OFFSIDE LABS

GAMMA Swap

Smart Contract Security Assessment

October 2024

Prepared for:

GooseFX

Prepared by:

Offside Labs

Yao Li

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 *gamma-swap* smart contracts, starting on Oct 14, 2024, and concluding on Oct 21, 2024.

Project Overview

Goose Automated Market Making Algorithm (GAMMA) is a decentralized exchange (DEX) protocol built on Solana. It provides dynamic fee AMM functionality with customizable fee structures and liquidity pool management. Key features include support for Token2022, a referral program, and dynamic fees that adjust based on market volatility.

Audit Scope

The assessment scope contains mainly the smart contracts of the gamma program for the *gamma-swap* project.

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

- gamma-swap
  - Codebase: https://github.com/GooseFX1/gamma-swap
  - Commit Hash: c6b5fd23f74ae6d7365a7766569e9bbe45350592

We listed the files we have audited below:

- gamma-swap
  - programs/gamma/src/curve/calculator.rs
  - programs/gamma/src/curve/constant_product.rs
  - programs/gamma/src/fees/dynamic_fee.rs
  - programs/gamma/src/fees/mod.rs
  - programs/gamma/src/fees/static_fees.rs
  - programs/gamma/src/instructions/deposit.rs
  - programs/gamma/src/instructions/init_user_pool_liquidity.rs
  - programs/gamma/src/instructions/initialize.rs
  - programs/gamma/src/instructions/swap_base_input.rs
  - programs/gamma/src/instructions/swap_base_output.rs
  - programs/gamma/src/instructions/withdraw.rs
  - programs/gamma/src/lib.rs
  - programs/gamma/src/states/config.rs
  - programs/gamma/src/states/events.rs
  - programs/gamma/src/states/oracle.rs
  - programs/gamma/src/states/pool.rs
  - programs/gamma/src/states/user_pool_liquidity.rs
  - programs/gamma/src/utils/math.rs

- programs/gamma/src/utils/token.rs
- programs/gamma/src/utils/swap_referral.rs

Findings

The security audit revealed:

- 0 critical issue
- 1 high issues
- 5 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	Frequent Swaps Can Cause a Denial of Service Attack on Pool	High	Fixed
02	Permanent Denial of Service Attack on Pool if All LPs are Withdrawn	Medium	Fixed
03	Permanent Denial of Service Attack on Pool if Open Time is Too High	Medium	Fixed
04	Incorrect Volatility Fee Due to Confused Observation Index	Medium	Fixed
05	High Volatility Fee Can be Reduced Substantially in a Short Period	Medium	Fixed
06	Cumulative Price in Observation State Can be Manipulated	Medium	Fixed
07	Invariant Check on Constant Product is Incorrect	Low	Fixed
08	Exchange Rate Manipulation during Pool Initialization	Low	Fixed
09	Referral Fee Transfer Can be Skipped If Post Fee Amount is Zero	Informational	Fixed
10	Math Issues In Volatility Component Formula	Informational	Fixed

4 Key Findings and Recommendations

4.1 Frequent Swaps Can Cause a Denial of Service Attack on Pool

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

Description

In the swap instructions, the `DynamicFee.get_price_range` function is used to calculate dynamic fees based on the observation state. The function computes prices using observations within a time window, where each observation is compared with its preceding observation, referred to as `last_observation`. The relevant code snippet is as follows:

```rust
    let cumulative_token_0_price = observation_with_index
        .observation
        .cumulative_token_0_price_x32;

    let last_observation =
    ↪   &observation_state.observations[last_observation_index];
    let last_cumulative_token_0_price =
    ↪   last_observation.cumulative_token_0_price_x32;

    let time_delta = observation_with_index
        .observation
        .block_timestamp
        .saturating_sub(last_observation.block_timestamp)
        as u128;

    let price = cumulative_token_0_price
        .checked_sub(last_cumulative_token_0_price)
    ...
```

programs/gamma/src/fees/dynamic_fee.rs#L252-L266

The issue arises when all observations within the time window are used, particularly when calculating the observation with the smallest timestamp. In such cases, the corresponding `last_observation` will have the largest timestamp, causing the `last_cumulative_token_0_price` to be larger than the `cumulative_token_0_price`. This leads to a mathematical overflow, which causes the transaction to revert.

Impact

If there are 100 swaps with varying timestamps in a pool within a one-hour time window, every subsequent swap will revert with an error. This condition persists until the observation with the smallest timestamp moves out of the one-hour window.

This situation is especially common for popular token pairs, where high trading activity can quickly fill up the observation state.

Additionally, an attacker can execute 100 swaps in a short time frame, causing the pool to be non-functional for up to one hour, effectively denying other users the ability to perform swaps in that pool during this time.

Recommendation

The smallest timestamp observation should be skipped if all observations fall within the time window

Mitigation Review Log

Fixed in commit 29d6dcb3e9f799e0816365e36d758962eba052fe.

4.2 Permanent Denial of Service Attack on Pool if All LPs are Withdrawn

Severity: Medium	**Status: Fixed**
Target: Smart Contract	**Category: DOS Risk**

Description

After pool initialization, users can provide or withdraw liquidity through deposit and withdraw instructions. Both instructions calculate the ratio of `token_amount` to `lp_supply` . However, the withdraw instruction allows users to withdraw all LPs, which causes `lp_supply = 0` . When this happens, any subsequent deposit attempts will fail due to a division by zero error on `lp_supply` .

Impact

The number of pools that can be created for a specific token pair is limited by the `ammconfig` count. An attacker could initialize pools for a specific token pair and then withdraw all LPs, causing the `lp_supply` to reach zero. This would result in a denial of service attack on that token pair, preventing further deposits into the pool.

Recommendation

To prevent this issue, lock a minimum amount of LPs (e.g., between 100 and 10,000) in the pool during initialization. This ensures that the `lp_supply` will never reach zero, preventing subsequent deposits from failing.

Mitigation Review Log

Fixed in commit 991595dfeffcbcee8f05e2bea7c38f37464e82d9.

4.3 Permanent Denial of Service Attack on Pool if Open Time is Too High

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: DOS Risk

Description

A pool is initialized with parameter `open_time` , and users are only allowed to swap after the `open_time` . However, `open_time` can be set arbitrarily high, potentially enabling a denial of service attack on the pool.

Impact

The number of pools that can be created for a specific token pair is limited by the `ammconfig` count. An attacker could initialize multiple pools for a specific token pair with `open_time = u64::MAX` , effectively preventing swaps in these pools for an indefinite period. This would lead to a DOS attack on the token pair, blocking users from accessing the pool.

Recommendation

Implement restrictions to ensure that `open_time` cannot be set excessively high. Additionally, consider allowing non-PDA (Program Derived Address) style pool initialization.

Mitigation Review Log

Fixed in commit fbd980de03605e35f2e8c09264cb088676acc37b.

4.4 Incorrect Volatility Fee Due to Confused Observation Index

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

Description

In the swap instructions, the `DynamicFee.get_price_range` function is used to calculate dynamic fees based on the observation state. The function computes prices using observations within a time window, where each observation is compared with its preceding observation, referred to as `last_observation`. The relevant code snippet is as follows:

```rust
    let mut descending_order_observations = observation_state
        .observations
        .iter()
        .filter(|x| {
            x.block_timestamp != 0
                && x.cumulative_token_0_price_x32 != 0
                && x.cumulative_token_1_price_x32 != 0
        })
        .enumerate()
        .filter_map(|(index, observation)| {
            index.try_into().ok().map(|index| ObservationWithIndex {
                index,
                observation: *observation,
            })
        })
        .collect::<Vec<_>>();
```

programs/gamma/src/fees/dynamic_fee.rs#L210-L225

The issue arises because the `enumerate` function is used after `filter`, causing the resulting indexes to no longer correspond to the original observation indexes. These altered indexes are then used to retrieve `last_observation`, resulting in an incorrect observation being selected.

Impact

The `DynamicFee.get_price_range` function will return incorrect prices, leading to miscalculations in the volatility component of the dynamic fee. This could result in inaccurate fee charges.

Recommendation

Apply `enumerate` before `filter` to ensure that the correct observation indexes are preserved and used for comparison with `last_observation`.

Mitigation Review Log

Fixed in commit 3b8b3415d1111606fcd9e7d6596d5e66e7760f0d.

4.5 High Volatility Fee Can be Reduced Substantially in a Short Period

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

Description

In the swap instructions, the volatility fee is calculated based on the observation state, which can go up to 10%. It measures the min/max/avg prices in the observation state to calculate the volatility ratio. During the final steps of the swap instructions, if the current timestamp exceeds the last timestamp in the observation state, an update is triggered.

This mechanism allows an attacker to bypass the high volatility fee over a short period. By performing multiple swaps with very small token amounts, the attacker can overwrite all observations with nearly identical prices, significantly lowering the volatility fee.

Impact

If the volatility fee is high, an attacker can conduct up to 100 small-amount swaps to overwrite all observations with closely matching prices, causing the volatility fee to drop substantially. In an ideal scenario, this attack could be carried out within a short period of around 2 minutes. Since there would likely be few interfering swaps by other users in such a short window, the attacker would have a high chance of successfully reducing the volatility fee.

Recommendation

Increase the update interval for the observation state to extend the time window for this kind of attack, making it more difficult to manipulate the volatility fee. This would reduce the likelihood of the attack succeeding.

Mitigation Review Log

Fixed in commit 67d095e689f821e8958e7fd38f081e3315aeaec5.

4.6 Cumulative Price in Observation State Can be Manipulated

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

Description

The observation state is designed to provide time-weighted average price (TWAP) price by recording the cumulative prices, which are updated during both `swap_base_input` and `swap_base_out` operations.

```rust
    ctx.accounts.observation_state.load_mut()?.update(
        oracle::block_timestamp(),
        token_0_price_x64,
        token_1_price_x64,
    );
```

programs/gamma/src/instructions/swap_base_input.rs#L341-L345

By TWAP definition, during a swap, the cumulative price should be updated using the current time and the price before the swap, as confirmed by implementations like Uniswap V2:

```solidity
    price0CumulativeLast +=
    ↪ uint(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) * timeElapsed;
    price1CumulativeLast +=
    ↪ uint(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) * timeElapsed;
```

contracts/UniswapV2Pair.sol#L79-L80

However, in this case, the observation state is updated using the price after the swap. This creates a vulnerability where an attacker can manipulate the price by executing a swap to artificially push the price to an abnormal level and then reversing the swap within the same transaction to mitigate exposure. As a result, the cumulative price can be manipulated due to this artificially altered price.

Impact

The current cumulative price does not align with the correct TWAP definition. An attacker can manipulate the cumulative price, and if other protocols rely on the observation state as a TWAP oracle, they could be misled, leading to potential financial losses.

Recommendation

To prevent manipulation, update the observation state using the price before the swap to ensure the integrity of the TWAP calculation.

Mitigation Review Log

Fixed in commit 4d3b6db291c57cb6b88b9600de8c0aa7adc0fa0f.

4.7 Invariant Check on Constant Product is Incorrect

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

Description

The invariant check for the constant product formula is performed in both swap_base_-input and swap_base_output functions.

```
    let constant_after = u128::from(result.new_swap_source_amount)
        .checked_mul(u128::from(result.new_swap_destination_amount))
        .ok_or(GammaError::MathOverflow)?;
    #[cfg(feature = "enable-log")]
    msg!(
        "source_amount_swapped:{}, destination_amount_swapped:{},...",
        result.source_amount_swapped,
        result.destination_amount_swapped,
        constant_before,
        constant_after
    );
    require_gte!(constant_after, constant_before);
```

programs/gamma/src/instructions/swap_base_input.rs#L160-L171

This check ensures that the product of the two token amounts will not decrease after a swap. `constant_before` represents the product of token amounts before the swap (excluding protocol fees and fund fees). `constant_after` represents the product of `new_swap_source_amount` and `new_swap_destination_amount` in the `SwapResult`, as calculated by the `CurveCalculator`.

However, the fees are calculated but not subtracted from `new_swap_source_amount` in both `swap_base_input` and `swap_base_output` of the `CurveCalculator`. This results in an inflated `new_swap_source_amount`, making the invariant check incorrect because `constant_after` is larger than it should be.

Impact

Although this issue has no direct effect on the swapped amounts (which are calculated correctly), the invariant check is not as strict as it should be. Fixing this will improve robustness and ensure the system is better prepared for future upgrades or potential unexpected issues.

Recommendation

Subtract the fees from `new_swap_source_amount` before calculating `constant_after`. This will ensure that the invariant check is accurate and strictly enforced.

Mitigation Review Log

Fixed in commit bfeca0edcb26e84af69891cf6bef20f02337d39b.

4.8 Exchange Rate Manipulation during Pool Initialization

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

Description

After pool initialization, users can provide or withdraw liquidity through deposit and withdraw instructions, both of which calculate the ratio of `token_amount` to `lp_supply` . However, `token_amount` is loaded from the pool vault's token account, and there are no restrictions to prevent the `lp_supply` from being extremely low during pool initialization. These two factors can be exploited by an attacker to manipulate the exchange ratio. Specifically, an attacker can initialize the pool with a small amount of tokens and then directly transfer additional tokens into the pool vault, skewing the ratio.

Impact

An attacker could initialize a pool by depositing just 1 unit of each token, receiving 1 unit of `lp_supply` . The attacker can then transfer additional tokens into the pool vault, causing the final token amounts to be x and y. As a result, subsequent users can only deposit in amounts that are multiples of x and y. This manipulation would prevent users with lower token amounts from making deposits, reducing accessibility and potentially harming liquidity.

Recommendation

To mitigate this issue, lock a minimum amount of LPs (e.g., between 100 and 10,000) in the pool during initialization.

Additionally, instead of relying solely on the token amounts in the pool vault's token account, use state variables to track the token amounts, ensuring accurate and secure exchange rate calculations.

Mitigation Review Log

Fixed in commit 991595dfeffcbcee8f05e2bea7c38f37464e82d9.

4.9 Informational and Undetermined Issues

Referral Fee Transfer Can be Skipped If Post Fee Amount is Zero

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Optimization

A `token2022` mint's transfer fee configuration can be set with a fee rate of 100% and a maximum fee of x. In this setup, when the transfer amount is equal to or greater than the maximum fee, the fee behaves as a constant value x. If the input token is `token2022` with a fee rate of 100% and the `referral_amount` does not exceed the `maximum_fee` , the post-fee amount will be zero. In such cases, the referral fee transfer can be skipped to avoid unnecessary waste.

Math Issues In Volatility Component Formula

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

The volatility component currently uses the min/max/average prices of the pool to calculate fees. However, the average price is derived using the arithmetic mean of observed prices. It would be more accurate to use the time-weighted average price (TWAP) instead, as it better reflects price movements over time. Additionally, the formula relies on one-sided data from `cumulative_token_0_price` . The result is not symmetrical if switched to `cumulative_token_1_price` , indicating a lack of fairness in the formula. This asymmetry could lead to users paying higher fees on one side calculation of the pool.

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