OFFSIDE Labs

Jupiter Aggregator

Smart Contract Security Assessment

April 2024

Prepared for:

Jupiter

Prepared by:

Offside Labs

Ronny Xing

Haijiang Xie

Yao Li

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 *Jupiter Aggregator* smart contracts, starting on April 17th, 2024, and concluding on April 28th, 2024.

Jupiter Aggregator Project Overview

Jupiter Aggregator is a decentralized exchange aggregator designed to provide the best rates for swapping SPL tokens on the Solana blockchain. It routes trades through multiple liquidity sources to ensure optimal prices, low slippage, and efficient transaction execution. Users benefit from its seamless interface, deep liquidity, and the ability to perform complex token swaps in a single transaction.

Audit Scope

The assessment scope contains mainly the smart contracts of the *jupiter-aggregator-program* for the *Jupiter* project.

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

- jupiter-aggregator-program
  - Branch: main
  - Commit Hash: 7c3a9ae6cbe9034f6108cf2bb260ac578667940c
  - Codebase Link

We listed the files we have audited below:

- jupiter-aggregator-program
  - programs/jupiter/src/*

Findings

The security audit revealed:

- 0 critical issue
- 0 high issues
- 1 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	Pre-calculation of the Raydium AMM Is Inaccurate in calculate_swap_in_amount	Medium	Fixed
02	Inaccurate Fee Calculation in apply_exact_out_fees_if_applicable	Low	Fixed
03	Missing SyncNative When Token Is WSOL in set_token_ledger	Low	Acknowledged
04	Integrity Checks on RoutePlanStep in execute_route_plan	Informational	Acknowledged
05	Error in Calculating Slippage for Small Amount or Large Slippage	Informational	Fixed
06	Token-2022 Support Should Not Be Enabled in Exact Out Mode	Informational	Fixed
07	Redundant PDA Signature in serum::create_open_orders IX	Informational	Fixed

4 Key Findings and Recommendations

4.1 Pre-calculation of the Raydium AMM Is Inaccurate in calculate_swap_in_amount

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

Description

For instructions related to `exact_out`, such as `exact_out_route`, it is necessary to first calculate the corresponding input amount using the expected `out_amount`. This approach allows for deriving a complete route plan. Finally, each step's swap is executed by calling an external dex based on the respective `out_amount`.

The main issue is that, for the `Swap::Raydium` swap step, it uses the `raydium_calculate_in_amount` function to get the input amount.

It directly uses the value of the token balance to calculate the invariant `K`, as shown in the pseudocode below:

```
let swap_source_amount =
    token::accessor::amount(&accounts.pool_coin_token_account)?;
let swap_destination_amount =
    token::accessor::amount(&accounts.pool_pc_token_account)?;

let K = swap_source_amount * swap_destination_amount;
```

However, in the Raydium AMM, the invariant calculation actually relies on the following pseudocode, if the AMM doesn't enable the Orderbook: raydium-io:raydium-amm/program/src/math.rs#L294-L335

```
K = (pc_amount - amm.state_data.need_take_pnl_pc) * (coin_amount -
    amm.state_data.need_take_pnl_coin)
```

Impact

Since the swap in/out amounts for the exact out swap are entirely based on pre-calculated values, and there is no additional amount verification during the actual external swap execution, inaccurate pre-calculations could result in users paying more in amount than expected, which typically leads to two outcomes:

1. The in amount actually paid by the user will be greater than the expected `maximum_in_amount`, which equates to a loss for the user, especially when they encounter potential MEV.
2. If Raydium AMM is used as an intermediate step in the route, the swap transaction may fail due to insufficient token amounts in the intermediate token accounts.

Recommendation

Get the `state` from the current AMM account and then accurately calculate `K` according to the Raydium AMM implementation.

Mitigation Review Log

Jupiter Team: Mitigation commit

In addition, we disabled exact out for pool with active mm, which `amm_info.status == 5` , because that's only 5 of them.

Offside Labs: Fixed.

4.2 Inaccurate Fee Calculation in apply_exact_out_fees_if_applicable

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

Description

Both `exact_in` and `exact_out` transactions use the same formula in `calculate_fee` to calculate platform fees. This formula multiplies `platform_fee_bps` directly with the transaction amount, which is `in_amount` in the `exact_out` and `out_amount` in the `exact_in` . However, the application of this formula should differ between the two transaction types due to the point in the transaction flow where fees are applied.

let A be the USD value of the user inputs, and let R be the fee bps. Assume that A and R are the same under the following `exact_in` and `exact_out` scenarios:

- For `exact_in` transactions: out_fee_value $= A \times R$, out_value_1 $= A \times (1 - R)$
- For `exact_out` transactions: out_value_2 $= \frac{A}{1+R}$, in_fee_value $= A \times \frac{R}{1+R}$

Because $A \times R > A \times \frac{R}{1+R}$, therefore out_fee_value $>$ in_fee_value, which means the fee of the `exact_out` transactions is slightly less than the nominal fee of the `exact_in` transactions.

And we can also get the following derivation: `out_value_1 < out_value_2` $\Leftrightarrow 1 - R^2 < 1 \Leftrightarrow R^2 > 0$, which means users always get more output tokens in the `exact_out` transactions with the same input amounts.

Impact

This discrepancy in fee calculation leads to inconsistent fee collection between `exact_in` and `exact_out` transactions. As a result, the expected revenue from transaction fees is potentially reduced.

Proof of Concept

Assuming `platform_fee_bps (R)` is set to the maximal value 2.55%, and A is USD value of the user inputs.

- For `exact_in` : The user receives $A \times (1 - 0.0255)$.
- For `exact_out` : The user receives $\frac{A}{1+0.0255}$, leading to an actual fee rate of approximately $1 - \frac{1}{1+0.0255} \times 100\% = 2.486\%$, which is less than the intended 2.55%.

Recommendation

For `exact_out` transactions, modify the fee calculation formula to `A * R / (1 - R)` in the `apply_exact_out_fees_if_applicable` function. This ensures the fee reflects the intended rate and aligns fee calculations across transaction types.

Mitigation Review Log

Jupiter Team: PR-164

Offside Labs: **Fixed**. After mitigation for `exact_out` transactions:

- out_value_2 $= \frac{A}{1+\frac{R}{1-R}} = A \times (1 - R)$
- in_fee_value $= A \times (1 - R) \times \frac{R}{1-R} = A \times R$

Now `in_fee_value` of the `exact_out` is equal to `out_fee_value` of the `exact_in` .

4.3 Missing SyncNative When Token Is WSOL in set_token_ledger

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

Description

If `token_account` is `WSOL` , `set_token_ledger` does not invoke `SyncNative` before setting account of token ledger:

```
pub fn set_token_ledger(ctx: Context<SetTokenLedger>) -> Result<()> {
    ctx.accounts.token_ledger.token_account =
ctx.accounts.token_account.key();
    ctx.accounts.token_ledger.amount =
ctx.accounts.token_account.amount;
    Ok(())
}
```

programs/jupiter/src/lib.rs#L760-764

Impact

If `token_account` is `WSOL` and its token amount is `not synchronized` with lamports in that account, `set_token_ledger` may set the initial `token_ledger.amount` to a smaller value.

This can lead to IXs ending with `_with_token_ledger` consuming more tokens than expected during the swap operation.

Recommendation

When token is `WSOL`, it is recommended to add `SyncNative` before setting the token amount in `set_token_ledger`. This ensures that the token ledger reflects the correct token amount.

Similarly, another optional fix is to also confirm lamports synchronization before getting `to_amount_before` in the `swap_wrapper` function, if the `destination_token_account` is `WSOL`.

Mitigation Review Log

Jupiter Team: Acknowledged.

Addition of system program would be a breaking change: programs/jupiter/src/account_structs.rs#L248

In addition, the token program transfers that will occur while swapping would not cause the extra lamports to be suddenly involved, so it seems like this user error does not have much effect. This also does not protect from user doing something wrong after so we decide not to fix.

Offside Labs: Yes, agree. There won't be any additional sync operations that could cause sudden changes to the token amount within the system's own instructions (i.e., between the `set_token_ledger` and `_with_token_ledger` series of instructions).

It might be a good idea to add a `SyncNative` IX before the `set_token_ledger` IX in the relevant SDK. This will prevent breaking change to the existing contract and avoid the issue about user errors.

4.4 Informational and Undetermined Issues

Integrity Checks on RoutePlanStep in execute_route_plan

Severity: Informational	Status: Acknowledged
Target: Smart Contract	Category: Integrity Check

The `execute_route_plan` function operates under the assumption that the last executed

`RoutePlanStep` is both the final and the only output token. The contract enforces a slippage check solely on this final output token, without evaluating the intermediate steps. This setup is generally sufficient when the route plan is intended to produce only a single output token.

However, if the route plan becomes corrupted while still passing the slippage check, any residual tokens in a shared token account might be lost for users. To mitigate this risk, we recommend implementing a basic integrity check on the token steps when the swap ends:

1. Ensure the `amount` is zero for all steps except the final one.
2. Confirm that the `consumed_amount` is zero for the final step. (Note: Current code constraints make this violation impossible.)

Jupiter Team: Acknowledged, malformed route plan is a client responsibility.

The *Jupiter aggregator* program is CPU heavy and additional validation to safeguard users is a nice to have but we believe we should not add any more "nice to have" for now.

Error in Calculating Slippage for Small Amount or Large Slippage

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

`get_minimum_out_amount` uses this formula to calculate minimum amount:

```
let (minimum_amount, _) = u128::from(amount)
        .checked_mul(FEE_DENOMINATOR.checked_sub(slippage_bps.into())?)?
        .checked_ceil_div(FEE_DENOMINATOR)?;
```

Within inner `checked_ceil_div` , if the numerator is smaller than the denominator, it will directly return `None` .

Suppose in the aforementioned formula, if `amount * (FEE_DENOMINATOR - slippage_bps) < FEE_DENOMINATOR` , then `checked_ceil_div` will return `None` . Consequently, `get_minimum_out_amount` will also return `None` , causing both `route` and `shared_accounts_route` IX to fail with `JupiterError::InvalidCalculation` .

It is possible for this inequality to hold true with a small `amount` and large `slippage_bps` . However, we believe that even in such cases, the minimum amount calculated based on these slippage parameters should also be reasonable.

To mitigate this risk, we recommend adding a basic None check in `get_minimum_out_amount` .

Jupiter Team: PR-164

Offside Labs: Fixed.

Token-2022 Support Should Not Be Enabled in Exact Out Mode

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

In the implementation of the contract, the 3 dex swap CPIs (`raydium` , `raydium_clmm` , `whirlpool`) supported under the "exact out" mode only support `spl-token` and do not support `token-2022` . Therefore, it is unnecessary to include the `token_2022_program` account and invoke the `get_token_program_on_mint` method in the IXs of the "exact out" operation.

Moreover, according to current implementation of "exact out" swap, contract should calculate the amount accurately for swap-in. However, the transfer fee extensions supported by `token-2022` can cause significant issues with this calculation.

Therefore, the "exact out" IX should avoid supporting `token-2022` .

To mitigate this risk, we recommend removing `get_token_program_on_mint` calls in `exact out` mode.

Jupiter Team: Check the effective in amount to prevent any invalid calculation to miss the risk check. As a result user is safe we don't need the extra token2022 checks or restricting to token2022 without a fee extension. Fix Commit: PR-164

Offside Labs: **Fixed**. The mitigation has effectively handled the accounting of actual transferred input token amounts from user source token accounts.
Note: If it needs to add support for transfer fees for exact out mode in the future, please ensure that the transfer fee is accurately calculated in the `calculate_swap_in_amount` implementation. The current mitigation only ensures the actual consumption amount of the input tokens, while the intermediate token consumption comes from pre-calculation. Therefore, if the transfer fee causes the actual input amount in the intermediate process to not match the pre-calculation, this may lead to the consumption of the original tokens in the token accounts during the intermediate swap process.

Redundant PDA Signature in serum::create_open_orders IX

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Redundant Code

In `create_open_orders` IX, a `signer_seeds` is generated and passed to `amm::serum::init_open_orders` . However, The signed PDA in this case refers to the `open_orders` account, whose ownership has already been transferred to `dex_program` . Therefore, this `signer_seeds` is no longer needed.

We recommend removing `signer_seeds` argument when invoking `amm::serum::init_open_orders` .

Jupiter Team: PR-164

Offside Labs: **Fixed**.

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.