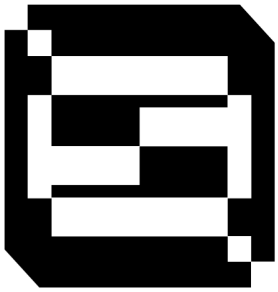

OFFSIDE Labs

Kamino Farms

Smart Contract Security Assessment

December 2023

Prepared for:

Hubble Protocol

Prepared by:

Offside Labs

Ronny Xing

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 *Kamino*'s *Farms* smart contracts, beginning on November 20, 2023, and concluding on December 8, 2023.

Farms Project Overview

The purpose of *Kamino*'s *Farms* is to provide a generic Farm/Staking Pool. Farms operate on a permissionless basis, with each farm having its own `global_admin`, which permits anyone to create their own farm.

Users can stake/unstake at any time and are able to harvest their accumulated rewards separately. Additionally, a minimum claimable duration can be set for each individual reward to enforce a minimum waiting time for users until it becomes claimable.

The staking logic is based on the current *Hubble* staking structure, which is also based on the *MasterChefV2* logic.

Audit Scope

The assessment scope primarily includes the smart contracts of the *Farms* project as of November 18, 2023.

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

- Farms
  - Branch: master
  - Commit Hash: 057356f71ee17c16c3b8142f5c71a30eb1db2a04
  - Codebase Link

We listed the files we have audited below:

- Farms
  - programs/farms/src/**/*.rs

Findings

The security audit revealed:

- 2 high issues
- 3 medium issues
- 2 low issues
- 2 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	convert_amount_to_stake Function Fails to Accurately Validate the Cold Start State	High	Fixed
02	deposit_to_farm_vault Instruction may Skirt Deposit Cap and Drain Constant-Type Rewards	High	Fixed
03	Oracle Price Factor Overflows Lead to DoS	Medium	Fixed
04	Admin Withdrawal may Result in Subsequent User Stake Deposits being Stuck in the Pending Stake	Medium	Fixed
05	A Portion of Rewards will be Permanently Lost Due to the Precision in user_refresh_reward	Medium	Fixed
06	Updates to scope_prices and scope_oracle_price_id Should be Bound	Low	Acknowledged
07	UpdateRewardScheduleCurvePoint is Not Fully Validated	Low	Fixed
08	withdraw_from_farm_vault Instruction Carries Centralization Risk	Informational	Fixed
09	advance_clock_timestamp Function in Test Util Cannot Guarantee Synchronization for Clock Sysvar	Informational	Fixed

4 Key Findings and Recommendations

4.1 convert_amount_to_stake Function Fails to Accurately Validate the Cold Start State

Severity: High	Status: Fixed
Target: Smart Contract	Category: Precision

Description

The `convert_amount_to_stake` method uses the condition `total_amount == 0` to determine if the system is in the initial state during cold start.

```
    if total_amount == 0 {
            ...
            Decimal::from(amount)
    } else {
        total_stake * amount / total_amount
    }
```

During the pool's initial phase, shares are allocated at a fundamental 1:1 ratio. Beyond this phase, share allocation adheres to the prevailing `total_stake` to `total_amount` ratio within the active pool.

However, the `total_amount == 0` condition can be broken by the `deposit_to_farm_vault` instruction. Anyone can use the `deposit_to_farm_vault` instruction to "donate" stake tokens to the farm vault, resulting in an `amount > 0` . Yet, the `total_stake` remains 0 when the active stake pool is empty. As a result, anyone staking after the "donate" deposit will not receive any stake shares that would have been due.

Proof of Concept

Attack flow in the PoC:

1. Setup a new farm. The active stake pool is empty at present.
2. The attacker calls `deposit_to_farm_vault` ix to deposit 1 unit of token to the farm vault and update the total amount of the farm state.
3. The victims stake 1000_000_000_000 tokens after the attack.
4. The victims receive 0 stake shares and thus lose all the tokens deposited.

Impact

When an empty farm is initialized or restarted, the attack can be executed in a front-running manner to ensure the exploitation is completed before any other stakes enter. What's more concerning is that if there is a farm with an enabled `deposit_warmup_period`

, the attacker only needs to complete the attack before the warm-up period of the first staker concludes.

Any stakers after the attack will lose all their principals and rewards.

Recommendation

Replace `if total_amount == 0 {` with `if total_stake == 0 {`

Mitigation Review Log

Kamino Team: Added a `total_stake == 0` to the specific case. Fixed in PR#57.

Offside Labs: **Fixed**. Two changes have been implemented to prevent this issue:

1. Now the `handler_deposit_to_farm_vault` ix can only be called with the signature of the `farm_admin` .
2. The condition in the `convert_amount_to_stake` has been modified to `total_stake == Decimal::zero() || total_amount == 0` .

4.2 deposit_to_farm_vault Instruction may Skirt Deposit Cap and Drain Constant-Type Rewards

Severity: High	Status: Fixed
Target: Smart Contract	Category: Logic

Description

The rewards of `RewardType::Constant` are determined using the following formula to calculate the reward amount:

```
726             // Calculate amount give RPS
727             let cumulative_amt = ...
728             ...
729             RewardType::Constant => {
730                 cumulative_amt * farm_state.total_staked_amount // Only
    ↪ the active stake earns fees
731             }
```

programs/farms/src/farm_operations.rs#L726

The fundamental issue arises from the incorrect usage of the denominator "Share" in RPS(Revenue Per Share). The `farm_state.total_staked_amount` represents the underlying staked token amount rather than the stake shares amount.

Ordinarily, this wouldn't pose a problem, as it operates on the assumption that share amounts are equal to or exceed the underlying token amount. However,

the `deposit_to_farm_vault` ix can invalidate this assumption. Additionally, the `deposit_to_farm_vault` ix can increase the `farm_state.total_staked_amount` directly without checking the `DepositCapAmount` .

When initializing an empty farm with `RewardType::Constant` type rewards, an attacker can exploit a flashloan to drain the reward vault in a single tx after just a few slots.

Proof of Concept

Attack flow in the PoC:

1. Setup a new farm with the following parameters:
    a. Reward amount: 5000000000
    b. Reward RPS 1:1 per stake_amount per sec
    c. Deposit cap: 10000
    d. Reward min drip duration should be 5000000000/10000 = 500000
    e. `deposit_warmup_period` and `withdraw_cooldown_period` : 0
2. The attacker becomes the first staker by staking 1 unit of token.
3. Wait for 1 second
4. The attacker borrows 4999999999 tokens via flashloan and uses the `deposit_to_farm_vault` ix to deposit them to the farm vault.
5. `harvest_reward` in the same tx.
6. Unstakes all the stake shares and repays the flashloan in the same tx.

Impact

The attack significantly affects farm pools with `RewardType::Constant` type.

For an empty pool, especially one with a `deposit_warmup_period` , attackers can directly obtain all rewards during the initialization using flash loans.

For a low-activity pool, attacks could occur not only during the initialization but also at later stages. In such cases, the losses due to attacker donations might be smaller than the rewards they receive.

Recommendation

The most straightforward remedy is to restrict calls to the `deposit_to_farm_vault` instruction, such as by allowing only the admin to call this instruction.

Mitigation Review Log

Kamino Team: We've made the deposit to farm ix permissionned. We don't think this inflation attack is profitable anyway, long term fix should be to consider deposit cap amount in the deposit to farm vault ix. Fixed in PR#57.

Offside Labs: **Fixed**. The `handler_deposit_to_farm_vault` ix can now be called exclusively with the signature of the `farm_admin` .

4.3 Oracle Price Factor Overflows Lead to DoS

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Math

Description

Once `ScopePricesAccount` and `ScopeOraclePriceId` are set, the `DatedPrice` from the scope will be used to calculate the reward during each execution of `refresh_global_reward` according to the following code:

```
    decimal_adjusted_amt * price.price.value / ten_pow(price.price.exp as
↪   usize)
```

Given that all three variables mentioned above are of type u64, an overflow will occur if `decimal_adjusted_amt * price.price.value > std::u64::MAX` .

Although Anchor includes overflow checks in its compilation configuration, eliminating a mathematical risk, an overflow during the execution of the critical `refresh_global_reward` function will halt the entire system.

Proof of Concept

We have reviewed the scope price account on the Solana mainnet: `3NJYftD5sjVfxSnUdZ1wVML8f3aC6m` . Three oracle types have been identified with `Price.exp = 15` : `MsolStake` , `SplStake` , and `CToken` (deprecated).

For the PoC test, we use the MSOL/SOL price with index 122.

Attack flow in the PoC:

1. Setup a new farm with the following parameters:
    a. Reward RPS 1:1 per stake_amount per sec
    b. `ScopePricesAccount` is forked from mainnet 3NJYftD5sjVfxSnUdZ1wVML8f3aC6mp1CXCL6
    c. `ScopeOraclePriceId` is 122, which is the MSOL/SOL price
2. `User1` stakes mSol, stake_amount = 10_000 lamport mSol
3. After 2 seconds, the reward is equivalent to 2*10_000 = 20_000 msol in sol
4. `User1` unstakes all shares. When calling `refresh_global_reward` during the unstaking process, the reward amount is calculated as follows:

```
20_000 * 1_150_644_463_992_915 / 1e15 = 23012889279858300000 / 1e15
```

The value `23012889279858300000` is surpasses `std::u64::MAX = 18446744073709551615` , causing the protocol to panic with a “multiply with overflow” error.

Please note that the value of 20_000 in the PoC is measured in Lamports, indicating the system will malfunction when the reward reaches 2e-05 sol.

Impact

The farm can't be refreshed and users can't unstake, stake or harvest_reward. Consequently, users with stakes will be unable to receive rewards or make withdrawals.

Recommendation

Consider dynamically scaling the excessively large `price.exp` in the Scope algorithm or scaling it to 8 uniformly.

Moreover, be aware that apart from the calculation of oracle_adjusted, there are other u64 * u64 operations that could also lead to a panic due to overflow issues.

programs/farms/src/farm_operations.rs#L737

programs/farms/src/farm_operations.rs#L756

programs/farms/src/state.rs#L187

programs/farms/src/stake_operations.rs#L575 and the next line.

programs/farms/src/utils/withdrawal_penalty.rs#L75

programs/farms/src/farm_operations.rs#L481

Mitigation Review Log

Kamino Team: Computation has been reworked. Fixed in PR#57.

Offside Labs: **Fixed**. The calculation has been extended from u64 to u128 bits to prevent overflow errors. It is advisable to also update programs/farms/src/stake_operations.rs#L588 and the subsequent line.

Kamino Team: Addressed in new PR: PR#99 .

4.4 Admin Withdrawal may Result in Subsequent User Stake Deposits being Stuck in the Pending Stake

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

Description

The `convert_amount_to_stake` function has an edge case where a full withdrawal from the farm causes an assert panic when `total_amount == 0` but `total_stake != 0` . In this scenario, users cannot emit new active shares. The situation is effectively described in the code comments:

```
    if total_amount == 0 {
        // total_stake != 0 should be impossible here BUT in case of a
        ↪  full withdrawal of the farm.
        // In this case we cannot emit new shares.
        assert_eq!(
```

However, this does not consider the scenario where the `deposit_warmup_period` is enabled. After a full (or a full - 1) withdrawal, users can still deposit tokens to the pending stake. Yet, these pending stakes after a full withdrawal can never be activated because the assert panic in the `convert_amount_to_stake` when calling `add_active_stake` .

Proof of Concept

Attack flow: The PoC describes a more general scenario where even if the pending stake is not zero during withdrawal, it can still prevent subsequent pending stakes from being withdrawn.

1. Setup a farm with deposit warmup period = 100s
2. `User1` stakes 1000 and activates it after the warmup period
3. `User2` stakes 1000 now
4. The admin withdraws 1999 at the same period
5. Due to the allocation of precision loss after the withdrawal:
    a. farm.total_active_amount becomes 0
    b. farm.total_pending_amount becomes 1
6. `User3` stakes 1000 after the withdrawal. It works and 1000 tokens are taken away from `user3` , because of farm.total_pending_amount != 0 .
7. When `user3` attempts to refresh himself to transition the stake from pending to active, the tx will panic because of the failed assert in `convert_amount_to_stake` .

Impact

Subsequent deposits are stuck in the pending queue and cannot be withdrawn.

Recommendation

Adding a pause flag could be an effective solution. This flag would automatically freeze the stake operations when the active stake amount has been fully withdrawn.

Mitigation Review Log

Kamino Team: Pending stake is unused at the moment. We should introduce “farm_locked” flag that is turned on on full withdrawal. Fixed in PR#94.

Offside Labs: **Fixed.** A lock flag named `is_farm_frozen` has been added to the farm state account. When `withdraw_from_farm_vault` ix triggers the lock condition in the `stake_ops::withdraw_farm` function, the farm will be locked.

4.5 A Portion of Rewards will be Permanently Lost Due to the Precision in user_refresh_reward

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Precision

Description

The `farm_operations::user_refresh_reward` function,

```
509     let rewards_tally =
        ↪ user_state.get_rewards_tally_decimal(reward_index);
510     let reward_per_share =
        ↪ farm_state.reward_infos[reward_index].get_reward_per_share_decimal();
511
512     let new_reward_tally: Decimal = if farm_state.is_delegated() { ... }
        ↪ else {
513         reward_per_share * user_state.get_active_stake_decimal()
514     };
515
516     let reward: u64 = (new_reward_tally - rewards_tally).try_floor()...
517     ...
```

programs/farms/src/farm_operations.rs#L509-L531

gets the current `reward_per_share` from `farm_state`. It calculates the accumulated rewards at the current time based on the user's share amount. The accumulated rewards are then subtracted by the previous `rewards_tally` to obtain the newly added rewards. Finally, these new rewards are added to `rewards_issued_unclaimed`.

The issue is that, although the intermediate process of accumulating rewards uses the decimal type, the final token amount recorded in `rewards_issued_unclaimed` will be rounded down to the u64 type. And the truncated portion will not be removed from the `rewards_tally`.

Impact

It might cause the loss of one unit of reward each time the user refreshes.On Solana, With low gas fees and frequent users refreshments, combined with small token decimals, the impact is magnified.

Since there's no direct interface for withdrawing rewards, these fractions of rewards will remain permanently locked in the contract.

Proof of Concept

Farms simulation:

1. Reward token Sollet WETH: 6 decimals; USD value: 2000 USD / 1e6 WETH
2. Reward token amount 100 * 1e6
3. RPS: 1e4 / s; reward should be drained in 10000 s = 100 * 100s
4. 101 users; each with 10 * 1e6 stake tokens

`users[0]` refreshes his state every 100 seconds, doing so a total of 101 times.

If every user refreshes as `users[0]` did, the PoC prints the final reward loss:

```
990000 990099009900990000000000
all_unclaimed 99990000
rewards lost 10000
```

It will be `10000 * 2000 USD / 1e6 = 20 USD`. For 1000 users, a common scenario in medium-sized DeFi, the loss could reach 200 USD.

Recommendation

Replace the user_state.set_rewards_tally_decimal(reward_index, new_reward_tally); with the following:

```
user_state.set_rewards_tally_decimal(reward_index,
↪   rewards_tally+reward.into());
```

After the mitigation, PoC output:

```
990099 990099000000000000000000
all_unclaimed 99999999
rewards lost 1
```

Mitigation Review Log

Kamino Team: Fix to implement, not a critical issue, users get less rewards than they could expect if refresh reward too often. Fixed in PR#94.

Offside Labs: **Fixed.** As noted in the code comments: `user_refresh_all_rewards` may leave a fraction of a token in the tally due to rounding, accept off by 1.

This fraction of a reward token is lost upon unstaking.

4.6 Updates to scope_prices and scope_oracle_price_id Should be Bound

Severity: Low	Status: Acknowledged
Target: Smart Contract	Category: Logic

Description

`farm_state.scope_prices` is used to fetch target price information in the `load_scope_price` function. This function requires that `farm_state.scope_prices` must be set when `farm_state.scope_oracle_price_id` is not set to the default value `u64::MAX`.

```
43      if farm_state.scope_oracle_price_id == u64::MAX {
44          Ok(None)
45      } else if scope_price_account.is_none() {
46          Err(FarmError::InvalidOracleConfig.into())
47      } else
```

programs/farms/src/utils/scope.rs#L43-L47

However, in the `update_farm_config` ix, the updates to `farm_state.scope_oracle_price_id` and `farm_state.scope_prices` are not necessarily synchronized.

Impact

Inconsistencies between updates to `scope_oracle_price_id` and `scope_prices` can cause the `load_scope_price` function to panic. Since the `load_scope_price` function is used in the rewards update of most major ix, such discrepancies can cause a partial DoS to the protocol.

Recommendation

Merge `FarmConfigOption::ScopePricesAccount` and `FarmConfigOption::ScopeOraclePriceId` into a single mode.

Mitigation Review Log

Kamino Team: No issue today as we only have one scope price feed but for future proofing we should implement the recommendation.

Offside Labs: Confirmed. This issue has no impact on the current production environment.

4.7 UpdateRewardScheduleCurvePoint is Not Fully Validated

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

Description

In the `UpdateRewardScheduleCurvePoint` of the `update_farm_config` ix

```
217             let reward_info = &mut farm_state.reward_infos[reward_index as
                    ↪ usize];
218             reward_info.reward_schedule_curve.set_point(
219                 point_index as usize,
220                 RewardPerTimeUnitPoint::new(ts_start, rps),
221             );
```

programs/farms/src/farm_operations.rs#L217-L221

the input data is directly written to `reward_info.reward_schedule_curve` without undergoing any validation.

Furthermore, global rewards are not refreshed before updating the rewards curve, potentially resulting in a change in the accumulated but unaccounted rewards amount.

Recommendation

Use the `RewardScheduleCurve:: from_points` function to verify the input curve points in the `UpdateRewardScheduleCurvePoint` mode of the `update_farm_config` ix.

It may also be beneficial to move the `UpdateRewardScheduleCurvePoint` mode to the reward update group at the beginning of the match cases. This makes it easier to refresh the global rewards.

Mitigation Review Log

Kamino Team: We implemented a full update of the curve in one ix so we can now validate the new curve after update. Fixed in PR#94.

Offside Labs: **Fixed.** Combining `UpdateRewardScheduleCurvePoints` with the initial reward-related updates in `update_farm_config` is advisable. By doing so, global rewards could be refreshed prior to each curve update, ensuring they remain consistent despite configuration changes.

Kamino Team: Addressed in PR#99 .

4.8 Informational and Undetermined Issues

withdraw_from_farm_vault Instruction Carries Centralization Risk

Severity: Informational	Status: Fixed
Target: Centralization	Category: Centralization

The `withdraw_from_farm_vault` instruction lacks withdrawal cap limits or time locks, posing significant centralization risks. If the admin private key is leaked, it could result in the theft of all funds.

Recommendations include the introduction of a pause mechanism for the protocol, along with the implementation of admin withdrawal cap limits or time locks. Additionally, ensure that a multi-signature wallet is utilized as the administrator on the mainnet deployment.

Kamino Team: The withdraw authority should always be a `multisig` if set. Added a comment to clarify this exported constraint.

Extra safety: we should add a clear error if the authority is unset.

Offside Labs: **Fixed**.

advance_clock_timestamp Function in Test Util Cannot Guarantee Synchronization for Clock Sysvar

Severity: Informational	Status: Fixed
Target: Test Suite	Category: Timing

The `advance_clock_timestamp` function,

```
128     clock.unix_timestamp += seconds as i64;
129     ctx.context.set_sysvar(&clock);
```

programs/farms/tests/common/runner.rs#L128

is used to set the current Clock Sysvar for test context. However, this cheat code is actually not an instruction call. It operates on the `bank_forks` handle in the TestContext, directly manipulating the current bank instance state. Consequently, it isn't constrained by the cached state limitations, such as `last_blockhash`, in the `banks_client`.

If the next call is bundled into the same slot, the Clock in that transaction will not be updated due to Solana's parallel processing. This could lead to unpredictable results in time-sensitive test cases.

Kamino Team: You are right it is solved by the usage of new blockhash both after updating the sysvar and at tx signature.

Offside Labs: **Fixed** by the commit.

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.

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.