OFFSIDE LABS

Dynamic AMM

Smart Contract Security Assessment

September 2024

Prepared for:

Meteora

Prepared by:

Offside Labs

Yao Li

Sirius Xie

Ronny Xing

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 *Dynamic AMM* smart contracts, starting on September 09, 2024, and concluding on September 20, 2024.

Project Overview

Dynamic AMM Pools are designed to provide sustainable liquidity for decentralized finance by leveraging a capital allocation layer. Rather than relying on unsustainable liquidity mining incentives, Dynamic AMM deposits assets directly into this layer, where they are dynamically allocated to external lending protocols. This allows LPs to earn yield and rewards on idle assets. By reducing dependence on liquidity mining, Dynamic AMM mitigates issues like token inflation and short-term liquidity loss, instead fostering long-term liquidity provision. This approach emphasizes value creation and aims to attract committed LPs to the ecosystem.

Audit Scope

The assessment scope contains mainly the smart contracts of the *amm* program for the *Dynamic AMM* project.

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

- Dynamic AMM
  - Branch: main
  - Commit Hash: 31edd5c16bcb31648e4bcfecf786ac3a07ffb797
  - Codebase Link

We listed the files we have audited below:

- Dynamic AMM
  - programs/amm/src/*.rs

Findings

The security audit revealed:

- 0 critical issue
- 1 high issues
- 2 medium issues
- 2 low issues
- 4 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	Slippage Check Bypassed When Output Token is Zero in Swap	High	Fixed
02	Fee Accumulation for Locked LP Exceeds the Actual Value	Medium	Fixed
03	Claim Fee Instruction Not Applicable to Stable Curve Pools	Medium	Fixed
04	Potential Precision Loss by Rounding for pool_-token_amount in add_balance_liquidity Instruction	Low	Acknowledged
05	Checks for Activation Point Are Not Entirely Applicable to Launch Pools	Low	Fixed
06	Lack of Check for Fees Config of SPL Stake Pool	Informational	Acknowledged
07	Lack of Constraint for Pool in update_-activation_point Instruction	Informational	Fixed
08	Inconsistent Constraints for a_token_vault and b_token_vault in Multiple Instructions	Informational	Fixed
09	Lack of Slippage Check in claim_fee Instruction	Informational	Acknowledged

4 Key Findings and Recommendations

4.1 Slippage Check Bypassed When Output Token is Zero in Swap

Severity: High	Status: Fixed
Target: Smart Contract	Category: MEV Risk

Description

In the swap instruction, the `check_exceed_slippage` verification is only triggered when `out_vault_lp_to_burn > 0`. Although there's a check in `curve_constant_product::swap` ensuring that `destination_amount_swapped` in the swap result is not zero. If `destination_amount_swapped` is insufficient to exchange for 1 `out_vault_lp`, then due to rounding down in `get_unmint_amount`, `out_vault_lp_to_burn` will be zero. In this case, the slippage check will be skipped.

```
if out_vault_lp_to_burn > 0 {
    ...
    check_exceed_slippage(actual_out_amount, minimum_out_amount)?;
    ...
}
```

programs/amm/src/instructions/swap.rs#L288-L328

Impact

MEV bots will initiate sandwich attacks on low-liquidity meme pools. Upon detecting a tx which user is attempting to buy the token, the bot can first purchase a large quantity of this token, resulting in the user only being able to obtain 1 token.

Since the vault lp rate is greater than 1, 1 token will extract 0 vault lp, thus bypassing the slippage check. This effectively results in the user donating their quote tokens to the pool.

Recommendation

Throw an error and panic the swap instruction when `out_vault_lp_to_burn == 0 && minimum_out_amount != 0`.

Mitigation Review Log

Meteora Team:

Fixed in relevant code implementation.

Offside Labs: Fixed.

4.2 Fee Accumulation for Locked LP Exceeds the Actual Value

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

Description

Each time LP is locked or fees are claimed, the accumulated fees of the locked LP will be converted to equivalent LP and added to `unclaimed_fee_pending`. Only when fees are claimed, this portion of LP accumulated as `unclaimed_fee_pending` will be burned and withdrawn.

The issue is that, before the fee LP is withdrawn, this corresponding fee LP is still retained by `total_locked_amount` and will continue to accumulate trading fees in the pool. This will result in more tokens being received when actually claiming fees than the fees accumulated by the locked LP.

Impact

After locking LP, the locker can repeatedly lock dust amounts of LP to refresh `unclaimed_fee_pending`. This will allow them to receive more tokens when they finally claim fees. The maximum amount would be the tokens currently corresponding to all the LP they have locked.

Proof of Concept

```
use crate::curve::base::SwapCurve;
use crate::curve::curve_constant_product::ConstantProductCurve;
use crate::math::u128x128_math::{shl_div, Rounding, SCALE_OFFSET};
use crate::math::{safe_math::SafeMath, utils_math::safe_mul_div_cast};

fn new_fee(token_a_amount:u64, token_b_amount:u64, lp_supply:u64,
    ↪ locked_lp:u128, pre_vp:u128)-> (u64, u128){
    let curve = ConstantProductCurve {};
    let d_new: u128 = curve.compute_d(token_a_amount,
        ↪ token_b_amount).unwrap().into();
    let vp_new = shl_div(d_new, lp_supply.into(), SCALE_OFFSET,
        ↪ Rounding::Down).unwrap();
    let new_fee = safe_mul_div_cast::<u64>(
        locked_lp,
        vp_new.safe_sub(pre_vp).unwrap(),
        vp_new,
        Rounding::Down,
    ).unwrap();
    (new_fee, vp_new)
}
```

```
    #[test]
    pub fn test_claim_once_or_four(){
        let curve = ConstantProductCurve {};
        let token_a_amount = 100;
        let token_b_amount = 100;
        let lp_supply = 100;
        let locked_lp = 50;
        print!("locked lp: {}", locked_lp);
        let d: u128 = curve.compute_d(token_a_amount,
        ↪  token_b_amount).unwrap().into();
        let vp0 = shl_div(d, lp_supply.into(), SCALE_OFFSET,
        ↪  Rounding::Down).unwrap();
        // claim once
        let (new_fee_4, vp_4) = new_fee(token_a_amount*4,
        ↪  token_b_amount*4, lp_supply, locked_lp, vp0);
        println!("claim once: fee lp {:#?} vp {}",new_fee_4, vp_4);
        // claim three times
        let (new_fee_4_2, vp_4_2) = new_fee(token_a_amount*2,
        ↪  token_b_amount*2, lp_supply, locked_lp, vp0);
        let (new_fee_4_3, vp_4_3) = new_fee(token_a_amount*3,
        ↪  token_b_amount*3, lp_supply, locked_lp, vp_4_2);
        let (new_fee_4_4, vp_4_4) = new_fee(token_a_amount*4,
        ↪  token_b_amount*4, lp_supply, locked_lp, vp_4_3);
        println!("claim(lock) three times: fee lp {:#?}, vp
        ↪  {}",new_fee_4_2+new_fee_4_3+new_fee_4_4, vp_4_4);
    }
```

logs:

```
-- state::lock_escrow::new_fee_test::test_claim_once_or_four stdout --
locked lp: 50claim once: fee lp 37 vp 73786976294838206464
claim(lock) three times: fee lp 53, vp 73786976294838206464
```

Recommendation

Each time `unclaimed_fee_pending` is refreshed, the LP accumulated as fees should be deducted from `total_locked_amount`.

Mitigation Review Log

Meteora Team:

Fixed in relevant code implementation

Offside Labs: Fixed.

4.3 Claim Fee Instruction Not Applicable to Stable Curve Pools

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

Description

Due to the claim fee instruction storing the current LP's virtual price (the D represented by a unit of LP) in the `lock_escrow`, direct adjustments to the D value render historical LP virtual prices invalid.

The virtual price of `ConstantProduct` type LPs only increases due to fee accumulation, while the virtual price of `Stable` type LPs is affected by multiple factors:

1. For a Depeg Stable pool, claim fee and lock instructions do not update the depeg price by `pool.update_curve_info` before updating `LockEscrow.lp_per_token` each time, which results in the recorded lp_per_token being lower than the actual value before the `total_locked_amount` is updated. When claiming fees, users will lose a portion of their fees, and for the lock instruction, users will steal a portion of fees that don't belong to them.
2. From another perspective, for Depeg Stable pools, claiming "fee" generated due to the increase in depeg price (growth in D value) should not be allowed. This portion of D value growth is included in the already locked token b, rather than coming from external fees.
3. The `override_curve_param` instruction will update the `AMP`, which will adjust the D value. When the D value increases, users will claim fees that don't originally belong to them. When the D value decreases, the `LockEscrow.lock` method will panic.

Impact

Multiple anomalies exist in the fee accumulation and claiming process for `LockEscrow` on Stable pools, potentially leading to fee loss and instruction failure.

Recommendation

Prevent the accumulation and claiming of fees in `LockEscrow` on Stable pools.

Mitigation Review Log

Meteora Team:

Fixed in relevant code implementation

Offside Labs: Fixed.

4.4 Potential Precision Loss by Rounding for pool_token_amount in add_balance_liquidity Instruction

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

Description

In the add_balance_liquidity instruction, the calculation for vault_lp_to_receive uses the formula:

```
vault_lp_to_receive = ceil(vault_lp_amount * pool_token_amount /
↪ pool_lp_supply)
```

```
26  let a_vault_lp_to_receive = utils::get_amount_by_share(
27      pool_token_amount,
28      pool_lp_supply,
29      ctx.accounts.a_vault_lp.amount,
30      RoundDirection::Ceiling, // user have to pay more due to
        ↪ precision loss
31  )
32  .ok_or(PoolError::MathOverflow)?;
```

programs/amm/src/instructions/add_balance_liquidity.rs#L26-L32

The rounding up to the ceiling introduces a discrepancy where the calculated `vault_lp_to_receive` may not match the original `pool_token_amount`. The issue arises because the final minted LP amount remains as `pool_token_amount`, but the intermediate calculations can result in a higher value due to rounding.

This can be expressed as:

```
floor(pool_lp_supply * vault_lp_to_receive / vault_lp_amount) >
↪ pool_token_amount
```

Thus, after rounding up, the minted amount may differ, leading to precision loss.

Impact

Current implementation may introduce precission loss for minted `pool_token_amount` due to rounding in the calculation of `vault_lp_to_receive`.

This potential precision loss can only exist in Stable Pools, ConstantProduct Pools are not affected.

Recommendation

Adjust the `pool_token_amount` based on the rounded `vault_lp_to_receive` to ensure that the precision is maintained and that the final minted LP amount reflects the correct

contribution.

Mitigation Review Log

Meteora Team:

As dicussion, user will bear that percision loss, so we acknowledge that.

4.5 Checks for Activation Point Are Not Entirely Applicable to Launch Pools

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Timing

Description

There is the check for `activation_point` in the permissionless pool initialization instruction:

```
614     let activation_point = if let Some(value) = activation_point {
615         // validate
616         require!(
617             value >= current_point && value <=
    ↪ current_point.safe_add(activation_duration)?,
618             PoolError::InvalidActivationPoint
619         );
620         value
621     } else {
622         current_point.safe_add(activation_duration)?
623     };
```

programs/amm/src/instructions/initialize_pool/initialize_permissionless_pool.rs#L614-L617

The `last_join_point` in *alpha-vault* is `activation_point - 13/12 * time_buffer`:

```
    pub fn get_last_join_point(&self) -> Result<u64> {
        let pre_activation_start_point =
            self.get_pre_activation_start_point()?;
        let last_join_point =

    pre_activation_start_point.safe_sub(self.time_buffer.safe_div(12)?)?;
    // 5 minutes
        Ok(last_join_point)
    }
```

So if the `activation_point` is less than `current_point + 13/12 * time_buffer`, the

alpha vault can not be initialized, because the `current_point` will be already greater than the `last_join_point` .

And this invalid `activation_point` can pass the check in the permissionless pool initialization instruction.

Recommendation

For the launch pool, it should be ensured that the `activation_point` set during initialization does not cause the `current_point` to exceed the `last_join_point` . In other words, `activation_point >= current_point + 13/12 * time_buffer` .

However, since only `activation_duration >= time_buffer` is verified in the `create_config` instruction, for the launch pool, it's also necessary to ensure that the config it uses satisfies `activation_duration >= 13/12 * time_buffer` .

Mitigation Review Log

Meteora Team:

Fixed in relevant code implementation

Offside Labs: Fixed.

4.6 Informational and Undetermined Issues

Lack of Check for Fees Config of SPL Stake Pool

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

SPL stake pool can add up to 100% fees on deposits and withdrawals, which would cause its `virtual_price` to significantly deviate from the current calculation formula `total_lamports / pool_token_supply` .

It is recommended to check whether the following related fee configurations are excessive when initializing the permissioned depeg pool with SPL stake pools:

- `sol_deposit_fee`
- `sol_withdrawal_fee`
- `next_sol_withdrawal_fee`

Mitigation Review Log:

Meteora Team: Acknowledged.

Lack of Constraint for Pool in update_activation_point Instruction

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

The `update_activation_point` instruction lacks a constraint for the `pool.enabled` flag.

Mitigation Review Log:

Meteora Team:

Fixed in relevant code implementation

Offside Labs: Fixed.

Inconsistent Constraints for a_token_vault and b_token_vault in Multiple Instructions

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

The constraints `a_vault.token_vault == a_token_vault.key()` and `b_vault.token_vault == b_token_vault.key()` are missing in certain instruction structs, such as:

- `BootstrapLiquidity`
- `InitializePermissionlessConstantProductPoolWithConfig`
- `InitializePermissionlessPool`
- `InitializePermissionlessPoolWithFeeTier`

These constraints are present in other instruction structs, such as:

- `AddOrRemoveBalanceLiquidity`
- `ClaimFee`
- `RemoveLiquiditySingleSide`
- `Swap`

Although both accounts will be validated in Dynamic Vault CPI, it's preferable to maintain consistent constraints.

Mitigation Review Log:

Meteora Team:

Fixed in relevant code implementation

Offside Labs: Fixed.

Lack of Slippage Check in claim_fee Instruction

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

The `claim_fee` instruction currently combines the unlocking and withdrawal of liquidity pool (LP) tokens. However, during the withdrawal step, there is no slippage check for the `fee_a` and `fee_b` amounts. This could lead to discrepancies between the expected and actual token amounts due to potential price changes prior to execution.

Without a slippage check, users may receive an unexpected amount of tokens if there is a price fluctuation before the instruction is executed.

Recommendation: Implement a slippage check for the `fee_a` and `fee_b` amounts during the withdrawal step to ensure users receive an expected and acceptable amount of tokens based on current prices.

Mitigation Review Log:

Meteora Team: Acknowledged.

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.

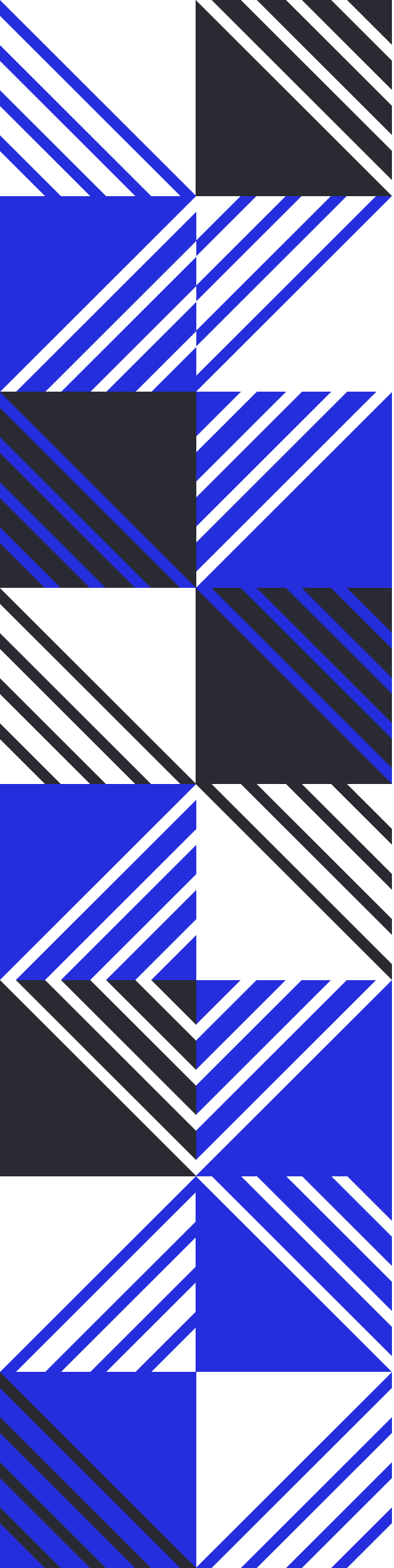
We expressly disclaim any liability for any losses or damages arising from the use of this report or from any security breaches that may occur in the future. We also recommend that our clients engage in multiple independent audits and establish a public bug bounty program as additional measures to bolster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