors

Severity: Low	**Status: Fixed**
Target: Smart Contract	**Category: Precision Issue**

Description

`withdraw_pending_fees` updates the `pending_fees` after extraction using the following code:

```
    common::update_pending_fees(
        vault,
        total_fees
            - available_to_send_to_user_f
            - Fraction::from(invested_liquidity_to_send_to_user),
    );
```

programs/kamino-vault/src/operations/vault_operations.rs#L288-L293

The issue is that `available_to_send_to_user_f` is a decimal, yet the tokens actually extracted are only the integer part of that decimal.

Impact

Every time fees are withdrew, the truncated decimal portion of `available_to_send_to_user_f` is lost.

Recommendation

It should subtract `Fraction::from(available_to_send_to_user)` instead of `available_to_send_to_user_f`.

Mitigation Review Log

Offside Labs: Fixed

4.7 Macro kmsg May Lead to DoS

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

Macro `kmsg!` use `ArrForm` to allocate a fixed size of memory on stack. For fmt string length not greater than 50 bytes, it will allocate 150 bytes of memory for the whole formatted string. But this might not enough for cases with too many parameters or long parameters, e.g.

```
                crate::kmsg!(
                    "Reserve {}: {}/{} target {} of total {}",
                    allocation.reserve,
                    allocation.target_allocation_weight,
                    total_weight,
                    token_target_allocation.to_floor::<u64>(),
                    total_tokens.to_floor::<u64>()
                );
```

programs/kamino-vault/src/state.rs#L359-L366

The fmt string len is 39, but the max length of the formatted string can be about 155 bytes.

Impact

Lead to panic when the formatted string is overflow.

Recommendation

If this log is intended to be treated as an event, it's best to encode as an event structure to define its length. If it's solely for debugging, it's recommended to either remove these msg logs in the release version or truncate excessively long strings at the end to avoid any unnecessary panics.

Mitigation Review Log

Offside Labs: Fixed in PR-20

4.8 Informational and Undetermined Issues

Incorrect Variable Name perf_fee_yearly

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Logic Error

```rust
let perf_fee_yearly = Fraction::from_bps(vault.performance_fee_bps);
```

programs/kamino-vault/src/operations/vault_operations.rs#L457-L457

The variable for the performance fee ratio does not have an annualized concept, yet it is named `perf_fee_yearly` .

Lack of Validation for the allocation_cap in the update_reserve_allocation Instruction

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

The `allocation_cap` is assigned without validating its effectiveness. It's recommended to add a check, such as `if target_allocation_weight > 0` , `allocation_cap` should be greater than `min_invest_amount` config.

Users Should Not Pay crank_fund_fee for Reserves Which are Disabled For Investment

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

With each deposit, users are charged `vault.crank_fund_fee_per_reserve` for every reserve.

```rust
    let num_reserve: u64 = vault.vault_allocation_strategy.iter()
        .filter(|r| r.reserve != Pubkey::default())
        .count().try_into().unwrap();
    let crank_funds_to_deposit = num_reserve *
    ↪   vault.crank_fund_fee_per_reserve;
```

programs/kamino-vault/src/operations/vault_operations.rs#L52-L59

However, using only `r.reserve != Pubkey::default()` as a filter means users might still incur fees for reserves where investment is disabled. These reserves are those in the `vault_allocation_strategy` where `c_token_allocation` is greater than 0, yet either `target_allocation_weight` or `token_allocation_cap` are 0. It is recommended to exclude these reserves from `num_reserve`.

Offside Labs: Fixed by new `get_reserves_with_allocation_count` function with check of non-zero `token_allocation_cap` and `target_allocation_weight`.

Rounding Error Loss in withdraw Can Be Optimized

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Precision Issue

When a user withdraws, the fractional part of the liquidity amount corresponding to burnt shares won't be withdrawn. This part, left in the vault accounting, has already absorbed some of the rounding error loss from ctoken withdrawal, allowing for optimization in the rounding error calculation code.

The condition programs/kamino-vault/src/operations/vault_operations.rs#L161 can be changed to:

```rust
if invested_liquidity_to_disinvest_f.frac() >=
↪   invested_liquidity_to_send_to_user_f.frac()
```

The use of `>=` instead of `>` accounts for the precision and truncation in the fractional part of the `fraction::Fraction`.

Socialized losses Might Cause Vaults to Malfunction

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Logic Error

In extreme cases, if Kamino Lending calls the `socialize_loss` instruction to eliminate bad debt, it may result in a decrease in the ctoken exchange rate. This causes a decrease in AUM and holdings. However, since pending fees are recorded directly as a liquidity token amount, if holdings are insufficient to cover, or equal to the pending fees, the vault will stop functioning.

A typical example is the `get_shares_to_mint` method.

Potential underflow:

```
let tokens_total_holdings = holdings.total_sum - vault.get_pending_fees();
```

programs/kamino-vault/src/operations/vault_operations.rs#L552-L552

Possible failing check:

```
let shares_to_mint = if tokens_total_holdings == Fraction::ZERO {
    // TODO: what about fees?
    panic!("Shares issued must be zero if tokens_total_holdings are also
    ↪   0");
}
```

programs/kamino-vault/src/operations/vault_operations.rs#L554-L557

Kamino Team: PR-338 adds an ix to give up pending fees (possibly allow to save a vault affected by socialized losses)

Offside Labs: Fixed.

Note: Since fees must be charged(calling `charge_fees`) before they can be waived, this instruction still breaks when `vault.token_available + invested.total < pending_fees`.

Withdrawals May Cause Users Unexpected Precision Loss

Severity: Informational	Status: Acknowledged
Target: Smart Contract	Category: Precision

Due to the amount of extracted liquidity being constrained by the available liquidity and the invested liquidity in the selected reserve, the withdraw function will recalculate the number of shares to be burned based on the actual amount of liquidity that can be extracted (`theoretical_amount_to_send_to_user_f`).

```
    let shares_to_burn = common::calculate_shares_to_burn(
        theoretical_amount_to_send_to_user_f,
        total_shares_supply,
        current_vault_aum,
        number_of_shares,
    );
```

programs/kamino-vault/src/operations/vault_operations.rs#L197-L197

When the share rate is large, the liquidity represented by the shares burned could be greater than the amount of liquidity actually sent to the user.

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